VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers

Leo de Castro, *MIT;* Keewoo Lee, *Seoul National University*

https://www.usenix.org/conference/usenixsecurity24/presentation/de-castro

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

VeriSimplePIR

Verifiability in SimplePIR at No Online Cost for Honest Servers

Leo de Castro
Massachusetts Institute of Technology

Keewoo Lee
Seoul National University

Abstract

We present VeriSimplePIR, a verifiable version of the state-of-the-art semi-honest SimplePIR protocol. VeriSimplePIR is a stateful verifiable PIR scheme guaranteeing that all queries are consistent with a fixed, well-formed database. It is the first efficient verifiable PIR scheme to not rely on an honest digest to ensure security; any digest, even one produced by a malicious server, is sufficient to commit to some database. This is due to our *extractable* verification procedure, which can extract the entire database from the consistency proof checked against each response.

Furthermore, VeriSimplePIR ensures this strong security guarantee without compromising the performance of SimplePIR. The online communication overhead is roughly 1.1-1.5× SimplePIR, and the online computation time on the server is essentially the *same*. We achieve this low overhead via a novel one-time *preprocessing* protocol that generates a *reusable* proof that can verify any number of subsequent query-response pairs as long as no malicious behavior is detected. As soon as the verification procedure rejects a response from the server, the offline phase must be rerun to compute a new proof. VeriSimplePIR represents an approach to maliciously secure cryptography that is highly optimized for honest parties while maintaining security even in the presence of malicious adversaries.

1 Introduction

Private information retrieval (PIR) [CKGS98] addresses the question of how users can fetch a record from a database without leaking their access patterns to the machine hosting the database. PIR schemes have found numerous uses in privacy-preserving applications, such as certificate transparency [LG15], metadata-hiding messaging [AS16, ACLS18], password-breach alerting [TPY+19, LPA+19, PIB+22], private contact discovery [KRS+19], and many others.

Despite the ubiquity of PIR, essentially all constructions, including all practical constructions, do not address the fundamental issue of *verifiability*. In other words, practical PIR schemes crucially assume that the host server behaves at least *semi-honestly*, which is to say that it correctly executes the protocol with some fixed database. However, in the real world, a malicious server can deviate arbitrarily from the prescribed protocol with potentially disastrous consequences. One class of attacks involves equivocating on the database used to respond to each client. Such attacks allow a malicious server to alter all entries of the database for a particular client, effectively choosing the client's output. Malicious servers can even compromise the client's query privacy by performing a *selective-failure attack* [KO97, KS06, HKE13], where the server mauls specific entries of the database in an attempt to learn the client's query.

At a higher level, the deployment of any semi-honest PIR scheme will be plagued by the strong assumptions required for security. The reality of many distributed systems is that "honest" behavior cannot always be guaranteed even without true malicious actors. Deviations from the honest protocol could be caused by software bugs, network delays, faulty hardware, or any number of emergent failures that occur when computing at scale. Any system with security that is contingent on such failures *never* occurring will be hard to justify in sensitive applications. Nevertheless, true malicious behavior remains exceedingly rare, and a major optimization challenge is how to achieve security against malicious adversaries without sacrificing the performance of the best semi-honest protocols. In this work, we address the challenge of verifiable, single-server PIR by asking the following question:

How can a client ensure that a server is using a fixed, well-formed database to respond to all PIR queries while maintaining the efficiency of semi-honest PIR?

1.1 Prior Work

Semi-honest PIR. The starting point for this work is the semi-honest PIR protocol of Henzinger et al. [HHCG+23], referred to as SimplePIR. This protocol represents the state-of-the-art

in semi-honest PIR, where queries can be answered as fast as the database can be read from memory. SimplePIR is a stateful PIR protocol, where the server sends an initial, one-time *digest* of the database to each client. The clients then use this digest to improve the efficiency of all subsequent queries. This work will follow the same approach, having each client engage in a one-time protocol with the server, and the output of this protocol is used to process and verify queries in the online phase.

Verifiable PIR. The verifiable PIR primitive was first introduced in the multi-server setting by Zhang and Safavi-Naini [ZSN14] and in the single-server setting by Wang, Zhao, and Huang [WZ18, ZWH21]. In these works, the client issues a challenge with each query to ensure that the server correctly computed the response. These challenges resemble plaintext checksums, essentially requiring that the server evaluate the PIR computation on several random queries in addition to the real query. Unlike our work, these verifiable PIR schemes do not have a database commitment, so they do not defend against attacks where a malicious server changes the database in between queries.

The work of Ben-David, Kalia, and Paneth [BDKP22] extends the notion of verifiable PIR by verifying more complex predicates over the database in addition to the PIR query. While this work also does not have a database commitment, both this work and our work achieve *extractability* of the database used to respond to the PIR query. Our work can be viewed as an intermediate protocol between the original security notion of verifying that the server is using some well-formed database and this stronger notion given by Ben-David, Kalia, and Paneth. Our protocol only verifies the consistency of the PIR queries (rather than more complex predicates), but it does achieve the strong security property of database extractability. In addition, we leverage the stateful properties of our protocol to prove that all responses are consistent with a *single* database, which is a stronger requirement than all prior works. In other words, once the one-time protocol is complete, the only responses that will pass verification must all be consistent with a fixed, well-formed database.

Authenticated PIR. A very recent work of Colombo, Nikitin, Corrigan-Gibbs, Wu, and Ford [CNCG+23] constructs a slightly weaker primitive called *authenticated* PIR (APIR). The single-server protocols in this work are also based on the SimplePIR protocol [HHCG+23]. In particular, these protocols are also stateful, where the server sends an initial, one-time digest of the database to each client. While both authenticated PIR and our verifiable PIR protocol use an initial database digest, authenticated PIR differs from the notion of verifiable PIR in that it assumes the initial digest sent by the server is well-formed. This assumption is crucial in their security proof. The authentication method for the APIR queries closely resembles a plaintext MAC. This introduces significant overhead since each plaintext bit is mapped to roughly λ (security parameter) bits in the encoding, and the defense against selective-failure attacks requires error-correcting codes implemented on top of the PIR query. In a somewhat unintuitive result, our work achieves far superior performance over the APIR protocols by relying on *stronger* security. As we discuss below, only responses that are *exactly* correct will pass verification with a probability greater than $2^{-\lambda}$. This allows us to retain the efficient database packing schemes from SimplePIR while also removing the honest digest assumption. We discuss in section 6 how this assumption can be reintroduced as a performance optimization.

1.2 Our Contributions

We construct and implement VeriSimplePIR, the first efficient, verifiable PIR scheme, achieving industrial-strength security without compromising state-of-the-art performance.

Morally, the security property we achieve is the strongest one could hope for in any (nontrivial) PIR scheme. The initial database digest is accompanied by a noninteractive proof-of-knowledge of some fixed database, and then every response is verified against this proof to ensure that it is consistent with the committed database. This forces all responses with respect to a given digest to be consistent with the database that is fixed before the client performs the first query. In addition, this initial proof allows us to remove the assumption that the digest is well-formed, since this is directly verified by the client at the start of the protocol.

This verifiability comes at almost no performance overhead as long as the server remains honest. This is due to our novel *preprocessing* protocol that the client runs once with the server when it downloads the initial database digest. The result of this preprocessing protocol is a single noninteractive proof that allows the client to verify an *arbitrary* number of query-response pairs. The soundness of the verification is maintained as long as the server does not attempt to learn information about the private randomness held by the client that is used for verification. Here, we leverage the very strict property of this proof system, where only the response ciphertext that is *exactly* correct will pass verification. Therefore, the server cannot deviate from the honest protocol and pass verification with a non-negligible probability. If the server is running the honest protocol, then there is no leakage from the client's verification accepting (our verification algorithm has perfect completeness), so the client can safely *reuse* the preprocessed proof for any number of queries. In other words, as long as the server remains honest, there is essentially *no overhead* from adding verification to the SimplePIR protocol.

In particular, the concrete communication overhead for the online phase is between 13% and 50% over SimplePIR, depending on database sizes and whether or not the honest-digest assumption is in place. The computation overhead for VeriSimplePIR is at most 40% if the honest digest assumption is not in place. More details are given in section 6.

Roadmap. We define a verifiable PIR scheme with a preprocessed proof in section 3, then define our new VLHE scheme in section 4. The VeriSimplePIR construction is given in section 5 as it follows immediately from the VLHE construction. We present the full implementation and compare it to the semi-honest SimplePIR in section 6. To demonstrate practicality, in section 7 we discuss applying VeriSimplePIR to an important industry application: secure password checking.

2 Preliminaries

Notation. For $m \in \mathbb{N}$, define $[m] := \{1, \ldots, m\}$. For $i \in [m]$, let $\mathbf{b}_i \in \mathbb{Z}_q^m$ be the vector of all zeros with 1 in the i^{th} position. For $\mathbf{M} \in \mathbb{Z}_q^{n \times m}$, let $\mathbf{M}[i]$ denote the i^{th} row of $\mathbf{M}$ and let $\mathbf{m}_j$ denote the j^{th} column of $\mathbf{M}$. Similarly, for a vector $\mathbf{m} \in \mathbb{Z}_q^n$, let $\mathbf{m}[i]$ denote the i^{th} element of $\mathbf{m}$. For a finite set $\mathcal{S}$, let $x \xleftarrow{\$} \mathcal{S}$ denote sampling an element x uniformly at random from $\mathcal{S}$.

2.1 Short Integer Solution Problem

We review the Short Integer Solution (SIS) problem [Ajt96] on which the verifiability of our scheme is based.

Definition 2.1 (Short Integer Solution Problem)**.** *The statement of short integer solution problem $\mathsf{SIS}_{n,m,q,\beta}$ is the following: Given a random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, find a vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} = \mathbf{0} \pmod q$ and $0 < ||\mathbf{x}||_\infty \leq \beta$.*

Note that we use ℓ_∞-norm in the definition instead of ℓ_2-norm.

For the concrete hardness of $\mathsf{SIS}_{n,m,q,\beta}$, we follow [Lyu12] and use the analysis of [GN08, MR09] to choose parameters. Based on experiments of [GN08], Micciancio-Regev [MR09] analyzed that the shortest solution to $\mathbf{Ax} = \mathbf{0} \pmod q$ one can obtain by lattice reduction algorithms is of length (up to ℓ_2-norm) roughly $\min(q, 2^{2\sqrt{n \log q \log \delta}})$. Here δ is the root Hermite factor which depends on the output quality of the lattice reduction algorithm being used. We set parameters to satisfy the following condition to ensure the concrete hardness of $\mathsf{SIS}_{n,m,q,\beta}$ (up to ℓ_∞-norm).

$$\beta\sqrt{m} < \min\left(q, 2^{2\sqrt{n \log q \log \delta}}\right) \tag{1}$$

In this paper, we set $\delta = 1.005$ when choosing parameters. We note that the current best lattice reduction algorithms achieve around $\delta = 1.01$ and [CN11] conjectured that $\delta = 1.005$ is *totally out of reach* with thorough experiments.

2.2 Extractable SIS Commitments

We review a statistically extractable SIS-based commitment scheme. The scheme is described in fig. 1. It is essentially the amortized zero-knowledge proof of knowledge from [BBC+18] but adapted to our setting where zero-knowledge is not needed.

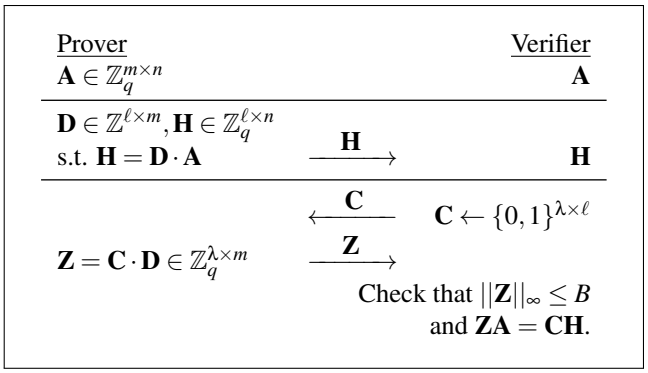

Figure 1: Extractable SIS-based Commitments [BBC+18]. The prover provides a commitment $\mathbf{H}$ to the verifier of some short matrix $\mathbf{D}$. The prover then provides $\mathbf{Z}$ in response to a challenge $\mathbf{C}$. $\mathbf{Z}$ is a proof of knowledge of some $\mathbf{D}'$ such that $||\mathbf{D}'||_\infty \leq 2B$ and $\mathbf{D}'\mathbf{A} = \mathbf{H}$.

Lemma 2.1 (Completeness)**.** *Let $\mathcal{P}$ be an honest prover (in fig. 1) who follows the protocol with respect to some $\mathbf{D}$ such that $||\mathbf{D}||_\infty \leq B'$ with $\ell \cdot B' \leq B$. Then, $\mathcal{P}$ always passes the verification.*

Proof. $||\mathbf{Z}||_\infty \leq \ell \cdot ||\mathbf{D}||_\infty \leq B$ and $\mathbf{ZA} = \mathbf{CDA} = \mathbf{CH}$. □

Lemma 2.2 (Extractability [BBC+18, Lemma 3])**.** *Let $\mathcal{P}$ be a prover (in fig. 1) who succeeds with probability $\varepsilon > 2^{-\lambda+2}$ over his random tape and the challenge choice $\mathbf{C} \leftarrow \{0,1\}^{\lambda \times \ell}$. Then, there exists a knowledge extractor $\mathcal{E}$ which makes $O(\ell \log \ell / \varepsilon)$ calls to $\mathcal{P}$ to extract a witness $\mathbf{D} \in \mathbb{Z}^{\ell \times m}$ such that $\mathbf{DA} = \mathbf{H}$ and $||\mathbf{D}||_\infty \leq 2B$ with overwhelming probability.*

We note that this extractability does not rely on the hardness of SIS; a matrix $\mathbf{D}$ such that $\mathbf{DA} = \mathbf{H}$ can be extracted from any prover that completes the protocol in fig. 1 with non-negligible probability. The hardness of SIS is used to guarantee the *computational uniqueness* of this matrix $\mathbf{D}$ (lemma 2.3). We refer the reader to [BBC+18, Lemma 3] for the full proof of lemma 2.2. Instead, we give a brief description of an extractor, since it will be relevant to the following lemmas.

Extractor for the SIS Commitment. We focus on extracting the i-th row of some $\mathbf{D}$ such that $\mathbf{D}[i] \cdot \mathbf{A} = \mathbf{H}[i]$ and $||\mathbf{D}||_\infty \leq 2B$. The idea of the extractor is to obtain valid responses $\mathbf{Z}$ and $\mathbf{Z}'$ to two challenges $\mathbf{C}$ and $\mathbf{C}'$ such that $\forall j \neq i$, $\mathbf{c}_j = \mathbf{c}'_j$ and $\mathbf{c}_i \neq \mathbf{c}'_i$. Since the responses are valid, we have the sum of the following outer products:

$$\mathbf{ZA} = \mathbf{CH} = \sum_{j=1}^{\ell} \mathbf{c}_j \cdot \mathbf{H}[j] \quad \text{and} \quad \mathbf{Z}'\mathbf{A} = \mathbf{C}'\mathbf{H} = \sum_{j=1}^{\ell} \mathbf{c}'_j \cdot \mathbf{H}[j].$$

Taking the difference gives $(\mathbf{Z} - \mathbf{Z}')\mathbf{A} = \sum_{j=1}^{\ell} (\mathbf{c}_j - \mathbf{c}'_j) \cdot \mathbf{H}[j] = (\mathbf{c}_i - \mathbf{c}'_i) \cdot \mathbf{H}[i]$. Consider an index $k \in [\lambda]$ such that

$\mathbf{c}_i[k] \neq \mathbf{c}'_i[k]$ and assume that $\mathbf{c}_i[k] - \mathbf{c}'_i[k] = 1$ without loss of generality. Then, if we consider the k-th row in the previous equality, we have $(\mathbf{Z}[k] - \mathbf{Z}'[k])\mathbf{A} = \mathbf{H}[i]$. Since $\mathbf{Z}$ and $\mathbf{Z}'$ passed verification, $||\mathbf{Z}[k] - \mathbf{Z}'[k]||_\infty \leq 2B$ holds. Thus, we can output $(\mathbf{Z}[k] - \mathbf{Z}'[k])$ as the i-th row of $\mathbf{D}$.

Binding Commitments from SIS Hardness. The hardness of SIS combined with the statistical extractability makes the protocol in fig. 1 a computationally binding commitment.

Lemma 2.3 (SIS Commitment Binding)**.** *Let $\mathcal{P}$ be a computationally bounded prover in fig. 1. Consider the extractor $\mathcal{E}$ guaranteed by lemma 2.2 running on $\mathcal{P}$. Either (i) the extractor $\mathcal{E}$ outputs a unique matrix $\mathbf{D}$ with all but negligible probability, or (ii) the prover $\mathcal{P}$ can be used to solve $\mathsf{SIS}_{n,m,q,\beta}$ for $\beta = 4B$.*

Proof. Suppose condition (i) does not hold. Then, we can run $\mathcal{E}$ polynomially many times to obtain two matrices $\mathbf{D} \neq \mathbf{D}'$ such that $||\mathbf{D}||_\infty, ||\mathbf{D}'||_\infty \leq 2B$ and $\mathbf{H} = \mathbf{D}\mathbf{A} = \mathbf{D}'\mathbf{A}$ by lemma 2.2. This gives a solution for $\mathsf{SIS}_{n,m,q,\beta}$ for $\beta = 4B$ of the form $(\mathbf{D} - \mathbf{D}')\mathbf{A} = 0$, since $\mathbf{D} - \mathbf{D}' \neq 0$. □

Let B' be the honest bound on the matrix $\mathbf{D}$; $B' \geq ||\mathbf{D}||_\infty$. Then, the correctness bound on $||\mathbf{Z}||_\infty$ is $B \geq \ell \cdot B'$. The extractability of the commitment scheme in fig. 1 guarantees that we can extract an SIS solution of size at most $4B$ (by taking the difference of extracted databases with size at least $2B$), which means that we need SIS to be hard for $4B$ in order to ensure computational binding. Setting $\beta = 4B = 4\ell B'$, we require that $4\ell B'\sqrt{m} \leq \min\left(q, 2^{2\sqrt{n \log q \log \delta}}\right)$.

A Useful Statistical Lemma. We prove a lemma on the probability of a vector being in the nullspace of $\mathbf{C} \xleftarrow{\$} \{0,1\}^{\lambda \times \ell}$.

Lemma 2.4 (Guessing an Element in the Nullspace)**.** *Let $\ell, \lambda \in \mathbb{N}$ and $p \geq 2$. For all $\mathbf{x} \in \mathbb{Z}_p^\ell$ such that $\mathbf{x} \neq \mathbf{0}$, the following holds:*

$$\Pr_{\mathbf{C} \xleftarrow{\$} \{0,1\}^{\lambda \times \ell}} [\, \mathbf{C} \cdot \mathbf{x} = \mathbf{0} \,] \leq 2^{-\lambda}.$$

Proof. Since $\mathbf{x} \neq \mathbf{0}$, there exists some $i \in [\ell]$ such that $\mathbf{x}[i] \neq 0$. Consider two vectors $\mathbf{c}_0, \mathbf{c}_1 \in \{0,1\}^\ell$ which are identical except the i-th bits: $\mathbf{c}_0[i] = 0$ and $\mathbf{c}_1[i] = 1$. Then, $\langle \mathbf{c}_0, \mathbf{x} \rangle$ and $\langle \mathbf{c}_1, \mathbf{x} \rangle$ cannot be simultaneously zero, since $\langle \mathbf{c}_1, \mathbf{x} \rangle - \langle \mathbf{c}_0, \mathbf{x} \rangle = \mathbf{x}[i] \neq 0$. Thus, $\Pr[\, \langle \mathbf{c}, \mathbf{x} \rangle = 0 \,] \leq 1/2$ holds, where the probability is over $\mathbf{c} \xleftarrow{\$} \{0,1\}^\ell$. Since λ rows of $\mathbf{C}$ are sampled independently from $\{0,1\}^\ell$, the lemma follows. □

Fiat-Shamir Transform. Below, we will use that fact that fig. 1 is a public-coin proof of knowledge with negligible soundness error, which allows us to apply the Fiat-Shamir transform [FS87] to generically make this protocol non-interactive in the random oracle model [PS96]. Concretely, the prover generates the challenge $\mathbf{C} \leftarrow \mathsf{Hash}(\mathbf{A}, \mathbf{H})$ and then computes the response $\mathbf{Z} \leftarrow \mathbf{C}\mathbf{D}$, where the hash function Hash is modeled as a random oracle.

2.3 Learning with Errors

The security of our PIR scheme is based on the decisional Learning with Errors (LWE) problem [Reg09]. Consider a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, a short *error* vector $\mathbf{e} \leftarrow \chi^m$ such that $\mathbf{e} \in \mathbb{Z}_q^m$, and uniformly random vectors $\mathbf{s} \in \mathbb{Z}_q^n$. Note that χ is an *error* distribution over $\mathbb{Z}_q$, typically a zero-centered discrete Gaussian modulo q. The LWE problem is to distinguish the distribution of $(\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e})$ from the distribution of $(\mathbf{A}, \mathbf{r})$, where $\mathbf{r}$ is uniformly sampled in $\mathbb{Z}_q^m$. More formally, we say that the (n, q, χ)-LWE problem with m samples is (T, ε)-hard if no adversary running in time T can distinguish these distributions with probability better than ε.

Linearly Homomorphic Encryption. We briefly define the linearly homomorphic encryption (LHE) scheme from LWE given by Regev [Reg09]. We use the same "hoisted" variant as in Henzinger et al. [HHCG+23], where the matrix $\mathbf{A}$ is reused across multiple ciphertexts but the secret $\mathbf{s}$ is sampled independently for each ciphertext, allowing us to preprocess data dependent on the matrix $\mathbf{A}$.

The plaintext space of the LHE scheme is vectors over $\mathbb{Z}_p$. Let $\mu \in \mathbb{Z}_p^m$ be a plaintext. To encrypt μ, sample a uniform matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, a uniform secret key $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and an error vector $\mathbf{e} \leftarrow \chi^m$. The ciphertext is $(\mathbf{A}, \mathbf{A}\mathbf{s} + \Delta\mu + \mathbf{e}) \in \mathbb{Z}_q^{m \times (n+1)}$, where $\Delta := \lfloor q/p \rfloor$. To decrypt a ciphertext $(\mathbf{A}, \mathbf{u}) \in \mathbb{Z}_q^{m \times (n+1)}$, compute $\mathbf{A}\mathbf{s}$ and output $\lfloor (\mathbf{u} - \mathbf{A}\mathbf{s})/\Delta \rfloor \in \mathbb{Z}_p^m$. The correctness of this decryption holds as long as $\Delta/2 > ||\mathbf{e}||_\infty$.

To see why this scheme is linearly homomorphic, consider a linear transformation $\mathbf{D} \in \mathbb{Z}^{\ell \times m}$. We can homomorphically compute $\mathbf{D}$ over an encryption of a vector μ. Consider the ciphertext $\mathsf{ct} = (\mathbf{A}, \mathbf{A}\mathbf{s} + \Delta\mu + \mathbf{e})$. Observe that the product $\mathbf{D} \cdot \mathsf{ct} = (\mathbf{D}\mathbf{A}, \mathbf{D}\mathbf{A}\mathbf{s} + \Delta\mathbf{D}\mu + \mathbf{D}\mathbf{e}) \in \mathbb{Z}_q^{\ell \times (n+1)}$ is a well-formed encryption of $\mathbf{D}\mu$. Decryption correctness holds as long as $\Delta/2 > ||\mathbf{D}\mathbf{e}||_\infty$.

LHE Correctness. We now discuss the constraints on the LWE parameters to ensure correct decryption with good probability after a homomorphic linear transformation. We follow the analysis of Henzinger et al. (see e.g. [HHCG+23, Theorem C.1]). We will make use of the following helpful lemma. Here, D_s denotes the discrete Gaussian distribution over $\mathbb{Z}$ with variance $s^2/2\pi$.

Lemma 2.5 ([LP11, Lemma 2.2], [Ban95, Lemma 2.4])**.** *For any real $s > 0$ and $T > 0$, and any $\mathbf{x} \in \mathbb{R}^n$, we have*

$$\Pr_{\mathbf{e} \leftarrow D_s^m} \left[|\langle \mathbf{x}, \mathbf{e} \rangle| \geq T \cdot s ||\mathbf{x}||_2 \right] < 2\exp(-\pi \cdot T^2).$$

For some $\mathbf{D} \in \mathbb{Z}^{\ell \times m}$, consider the decryption of the ciphertext $(\mathbf{A}, \mathbf{A}\mathbf{s} + \Delta\mu + \mathbf{D}\mathbf{e}) \in \mathbb{Z}_q^{m \times (n+1)}$, where $\mathbf{e}$ is a fresh sample from D_s^m. As noted above, we need $\Delta/2 > ||\mathbf{D}\mathbf{e}||_\infty$ to decryption correctly. Regarding $||\mathbf{D}\mathbf{e}||_\infty$, we apply lemma 2.5 to each row of $\mathbf{D}$. Note that we can bound the ℓ_2 norm of any row of $\mathbf{D}$ by $\sqrt{m} \cdot ||\mathbf{D}||_\infty$. Then, setting $T = \Delta/(s\sqrt{m} \cdot ||\mathbf{D}||_\infty)$,

we have that $\Pr[||\mathbf{De}||_\infty \geq \Delta/2] < \delta$, where

$$\delta = 2\exp\left(-\pi\left(\frac{\Delta}{s\sqrt{m}\cdot||\mathbf{D}||_\infty}\right)^2\right).$$

Equivalently, we have decryption failure probability of at most δ if

$$\Delta \geq s\sqrt{m}\cdot||\mathbf{D}||_\infty\cdot\sqrt{\frac{\ln(2/\delta)}{\pi}} = \sigma\cdot||\mathbf{D}||_\infty\sqrt{2m\ln(2/\delta)}.$$

Here, σ denotes the standard deviation of D_s, i.e. $\sigma^2 = s^2/2\pi$. In summary, this gives us a lower bound on q as

$$q \geq p\cdot\sigma\cdot||\mathbf{D}||_\infty\sqrt{2m\ln(2/\delta)}. \tag{2}$$

This saves us roughly a $\sqrt{m}$ factor over the naïve worst-case analysis.

2.4 Semi-honest PIR

We begin by giving the API for PIR with a preprocessed database. These core algorithms will be identical to those used in the preprocessed verifiable PIR below. The database elements are treated as elements of $\mathbb{Z}_t$, where t is some integer modulus. The parameters for this primitive are a modulus t, a database length N, and a security parameter λ. The algorithms are described below, and the usage of the API is given in fig. 2

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N, t)$
  Outputs public parameters pp.
- $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pp}, \mathbf{D} \in \mathbb{Z}_t^N)$
  Takes in a database $\mathbf{D}$ of size N of elements of $\mathbb{Z}_t$ and outputs a *short* digest d (i.e. $|\mathsf{d}| \ll |\mathbf{D}|$).
- $\mathsf{q}, \mathsf{st} \leftarrow \mathsf{Query}(\mathsf{pp}, i \in [N])$
  Takes in an index $i \in [N]$ and produces a query q with a query state st.
- $\mathsf{a} \leftarrow \mathsf{Answer}(\mathsf{pp}, \mathbf{D}, \mathsf{q})$
  Takes in a database $\mathbf{D}$ and a query q and produces an answer a.
- $\mathsf{r} \leftarrow \mathsf{Recover}(\mathsf{pp}, \mathsf{st}, \mathsf{a}, \mathsf{d})$
  Takes in a query state st generated from an index $i \in [N]$, an answer a, and a digest d of $\mathbf{D}$. Outputs a database element r. If inputs are honestly generated, the output r should be $\mathbf{D}[i]$. This is formalized below (definition 2.2).

We now define basic correctness and security requirments for the semi-honest PIR primitive.

Definition 2.2 (Correctness)**.** *We say that a PIR scheme has* correctness error δ *if, for security parameter λ, for all databases $\mathbf{D} \in \mathbb{Z}_t^N$ and for all indices $i \in [N]$,*

$$\Pr\left[\begin{array}{r|l} \mathsf{Recover}(\mathsf{st},\mathsf{a},\mathsf{d}) & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N, t) \\ = \mathbf{D}[i] & \mathsf{d} \leftarrow \mathsf{Digest}(\mathbf{D}) \\ & \mathsf{q}, \mathsf{st} \leftarrow \mathsf{Query}(i) \\ & \mathsf{a} \leftarrow \mathsf{Answer}(\mathbf{D}, \mathsf{q}) \end{array}\right] \geq 1 - \delta.$$

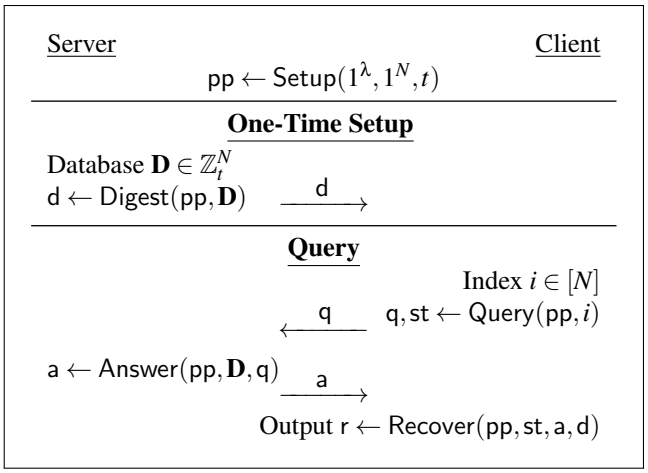

Figure 2: Usage of the Semi-honest PIR API.

The public parameters pp are implicit inputs in all algorithms following Setup.

Definition 2.3 (Query Hiding)**.** *We say that a PIR scheme is (T, ε)-*Query Hiding *if, for all adversaries $\mathcal{A}$ running in time at most T, on database size N and for all $i, j \in [N]$,*

$$\Big|\Pr[\mathcal{A}(\mathsf{pp},\mathsf{q}) = 1 : (\mathsf{q},\mathsf{st}) \leftarrow \mathsf{Query}(\mathsf{pp}, i)] - \Pr[\mathcal{A}(\mathsf{pp},\mathsf{q}) = 1 : (\mathsf{q},\mathsf{st}) \leftarrow \mathsf{Query}(\mathsf{pp}, j)]\Big| \leq \varepsilon$$

Remark 2.1 (Selective Failures & Query Hiding)**.** *We note that definition 2.3 is insufficient when considering malicious adversaries, since selective failure attacks could compromise query hiding even when the query message alone reveals nothing about the client's desired index.*

2.4.1 SimplePIR

We give the semi-honest PIR protocol, denoted SimplePIR, from [HHCG+23] below. The API usage is given in fig. 2. Parsing the public parameters pp as $\mathbf{A}$ and the digest d as $\mathbf{H}$ is implicit. Note that the database here is represented as a matrix $\mathbf{D} \in \mathbb{Z}_p^{\ell\times m}$. We omit the details of encoding vectors in $\mathbb{Z}_t^N$ to matrices over $\mathbb{Z}_p$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N, t)$
  Output a uniform $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ as pp.
- $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pp}, \mathbf{D})$
  Compute $\mathbf{H} \leftarrow \mathbf{DA} \in \mathbb{Z}_q^{\ell\times n}$. Output $\mathsf{d} \leftarrow \mathbf{H}$.
- $\mathsf{q}, \mathsf{st} \leftarrow \mathsf{Query}(\mathsf{pp}, i \in [N])$
  Write i as $(i_r, i_c) \in [\ell] \times [m]$. Sample a uniform secret $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and an error vector $\mathbf{e} \leftarrow \chi^m$. Compute $\mathbf{u} \leftarrow \mathbf{As} + \mathbf{e} + \lfloor q/p \rfloor \mathbf{b}_{i_c}$. Output $(\mathsf{q}, \mathsf{st}) \leftarrow (\mathbf{u}, (\mathbf{s}, i))$.
- $\mathsf{a} \leftarrow \mathsf{Answer}(\mathsf{pp}, \mathbf{D}, \mathsf{q})$
  Parse q as $\mathbf{u} \in \mathbb{Z}_q^m$. Output $\mathbf{v} \leftarrow \mathbf{Du}$ as a.

- $\mathsf{r} \leftarrow \mathsf{Recover}(\mathsf{pp}, \mathsf{st}, \mathsf{a}, \mathsf{d})$
  Parse $(\mathbf{s}, i) \leftarrow \mathsf{st}$ and $\mathbf{v} \leftarrow \mathsf{a}$. Write i as $(i_\mathrm{r}, i_\mathrm{c}) \in [\ell] \times [m]$. Compute $\hat{\mathsf{r}} \leftarrow \mathbf{v}[i_\mathrm{r}] - \langle \mathbf{H}[i_\mathrm{r}], \mathbf{s} \rangle \in \mathbb{Z}_q$. Output $\mathsf{r} \leftarrow \lfloor p \cdot \hat{\mathsf{r}}/q \rceil \in \mathbb{Z}_p$.

The constraints on the LWE parameters to ensure correctness of the protocol are given in eq. (2) in section 2.3.

3 Verifiable PIR Definitions

In this section, we define the algorithms that comprise the preprocessed verifiable PIR primitive as well as the correctness and security definitions that this primitive must satisfy.

As in the semi-honest definition, the database elements are treated as elements of $\mathbb{Z}_t$, where t is some integer modulus. The parameters for this primitive are a modulus t, a database length N, and a security parameter λ. The API consists of the following algorithms in addition to the algorithms in the semi-honest PIR API (see section 2.4). The usage of this API is described in fig. 3.

- $\mathsf{Accept}/\mathsf{Reject} \leftarrow \mathsf{DigVer}(\mathsf{pp}, \mathsf{d})$
  Takes in the public parameters and a digest and outputs Accept if the digest is well-formed with respect to some database. Outputs Reject otherwise.

- $\mathsf{q}_\pi, \mathsf{st}_\pi \leftarrow \mathsf{PrQry}(\mathsf{pp})$
  This is the proof query algorithm, where the client generates a request used by the server to precompute the proof for the query phase. The output is a proof query q_π and a proof query state st_π.

- $\mathsf{a}_\pi \leftarrow \mathsf{PrAns}(\mathsf{pp}, \mathbf{D}, \mathsf{d}, \mathsf{q}_\pi)$
  This is the proof answer algorithm that is run by the server to respond to a proof query. The output is returned to the client.

- $\pi/\perp \leftarrow \mathsf{PrRec}(\mathsf{pp}, \mathsf{d}, \mathsf{q}_\pi, \mathsf{st}_\pi, \mathsf{a}_\pi)$
  This is the final algorithm in the proof preprocessing phase, which is run by the client to verify the response from the server and output the proof. If the response does not verify, the algorithm outputs $\perp$. Otherwise, it outputs the proof π.

- $\mathsf{Accept}/\mathsf{Reject} \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{d}, \mathsf{q}, \mathsf{a}, \pi)$
  Takes in the digest d, a query q, an answer a, and a proof π. Outputs Accept if the answer a was correctly computed over the query q and the digest d.

At a high level, the goal of this API is to allow the client to query the server for a *single* proof that is then *reused* across many PIR queries. This proof can be safely reused as long as the verification does not fail. As soon as the verification fails, the client must rerun the proof preprocessing phase in order to maintain the integrity of the verification.

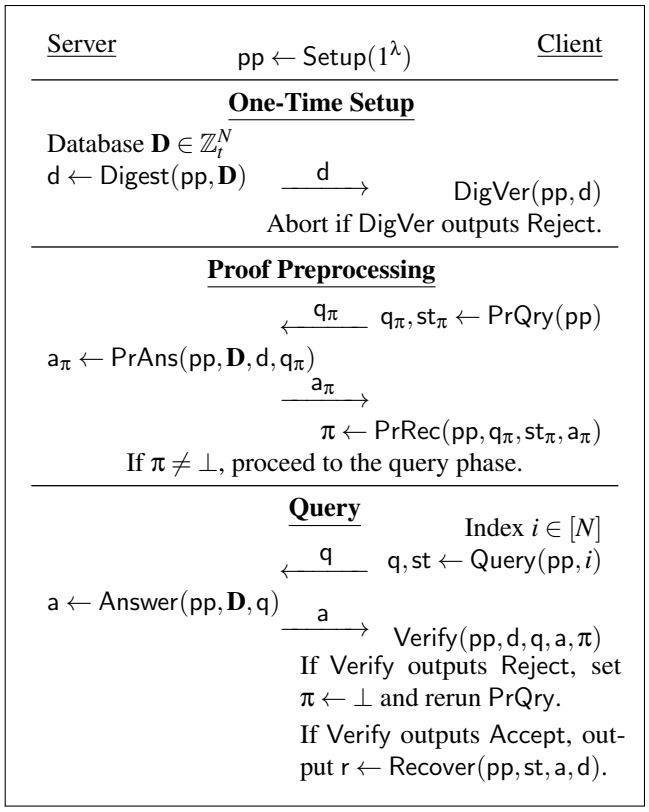

Figure 3: Usage of the Preprocessed Verifiable PIR API.

Security Definitions. We give an intuition for the security definitions of the vPIR primitive here. The full definitions are in appendix A.

- **Completeness.** Our verification procedures will have perfect completeness; messages produced by honest servers will always verify. This is in addition to the standard PIR correctness given in definition 2.2.

- **Digest Binding.** We require that the digest be a computationally binding commitment to the database $\mathbf{D}$. This will allow us to define honest behavior with respect to this fixed database as long as the initial DigVer check passes.

- **Soundness.** The soundness requirement for the verification procedure is very strict. Not only must the returned answer a give the correct output upon running Recover, but a must be the *unique* value defined by the query q and the fixed database $\mathbf{D}$. This uniqueness requirement is extended to the proof π. The only value π that will pass verification in the preprocessing phase is the exact value defined by pp, q_π, st_π, and the fixed database $\mathbf{D}$.

Reusable Proofs from Strict Soundness. The strict soundness requirements achieved by this primitive immediately imply that it is safe to reuse the proof. If the only answers that

the server can return are exactly the honest values, then there is no leakage on the proof from client passing verification, since due to the perfect completeness the honest messages will always pass verification. Furthermore, since the proof π does not depend on the query or answer, the safety of using this proof once extends to reusing this proof many times, since the server has no additional information about π once a query phase is complete.

4 Verifiable Linearly Homomorphic Encryption

In this section, we construct our verifiable linearly homomorphic encryption scheme, denoted VLHE. This scheme begins with the Regev LHE described in section 2.3. In this scheme, a ciphertext that encrypts a vector $\mu \in \mathbb{Z}_p^m$ has the structure

$$(\mathbf{A}, \mathbf{As} + \lfloor q/p \rfloor \mu + \mathbf{e}) = (\mathbf{A}, \mathbf{u}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^m.$$

This scheme supports the homomorphic evaluation of linear functions $\mathbf{D} \in \mathbb{Z}^{\ell \times m}$ that map $\mathbb{Z}_p^m$ to $\mathbb{Z}_p^\ell$. Note that we allow $\mathbf{D}$ to define linear maps between larger rings, e.g., $\mathbb{Z}_q^m \to \mathbb{Z}_q^\ell$ for $q > p$. The ciphertext that encrypts the output of the function $\mathbf{D}$ has the form

$$(\mathbf{DA}, \mathbf{DAs} + \lfloor q/p \rfloor \mathbf{D}\mu + \mathbf{De}) = (\mathbf{H}, \mathbf{v}) \in \mathbb{Z}_q^{\ell \times n} \times \mathbb{Z}_q^\ell,$$

where we defined $\mathbf{v} := \mathbf{Du} \in \mathbb{Z}_q^\ell$ and $\mathbf{H} := \mathbf{DA} \in \mathbb{Z}_q^{\ell \times n}$. At a high level, we use the fact that the matrix $\mathbf{H}$ is an extractable commitment to the matrix $\mathbf{D}$. Furthermore, we can use the extractable proof described in section 2.2 to efficiently prove knowledge of the function $\mathbf{D}$. Overall, the basic equation that we will use throughout this section is

$$\mathbf{D} \begin{bmatrix} \mathbf{A} & \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \mathbf{v} \end{bmatrix}, \tag{3}$$

which also has the form of an extractable SIS commitment described in section 2.2.

4.1 Verifiable LHE Construction

We now give the API for the (secret-key) VLHE primitive along with the construction based on the Regev LHE scheme [Reg09].

Construction 4.1 (Verifiable Linearly Homomorphic Encryption)**.** *The scheme is parametrized by a computational security parameter n, a statistical security parameter λ, a plaintext modulus p, a ciphertext modulus q, and an error distribution χ. The final parameter is a hash function* Hash *that is modeled as a random oracle. This hash function is used to apply the Fiat-Shamir transform [FS87] to the public-coin proof described in section 2.2 on the commitment defined in eq. (3).*

- $\mathsf{pp} \leftarrow \mathsf{VLHE.Setup}(1^n, 1^m, q)$
  Samples a uniformly random matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}$ and outputs $\mathsf{pp} \leftarrow \mathbf{A}$ as the public parameters.

- $\mathbf{H}, \mathbf{Z} \leftarrow \mathsf{VLHE.Commit}(\mathbf{A}, \mathbf{D} \in \mathbb{Z}_p^{\ell \times m})$
  Takes in a linear function $\mathbf{D}$ and computes $\mathbf{H} \leftarrow \mathbf{DA} \in \mathbb{Z}_q^{\ell \times n}$ as the commitment to $\mathbf{D}$. Compute $\mathbf{C} \in \{0,1\}^{\lambda \times \ell}$ as $\mathbf{C} = \mathsf{Hash}(\mathbf{A}, \mathbf{H})$ and set $\mathbf{Z} \leftarrow \mathbf{C} \cdot \mathbf{D}$. Output $\mathbf{H}, \mathbf{Z}$.

- $\mathsf{Accept}/\mathsf{Reject} \leftarrow \mathsf{VLHE.VerCom}(\mathbf{A}, \mathbf{H}, \mathbf{Z})$
  This verifies the initial commitment to the linear function. Check $||\mathbf{Z}||_\infty \leq \ell \cdot p$ and output Reject *if this check fails. Compute $\mathbf{C} \leftarrow \mathsf{Hash}(\mathbf{A}, \mathbf{H})$ exactly as in* Commit*. Check that $\mathbf{ZA} = \mathbf{CH}$. Output* Reject *if this check fails and* Accept *if this check passes.*

- $(\mathbf{u}, \mathbf{s}) \leftarrow \mathsf{VLHE.Encrypt}(\mathbf{A}, \mu \in \mathbb{Z}_p^m)$
  Sample a uniform secret $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and an error vector $\mathbf{e} \leftarrow \chi^m$. For $\Delta := \lfloor q/p \rfloor$, compute the ciphertext $\mathbf{u} := \mathbf{As} + \Delta\mu + \mathbf{e} \in \mathbb{Z}_q^m$. Note that this is identical to the Regev LHE encryption. Output $(\mathbf{u}, \mathbf{s})$.

- $\mathbf{v} \leftarrow \mathsf{VLHE.Eval}(\mathbf{D}, \mathbf{u})$
  This is homomorphic evaluation; output $\mathbf{v} \leftarrow \mathbf{Du}$.

- $\nu \leftarrow \mathsf{VLHE.Decrypt}(\mathbf{H} \in \mathbb{Z}_q^{\ell \times n}, \mathbf{s} \in \mathbb{Z}_q^n, \mathbf{v} \in \mathbb{Z}_q^\ell)$
  We define this operation for the outputs of homomorphic evaluation. This operation is identical to Regev decryption, where the output is $\nu \leftarrow \lfloor (\mathbf{v} - \mathbf{H} \cdot \mathbf{s})/\Delta \rfloor \in \mathbb{Z}_p^\ell$.

- $\mathbf{Z} \leftarrow \mathsf{VLHE.Prove}(\mathbf{A}, \mathbf{H}, \mathbf{u}, \mathbf{v}, \mathbf{D})$
  Compute $\mathbf{C} \in \{0,1\}^{\lambda \times \ell}$ as $\mathbf{C} \leftarrow \mathsf{Hash}(\mathbf{A}, \mathbf{H}, \mathbf{u}, \mathbf{v})$. Output $\mathbf{Z} \leftarrow \mathbf{CD}$.

- $\mathsf{Accept}/\mathsf{Reject} \leftarrow \mathsf{VLHE.Verify}(\mathbf{A}, \mathbf{H}, \mathbf{u}, \mathbf{v}, \mathbf{Z})$
  Check $||\mathbf{Z}||_\infty \leq \ell \cdot p$. Output Reject *if this check fails. Compute $\mathbf{C} \leftarrow \mathsf{Hash}(\mathbf{A}, \mathbf{H}, \mathbf{u}, \mathbf{v})$ exactly as in* Prove*. Check that $\mathbf{Z} \begin{bmatrix} \mathbf{A} & \mathbf{u} \end{bmatrix} = \mathbf{C} \begin{bmatrix} \mathbf{H} & \mathbf{v} \end{bmatrix}$. Output* Reject *if this check fails and* Accept *if this check passes.*

Observe that Verify performs the exact checks that correspond to verifying an opening of the commitment defined in eq. (3).

4.1.1 Verifiable LHE Correctness & Security

We first prove the security of the verification, then show how to obtain correctness for all (even malicious) functions that pass verification.

Security. Semantic security follows directly from the security of the Regev encryption scheme and the hardness of LWE. Below, we give the constraint parameters that guarantees correct decryption as long as VLHE verification passes.

We can now focus on the security of the proof verification. We obtain a very strong security guarantee, which is that only *one* ciphertext (the correct ciphertext) will pass verification. We focus our security proofs on the interactive variant of the verification given in fig. 5, and the security of construction 4.1 follows by applying the Fiat-Shamir heuristic [FS87].

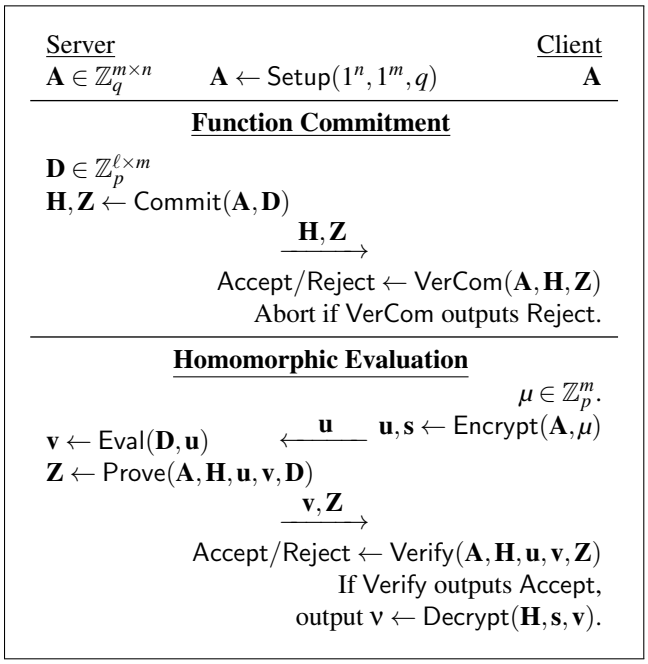

Figure 4: Usage of the VLHE API in construction 4.1.

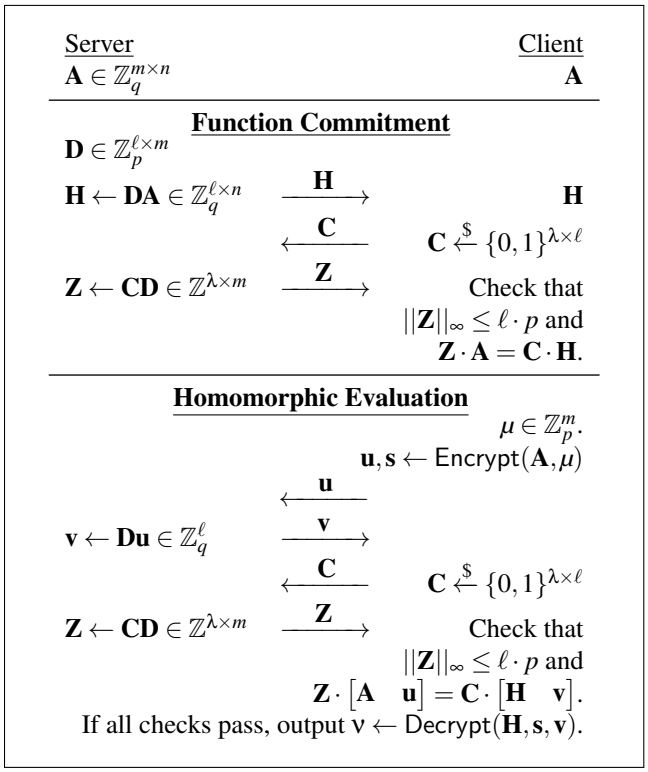

Figure 5: Variant of the VLHE construction in that uses interactive commitment openings. Construction 4.1 can be derived by applying the Fiat-Shamir heuristic to this protocol.

Lemma 4.1 (Verification Uniqueness)**.** *Let $\mathcal{P}$ be a computationally bounded server (prover) in fig. 5, and let $\mathbf{H}$ be the matrix sent by $\mathcal{P}$ in the function commitment phase. Consider the extractor $\mathcal{E}$ from lemma 2.2 running on $\mathcal{P}$ in the function commitment phase. Let $\mathbf{D} \in \mathbb{Z}^{\ell \times m}$ be the unique matrix that can be extracted from $\mathcal{E}$, under hardness assumption of $\mathsf{SIS}_{n,m,q,\beta}$ for $\beta = 4\ell p$ (lemma 2.3). Consider a ciphertext $\mathbf{u}$ as in fig. 5 and a response $\tilde{\mathbf{v}}$. Assuming the hardness of $\mathsf{SIS}_{n,m,q,\beta}$ for $\beta = 4\ell p$, $\tilde{\mathbf{v}} = \mathbf{Du}$ must hold.*

Proof. Assume for contradiction that the response ciphertext $\tilde{\mathbf{v}} \neq \mathbf{Du}$, but the server is able to produce some $\widetilde{\mathbf{Z}}$ such that $||\widetilde{\mathbf{Z}}||_\infty \leq \ell p$ and $\widetilde{\mathbf{Z}} \cdot \begin{bmatrix} \mathbf{A} & \mathbf{u} \end{bmatrix} = \mathbf{C} \cdot \begin{bmatrix} \mathbf{H} & \tilde{\mathbf{v}} \end{bmatrix}$. We can invoke lemma 2.2 on the commitment defined by $\begin{bmatrix} \mathbf{A} & \mathbf{u} \end{bmatrix}$ and $\begin{bmatrix} \mathbf{H} & \tilde{\mathbf{v}} \end{bmatrix}$ to extract some database $\mathbf{D}'$ such that $||\mathbf{D}'||_\infty \leq 2\ell p$ and $\mathbf{D}' \begin{bmatrix} \mathbf{A} & \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \tilde{\mathbf{v}} \end{bmatrix}$. If $\mathbf{D}' \neq \mathbf{D}$, then we have $(\mathbf{D}' - \mathbf{D})\mathbf{A} = \mathbf{0}$, where $\mathbf{D}' - \mathbf{D} \neq \mathbf{0}$. This is an SIS solution for $\mathbf{A}$ of magnitude $||\mathbf{D}' - \mathbf{D}||_\infty \leq 4\ell p$. This contradicts the hardness of $\mathsf{SIS}_{n,m,q,\beta}$ for $\beta = 4\ell p$, so we must have $\mathbf{D}' = \mathbf{D}$. Therefore, $\tilde{\mathbf{v}} = \mathbf{Du}$. □

Correctness. In the original Regev scheme, correctness was only guaranteed for a function $\mathbf{D} \in \mathbb{Z}_p^{\ell \times m}$. The correctness requirement for this scheme must account for the slack between the honest entry-size p and the bound guaranteed by extractability, which is $||\mathbf{D}||_\infty \leq 2\ell p$ (recall lemma 2.2). By slightly increasing the ciphertext modulus, we can account for any matrix $\mathbf{D} \in \mathbb{Z}_{2\ell p}^{\ell \times m}$, which allows us to guarantee that if verification passes then decryption will only fail with negligible probability. We can use this bound in eq. (2) to obtain the following correctness requirement for the VLHE scheme:

$$q \geq \sigma \cdot 2\ell p^2 \sqrt{2m \ln(2/\delta)}. \tag{4}$$

4.1.2 Batch Verification of Output Ciphertexts

We briefly note that the proofs in construction 4.1 can efficiently verify a batch of input-output pairs. More formally, we batch-verify t input-output ciphertext pairs $(\mathbf{u}_i, \mathbf{v}_i) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^\ell$ by first defining the matrices

$$\mathbf{U} := \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_t \end{bmatrix} \in \mathbb{Z}_q^{m \times t}$$
$$\mathbf{V} := \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_t \end{bmatrix} \in \mathbb{Z}_q^{\ell \times t}.$$

We then observe that the server's computation across all t pairs can be viewed as an SIS commitment $\mathbf{D} \cdot \begin{bmatrix} \mathbf{A} & \mathbf{U} \end{bmatrix} = \begin{bmatrix} \mathbf{H} & \mathbf{V} \end{bmatrix}$ which can be verified using the relation

$$\mathbf{Z} \cdot \begin{bmatrix} \mathbf{A} & \mathbf{U} \end{bmatrix} = \mathbf{C} \cdot \begin{bmatrix} \mathbf{H} & \mathbf{V} \end{bmatrix}. \tag{5}$$

Observe that the matrix $\mathbf{Z}$ in eq. (5) is identical to the $\mathbf{Z}$ matrix in eq. (3). In addition, the $\mathbf{C}$ matrices in both equations are also the identically distributed, which results in identical norm bound on the $\mathbf{Z}$ matrices. In fact, the only piece of the BatchProve and BatchVerify algorithms that grow with t is the computation of the hash function Hash on the t tuples and, of course, the matrix multiplication itself, which now

has $n+t$ columns rather than $n+1$ columns. The algorithms are give formally below.

- $\mathbf{Z} \leftarrow \mathsf{VLHE.BatchProve}(\mathbf{A},\mathbf{H},\mathbf{U},\mathbf{V},\mathbf{D})$
  Compute $\mathbf{C} \leftarrow \mathsf{Hash}(\mathbf{A},\mathbf{H},\mathbf{U},\mathbf{V})$ where $\mathbf{C} \in \{0,1\}^{\lambda \times \ell}$. Compute $\mathbf{Z} \leftarrow \mathbf{CD} \in \mathbb{Z}^{\lambda \times m}$ and output $\mathbf{Z}$.

- $\mathsf{Accept/Reject} \leftarrow \mathsf{VLHE.BatchVerify}(\mathbf{A},\mathbf{H},\mathbf{U},\mathbf{V},\mathbf{Z})$
  Compute $\mathbf{C} \leftarrow \mathsf{Hash}(\mathbf{A},\mathbf{H},\mathbf{U},\mathbf{V})$ exactly as in BatchProve. Check that $||\mathbf{Z}||_\infty \leq \ell \cdot p$ and that $\mathbf{Z} \cdot \begin{bmatrix}\mathbf{A} & \mathbf{U}\end{bmatrix} = \mathbf{C} \cdot \begin{bmatrix}\mathbf{H} & \mathbf{V}\end{bmatrix}$. If all the checks pass, output Accept. Otherwise, output Reject.

The security of these functions follows an identical argument to the proof of lemma 4.1.

4.2 Reusable VLHE Proof

We present a useful protocol to compute a *reusable* VLHE proof. When a fixed linear function is applied to many encrypted vectors, this reusable proof a prevents repeatedly computing and sending a fresh $\mathbf{Z}$ for every response ciphertext. Looking ahead, our final VeriSimplePIR protocol leverages this proof preprocessing protocol to allow verifiability at no online cost in SimplePIR.

Our proof preprocessing protocol combines two observations. The first is that the proof $\mathbf{Z} = \mathbf{CD}$ does not directly depend on any other aspect of the protocol beyond the matrix $\mathbf{D}$. This allows us to precompute this proof before the input ciphertext is given. A caveat here is that the proof $\mathbf{Z}$ must be computed while keeping the client's challenge $\mathbf{C}$ secret from the server. Otherwise, the server might use its knowledge of the challenge for malicious actions in homomorphic evaluation. Meanwhile, if the proof is computed while keeping the challenge secret, we can even reuse the preprocessed proof across several queries as long as C is not leaked.

The second observation is that the proof $\mathbf{Z} = \mathbf{CD}$ is, in fact, computed as a *linear* function evaluation defined by $\mathbf{D}$. That is, we can use construction 4.1 on the matrix $\mathbf{D}^T$ where the ciphertexts encrypt the rows of the challenge $\mathbf{C}$. This allows the server to *verifiably* compute $\mathbf{D}^T\mathbf{C}^T = \mathbf{Z}^T$ for the client while not knowing the challenge $\mathbf{C}$.

The full proof preprocessing protocol, along with the corresponding verifiable homomorphic evaluation, is given in fig. 6. In the construction, two commitments, $\mathbf{H}_1$ and $\mathbf{H}_2$, are used. The first commitment $\mathbf{H}_1$ plays the same role as the commitment $\mathbf{H}$ in the original VLHE construction (section 4.1, fig. 5). The second commitment $\mathbf{H}_2$ is introduced to commit to $\mathbf{D}^T$. This commitment is used when checking whether the server has run the proof preprocessing phase honestly. This complication occurs since the direction of the $\mathbf{D}$ multiplication differs when preprocessing proofs and homomorphically evaluating answers. We also replace Verify with a simpler PreVerify, which verifies with preprocessed proof rather than fresh proofs.

- $\mathsf{Accept/Reject} \leftarrow \mathsf{VLHE.PreVerify}(\mathbf{u},\mathbf{v},\mathbf{C},\mathbf{Z})$
  Check that $\mathbf{Zu} = \mathbf{Cv}$. If the check passes, output Accept. Otherwise, output Reject.

Correctness. In order to accommodate the encryption of a row of $\mathbf{Z}$, we will need an Encrypt algorithm with the plaintext space $\mathbb{Z}_{p\ell}$ and the evaluation of the linear function $\mathbf{D}^T \in \mathbb{Z}_p^{m \times \ell}$. Let $\mathsf{Encrypt}_{p\ell}$ denote this algorithm and let $\mathsf{Decrypt}_{p\ell}$ denote the corresponding decryption algorithm. We have an updated correctness requirement from eq. (4):

$$q \geq \sigma \cdot 2\ell^2 p^2 \sqrt{2\ell \ln(2/\delta)}. \tag{6}$$

to account for this increase in the plaintext modulus increasing from p to ℓp in the preprocessing encryptions and the length ℓ of the rows of $\mathbf{D}^T$. Note that the encryptions in the homomorphic evaluation phase still use p as the plaintext modulus.

The correctness of the proof preprocessing protocol follows directly from the correctness of the homomorphic computation of $\mathbf{Z} = \mathbf{CD} = (\mathbf{D}^T\mathbf{C}^T)^T$. Observe that the homomorphic computation produces ciphertexts that encrypt each row of $\mathbf{Z}$. These rows are computed as $\mathbf{Z}[i] = \left(\mathbf{D}^T \cdot (\mathbf{C}[i])^T\right)^T$ for the sampled $\mathbf{C}$, where $\mathbf{C}[i]$ is the i^{th} row of $\mathbf{C}$.

Security. We give an intuition for the security of this preprocessed protocol. The full security proofs are given in appendix B.

The security of the proof processing protocol in fig. 6 follows from the security of our VLHE scheme (construction 4.1). In particular, the server does not learn any information on the challenge $\mathbf{C}$ during the protocol since the semantic security of the encryption hides $\mathbf{C}$. For soundness, we argue that $\mathbf{Z} = \mathbf{CD}$ must hold if the checks pass (in particular, if the check $\mathbf{ZA}_1 = \mathbf{CH}_1$ passes). At a high level, this follows from the soundness of our VLHE (lemma 4.1) together with lemma 2.4: By lemma 4.1, the server has committed to some fixed linear function $\mathbf{D}'$ with $\mathbf{H}_2$, and it is guaranteed that $\mathbf{Z} = \mathbf{C}(\mathbf{D}')^T$. Therefore, we have $\mathbf{ZA}_1 = \mathbf{C}(\mathbf{D}')^T\mathbf{A}_1 = \mathbf{CH}_1$, so $\mathbf{C}\left((\mathbf{D}')^T\mathbf{A}_1 - \mathbf{H}_1\right) = 0$ holds. By lemma 2.4, this occurs with negligible probability if $(\mathbf{D}')^T\mathbf{A}_1 - \mathbf{H}_1 \neq 0$, so we have that $(\mathbf{D}')^T\mathbf{A}_1 = \mathbf{H}_1$ with overwhelming probability. Then, by SIS hardness, we must have that $(\mathbf{D}')^T = \mathbf{D}$ and thus $\mathbf{Z} = \mathbf{CD}$.

The security of the simplified homomorphic evaluation phase in fig. 6 is straightforward from the security of our VLHE scheme (construction 4.1). The only difference is that we can skip a fraction of the checks as it is already done in the proof preprocessing phase. We note that, however, when verification fails, leakage on $\mathbf{C}$ may occur. Then, the client must discard $\mathbf{C}$ and the corresponding $\mathbf{Z}$ and rerun the proof preprocessing phase.

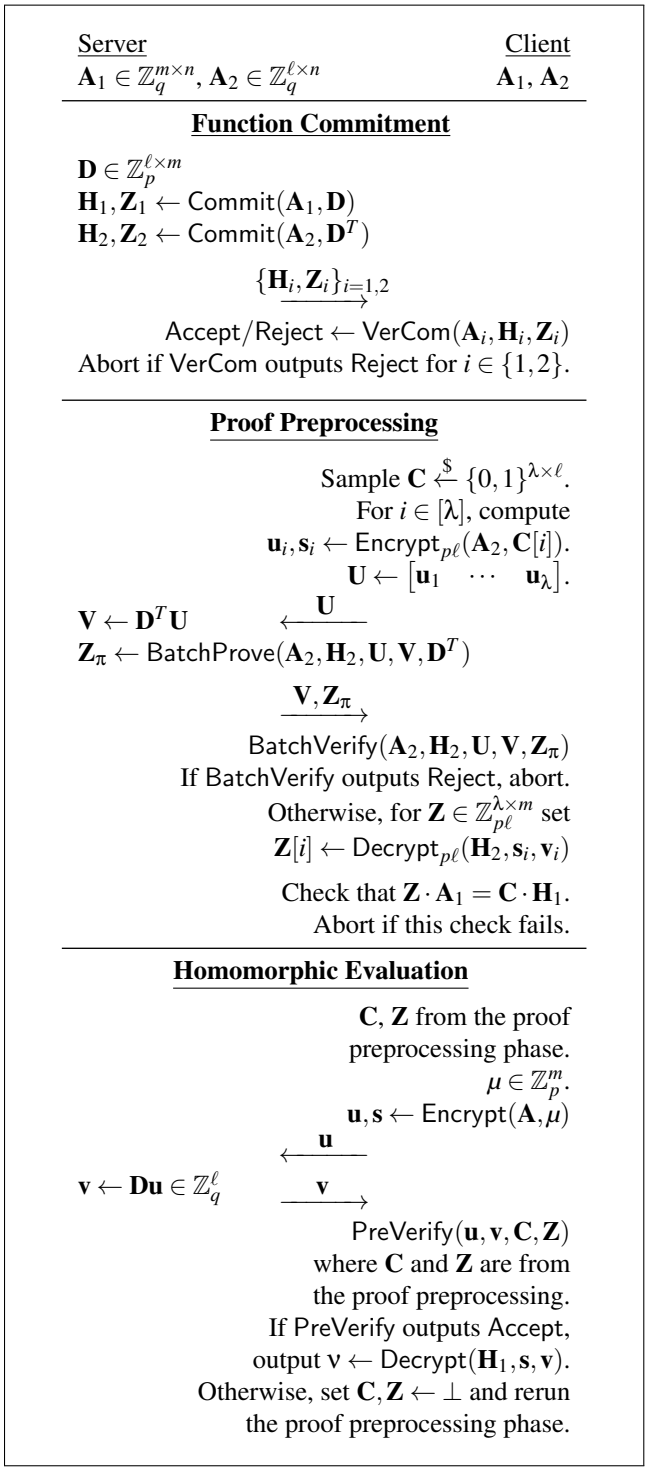

Figure 6: Protocol for computing reusable VLHE proofs. The proof is safe to reuse in many homomorphic evaluations as long as verification PreVerify passes. Once verification fails, a new proof must be computed.

5 Verifiable SimplePIR Construction

In this section, we present our full VeriSimplePIR protocol. The protocol follows directly from the VLHE construction from section 4. Recall from section 2.4.1 that we can view a database as a matrix $\mathbf{D} \in \mathbb{Z}_p^{\ell \times m}$ and queries as one-hot vectors $\mathbf{b}_i$. The response to this query is $\mathbf{d}_i = \mathbf{D} \cdot \mathbf{b}_i$, the i^{th} column of $\mathbf{D}$. The scheme is instantiated using the preprocessed VLHE scheme described in fig. 6. Observe the parallel structure of the preprocessed VLHE construction in fig. 6 and the preprocessed verifiable PIR API in fig. 3. We give the construction in construction 5.1. Note that the algorithms Query, Answer, and Recover are identical to the SimplePIR definitions in section 2.4.1.

Construction 5.1 (Preprocessed Verifiable PIR)**.** *The input parameters to this scheme are a computational security parameter n, a statistical security parameter λ, a plaintext modulus t, and a database size N. Let $p, \ell, m \in \mathbb{N}$ be such that a database in $\mathbb{Z}_t^N$ can be packed into an element of $\mathbb{Z}_p^{\ell \times m}$. For q that satisfies eq. (6), assume the hardness of $\mathsf{SIS}_{n,m,q,\beta_1}$ for $\beta_1 = 4\ell p$ and $\mathsf{SIS}_{n,\ell,q,\beta_2}$ for $\beta_2 = 4mp$. Furthermore, assume the hardness of (n,ℓ,q)-LWE and (n,m,q)-LWE.*

Let VLHE *denote construction 4.1 instantiated with plaintext modulus p and ciphertext modulus q.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N, t)$
  Output
  $$\mathbf{A}_1 \leftarrow \mathsf{VLHE.Setup}(1^n, 1^m, q)$$
  $$\mathbf{A}_2 \leftarrow \mathsf{VLHE.Setup}(1^n, 1^\ell, q)$$
  as pp.

- $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pp}, \mathbf{D})$
  Compute
  $$\mathbf{H}_1, \mathbf{Z}_1 \leftarrow \mathsf{VLHE.Commit}(\mathbf{A}_1, \mathbf{D})$$
  $$\mathbf{H}_2, \mathbf{Z}_2 \leftarrow \mathsf{VLHE.Commit}(\mathbf{A}_2, \mathbf{D}^T).$$
  Output $\mathsf{d} \leftarrow (\mathbf{H}_1, \mathbf{Z}_1, \mathbf{H}_2, \mathbf{Z}_2)$.

- $\mathsf{Accept/Reject} \leftarrow \mathsf{DigVer}(\mathsf{pp}, d)$
  Run $\mathsf{VLHE.VerCom}(\mathbf{A}_i, \mathbf{H}_i, \mathbf{Z}_i)$ *for* $i \in \{1,2\}$. *Output* Reject *if either* VerCom *iterations outputs* Reject. *Otherwise, output* Accept.

- $\mathsf{q}_\pi, \mathsf{st}_\pi \leftarrow \mathsf{PrQry}(\mathsf{pp})$
  Sample $\mathbf{C} \xleftarrow{\$} \{0,1\}^{\lambda \times \ell}$. *For* $i \in [\lambda]$, *compute* $\mathbf{u}_i, \mathbf{s}_i \leftarrow \mathsf{VLHE.Encrypt}_{p\ell}(\mathbf{A}_2, \mathbf{C}[i])$. *Set* $\mathbf{U} \leftarrow \begin{bmatrix} \mathbf{u}_1 & \cdots & \mathbf{u}_\lambda \end{bmatrix}$. *Output* $\mathsf{q}_\pi \leftarrow \mathbf{U}$ *and* $\mathsf{st}_\pi \leftarrow (\mathbf{C}, \{\mathbf{s}_i\}_{i \in \lambda})$.

- $\mathsf{a}_\pi \leftarrow \mathsf{PrAns}(\mathsf{pp}, \mathbf{D}, \mathsf{d}, \mathsf{q}_\pi)$
  Set $\mathbf{V} \leftarrow \mathbf{D}^T\mathbf{U}$ *and compute*
  $$\mathbf{Z}_\pi \leftarrow \mathsf{VLHE.BatchProve}(\mathbf{A}_2, \mathbf{H}_2, \mathbf{U}, \mathbf{V}, \mathbf{D}^T).$$
  Output $\mathsf{a}_\pi \leftarrow \mathbf{V}, \mathbf{Z}_\pi$.

- $\pi/\perp \leftarrow \mathsf{PrRec}(\mathsf{pp}, \mathsf{d}, \mathsf{q}_\pi, \mathsf{st}_\pi, \mathsf{a}_\pi)$
  Run $\mathsf{VLHE.BatchVerify}(\mathbf{A}_2, \mathbf{H}_2, \mathbf{U}, \mathbf{V}, \mathbf{Z}_\pi)$ *and output* $\perp$ *if* BatchVerify *outputs* Reject. *Otherwise, for* $\mathbf{Z} \in \mathbb{Z}_{p\ell}^{\lambda \times m}$ *set* $\mathbf{Z}[i] \leftarrow \mathsf{VLHE.Decrypt}_{p\ell}(\mathbf{H}_2, \mathbf{s}_i, \mathbf{v}_i)$. *Check that* $\mathbf{Z} \cdot \mathbf{A}_1 = \mathbf{C} \cdot \mathbf{H}_1$. *If this check fails, output* $\perp$. *Otherwise, output* $\pi \leftarrow \mathbf{Z}$.

- $\mathsf{q}, \mathsf{st} \leftarrow \mathsf{Query}(\mathsf{pp}, i \in [N])$
  Write i *as* $(i_r, i_c) \in [\ell] \times [m]$. *Compute*
  $$\mathbf{u}, \mathbf{s} \leftarrow \mathsf{VLHE.Encrypt}(\mathbf{A}_1, \mathbf{b}_{i_c}).$$
  Output $(\mathsf{q}, \mathsf{st}) \leftarrow (\mathbf{u}, (\mathbf{s}, i))$.

- $\mathsf{a} \leftarrow \mathsf{Answer}(\mathsf{pp}, \mathbf{D}, q)$
  Output $\mathbf{v} \leftarrow \mathbf{D}\mathbf{u}$ *as* a.

- $\mathsf{r} \leftarrow \mathsf{Recover}(\mathsf{pp}, \mathsf{st}, \mathsf{a}, \mathsf{d})$
  Write i *as* $(i_r, i_c) \in [\ell] \times [m]$. *Compute*
  $$\nu \leftarrow \mathsf{VLHE.Decrypt}(\mathbf{H}_1, \mathbf{s}, \mathbf{v})$$
  for $\nu \in \mathbb{Z}_p^\ell$. *Output* $r \leftarrow \nu[i_r]$.

- $\mathsf{Accept}/\mathsf{Reject} \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathsf{d}, \mathsf{q}, \mathsf{a}, \pi)$
  Output the result of $\mathsf{VLHE.PreVerify}(\mathbf{A}_1, \mathbf{H}_1, \mathbf{u}, \mathbf{v}, \mathbf{C}, \mathbf{Z})$.

The security of this scheme follows immediately from the security of the preprocessed VLHE scheme presented in section 4. The full proofs are given in appendix C.

6 Implementation & Performance Analysis

In this section, we present our implementation of the preprocessed VeriSimplePIR protocol[1]. We give performance benchmarks and compare against the semi-honest SimplePIR protocol.

Simple Optimizations. We enumerate some simple optimizations that we incorporate into our implementation. These optimizations are designed to maximize the performance for the honest server, who will essentially never have to run the proof preprocessing more than once for each client.

1. **Optimizing Client Storage.** The client does not need to store the second database commitment $\mathbf{H}_2$ once the preprocessing phase is complete. As soon as the client holds valid $\mathbf{C}$ and $\mathbf{Z}$ matrices, it discards all other messages from the commitment and preprocessing phases. The persistent client storage for the online phase is only the matrices $\mathbf{H}_1$, $\mathbf{C}$, and $\mathbf{Z}$. We include the download of $\mathbf{H}_2$ and the corresponding $\mathbf{Z}_2$ in the communication of the proof preprocessing as an initial message before the encrypted $\mathbf{C}$ is sent to the server.

2. **Minimizing Ciphertext Overhead.** Observe that the correctness constraint on the VLHE ciphertext modulus for the preprocessing operations (given in eq. (6)) is more strict than the constraint on this modulus in the online phase (given in eq. (4)). Observe that if a modulus q that satisfies the online constraint in eq. (4) then $q' \geq \kappa \cdot q \geq \sqrt{\ell^3/m}$ will satisfy the preprocessing constraint in eq. (6). We set the online modulus q to be a machine word size (either 2^{32} or 2^{64}), then run the preprocessing over the larger modulus $q' \geq \kappa \cdot q$. This introduces only a slight overhead to the preprocessing phase, since if κ is odd then q and κ are coprime. Therefore, $\mathbb{Z}_{q'} = \mathbb{Z}_q \otimes \mathbb{Z}_\kappa$, so almost no multiprecision arithmetic is required to implement the VLHE operations over $\mathbb{Z}_{q'}$. This allows us to maximize the usage of the machine-word modulus in the online phase.

3. **Optional Assumption of Honest Digest.** Recall that the VLHE correctness must account for a database $\mathbf{D}$ that is as large as the extractability bound of $||\mathbf{D}||_\infty \leq 2\ell p$, which is larger than the honest database bound of $||\mathbf{D}||_\infty \leq p$. Since the database digest is the same across all clients, additional checks on this digest could be implemented fairly easily. In particular, if the digest is signed by a sufficient number of trusted parties, a client can be confident that the digest was generated with a database $\mathbf{D}$ such that $||\mathbf{D}||_\infty \leq p$. Note that the SIS assumption still fixes this database (since $p \leq 2\ell p$) so even if the server is corrupted at a later time the database must still be this fixed, honest value. This allows us to further relax the correctness requirement on the VLHE ciphertext modulus q (see eq. (2)).

Below, we will benchmark optimal parameters with and without the assumption of an honest digest.

Experimental Setup. Our implementation is in C/C++. All computational benchmarks presented here were run on a single thread of a machine running Ubuntu 20 with an Intel i7 chip operating at 2.5 GHz with 32 GB of RAM. The VeriSimplePIR code was compiled with version 10 of the `clang++` compiler using the `-O3` option. All SimplePIR benchmarks were taken with the VeriSimplePIR library using optimal parameters. Both VeriSimplePIR and SimplePIR parameters were chosen to minimize online communication. In particular, all SimplePIR benchmarks throughout this work use $\log(q) = 32$, $\log(n) = 10$, and $\log(p)$ chosen to minimize online communication. We also give benchmarks of the SimplePIR protocol with a 64-bit modulus, which is optimized for online computation on a 64-bit machine. For all of these benchmarks, database entries are one bit ($d = 1$). Note that due to the tight database packing scheme, the database entry size has essentially no effect on the parameters[2]. Only changing the overall database size (i.e. $N \cdot d$) will change

[1] https://github.com/leodec/VeriSimplePIR

[2] The only exception to this is when the database entry size is greater than a $\sqrt{N}$ fraction of the database, but this is much larger than entries in essentially all practical applications.

the parameters of both SimplePIR and VeriSimplePIR. All benchmarks are presented in terms of the total database size.

Benchmarks: Online Phase. In fig. 7, we present the online performance of preprocessed VeriSimplePIR and compare to the online performance of SimplePIR. This comparison shows that verifiability can be achieved with very low overhead. In particular, VeriSimplePIR without the honest digest assumption has a 12% to 40% slowdown in the compute time and a 40% to 50% increase in the total communication. When the honest assumption is introduced, the communication overhead of VeriSimplePIR drops to 13% to 20%, and the online computation actually *outperforms* the original semi-honest SimplePIR protocol. Note that this is because parameters are optimized to minimize communication rather than computation. The minimal communication parameters from SimplePIR use $\log(q) = 32$, while the minimal parameters for VeriSimplePIR use $\log(q) = 64$. This means that the VeriSimplePIR protocol is able to pack more database bits into each machine word, which results in an improved throughput on a 64-bit machine. Growing the SimplePIR modulus to 64 bits would more than double the size of the database digest, as we will see below.

Benchmarks: Offline Phase. In fig. 8, we present the one-time offline communication required to both download the initial database commitment and preprocess a proof. The VeriSimplePIR bars in fig. 8 are split into two parts. The bottom part of the bar represents the data that must be stored locally on the client's machine throughout the protocol (including the online phase). This data is dominated ($> 95\%$) by the size of the digest $\mathbf{H}_1$, although it also contains the $\mathbf{C}$ and $\mathbf{Z}$ matrices that are output from the proof preprocessing phase. The top part of the bar represents the communication in the proof preprocessing phase that is *not* stored once the proof preprocessing phase is finished. Again, this data is almost entirely ($> 95\%$) the matrix $\mathbf{H}_2$, the commitment to $\mathbf{D}^T$.

Due to space constraints, we briefly summarize the offline computation benchmarks. On a single-core, the offline computation with a dishonest digest is roughly 100 seconds for a 4 GB database is roughly 200 seconds and for an 8 GB database. The computation is almost entirely on the server and scales linearly with the database. As we mentioned above, a full system would take advantage of the parallel nature of the server's computation and the optimizations admitted by the machine-words modulus.

6.1 Comparing APIR with VLHE PIR

We now compare the performance of our *non-preprocessed* PIR protocol to the authenticated PIR protocols (called APIR and APIR+) of Colombo et al. [CNCG+23]. The performance of the APIR protocol is essentially unaffected by malicious behavior, while our preprocessed protocol would require rerunning the proof precomputation after each verification failure. In order to account for a setting where the server is frequently malicious, we compare against a simplified variant of our protocol that simply applies the (non-preprocessed) verifiable linearly homomorphic encryption scheme defined in fig. 4 to the PIR computation. We call this variant VLHE PIR, and this version requires no per-client offline computation; the client simply downloads the database commitment and expects the server to send a fresh proof with each PIR response. To account for the larger dependence on the row length in the proof $\mathbf{Z}$, we rebalance the database dimensions to get a roughly $\sqrt{\lambda}$ communication overhead over SimplePIR. Note that all benchmarks in this section consider VLHE PIR *without* the honest hint assumption.

While all benchmarks in the previous section considered single-bit database entries, it is important to consider the database entry size when benchmarking the APIR protocols. While SimplePIR and VLHE PIR scale only with the size of the database, APIR and APIR+ crucially rely on 1-bit entries for security, handling larger entries via repetition.

Due to space constraints, we briefly summarize the online performance comparison, where VLHE PIR outperforms APIR and APIR+ in both communication and computation for all database sizes and all entry widths. When the database entries are a single bit, the total runtime of APIR+ is roughly $5\times$ greater than the runtime of VLHE PIR, and the runtime of APIR is about $25\times$ greater than VLHE PIR. When compared the the APIR protocol, VLHE PIR sees a near 40% reduction in the online communication. The comparison to APIR+ is even better, where the VLHE PIR online communication is 7-12$\times$ smaller.

The only metric for which either protocol outperforms VHLE PIR is the offline communication, with the APIR+ offline communication at about $4.5\times$ smaller than VLHE PIR when the database entries are a single bit. However, as displayed in fig. 9, the performance of APIR+ quickly decays as the database entries grow. For example, with a 64 GiB database with 32-bit entries, VLHE PIR outperforms both protocols on all metrics. By the time the database entry bitwidth reaches 1024, the offline communication of APIR+ is roughly $7\times$ the offline communication of VLHE PIR.

7 Application: Password Leak Detection

In this section, we discuss an application of VeriSimplePIR: secure password leak detection. The goal of this application is to allow clients to query a database of leaked passwords from sources such as "Have I Been Pwnd?"[3] without revealing the queried password they are requesting to the server. The need for privacy in this application is clear, since honest clients would essentially always be interested in their own passwords. Revealing the query to the server would reveal the client's credentials.

[3] https://haveibeenpwned.com/Passwords

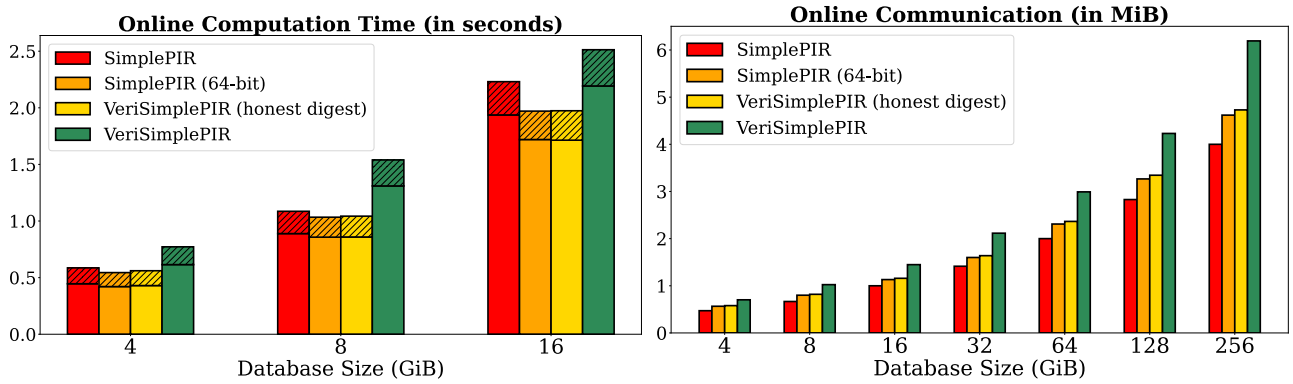

Figure 7: Online performance of SimplePIR and preprocessed VeriSimplePIR for databases of various sizes. The left figure is the per-query online computation, and the right figure is the per-query online communication. The hatched portion of the computation graph represents work on the client side (Query, Verify, and Recover). In the communication benchmarks, the upload and download sizes are equal.

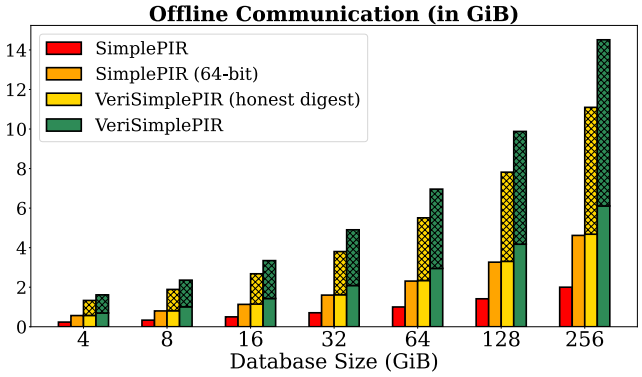

Figure 8: Offline communication of SimplePIR and preprocessed VeriSimplePIR for databases of various sizes. The hatched portions in the VeriSimplePIR columns represent the communication that is *not stored* once the precomputation is finished. The unhatched portion represents the data that the client must store locally throughout the protocol (including the online phase).

This application is based on the Blyss private password checker[4]. In the Blyss implementation, the database consists of around 400 million SHA1 hashes (20 bytes each) of passwords that have appeared in data breaches. These passwords are stored in a hash table about 8 GB in size. The hash function is public, so a client can map their hashed password to a candidate database index. The Blyss implementation uses either Spiral [MW22] or DoublePIR [HHCG+23], both of which are semi-honest implementations. Despite the significantly weaker security guarantees, VeriSimplePIR is highly competitive with the online performance of both schemes. DoublePIR is worse in both online communication and online computation, and Spiral has around 25× slower online computation [MW22, Table 2] than VeriSimplePIR. The only overhead from VeriSimplePIR is the roughly 800 MB of data that each client must locally store throughout the online phase. This is perhaps too large for mobile devices, but it is certainly feasible for a laptop or desktop computer. Furthermore, it is likely that queries requiring a higher level of security would come from desktop or laptop computers, such as the master password for a password manager. In such cases, clients would require the strong guarantees of verifiable PIR and would likely have the local storage to keep the digest.

If clients make infrequent queries, they could avoid storing the full digest and instead only store a hash of the digest. Redownloading the digest from the server would maintain verifiability by checking against the stored hash. Note that the client must store the preprocessed proof or offload this data only in encrypted form. We leave a full implementation of this system as well as exploration of other applications (such as the PGP server proposed in the work of Colombo et al. [CNCG+23]) for future work.

[4] https://playground.blyss.dev/passwords

8 Related Work

The first PIR scheme was presented by Kushilevitz and Ostrovsky [KO97]. This work was also the first to identify that a malicious server could dynamically maul database to try to learn the client's query in what became known as a selective failure attack.

Stateless verifiable PIR. Recent work [BDKP22] introduced another notion of verifiable PIR in which the client can verify arbitrary local properties of the database used for the response. This verifiability notion differs from our work in that it is *stateless*, which is to say there is no initial commitment to a database. The stateless notion of verifiability is defined *per* query and thus does not defend against attacks that modify the database across the queries. In contrast, our verifiability is defined for a database that is fixed throughout the lifetime of the scheme (or at least the digest). We also note that this stateless construction is based on recent developments in the batch argument for NP, placing it outside the realm of

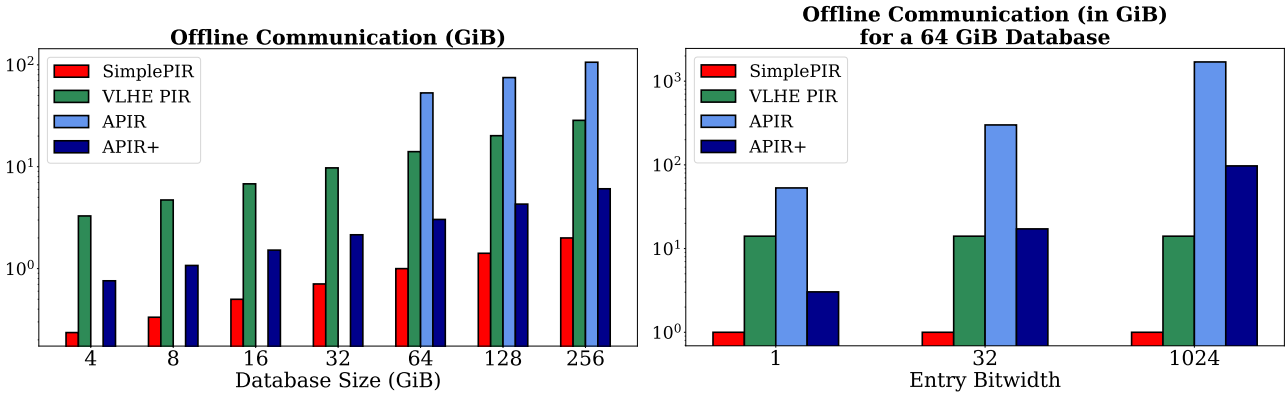

Figure 9: Offline communication comparison of VLHE PIR with APIR and APIR+. The left plot is for 1-bit database entries. APIR benchmarks below 45 GB are ommitted since the download is larger than the database size.

practicality.

Several other prior works [WZ18, ZWH21] have considered verifiable PIR in the single-server setting. However, the security guarantees achieved in these works are much weaker than in ours. In particular, they only guarantee that the PIR response was generated with respect to some well-formed database, but there is no commitment to fix this database across multiple queries.

Semi-honest PIR. In addition to SimplePIR, a number of recent works have improved both the theoretical and practical performance of semi-honest PIR. This includes FrodoPIR [DPC23], which is essentially the same protocol as SimplePIR. There have been several recent works exploring the exciting direction of PIR with *sublinear* online complexity [CGK20, CGHK22, LMW23, ZPSZ23]. It remains an interesting open question how to add malicious security to these schemes.

Acknowledgments

We would like to thank Vinod Vaikuntanathan for insightful conversations and Alexandra Henzinger for helpful discussions, especially regarding the implementation.

Research was supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149, a Simons Investigator award, and a Google-Schwarzman fellowship. The second author was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIT) (No.2017R1A5A1015626).

References

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 962–979, 2018.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery.

[AS16] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, Savannah, GA, November 2016. USENIX Association.

[Ban95] Wojciech Banaszczyk. Inequalites for convex bodies and polar reciprocal lattices in R^n. *Discret. Comput. Geom.*, 13:217–231, 1995.

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafael del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 669–699, Cham, 2018. Springer International Publishing.

[BDKP22] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable private information retrieval. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography*, pages 3–32, Cham, 2022. Springer Nature Switzerland.

[CGHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022*, pages 3–33, Cham, 2022. Springer International Publishing.

[CGK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 44–75, Cham, 2020. Springer International Publishing.

[CKGS98] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, nov 1998.

[CN11] Yuanmi Chen and Phong Q. Nguyen. Bkz 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[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.

[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.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, pages 31–51, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[HHCG+23] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905, Anaheim, CA, August 2023. USENIX Association.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 18–35, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, 1997.

[KRS+19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1447–1464, Santa Clara, CA, August 2019. USENIX Association.

[KS06] Mehmet Kiraz and Berry Schoenmakers. A protocol issue for the malicious case of Yao's garbled circuit construction. In *27th Symposium on Information Theory in the Benelux*, volume 29, pages 283–290, 2006.

[LG15] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security*, pages 168–186, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, page 595–608, New York, NY, USA, 2023. Association for Computing Machinery.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, pages 319–339, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[LPA+19] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1387–1403, New York, NY, USA, 2019. Association for Computing Machinery.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Post-Quantum Cryptography*, pages 147–191, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[MW22] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 930–947, 2022.

[PIB+22] Bijeeta Pal, Mazharul Islam, Marina Sanusi Bohuk, Nick Sullivan, Luke Valenta, Tara Whalen, Christopher Wood, Thomas Ristenpart, and Rahul Chatterjee. Might I get pwned: A second generation compromised credential checking service. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1831–1848, Boston, MA, August 2022. USENIX Association.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 387–398, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[TPY+19] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1556–1571, Santa Clara, CA, August 2019. USENIX Association.

[WZ18] Xingfeng Wang and Liang Zhao. Verifiable single-server private information retrieval. In David Naccache, Shouhuai Xu, Sihan Qing, Pierangela Samarati, Gregory Blanc, Rongxing Lu, Zonghua Zhang, and Ahmed Meddahi, editors, *Information and Communications Security*, pages 478–493, Cham, 2018. Springer International Publishing.

[ZPSZ23] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server PIR with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452, 2023. https://eprint.iacr.org/2023/452.

[ZSN14] Liang Feng Zhang and Reihaneh Safavi-Naini. Verifiable multi-server private information retrieval. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security*, pages 62–79, Cham, 2014. Springer International Publishing.

[ZWH21] Liang Zhao, Xingfeng Wang, and Xinyi Huang. Verifiable single-server private information retrieval from lwe with binary errors. *Information Sciences*, 546:897–923, 2021.

A Verifiable PIR Detailed Definitions

In this section, we give detailed definitions for preprocessed verifiable PIR correctness and security.

Completeness Definition. In addition to the correctness requirement for the PIR output (definition 2.2), we require that messages produced by honest servers pass verification in both the preprocessing and the online phase. We formalize this requirement as the following.

Definition A.1 (Verification Completeness)**.** *We say a vPIR scheme is* complete *if the following holds for all security parameter* λ*, database* $\mathbf{D} \in \mathbb{Z}_t^N$*, and index* $i \in [N]$*. (The public parameter* pp *is an implicit input for all algorithms other than* Setup*.)*

$$\Pr\left[\begin{array}{c|r} \pi \neq \perp \\ \& \\ \mathsf{Verify}(\mathsf{d},\pi,\mathsf{q},\mathsf{a}) \\ = \mathsf{Accept} \end{array} \middle| \begin{array}{r} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N, t) \\ \mathsf{d} \leftarrow \mathsf{Digest}(\mathbf{D}) \\ \mathsf{q}_\pi, \mathsf{st}_\pi \leftarrow \mathsf{PrQry}(\mathsf{d}) \\ \mathsf{a}_\pi \leftarrow \mathsf{PrAns}(\mathbf{D}, \mathsf{q}_\pi) \\ \pi \leftarrow \mathsf{PrRec}(\mathsf{d}, \mathsf{q}_\pi, \mathsf{st}_\pi, \mathsf{a}_\pi) \\ \mathsf{q}, \mathsf{st} \leftarrow \mathsf{Query}(i) \\ \mathsf{a} \leftarrow \mathsf{Answer}(\mathbf{D}, \mathsf{q}) \end{array}\right] = 1.$$

Security Definitions. The following are security requirements for vPIR. At a high level, we require any malicious behavior of a server in vPIR should lead to an abort. Additionally, we need a stronger notion of privacy than the query-hiding of semi-honest PIR (definition 2.3) to defend against malicious attacks such as selective failures.

Definition A.2 (Digest Binding)**.** *For a security parameter* λ*, let* $\mathcal{S}$ *be a computationally bounded server. Let* pp *be an honestly-generated public parameter. We say a vPIR scheme is* digest binding *if the following holds:*

1. *It is infeasible for* $\mathcal{S}$ *to produce two databases* $\mathbf{D}$ *and* $\mathbf{D}'$ *such that* $\mathsf{Digest}(\mathsf{pp}, \mathbf{D}) = \mathsf{Digest}(\mathsf{pp}, \mathbf{D}')$.

2. *Suppose $\mathcal{S}$ can output a_π such that* $\mathsf{PrRec}(\mathsf{pp},\mathsf{d},\mathsf{q}_\pi,\mathsf{st}_\pi,\mathsf{a}_\pi) \neq \perp$ *for* $\mathsf{q}_\pi,\mathsf{st}_\pi \leftarrow \mathsf{PrQry}(\mathsf{d})$ *with non-negligible probability, where the probability is over the randomness of* PrQry *and* $\mathcal{S}$*. Then, there exists an efficient extractor $\mathcal{E}$ that extracts some database $\mathbf{D}$ such that* $\mathsf{Digest}(\mathsf{pp},\mathbf{D}) = \mathsf{d}$ *by rewinding $\mathcal{S}$, with probability at least* $1-2^{-\lambda}$.

Definition A.3 (Verification Soundness)**.** *For a security parameter λ, let $\mathcal{S}$ be a computationally bounded server. For an honestly-generated public parameter* pp*, let* $\mathsf{d} \leftarrow \mathsf{Digest}(\mathsf{pp},\mathbf{D})$ *for some $\mathbf{D}$. Furthermore, let $\pi \neq \perp$ be the auxiliary information computed in the preprocessing phase described in fig. 3. Let* a *be a response computed by $\mathcal{S}$ for a query* q*. We say a vPIR scheme satisfies* verification soundness *if the following holds with probability at least* $1-2^{-\lambda}$*:*

$$\mathsf{Verify}(\mathsf{pp},\mathsf{d},\pi,\mathsf{q},\mathsf{a}) = \mathsf{Accept} \implies \mathsf{a} = \mathsf{Answer}(\mathbf{D},\mathsf{q}).$$

Definition A.4 (Query Hiding Against a Malicious Server)**.** *For a security parameter λ, let* pp *be an honestly-generated public parameter. We say a vPIR scheme that satisfies* verification completeness *(definition A.1),* digest binding *(definition A.2), and* verification soundness *(definition A.3), is also* query hiding against a malicious server *if the following holds.*

For every computationally bounded adversary $\mathcal{A}$ let $\mathsf{d},\mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{pp})$ *be the digest produced by $\mathcal{A}$ such that* $\mathsf{Accept} \leftarrow \mathsf{DigVer}(\mathsf{pp},\mathsf{d})$*. Let $\mathbf{D}$ be the database the adversary used to produce* d*; the existence and uniqueness of $\mathbf{D}$ is guaranteed by definition A.2. For every index sequence $\vec{i} = \{i_1,\ldots,i_L\} \in [N]^L$, define the distribution*

$$REAL_{\mathcal{A},\vec{i}} := \left\{ \begin{array}{c} \mathsf{q}_\pi,\mathsf{st}_\pi \leftarrow \mathsf{PrQry}(\mathsf{pp},\mathsf{d}) \\ \mathsf{a}_\pi,\mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{q}_\pi) \\ \pi \leftarrow \mathsf{PrRec}(\mathsf{pp},\mathsf{d},\mathsf{q}_\pi,\mathsf{st}_\pi,\mathsf{a}_\pi) \\ \beta \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A}) \quad \mathit{if}\ \ \pi = \perp \\ \mathit{While}\ \ \pi \neq \perp \\ \beta : \left\{ \begin{array}{c} \mathsf{q},\mathsf{st} \leftarrow \mathsf{Query}(\mathsf{pp},i_k) \\ \mathsf{a},\mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{q}) \\ \mathsf{b} \leftarrow \mathsf{Verify}(\mathsf{pp},\mathsf{d},\pi,\mathsf{q},\mathsf{a}) \\ \mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{b}) \\ \mathit{If}\ \ \mathsf{b} = \mathsf{Reject},\ \ \mathit{set}\ \ \pi \leftarrow \perp \\ \mathit{Otherwise} \\ r \leftarrow \mathsf{Recover}(\mathsf{pp},\mathsf{st},\mathsf{a},\mathsf{d}) \\ \mathsf{b}' \leftarrow 1[r = \mathbf{D}[i_k]] \\ \mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{b}') \end{array} \right\}_{k=1}^{L} \\ \beta \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A}), \end{array} \right\}$$

where the L iterations are run sequentially. Similarly, for a computationally bounded simulator $\mathcal{X}$, define the distribution

$$IDEAL_{\mathcal{A},\mathcal{X}} := \left\{ \begin{array}{c} \mathsf{q}_\pi,\mathsf{st}_\pi \leftarrow \mathsf{PrQry}(\mathsf{pp},\mathsf{d}) \\ \mathsf{a}_\pi,\mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{q}_\pi) \\ \pi,\mathsf{st}_\mathcal{X} \leftarrow \mathcal{X}(\mathsf{pp},\mathsf{d},\mathsf{q}_\pi,\mathsf{st}_\pi,\mathsf{a}_\pi) \\ \beta \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A}) \quad \mathit{if}\ \ \pi = \perp \\ \mathit{While}\ \ \pi \neq \perp \\ \beta : \left\{ \begin{array}{c} \mathsf{q},\mathsf{st}_\mathcal{X} \leftarrow \mathcal{X}(\mathsf{pp},\mathsf{st}_\mathcal{A}) \\ \mathsf{a},\mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{q}) \\ \mathsf{b},\mathsf{b}' \leftarrow \mathcal{X}(\mathsf{pp},\mathsf{d},\pi,\mathsf{q},\mathsf{a},\mathsf{st}_\mathcal{A}) \\ \mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{b}_k) \\ \mathit{If}\ \ \mathsf{b} = \mathsf{Reject},\ \ \mathit{set}\ \ \pi \leftarrow \perp \\ \mathit{Otherwise} \\ \mathsf{st}_\mathcal{A} \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A},\mathsf{b}') \end{array} \right\}_{k=1}^{L} \\ \beta \leftarrow \mathcal{A}(\mathsf{st}_\mathcal{A}), \end{array} \right\}$$

where the L iterations are run sequentially as in REAL. We say that a vPIR scheme is query hiding against a malicious server *if the following holds:*

$$|\Pr[REAL_{\mathcal{A},\vec{i}} = 1] - \Pr[IDEAL_{\mathcal{A},\mathcal{X}} = 1]| \leq \mathsf{negl}(\lambda).$$

B Security Proofs for Reusable VLHE

Here, we give the security proofs for the VLHE scheme with preprocessed proofs presented in section 4.2.

Lemma B.1 (Semi-honest Preprocessed Protocol)**.** *Let $\mathcal{S}$ be a computationally bounded server that correctly runs the protocol described in fig. 6. Let $b_F \in \{0,1\}$ indicate if the client aborts on the function commitment phase and let $b_P \in \{0,1\}$ indicate if the client aborts on the proof preprocessing phase. Let $\mathbf{U}$ be the client's message in the proof preprocessing phase, as in fig. 6. The view of $\mathcal{S}$ at the end of the preprocessing phase is $(b_F,\mathbf{U},b_P)$.*

For q that satisfies eq. (6), assume the hardness of (n,ℓ,q)-LWE. The view of $\mathcal{S}$ is computationally indistinguishable from $(0,\mathbf{R},0)$, where $\mathbf{R}$ is a uniformly random element of $\mathbb{Z}_q^{\ell\times\lambda}$.

Proof. This follows directly from the semantic security of the Regev encryption scheme and the perfect completeness of the verification algorithms VerCom and BatchVerify. □

Lemma B.2 (Correct Encryption of $\mathbf{Z}$)**.** *Let $\mathcal{S}$ be a computationally bounded adversary acting as the server in the protocol described in fig. 6. Define the view of $\mathcal{S}$ at the end of the proof preprocessing phase $(b_F,\mathbf{U},b_P)$ as in lemma B.1. Let q, ℓ, p, and m satisfy eq. (6). Assume the hardness of $\mathsf{SIS}_{n,m,q,\beta_1}$ for $\beta_1 = 4\ell p$ and $\mathsf{SIS}_{n,\ell,q,\beta_2}$ for $\beta_2 = 4mp$. As in lemma B.1, assume the hardness of (n,ℓ,q)-LWE.*

Consider the commitment defined by $(\mathbf{A}_1,\mathbf{H}_1)$ and the extractor $\mathcal{E}$ guaranteed by lemma 2.2 running on $\mathcal{S}$ in the function commitment phase. Let $\mathbf{D} \in \mathbb{Z}^{\ell\times m}$ be the unique matrix that can be extracted from $\mathcal{E}$ on the commitment $(\mathbf{A}_1,\mathbf{H}_1)$ (i.e. $\mathbf{H}_1 = \mathbf{D}\mathbf{A}_1$), under hardness assumption of $\mathsf{SIS}_{n,m,q,\beta_1}$

(lemma 2.3). Define $\mathbf{D}' \in \mathbb{Z}^{m \times \ell}$ *similarly on the commitment* $(\mathbf{A}_2, \mathbf{H}_2)$ *(i.e.* $\mathbf{H}_2 = \mathbf{D}'\mathbf{A}_2$*), under hardness assumption of* $\mathsf{SIS}_{n,m,q,\beta_2}$.

If a client running the prescribed protocol completes the proof preprocessing without aborting, then with probability at least $1 - 2^{-\lambda}$ *the* $\mathbf{Z}$ *value held by the client equals* $\mathbf{CD}$ *for the client-sampled* $\mathbf{C}$*. Furthermore, the view of* $\mathcal{S}$ *is indistinguishable from the semi-honest view in lemma B.1 of with probability at least* $1 - 2^{-\lambda}$.

Proof. To see that the client's $\mathbf{Z}$ value will equal $\mathbf{CD}$, observe that since the ciphertext modulus q satisfies eq. (6), applying the linear function $\mathbf{D}' \in \mathbf{Z}^{m \times \ell}$ to the ciphertexts comprising the columns of $\mathbf{U}$ will result in correctly decrypting ciphertexts $\mathbf{V}$. Since BatchVerify accepts, then by lemma 4.1 and the hardness of $\mathsf{SIS}_{n,\ell,q,\beta_2}$ for $\beta_2 = 4mp$ it must be that $\mathbf{V} = \mathbf{D}'\mathbf{U}$. Therefore, by the correctness of the VLHE scheme, it must be that the plaintext encrypted in the columns of $\mathbf{V}$ is $\mathbf{Z}^T = \mathbf{D}'\mathbf{C}^T$.

Meanwhile, the check $\mathbf{Z} \cdot \mathbf{A}_1 = \mathbf{C} \cdot \mathbf{H}_1 = \mathbf{C}(\mathbf{D}')^T\mathbf{A}_1$ passes, and thus we have $\mathbf{C}((\mathbf{D}')^T\mathbf{A}_1 - \mathbf{H}_1) = 0$. Since $\mathbf{A}_1$, $\mathbf{H}_1$, and $\mathbf{D}'$ are all fixed before $\mathbf{C}$ is sampled, then lemma 2.4 implies that $(\mathbf{D}')^T\mathbf{A}_1 - \mathbf{H}_1 = \mathbf{0}$ with probability at least $1 - 2^{-\lambda}$. We now have that $(\mathbf{D}')^T = \mathbf{D}$ by the hardness of $\mathsf{SIS}_{n,m,q,\beta_1}$ for $\beta_1 = 4\ell p$, which implies $\mathbf{Z} = \mathbf{CD}$.

The claim regarding the view of $\mathcal{S}$ follows from lemma 4.1. The correct ciphertext $\mathbf{V}$ is defined by the linear function committed with $(\mathbf{A}_2, \mathbf{H}_2)$. Since anything other than the correct ciphertext $\mathbf{V}$ will be rejected by BatchVerify with probability at least $1 - 2^{-\lambda}$, then the server must return the correct ciphertext with at least this probability. Since the ciphertext $\mathbf{V}$ is correct, then there is no leakage from verification and decryption proceeding as in the prescribed protocol. The remainder of the lemma follows from the semantic security of the Regev encryption scheme, which makes $\mathbf{U}$ computationally indistinguishable from a random element in $\mathbb{Z}_q^{m \times \lambda}$. □

Lemma B.3 (Soundness of Preprocessed Verification)**.** *Let* $\mathcal{S}$ *be a computationally bounded adversary acting as the server in the protocol described in fig. 6. Define the view of* $\mathcal{S}$ *at the end of the proof preprocessing phase* $(b_F, \mathbf{U}, b_P)$ *as in lemma B.1. Let* q, ℓ, p, *and* m *satisfy eq. (6). Assume the hardness of* $\mathsf{SIS}_{n,m,q,\beta_1}$ *for* $\beta_1 = 4\ell p$ *and* $\mathsf{SIS}_{n,\ell,q,\beta_2}$ *for* $\beta_2 = 4mp$*. As in lemma B.1, assume the hardness of* (n, ℓ, q)*-LWE.*

Consider the commitment defined by $(\mathbf{A}_1, \mathbf{H}_1)$ *and the extractor* $\mathcal{E}$ *guaranteed by lemma 2.2 running on* $\mathcal{S}$ *in the function commitment phase. Let* $\mathbf{D} \in \mathbb{Z}^{\ell \times m}$ *be the unique matrix that can be extracted from* $\mathcal{E}$ *on the commitment* $(\mathbf{A}_1, \mathbf{H}_1)$ *(i.e.* $\mathbf{H}_1 = \mathbf{D}\mathbf{A}_1$*), under hardness assumption of* $\mathsf{SIS}_{n,m,q,\beta_1}$ *(lemma 2.3).*

If the client has completed the proof preprocessing phase without aborting, then $\mathsf{Accept} \leftarrow \mathsf{PreVerify}(\mathbf{u}, \mathbf{v}, \mathbf{C}, \mathbf{Z})$ *implies that* $\mathbf{v} = \mathbf{Du}$ *with probability at least* $1 - 2^{-\lambda}$*. Furthermore, this guarantee is maintained across any number of homomorphic evaluation protocols as long as* PreVerify *does not output* Reject.

Proof. The security of a single instance of the homomorphic evaluation phase follows directly from lemma B.2, since the server has no information about the challenge $\mathbf{C}$ used to verify the response. The uniqueness of the ciphertext $\mathbf{v}$ then follows from lemma 4.1.

The second claim follows from the simulatability of the client accepting the correct transcript. Strong uniqueness of the response ciphertext $\mathbf{v}$ guarantees that the only way the server will pass verification with probability better than $2^{-\lambda}$ is if $\mathbf{v} = \mathbf{Du}$. If the server returns the correct ciphertext, then clearly there is no additional information about $\mathbf{C}$ that is learned, since the fact that the client accepts this transcript follows directly from the perfect completeness of the PreVerify algorithm. Therefore, with probability at least $1 - 2^{-\lambda}$, no information about $\mathbf{C}$ is leaked if PreVerify outputs Accept. This means that the challenge $\mathbf{C}$ and the corresponding $\mathbf{Z}$ can be safely reused as long as PreVerify outputs Accept. □

C VeriSimplePIR Security Proofs

We give the security proofs of the VeriSimplePIR construction in section 5.

Lemma C.1 (VeriSimplePIR Completeness)**.** *Construction 5.1 satisfies definition A.1.*

Proof. This proof follows directly from the perfect completeness of the VLHE verification in section 4. In particular, the bound on $\mathbf{Z}$ will never be exceeded as long as the database $\mathbf{D}$ is within the honest bound $||\mathbf{D}||_\infty \leq p$. The perfect completeness of the check $\mathbf{ZA} = \mathbf{CH}$ follows immediately from the honest values of $\mathbf{H}$ and $\mathbf{Z}$. □

Lemma C.2 (VeriSimplePIR Digest Binding)**.** *Construction 5.1 satisfies definition A.2.*

Proof. This follows directly from the SIS hardness assumptions given in construction 5.1. This SIS assumption immediately implies computational binding for the digest from lemma 2.3. □

Lemma C.3 (VeriSimplePIR Verification Soundness)**.** *Construction 5.1 satisfies definition A.3.*

Proof. This follows directly from lemma B.3. □

Lemma C.4 (VeriSimplePIR Query Hiding)**.** *Construction 5.1 satisfies definition A.4.*

Proof. This follows from lemma B.3 along with the correctness of the VLHE parameters. Lemma B.3 guarantees that the only ciphertexts that pass verification with probability better than $2^{-\lambda}$ are the honest ciphertexts. Decryption of the honest ciphertext will yield the correct result with probability at least $1 - 2^{-\lambda}$ by lemma 2.5. Therefore, the second bit b' in the distributions in definition A.4 are also simulatable. □