Actively Secure PIR from Verifiable Linear Relations over Rings

Louis Tremblay Thibault*

École de technologie supérieure and Mila, Montréal, Canada

Abstract. Privately retrieving data from a database without revealing which element is being retrieved is a fundamental problem in cryptography with many diverse applications. State-of-the-art solutions to this problem are generally based on homomorphic encryption, which offers security only against passive adversaries. In this work, we contribute to the recently introduced notion of Verifiable Private Information Retrieval (vPIR), which leverages verifiable computation to achieve security against active adversaries. We present a new sum-check based interactive oracle proof for linear relations over power-of-two cyclotomic rings, and show how to instantiate it into a vPIR protocol using Ring-LWE. Our solution is concretely efficient, offers the best possible security guarantees and is plausibly post-quantum secure. We additionally provide an open-source Rust implementation. Our server time improves on prior work's IOP by more than an order of magnitude, leading to a factor two end-to-end performance improvement. Compared to vPIR prior work, on a 4 GiB database, our offline communication and client storage are of 128 MiB, against 3.29 GiB. Preprocessing is run once per database and amortized over an unbounded number of queries.

Keywords: Private information retrieval · Verifiable computation · Lattice-based cryptography

1 Introduction

Private Information Retrieval (PIR) [CGKS95] is a cryptographic protocol between a server holding a database and a client holding a private query which allows the client to retrieve a database entry without revealing to the server which entry it retrieved. A long line of work has successfully studied the problem of improving the efficiency of PIR protocols, leading in real-world deployments of PIR as a component in larger privacy-preserving systems [DPC23,HDCGZ23,App24]. However, most of these works posit strong assumptions on the server's behavior, namely that the server is "semi-honest" or "honest-but-curious". More precisely, this means that the PIR protocols guarantee client privacy against an adversarial server, only if the server simply eavesdrops without deviating from the prescribed protocol. Furthermore, a fully malicious server (i.e., an active adversary) can deviate from the intended protocol to mount selective failure attacks [KO97], which break client privacy.

* Work performed while at Zama, Paris, France.

A more recent line of work has studied a stronger class of PIR protocols (sometimes called verifiable PIR, or vPIR) which broadly aims to be secure against active adversaries [CNCG+23,dCL24,DT24,FMS25,RLP26], however these works often offer slightly different security guarantees and make different trade-offs between efficiency and security. The present work sets itself in this lineage by proposing an actively secure PIR protocol which is concretely efficient and offers strong security guarantees. More precisely, our protocol comprises a one-time preprocessing phase and is resistant to selective failure attacks, even if the server misbehaves during the offline and/or online phases. Further, both the digest and the work of the server are publicly verifiable, which allows anyone to detect malicious behaviour without compromising the client's privacy.

Paper organization. In the rest of this section, we provide an overview of adjacent works in the literature and highlight the differences with this work. We then summarize our novel contributions. In Section 2, we offer the knowledgeable reader a technical overview which highlights the key technical ideas behind our protocol. We give technical preliminaries in Section 3 and warm up with a passively secure version of our protocol in Section 4. In Section 5, we present a verifiable version of the previous protocol, which we show how to use in Section 6 to construct verifiable PIR. Finally, we provide experimental results in Section 7 and conclude with a discussion regarding comparable works.

1.1 Related work

Authenticated PIR. To enhance security of PIR against selective failure attacks, [CNCG+23] introduces the concept of authenticated PIR (APIR), which allows a client to verify the integrity of the data retrieved from a PIR server. In the single server setting, which is the setting we consider in this work, the client obtains a database digest *out-of-band* prior to querying the server. The client can then use this digest to verify that the response from the server is correct, i.e., that it corresponds to the database associated with the known digest. However, their security model does not consider maliciously generated digests. As highlighted by further work [DT24], it is natural to consider a security model that encompasses fully malicious servers, i.e. those servers that may want to cheat not only during query answering, but also during digest generation.

VeriSimplePIR. To address the aforementioned limitation, [dCL24] proposes a stronger notion of security for PIR, called verifiable PIR. In this model, the server is fully malicious and may try to cheat during both digest generation and query answering. To achieve this stronger notion, the authors propose a protocol which relies on extractable commitments. The commitments being binding force servers to commit to a unique database during the digest generation phase. Then, during query answering, the server provides a proof that the response is consistent with the committed database. This approach has a few drawbacks, namely that the digest is large (for a database of 4 GB, the digest is of 600 MB) and that the expensive digest generation phase has to be run *each time* malicious behavior is detected.

Fully Malicious APIR. [DT24] also bridges the "out-of-band digest" gap from the solution proposed in [CNCG+23], but does so in a different manner than [dCL24]. Instead of relying on extractable commitments and proofs of consistent computation, they build on the protocol from [CNCG+23] with honestly generated commitments and add a validation phase. From the point of view of the server, this validation phase is indistinguishable from normal querying. If the client and server successfully complete this validation phase, the client is assured that the subsequent queries are correct with respect to the database used to generate the digest. As such, selective failure attacks are thwarted.

PIR to mPIR Compiler. Another recent work [FMS25] proposes a compiler that uses locally-decodable codes to transform any PIR scheme into a maliciously secure PIR scheme. While this work provides a strong theoretical result in that their approach is generic and can be applied to any PIR scheme, no concrete performance analysis is currently available for this technique.

Verifiable PIR with small client storage. Rathee et al., [RLP26] proposes a verifiable PIR scheme in the single server setting that has the advantage of having very small client storage requirements. This work is similar in spirit to VeriSimplePIR [dCL24] which has large client storage requirements. One of the main ideas of the paper to reduce costs is to make use of the FHE-SNARK paradigm instead of the SNARK-FHE paradigm. This approach consists in adding verifiability to an FHE-delegated computation on the *plaintext* domain instead of on the *ciphertext* domain. As noted by the authors of [RLP26] and carefully discussed in [ZWL+25], verifiable and private computation delegation using FHE-SNARK is more efficient but strictly less secure than using SNARK-FHE. In particular, proofs of correct computation in FHE-SNARK are not publicly verifiable - the verifier must first decrypt the proof before being able to check its validity, leading to active attacks leaking information about the client's secret data. While the authors of [RLP26] argue that this weaker security notion is acceptable in their protocol setting, it remains that a malicious server can leverage this vulnerability to mount repudiable denial-of-service (DoS) attacks against the client. More precisely, a malicious server can return an incorrect response to the client's query, along with a proof that the designated verifier rejects. The server can then publicly deny having sent an incorrect response. To show the public that the server truly misbehaved, the client must reveal information about its secret key to the public, which is a severe drawback. Admittedly, the authors explicitly state that their protocol does not provide security against any DoS attacks, in particular repudiable DoS attacks as described above. We still take care to highlight this point as it is an important limitation of their work in certain application settings.

Comparison to this work. Our work differs from [CNCG+23] in that it sets itself in the fully malicious setting. As such, it is closer in spirit to [dCL24,DT24,FMS25].

Our improvement upon [dCL24] is twofold:

1. Our digest generation phase needs only to be run once, independently of the number of subsequent queries and even if malicious behavior is detected later on, and its output can be verified by anyone;
2. The digest output from our digest generation phase is significantly smaller. As such, the client storage requirements are importantly reduced.

Both this work and [dCL24] provide security against fully malicious adversaries and give proofs in the random oracle model using standard lattice assumptions. As such, both works are plausibly post-quantum secure.

Compared to [DT24], our work provides the same theoretical tradeoff as [dCL24] in that it relies on stronger cryptographic assumptions but provides plausible post-quantum security. Furthermore, our approach replaces the need for a validation phase during which the client sends λ additional queries to the server by a one-time preprocessing phase.

Our work provides an instantiation of a maliciously secure PIR scheme that is concretely efficient in practice. In contrast, the work of [FMS25] does not exhibit concretely efficient performance. In other words, our contribution is orthogonal to theirs: their compiler could be applied to our underlying PIR scheme to obtain a maliciously secure PIR scheme, the concrete performance analysis of which would be an interesting avenue for future work.

Both [WLZ+23] and [AR26] propose constructions in the "amortized verifiable PIR" setting where query-dependent computation is offloaded to the preprocessing phase. This comes with its own drawbacks, namely that preprocessing phase has to be run multiple times and that the client may experience significant delays before being able to expose a malicious server's misbehaviour [AR26]. Our work rather focuses on the standard verifiable PIR setting where there is only one preprocessing phase, independent of the number of queries.

Lastly, compared to [RLP26], our work sets itself in the SNARK-FHE paradigm, providing stronger security guarantees at the cost of some efficiency. In particular, our protocol is secure against repudiable DoS attacks, which is not the case in [RLP26]. This forces any potentially malicious server to either behave correctly, be publicly caught misbehaving while preserving client privacy or not answer at all.

1.2 Contributions

Our contributions are as follows:

1. We give an Interactive Oracle Proof (IOP) for matrix-vector products $\mathbf{M} \times \mathbf{c} = \mathbf{c}'$ over power-of-two cyclotomic rings. Writing the block-circulant matrix $\mathbf{N} \in \mathbb{F}_q^{md \times nd}$ as $\widetilde{\mathbf{N}}(w,x,y,z) = \sum_l \widetilde{\mathbf{F}'}(x,l) \cdot \widetilde{\mathbf{M}}(w,y,l) \cdot \widetilde{\mathbf{F}}(l,z)$, the prover commits only to the NTT representation $\bar{\mathbf{M}}$ and never materializes $\mathbf{N}$. Verification reduces to two nested sum-checks and a single evaluation query to $\bar{\mathbf{M}}$.
2. We extend the IOP to prove sample extraction and batch all ciphertext components at $\mathcal{O}(\kappa/|\mathcal{C}|)$ soundness cost, halving the answer size at no extra prover cost.

3. We define verifiable PIR with a one-time, publicly verifiable preprocessing and build a scheme realizing it under GLWE in the ROM. Security holds against repeatedly malicious servers and preprocessing is never rerun.
4. We provide implementation-level optimizations leveraging the algebraic structure of cyclotomic rings, an open-source implementation exploiting those insights as well as a comprehensive experimental evaluation of our protocol. We conclude by discussing our results and comparing them with those from comparable prior works.

2 Technical overview

In this section we present the main ideas behind our results. One of the key contributions of this work is the design of a new sum-check based protocol for proving the correctness of linear relations over rings. More precisely, letting $\mathcal{R}_q$ be a power-of-two cyclotomic ring over $\mathbb{F}_q$ and given a fixed public $a \in \mathcal{R}_q$ and a user-provided $b \in \mathcal{R}_q$, we aim to convince a verifier that $a \cdot b = c$ over $\mathcal{R}_q$.
A straightforward approach to achieve this is to rely on the observation that the product of two polynomials can be expressed as a linear algebraic relation over their coefficients. More precisely, let $\mathbf{a} = (a_0, \ldots, a_{d-1})$ and $\mathbf{b} = (b_0, \ldots, b_{d-1})$ denote the coefficient vectors of a and b respectively. Then, we have $a \cdot b = c$ if and only if $\mathsf{circ}(\mathbf{a}) \times \mathbf{b} = \mathbf{c}$ where $\mathsf{circ}(\mathbf{a}) \in \mathbb{F}_q^{d \times d}$. The straightforward approach works as follows. A prover can first commit to the multilinear extension of the circulant matrix $\mathbf{A} \coloneqq \mathsf{circ}(\mathbf{a})$. Then, upon reception of a query $b \in \mathcal{R}_q$ from the verifier, the prover computes $\mathbf{c}$ and sends it to the verifier. The verifier then samples a random challenge ρ from an appropriately chosen challenge space $\mathcal{C}$ and sends it to the prover. The prover and verifier engage in a sum-check protocol to prove the following statement:

$$\widetilde{\mathbf{c}}(\rho) = \sum_{\mathbf{i} \in \{0,1\}^{\log d}} \widetilde{\mathbf{A}}(\rho, \mathbf{i}) \cdot \widetilde{\mathbf{b}}(\mathbf{i}). \tag{1}$$

At the end of the sum-check protocol, the verifier is convinced that $c = a \cdot b$ provided $\widetilde{\mathbf{A}}(\rho, \mathbf{r}) \cdot \widetilde{\mathbf{b}}(\mathbf{r}) = s$ for a random $\mathbf{r} \in \mathcal{C}^{\log d}$ and $s \in \mathcal{C}$ defined throughout the execution of the protocol. This can be accomplished by having the prover use the commitment scheme to provide an evaluation proof of $\widetilde{\mathbf{A}}$ at the point $(\rho, \mathbf{r})$. The verifier can compute $\widetilde{\mathbf{b}}(\mathbf{r})$ efficiently itself and check that the relation holds.

Notice that this straightforward approach requires the prover to commit to and evaluate the multilinear extension of the $d \times d$ circulant matrix $\mathbf{A}$, requiring $\mathcal{O}(d^2)$ operations in general and incurring significant overhead when generating evaluation proofs. We devise a new protocol that avoids this overhead by leveraging the structure of the circulant matrix $\mathbf{A}$ as well as the structure of multilinear extensions. The protocol works as follows.
Let $\mathbf{F}$ and $\mathbf{F}'$ be the $d \times d$ matrices such that $\mathbf{F} \times \mathbf{a} = \mathsf{NTT}(a)$ and $\mathbf{F}' \times \mathbf{a} = \mathsf{NTT}^{-1}(a)$. Observe that

$$\mathbf{A} = \mathbf{F}' \times \mathsf{diag}(\mathsf{NTT}(a)) \times \mathbf{F}$$

where $\mathsf{diag}(\mathsf{NTT}(a))$ is the diagonal matrix containing the NTT representation of a. Since $\mathbf{F}$ and $\mathbf{F}'$ are public and fixed, the prover can simply commit to the multilinear extension of the vector $\mathsf{NTT}(a)$ instead of the matrix $\mathbf{A}$. Then, upon reception of a query $b \in \mathcal{R}_q$ from the verifier, the prover proceeds as before: it computes $\mathbf{c}$ and sends it to the verifier, receives a random challenge ρ from the verifier and engages in a sum-check protocol to prove the statement defined in Equation (1). Again, at the end of the sum-check protocol, the verifier is convinced that $c = a \cdot b$ provided $\widetilde{\mathbf{A}}(\rho, \mathbf{r}) \cdot \widetilde{\mathbf{b}}(\mathbf{r}) = s$ for a random $\mathbf{r} \in \mathcal{C}^{\log d}$ and $s \in \mathcal{C}$ defined throughout the execution of the protocol. Let $\widetilde{\widehat{\mathbf{a}}}$ be the multilinear extension of the NTT representation of a. This time around, we leverage the fact that

$$\widetilde{\mathbf{A}}(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{i} \in \{0,1\}^{\log d}} \widetilde{\mathbf{F}'}(\mathbf{x}, \mathbf{i}) \cdot \widetilde{\widehat{\mathbf{a}}}(\mathbf{i}) \cdot \widetilde{\mathbf{F}}(\mathbf{i}, \mathbf{y})$$

to have the prover compute the value $\widetilde{\mathbf{A}}(\rho, \mathbf{r})$ in time $\mathcal{O}(d)$ and send it to the verifier. The verifier can then complete the verification of the previous sum-check protocol execution. It remains for the prover to convince the verifier that $\widetilde{\mathbf{A}}(\rho, \mathbf{r})$ is correctly computed. This can be done by having the prover and verifier engage in a second sum-check protocol to prove

$$\widetilde{\mathbf{A}}(\rho, \mathbf{r}) = \sum_{\mathbf{i} \in \{0,1\}^{\log d}} \widetilde{\mathbf{F}'}(\rho, \mathbf{i}) \cdot \widetilde{\widehat{\mathbf{a}}}(\mathbf{i}) \cdot \widetilde{\mathbf{F}}(\mathbf{i}, \mathbf{r}).$$

Now the prover can simply use the commitment scheme to provide an evaluation proof of $\widetilde{\widehat{\mathbf{a}}}$ at the point $\mathbf{r}'$ chosen by the verifier during the second sum-check and the verifier can compute $\widetilde{\mathbf{F}'}(\rho, \mathbf{r}')$ and $\widetilde{\mathbf{F}}(\mathbf{r}', \mathbf{r})$ efficiently itself to perform the final check of this execution of the sum-check protocol. Naturally, this new protocol is much more efficient in terms of prover time, proof size and verifier time compared to the straightforward approach described above. This is due to the fact that the sum-check protocol is extremely efficient and yields very small proofs whereas the main performance costs come from the PCS evaluation proofs.

In Section 5, we extend and detail the above ideas to matrix-vector products over rings, which is an important building block for our actively secure vPIR protocol.

3 Preliminaries

3.1 Notation

Let q be a prime and d a power of two. We let $\mathcal{R}$ denote the polynomial ring $\mathbb{Z}[X]/\langle X^d+1\rangle$ and $\mathcal{R}_q \coloneqq \mathcal{R}/q\mathcal{R}$. For a ring element $\sum_{i=0}^{d-1} a_i X^i \in \mathcal{R}_q$, we denote by $\mathsf{coeff}(a)$ the vector of coefficients $(a_0, \ldots, a_{d-1}) \in \mathbb{F}_q^d$. For a divisor t of d and a prime q satisfying $q = 1 + 2t \bmod 4t$, $\mathbb{F}_q$ has t primitive $2t$-th roots of unity $(\zeta_j)_{j=1}^t$. Furthermore, $X^d + 1 = \prod_{j=1}^t (X^{d/t} - \zeta_j) \bmod q$ is a factorization into irreducible factors $\{(X^{d/t} - \zeta_j)\}_j$. Setting $\tau = d/t$, it follows by the Chinese Remainder Theorem that $\mathcal{R}_q \cong \mathbb{F}_{q^\tau}^t$ where addition and multiplication in $\mathbb{F}_{q^\tau}^t$

are performed component-wise. This isomorphism is realized by the number theoretic transform (NTT). In words, we say that $\mathcal{R}_q$ splits into t fields $\mathbb{F}_{q^\tau}$, each of degree τ. When $t = d$, we have $\mathcal{R}_q \cong \mathbb{F}_q^d$ and say that the ring $\mathcal{R}_q$ fully splits — in what follows, we will stay in this regime. For some $r \in \mathcal{R}_q$, we denote by $\mathsf{NTT}(r) \in \mathbb{F}_q^d$ the row vector containing its NTT representation. For a matrix $\mathbf{M} \in \mathcal{R}_q^{m \times n}$, we denote by $\mathsf{NTT}(\mathbf{M})$ the matrix composed of the NTT representation of each element, i.e. $\mathsf{NTT}(\mathbf{M})_{i,k} \coloneqq \mathsf{NTT}(\mathbf{M}_{i,k}) \in (\mathbb{F}_q^d)^{m \times n}$. We use $\lfloor \cdot \rceil$ to denote rounding to the nearest integer. Let $\mathbf{u} \in \mathbb{F}_q^{\log n}$. We define $\mathbf{u} \circ \mathbf{v} = (\mathbf{u}_1 \cdot \mathbf{v}_1, \ldots, \mathbf{u}_{\log n} \cdot \mathbf{v}_{\log n})$ and $\mathsf{tensor}(\mathbf{u}) \coloneqq \bigotimes_{i=1}^{\log n} [(1 - \mathbf{u}_i), \mathbf{u}_i] \in \mathbb{F}_q^n$ and denote the skewed circulant matrix defined by $\mathbf{v} \in \mathbb{F}_q^n$ as

$$\mathsf{circ}(\mathbf{v}) \coloneqq \begin{pmatrix} \mathbf{v}_1 & -\mathbf{v}_n & -\mathbf{v}_{n-1} & \cdots & -\mathbf{v}_2 \\ \mathbf{v}_2 & \mathbf{v}_1 & -\mathbf{v}_n & \cdots & -\mathbf{v}_3 \\ \mathbf{v}_3 & \mathbf{v}_2 & \mathbf{v}_1 & \cdots & -\mathbf{v}_4 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{v}_n & \mathbf{v}_{n-1} & \mathbf{v}_{n-2} & \cdots & \mathbf{v}_1 \end{pmatrix}$$

We denote by $\mathcal{C}$ a challenge space from which random values are sampled for polynomial testing and during the sum-check protocol. In practice, $\mathcal{C}$ will be an extension field of $\mathbb{F}_q$ of degree δ large enough to ensure appropriate soundness. As such, field operations between elements of $\mathcal{C}$ and $\mathbb{F}_q$ are well-defined and always implicitly performed in $\mathcal{C}$.

3.2 (G)LWE encryption

Definition 1 (GLWE encryption scheme [LPR13]). *Let q be a prime ciphertext modulus, $p < q$ a plaintext modulus, d a power of two cyclotomic ring degree and κ the GLWE dimension parameter. We let χ_s and χ_e denote the secret key and error distributions over $\mathcal{R}_q$ respectively. Finally, we let $\mathcal{U}$ denote the uniform distribution, $\mathcal{R}_p$ denote the message space of the encryption scheme and $\Delta \coloneqq \lfloor q/p \rceil$. An GLWE encryption scheme $\mathcal{E} \coloneqq (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ consists of the following algorithms:*

- $\mathsf{KGen}(1^\lambda)$*: Takes as input a security parameter 1^λ and returns a sampled secret key $\mathbf{s} \xleftarrow{\$} \chi_s^\kappa$.*
- $\mathsf{Enc}(\mathbf{s}, m)$*: Takes as input a secret key $\mathbf{s}$ and a message $m \in \mathcal{R}_p$ and returns a ciphertext $\mathbf{c} \in \mathcal{R}_q^{\kappa+1}$ computed as follows:*

$$\mathbf{a} \xleftarrow{\$} \mathcal{U}(\mathcal{R}_q)^\kappa, \quad e \xleftarrow{\$} \chi_e, \quad \mathbf{c} = (\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e + \Delta m) \in \mathcal{R}_q^{\kappa+1}.$$

- $\mathsf{Dec}(\mathbf{s}, \mathbf{c})$*: Takes as input a secret key $\mathbf{s}$ and a ciphertext $\mathbf{c} = (\mathbf{a}, b) \in \mathcal{R}_q^{\kappa+1}$ and returns a message $m \in \mathcal{R}_p$ computed as follows:*

$$m = \left\lfloor \frac{b - \langle \mathbf{a}, \mathbf{s} \rangle}{\Delta} \right\rceil$$

This encryption scheme is secure under the GLWE assumption for appropriate choices of parameters. For a set of parameters, its security can be estimated using the widely used lattice estimator [APS15].

The above encryption scheme is additively homomorphic and allows for multiplication by plaintext elements, provided the noise remains "small". More precisely, given two ciphertexts $\mathbf{c}_1 = (\mathbf{a}_1, b_1)$ and $\mathbf{c}_2 = (\mathbf{a}_2, b_2)$ encrypting messages $m_1, m_2 \in \mathcal{R}_p$ respectively, we have:

- Addition: $\mathbf{c}_{\text{add}} = \mathbf{c}_1 + \mathbf{c}_2 = (\mathbf{a}_1 + \mathbf{a}_2, b_1 + b_2)$ is a ciphertext encrypting $m_{\text{add}} = m_1 + m_2$.
- Scalar multiplication: For any plaintext $y \in \mathcal{R}_p$, $\mathbf{c}_{\text{scal}} = y \cdot \mathbf{c}_1 = (y \cdot \mathbf{a}_1, y \cdot b_1)$ is a ciphertext encrypting $m_{\text{scal}} = y \cdot m_1$.

These properties hold provided the accumulated noise in the resulting ciphertexts remains small enough to allow for correct decryption, i.e. $\|e/\Delta\|_\infty \leq 1/2$.

LWE encryption scheme. The above definition is a generalization of the well-known LWE encryption scheme, which is obtained by setting $d = 1$.

Sample extraction. Given a ciphertext $\mathbf{c} = (\mathbf{a}, b) \in \mathcal{R}_q^{\kappa+1}$ encrypting a plaintext $m \in \mathcal{R}_p$, it is possible to extract a LWE sample c' encrypting the i-th coefficient of m for any $i \in \{0, \ldots, d-1\}$. The sample extraction procedure is defined as follows. Given $(\mathbf{a}, b) \in \mathcal{R}_q^{\kappa+1}$, we define the extracted sample $c' = (\mathbf{a}', b')$ for coefficient i as

$$\begin{aligned}
\mathbf{a}'_{k\cdot d+j} &\coloneqq \mathbf{a}_{k,i-j} && \text{for } k \in \{0, \ldots, \kappa-1\}, j \in \{0, \ldots, i\} \\
\mathbf{a}'_{k\cdot d+j} &\coloneqq -\mathbf{a}_{k,i-j+d} && \text{for } k \in \{0, \ldots, \kappa-1\}, j \in \{i+1, \ldots, d-1\} \\
b' &\coloneqq b_i
\end{aligned}$$

It is easy to see that c' is a valid LWE encryption of m_i under the secret key $\mathbf{s}' = (\mathbf{s}_{0,0}, \ldots, \mathbf{s}_{0,d-1}, \ldots, \mathbf{s}_{\kappa-1,0}, \ldots, \mathbf{s}_{\kappa-1,d-1})$ with error term $e' = e_i$.

3.3 Multilinear extensions, sum-check and polynomial commitment schemes

Definition 2 (Multilinear extension). *Let $f : \{0,1\}^\nu \to \mathbb{F}$ be any function. We denote by $\widetilde{f}$ its* multilinear extension *defined as the unique multilinear polynomial $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\nu]$ such that $\widetilde{f}(\mathbf{b}) = f(\mathbf{b})$ for all $\mathbf{b} \in \{0,1\}^\nu$. $\widetilde{f} : \mathbb{F}^\nu \to \mathbb{F}$ can be explicitly written as*

$$\widetilde{f}(X_1, \ldots, X_\nu) \coloneqq \sum_{\mathbf{b} \in \{0,1\}^\nu} f(\mathbf{b}) \cdot \mathsf{eq}(\mathbf{b}, \mathbf{X})$$

where $\mathsf{eq}(\mathbf{b}, \mathbf{X}) \coloneqq \prod_{i=1}^{\nu} [(1 - b_i)(1 - X_i) + b_i X_i]$.

Lemma 1. *(DeMillo-Lipton-Schwartz-Zippel lemma [DL78,Zip79,Sch80]) Let $\mathbb{F}$ be a field and let $p : \mathbb{F}^\nu \to \mathbb{F}$ be a non-zero ν-variate polynomial of total degree $\leq \mathsf{deg}$.*

$$\Pr_{\mathbf{r} \xleftarrow{\$} \mathbb{F}^\nu} [p(\mathbf{r}) = 0] \leq \frac{\mathsf{deg}}{|\mathbb{F}|}$$

We next recall the description of the sum-check protocol, which is a fundamental tool for our results.

Definition 3 (Sum-check protocol [LFKN92]). *Let $\mathbb{F}$ be a field, ν a positive integer and $p : \mathbb{F}^\nu \to \mathbb{F}$ a ν-variate polynomial of maximum individual degree deg. The sum-check protocol is an interactive proof between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ to prove the following claim:*

$$S \coloneqq \sum_{\mathbf{b} \in \{0,1\}^\nu} p(\mathbf{b}).$$

The protocol reduces the veracity of this statement to the evaluation of p at a random point $\mathbf{r} \xleftarrow{\$} \mathcal{C}^\nu$. It has soundness error $\nu\mathsf{deg}/|\mathcal{C}|$ where $\mathcal{C}$ is the challenge space from which $\mathbf{r}$ is sampled. We give the full description of the protocol in Appendix A.1.

Definition 4 (Polynomial commitment scheme). *A polynomial commitment scheme PCS over a field $\mathbb{F}$ is a quadruple of PPT algorithms $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ such that:*

- $\mathsf{Gen}(1^\lambda) \to \mathsf{pp}$ *takes as as input security parameters and outputs public parameters* pp.
- $\mathsf{Commit}(\mathsf{pp}, f) \to c$ *takes as input the public parameters and a polynomial $f \in \mathbb{F}^{\leq d}[X]$ of degree at most d and outputs a commitment c to this polynomial.*
- $\mathsf{Open}(\mathsf{pp}, c, f, x, y) \to \pi$ *takes as input the public parameters, a commitment c, a polynomial $f \in \mathbb{F}^{\leq d}[X]$, an evaluation point $x \in \mathbb{F}$ and a claimed evaluation $y \in \mathbb{F}$ of f at x and outputs a proof of evaluation π.*
- $\mathsf{Verify}(\mathsf{pp}, c, x, y, \pi) \to \mathsf{Accept}/\mathsf{Reject}$ *takes as input the public parameters, a commitment c, an evaluation point x and a claimed evaluation y of f at x and a proof of evaluation π and accepts or rejects.*

Satisfying the following properties:

1. ***Completeness**: For any polynomial $f \in \mathbb{F}^{\leq d}[X]$ and all domain elements x,*

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, c, x, y, \pi) = \mathsf{Accept} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ c \leftarrow \mathsf{Commit}(\mathsf{pp}, f) \\ y \leftarrow f(x) \\ \pi \leftarrow \mathsf{Open}(\mathsf{pp}, c, f, x, y) \end{array}\right] = 1.$$

2. ***Evaluation binding**: For any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} y \neq y' \wedge \\ \mathsf{Verify}(\mathsf{pp}, c, x, y, \pi) = \mathsf{Accept} \wedge \\ \mathsf{Verify}(\mathsf{pp}, c, x, y', \pi') = \mathsf{Accept} \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ (c, x, y, y', \pi, \pi') \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] \in \mathsf{negl}(\lambda).$$

3. ***Knowledge soundness****: For any interactive deterministic polynomial time $\mathcal{P}^*$, there exists an expected polynomial time extractor $\mathsf{Ext}^{\mathcal{P}^*}$ such that for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(\mathsf{pp}, c, x, y, \pi) = \mathsf{Accept} \wedge & \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ (c \notin \mathit{Im}(\mathsf{Commit}(\mathsf{pp}, f)) \vee y \neq f(x)) & (c, x, y, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ & \pi \leftarrow \mathcal{P}^*(\mathsf{pp}, c, x, y, \mathsf{st}_{\mathcal{A}}) \\ & f \leftarrow \mathsf{Ext}^{\mathcal{P}^*(\mathsf{pp}, c, x, y, \mathsf{st}_{\mathcal{A}})}(\mathsf{pp}, c, x, y, \pi) \end{array}\right] \in \mathsf{negl}(\lambda).$$

4 Passively secure PIR from plaintext matrix - encrypted vector multiplication

In this section we recall a passively secure private computation delegation and show how it directly implies a passively secure PIR protocol. The delegated computation in question is the multiplication of a plaintext matrix by an encrypted vector, where the encryption scheme is linearly homomorphic. The protocol is borrowed from [FBK+25].

Plaintext matrix-encrypted vector multiplication. The GLWE-based encryption scheme presented in Definition 1 encrypts message vectors $\mathbf{x} \in \mathbb{Z}_p^d$ (where p is the plaintext modulus) by viewing them as polynomials $x \in \mathcal{R}_p = \mathbb{Z}_p[X]/\langle X^d + 1\rangle$ using the natural coefficient embedding. The ciphertexts are additively homomorphic and allow for multiplication by plaintext elements $y \in \mathcal{R}_p$ resulting in a ciphertext encrypting $x \cdot y$. The key point of the multiplication algorithm from [FBK+25] is the observation that the constant term of $z = x \cdot y$ can be written as

$$z_0 = \langle \mathbf{x}, (y_0, -y_{d-1}, -y_{d-2}, \ldots, -y_1)\rangle$$

Let $\tau : \mathbb{Z}_p^d \to \mathcal{R}_q$ be the encoding such that $\mathbf{y} \mapsto y_0 - \sum_{i=1}^{d-1} y_{d-i} X^i$. Then $\langle \mathbf{x}, \mathbf{y}\rangle = (x \cdot \tau(\mathbf{y}))_0$ where $x \in \mathcal{R}_q$ is the natural embedding of $\mathbf{x}$. We extend τ to $\tau_d : \mathbb{Z}_p^{nd} \mapsto \mathcal{R}_p^n$ to vectors of size nd in the natural way, i.e. by applying τ to subvectors of dimension d and concatenating the result into a vector of dimension n over $\mathcal{R}_p$. By also naturally encoding the message $\mathbf{x} \in \mathbb{Z}_p^{nd}$ into n elements $x = (x_1, x_2, \ldots, x_n) \in \mathcal{R}_p^n$, the inner product computation becomes

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

Given a matrix $\mathbf{A} \in \mathbb{Z}_p^{m \times nd}$ and a vector $\mathbf{x} \in \mathbb{Z}_p^{nd}$, the matrix multiplication algorithm now follows naturally by applying the inner product computation row-wise.

In summary, the algorithm is as follows:

1. The client encodes $\mathbf{x} \in \mathbb{Z}_p^{nd}$ into $x = (x_1, x_2, \ldots, x_n) \in \mathcal{R}_p^n$ and encrypts each of the n message polynomials, obtaining a ciphertext vector $\mathbf{c}$ which it sends to the server.
2. The server computes $\tau_d(\mathbf{A}_i) \cdot x$ for each row $\mathbf{A}_i$ of the matrix, yielding in a vector of ciphertexts $\mathbf{c}'$. Finally, the server sends the ciphertexts to the client.
3. The client decrypts the ciphertexts to obtain the desired result.

Passively secure PIR. It is easy to see that the above protocol directly implies a passively secure PIR protocol where the database is the plaintext matrix $\mathbf{A} \in \mathbb{Z}_p^{m \times nd}$ and the database elements are its columns $\mathbf{A}_{*,j} \in \mathbb{Z}_p^m$ for $j \in [nd]$. The client can encrypt its selection vector $\mathbf{u}_j \in \{0,1\}^{nd}$ of Hamming weight 1 resulting in a vector of n ciphertexts $\mathbf{c} \coloneqq (c_1, \ldots, c_n)$ as above. The server can then answer a PIR query $\mathbf{c}$ by homomorphically multiplying it by each encoded row $\tau_d(\mathbf{A}_i)$ of the database $\mathbf{A}$ and returning the result to the client for decryption. The client recovers the database element $\mathbf{A}_{*,j}$ from the decrypted result. Informally, one can verify that this PIR protocol is secure and correct provided GLWE is assumed to be hard, the error distribution is chosen appropriately and the database entries lie in $\mathbb{Z}_p$. Looking ahead, the last condition must be enforced explicitly in the verifiable setting, where the server may commit to a database with entries outside of $\mathbb{Z}_p$ (cf. Section 6).

In the next section, we design a verifiable computation protocol which allows to prove the correct execution of the server algorithm from the above matrix-vector multiplication protocol. As we will see in the subsequent section, this in turn allows to upgrade the above passively secure PIR protocol to a verifiable PIR protocol secure against fully malicious adversaries.

5 Verifiability for matrix-vector product over rings

Consider the passively secure protocol proposed in the previous section for a plaintext matrix $\mathbf{A}$ and its encoding $\mathbf{M} \coloneqq \tau(\mathbf{A})$. There are two known approaches to efficiently proving the correctness of the computation performed by the server. The first approach is to use a succinct argument for arithmetic circuits over $\mathcal{R}_q$ as proposed by [HMZ25] and applied in [TWZ26]. The second approach is to express $\mathcal{R}_q$-arithmetic as arithmetic over d-dimensional vectors over $\mathbb{F}_q$ and use a SNARK for arithmetic circuits over $\mathbb{F}_q$. In this work we focus on the latter approach which remains poorly explored. Let

$$\begin{pmatrix} \mathbf{a}_{1,1} & \mathbf{a}_{1,2} & \cdots & \mathbf{a}_{1,\kappa} & b_1 \\ \mathbf{a}_{2,1} & \mathbf{a}_{2,2} & \cdots & \mathbf{a}_{2,\kappa} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{a}_{n,1} & \mathbf{a}_{n,2} & \cdots & \mathbf{a}_{n,\kappa} & b_n \end{pmatrix} \in \mathcal{R}_q^{n \times (\kappa+1)}$$

be the ciphertext vector sent by the client. To keep full generality, we assume that the ciphertexts sent by the client are in the coefficient representation as opposed to the NTT representation. For ease of exposition, we focus on proving the correctness of the operation applied to a vector of only one ciphertext component, e.g. $\mathbf{c} = (\mathbf{a}_{1,1}, \mathbf{a}_{2,1} \ldots \mathbf{a}_{n,1})^\top \in \mathcal{R}_q^n$. We will see later how to extend this to all $\kappa + 1$ components.

5.1 Statement to prove

As observed above, the computation to be proven correct consists of a matrix-vector product over $\mathcal{R}_q$ of the form $\mathbf{M} \times \mathbf{c} = \mathbf{c}'$. We rely on the cyclotomic

structure of $\mathcal{R}_q$ to express this matrix-vector product as a linear algebraic relation over $\mathbb{F}_q$. It is a known fact that for any $a, b, c \in \mathcal{R}_q$ and $\mathbf{a}, \mathbf{b}, \mathbf{c}$ their respective coefficient vectors, we have $a \cdot b = c$ if and only if $\mathsf{circ}(\mathbf{a}) \times \mathbf{b} = \mathbf{c}$. As such, we can reduce the matrix-vector product $\mathbf{M} \times \mathbf{c} = \mathbf{c}'$ over $\mathcal{R}_q$ to an expression over $\mathbb{F}_q$ as follows:

$$\sum_{k \in [n]} \mathsf{circ}(\mathbf{M}_{i,k}) \times \mathsf{coeff}(\mathbf{c}_k) = \mathsf{coeff}(\mathbf{c}'_i)$$

for all $i \in [m]$. Let $\mathbf{N}$ be the block matrix in $\mathbb{F}_q^{md \times nd}$ defined as $\mathbf{N}_{i,k} \coloneqq \mathsf{circ}(\mathbf{M}_{i,k})$. The above relation can be rewritten as follows:

$$\sum_{k \in [n]} \mathbf{N}_{i,k} \times \mathsf{coeff}(\mathbf{c}_k) = \mathsf{coeff}(\mathbf{c}'_i). \tag{2}$$

for all $i \in [m]$. Taking the multilinear extensions of $\mathbf{N}, \mathbf{c}$ and $\mathbf{c}'$, which are respectively $\log(mdnd)$, $\log(nd)$ and $\log(md)$-variate, we can encode these md statements into a single statement over multivariate polynomials:

$$\widetilde{\mathbf{c}}'(\mathbf{w}, \mathbf{x}) = \sum_{\mathbf{k} \in \{0,1\}^{\log n}, \mathbf{l} \in \{0,1\}^{\log d}} \widetilde{\mathbf{N}}(\mathbf{w}, \mathbf{x}, \mathbf{k}, \mathbf{l}) \cdot \widetilde{\mathbf{c}}(\mathbf{k}, \mathbf{l}) \tag{3}$$

for all $\mathbf{w}, \mathbf{x} \in \{0,1\}^{\log m + \log d}$, which will be subjected to the sum-check protocol as described in what follows.

5.2 The Protocol

In this section we provide an IOP for the correctness of the matrix-vector product $\mathbf{M} \times \mathbf{c} = \mathbf{c}'$. Let $\bar{\mathbf{M}} \in \mathbb{F}_q^{m \times nd}$ be the NTT representation of $\mathbf{M} \in \mathcal{R}_q^{m \times n}$. We assume the verifier has oracle access to the $\log mnd$-variate multilinear extension of $\bar{\mathbf{M}}$, denoted $\widetilde{\bar{\mathbf{M}}}$ and we use $\mathcal{C}$ to denote a sufficiently large challenge space. The IOP works as follows. First, the prover computes $\mathbf{c}' = \mathbf{M} \times \mathbf{c}$ and sends $\mathbf{c}'$ to the verifier. The verifier then samples a random challenge $\rho \xleftarrow{\$} \mathcal{C}^{\log(md)}$ which it sends to the prover. Both parties then run the sum-check protocol for the following statement:

$$\widetilde{\mathbf{c}}'(\rho) = \sum_{\mathbf{k} \in \{0,1\}^{\log n}, \mathbf{l} \in \{0,1\}^{\log d}} \widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l}) \cdot \widetilde{\mathbf{c}}(\mathbf{k}, \mathbf{l}). \tag{4}$$

By the DeMillo-Lipton-Schwartz-Zippel lemma, if Equation (4) holds then with overwhelming probability Equation (3) also holds, provided $|\mathcal{C}|$ is large enough. The sum-check protocol reduces the correctness of Equation (4) to checking that

$$\widetilde{\mathbf{N}}(\rho, \sigma) \cdot \widetilde{\mathbf{c}}(\sigma) = s \tag{5}$$

where $\sigma \in \mathcal{C}^{\log(nd)}$ and $s \in \mathcal{C}$ are defined throughout the execution of the protocol. For the verifier to check Equation (5), it must possess the values $\widetilde{\mathbf{N}}(\rho, \sigma)$

and $\widetilde{\mathbf{c}}(\sigma)$. It can compute the latter efficiently by itself, but the prover needs to help by providing the former evaluation and later proving the function $\widetilde{\mathbf{N}}$ was evaluated correctly.
To this end, the prover sends $\widetilde{\mathbf{N}}(\rho, \sigma)$ to the verifier. Before describing how the prover proceeds to prove that the provided evaluation is correct, we first give a helpful expression of $\widetilde{\mathbf{N}}(\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z})$. Let ω denote a primitive $2d$-th root of unity in $\mathbb{F}_q$ and let $\mathbf{F} \in \mathbb{F}_q^{d \times d}$ denote the NTT matrix defined as follows:

$$\begin{pmatrix} 1 & \omega^1 & \omega^2 & \cdots & \omega^{d-1} \\ 1 & \omega^3 & \omega^6 & \cdots & \omega^{3(d-1)} \\ 1 & \omega^5 & \omega^{10} & \cdots & \omega^{5(d-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2d-1} & \omega^{(2d-1)2} & \cdots & \omega^{(2d-1)(d-1)} \end{pmatrix}$$

Similarly, let $\mathbf{F}' \in \mathbb{F}_q^{d \times d}$ denote the inverse NTT matrix defined as follows:

$$\frac{1}{d} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ \omega^{-1} & \omega^{-3} & \omega^{-5} & \cdots & \omega^{-(2d-1)} \\ \omega^{-2} & \omega^{-6} & \omega^{-10} & \cdots & \omega^{-(2d-1)2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega^{-(d-1)} & \omega^{-3(d-1)} & \omega^{-5(d-1)} & \cdots & \omega^{-(2d-1)(d-1)} \end{pmatrix}$$

Observe that for any $a \in \mathcal{R}_q$, we have that

$$\mathsf{circ}(a) = \mathbf{F}' \times \mathsf{diag}(\mathsf{NTT}(a)) \times \mathbf{F}$$

where $\mathsf{diag}(\mathsf{NTT}(a))$ is the diagonal matrix defined by the NTT representation of a. Thus, the block matrix $\mathbf{N}$ defined as

$$\mathbf{N}_{i,k} = \mathsf{circ}(\mathbf{M}_{i,k}) = \mathbf{F}' \times \mathsf{diag}(\bar{\mathbf{M}}_{i,k}) \times \mathbf{F}$$

has a multilinear extension $\widetilde{\mathbf{N}}$ that can be expressed as follows:

$$\widetilde{\mathbf{N}}(\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}) \coloneqq \sum_{\mathbf{l}} \widetilde{\mathbf{F}}'(\mathbf{x}, \mathbf{l}) \cdot \widetilde{\bar{\mathbf{M}}}(\mathbf{w}, \mathbf{y}, \mathbf{l}) \cdot \widetilde{\mathbf{F}}(\mathbf{l}, \mathbf{z}). \tag{6}$$

where $\widetilde{\mathbf{F}}$ and $\widetilde{\mathbf{F}}'$ are the $2 \log d$-variate multilinear extensions of $\mathbf{F}$ and $\mathbf{F}'$ respectively. As such, letting $\rho = (\rho_{\mathsf{outer}}, \rho_{\mathsf{inner}})$ and $\sigma = (\sigma_{\mathsf{outer}}, \sigma_{\mathsf{inner}})$, we have that the value sent by the prover to help the verifier check Equation (5) is exactly

$$\widetilde{\mathbf{N}}(\rho, \sigma) = \sum_{\mathbf{l} \in \{0,1\}^{\log d}} \widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \mathbf{l}) \cdot \widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{l}) \cdot \widetilde{\mathbf{F}}(\mathbf{l}, \sigma_{\mathsf{inner}}) \tag{7}$$

This evaluation fits perfectly within the sum-check protocol framework and the prover and verifier can run a second execution of this protocol to prove that Equation (7) holds. This reduces the correctness of Equation (7) to checking that

$$\widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \tau) \cdot \widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau) \cdot \widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}}) = t \tag{8}$$

for some $\tau \in \mathcal{C}^{\log d}$ and $t \in \mathcal{C}$ defined throughout the execution of the protocol. By assumption, the verifier has oracle access to the multilinear extension $\widetilde{\widetilde{\mathbf{M}}}$ and can obtain its evaluation at $(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau)$. The verifier can also compute $\widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \tau)$ and $\widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}})$ efficiently by itself, which allows it to check that Equation (8) holds, completing the protocol. The full protocol is summarized in Figure 1.

Interactive Oracle Proof Analysis We now analyse the completeness and soundness of the IOP described above. We first note that the IOP is perfectly complete since if the prover is honest all verifier checks pass and the verifier obtains the correct result. The soundness error of the IOP stems from three sources:

1. The soundness error ϵ_1 induced by the polynomial testing on challenge ρ in Equation (4);
2. The soundness error ϵ_2 induced by the execution of the first sum-check protocol on Equation (4);
3. The soundness error ϵ_3 induced by the execution of the second sum-check protocol on Equation (7).

For ϵ_1, the soundness error is at most $\log md/|\mathcal{C}|$ by the DeMillo-Lipton-Schwartz-Zippel lemma (Lemma 1). For ϵ_2, the first sum-check protocol relates the product of two $\log nd$-variate multilinear polynomials. The polynomial in Equation (4) is a product of two $\log nd$-variate multilinear polynomials, which implies that the soundness error induced by the sum-check protocol is $\epsilon_2 \leq 2 \log nd/|\mathcal{C}|$. Finally, the polynomial g_2 in Equation (7) is the product of three multilinear polynomials in $\log d$ variables, so it has maximum individual degree 3, which gives $\epsilon_3 \leq 3 \log d/|\mathcal{C}|$. By the union bound, the total soundness error of the IOP is at most

$$\epsilon_1 + \epsilon_2 + \epsilon_3 \leq \frac{(\log md + 2 \log nd + 3 \log d)}{|\mathcal{C}|}.$$

Batching ciphertext components Recall that we assumed above that the client sends a vector $\mathbf{c}$ containing only one component of the ciphertexts, e.g. $\mathbf{c} = (\mathbf{a}_{1,1}, \mathbf{a}_{2,1} \ldots \mathbf{a}_{n,1})^\top$. This does not capture real world scenarios where a ciphertext is composed of $\kappa + 1$ components with $\kappa \geq 1$. Fortunately, the sum-check protocol is amenable to batching and as a result the IOP described above can easily be extended to handle this setting by batching the $\kappa + 1$ ciphertext components into one, incurring minimal overhead. This is done by taking a random linear combination of the $\kappa+1$ components of the ciphertexts and proving the correctness of the resulting linear combination. To achieve this, the verifier samples an extra random challenge $\gamma \xleftarrow{\$} \mathcal{C}$ and send it to the prover together with ρ. The prover and verifier then run the same IOP as described above but

Parameters: Challenge space $\mathcal{C}$.

Preprocessing: $\bar{\mathbf{M}} \coloneqq \mathsf{NTT}(\mathbf{M}) \in \mathbb{F}_q^{m \times nd}$.

Input: Both $\mathcal{P}$ and $\mathcal{V}$ hold $\mathbf{F}, \mathbf{F}'$ and $\mathbf{c}$. $\mathcal{P}$ further holds $\bar{\mathbf{M}}$ and $\mathcal{V}$ has oracle access to $\widetilde{\widetilde{\mathbf{M}}}$.

Output: $\mathcal{V}$ receives $\mathbf{c}' \in \mathcal{R}_q^m$ and outputs Accept or Reject.

The protocol $\Pi(\mathcal{P}(\mathbf{M}, \mathbf{c}), \mathcal{V}^{\widetilde{\widetilde{\mathbf{M}}}}(\mathbf{c}))$:

1. $\mathcal{P} \to \mathcal{V}$: $\mathcal{P}$ computes $\mathbf{c}' \coloneqq \mathbf{M} \times \mathbf{c}$ and sends $\mathbf{c}'$ to $\mathcal{V}$.
2. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathcal{C}^{\log(md)}$ and sends ρ to $\mathcal{P}$.
3. $\mathcal{P} \leftrightarrow \mathcal{V}$: $\mathcal{P}$ and $\mathcal{V}$ run a sum-check protocol for the claim
$$\widetilde{\mathbf{c}}'(\rho) = \sum_{\beta \in \{0,1\}^{\log(nd)}} g_1(\beta), \tag{9}$$
where the polynomial g_1 with $(\mathbf{k}, \mathbf{l}) \in \{0,1\}^{\log n} \times \{0,1\}^{\log d}$ is defined as
$$g_1(\mathbf{k}, \mathbf{l}) \coloneqq \widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l}) \cdot \widetilde{\mathbf{c}}(\mathbf{k}, \mathbf{l}).$$
The sum-check protocol reduces checking (9) to the evaluation claim $g_1(\sigma) = s$, where $\sigma = (\sigma_{\mathsf{outer}}, \sigma_{\mathsf{inner}}) \in \mathcal{C}^{\log(nd)}$ is a sum-check challenge sampled by $\mathcal{V}$.
4. $\mathcal{P} \to \mathcal{V}$: $\mathcal{P}$ sends to $\mathcal{V}$ the value $\theta \coloneqq \widetilde{\mathbf{N}}(\rho, \sigma)$. $\mathcal{V}$ computes $\widetilde{\mathbf{c}}(\sigma)$ and checks that
$$\theta \cdot \widetilde{\mathbf{c}}(\sigma) \stackrel{?}{=} s.$$
5. $\mathcal{P} \leftrightarrow \mathcal{V}$: Letting $\rho = (\rho_{\mathsf{outer}}, \rho_{\mathsf{inner}})$, $\mathcal{P}$ and $\mathcal{V}$ run a second sum-check protocol for the claim
$$\theta = \sum_{\mathbf{l} \in \{0,1\}^{\log d}} g_2(\mathbf{l}), \tag{10}$$
where the polynomial g_2 is defined as
$$g_2(\mathbf{l}) \coloneqq \widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \mathbf{l}) \cdot \widetilde{\widetilde{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{l}) \cdot \widetilde{\mathbf{F}}(\mathbf{l}, \sigma_{\mathsf{inner}}).$$
The sum-check protocol reduces checking (10) to the evaluation claim $g_2(\tau) = t$, where $\tau \in \mathcal{C}^{\log d}$ is a sum-check challenge sampled by $\mathcal{V}$.
6. $\mathcal{V}$ queries the oracle for $\widetilde{\widetilde{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau)$, computes $\widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \tau)$ and $\widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}})$, and checks that
$$\widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \tau) \cdot \widetilde{\widetilde{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau) \cdot \widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}}) \stackrel{?}{=} t.$$
If all checks pass, $\mathcal{V}$ outputs Accept; otherwise $\mathcal{V}$ outputs Reject.

Fig. 1. Interactive Oracle Proof for the correctness of $\mathbf{M} \times \mathbf{c} = \mathbf{c}'$ over $\mathcal{R}_q$.

with Equation (4) replaced by the following statement:

$$\sum_{\iota=0}^{\kappa} \gamma^{\iota} \cdot \widetilde{\mathbf{c}}'_{\iota}(\rho) = \sum_{\mathbf{k}\in\{0,1\}^{\log n}, \mathbf{l}\in\{0,1\}^{\log d}} \widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l}) \cdot \sum_{\iota=0}^{\kappa} \gamma^{\iota} \cdot \widetilde{\mathbf{c}}_{\iota}(\mathbf{k}, \mathbf{l}). \tag{11}$$

This comes at an extra soundness cost of $\frac{\kappa}{|\mathcal{C}|}$ which remains negligible for sufficiently large $|\mathcal{C}|$. The performance cost of this batching is also marginal as the bulk of the prover time is spent computing values related to $\widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l})$.

Sample extraction Recall from Section 4 that the matrix-vector product $\mathbf{M} \times \mathbf{c}$ where $\mathbf{c}$ encrypts a plaintext vector $\mu \in \mathcal{R}_p^n$ results in a vector of GLWE ciphertexts $\mathbf{c}' \in \mathcal{R}_q^m$ encrypting a plaintext vector $\mu' \in \mathcal{R}_p^m$. The first coefficient of the i-th entry of μ' equates the inner-product of the i-th row of $\mathbf{A}$ with μ, where $\mathbf{A}$ is the original matrix encoded by $\mathbf{M}$. Put differently,

$$\mu'_{i,0} = \langle \mathbf{A}_i, \mu \rangle$$

so the client is only interested in the first coefficient of each entry of μ'. Further, as discussed in Section 3.2, sample extraction allows to extract a LWE ciphertext $(\mathbf{a}', b')$ from the i-th entry of $\mathbf{c}'$ such that this LWE ciphertext encrypts one of the d coefficients the i-th entry of μ'. Performing sample extraction for the first coefficient for each entry of $\mathbf{c}'$ yields exactly the result of the matrix-vector product over $\mathbb{Z}_p$ which the client is interested in. In what follows, we consider sample extraction for the first coefficient of each entry of $\mathbf{c}'$ but the same techniques can be applied to any coefficient.
Performing sample extraction has the advantage of reducing the size of the ciphertext vector sent back by the server from $md(\kappa+1)$ to $m(d\kappa+1)$ $\mathbb{F}_q$-elements, which is a significant improvement in practice ($\approx 2\times$) where usual parameters set $\kappa = 1$.

Conveniently, sample extraction is a very simple operation. This allows us to amend the above IOP to support proving its correct execution without any significant modification to the protocol. To understand how the protocol should be modified, it is helpful to observe that the main difference between a GLWE result and a sample extracted LWE result is that the body component of the former is an $\mathcal{R}_q$-element (represented by a vector $\mathbf{b} \in \mathbb{F}_q^d$) whereas the body component of the latter is a single element in $\mathbb{F}_q$, namely $\mathbf{b}_0$. As such, to implement sample extraction, it suffices to modify the sum-check statement for the body components of the ciphertexts only. This modification simply "ignores" the $d-1$ last coefficients of the GLWE body components and only proves the correctness of the first coefficient, which is exactly the result of the sample extraction. The sum-check statements for the mask components of the ciphertexts remain unchanged as they are not affected by sample extraction. Using well-known sum-check batching techniques, we can batch the sum-check for the mask components and the new sum-check for the body components together to maintain performance at the cost of marginal additional soundness error.

The key insight here is that since we are only interested in the first out of d components of the output body components, we can modify the definition of the block matrix $\mathbf{N}$ to only contain the first row of each block $\mathbf{N}_{i,k}$. Effectively, this means that Equation (4) in the original protocol is replaced by the following sum-check statement:

$$\widetilde{\mathbf{b}}'(\rho_{\mathsf{outer}}, \mathbf{0}) = \sum_{\mathbf{k}\in\{0,1\}^{\log n}, \mathbf{l}\in\{0,1\}^{\log d}} \widetilde{\mathbf{N}}(\rho_{\mathsf{outer}}, \mathbf{0}, \mathbf{k}, \mathbf{l}) \cdot \widetilde{\mathbf{b}}(\mathbf{k}, \mathbf{l}). \tag{12}$$

where $\mathbf{0} \coloneqq (0, \ldots, 0) \in \mathbb{F}_q^{\log d}$ and $\widetilde{\mathbf{b}}$ (resp. $\widetilde{\mathbf{b}}'$) denotes the multilinear extension of the vector of input body components $\mathbf{b}$ (resp. vector of output body components b resulting from sample extraction). In general, to retrieve the i-th coefficient for $i \in \{0, \ldots, d-1\}$ from the output ciphertext, we can simply replace $\mathbf{0}$ by the binary representation of i.

We give a description of this protocol adaptation with sum-check batching in Figure 2 with differences from Figure 1 highlighted in red. Notation is changed from $\mathbf{c}$ representing some GLWE ciphertext vector to $\mathbf{a}$ representing the vector of GLWE mask components and $\mathbf{b}$ representing the vector of GLWE body components, and $\mathbf{a}'$ represents the vector of LWE mask components and $\mathbf{b}'$ represents the vector of LWE body components after sample extraction. We do not concern ourselves with reordering and negating the mask elements as the verifier can efficiently do this task itself. For ease of exposition, we assume $\kappa = 1$ which reflects practical parameter choices. The extra soundness error induced by sum-check batching is at most $\frac{1}{|\mathcal{C}|}$ which remains negligible for sufficiently large $|\mathcal{C}|$. In the case where $\kappa > 1$, the batching technique described in the previous section for the κ mask elements can be applied in conjunction with this sample extraction batching techique, but we omit the details for brevity.

Compiling the IOP. We have described a public-coin IOP for proving the correctness of a matrix-vector product over $\mathcal{R}_q$. To obtain a succinct non-interactive argument (SNARG), it suffices to compile it with the Fiat-Shamir transform [FS87] and combine it with an efficient polynomial commitment scheme. Importantly, this PCS is used as a black-box and may be exchanged later for performance purposes.

In the following section, we give efficient prover and verifier algorithms for the IOP described in Figure 1. These algorithms can be straightforwardly adapted to the batching optimizations discussed above.

Efficient prover and verifier algorithms. We present in Appendix C algorithm-level optimizations for the prover and verifier of the IOP. Broadly, these optimizations rely on the block-circulant structure of the matrix $\mathbf{N}$ and the tensor structure of folding challenges. The key observation is that multiplying the matrix $\mathbf{N}$ with the challenge vector becomes equivalent to performing $\mathcal{R}_q$-arithmetic between the original matrix $\mathbf{M}$ and the ring elements resulting from the inverse NTT of the challenge vector. This trick allows the prover (resp. verifier) to compute the necessary sum-check messages (resp. checks) very efficiently without

Parameters: Challenge space $\mathcal{C}$.

Preprocessing: $\bar{\mathbf{M}} \coloneqq \mathsf{NTT}(\mathbf{M}) \in \mathbb{F}_q^{m \times nd}$.

Input: Both $\mathcal{P}$ and $\mathcal{V}$ hold $\mathbf{F}, \mathbf{F}'$ and $\mathbf{c}$= $(\mathbf{a}, \mathbf{b})$. $\mathcal{P}$ further holds $\bar{\mathbf{M}}$ and $\mathcal{V}$ has oracle access to $\widetilde{\bar{\mathbf{M}}}$.

Output: $\mathcal{V}$ receives $\mathbf{a}' \in \mathcal{R}_q^m, \mathbf{b}' \in \mathbb{F}_q^m$ and outputs Accept or Reject.

The protocol $\Pi(\mathcal{P}(\mathbf{M}, \mathbf{c}), \mathcal{V}^{\widetilde{\bar{\mathbf{M}}}}(\mathbf{c}))$:

1. $\mathcal{P} \to \mathcal{V}$: $\mathcal{P}$ computes $\mathbf{a}' \coloneqq \mathbf{M} \times \mathbf{a}$, $\mathbf{b}^* \coloneqq \mathbf{M} \times \mathbf{b}$, sets $\mathbf{b}'$ to be the vector containing the first coefficient of every entry of $\mathbf{b}^*$ and sends $\mathbf{a}'$ and $\mathbf{b}'$ to $\mathcal{V}$.
2. $\mathcal{V} \to \mathcal{P}$: $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathcal{C}^{\log(md)}, \delta \xleftarrow{\$} \mathcal{C}$ and sends ρ, δ to $\mathcal{P}$.
3. $\mathcal{P} \leftrightarrow \mathcal{V}$: $\mathcal{P}$ and $\mathcal{V}$ run a sum-check protocol for the claim

$$\widetilde{\mathbf{b}}'(\rho_{\mathsf{outer}}, \mathbf{0}) + \delta \cdot \widetilde{\mathbf{a}}'(\rho) = \sum_{\beta \in \{0,1\}^{\log(nd)}} g_{1,b}(\beta) + \delta \cdot g_{1,a}(\beta), \tag{13}$$

   where the polynomials $g_{1,b}$ and $g_{1,a}$ with $(\mathbf{k}, \mathbf{l}) \in \{0,1\}^{\log n} \times \{0,1\}^{\log d}$ are defined as

$$g_{1,a}(\mathbf{k}, \mathbf{l}) \coloneqq \widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l}) \cdot \widetilde{\mathbf{a}}(\mathbf{k}, \mathbf{l})$$
$$g_{1,b}(\mathbf{k}, \mathbf{l}) \coloneqq \widetilde{\mathbf{N}}(\rho_{\mathsf{outer}}, \mathbf{0}, \mathbf{k}, \mathbf{l}) \cdot \widetilde{\mathbf{b}}(\mathbf{k}, \mathbf{l}).$$

   The sum-check protocol reduces checking (13) to the evaluation claim $g_{1,b}(\sigma) + \delta \cdot g_{1,a}(\sigma) = s$, where $\sigma = (\sigma_{\mathsf{outer}}, \sigma_{\mathsf{inner}}) \in \mathcal{C}^{\log(nd)}$ is a sum-check challenge sampled by $\mathcal{V}$.
4. $\mathcal{P} \to \mathcal{V}$: $\mathcal{P}$ sends to $\mathcal{V}$ the values $\theta_a \coloneqq \widetilde{\mathbf{N}}(\rho, \sigma)$ and $\theta_b \coloneqq \widetilde{\mathbf{N}}(\rho_{\mathsf{outer}}, \mathbf{0}, \sigma)$. $\mathcal{V}$ computes $\widetilde{\mathbf{a}}(\sigma), \widetilde{\mathbf{b}}(\sigma)$ and checks that

$$\theta_b \cdot \widetilde{\mathbf{b}}(\sigma) + \delta \cdot \theta_a \cdot \widetilde{\mathbf{a}}(\sigma) \stackrel{?}{=} s.$$

5. $\mathcal{P} \leftrightarrow \mathcal{V}$: Letting $\rho = (\rho_{\mathsf{outer}}, \rho_{\mathsf{inner}})$, $\mathcal{P}$ and $\mathcal{V}$ run a second sum-check protocol for the claim

$$\widetilde{\mathbf{b}}(\sigma) \cdot \theta_b + \delta \cdot \widetilde{\mathbf{a}}(\sigma) \cdot \theta_a = \sum_{\mathbf{l} \in \{0,1\}^{\log d}} \widetilde{\mathbf{b}}(\sigma) \cdot g_{2,b}(\mathbf{l}) + \delta \cdot \widetilde{\mathbf{a}}(\sigma) \cdot g_{2,a}(\mathbf{l}), \tag{14}$$

   where the polynomials $g_{2,b}$ and $g_{2,a}$ are defined as

$$g_{2,b}(\mathbf{l}) \coloneqq \widetilde{\mathbf{F}}'(\mathbf{0}, \mathbf{l}) \cdot \widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{l}) \cdot \widetilde{\mathbf{F}}(\mathbf{l}, \sigma_{\mathsf{inner}})$$
$$g_{2,a}(\mathbf{l}) \coloneqq \widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \mathbf{l}) \cdot \widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{l}) \cdot \widetilde{\mathbf{F}}(\mathbf{l}, \sigma_{\mathsf{inner}}).$$

   The sum-check protocol reduces checking (14) to the evaluation claim $\widetilde{\mathbf{b}}(\sigma) \cdot g_{2,b}(\tau) + \delta \cdot \widetilde{\mathbf{a}}(\sigma) \cdot g_{2,a}(\tau) = t$, where $\tau \in \mathcal{C}^{\log d}$ is a sum-check challenge sampled by $\mathcal{V}$.
6. $\mathcal{V}$ queries the oracle for $\widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau)$, computes $\widetilde{\mathbf{F}}'(\mathbf{0}, \tau), \widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \tau)$, and $\widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}})$, and checks that

$$\widetilde{\mathbf{b}}(\sigma) \cdot \widetilde{\mathbf{F}}'(\mathbf{0}, \tau) \cdot \widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau) \cdot \widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}}) +$$
$$\delta \cdot \widetilde{\mathbf{a}}(\sigma) \cdot \widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \tau) \cdot \widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau) \cdot \widetilde{\mathbf{F}}(\tau, \sigma_{\mathsf{inner}}) \stackrel{?}{=} t.$$

   If all checks pass, $\mathcal{V}$ outputs Accept; otherwise $\mathcal{V}$ outputs Reject.

Fig. 2. Interactive Oracle Proof for the correctness of $\mathbf{M} \times \mathbf{a} = \mathbf{a}'$ and $(\mathbf{M} \times \mathbf{b}) = \mathbf{b}'$ over $\mathcal{R}_q$, where $\mathbf{b}'$ is the body component resulting from sample extraction.

explicitly materializing the matrix $\mathbf{N}$. More precisely, if the matrix $\mathbf{N}$ is of size $md \times nd$, the prover can compute its messages for the first sum-check in time $\mathcal{O}(mnd + nd \log d)$, effectively saving a factor d compared to the straightforward approach. For the second sum-check, we show that the prover can compute its messages in time $\mathcal{O}(nd)$ instead of $\mathcal{O}(mnd + d^2)$, incurring significant savings in practice. The details of these optimizations, as well as supporting algebraic claims, are provided in Appendix C.

6 Verifiable PIR

As previously discussed, we can enhance the passively secure PIR protocol described in Section 4 with the verifiable computation protocol introduced in Section 5 to obtain a verifiable PIR protocol secure against fully malicious adversaries. In what follows we provide definitions capturing our vPIR scheme and its security properties, present our construction and prove its security.

Given that the field of actively secure PIR still nascent, existing definitions do not quite fit our scheme. As a consequence, we provide a new definition of verifiable PIR with preprocessing which captures the security properties of our scheme.

Definition 5 (Verifiable PIR with preprocessing). *Let $M, N \in \mathbb{N}$. Public parameters* pp *are implicitly given as input to all algorithms following the setup phase.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*: outputs public parameters* pp*.*
- $d \leftarrow \mathsf{Digest}(\mathbf{D})$*: takes a database $\mathbf{D} \in \mathbb{Z}_M^N$ and outputs a short digest d.*
- $\mathsf{Accept/Reject} \leftarrow \mathsf{DigVer}(d)$*: takes a digest d and accepts or rejects.*
- $(q, \mathsf{st}) \leftarrow \mathsf{Query}(i \in [N])$*: takes in an index $i \in [N]$ and outputs a query-state pair (q, st).*
- $(a, \pi) \leftarrow \mathsf{Answer}(\mathbf{D}, q)$*: takes in a database $\mathbf{D}$ and a query q and outputs an answer-proof pair (a, π).*
- $r/\perp \leftarrow \mathsf{Recover}(\mathsf{st}, d, q, a, \pi)$*: takes in a query state* st*, a database digest d, a query q, an answer a, and a proof π and outputs a database element $r \in \mathbb{Z}_M$ or the null symbol $\perp$.*

The algorithm DigVer is run once by the client upon obtaining the digest and before issuing any query. Its purpose is to guarantee that the digest is well-formed, i.e. that it commits to a database in $\mathbb{Z}_M^N$ rather than to an arbitrary object accepted by the underlying commitment scheme. Recall from Section 4 that correct decryption of the server's answer holds provided the database entries are "small", namely in $\mathbb{Z}_p$. A digest committing to a matrix with entries outside of this range would still admit accepting answers, but the client's decryption of those answers may then depend on its encryption randomness for some indices and not for others. This yields a selective failure attack which DigVer prevents by rejecting such digests. Notice that DigVer takes only public inputs. As such, anyone may verify a digest and a client never sends a query to a server whose digest was rejected. Our vPIR protocol that implements this definition is described in Figure 3.

The vPIR scheme Π is composed of the following algorithms, where $\mathcal{P}$ (server) holds a public database $\mathbf{A} \in \mathbb{Z}_p^{m \times nd}$ and $\mathcal{V}$ (client) holds an index $i \in [nd]$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: outputs public parameters pp consisting of GLWE parameters $(q, d, \kappa, p, \chi_s, \chi_e)$ with $M = p^m$ and $N = nd$, a challenge space $\mathcal{C}$ of sufficient size, and a polynomial commitment scheme PCS for multilinear polynomials over $\mathbb{F}_q$ with evaluation domain $\mathcal{C}$.
- $d \leftarrow \mathsf{Digest}(\mathbf{A})$: computes $\mathbf{M} \coloneqq \tau(\mathbf{A}) \in \mathcal{R}_q^{m \times n}$ and $\bar{\mathbf{M}} \coloneqq \mathsf{NTT}(\mathbf{M}) \in \mathbb{F}_q^{m \times nd}$. Computes $c_{\mathbf{A}} \coloneqq \mathsf{PCS.Commit}(\widetilde{\mathbf{A}})$, $c_{\bar{\mathbf{M}}} \coloneqq \mathsf{PCS.Commit}(\widetilde{\bar{\mathbf{M}}})$ and a proof π_d that the digest is well-formed with respect to $(c_{\mathbf{A}}, c_{\bar{\mathbf{M}}})$ (cf. Section 6). Outputs the digest $d \coloneqq (c_{\mathbf{A}}, c_{\bar{\mathbf{M}}}, \pi_d)$.
- $\mathsf{Accept}/\mathsf{Reject} \leftarrow \mathsf{DigVer}(d)$: parses $d = (c_{\mathbf{A}}, c_{\bar{\mathbf{M}}}, \pi_d)$ and outputs the result of verifying π_d with respect to $(c_{\mathbf{A}}, c_{\bar{\mathbf{M}}})$.
- $(q, \mathsf{st}) \leftarrow \mathsf{Query}(i)$: samples $\mathbf{s} \xleftarrow{\$} \chi_s^\kappa$ and encrypts the i-th unit vector $\mathbf{e}_i \in \{0,1\}^{nd}$ under $\mathbf{s}$, obtaining a ciphertext vector $\mathbf{c} \in \mathcal{R}_q^{n \times (\kappa+1)}$. Outputs $(q \coloneqq \mathbf{c}, \mathsf{st} \coloneqq \mathbf{s})$.
- $(a, \pi) \leftarrow \mathsf{Answer}(\mathbf{A}, q)$: Parses $\mathbf{M}$ and $\bar{\mathbf{M}}$ from Digest and parses $q = \mathbf{c}$. Computes $\mathbf{c}'_\iota \coloneqq \mathbf{M} \times \mathbf{c}_\iota$ for each $\iota \in [\kappa + 1]$ and sets a to be the corresponding sample extracted LWE ciphertexts. Invokes the compiled IOP from Figure 2 together with PCS to generate a proof π attesting to the integrity of a with respect to the digest d and the query q. Outputs (a, π).
- $r/\perp \leftarrow \mathsf{Recover}(\mathsf{st}, d, q, a, \pi)$: runs the verification procedure of the compiled IOP on input (d, q, a, π). If verification fails, outputs $\perp$. Otherwise, decrypts a using $\mathsf{st} = \mathbf{s}$ and outputs the resulting value $r \in \mathbb{Z}_p^m$.

Fig. 3. Verifiable PIR scheme combining the passively secure PIR from Section 4 with the IOP from Section 5.2, compiled using a polynomial commitment scheme.

Digest well-formedness. The digest contains commitments to two polynomials: the multilinear extension $\widetilde{\mathbf{A}}$ of the database $\mathbf{A} \in \mathbb{Z}_p^{m \times nd}$ and the multilinear extension $\widetilde{\bar{\mathbf{M}}}$ of its encoded NTT representation, which is the one used by the IOP of Section 5. The proof π_d attests to the following two statements:

1. $\widetilde{\mathbf{A}}(\mathbf{x}) \in \mathbb{Z}_p$ for all $\mathbf{x} \in \{0,1\}^{\log mnd}$;
2. $\widetilde{\bar{\mathbf{M}}}(\mathbf{w}, \mathbf{y}, \mathbf{z}) = \sum_{\mathbf{l} \in \{0,1\}^{\log d}} \widetilde{\mathbf{G}}(\mathbf{z}, \mathbf{l}) \cdot \widetilde{\mathbf{A}}(\mathbf{w}, \mathbf{y}, \mathbf{l})$ for all $(\mathbf{w}, \mathbf{y}, \mathbf{z}) \in \{0,1\}^{\log m} \times \{0,1\}^{\log n} \times \{0,1\}^{\log d}$, where $\mathbf{G} \coloneqq d \cdot \mathbf{F}'^{\top}$.

The second statement encodes $\bar{\mathbf{M}}_{i,k} = \mathsf{NTT}(\tau(\mathbf{A}_{i,k}))$. The first statement is a range check. A sum-check for $0 = \sum_{\mathbf{x}} \mathsf{eq}(\mathbf{r}, \mathbf{x}) \cdot \prod_{v=0}^{p-1} (\widetilde{\mathbf{A}}(\mathbf{x}) - v)$ would have individual degree $p + 1$ and a prover running in time $\mathcal{O}(p \cdot mnd)$, which is prohibitive for large plaintext moduli. Instead, we use a lookup argument based on logarithmic derivatives [Hab22]. Let $\widetilde{\mu}$ be the multilinear extension of the vector $\mu \in \mathbb{F}_q^p$ of multiplicities, where μ_v counts the entries of $\mathbf{A}$ equal to v, to which the prover commits as part of π_d. The verifier samples $\beta \xleftarrow{\$} \mathcal{C}$ and both parties prove that $\sum_{\mathbf{x} \in \{0,1\}^{\log mnd}} (\beta - \widetilde{\mathbf{A}}(\mathbf{x}))^{-1} = \sum_{\mathbf{v} \in \{0,1\}^{\log p}} \widetilde{\mu}(\mathbf{v}) \cdot (\beta - \mathbf{v})^{-1}$,

which holds if and only if all entries of $\mathbf{A}$ lie in $\mathbb{Z}_p$, provided $|\mathcal{C}| \gg mnd + p$. With the layered sum-check of [PH23], no auxiliary polynomial of size mnd is committed, the prover runs in time $\mathcal{O}(mnd + p)$ and the protocol ends with one query to each of $\widetilde{\mathbf{A}}$ and $\widetilde{\mu}$. For the second statement, the verifier samples $(\rho_1, \rho_2, \rho_3) \xleftarrow{\$} \mathcal{C}^{\log mnd}$, queries $\widetilde{\widetilde{\mathbf{M}}}(\rho_1, \rho_2, \rho_3)$ and both parties run a sum-check for $\widetilde{\widetilde{\mathbf{M}}}(\rho_1, \rho_2, \rho_3) = \sum_{\mathbf{l} \in \{0,1\}^{\log d}} \widetilde{\mathbf{G}}(\rho_3, \mathbf{l}) \cdot \widetilde{\mathbf{A}}(\rho_1, \rho_2, \mathbf{l})$, which ends with a query to $\widetilde{\mathbf{A}}$. The verifier evaluates $\widetilde{\mathbf{G}}$ by itself as it does for $\widetilde{\mathbf{F}'}$ in Section 5. The soundness error $\epsilon_{\mathsf{Digest}}$ is dominated by the $(mnd + p)/|\mathcal{C}|$ term of the logarithmic derivative identity and is negligible for the challenge spaces we consider. The proof is compiled with PCS and the Fiat-Shamir transform as in Section 5. This work is performed once per database, independently of the number of queries, and the resulting proof is publicly verifiable in time polylogarithmic in mnd.

We can now define the desired correctness and security definitions for verifiable PIR schemes.

Definition 6 (Correctness). *A vPIR scheme is said to have* negligible correctness error *if, for any security parameter λ, for all databases $\mathbf{D} \in \mathbb{Z}_M^N$ and for all indices $i \in [N]$,*

$$\Pr\left[\begin{array}{c} \mathsf{DigVer}(d) = \mathsf{Accept} \wedge \\ \mathsf{Recover}(\mathsf{st}, d, q, a, \pi) = \mathbf{D}[i] \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ d \leftarrow \mathsf{Digest}(\mathbf{D}) \\ (q, \mathsf{st}) \leftarrow \mathsf{Query}(i) \\ (a, \pi) \leftarrow \mathsf{Answer}(\mathbf{D}, q) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Lemma 2. *Let* PCS *be a polynomial commitment scheme satisfying perfect completeness, let* IOP *be a perfectly complete interactive oracle proof system (both for the statement of Section 5 and for digest well-formedness), and let $\epsilon_{\mathsf{GLWE}} \in \mathsf{negl}(\lambda)$ be the correctness error of the underlying GLWE encryption scheme for the parameters* pp*. The vPIR scheme Π described in Figure 3 compiled with* PCS *has negligible correctness error.*

Proof. We defer the proof to Appendix B.1.

Definition 7 (Passive privacy). *We say that a vPIR scheme is* passively private *if for any security parameter λ and honestly generated public parameters* pp*, for all databases $\mathbf{D} \in \mathbb{Z}_M^N$ of size* poly(λ)*, for all indices $i, j \in [N]$ and for every* PPT *$\mathcal{A}$, we have that*

$$\big|\Pr[\mathcal{A}(\mathsf{pp}, q) = 1 \mid (q, \mathsf{st}) \leftarrow \mathsf{Query}(i)] - \Pr[\mathcal{A}(\mathsf{pp}, q) = 1 \mid (q, \mathsf{st}) \leftarrow \mathsf{Query}(j)]\big| \leq \mathsf{negl}(\lambda).$$

Lemma 3. *Under the GLWE assumption, the vPIR scheme Π described in Figure 3 is passively private.*

Proof. Since an honestly generated query is a set of GLWE encryptions, the passive privacy of Π follows immediately from the IND-CPA-security of the GLWE encryption scheme.

We introduce the notion of integrity which combines the soundness of the IOP and the knowledge soundness of the PCS.

Definition 8 (Integrity). *We say that a vPIR scheme satisfies* integrity *if for all security parameter λ, for all $i \in [N]$ and for all interactive deterministic polynomial time $\mathcal{P}^*$, there exists an extractor $\mathcal{E}^{\mathcal{P}^*}$ such that for all PPT adversaries $\mathcal{A}$,*

$$\Pr\left[\begin{array}{c} \mathsf{DigVer}(d) = \mathsf{Accept} \wedge \\ \mathsf{Recover}(\mathsf{st}, d, q, a, \pi) \neq \bot \wedge \\ \mathsf{Recover}(\mathsf{st}, d, q, a, \pi) \neq \mathbf{D}[i] \end{array} \middle| \begin{array}{c} (d, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (q, \mathsf{st}) \leftarrow \mathsf{Query}(i) \\ ((a, \pi), \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{st}_{\mathcal{A}}, q) \\ \mathbf{D} \leftarrow \mathcal{E}^{\mathcal{A}, \mathcal{P}^*(\mathsf{pp}, \mathsf{st}_{\mathcal{A}}, q)}(\mathsf{pp}, d, q, \pi) \end{array} \right] \leq \mathsf{negl}(\lambda).$$

Here, the extractor $\mathcal{E}^{\mathcal{A}, \mathcal{P}^*}$ has full access to the states of $\mathcal{A}$ and $\mathcal{P}^*$ and can rewind them to any point in the interaction.

Lemma 4. *Let Π be the vPIR scheme presented in Figure 3 composed of the GLWE encryption scheme from Definition 1 instantiated with parameters ensuring correctness error $\epsilon_{\mathsf{GLWE}} \in \mathsf{negl}(\lambda)$, a polynomial commitment scheme PCS satisfying completeness, evaluation binding and having knowledge soundness error $\epsilon_{\mathsf{PCS}} \in \mathsf{negl}(\lambda)$ with evaluation domain $\mathcal{C}$ such that $1/|\mathcal{C}| \in \mathsf{negl}(\lambda)$, and an interactive oracle proof IOP using challenge space $\mathcal{C}$ satisfying completeness and having soundness error $\epsilon_{\mathsf{IOP}} \in \mathsf{negl}(\lambda)$, and let the digest well-formedness proof have soundness error $\epsilon_{\mathsf{Digest}} \in \mathsf{negl}(\lambda)$. Then Π satisfies integrity.*

Proof. We defer the proof to Appendix B.2.

Definition 9 (Active privacy). *We say that a vPIR scheme is* actively private *if for any security parameter λ and honestly generated public parameters pp, for all databases in $\mathbb{Z}_M^N$ of size $\mathsf{poly}(\lambda)$, for every PPT $\mathcal{A}$, the scheme satisfies* integrity *as well as the following condition. This definition is analoguous to that of [dCL24].*

Let $(d, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp})$ be the digest and adversary state generated by $\mathcal{A}$ after receiving the public parameters pp and let $\mathbf{D}$ be the unique database extractable by $\mathcal{E}^{\mathcal{P}^*}$ given an accepting opening generated by $\mathcal{A}$. In both experiments below, the client runs $\mathsf{DigVer}(d)$ upon receiving the digest and halts if it rejects, in which case the experiment outputs $\mathcal{A}(\mathsf{st}_{\mathcal{A}})$. Now, for any index $i \in [N]$, define the distribution $\mathsf{REAL}_{\mathcal{A},i}$ as follows.

$$\mathsf{REAL}_{\mathcal{A},i} \coloneqq \left\{ \begin{array}{c} (d, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (q, \mathsf{st}) \leftarrow \mathsf{Query}(i) \\ ((a, \pi), \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, q) \\ b \leftarrow [\mathsf{Recover}(\mathsf{st}, d, q, a, \pi) = \mathbf{D}[i]] \\ \mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, b) \\ \mathcal{A}(\mathsf{st}_{\mathcal{A}}) \end{array} \right\}$$

Similarly, for a PPT *simulator* $\mathcal{S}$*, define the distribution* $\mathsf{IDEAL}_{\mathcal{A},\mathcal{S}}$ *as follows.*

$$\mathsf{IDEAL}_{\mathcal{A},\mathcal{S}} \coloneqq \left\{ \begin{array}{c} (d, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (q, \mathsf{st}) \leftarrow \mathcal{S}(\mathsf{pp}) \\ ((a, \pi), \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, q) \\ b \leftarrow \mathcal{S}(\mathsf{pp}, d, q, a, \pi) \\ \mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}(\mathsf{st}_{\mathcal{A}}, b) \\ \mathcal{A}(\mathsf{st}_{\mathcal{A}}) \end{array} \right\}$$

It must hold that

$$|\Pr[\mathsf{REAL}_{\mathcal{A},i} = 1] - \Pr[\mathsf{IDEAL}_{\mathcal{A},\mathcal{S}} = 1]| \leq \mathsf{negl}(\lambda).$$

Lemma 5. *The protocol described in Figure 3 satisfying the same assumptions as in Lemma 4 is actively private.*

Proof. We defer the proof to Appendix B.3.

7 Experimental results

We provide an open-source[1] Rust implementation of the vPIR protocol described in the previous section and evaluate its performance on various database sizes. We compare the performance of our vPIR protocol to prior work on actively secure PIR, and we also evaluate the performance of the underlying IOP protocol for verifying matrix-vector products over rings, which is the main technical contribution of this work and may be of independent interest.

7.1 Implementation description

We implement the protocol described in Figure 3 with the optimization described in Figure 2. We use WHIR [ACFY25] as the PCS to compile the IOP and we use Fiat-Shamir [FS87] with BLAKE3 [OANWO21] to render the protocol non-interactive. Looking ahead, the PCS is by far the most expensive component of the compiled protocol. It is worth noting that it is used in a black-box manner, so any improvement in the efficiency of the PCS would directly translate to an improvement in the efficiency of our vPIR protocol. We note that the batching trick described in Figure 2 may be used to batch n PIR queries at the cost of increasing the soundness error by a factor of n. Since the server work is dominated by the PCS evaluation proof, the overall cost would be only marginally increased.

We use the GLWE parameters from the first row of Table 1, which allow us to claim at least 100 bits of security according to the Lattice Estimator [APS15]. Further, these parameters can be shown to provide sufficient correctness for the PIR scheme, depending on the dimensions of the database.

The parameters define the field $\mathbb{F}_q$ and the ring $\mathcal{R}_q \coloneqq \mathbb{F}_q[X]/(X^d+1)$ used in the IOP. To obtain 128 bits of security in the IOP, we use the quadratic extension

[1] `https://anonymous.4open.science/r/vpir-566B/`

Table 1. GLWE parameters. The first row is used throughout Section 7.2, the second row for the comparison with VeriSimplePIR in Section 7.3.

Setting	q	d	κ	σ	p
Standalone evaluation	$2^{64} - 2^{32} + 1$	1024	1	2^{33}	16
VeriSimplePIR comparison	$2^{64} - 2^{32} + 1$	2048	1	6.4	2^{24}

of $\mathbb{F}_q$ as the challenge space and evaluation domain of the PCS. Lastly, we configure WHIR with a parameter regime targeting 100 bits of security without relying on conjectured decoding regimes. We note that the WHIR commitment and proof size are marginal in the total communication cost of the vPIR protocol.

Our implementation does not currently include the digest well-formedness proof of Section 6. Its cost of $\mathcal{O}(mnd + p)$ prover work is of the same order as that of the first sum-check of the online IOP, it is paid once per database and it is amortized across all queries from all clients. Clients only verify the resulting proof, which takes time polylogarithmic in mnd.

7.2 Experiments

As discussed in Section 4, the vPIR protocol consists essentially of a verifiable matrix-vector multiplication protocol over $\mathcal{R}_q$. As such, we evaluate the performance of our vPIR protocol by evaluating the performance of the underlying verifiable matrix-vector multiplication protocol over $\mathcal{R}_q$.

For each experiment, we run the task at least 10 times and take the median value as the representative. To allow for fair comparison with prior work, we use an AWS EC2 instance of type `Hpc7a.24xlarge` equipped with 48 CPU cores and 768 GB of memory. In what follows, we consider a database $\mathbf{D} \in \mathbb{Z}_p^{m \times nd}$ having nd entries in $\mathbb{Z}_p^m$.

Client time. To model a computationally bounded client, we follow prior art [TWZ26] and restrict the client to only using one CPU core, while allowing the server to use all available cores. Figure 4 provides different client time values as a function of m for different values of n. Further timings are deferred to Figure 7. The client time is taken as the sum of the time taken to encrypt the query, to verify the proof of correctness of the answer, and to decrypt the answer. Since the client is essentially outsourcing a matrix-vector multiplication, we also provide the time taken to perform a single-threaded plaintext $\mathcal{O}(mnd)$ matrix-multiplication protocol over $\mathbb{Z}_p$ for reference. This measure indicates how much time a client might save by outsourcing the computation to the server, compared to performing the matrix-vector multiplication locally in plaintext.

Server time. Table 2 provides different server time values as a function of m and n, both including and excluding the time taken to generate the evaluation proof using the WHIR PCS. Further timings are deferred to Table 7. We observe that the server time is largely dominated by the time taken to generate the evaluation

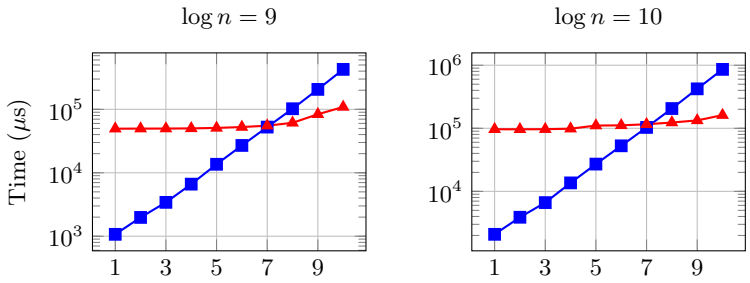

Fig. 4. Plaintext matrix–vector product (blue) vs. verification pipeline (red), log–log scale, for various n values.

proof. It is worth noting that the server time can be significantly improved by using a more efficient PCS, and this would directly translate to an improvement in the efficiency of our vPIR protocol.

Table 2. Prover computation time (in seconds) as a function of n and m. Each cell shows with PCS (top) and without PCS (bottom).

$\log n$ \ $\log m$	1	2	3	4	5	6	7	8	9	10
9	0.40	0.74	1.16	2.26	3.70	7.90	13.51	29.62	52.53	117.42
	0.11	0.11	0.12	0.13	0.13	0.20	0.32	0.93	1.70	3.23
10	0.87	1.26	2.41	3.83	8.00	13.65	29.77	52.98	117.68	210.86
	0.22	0.22	0.23	0.24	0.27	0.42	1.02	2.02	3.75	7.17

Communication. A common technique for reducing the online size of GLWE ciphertexts is to use a 128-bit seed to generate the mask component of the ciphertext. As such, the dominating factor in the client upload size is the body component of the GLWE ciphertexts, which is of size $nd \cdot \log q$ bits. Table 5 provides the size of the client upload for different database widths, using the GLWE parameters from the first row of Table 1. On the other hand, Table 6 provides the size of the client download for different database heights, using the same GLWE parameters. Here, the dominating factor is the mask component of the GLWE ciphertexts, since the body component is sample extracted to give a single field element. This mask component is of size $m \cdot d\kappa \cdot \log q$

7.3 Comparison with prior work

Verifiable matrix-vector multiplication over $\mathcal{R}_q$. The work that most closely resembles ours is that of [TWZ26], which also provides a verifiable matrix-

vector multiplication protocol over rings in a similar parameter regime. Experiments reported in Table 3 show that our protocol provides an improvement of slightly more than an order of magnitude in IOP time compared to the protocol of [TWZ26], while providing similar correctness guarantees and level of security. Since PCS dominates the end-to-end cost, our protocol essentially provides a factor of two improvement in overall performance. The client in our protocol exhibits slightly better performance but remains mostly on par. Finally, our integration of the sample extraction technique in Section 5.2 reduces the size of the answer ciphertext by a factor 2 compared to the naive approach of [TWZ26], which does not leverage sample extraction.

Table 3. Comparison of prover (server) time (in seconds) using measures provided by [TWZ26]. Timings exclude the generation of the WHIR proofs for both works.

$\log n = \log m$	6	8	10
This work	0.04	0.28	7.17
[TWZ26]	0.61	8.84	184.21

VeriSimplePIR. The construction of [dCL24] is also similar in spirit to ours in that it provides a verifiable PIR protocol in the single-server setting. However, their proposal considers PIR with preprocessing, i.e. the offline phase of their protocol provides the client with useful information that allows for a more efficient online phase.

Our solution necessarily requires a preprocessing phase comprising at least the commitment to the public database. However, the preprocessing phase presented above can be augmented without changing the underlying IOP or cryptographic assumptions in order to save performance in the online phase. Importantly, this phase can be run once for all users, and amortized over all queries to the same database. During preprocessing, a mask vector $\mathbf{a}$ is determined by the client (cf. Section 5) which will be used to encrypt online queries[2]. The server non-interactively executes the compiled IOP to generate a publicly verifiable proof that the matrix-vector multiplication between the database and the mask components is correct, and publishes the resulting $\mathbf{a}'$ for later use by clients, who may now download, verify, and save $\mathbf{a}'$. During the online phase, the client first encrypts its query using $\mathbf{a}$. The client then sends the resulting ciphertext body components to the server, which computes the sample extracted answer as well as a proof of its well-formedness, and sends it back to the client, which in turn decrypts it using the formerly saved $\mathbf{a}'$ to obtain the final result. Clearly, this adaptation significantly reduces the online communication costs, while augmenting the client storage costs as well as the offline phase cost.

[2] A small PRNG seed can be determined here instead of the entire expanded masks.

We compare our solution in this preprocessing setting to that of [dCL24] (which is presented in the preprocessing setting only) for databases of 4 GiB, using their open-source implementation[3]. Table 4 provides the comparison results in terms of communication and client storage. Our protocol's offline communication and storage is dominated by the client's download of $\mathbf{a}'$, whereas the online communication is dominated by the client's upload of the body component of the query ciphertexts ($\mathbf{b}$), as noted in the communication analysis of the prior subsection. Our protocol exhibits improvements of an order of magnitude in the offline phase and client storage, and a factor of 1.5 in the online phase.

In terms of latency however, our protocol is slower than VeriSimplePIR. Indeed, while their online server time is on the order of a second, our online server time is of about 7 minutes. Verifier time is roughly on par, in the order of hundreds of milliseconds. We recall that Table 7 shows that the online latency is dominated by the time taken to generate an evaluation proof using the WHIR PCS. Given that our ptocol is PCS-agnostic, replacing WHIR by a more performant PCS (e.g., [NS24,NOZ26]) would significantly reduce the online latency of our protocol, while maintaining the same communication costs.

Despite the increased latency, our protocol has the practical advantage of being secure even against repeatedly-malicious adversaries, while the security of VeriSimplePIR is only guaranteed until the adversary exhibits dishonest behaviour. When malicious behaviour is detected, their offline phase has to be re-run to restore security, which may not be practical in several settings due to the additional communication cost of north of 75% of the database size.

Table 4. Comparison of our preprocessing vPIR protocol with VeriSimplePIR [dCL24] on a 4 GiB database. We use similar cryptographic parameters as VeriSimplePIR (second row of Table 1) and database parameters $m = 2^{13}, nd = 2^{17.33}$ for negligible correctness error.

	Offline communication	Client storage	Online communication
VeriSimplePIR [dCL24]	3.29 GiB	3.29 GiB	3.22 MiB
This work (preprocessing)	$\approx$ 128 MiB	$\approx$ 128 MiB	$\approx$ 2 MiB

8 Conclusion

We have presented a new PIOP for linear relation over rings, which can be instantiated into a verifiable plaintext matrix - encrypted vector multiplication protocol. We then use this protocol to build a verifiable PIR protocol secure against active adversaries. Our protocol is concretely efficient, offers strong security guarantees, and is plausibly post-quantum secure. Further, it is PCS agnostic, thereby exposing it to further improvements as more performant PCS are developed.

[3] `https://github.com/leodec/VeriSimplePIR`

References

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, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 214–243. Springer, Cham, May 2025. doi:10.1007/978-3-031-91134-7_8.

App24. Apple. Combining machine learning and homomorphic encryption in the Apple ecosystem. Apple Research Highlight, 2024. URL: https://machinelearning.apple.com/research/homomorphic-encryption.

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. URL: http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml.

AR26. Pranav Shriram Arunachalaramanan and Ling Ren. Single-Server Stateful PIR with Verifiability and Balanced Efficiency . In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1385–1402, Los Alamitos, CA, USA, May 2026. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/SP63933.2026.00019, doi:10.1109/SP63933.2026.00019.

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995. doi:10.1109/SFCS.1995.492461.

CNCG+23. Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. Authenticated private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3835–3851, Anaheim, CA, August 2023. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/colombo.

dCL24. Leo de Castro and Keewoo Lee. Verisimplepir: Verifiability in simplepir at no online cost for honest servers. In *Proceedings of the 33rd USENIX Conference on Security Symposium*, SEC '24, USA, 2024. USENIX Association.

DL78. Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978. doi:10.1016/0020-0190(78)90067-4.

DPC23. Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. *Proc. Priv. Enhancing Technol.*, 2023(1):365–383, 2023. URL: https://doi.org/10.56553/popets-2023-0022, doi:10.56553/POPETS-2023-0022.

DT24. Marian Dietz and Stefano Tessaro. Fully malicious authenticated PIR. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 113–147. Springer, Cham, August 2024. doi:10.1007/978-3-031-68400-5_4.

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.

FMS25. Brett Falk, Pratyush Mishra, and Matan Shtepel. Malicious security for PIR (almost) for free. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 175–210. Springer, Cham, August 2025. doi:10.1007/978-3-032-01913-4_6.

FS87. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

Hab22. Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022. URL: https://eprint.iacr.org/2022/1530.

HDCGZ23. Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 396–416, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3600006.3613134.

HMZ25. Mi-Ying Miryam Huang, Xinyu Mao, and Jiapeng Zhang. Sublinear proofs over polynomial rings. Cryptology ePrint Archive, Paper 2025/199, 2025. URL: https://eprint.iacr.org/2025/199.

KO97. Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, October 19-22, 1997*, pages 364–373. IEEE Computer Society, 1997. doi:10.1109/SFCS.1997.646125.

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

LLX+26. Fengrun Liu, Haofei Liang, Xiang Xie, Yu Yu, Wenting Zheng, and Yuncong Hu. HasteBoots: Proving TFHE programmable bootstrapping in seconds. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 41–60, Baltimore, MD, August 2026. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity26/presentation/liu-fengrun.

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013. doi:10.1145/2535925.

LXZ21. Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2968–2985, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3460120.3485379.

NOZ26. Ngoc Khanh Nguyen, George O'Rourke, and Jiapeng Zhang. Hachi: Efficient lattice-based multilinear polynomial commitments over extension fields. Cryptology ePrint Archive, Paper 2026/156, 2026. URL: https://eprint.iacr.org/2026/156.

NS24. Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast polynomial commitments from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 243–275. Springer, Cham, August 2024. doi:10.1007/978-3-031-68403-6_8.

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

PH23. Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Report 2023/1284, 2023. URL: https://eprint.iacr.org/2023/1284.

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.

Sch80. J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. doi:10.1145/322217.322225.

Tha13. Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 71–89. Springer, Berlin, Heidelberg, August 2013. doi:10.1007/978-3-642-40084-1_5.

TWZ26. Louis Tremblay Thibault, Michael Walter, and Jiapeng Zhang. Practical SNARGs for matrix multiplications over encrypted data. Cryptology ePrint Archive, Paper 2026/027, 2026. URL: https://eprint.iacr.org/2026/027.

WLZ+23. Yinghao Wang, Xuanming Liu, Jiawen Zhang, Jian Liu, and Xiaohu Yang. Crust: Verifiable and efficient private information retrieval with sublinear online time. Cryptology ePrint Archive, Paper 2023/1607, 2023. URL: https://eprint.iacr.org/2023/1607.

XZZ+19. Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Cham, August 2019. doi:10.1007/978-3-030-26954-8_24.

Zip79. Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

ZWL+25. 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.

A Deferred Preliminaries

A.1 Sum-check protocol

Definition 10 (Sum-check protocol [LFKN92]). *Let $\mathbb{F}$ be a field, ν a positive integer and $p : \mathbb{F}^\nu \to \mathbb{F}$ a ν-variate polynomial of maximum individual degree* deg. *The sum-check protocol is an interactive proof between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ to prove the following claim:*

$$S \coloneqq \sum_{\mathbf{b}\in\{0,1\}^\nu} p(\mathbf{b}).$$

The protocol reduces the veracity of this statement to the evaluation of p at a random point $\mathbf{r} \xleftarrow{\$} \mathcal{C}^\nu$. It has soundness error $\nu\mathsf{deg}/|\mathcal{C}|$ where $\mathcal{C}$ is the challenge space from which $\mathbf{r}$ is sampled. The protocol consists of ν rounds and proceeds as follows.

1. *In the first round, the prover sends a univariate polynomial*

$$p_1(X_1) \coloneqq \sum_{b_2,\ldots,b_\nu \in \{0,1\}} p(X_1, b_2, \ldots, b_\nu)$$

   to the verifier, who checks that $S = p_1(0) + p_1(1)$. The verifier samples a random challenge $r_1 \xleftarrow{\$} \mathcal{C}$ and sends it to the prover.
2. *In the i-th round, for $i \in \{2, \ldots, \nu - 1\}$, the prover sends a univariate polynomial*

$$p_i(X_i) \coloneqq \sum_{b_{i+1},\ldots,b_\nu \in \{0,1\}} p(r_1, \ldots, r_{i-1}, X_i, b_{i+1}, \ldots, b_\nu)$$

   to the verifier, who checks that $p_{i-1}(r_{i-1}) = p_i(0) + p_i(1)$. The verifier samples a random challenge $r_i \xleftarrow{\$} \mathcal{C}$ and sends it to the prover.
3. *In the last round, the prover sends a univariate polynomial*

$$p_\nu(X_\nu) \coloneqq p(r_1, \ldots, r_{\nu-1}, X_\nu)$$

   to the verifier, who checks that $p_{\nu-1}(r_{\nu-1}) = p_\nu(0) + p_\nu(1)$. The verifier now samples a final challenge $r_\nu \xleftarrow{\$} \mathcal{C}$ and accepts if and only if $p_\nu(r_\nu) = p(r_1, \ldots, r_\nu)$. The means by which the verifier obtains the value $p(r_1, \ldots, r_\nu)$ is left unspecified in the above description of the protocol, as it depends on the context in which the sum-check protocol is invoked.

B Deferred Proofs

B.1 Proof of Lemma 2

Proof. Recall that $\mathsf{Query}(i) \mapsto (q, \mathsf{st})$ outputs a valid GLWE encryption of the i-th unit vector $\mathbf{e}_i$ and a state st. $\mathsf{Answer}(\mathbf{D}, q) \mapsto (\tau_d(\mathbf{D}) \times q, \pi)$ outputs a GLWE encryption of $\mathbf{D}[i]$ as well as a proof attesting to the validity of the answer. $\mathsf{Recover}(\mathsf{st}, d, q, a, \pi)$ first checks the validity of the proof π with respect to the digest d, the query q and the answer a. By the perfect completeness of PCS and IOP, both DigVer and the proof verification will succeed with probability 1 for honestly generated digests and proofs. Then, $\mathsf{Recover}$ decrypts a using the state st and outputs the result. The decryption of a will yield $\mathbf{D}[i]$ with probability $1 - \epsilon_{\mathsf{GLWE}}$ where ϵ_{GLWE} is the correctness error of the underlying GLWE encryption scheme.

B.2 Proof of Lemma 4

Proof. Let $\mathcal{A}$ be any PPT adversary acting as the server in Figure 3 and $\mathcal{E}^{\mathcal{P}^*}$ be the extractor guaranteed by the knowledge soundness property of PCS. Let $(d, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{pp})$ be the digest generated by $\mathcal{A}$ after receiving the public parameters pp. Let $(q, \mathsf{st}) \leftarrow \mathit{\Pi}.\mathsf{Query}(i)$ be an honestly generated query-state pair for some index $i \in [N]$. Let $(a, \pi) \leftarrow \mathcal{P}^*(\mathsf{pp}, \mathsf{st}_{\mathcal{A}}, q)$ be the answer-proof pair generated by $\mathcal{P}^*$ after receiving the query q, and let ρ, σ, τ be the verifier public coins defined over the course of the IOP (cf. Section 5.2). Suppose that $\mathsf{DigVer}(d) = \mathsf{Accept}$ and $\mathit{\Pi}.\mathsf{Recover}(\mathsf{st}, d, q, a, \pi) \neq \bot$. Then, both (i) the PCS verification procedure and (ii) the IOP verification procedure succeed. From the former, we have by the knowledge soundness of the PCS that the (rewinding) extractor $\mathcal{E}^{\mathcal{A}, \mathcal{P}^*(\mathsf{pp}, \mathsf{st}_{\mathcal{A}}, q)}(\mathsf{pp}, d, q, \pi)$ outputs with probability $1 - \epsilon_{\mathsf{PCS}}$ a multilinear extension $\widetilde{\widetilde{\mathbf{M}}}$ such that

$$\widetilde{\widetilde{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \tau) = v \tag{15}$$

for some $v \in \mathcal{C}$. Further, since $\mathsf{DigVer}(d) = \mathsf{Accept}$, the knowledge soundness of the PCS applied to $c_{\mathbf{A}}$ and the soundness of the well-formedness proof imply that, with probability $1 - \epsilon_{\mathsf{PCS}} - \epsilon_{\mathsf{Digest}}$, the extractor also outputs $\widetilde{\mathbf{A}}$ with $\mathbf{A} \in \mathbb{Z}_p^{m \times nd}$ and $\bar{\mathbf{M}} = \mathsf{NTT}(\tau_d(\mathbf{A}))$. We set $\mathbf{D} \coloneqq \mathbf{A}$. From (ii), we have that the IOP verifier accepts given this evaluation v. By the soundness of the IOP, this means that with probability $1 - \epsilon_{\mathsf{IOP}}$, $a = \mathbf{M} \times q$ over $\mathcal{R}_q$. Recall from Section 4 that since q is an encryption of $\mathbf{e}_i$ and $\mathbf{A} \in \mathbb{Z}_p^{m \times nd}$, then a is an encryption of $\mathbf{D}[i]$ whose noise is distributed as in an honest execution. Finally, the correctness of the GLWE encryption scheme implies that $\mathsf{Recover}(\mathsf{st}, d, q, a, \pi) = \mathbf{D}[i]$ with probability $1 - \epsilon_{\mathsf{GLWE}}$. Putting everything together, we have that $\mathsf{Recover}(\mathsf{st}, d, q, a, \pi) \neq \mathbf{D}[i]$ with probability at most $(2\epsilon_{\mathsf{PCS}} + \epsilon_{\mathsf{Digest}} + \epsilon_{\mathsf{IOP}} + \epsilon_{\mathsf{GLWE}}) \in \mathsf{negl}(\lambda)$.

B.3 Proof of Lemma 5

Proof. If $\mathsf{DigVer}(d) = \mathsf{Reject}$, both experiments halt before any query is issued and are identically distributed. We thus assume $\mathsf{DigVer}(d) = \mathsf{Accept}$ in what follows. The experiment $\mathsf{REAL}_{\mathcal{A},i}$ depends on the queried index i in two places: the query $q \leftarrow \mathsf{Query}(i)$ and the bit $b = [\mathsf{Recover}(\mathsf{st}, d, q, a, \pi) = \mathbf{D}[i]]$. We define the simulator $\mathcal{S}$ as follows: on input pp, it samples $j \xleftarrow{\$} [N]$, runs $(q, \mathsf{st}) \leftarrow \mathsf{Query}(j)$, and outputs q; on input $(\mathsf{pp}, d, q, a, \pi)$, it outputs the verification bit $b' \coloneqq [\mathsf{Recover}(\mathsf{st}, d, q, a, \pi) \neq \bot]$. We show $\mathsf{REAL}_{\mathcal{A},i} \approx_c \mathsf{IDEAL}_{\mathcal{A},\mathcal{S}}$ via two hybrids. Let H_1 denote the experiment obtained from $\mathsf{REAL}_{\mathcal{A},i}$ by replacing b with b'. Since $\mathbf{D}[i] \neq \bot$, we have $b \neq b'$ only if $\mathsf{Recover} \neq \bot$ and $\mathsf{Recover} \neq \mathbf{D}[i]$, which by Lemma 4 occurs with probability at most $2\epsilon_{\mathsf{PCS}} + \epsilon_{\mathsf{Digest}} + \epsilon_{\mathsf{IOP}} + \epsilon_{\mathsf{GLWE}} \in \mathsf{negl}(\lambda)$ for any PPT $\mathcal{A}$. Hence $\mathsf{REAL}_{\mathcal{A},i} \approx_c H_1$. Let H_2 denote H_1 with $\mathsf{Query}(i)$ replaced by $\mathsf{Query}(j)$ for $j \xleftarrow{\$} [N]$. A distinguisher between H_1 and H_2 yields a distinguisher against the passive privacy of the underlying PIR scheme, which is computationally hard under the GLWE assumption: all other components of the experiment are simulable from pp and q alone. Hence $H_1 \approx_c H_2$. Finally, H_2

and $\mathsf{IDEAL}_{\mathcal{A},\mathcal{S}}$ are identically distributed by construction of $\mathcal{S}$, which concludes the proof.

C Efficient sum-check for matrix-vector multiplication over rings

We now present the design of our efficient algorithms (both prover and verifier) for the IOP described above. That is, despite the fact that the sum-check protocol targets a $\log mdnd$-variate polynomial $\widetilde{\mathbf{N}}$, the prover runs in time $\mathcal{O}(mnd)$ which is sublinear in the size of the input $\mathbf{N}$. To achieve this, we exploit the structure of the large matrix $\mathbf{N}$ to avoid ever materializing it, thereby saving a factor d in performance.

Let

$$S = \sum_{\beta \in \{0,1\}^\nu} p(b)$$

be the subject of an execution of the sum-check protocol where p is a ν-variate polynomial. We know from [Tha13,XZZ+19] that there exists a linear time algorithm that takes as input 1) the table $\mathbf{A}_p$ containing the list of evaluations of p on the boolean hypercube $\{0,1\}^\nu$ and 2) the verifier challenges $(r_1, \ldots, r_\nu) \in \mathcal{C}^\nu$ and outputs the list of prover messages of the sum-check protocol (the univariate polynomials $p_i(X)$ for $i \in [\nu]$). We omit the details of this linear time algorithm and refer the reader to the aforementioned works. The table $\mathbf{A}_p$ is referred to as a "bookkeeping table". Computing the initial values of the bookkeeping table $\mathbf{A}_p$ is a non-trivial task and may be computationally expensive depending on the setting. In what follows, we give optimal algorithms to compute the relevant bookkeeping tables for both instances of the sum-check protocol described above. We assume the prover holds the matrix $\bar{\mathbf{M}}$, i.e. the NTT representation of $\mathbf{M}$. Since the matrix $\mathbf{M}$ is fixed for all client queries, the prover can compute $\bar{\mathbf{M}}$ at preprocessing time and store it for later use, and dispose of $\mathbf{M}$.

Computing the bookkeeping table for the first sum-check Recall that the first sum-check protocol is applied to the statement in Equation (4) which involves the $\log mdnd$-variate polynomial $\widetilde{\mathbf{N}}(\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z})$ with the first $\log md$ variables fixed at the random challenge $\rho \in \mathcal{C}^{\log md}$ sent by the verifier. As discussed above, provided we can initialize the table containing all the evaluations of $\widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l})$ for all $\mathbf{k} \in \{0,1\}^{\log n}, \mathbf{l} \in \{0,1\}^{\log d}$, there exists a prover algorithm running in time $\mathcal{O}(nd)$. Using the definition of multilinear extensions, it is not hard to check that generating this list of evaluations is equivalent to taking a weighted linear combination of the md rows of the matrix $\mathbf{N}$ where the weights are defined by the entries of the vector $\mathsf{tensor}(\rho)$. More precisely, the list of evaluations required is exactly the vector

$$\mathsf{tensor}(\rho) \times \mathbf{N}.$$

Since $\mathbf{N} \in \mathbb{F}_q^{md \times nd}$, computing this can be done naively in $\mathcal{O}(d^2mn)$ time. However, it is possible to exploit the structure of $\mathbf{N}$ to perform this task more efficiently. Recall that each block $\mathbf{N}_{i,k}$ of $\mathbf{N}$ is a skewed circulant matrix defined by the vector $\mathbf{M}_{i,k}$, i.e.

$$\mathbf{N}_{i,k} = \mathsf{circ}(\mathbf{M}_{i,k}) = \mathbf{F}' \times \mathsf{diag}(\bar{\mathbf{M}}_{i,k}) \times \mathbf{F}.$$

Observe also that for any two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^d$ and scalars $\alpha, \beta \in \mathbb{F}_q$, we have that

$$\alpha \cdot \mathsf{circ}(\mathbf{u}) + \beta \cdot \mathsf{circ}(\mathbf{v}) = \mathsf{circ}(\alpha\mathbf{u} + \beta\mathbf{v}).$$

This allows us to compute $\mathsf{tensor}(\rho) \times \mathbf{N}$ efficiently in two steps. Let $\rho = (\rho_{\mathsf{outer}}, \rho_{\mathsf{inner}})$ so that $\mathsf{tensor}(\rho) = \mathsf{tensor}(\rho_{\mathsf{outer}}) \otimes \mathsf{tensor}(\rho_{\mathsf{inner}})$. We first take a weighted linear combination of the md rows of $\mathbf{N}$ split in chunks of size d, where the weights are defined by the entries of $\mathsf{tensor}(\rho_{\mathsf{outer}}) \in \mathcal{C}^m$. This can be done efficiently in practice by simply taking a linear combination of the m vectors $\bar{\mathbf{M}}_{i,*} \coloneqq (\bar{\mathbf{M}}_{i,1}, \bar{\mathbf{M}}_{i,2}, \ldots, \bar{\mathbf{M}}_{i,n})$ for $i \in [m]$ that define each row chunk. More precisely, in terms of the block matrix $\mathbf{N}$, this can be expressed as

$$\begin{aligned}
\mathbf{N}' &\coloneqq \sum_{i \in [m]} \mathsf{tensor}(\rho_{\mathsf{outer}})_i \cdot \mathbf{N}_{i,*} \qquad (16)\\
&= \sum_{i \in [m]} \mathsf{tensor}(\rho_{\mathsf{outer}})_i \cdot (\mathsf{circ}(\mathbf{M}_{i,1}), \mathsf{circ}(\mathbf{M}_{i,2}), \ldots, \mathsf{circ}(\mathbf{M}_{i,n}))\\
&= \left(\mathsf{circ}\left(\sum_{i \in [m]} \mathsf{tensor}(\rho_{\mathsf{outer}})_i \cdot \mathbf{M}_{i,1} \right), \ldots, \mathsf{circ}\left(\sum_{i \in [m]} \mathsf{tensor}(\rho_{\mathsf{outer}})_i \cdot \mathbf{M}_{i,n} \right) \right)
\end{aligned}$$

which lies in $\mathcal{C}^{d \times nd}$. Importantly, this new matrix is never materialized as it is entirely determined by the n vectors of dimension d $\bar{\mathbf{M}}'_k \coloneqq \sum_{i \in [m]} \mathsf{tensor}(\rho_{\mathsf{outer}})_i \cdot \bar{\mathbf{M}}_{i,k}$ for all $k \in [n]$. In practice however, the prover computes these n vectors by applying the weighted linear combination to the m rows of $\bar{\mathbf{M}}$. In other words, it never materializes the full matrix $\mathbf{N}$ but rather operates on the m vectors of dimension nd that define it (i.e., the rows of $\bar{\mathbf{M}}$), which allows it to compute $\mathbf{N}'$ in time $\mathcal{O}(mnd)$. This time is linear in the size of $\bar{\mathbf{M}}$ and consequently sublinear in the size of $\mathbf{N}$. Figure 5 depicts the above in relation to $\mathbf{N}$.

For the second step, we observe that by the definition of tensor,

$$\begin{aligned}
\mathsf{tensor}(\rho) \times \mathbf{N} &= (\mathsf{tensor}(\rho_{\mathsf{outer}}) \otimes \mathsf{tensor}(\rho_{\mathsf{inner}})) \times \mathbf{N}\\
&= \mathsf{tensor}(\rho_{\mathsf{inner}}) \times \mathbf{N}'
\end{aligned}$$

We can thus conclude the computation of $\mathsf{tensor}(\rho) \times \mathbf{N}$ by using $\mathbf{N}'$ to compute the following matrix-vector product:

$$\mathsf{tensor}(\rho_{\mathsf{inner}}) \times \mathbf{N}' \qquad (17)$$

which may be computed in time in $\mathcal{O}(nd \log d)$ by exploiting the circulant structure of the blocks of $\mathbf{N}'$. To see this, notice that the vector $\mathsf{tensor}(\rho_{\mathsf{inner}})$ is of

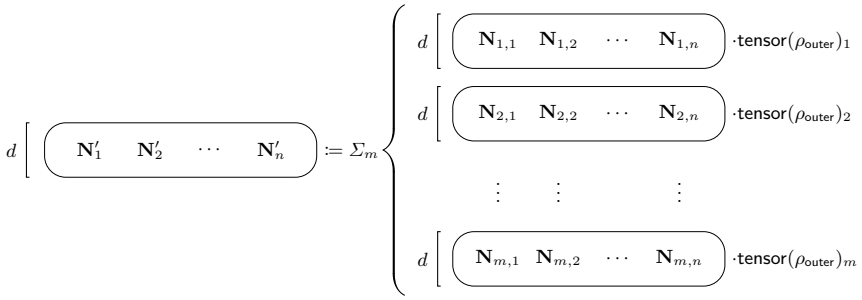

Fig. 5. First step in computing $\mathsf{tensor}(\rho) \times \mathbf{N}$. A weighted linear combination of chunks of rows of $\mathbf{N}$ is taken using weights defined by the entries of $\mathsf{tensor}(\rho_{\mathsf{outer}})$. The result lies in $\mathcal{C}^{d \times nd}$. In practice, the prover does not compute this and instead operates on the defining vectors of $\mathbf{N}$ to obtain the defining vectors of $\mathbf{N}'$.

dimension d and the matrix $\mathbf{N}'$ is of dimension $d \times nd$, where the k-th block of dimension $d \times d$ is of the form $\mathsf{circ}(\bar{\mathbf{M}}'_k)$ with $\mathbf{M}'_k = \mathsf{NTT}^{-1}(\bar{\mathbf{M}}'_k) \in \mathcal{R}_q$. The key insight here is that by viewing $\mathsf{tensor}(\rho_{\mathsf{inner}})$ as an element of $\mathcal{R}_q$, the operation in Equation (17) can be expressed as n products in $\mathcal{R}_q$. $\mathcal{R}_q$-products can be computed in $\mathcal{O}(d \log d)$ yielding the claimed time complexity of $\mathcal{O}(nd \log d)$ for computing $\mathsf{tensor}(\rho_{\mathsf{inner}}) \times \mathbf{N}'$. We formalize this in the following lemmas.

Lemma 6. *Let $\mathcal{R}_q$ be a power-of-two cyclotomic ring splitting fully over $\mathbb{F}_q$, $r(X) \in \mathcal{R}_q$ and $\mathbf{r} \coloneqq \mathsf{coeff}(r(X))$. Let $\mathsf{aut}_{-1} : \mathcal{R}_q \to \mathcal{R}_q$ be the automorphism mapping $r(X)$ to $r(X^{-1})$. Then, $\mathbf{r} \times \mathbf{F} = \mathsf{aut}_{-1}(d \cdot \mathbf{F}' \times \mathbf{r})$ and $\mathbf{r} \times \mathbf{F}' = \mathsf{rev}(d^{-1} \cdot \mathbf{F} \times \mathbf{r})$.*

Proof. Firstly, note that

$$\mathbf{F} \times \mathbf{r} = \mathsf{NTT}(r(X)) \quad \text{and} \quad \mathbf{F}' \times \mathbf{r} = \mathsf{NTT}^{-1}(\mathbf{r}).$$

Since both the NTT and aut_{-1} are isomorphisms, the order of their composition may be swapped. We recall the following facts related to these isomorphisms.

$$\mathsf{NTT}(\mathsf{aut}_{-1}(r(X))) = \mathsf{rev}(\mathsf{NTT}(r(X)))$$
$$\mathsf{NTT}^{-1}(\mathsf{rev}(\mathbf{r})) = \mathsf{aut}_{-1}(\mathsf{NTT}^{-1}(\mathbf{r}))$$

These equations characterize the commutating diagram engendered by these transforms. With this in hand, the lemma mostly follows by inspection. Namely, we have that

$$\mathbf{r} \times \mathbf{F} = \mathbf{F}^\top \times \mathbf{r}^\top \tag{18}$$

where $\mathbf{F}^\top$ is essentially $\mathbf{F}'$ where each entry is replaced by its multiplicative inverse and scaled by $1/d$, i.e. $\mathbf{F}^\top_{i,j} = \frac{1}{d}(\mathbf{F}'_{j,i})^{-1}$. As such, by the definition of aut_{-1}, we can replace $\mathbf{F}^\top$ by $\mathbf{F}' \cdot d$ in the right hand side of Equation (18) provided we apply the automorphism to $\mathbf{r}$, i.e.

$$\mathbf{F}^\top \times \mathbf{r}^\top = \mathbf{F}' \cdot d \times \mathsf{rev}(\mathbf{r})$$

which yields the first identity in the lemma. An analogous argument allows us to obtain the second identity.

Lemma 7. *Let $\mathcal{R}_q$ be a power-of-two cyclotomic ring splitting fully over $\mathbb{F}_q$ and $r, s \in \mathcal{R}_q$. Let $\mathsf{aut}_{-1} : \mathcal{R}_q \to \mathcal{R}_q$ be the automorphism mapping $a(X)$ to $a(X^{-1})$. Then, $\mathsf{coeff}(s) \times (\mathbf{F}' \times \mathsf{diag}(\mathsf{NTT}(r)) \times \mathbf{F}) = s \cdot \mathsf{aut}_{-1}(r)$.*

Proof.

$$
\begin{aligned}
&\mathsf{coeff}(s) \times (\mathbf{F}' \times \mathsf{diag}(\mathsf{NTT}(r)) \times \mathbf{F}) \\
&= (\mathbf{F}^\top \times \mathsf{diag}(\mathsf{NTT}(r))^\top \times \mathbf{F}'^\top) \times \mathsf{coeff}(s)^\top \\
&= \mathbf{F}^\top \times \mathsf{diag}(\mathsf{NTT}(r)) \times d^{-1} \cdot \mathbf{F} \times \mathsf{coeff}(\mathsf{aut}_{-1}(s))^\top \\
&= \mathbf{F}^\top \times d^{-1} \cdot (\mathsf{NTT}(r) \circ \mathsf{NTT}(\mathsf{aut}_{-1}(s))) \\
&= \mathsf{aut}_{-1}(\mathbf{F}' \times d \cdot d^{-1} \cdot (\mathsf{NTT}(r) \circ \mathsf{NTT}(\mathsf{aut}_{-1}(s)))) \\
&= \mathbf{F}' \times (\mathsf{NTT}(\mathsf{aut}_{-1}(r)) \circ \mathsf{NTT}(s)) \\
&= \mathsf{NTT}^{-1}(\mathsf{NTT}(\mathsf{aut}_{-1}(r)) \circ \mathsf{NTT}(s)) \\
&= \mathsf{aut}_{-1}(r) \cdot s
\end{aligned}
$$

where the first equality follows from the fact that $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$, the second and fourth equalities hold by Lemma 6 and the fifth equality by isomorphism.

As discussed above, from Lemmas 6 and 7 we have that in practice the computation expressed in Equation (17) can be performed efficiently by performing n $\mathcal{R}_q$-multiplications between the ring element having $\mathsf{tensor}(\rho)$ as coefficients and $\mathsf{aut}_{-1}(\mathbf{M}'_k)$. More formally and in terms of $\bar{\mathbf{M}}'$, we have that

$$
\begin{aligned}
\mathsf{tensor}(\rho_{\mathsf{inner}}) \times \mathbf{N}' = (&\mathsf{NTT}^{-1}(\mathsf{NTT}(\mathsf{tensor}(\rho_{\mathsf{inner}})) \circ \mathsf{rev}(\bar{\mathbf{M}}'_1)), \\
&\ldots, \\
&\mathsf{NTT}^{-1}(\mathsf{NTT}(\mathsf{tensor}(\rho_{\mathsf{inner}})) \circ \mathsf{rev}(\bar{\mathbf{M}}'_n)))
\end{aligned}
$$

which, given Lemma 8, may be computed in time in $\mathcal{O}(nd \log d)$ as claimed.

Lemma 8. *Let $\mathcal{R}_q \coloneqq \mathbb{F}_q[X]/\langle X^d+1\rangle$ be a power-of-two cyclotomic ring splitting fully over $\mathbb{F}_q$. Let $\mathbb{K}$ be an extension field of $\mathbb{F}_q$ of fixed degree δ and let $\mathcal{R}_\mathbb{K} \coloneqq \mathbb{K}[X]/\langle X^d + 1\rangle$. If there exists an algorithm in $\mathcal{O}(d \log d)$ to compute the NTT of $r(X) \in \mathcal{R}_q$, then there exists an algorithm in $\mathcal{O}(d \log d)$ to compute the NTT of $s(X) \in \mathcal{R}_\mathbb{K}$.*

Proof. Let $(1, Y, Y^2, \ldots, Y^{\delta-1})$ be a basis of $\mathbb{K}$ over $\mathbb{F}_q$. Then any element $s(X) \in \mathcal{R}_\mathbb{K}$ may be represented as $s(X) = \sum_{i=0}^{\delta-1} s_i(X) Y^i$ where $s_i(X) \in \mathcal{R}_q$. Since the ring fully splits over $\mathbb{F}_q$, we have that $\mathbb{F}_q$ contains a $2d$-th root of unity ω, as well as its odd powers $(\omega, \omega^3, \omega^5, \ldots, \omega^{2d-1})$. Since $\mathsf{NTT}(s(X))$ is defined as the evaluations of $s(X)$ at these roots of unity which hold no weight in the dimensions $(Y, \ldots, Y^{\delta-1})$, we have by linearity that

$$
\mathsf{NTT}(s(X)) = \left(\sum_{i=0}^{\delta-1} \mathsf{NTT}(s_i(X))_j Y^i \right)_{j \in [d]}
$$

In sum, the prover may compute the table containing the evaluations of $\widetilde{\mathbf{N}}(\rho, \mathbf{k}, \mathbf{l})$ for all $\mathbf{k} \in \{0,1\}^{\log n}, \mathbf{l} \in \{0,1\}^{\log d}$ in time in $\mathcal{O}(mnd + nd \log d)$.

Computing the bookkeeping tables for the second sum-check Recall that the second instance of the sum-check protocol is applied to the statement in Equation (7) which involves the product of three partially evaluated multilinear polynomials:

1. $\widetilde{\mathbf{F}}'(\rho_{\mathsf{inner}}, \mathbf{l})$, the $\log d^2$-variate multilinear extension of the inverse NTT matrix $\mathbf{F}'$ with the first $\log d$ variables fixed at the random challenge $\rho_{\mathsf{inner}} \in \mathcal{C}^{\log d}$;
2. $\widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{l})$, the $\log mnd$-variate multilinear extension of the matrix $\bar{\mathbf{M}}$ with the first $\log m$ variables fixed at the random challenge $\rho_{\mathsf{outer}} \in \mathcal{C}^{\log m}$ and the subsequent $\log n$ variables fixed at the random challenge $\sigma_{\mathsf{outer}} \in \mathcal{C}^{\log n}$;
3. $\widetilde{\mathbf{F}}(\mathbf{l}, \sigma_{\mathsf{inner}})$, the $\log d^2$-variate multilinear extension of the NTT matrix $\mathbf{F}$ with the last $\log d$ variables fixed at the random challenge $\sigma_{\mathsf{inner}} \in \mathcal{C}^{\log d}$.

Again, [Tha13,XZZ+19] shows that there exists a linear time algorithm that takes as input the tables containing the list of evaluations of the three multilinear polynomials above on the boolean hypercube and the verifier challenges and outputs the prover messages of the sum-check protocol. In what follows we give efficient algorithms to compute the required tables in time $\mathcal{O}(nd)$ thereby allowing us to employ the $\mathcal{O}(d)$ algorithm to generate the prover messages.

Computing the values for the three bookkeeping tables naively would take time d^2, mnd and d^2 respectively. However, we may again take advantage of the structure of the matrices to gain in performance.

Bookkeeping tables for $\mathbf{F}'$ *and* $\mathbf{F}$. For the first and third bookkeeping tables, we borrow a technique originally proposed for the Fast Fourier Transform (FFT) setting in [LXZ21] and adapted to the NTT setting in [LLX+26]. We overview the technique here and refer the reader to the cited papers for details.

As discussed in the Appendix C, fixing the first variables of the multilinear extension of a matrix to a challenge vector γ amounts to taking a weighted linear combination of the rows of the matrix, where the weights are defined by the entries of the vector $\mathsf{tensor}(\gamma)$. The insight from [LXZ21,LLX+26] is that computing a product between a matrix exhibiting NTT-like structure and a vector exhibiting tensor structure can be done in time $\mathcal{O}(d)$ using dynamic programming. We adapt their techniques to our setting and give the algorithms to compute the two bookkeeping tables for $\mathbf{F}'$ and $\mathbf{F}$ in Algorithms 1 and 2 respectively. We refer the reader to the cited works for details on the correctness and performance of the algorithms.

Bookkeeping table for $\mathbf{M}$. The second bookkeeping table should contain the evaluations of $\widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{l})$ for all $\mathbf{l} \in \{0,1\}^{\log d}$. We leverage the fact that the matrix $\bar{\mathbf{M}}' \in \mathcal{C}^{nd}$ computed in Appendix C already represents the evaluations of $\widetilde{\bar{\mathbf{M}}}(\rho_{\mathsf{outer}}, \mathbf{k}, \mathbf{l})$ on the boolean hypercube. In other words, in $\bar{\mathbf{M}}'$, the first $\log m$

variables of $\widetilde{\widetilde{\mathbf{M}}}$ are already fixed to ρ_{outer} and it remains to fix the subsequent $\log n$ variables to σ_{outer} to obtain the required table. Observe again that fixing the subsequent $\log n$ variables to σ_{outer} amounts to taking a weighted linear combination of the n subvectors of dimension d which define $\bar{\mathbf{M}}'$, where the weights are defined by the entries of $\mathsf{tensor}(\sigma_{\mathsf{outer}})$. That is, $\forall \mathbf{b} \in \{0,1\}^{\log d}$,

$$\widetilde{\widetilde{\mathbf{M}}}(\rho_{\mathsf{outer}}, \sigma_{\mathsf{outer}}, \mathbf{b}) \coloneqq \sum_{k \in [n]} \mathsf{tensor}(\sigma_{\mathsf{outer}})_k \cdot \bar{\mathbf{M}}'_k \in \mathcal{C}^d$$

which may be computed in time in $\mathcal{O}(nd)$. This is depicted in Figure 6.

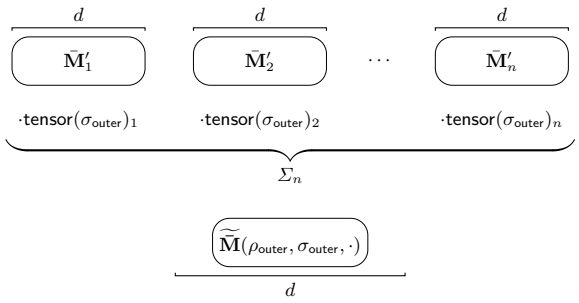

Fig. 6. Computing the second bookkeeping table using $\bar{\mathbf{M}}'$. The required values are obtained by taking a weighted linear combination of the n subvectors of dimension d, where the weights are defined by the entries of $\mathsf{tensor}(\sigma_{\mathsf{outer}})$.

In sum, the prover may compute all required bookkeeping tables for the second sum-check in time in $\mathcal{O}(nd)$. This allows us to employ the $\mathcal{O}(d)$-time algorithm to generate the prover messages of the second sum-check, yielding an overall time complexity of $\mathcal{O}(nd)$ for the prover in the second sum-check protocol.

D Deferred Algorithms

Algorithms 1 and 2 give the dynamic-programming routines used in Appendix C to initialize the bookkeeping tables for $\widetilde{\mathbf{F}}'$ and $\widetilde{\mathbf{F}}$ respectively.

Algorithm 1 $\mathbf{T} \leftarrow$ Initialize_inv_ntt(ω, ρ, d)

Require: $2d$-th root of unity ω, random point $\rho \in \mathcal{C}^{\log d}$ and degree d;
Ensure: $\mathbf{T}$ storing $\widetilde{\mathbf{F}}'(\rho, \mathbf{l})$ for all $\mathbf{l} \in \{0,1\}^{\log d}$.

$\mathbf{T}[0] \leftarrow 1$;
for $i \leftarrow 0, \ldots, \log d - 1$ **do**
  layer_root $\leftarrow \omega^{d/2^{i+1}}$;
  for $j \leftarrow 2^i - 1, \ldots, 0$ **do**
    pow_omega $\leftarrow$ layer_root^{-2j+1};
    prev $\leftarrow \mathbf{T}[j]$;
    $\mathbf{T}[j + 2^i] =$ prev $\cdot (1 - \rho_i - \rho_i \cdot$ pow_omega$)$;
    $\mathbf{T}[j] =$ prev $\cdot (1 - \rho_i + \rho_i \cdot$ pow_omega$)$;
  end for
end for
return $\mathbf{T} \cdot d^{-1}$;

Algorithm 2 $\mathbf{T} \leftarrow$ Initialize_fwd_ntt(ω, σ, d)

Require: $2d$-th root of unity ω, random point $\sigma \in \mathcal{C}^{\log d}$ and degree d;
Ensure: $\mathbf{T}$ storing $\widetilde{\mathbf{F}}(\mathbf{l}, \sigma)$ for all $\mathbf{l} \in \{0,1\}^{\log d}$.

$\mathbf{T}[0] \leftarrow 1$;
for $i \leftarrow 0, \ldots, \log d - 1$ **do**
  layer_root $\leftarrow \omega^{d/2^{i+1}}$;
  for $j \leftarrow 2^i - 1, \ldots, 0$ **do**
    pow_omega $\leftarrow$ layer_root^{2j+1};
    prev $\leftarrow \mathbf{T}[j]$;
    $\mathbf{T}[j + 2^i] =$ prev $\cdot (1 - \sigma_i - \sigma_i \cdot$ pow_omega$)$;
    $\mathbf{T}[j] =$ prev $\cdot (1 - \sigma_i + \sigma_i \cdot$ pow_omega$)$;
  end for
end for
return $\mathbf{T}$;

E Deferred Figures and Tables

Table 5. Compressed client upload size, using parameters from the first row of Table 1 (in kB). For convenience, we omit the 128-bit seed in the upload size.

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

Figure 7 reports client-side timings for the remaining values of n, and Table 7 reports server-side timings for all combinations of m and n.

Table 6. Sample extracted client download size, using parameters from the first row of Table 1 (in kB). For convenience, we omit the $m \cdot \log q$ bits of body components in the download size.

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

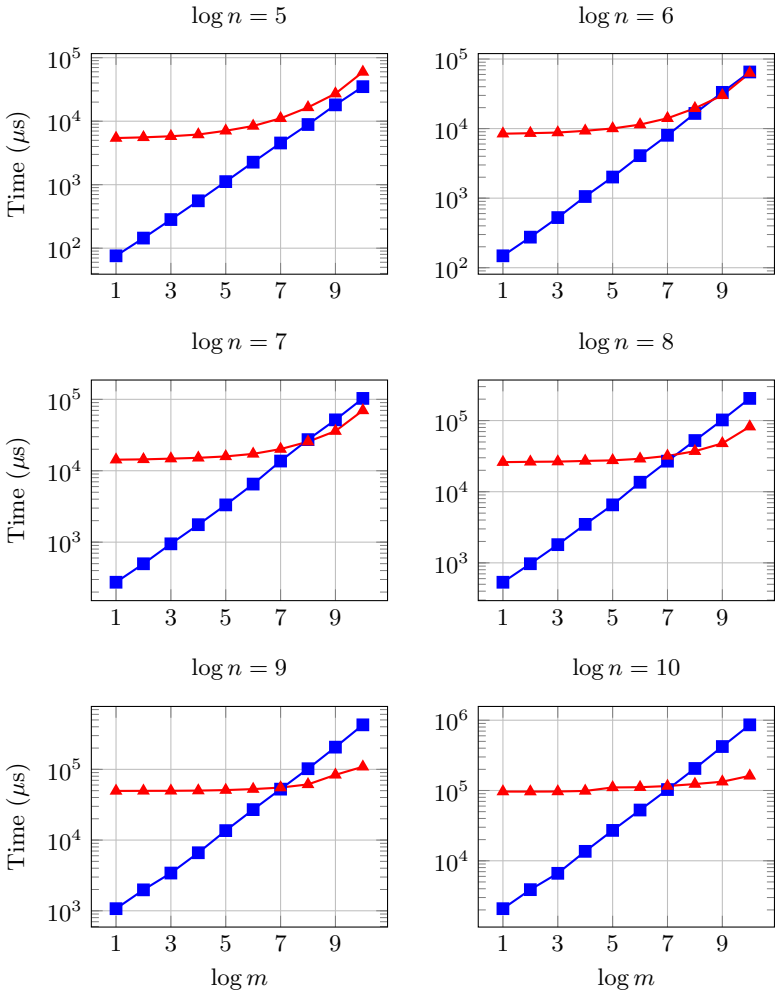

Fig. 7. Plaintext matrix–vector product (blue) vs. verification pipeline (red), log–log scale, for various n values.

Table 7. Prover computation time (in seconds) as a function of n and m. Each cell shows with PCS (top) and without PCS (bottom).

$\log n$ \ $\log m$	1	2	3	4	5	6	7	8	9	10
1	0.02	0.03	0.03	0.04	0.07	0.10	0.14	0.21	0.31	0.66
	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.03
2	0.03	0.04	0.05	0.07	0.11	0.14	0.20	0.33	0.66	1.04
	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.02	0.03
3	0.04	0.05	0.07	0.10	0.14	0.19	0.29	0.65	1.06	2.14
	0.01	0.01	0.01	0.02	0.02	0.02	0.02	0.02	0.03	0.04
4	0.05	0.07	0.10	0.14	0.20	0.27	0.61	1.06	2.14	3.57
	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.03	0.04	0.06
5	0.08	0.11	0.15	0.20	0.29	0.60	1.01	2.11	3.54	7.80
	0.03	0.03	0.03	0.03	0.02	0.03	0.03	0.05	0.07	0.13
6	0.12	0.16	0.21	0.30	0.64	1.05	2.14	3.60	7.76	13.28
	0.04	0.04	0.04	0.04	0.04	0.04	0.05	0.07	0.13	0.24
7	0.17	0.23	0.33	0.66	1.07	2.18	3.61	7.76	13.34	29.27
	0.05	0.05	0.05	0.05	0.05	0.06	0.08	0.14	0.25	0.83
8	0.26	0.33	0.55	0.74	1.55	3.07	7.60	13.39	29.31	52.56
	0.07	0.07	0.08	0.07	0.08	0.10	0.16	0.28	0.85	1.68
9	0.40	0.74	1.16	2.26	3.70	7.90	13.51	29.62	52.53	117.42
	0.11	0.11	0.12	0.13	0.13	0.20	0.32	0.93	1.70	3.23
10	0.87	1.26	2.41	3.83	8.00	13.65	29.77	52.98	117.68	210.86
	0.22	0.22	0.23	0.24	0.27	0.42	1.02	2.02	3.75	7.17