HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical

Diego F. Aranha[1], Anamaria Costache[2], Antonio Guimarães[3], and Eduardo Soria-Vazquez[4]

[1] Aarhus University, Denmark. `dfaranha@cs.au.dk`
[2] NTNU, Norway. `anamaria.costache@ntnu.no`
[3] IMDEA Software Institute, Spain. `antonio.guimaraes@imdea.org`
[4] Technology Innovation Institute, UAE. `eduardo.soria-vazquez@tii.ae`

Abstract. Homomorphic encryption (HE) enables computation on encrypted data, which in turn facilitates the outsourcing of computation on private data. However, HE offers no guarantee that the returned result was honestly computed by the cloud. In order to have such guarantee, it is necessary to add verifiable computation (VC) into the system.

The most efficient recent works in VC over HE focus on verifying operations on the ciphertext space of the HE scheme, which usually lacks the algebraic structure that would make it compatible with existing VC systems. For example, multiplication of ciphertexts in the current most efficient HE schemes requires non-algebraic operations such as real division and rounding. Therefore, existing works for VC over HE have to either give up on those efficient HE schemes, or incur a large overhead (an amount of constraints proportional to the ciphertext ring's size) in order to emulate these non-algebraic operations.

In this work, we move away from that paradigm by placing the verification checks in the *plaintext space* of HE, all while the prover remains computing on ciphertexts. We achieve this by introducing a general transformation for Interactive Oracle Proofs (IOPs) to work over HE, whose result we denote as HE-IOPs. We apply this same transformation to the FRI [Ben-Sasson et al., ICALP 2018] IOP of proximity and we show how to compile HE-Reed Solomon-encoded IOPs and HE-δ-correlated-IOPs with HE-FRI into HE-IOPs. Furthermore, our construction is compatible with a prover that provides input in zero-knowledge, and only relies on building blocks that are plausibly quantum-safe.

Aligning the security parameters of HE and FRI is a difficult task for which we introduce several optimizations. We demonstrate their efficiency with a proof-of-concept implementation and show that we can run FRI's commit phase for 4096 encrypted Reed Solomon codewords with degree bound 2^{11} in just 5.4 seconds (using 32 threads) on a `c6i.metal` instance using less than 4GB of memory. Verification takes just 12.3 milliseconds (single-threaded) for the same parameter set and can be reduced to just 5.6ms with parameters optimized for the verifier.

1 Introduction

There are many usability and economic benefits for citizens and companies to outsource data storage/processing to remote servers, but cloud computing brings significant integrity and privacy risks. Homomorphic Encryption (HE) has been referred to by many as the "holy grail" technology to address the privacy risks of outsourced computing. Since the first scheme introduced by Gentry in 2009 [Gen09b], numerous advances and improvements have followed [BGV12, Bra12, FV12, CGGI20, CKKS17]. In particular, a privacy-preserving variant of the use-case of Machine Learning as a Service (MLaaS) has been shown to be particularly suitable for HE by a recent line of work [BMMP18, BCCW19, BGBE19].

However, HE by itself does not guarantee the integrity of the computing party. Dealing with this issue falls within the scope of Verifiable Computation (VC), which describes a collection of techniques ensuring that the output returned by the cloud servers is indeed the honest result of applying the requested function to the designated data. On the other hand, VC on its own does not protect the privacy of the outsourced yet sensitive data from the clients.

The cryptographic community realized that VC and HE are very complementary, since the limitations of the one are perfectly covered by the features of the other. Combining the two techniques is often referred to as "privacy-preserving verifiable computation" or "verifiable computation on encrypted data". The first solution was proposed by Gennaro, Gentry and Parno [GGP10] in 2010, and employs a heavy combination of Yao's garbled circuit for an one-time verifiable computation together with HE to reuse the garbled circuit across many inputs.

The later work of [FGP14] is efficient, but very limited in expressiveness. The use of homomorphic MACs limits the application to depth-1 circuits and requires to keep a secret verification key hidden from the prover, hence eliminating public verifiability. The work of [FNP20] improves the expressiveness of [FGP14], by allowing to efficiently compute circuits of (arbitrary) constant depth. Nevertheless, they only deal with a very narrow subset of inefficient HE schemes: a variant of BV [BV14], where the integer ciphertext modulus q matches the (prime) order of the source groups in the underlying pairing-based SNARK.

Both [GNS23] and [BCFK21] overcome the limitation in the selection of HE schemes in [FNP20] by supporting an arbitrary rather than a prime ciphertext modulus q. Still, for both works, dealing with the more complex HE operations such as modulo switching and key switching is very expensive. This makes it unclear whether it would be more practical to support efficient but complex schemes such as BGV and BFV, or BV with a potentially non-prime q.

The main problem for all those works is that they verify whether certain operations are done on (simplified versions of) the ciphertext space. Furthermore, they need to emulate the arithmetic of HE ciphertexts in one way or another, incurring large overheads. While the addition of ciphertexts can be easily emulated (as addition of elements in the ciphertext space R_q^2), the product of ciphertexts is less algebraically structured and hard to emulate, since it mixes a number of computational steps such as bit-wise operations, real division, rounding, the product of elements in R_q and changing q during modulo switching operations.

As an example of how expensive these techniques were, consider trying to emulate the `HE` arithmetic within R_q (as in [GNS23, FNP20]), which is arguably the closest algebraic structure. Every bit-wise operation involved in the product of ciphertexts, such as rounding (present in the `HE.ModSwitch` operation in BGV and BFV), has the cost of one constraint[5] per bit of the ciphertext ring R_q.

This cost increases rapidly as the multiplicative depth of the circuit grows, not only because of the number of such operations but also because of how the `HE` parameters (including ciphertext size) grow accordingly. In the BGV scheme (as well as the BFV and CKKS one), increasing the multiplicative depth of the circuit by one usually requires to add a prime to the prime chain that makes up the ciphertext modulus. In more practical terms, this corresponds to increasing the ciphertext modulus by $30-50$ bits every time we increase the multiplicative depth by one. This means that the ciphertext modulus will grow exponentially with the depth d of the circuit that one wants to evaluate (this takes the security level into account; see for example [APS15, CP19]). Alternatively, works like [BCFK21] circumvent the issue of doing bit-wise operations by using the BV scheme. In their work, the size of the ciphertext ring grows exponentially with the number of ciphertext-ciphertext multiplications that one wants to evaluate.

[5] The number of *constraints* in R1CS or other models of computation are the main metric for the efficiency in SNARKs.

In this work, we deviate from this paradigm by enabling the verification of operations on the *plaintext space* of the HE scheme. At a high level, we show how to adapt holographic IOP-based SNARKs so that, on the one hand, the prover computes obliviously on the encrypted values while, on the other hand, the verifier performs the verification checks on the plaintext space. We choose to focus on holographic IOPs as a departure point for our verifiable computation protocol, since the holography property is particularly well suited for outsourcing scenarios. Nevertheless, our techniques could easily be adapted to non-holographic IOPs. We call our overall framework HElIOPolis, since its central components are homomorphic encryption (HE) and Interactive Oracle Proofs (IOPs).

1.1 Technical overview

We manage to move verification from the ciphertext to the plaintext space by replacing the IOP oracles $\mathcal{O}$ with *encrypted* oracles $\mathcal{O}_{\mathtt{HE}}$, which are oracles to data that is homomorphically encrypted. While the prover $\mathcal{P}$ does not know *what* the plaintexts they are computing on are, $\mathcal{P}$ knows how to arrange them into oracles (i.e. $\mathcal{P}$ can place $\mathtt{HE.Enc}(x)$ into an oracle $\mathcal{O}_{\mathtt{HE}}$, rather than x into $\mathcal{O}$). Whereas the modified IOP (denoted HE-IOP) can now only be verified by whoever has the HE decryption key[6], this new abstraction is very powerful: not only is the prover much more efficient, it is also very simple to reduce the security of an HE-IOP to that of its corresponding IOP (Theorem 2). Moreover, we also adapt several results in the literature compiling different variants of (zk)IOPs into (zk)SNARKs [COS20, BGK$^+$23]. Our resulting zkSNARKs are plausibly post-quantum, since so are the BCS transform [BCS16, CMS19] and all the efficient HE schemes we have today.

Once oracles are replaced with *encrypted* oracles, our approach is black box on the different components of these compilers. A central part of these is the use of an IOP of proximity (IOPP) to Reed-Solomon Codes, which is interpreted either as a low degree test or a correlated agreement test [BCI$^+$20, BGK$^+$23]. As done in practice for unencrypted IOPs, we choose FRI [BBHR18] to instantiate this IOPP component. FRI is, a priori, particularly HE-friendly, in the sense that it only runs linear operations on the functions being tested, and products in HE are particularly expensive. Nevertheless, there are several challenges when trying to align the security parameters of HE schemes and FRI. This constitutes a significant part of our work, for which we discuss trade-offs and optimizations (Section 6) as well as we provide experimental data (Section 7).

Aligning security parameters. The first challenge is enabling FRI to work over a field of size $|\mathbb{F}_{p^D}| \approx 2^{256}$ for some prime p, This ensures that FRI remains secure when making it non-interactive through Fiat-Shamir for any Reed-Solomon codeword we would encounter in practice when compiling IOPs [BGK$^+$23]. Using p^D as a plaintext modulus in the HE scheme would result in unmanageable parameters. We address this by emulating $\mathbb{F}_{p^D}$ arithmetic using D ciphertexts, each encrypting elements from $\mathbb{F}_p$.

Reducing depth and exploiting HE packing. A second challenge to the homomorphic evaluation of FRI is its multiplicative depth. Although it only involves multiplications between plaintexts and ciphertexts, the noise level can increase almost as much as with ciphertext multiplications, since plaintext are random elements of $\mathbb{F}_p$. A typical implementation of FRI would have depth $2n$ for an input of size 2^n, which represents a performance challenge for HE schemes. We solve this problem by introducing low-depth versions of every sub-routine required to evaluate FRI, and

[6] In concurrent work on IOPs over encrypted data [GGW23], the authors discuss the use of fully homomorphic *commitments* [GVW15] as a way to recover public verifiability, but all known constructions for such primitive are very inefficient.

show how to perform Reed-Solomon codeword encoding with small fixed depth, as opposed to the traditional depth-n methods. We also propose a "Shallow Fold" algorithm to replace FRI's standard `Fold` operation, which reduces the depth to 1 (from n), at the cost of increasing the complexity to $O(2^n \log(2^n))$ (from $O(2^n)$), which does not change the overall asymptotics. Additionally, we exploit packing within the `HE` scheme to further reduce the cost of Reed-Solomon encoding. In more detail, we consider packing methods that trade off memory consumption and execution time to accelerate the prover.

Minimizing HE overhead for the verifier. Finally, we take advantage of techniques proposed in [CGGI20, CLOT21] to implement a repacking and recomposing technique, which significantly reduces the overhead of ciphertext decryption for the verifier. During the commit phase of FRI, the prover performs computations using RLWE samples of dimension N encrypting N messages in $\mathbb{F}_p$. During the query phase, however, the verifier only needs to learn two evaluation points in $\mathbb{F}_{p^D}$ per round for each linearity check. If those ciphertexts are fully packed, an overhead of at least $N/(2D)$ is introduced. In order to avoid this, we extract the evaluation points from the RLWE samples of dimension N and repack them in another RLWE sample, but of a much smaller dimension, reducing decryption costs up to 128 times depending on the selected parameters. One key observation is that at this point, we *do not need to preserve any homomorphic properties*, as the verifier does not perform any further operations on these. Indeed, once the commit phase is finished, we can view the evaluation points as simply strings of bits, and our goal then becomes to encrypt them in the smallest possible ciphertext. This also makes the HE parameters adopted by the verifier completely independent of the ones adopted by the prover or of the input size 2^n.

All our optimizations are specifically targeted for FRI. Whether other existing IOPs of proximity (e.g. [ACY23]) or new ones could be better aligned in practice with `HE` schemes such as BGV and BFV is an interesting open work that our theoretical machinery already supports.

Proof-of-Concept Implementation. To demonstrate the practicality of our construction, we implement a proof-of-concept over the FRI implementation of Szepieniec *et al.* [S+21]. We extend it to work over non-prime fields and connect it to optimized FHE libraries. We test two parameter sets for encrypted codewords of size up to 2^{16} representing polynomials with degree bound up to $d = 2^{15}$. For a batch of 4096 polynomials with degree bound $d = 2^{11}$, our implementation takes just 5.4 seconds to run FRI's commit phase (including the Reed-Solomon code encoding) on 32 threads in a `c6i.metal` instance on AWS and requires less than 4GB of memory. Verification is much faster, taking just 12.3ms single-threaded (also for a batch of 4096 polynomials). With a parameter set optimized for the verifier, verification time drops to just 5.6ms single-threaded, at the cost of increasing the prover execution to 1.3 minutes. Our implementation is publicly available at `https://github.com/antoniocgj/HELIOPOLIS`.

Oracle attacks. Moving verification from the ciphertext to the plaintext space inherently opens up for side-channel and composition attacks. If the verifier signals to the prover whether verification passes, this leaks 1 bit of information about the plaintext/secret key. A priori, it could seem that reusing the plaintext output obtained by the verifier as input to another protocol could also leak about the secret key, but such issues can be avoided altogether by having the prover provide Zero Knowledge Proofs of Knowledge for any ciphertext it would use as input. We discuss these risks in more detail in Section 3.1. We believe that, for many applications, the speed-ups we achieve through this paradigm shift far outweigh the one-bit privacy loss. This is reflected by the lack of implementations for works on the previous paradigm (verification over the ciphertext space), for which, except for very low-depth circuits, their lack of efficiency is a non-starter.

1.2 Comparison with existing works

In concurrent work [GGW23], Garg, Goel and Wang offer a framework to prove statements on values that are hidden from the prover. Their framework is based on making FRI work over such hidden values, and they show how to compile Polynomial IOPs into SNARKs given such a tool. Besides $\mathtt{HE}$, their work considers more general ways to hide these values from the prover, such as homomorphic commitments and group exponentiation, grouped under the abstraction of Linearly-Homomorphic Encapsulations.

A formal issue in [GGW23] is that their notion of a *decryptable* (or that of *linearly-homomorphic w.r.t. randomness*) Linearly-Homomorphic Encapsulation is not sufficient when such an encapsulation is a building block of more complex components such as FRI or polynomial commitments. Namely, their notion only considers decryption of a freshly encrypted ciphertext on which no operations have been performed. This overlooks the fact that the evaluation correctness of $\mathtt{HE}$ schemes, which are based on (Ring) Learning with Errors, is function-specific and needs to support the operations computed within those components. Whereas FRI only requires to perform a series of linear combinations on the ciphertexts, it turns out that the size of the coefficients in the linear combination and the additive depth of FRI constitutes a significant obstacle for noise management in practice (see Section 6).

While [GGW23] is more theoretical and lacks implementation, our work focuses on concretely accelerating VC on encrypted data. In addition to various technical optimizations and trade-offs informed by our experimentation, we address two key aspects that [GGW23] does not: scenarios where the prover provides inputs in zero knowledge, and the direct use of FRI (e.g. by compiling δ-correlated IOPs into (zk)SNARKs [BGK$^+$23]), rather than going through a polynomial commitment abstraction. The former greatly improves the parameters of the $\mathtt{HE}$ scheme in several applications, such as Privacy-Preserving Machine Learning (where the prover provides their model as a plaintext in zero-knowledge while the verifier's input are $\mathtt{HE}$ ciphertexts), whereas the latter improves FRI's parameters by allowing to use a proximity parameter up to the Johnson bound.

2 Preliminaries

2.1 Notation

We use $R[\mathtt{X}]_{\leq d}$ to refer to polynomials with coefficients in a finite, commutative ring R and degree at most d. For an element $a \in R$, we write $[a]_q$ to denote the reduction of a modulo q (coefficient-wise), with the set of representatives of coefficients lying in $\{0, \ldots, q-1\}$. This should not be confused with $[n]$, which we will sometimes use to denote the set of integers $\{1, \ldots, n\}$.

We use bold notation (e.g. $\mathbf{b}$) to refer to vectors. We use $y \leftarrow C$ to denote that y is the output of a given computation C. We use $a \leftarrow A$ for sampling an element a from a distribution A. When A is a set rather than a distribution, we write $a \overset{\$}{\leftarrow} A$ for sampling a uniformly at random in A. We write $[\![f]\!]$ to denote an oracle to f, and M^f to denote that M has oracle access to f. We abbreviate Probabilistic Polynomial Time as PPT. We let λ denote a computational security parameter. We denote computational indistinguishability by $\overset{c}{\approx}$ when no PPT algorithm can distinguish between two distributions except with negligible probability.

2.2 Basic algebra and Galois theory

Next, we present some Number and Galois Theory facts that were noted in the context of FHE in [SV14], but that are fairly standard. Let p be a prime, $F(x) \in \mathbb{F}_p[x]$ be a polynomial, $\deg(F) = N$, and assume that it factorises $\pmod{p}$ into ℓ factors, all of degree D, for $D \cdot \ell = N$, i.e.

$$F(x) = \prod_{i=1}^{\ell} F_i(x) \pmod{p},$$

where $\deg(F_i) = D,\ \forall i \in [\ell]$.

Then we can define $R_p \coloneqq \mathbb{F}_p[x]/(F)$, and the following holds.

$$\begin{aligned} R_p &\cong \mathbb{F}_p[x]/(F_1) \times \ldots \times \mathbb{F}_p[x]/(F_\ell) \\ &\cong \mathbb{F}_{p^D} \otimes \ldots \otimes \mathbb{F}_{p^D}. \end{aligned} \tag{1}$$

Let $F = \Phi_{2N}$ be the $2N^{\text{th}}$ cyclotomic polynomial, for N a power of two, and $\deg(\Phi_{2N}) = N$. We note in particular that the above implies that, if p is a prime such that $\mathbb{F}_p$ contains a primitive $2\ell^{\text{th}}$ root of unity, we have that

$$\Phi_{2N}(x) = \prod_{i \in (\mathbb{Z}/2\ell\mathbb{Z})^\times} (x^D - \zeta^i) \pmod{p}.$$

In the *fully-splitting* case, we have that $D = 1$, and can therefore write

$$R_p \cong \underbrace{\mathbb{F}_p \otimes \ldots \otimes \mathbb{F}_p}_{N\text{copies}}.$$

We refer to the cases where D is small with respect to N as *almost-fully splitting.*

2.3 Homomorphic Encryption

Definition 1 *A public-key Homomorphic Encryption scheme* $\mathtt{HE}$ *over a set of admissible circuits* $\widehat{\mathcal{C}\text{irc}}$ *consists of the following algorithms.*

- $(\mathtt{pp}, \mathcal{C}) \leftarrow \mathtt{HE.Setup}(1^\lambda, \mathcal{M}, \widehat{\mathcal{C}\text{irc}})$: *Given a message space* $\mathcal{M}$*, a set of admissible circuits* $\widehat{\mathcal{C}\text{irc}}$*, output the public parameters* $\mathtt{pp}$ *and the ciphertext space* $\mathcal{C}$ *such that the scheme is semantically secure.*
- $(\mathtt{sk}, \mathtt{pk}, \mathtt{evk}) \leftarrow \mathtt{HE.KeyGen}(1^\lambda, \mathcal{C}, \mathtt{pp})$: *Given the public parameters* $\mathtt{pp}$ *and the ciphertext space* $\mathcal{C}$*, output the secret key* $\mathtt{sk}$*, the public key* $\mathtt{pk}$ *and the evaluation key* $\mathtt{evk}$*.*
- $\mathtt{ct} \leftarrow \mathtt{HE.Enc}(\mathtt{pk}, m)$: *Given a message* $m \in \mathcal{M}$*, output its encryption* $\mathtt{ct}$*.*
- $m' \leftarrow \mathtt{HE.Dec}(\mathtt{sk}, \mathtt{ct})$: *Given a ciphertext* $\mathtt{ct}$ *and its corresponding secret key* $\mathtt{sk}$*, output the decryption of* $\mathtt{ct}$*,* m'*.*
- $\mathtt{ct}' \leftarrow \mathtt{HE.Eval}(\mathtt{evk}, \mathtt{ct}, \hat{C})$: *Given a circuit* $\hat{C} \in \widehat{\mathcal{C}\text{irc}}$*, output the evaluation of* $\hat{C}$ *on* $\mathtt{ct}$*.*

When the context is clear, we will omit specifying the $\mathtt{sk}, \mathtt{pk}, \mathtt{pp}, \mathcal{C}$ parameters. The (standard) definition for semantic security of HE schemes is given in Appendix B.

Definition 2 *Let* HE *be a homomorphic scheme as in Definition 1. We say that* HE *is* correct *if the equation*

$$\mathtt{HE.Dec}(sk, \mathtt{HE.Eval}(evk, \mathtt{HE.Enc}(pk, m), \hat{C})) = C(m)$$

holds with overwhelming probability for all admissible circuits $\hat{C} \in \widehat{\mathcal{C}\mathrm{irc}}$.

We note that the set $\widehat{C}$ refers to the set of circuits that the scheme can support – for example, some HE schemes support non-linear operations such as ReLu, and some do not. In particular, each circuit $\hat{C} \in \widehat{\mathcal{C}irc}$ has a corresponding circuit C on the *plaintext space*. To give a concrete example, if we want to evaluate the homomorphic multiplication of two ciphertexts, $\hat{C}$ could be a multiplication followed by a bootstrapping, whereas C would simply be a multiplication.

2.4 Reed Solomon Codes

Reed-Solomon (RS) codes are arguably the most common linear error correction codes. Let us recall their definition and fix some notation relative to them. In this work, we will parameterize them by a finite field $\mathbb{F}$, a multiplicative subgroup $L \subseteq \mathbb{F}^*$ and a degree bound d. Hence, $\mathtt{RS}[\mathbb{F}, L, d]$ is defined as follows:

$$\mathtt{RS}[\mathbb{F}, L, d] = \{(f(x))_{x \in L} \in \mathbb{F}^{|L|} : f \in \mathbb{F}[\mathtt{X}]_{<d}\}.$$

The code rate of $\mathtt{RS}[\mathbb{F}, L, d]$ is $\rho = d/|L|$. Unless we state it otherwise, in this work we will assume that $|L| = 2^k$, $\rho = 2^{-R}$ and $d = 2^{k-R}$.

For two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^n$, we let $\Delta(u, v)$ denote the *relative Hamming distance* between $\boldsymbol{u}$ and $\boldsymbol{v}$, defined as $\Delta(\boldsymbol{u}, \boldsymbol{v}) := |\{u_i \neq v_i | i \in \{1, \dots, n\}|/n$. For a set of vectors $S \subset \mathbb{F}^n$ and any vector $\boldsymbol{u} \in \mathbb{F}^n$, we define $\Delta(\boldsymbol{u}, S) = \Delta(S, \boldsymbol{u}) := \min_{\boldsymbol{v} \in S}\{\Delta(\boldsymbol{u}, \boldsymbol{v})\}$. For $\delta \in (0, 1)$, we say that $\boldsymbol{u}$ is *δ-far from S* if $\Delta(\boldsymbol{u}, S) \geq \delta$. Otherwise, we say that $\boldsymbol{u}$ is *δ-close to S*. Equivalently, $\boldsymbol{u}$ is δ-far from S if $\Delta(\boldsymbol{u}, S) \geq \delta$ for all $\boldsymbol{v} \in S$, and $\boldsymbol{u}$ is δ-close to S if there exists $\boldsymbol{v}^* \in S$ such that $\Delta(\boldsymbol{u}, \boldsymbol{v}^*) < \delta$. We refer to δ as the *proximity parameter*. When $\delta < (1 - \rho)/2$, we say that δ is within the *unique decoding radius*; and when $\delta < 1 - \sqrt{\rho}$, we say that δ is within the Johnson bound.

Definition 3 (Correlated agreement) *Let $\delta \in (0, 1)$. Let $V = \mathtt{RS}[\mathbb{F}, L, d]$ and let $W = \{w_1, \dots, w_k\} \subseteq \mathbb{F}^{|L|}$. We say W has δ-*correlated agreement *with V on an* agreement set *$S \subseteq L$ if $|S|/|L| \geq 1 - \delta$ and there exist $v_1, \dots, v_k \in V$ such that, $\forall x \in S$, $w_i(x) = v_i(x)$.*

2.5 Interactive Oracle Proofs (of Proximity)

There are several variations of the IOP abstraction [BCS16]. Polynomial IOPs (PIOPs) ask for the IOP oracles to be polynomials evaluated over the entire field $\mathbb{F}$, whereas for the weaker notion of Reed Solomon-encoded IOPs (RS-IOPs) those are Reed-Solomon codewords (i.e. the evaluation of a polynomial over some specific domain $L \subset \mathbb{F}$). In this work, we focus on RS-encoded IOPs and on δ-correlated IOPs, which were introduced in [BGK+23]. Our results could nevertheless be easily adapted to other IOP flavors, e.g. to PIOPs as in [GGW23]. The main attractive of δ-correlated IOPs is that they allow for a better proximity parameter δ (up to the Johnson bound, rather than within the unique decoding radius) when they are compiled into SNARKs. When $\delta = 0$, δ-correlated IOPs can be seen as a subclass of RS-encoded IOPs [BCR+19, COS20].

Definition 4 *An indexed relation $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$. The string $\mathbb{x}$ is the called input, statement or instance, the string $\mathbb{w}$ is called the witness and the string $\mathbb{i}$ is an index. The index can be thought as something that is fixed at setup time, and chooses among a universe of binary relations $\mathcal{R}_{\mathbb{i}} = \{(\mathbb{x}; \mathbb{w}) : (\mathbb{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}\}$.*

In the setting of holographic proofs, a good example is an indexed relation for circuit satisfiability, where the index $\mathbb{i}$ is a description of the circuit, the statement $\mathbb{x}$ contains the "public" values on some of the circuit's input wires and the witness $\mathbb{w}$ consists in the values taken by the remaining "private" wires.

Definition 5 ([BGK+23]) *Let $H \subseteq \mathbb{F}$ and $d \geq 0$. An indexed $(\mathbb{F}, H, d)$-polynomial oracle relation $\mathcal{R}$ is an indexed relation where for each $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the index $\mathbb{i}$ and input $\mathbb{x}$ may contain oracles to codewords from $\mathtt{RS}[\mathbb{F}, H, d]$ and the actual codewords corresponding to these oracles are contained in $\mathbb{w}$.*

Definition 6 *A μ-round* holographic interactive oracle proof (hIOP) *for an indexed relation $\mathcal{R}$ is a tuple of PPT interactive algorithms $\Pi = (\mathcal{P}, \mathcal{V})$ and a deterministic polynomial-time algorithm $\mathtt{Ind}$ (the indexer), with two phases:*

- *In an offline phase, given an index $\mathbb{i}$, $\mathtt{Ind}$ computes an encoding of it, $\mathtt{Ind}(\mathbb{i})$.*
- *In an online phase, $\mathcal{P}(\mathtt{Ind}(\mathbb{i}), \mathbb{x}, \mathbb{w})$ and $\mathcal{V}^{\mathtt{Ind}(\mathbb{i})}(\mathbb{x})$ exchange $2\mu+1$ messages, where $\mathcal{P}$ sends the fist and last message. $\mathcal{V}$ gets only oracle access to $\mathcal{P}$'s messages, and after $\mathcal{P}$'s final message, $\mathcal{V}$ either accepts or rejects.*

Furthermore, an hIOP has to satisfy the two following properties:

Completeness: *For all $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$, we have that*

$$\Pr[\langle \mathcal{P}(\mathtt{Ind}(\mathbb{i}), \mathbb{w}), \mathcal{V}^{\mathtt{Ind}(\mathbb{i})} \rangle (\mathbb{x}) = 1] \geq \gamma,$$

where the probability is taken over the random coins of $\mathcal{V}$. If, for all $\mathbb{x}$, $\gamma = 1$, then the hIOP has perfect completeness.

Soundness: *For any $\mathbb{x} \notin \mathcal{L}_{\mathcal{R}}$ and any unbounded malicious $\mathcal{P}^*$,*

$$\Pr[\langle \mathcal{P}^*(\mathtt{Ind}(\mathbb{i}), \mathbb{w}), \mathcal{V}^{\mathtt{Ind}(\mathbb{i})} \rangle (\mathbb{x}) = 1] \leq \epsilon,$$

where the probability is taken over the random coins of $\mathcal{V}$.

In δ-correlated hIOPs, the prover is supposed to send oracles to maps that agree with low degree polynomials on a fraction of $1 - \delta$ points (see Definition 3). On top of checking all the received oracles correspond indeed to δ-correlated maps (which we capture by the relation in Definition 7), it is necessary to verify some algebraic equalities involving some evaluations of those maps. These are made concrete in Definition 8.

Definition 7 ([BGK+23]) *Let $0 \leq \delta < 1$. The δ-correlated agreement relation for $\mathtt{RS}[\mathbb{F}, L, d]$ is the following indexed $(\mathbb{F}, L, d)$-polynomial oracle relation:*

$$\mathtt{CoAgg} = \left\{ \begin{pmatrix} \mathbb{i} \\ \mathbb{x} \\ \mathbb{w} \end{pmatrix} = \begin{pmatrix} (\mathbb{F}, L, d, \delta, r) \\ (\llbracket f_i \rrbracket)_{i \in [r]} \\ (f_i)_{i \in [r]} \end{pmatrix} : \begin{array}{l} r, \delta \geq 0, \rho = d/|L| \\ f_i \in \mathbb{F}^L \ \forall i \in [r] \\ (f_i)_{i \in [r]} \text{ has } \delta\text{-correlated agreement with} \\ \mathtt{RS}[\mathbb{F}, L, d] \end{array} \right\}$$

Definition 8 (**δ-correlated hIOP, [BGK+23]**) *Let $L = \langle \omega \rangle$ be a smooth multiplicative subgroup of $\mathbb{F}^*$ of order $d = 2^v/\rho$ for some $v \geq 1$ and rate $0 < \rho < 1$ and define the Reed-Solomon code $\mathtt{RS}[\mathbb{F}, L, d]$. Let $0 \leq \delta < 1$ and let $\mathcal{R}$ be an indexed $(\mathbb{F}, L, d)$-polynomial oracle relation. Let Π be a hIOP for $\mathcal{R}$. Given a (possibly partial) transcript $(\mathbb{x}, \tau)$ generated during Π, let $\mathtt{Words}(\mathbb{x}, \tau)$ be the words from $\mathbb{F}^L$ that fully describe the oracles appearing in $(\mathbb{x}, \tau)$. We say that Π is δ-*correlated *if:*

- *The verifier $\mathcal{V}$ has oracle access to the δ-correlated agreement relation $\mathtt{CoAgg}(\delta)$.*
- *For all $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$:*
  - *In the last round of interaction between $\mathcal{P}(\mathtt{Ind}(\mathbb{i}), \mathbb{x}, \mathbb{w})$ and $\mathcal{V}^{\mathtt{Ind}(\mathbb{i}), \mathtt{CoAgg}(\delta)}(\mathbb{x})$, the verifier sends a field element z uniformly sampled from a subset of (a field extension of) $\mathbb{F}$ and the honest prover replies with the values:*

$$\mathtt{Evals}(\mathbb{x}, \tau, z) = (w(\omega^{k_{w,1}} z), \ldots, w(\omega^{k_{w,n_w}} z) : w \in \mathtt{Words}(\mathbb{x}, \tau))$$

  where τ is the transcript so far and $\kappa = \{k_{w,i} : w \in \mathtt{Words}(\mathbb{x}, \tau), i \in [n_w]\}$ is a fixed set of integers which are output by $\mathtt{Ind}$.
  - *To decide whether to accept or reject a proof, $\mathcal{V}^{\mathtt{Ind}(\mathbb{i}), \mathtt{CoAgg}(\delta)}(\mathbb{x})$ makes the two following checks:*

    Check 1 *Assert whether the received values $\mathtt{Evals}(\tau, z)$ are a root to some multivariate polynomial $F_{\mathbb{i}, \mathbb{x}, \tau}$ depending on $\mathbb{i}$, $\mathbb{x}$ and τ.*

    Check 2 *Assert whether the maps*

$$\mathtt{quotients}(\mathbb{x}, \tau, z) = \left\{ \frac{w(\mathtt{X}) - w(\omega^{k_{w,j}} z)}{\mathtt{X} - \omega^{k_{w,j}} z} : w \in \mathtt{Words}(\mathbb{x}, \tau), j \in [n_w] \right\}$$

    have δ-correlated agreement in $\mathtt{RS}[\mathbb{F}, L, d-1]$ by using the $\mathtt{CoAgg}(\delta)$ oracle on the oracles to such maps.

Next, we define the notions of round-by-round (RBR) soundness and knowledge soundness [CCH+19] for holographic IOPs. Since those are a superset of δ-correlated hIOPs, the same definition applies to the latter.

Definition 9 *A holographic IOP for an indexed relation $\mathcal{R}$ has* round-by-round (RBR) soundness *with error ϵ if for every index $\mathbb{i}$ there exists a "doomed set" $\mathcal{D}(\mathbb{i})$ of partial and complete transcripts such that:*

1. *If $\mathbb{x} \notin \mathcal{L}_{\mathcal{R}_{\mathbb{i}}}$, then $(\mathbb{x}, \emptyset) \in \mathcal{D}(\mathbb{i})$, where $\emptyset$ denotes the empty transcript.*
2. *For every possible input $\mathbb{x}$ and complete transcript τ, if $(\mathbb{x}, \tau) \in \mathcal{D}(\mathbb{i})$, then $\mathcal{V}^{\mathtt{Ind}(\mathbb{i})}(\mathbb{x}, \tau) =$ $\mathtt{reject}$.*
3. *If $i \in [\mu]$ and $(\mathbb{x}, \tau)$ is a $(i-1)$-round partial transcript such that $(\mathbb{x}, \tau) \in \mathcal{D}(\mathbb{i})$, then $\Pr_{c \overset{\$}{\leftarrow} C_i}[(\mathbb{x}, \tau, m, c) \notin \mathcal{D}(\mathbb{i})] \leq \epsilon(\mathbb{i})$ for every possible next prover message m.*

Definition 10 *A holographic IOP for an indexed relation $\mathcal{R}$ has* round-by-round (RBR) knowledge soundness *with error ϵ_k if there exists a polynomial time extractor* $\mathtt{Ext}$ *and for every index $\mathbb{i}$ there exists a "doomed set" $\mathcal{D}(\mathbb{i})$ of partial and complete transcripts such that:*

1. *For every possible input $\mathbb{x}$ (regardless of whether $\mathbb{x} \notin \mathcal{L}_{\mathcal{R}_{\mathbb{i}}}$ or not), $(\mathbb{x}, \emptyset) \notin \mathcal{D}(\mathbb{i})$.*
2. *For every possible input $\mathbb{x}$ and complete transcript τ, if $(\mathbb{x}, \tau) \in \mathcal{D}(\mathbb{i})$, then $\mathcal{V}^{\mathtt{Ind}(\mathbb{i})}(\mathbb{x}, \tau) =$ $\mathtt{reject}$.*

3. *Let $i \in [\mu]$ and $(\mathbb{x}, \tau)$ be a $(i-1)$-round partial transcript such that $(\mathbb{x}, \tau) \in \mathcal{D}(\mathbb{i})$. If for every possible next prover message m it holds that*

$$\Pr_{c \xleftarrow{\$} C_i} [(\mathbb{x}, \tau, m, c) \notin \mathcal{D}(\mathbb{i})] > \epsilon_k(\mathbb{i}),$$

then $\texttt{Ext}(\mathbb{i}, \mathbb{x}, \tau, m)$ outputs a valid witness for $\mathbb{x}$.

Finally, let us also discuss zero knowledge, which will be particularly interesting in our work.

Definition 11 *An hIOP Π for an indexed relation $\mathcal{R}$ has* statistical zero knowledge with query bound b *if there exists a PPT simulator $\mathcal{S}$ such that for every $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ and any $\mathcal{V}^*$ making less than b queries in total to its oracles, the random variables $\mathbf{View}(\mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathcal{V}^*)$ and $\mathcal{S}^{\mathcal{V}^*}(\mathbb{i}, \mathbb{x})$, defined below, are statistically indistinguishable*

– $\mathbf{View}(\mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w}), \mathcal{V}^*)$ *is the view of $\mathcal{V}^*$, i.e. the random variable $(r, a_1, \ldots, a_q)$ where r is $\mathcal{V}^*$ randomness and $a_1, \ldots, a_q$ are the responses to $\mathcal{V}^*$'s queries determined by the oracles sent by $\mathcal{P}$.*
– $\mathcal{S}^{\mathcal{V}^*}(\mathbb{i}, \mathbb{x})$ *is the output of $\mathcal{S}(\mathbb{i}, \mathbb{x})$ when given straightline (i,e, without rewinding) access to $\mathcal{V}^*$, prepended with $\mathcal{V}^*$ randomness r.*

Π is honest-verifier zero knowledge *if the above holds with $\mathcal{V}^* = \mathcal{V}^{\texttt{Ind}(\mathbb{i})}(\mathbb{x})$.*

Some examples of δ-correlated hIOPs are Plonky2, RISC Zero, ethSTARK, Aurora and Fractal [COS20]. One particular advantage of hIOPs is how easy it is to compile them into SNARKs through the so-called BCS transformation [BCS16]. In a nutshell, this consists in replacing oracles sent by the prover with Merkle-tree-based commitments and then removing interaction with the verifier by applying the Fiat-Shamir transform. It has been proved that if an hIOP is round-by-round sound, applying the BCS transformation results in a SNARK that is adaptively knowledge sound versus both classic and quantum adversaries in the random oracle model [CMS19, COS20].

A similar concept to the above one is that of an IOP of Proximity (IOPP), which is an IOP to test proximity to a specific code. In this work, we restrict ourselves to IOPPs for Reed Solomon Codes.

Definition 12 *Let $\texttt{RS}$ denote the family of Reed Solomon codes $\texttt{RS}[\mathbb{F}, L, d]$. A protocol between a pair of interactive machines $\langle \mathcal{P}, \mathcal{V} \rangle$ is an r-round interactive oracle proof of δ-proximity for $\texttt{RS}$ is an IOP with the following modifications*

– ***Input format:*** *The first message from $\mathcal{P}$ is $f_0 : L \to \mathbb{F}$, allegedly a $\texttt{RS}$ codeword.*
– ***Completeness:*** $\Pr[\langle \mathcal{P}, \mathcal{V} \rangle = 1 : \Delta(f_0, \texttt{RS}) = 0] = 1 - \theta$ *for a negligible θ. If $\theta = 0$ we refer to this as* perfect *completeness.*
– ***ϵ-soundness:*** *For any unbounded $\mathcal{P}^*$,* $\Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle = 1 : \Delta(f_0, \texttt{RS}) \geq \delta] \leq \epsilon$.

The next theorem summarizes the compilation results of [BGK+23]. Informally, given a δ-correlated hIOP, it suffices to analyse its RBR knowledge soundness when $\delta = 0$ and replace oracles with a δ-correlation check[7] to produce an RBR knowledge sound hIOP as a result. This hIOP can then be turned into a SNARK through the usual BCS transformation [BCS16]. Interestingly, having $\delta \neq 0$ (and actually up to the Johnson bound!) does not affect knowledge soundness when following the [BGK+23] recipe, whereas previous compilers [CMS19, COS20] were restricted to the unique decoding regime, i.e. $\delta < (1 - \rho)/2$.

[7] Such as batched FRI

Theorem 1 ([BGK+23]). *Let $\Pi_\delta^{\mathcal{O}}$ be a δ-correlated hIOP, where $\mathcal{O}$ is an oracle for δ-correlated agreement. Let $0 < \eta \leq 1$ and $\rho > 0$ be such that $\delta = 1 - \sqrt{\rho} - \eta$ is strictly positive. Assume $\Pi_0^{\mathcal{O}}$ has RBR knowledge soundness with error ϵ. Then, $\Pi_\delta^{\mathcal{O}}$ has RBR knowledge soundness with error $\epsilon/(2\eta\sqrt{\rho})$.*

Moreover if $\Pi_{\mathtt{CA}}$ is an IOPP for δ-correlated agreement in $\mathtt{RS}[\mathbb{F}, L, d]$ with RBR soundness error $\epsilon_{\mathtt{CA}}$, then the protocol $\Pi_\delta^{\Pi_{\mathtt{CA}}}$ obtained by replacing $\mathcal{O}$ with $\Pi_{\mathtt{CA}}$ in $\Pi_\delta^{\mathcal{O}}$ has RBR knowledge soundness error $\epsilon_1 = \max\{\epsilon/(2\eta\sqrt{\rho}), \epsilon_{\mathtt{CA}}\}$.

Furthermore, given a random oracle with λ-bit output and a query bound Q, compiling $\Pi_\delta^{\Pi_{\mathtt{CA}}}$ with the BCS transformation [BCS16] yields a SNARK with knowledge error $Q \cdot \max\{\epsilon/(2\eta\sqrt{\rho}), \epsilon_{\mathtt{CA}}\} + O(Q^2/2^\lambda)$.

3 Verifiable Computation over encrypted data

The notion of verifiable computation (VC) proposed by Gennaro et al. in [GGP10] tries to better capture the way in which proof and argument systems are used in practice. In Definition 1 we tweak their definition and syntax to fit our constructions. In particular, we allow for the verification algorithm to be interactive, since we will often discuss at the IOP rather than SNARK level of abstraction.

Furthermore, we would like to support circuits of the form $C(\mathbb{x}, \mathbb{w}_{\mathcal{P}})$ where (the homomorphic encryption of) the input $\mathbb{x}$ is provided by the verifier, while the prover specifies $\mathbb{w}_{\mathcal{P}}$, which includes plaintexts and/or ciphertexts revealed to $\mathcal{V}$ during verification. Without threshold decryption to prevent the verifier from unauthorized decryptions, all $\mathbb{w}_{\mathcal{P}}$ values are exposed to the verifier. Thus, for single-client outsourcing, $\mathbb{w}_{\mathcal{P}}$ should consist only of plaintexts.

If $\mathbb{w}_{\mathcal{P}}$ must remain hidden, it can be treated as a private witness mixed with the encryption $\mathbb{x}$ in zero-knowledge. For example, one could think about a private Machine-Learning-as-a-Service [AHH+24], where the client sends encrypted queries $\mathtt{HE.Enc}(\mathbb{x})$ to the server, who applies their private model $\mathbb{w}_{\mathcal{P}}$. This approach, also pursued in [BCFK21, GNS23], is advantageous because multiplying plaintexts (such as those in $\mathbb{w}_{\mathcal{P}}$) with ciphertexts (the encryptions of $\mathbb{x}$ and the outputs that result from operating on them) is much cheaper than multiplying ciphertexts.

Definition 1 (Verifiable Computation). *A verifiable computation scheme $\mathtt{VC}$ is a tuple of polynomial time algorithms* $(\mathsf{VC.Setup}, \mathsf{VC.ProbGen}, \mathsf{VC.Compute}, \mathsf{VC.Ver})$ *defined as follows.*

– $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{VC.Setup}(1^\lambda, C)$*: A randomized key generation algorithm takes a circuit C as input and outputs a secret key* SK *and a public key* PK.
– $(\boldsymbol{\sigma_x}, \mathsf{VK}_{\boldsymbol{x}}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{PK}, \boldsymbol{x})$*: A randomized problem generation algorithm (to be run by $\mathcal{V}$) takes the public key* PK*, an input $\boldsymbol{x}$, and outputs a public encoding $\boldsymbol{\sigma_x}$ of $\boldsymbol{x}$, together with a private verification key* $\mathsf{VK}_{\boldsymbol{x}}$.
– $\boldsymbol{\eta_y} \leftarrow \mathsf{VC.Compute}(\mathsf{PK}, \boldsymbol{\sigma_x}, \mathbb{w}_{\mathcal{P}}, C)$*: Given a public key* PK *for a circuit C, the encoded input $\boldsymbol{\sigma_x}$ and input $\mathbb{w}_{\mathcal{P}}$, $\mathcal{P}$ computes $\boldsymbol{\eta_y}$, which consists of an encoded version $\boldsymbol{\sigma_y}$ of the circuit's output $\boldsymbol{y} = C(\boldsymbol{x}, \mathbb{w}_{\mathcal{P}})$ and data to answer challenges about that statement.*
– $\mathtt{acc} \leftarrow \mathsf{VC.Ver}\langle\mathcal{P}(\mathsf{PK}, \boldsymbol{\eta}_y), \mathcal{V}(\mathsf{SK}, \mathsf{VK}_{\boldsymbol{x}}, \boldsymbol{\sigma_y})\rangle(C)$ *: The interactive verification algorithm uses the input-specific verification key* $\mathsf{VK}_{\boldsymbol{x}}$*, the setup secret key* SK *and a proof $\boldsymbol{\eta_y}$ to return $\boldsymbol{\sigma_y}$ together with a bit $\mathtt{acc} \in \{0, 1\}$ such that $\mathtt{acc} = 1$ if* $\mathsf{VC.Decode}(\boldsymbol{\sigma_y}, \mathsf{SK}) = C(\boldsymbol{x}, \mathbb{w}_{\mathcal{P}})$ *or $\mathtt{acc} = 0$ otherwise.*

– $\boldsymbol{y} \leftarrow \mathsf{VC.Decode}(\boldsymbol{\sigma_y}, \mathsf{SK})$*: Using the secret key, the decoding algorithm outputs the value* $\boldsymbol{y}$ *behind the public encoding* $\boldsymbol{\sigma_y}$*.*

A verifiable computation scheme can satisfy a range of properties which we next define. We omit the `VC.` prefix in the different algorithms for ease of readability. In our work, we will always be interested in all of the following ones whenever $\mathtt{w}_{\mathcal{P}}$ does not need to be kept private.

– *Correctness.* Correctness guarantees that if $\mathcal{P}$ is honest, the verification test will pass. That is, for all C, and for all valid inputs $\boldsymbol{x}, \mathtt{w}_{\mathcal{P}}$ of C the following probability equals $1 - \mathsf{negl}(\lambda)$.

$$\Pr\left(\begin{array}{c} \mathsf{acc} = 1 \\ \mathsf{Decode}(\boldsymbol{\sigma_y}, \mathsf{SK}) = C(\boldsymbol{x}, \mathtt{w}_{\mathcal{P}}) \end{array} : \begin{array}{c} (\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, C) \\ (\boldsymbol{\sigma_x}, \mathsf{VK}_{\boldsymbol{x}}) \leftarrow \mathsf{ProbGen}(\mathsf{PK}, \boldsymbol{x}) \\ \boldsymbol{\eta_y} \leftarrow \mathsf{Compute}(\mathsf{PK}, \boldsymbol{\sigma_x}, \mathtt{w}_{\mathcal{P}}, C) \\ \mathsf{acc} \leftarrow \mathsf{Ver}\langle \mathcal{P}(\mathsf{PK}, \boldsymbol{\eta}_y), \mathcal{V}(\mathsf{SK}, \mathsf{VK}_{\boldsymbol{x}}, \boldsymbol{\sigma_y})\rangle(C) \end{array}\right)$$

– *Outsourceability.* A VC scheme is outsourceable if for any $\boldsymbol{x}$ and any $\boldsymbol{\eta_y}$, the time required by $\mathcal{V}$ to run $\mathsf{ProbGen}(\boldsymbol{x})$, $\mathsf{Ver}\langle \mathcal{P}(\mathsf{PK}, \boldsymbol{\eta}_y), \mathcal{V}(\mathsf{SK}, \mathsf{VK}_{\boldsymbol{x}})\rangle(C)$ and $\mathsf{Decode}(\boldsymbol{\sigma_y}, \mathsf{SK})$ is $o(T)$, where T is the time required to compute $C(\boldsymbol{x}, \mathtt{w}_{\mathcal{P}})$.
– *ϵ-Soundness.* A VC scheme is ϵ-sound if a malicious $\mathcal{P}$ cannot make the verification algorithm accept an incorrect answer for any valid circuit C. That is, a scheme is sound if the advantage of any PPT adversary $\mathcal{A}$ in the game $\mathsf{Exp}_{\mathcal{A}}^{Ver}$ defined as $\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{Ver}[VC, C, \lambda] = 1\right]$ is ϵ.
– *Verifier-Privacy.* For any valid circuit C,

$$\Pr\left[\mathsf{Exp}_{\mathcal{A}}^{\mathcal{V}.Priv}[VC, C, \lambda] = 1\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

```
1  Game $\mathsf{Exp}_{\mathcal{A}}^{Ver}(VC, C, \lambda)$
2      $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, C)$
3      $\boldsymbol{x} \leftarrow \mathcal{A}(\mathsf{PK}, C)$
4      $(\boldsymbol{\sigma_x}, \mathsf{VK}_{\boldsymbol{x}}) \leftarrow \mathsf{ProbGen}(\mathsf{PK}, \boldsymbol{x})$
5      $\boldsymbol{\eta_y} \leftarrow \mathcal{A}(\mathsf{PK}, \boldsymbol{\sigma_x}, C)$
6      $\mathsf{acc} \leftarrow$
         $\mathsf{Ver}\langle \mathcal{P}(\mathsf{PK}, \boldsymbol{\eta}_y), \mathcal{V}(\mathsf{SK}, \mathsf{VK}_{\boldsymbol{x}}, \boldsymbol{\sigma_y})\rangle(C)$
7      $\boldsymbol{y} \leftarrow \mathsf{Decode}(\boldsymbol{\sigma_y}, \mathsf{SK})$
8      if $acc = 1 \wedge \nexists \mathtt{w}_{\mathcal{P}} : C(\boldsymbol{x}, \mathtt{w}_{\mathcal{P}}) = \boldsymbol{y}$
9          return 1
10     return 0
```

```
1  Game $\mathsf{Exp}_{\mathcal{A}}^{\mathcal{V}.Priv}(VC, C, \lambda)$
2      $b \xleftarrow{\$} \{0, 1\}$
3      $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, C)$
4      $(\boldsymbol{x}_0, \boldsymbol{x}_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{PK}, C)$
5      $(\boldsymbol{\sigma}_{\boldsymbol{x}_b}, \mathsf{VK}_{\boldsymbol{x}_b}) \leftarrow \mathsf{ProbGen}(\mathsf{PK}, \boldsymbol{x}_b)$
6      $\hat{b} \leftarrow \mathcal{A}(\mathsf{state}, \boldsymbol{\sigma}_{\boldsymbol{x}_b})$
7      return $b \overset{?}{=} \hat{b}$
```

3.1 On verifier privacy and oracle attacks

Our definition of verifier-privacy assumes that the verifier does not signal to the prover whether verification passes or not (i.e. there are no verification oracles) and that the value $\mathtt{y}$ it obtains after running the Decode algorithm is not used as input to future algorithms (which we refer to as a decryption oracle). We briefly discuss the privacy impact of such oracles, which is inherent to moving verification from the ciphertext to the plaintext layer.

Verification oracles: When a malicious prover provides tampered ciphertexts (e.g. by manipulating their noise), the result of decryption is a function of such changes. The results of decryption also depend on the secret key and underlying message. Since the actions of the verifier (such as rejecting a proof or not, i.e. the value $\mathtt{acc} \in \{0,1\}$) depend on the decrypted ciphertexts, this is a source of leakage. All protocols we present are non-interactive, implying that there is at most one such verification oracle per proof sent. Hence, a malicious prover can learn at most a one-bit leakage predicate (which evaluates to $\mathtt{acc}$) about the secret key/message. Regarding leakage about the key, the concrete security loss this incurs can be measured with tools such as the Leaky Estimator [DDGR20]. Additionally, stopping any further executions with a prover for which $\mathtt{acc} = 0$, refreshing the keys and/or increasing their length are easy solutions.

Decryption oracles: First, note that the output of the decryption of the prover's answers to verifier challenges has no use beyond verification. Therefore, the only value we need to be concerned about in a decryption oracle is the output of the computation itself. Consequently, decryption oracles may only occur in the composability case – i.e., when $\mathsf{VC.Decode}(\boldsymbol{\sigma_y}, \mathsf{SK})$ is used as input in another protocol. Further, these decryption oracles arise only in the case where verification passes, as the Verifier would otherwise abort (and then, at worst, we would be in the presence of a verification oracle, as explained above). Finally, we note that the output of the computation ($C(\boldsymbol{x}, \mathbb{w}_{\mathcal{P}})$) leaks nothing unexpected, as long as $\mathbb{w}_{\mathcal{P}}$ does not contain ciphertexts for which $\mathcal{P}$ did not provide a zero knowledge proof of knowledge (ZKPoK) for the underlying plaintext. In summary, the only scenario in which a decryption oracle may pose a threat is: the prover provides as input ciphertext without an accompanying ZKPoK, verification passes and the HELIOPOLIS output is passed along to another protocol for further processing.

3.2 Prover privacy

In previous works on verifiable computation over encrypted data, the goal of keeping $\mathbb{w}_{\mathcal{P}}$ private was modeled as a *context-hiding* property of the $\mathtt{VC}$ scheme [BCFK21, GNS23]. This is a reminiscence of a similar notion in the setting where $\mathbb{w}_{\mathcal{P}}$ did not exist, but the parties running $\mathsf{VC.ProbGen}$ and $\mathsf{VC.Ver}$ were different [FNP20]. In our following sections we will deviate from that modeling and hence refrain from the context-hiding property. We do this not only because we focus on the more common scenario where the verifier is running both $\mathsf{VC.ProbGen}$ and $\mathsf{VC.Ver}$, but also because context-hiding would not be able to model e.g. the interactivity of $\mathsf{VC.Ver}$. We believe that our new security modeling will be useful for future work in this area.

We formalize the notion of *honest-verifier prover privacy* (HVPP) by showing that whatever a semi-honest $\mathcal{V}$ can compute by participating in the protocol, $\mathcal{V}$ could compute merely from its input and prescribed output. Our definition is in the simulation paradigm and thus we have a stateful simulator $\mathcal{S}$ that generates $\mathcal{V}$'s view given its input and output. We remark that, since $\mathcal{V}$ is semi-honest, it is guaranteed that it uses its actual input and random tapes. In particular, $\mathcal{S}$ can furthermore generate $\mathcal{V}$'s random tape and, at that point, generate the whole protocol transcript on its own without ever needing to interact with $\mathcal{V}$.

Definition 2. *We say that a $\mathtt{VC}$ protocol is* honest-verifier prover-private *(HVPP) if there exists a PPT simulator $\mathcal{S}$ such that for every circuit C:*

$$\{\mathcal{S}(1^\lambda, \mathsf{PK}, \boldsymbol{x}, C(\boldsymbol{x}, \mathbb{w}_{\mathcal{P}}))\}_{\boldsymbol{x}, \mathbb{w}_{\mathcal{P}}, \lambda, \mathsf{PK}} \overset{c}{\approx} \{\mathbf{View}_{\mathcal{V}}(\mathsf{SK}, \mathsf{PK}, \mathbf{x}, \mathbb{w}_{\mathcal{P}}, \lambda)\}_{\mathbf{x}, \mathbf{y}, \lambda, \mathsf{SK}, \mathsf{PK}}$$

where $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{VC.Setup}(1^\lambda, C)$ *and* $\mathbf{View}_{\mathcal{V}}(\mathsf{SK}, \mathsf{PK}, \mathbf{x}, \mathbb{w}_{\mathcal{P}}, \lambda)$ *denotes the view of $\mathcal{V}$ during an execution of the protocol on inputs $(\boldsymbol{x}, \mathbb{w}_{\mathcal{P}})$ and security parameter λ, that is* $(\mathsf{SK}, \mathsf{PK}, \boldsymbol{x}, \boldsymbol{r}; m_1, \ldots, m_e, \boldsymbol{out})$

where $\boldsymbol{r}$ *is* $\mathcal{V}$*'s random tape, each* m_i *value is the* i*-th message* $\mathcal{V}$ *receives and* $\texttt{out}$ *denotes* $\mathcal{V}$*'s output, which is computed from all other values in its own view of the execution.*

In Figure 1, we provide our general recipe for a correct, sound and verifier-private Verifiable Computation scheme. Verifier-privacy follows from the use of encryption within the ProbGen step, and correctness from the fact that such encryption is homomorphic. Soundness is less immediate, since it requires to have an HE-IOP at hand, which is an object we define and construct in Section 4. Notice that, since we specialize the construction to use *holographic* IOPs (as a means to achieve outsourceability), the syntax of the verification is slightly modified, replacing the circuit C with the corresponding indexer algorithm.

Verifiable Computation over encrypted data

Let IOP be an holographic Interactive Oracle Proof. Let HE be an exact homomorphic encryption scheme with plaintext space R_p. Let $\phi : R_p \to \bigotimes_{i=1}^{\ell} \mathbb{F}_{p^D}$ be the CRT isomorphism. Let $C : \mathbb{F}_p^{inp} \times \mathbb{F}_p^{wit} \to \mathbb{F}_p^{out}$ be an arithmetic circuit that we want to verify. Let $L_x = \lceil inp/\ell \rceil$, $L_w = \lceil wit/\ell \rceil$.

$(\mathsf{sk}, (\mathsf{pk}, \mathsf{evk})) \leftarrow \mathsf{Setup}(1^\lambda, C)$: Run HE setup:

1. $\mathcal{V}$ determines a set of admissible circuits $\widehat{\mathcal{C}irc}$ which contains a circuit $\widehat{C}$ that homomorphically computes C.
2. $\mathcal{V}$ runs $(\mathsf{pp}, R_q^2) \leftarrow \mathtt{HE.Setup}(1^\lambda, R_p, \widehat{\mathcal{C}irc})$.
3. $\mathcal{V}$ runs $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathtt{HE.KeyGen}(1^\lambda, \mathcal{C}, \mathsf{pp})$.
4. Given an index $\mathbb{i}$ for the circuit C, the indexer $\mathtt{Ind}$ computes an encoding of it, $\mathtt{Ind}(\mathbb{i})$.

$(\boldsymbol{\sigma}_{\mathbb{x}}, \mathsf{vk}_{\mathbb{x}}) \leftarrow \mathsf{ProbGen}(\mathbb{x}, \mathsf{pk})$: $\mathcal{V}$ parses $\mathbb{x} = (x_0, \ldots, x_{inp-1}) \in \mathbb{F}_p^{inp}$. For $i = 0, \ldots, L_x - 1$, let $m_{\mathbb{x},i} = \phi^{-1}(x_{i\ell}, \ldots, x_{(i+1)\ell-1})$. Encrypt these inputs as $\boldsymbol{\sigma}_{\mathbb{x}} = \{\mathtt{HE.Enc}(\mathsf{pk}, m_{\mathbb{x},i})\}_{i=0}^{L_x-1}$. Set $\mathsf{vk}_{\mathbb{x}} = (\mathsf{sk}, \mathbb{x})$.

$\mathbb{w} \leftarrow \mathsf{Compute}(\mathsf{evk}, \boldsymbol{\sigma}_{\mathbb{x}}, \mathbb{w}_{\mathcal{P}}, \widehat{C})$: $\mathcal{P}$ parses $\boldsymbol{\sigma}_{\mathbb{x}}$ and evaluates $\widehat{C}(\boldsymbol{\sigma}_{\mathbb{x}}, \mathbb{w}_{\mathcal{P}})$, by which it obtains the rest of the witness: the values on intermediate wires $\mathbb{w}_C$ and the circuit output $\boldsymbol{\sigma}_{\mathbb{y}}$. Notice that the whole witness becomes $\mathbb{w} = (\mathbb{w}_{\mathcal{P}}, \mathbb{w}_C, \boldsymbol{\sigma}_{\mathbb{y}})$, which is a mix of plaintext values (such as $\mathbb{w}_{\mathcal{P}}$) and ciphertext values (those depending on any input $\boldsymbol{\sigma}_{\mathbb{x}}$).

$\mathsf{acc} \leftarrow \mathsf{Ver}\langle \mathcal{P}(\mathtt{Ind}(\mathbb{i}), \mathsf{evk}, (\boldsymbol{\sigma}_{\mathbb{x}}, \mathbb{w})), \mathcal{V}^{\mathtt{Ind}(\mathbb{i})}(\mathsf{sk}, (\mathsf{vk}_{\mathbb{x}}, \boldsymbol{\sigma}_{\mathbb{y}}))\rangle$: $\mathcal{P}$ and $\mathcal{V}$ run the HE-IOP corresponding to the IOP for the plaintext circuit C, i.e. $\langle \mathcal{P}(\mathtt{Ind}(\mathbb{i}), \mathsf{evk}, \boldsymbol{\sigma}_{\mathbb{x}}, \mathbb{w}), \mathcal{V}^{\mathtt{Ind}(\mathbb{i})}(\mathsf{sk}, (\mathbb{x}, \mathtt{HE.Dec}(\mathsf{sk}, \boldsymbol{\sigma}_{\mathbb{y}})))\rangle$.

$\mathbb{y} \leftarrow \mathsf{Decode}(\boldsymbol{\sigma}_{\mathbb{y}}, \mathsf{sk})$: $\mathcal{V}$ outputs $\mathbb{y} = \mathtt{HE.Dec}(\mathsf{sk}, \boldsymbol{\sigma}_{\mathbb{y}})$.

Fig. 1: Verifiable Computation over Encrypted Data through HE-IOPs.

4 Compiling Interactive Oracle Proofs to work over HE

Given an IOP which was not conceived to work over encrypted data, we show how to adapt it to work with HE in Definition 13.

Definition 13 (HE-transformation) *Let* $\langle \mathcal{P}(\mathbb{x}, \mathbb{w}), \mathcal{V}(\mathbb{x})\rangle$ *be an IOP, where the elements of* $\mathbb{x}$ *and* $\mathbb{w}$ *belong to a finite field* $\mathbb{F}$*. We define its* encrypted version *HE-IOP, for some* $\mathtt{HE}$ *scheme as follows:*

- *There is a trusted setup* $(\mathsf{pp}, \mathcal{C}) \leftarrow \mathtt{HE.Setup}(1^\lambda, R_p, \widehat{\mathcal{C}\mathrm{irc}})$*, where* R_p *splits into copies of* $\mathbb{F}$ *and* $\widehat{\mathcal{C}\mathrm{irc}}$ *is a family of admissible circuits that captures all necessary computation within the IOP as well as any preceding/posterior one (such as coming up with parts of the witness, or using an IOPP and the BCS transformation to compile into a SNARK).*
- *There is also a trusted key generation step* $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathtt{HE.KeyGen}(1^\lambda, \mathcal{C}, \mathsf{pp})$*.* $\mathcal{P}$ *has as an additional input* evk *and* $\mathcal{V}$ *has as an additional input* sk*.*

– *$\mathcal{P}$'s input $\mathbb{x}$ is replaced by its encryption $\texttt{HE.Enc}(\mathbb{x})$. Parts of $\mathbb{w}$ could be also replaced by their homomorphic encryption.*
– *As a result, some oracles in the HE-IOP might now contain ciphertexts. We refer to them as* HE-oracles *or* encrypted oracles. *$\mathcal{V}$ has to decrypt the ciphertexts obtained from HE-oracles and perform the same checks as in $\mathbb{F}$.*

If we need to refer explicitly to the HE-IOP, we denote it as $\langle \mathcal{P}(\texttt{evk}, \texttt{HE.Enc}(\mathbb{x}), \mathbb{w}), \mathcal{V}(\texttt{sk}, \mathbb{x})\rangle$, as a slight abuse of notation of the original $\mathcal{P}$ and $\mathcal{V}$. The different properties of IOPs (completeness, soundness, round-by-round soundness, round-by-round knowledge soundness) can be trivially redefined for HE-IOPs.

One of the main interests of our HE-transformation, besides its simplicity, is that it preserves most parameters of the original IOP, with only some negligible degradation due to the use of homomorphic encryption.

Theorem 2. *Let $\texttt{IOP}$ be an ϵ_k RBR knowledge sound, $\epsilon_{\texttt{rbr}}$ RBR sound, ϵ-sound, complete IOP. It's encrypted version $\texttt{HE-IOP}$ is $\epsilon_k + \mathsf{negl}(\lambda)$ RBR knowledge sound, $\epsilon_{\texttt{rbr}} + \mathsf{negl}(\lambda)$ RBR sound, $\epsilon + \mathsf{negl}(\lambda)$-sound and complete.*

Proof. Completeness follows from the evaluation correctness of the $\texttt{HE}$ scheme and the way the circuit family $\widehat{\mathcal{C}irc}$ was chosen. There is only a negligible loss in γ due to the way evaluation correctness is defined (Definition 2).

The soundness, RBR soundness and RBR (knowledge) soundness of an $\texttt{HE-IOP}$ can be reduced to that of $\texttt{IOP}$ as follows. Let $\mathcal{A}$ be an adversary against $\texttt{HE-IOP}$ with an advantage bigger than adding a factor $\mathsf{negl}(\lambda)$ to the one for the corresponding notion of the $\texttt{IOP}$ ($\epsilon, \epsilon_{\texttt{rbr}}, \epsilon_k$ respectively). We will build an adversary $\mathcal{A}'$ against $\texttt{IOP}$ with the same such greater advantage and hence reach a contradiction. $\mathcal{A}'$ runs $(\texttt{sk}, \texttt{pk}, \texttt{evk}) \leftarrow \texttt{HE.KeyGen}(1^\lambda, \mathcal{C}, \texttt{pp})$ for the relevant $\texttt{HE}$ scheme, obtaining in particular $\texttt{sk}$. Given any input, $\mathcal{A}'$ encrypts it under $\texttt{pk}$ and forwards it to $\mathcal{A}$. For every message received from $\mathcal{V}$, $\mathcal{A}'$ directly forwards it to $\mathcal{A}$. In order to reply to those, $\mathcal{A}'$ queries the encrypted oracles $[\![f]\!]_{\texttt{HE}}$ received from $\mathcal{A}$ at every point, decrypts the answers using $\texttt{sk}$ to recover f, and then sends the oracle $[\![f]\!]$ to $\mathcal{V}$. Clearly, if $\mathcal{A}$ succeeds, so does $\mathcal{A}'$. ∎

The HE-transformation applies to all variants of IOPs presented in this paper, such as holographic IOPs, RS-encoded hIOPs, δ-correlated hIOPs and IOPs of proximity. In order to denote this transformation, we will also add the HE prefix to those (HE-hIOPs, δ-correlated HE-hIOPs, HE-IOPP, etc).

The upcoming subsections are organized as follows. In Section 4.1, we discuss how to keep $\mathbb{w}$ hidden from the verifier through zero knowledge. Sections 4.2 and 4.3 show how to compile these HE-IOPs into $\texttt{HE}$-friendly SNARKs using an $\texttt{HE}$-friendly low degree test (such as the $\texttt{HE}$ transformation of the Batched FRI protocol, which we will show in Section 5.1).

4.1 Achieving prover-privacy from ZK-IOPs

We first consider the case of honest-verifier prover-privacy (HVPP, see Definition 2), since it allows for a more practical construction and it also acts as a stepping stone towards understanding the malicious case. There are three main aspects to consider when compiling using IOPs for a prover-private version of Figure 1, which we describe next. Two of them (Consideration #1 and #3) are specific to the use of homomorphic encryption.

Consideration #1: Circuit privacy. A requirement for prover-private constructions is the fact that the HE scheme needs to support *circuit-privacy*. Namely, all the ciphertexts of the HE-IOP that are exposed to the verifier need to be re-randomized, since their noise carries information about the circuit that was computed on them and hence[8] about $\boldsymbol{w}_{\mathcal{P}}$. We provide our own definition of the circuit-privacy notion that is best aligned with our HVPP goal.

Definition 14 *A homomorphic encryption scheme* $\mathtt{HE}$ *(see Definition 1) is* circuit-private *if there exists a rerandomization algorithm* $\mathtt{HE.Rerand}(\mathtt{evk}, \mathtt{pk}, \hat{C}, \mathtt{ct})$ *and a simulator* $\mathcal{S}_{\mathtt{HE}}$ *such that, for any admissible circuit* $\hat{C}$ *with inputs* $\mathtt{HE.Enc}(x_1), \ldots, \mathtt{HE.Enc}(x_n)$ *and outputs* $\mathtt{ct}_{y_1}, \ldots, \mathtt{ct}_{y_m}$*, it holds that (some inputs omitted for simplicity):*

$$(\mathtt{sk}, \mathtt{HE.Rerand}(\mathtt{ct}_{y_1}), \ldots, \mathtt{HE.Rerand}(\mathtt{ct}_{y_m})) \overset{\mathrm{c}}{\approx} (\mathtt{sk}, \mathcal{S}_{\mathtt{HE}}(\mathtt{pk}, C(x_1, \ldots, x_n))).$$

One of the standard ways that the above definition can be achieved is by employing noise flooding to instantiate $\mathtt{HE.Rerand}$, as done in [Gen09a]. In more detail, we add to the verifier-exposed ciphertexts an encryption of 0 with large enough noise to statistically hide the noise of the circuit that led to the production of that specific ciphertext. We will denote by $\Omega_{0,C}$ the set of such encryptions of zero. Notice that since all messages within an encrypted oracle are susceptible of being queried, we need to add such an encryption of zero to each of them *before* putting them within the oracle.

Consideration #2: Combining zkIOPs with LDTs. Assume either a zero-knowledge RS-hIOP or a δ-correlated hIOP is given. To compile it into a zk-IOP while making black-box use of a pre-existing Low Degree Test (in the form of an IOPP such as FRI), it is necessary for the prover to additionally send a random codeword r ahead of time, which is added to the linear combination of functions that are being tested for low-degreeness. Adding such an r does not affect soundness. However, since input to the LDT is now a *random* codeword, there is no need to worry about its inner workings beyond knowing what is the amount of queries made to the random codeword (which links with Consideration #3). For a more detailed leakage analysis when using FRI, see [Hab22].

Consideration #3: Combining zkIOPs with LDTs – query blow-up from packing. Compilers, such as the ones discussed in Consideration #2 and the one presented in Section 4.2, incur losses in several parameters of the resulting output IOP according to the number of queries to the LDT. This includes soundness, which in turn also loops into increasing the size of the underlying field in order to compensate. But, most importantly, the increase in the number of queries through the introduction of the LDT also degrades the query bound for zero knowledge (see Definition 11). As an example, see [COS20, Theorem 8.1.]

To make things worse, the HE-transformation of these protocols replaces oracles with *encrypted oracles*, where ciphertexts (rather than plaintexts) are placed within them. This means that, if the chosen HE scheme supports plaintext packing and we are exploiting this property, whenever the verifier $\mathcal{V}$ would need to query only one of the plaintexts m_i on the ciphertext $\mathtt{ct} = \mathtt{HE.Enc}(m_1, \ldots, m_l)$ behind the oracle, $\mathcal{V}$ learns every other plaintext $m_j, j \neq i$ within it. Effectively, this blows-up the query loss for zero knowledge by a factor of up to l.[9] Packing becomes then as devastating (or

[8] Notice that one can think about the circuit evaluation $C(\boldsymbol{x}, \boldsymbol{w}_{\mathcal{P}})$ with a private $\boldsymbol{w}_{\mathcal{P}}$ as providing the evaluation of some unspecified circuit from the family $\{C_{\boldsymbol{w}_{\mathcal{P}}}(\boldsymbol{x})\}_{\boldsymbol{w}_{\mathcal{P}}}$.

[9] It could be that $\mathcal{V}$ sometimes happens to query values that happen to be packed within the same ciphertext, slightly reducing the blow-up in this case.

even more) for zero knowledge as it is an improvement for computational efficiency, which is a very problematic tension in practice. Hence, it is paramount to reduce the packing-induced multiplicative loss while maintaining efficiency. We provide a solution for this in Section 6.5.

Malicious verifier We will only briefly address handling a malicious verifier. Instead of creating an ad-hoc "malicious-verifier prover-private" notion, it is best to model security as a maliciously secure 2-party computation protocol (see [CCL15]). Beyond the precautions for an honest verifier, we must ensure honest behavior in the Setup and ProbGen steps. This can be achieved through a trusted setup and enforced via zero-knowledge proofs, similar to a GMW-style compiler from passive to active security [GMW87].

For the ProbGen step, it is crucial to ensure the verifier provides valid ciphertexts, meaning the noise must be within bounds in lattice-based HE. Using a zero-knowledge proof of knowledge (ZKPoK) ensures the right bounds for the cleartext and encryption randomness (see e.g. [DPSZ12, Figure 9]).

4.2 A compiler for RS-encoded IOPs

Our first compiler is for Reed-Solomon encoded IOPs, and is a result of adapting the works of Aurora [BCR+19] and Fractal [COS20].

Protocol 1 (Aurora/Fractal) *Let $(\mathcal{P}_{\mathcal{R}}(\mathbb{x}, \mathbb{w}), \mathcal{V}_{\mathcal{R}}(\mathbb{x}))$ be an RS-encoded hIOP over $L \subseteq \mathbb{F}$, with maximum degree (d_c, d_e) for an indexed relation $\mathcal{R}$. Let $\texttt{HE-IOP}$ be its HE-transformation. Let $(\mathcal{P}_{\texttt{LDT}}, \mathcal{V}_{\texttt{LDT}})$ be an IOPP for the $\texttt{RS}$ code $\texttt{RS}[\mathbb{F}, L, d_c]$ with proximity parameter $\delta < \min(\frac{1-2\rho_c}{2}, \frac{1-\rho_c}{3}, 1-\rho_e)$ where $\rho_c = (d_c+1)/|L|$ and $\rho_e = (d_e+1)/|L|$. Let $\texttt{HE-IOPP}$ be its HE-transformation. Proceed as follows:*

1. ***Masking codeword for low-degree test:*** *$\mathcal{P}$ sends $\mathcal{V}$ an oracle to a random $r \in \texttt{RS}[\mathbb{F}, L, d_c]$. This step can be skipped when not interested in obtaining a zk-HE-hIOP.*
2. ***$\texttt{RS}$-encoded $\texttt{HE-IOP}$ for $\mathcal{R}$:*** *In parallel to the above, $\mathcal{P}$ and $\mathcal{V}$ simulate $(\mathcal{P}_{\mathcal{R}}(\texttt{HE.Enc}(\mathbb{x}), \mathbb{w}), \mathcal{V}_{\mathcal{R}}(\mathbb{x}))$. Over the course of this protocol, the prover sends encrypted oracles containing codewords $\pi_1 \in \texttt{RS}[\mathbb{F}, L, \boldsymbol{d}_1], \ldots, \pi_{k^{\mathcal{R}}} \in \texttt{RS}[\mathbb{F}, L, \boldsymbol{d}_{k^{\mathcal{R}}}]$, and the verifier specifies a set of rational constraints $\mathfrak{C}$ [COS20, Definition 4.1]. Let $l := \sum_{i=1}^{k^{\mathcal{R}}} l_i + |\mathfrak{C}|$.*
3. ***Random linear combination:*** *$\mathcal{V}$ samples $\boldsymbol{v} \in \mathbb{F}^{2l}$ uniformly at random and sends it to $\mathcal{P}$*
4. ***Low-degree test through $\texttt{HE-IOPP}$:*** *$\mathcal{P}$ and $\mathcal{V}$ simulate $(\mathcal{P}_{\texttt{LDT}}(\boldsymbol{v}^\top \Pi + r), \mathcal{V}_{\texttt{LDT}}^{\boldsymbol{v}^\top \Pi + r})$, where $\Pi := \binom{\Pi_0}{\Pi_1} \in \mathbb{F}^{2l \times L}$ is defined as in [BCR+19, Protocol 8.2].*
5. *$\mathcal{V}$ accepts if and only if $\mathcal{V}_{\texttt{LDT}}$ accepts*

Theorem 3. *Protocol 1 is an HE-hIOP for $\mathcal{R}$ with the following parameters, where the $\mathcal{R}$ (resp. $\texttt{LDT}$) superscript denotes the parameters of the RS-encoded IOPP (resp. IOPP):*

- *Round complexity: $k^{\mathcal{R}} + k^{\texttt{LDT}}$.*
- *Query complexity: $q_\pi^{\texttt{LDT}} + q_{\mathbb{w}}^{\texttt{LDT}}(k^{\mathcal{R}} + 1)$.*
- *Proof length $\texttt{HE.Expand}(L^{\mathcal{R}} + L^{\texttt{LDT}})$, where $\texttt{HE.Expand}$ is a ciphertext expansion function that depends on the specific $\texttt{HE}$ scheme and how the different intermediate values are computed.*
- *Round-by-round soundness error: $\epsilon_1 = \max(\epsilon_{\texttt{rbr}}^{\mathcal{R}}, \epsilon_{\texttt{rbr}}^{\texttt{LDT}}, |L|/|\mathbb{F}|)$.*
- *Round-by-round knowledge error: $\epsilon_2 = \max(\epsilon_{\texttt{knw}}^{\mathcal{R}}, \epsilon_{\texttt{rbr}}^{\texttt{LDT}}, |L|/|\mathbb{F}|)$.*

Furthermore, if the RS-encoded IOP is zero-knowledge, then so is Protocol 1, with the same query bound.

Proof. All the claimed parameters can be reduced to the ones claimed in [COS20, Theorem 8.2]. The round and query complexity clearly remain the same as in there, and the proof length is only affected by the ciphertext expansion of the HE scheme. Completeness and RBR (knowledge) soundness follow from Theorem 2 and [COS20, Theorem 8.2]. ∎

4.3 A correlated-agreement-based compiler

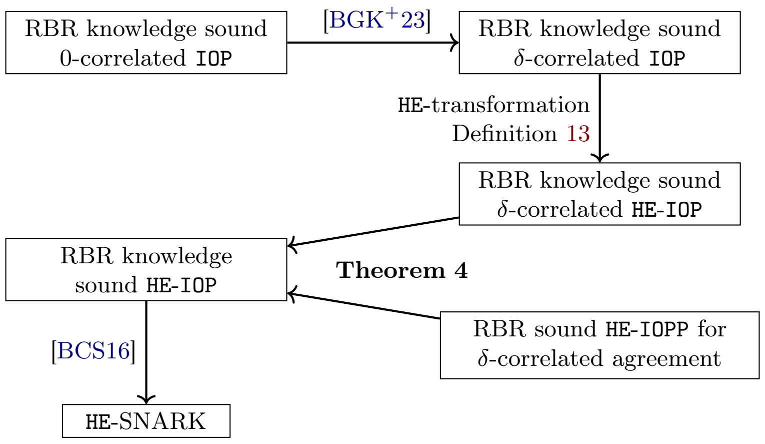
Our second compiler allows to set the proximity parameter up to the Johnson bound, which improves efficiency. It is the result of adapting one of the central theorems in [BGK$^+$23] through the application of the HE transformation (Definition 13). The overall compiler appears in Figure 2.

Theorem 4. *Let $\Pi_\delta^{\mathcal{O}}$ be a δ-correlated HE-hIOP, where $\mathcal{O}$ is an HE-oracle for δ-correlated agreement in $\mathtt{RS}[\mathbb{F}, L, d]$. Let $0 < \eta \leq 1$ and $\rho > 0$ be such that $\delta = 1 - \sqrt{\rho} - \eta$ is strictly positive. Assume $\Pi_0^{\mathcal{O}}$ has RBR knowledge soundness with error ϵ. Then, $\Pi_\delta^{\mathcal{O}}$ has RBR knowledge soundness with error $\epsilon/(2\eta\sqrt{\rho})$.*

Let $\mathtt{HE}$ be an homomorphic encryption scheme whose plaintext space R_p splits into copies of $\mathbb{F}$. If $\Pi_{\mathtt{HE\text{-}CA}}$ is an HE-IOPP for δ-correlated agreement in $\mathtt{RS}[\mathbb{F}, L, d]$ with RBR soundness error $\epsilon_{\mathtt{CA}}$, then the protocol $\Pi_\delta^{\Pi_{\mathtt{HE\text{-}CA}}}$ obtained by replacing $\mathcal{O}$ with $\Pi_{\mathtt{HE\text{-}CA}}$ in $\Pi_\delta^{\mathcal{O}}$ has RBR knowledge soundness error $\epsilon_1 = \max\{\epsilon/(2\eta\sqrt{\rho}), \epsilon_{\mathtt{CA}}\}$.

Furthermore, given a random oracle with λ-bit output and a query bound Q, compiling $\Pi_\delta^{\Pi_{\mathtt{HE\text{-}CA}}}$ with the BCS transformation [BCS16] yields a SNARK (over encrypted data) with knowledge error $Q \cdot \max\{\epsilon/(2\eta\sqrt{\rho}), \epsilon_{\mathtt{CA}}\} + O(Q^2/2^\lambda)$.

Proof. Consequence of combining Theorems 2 and 1. ∎



RBR knowledge sound 0-correlated IOP
[BGK+23]
RBR knowledge sound δ-correlated IOP
HE-transformation Definition 13
RBR knowledge sound δ-correlated HE-IOP
RBR knowledge sound HE-IOP
Theorem 4
RBR sound HE-IOPP for δ-correlated agreement
[BCS16]
HE-SNARK

Fig. 2: Summary of compilation flow for δ-correlated IOPs.

5 Low Degree Tests for encrypted polynomials

The compilers from Section 4 need to eventually test whether the oracles sent by the IOP prover correspond to low-degree polynomials or not, with different variations of what such a test should exactly verify (δ-correlated agreement or merely closeness to an RS code). First of all, we need to think about how polynomials mix with $\texttt{HE}$. For example, the following map

$$\begin{aligned} f : R_p &\to R_p \\ a &\mapsto \texttt{HE.Dec}(\sum_{i=0}^{d} \texttt{ct}_i \cdot a^i), \quad \texttt{ct}_i = \texttt{HE.Enc}(f_i) \end{aligned}$$

only corresponds to a degree-d polynomial $f \in R_p[\mathtt{X}]$ as long as $f(a) = \sum_{i=0}^{d} f_i a^i \quad \forall a \in R_p$, i.e. as long as it preserves evaluation correctness[10].

In this work, as it is common in the IOP literature, polynomials are given in a point-value representation, which matches the definition of a Reed Solomon codeword. There are a series of operations that the prover (and maybe the verifier) will have to perform on the ciphertexts within those oracles, so we also need to make sure to preserve evaluation correctness when presented with such a representation. To achieve this, we introduce the notion of *encrypted polynomials.*

Definition 15 *Let $\texttt{HE}$ be a homomorphic encryption scheme with plaintext space $R_p \cong \prod_{j=1}^{\ell} \mathbb{F}_{p^D}$ and ciphertext space R_q^2. Let $\texttt{HE-IOPP}$ be an HE-IOP of proximity. Let $L = \{x_{i,j}\}_{i\in[d],j\in[\ell]}$, $L \subseteq \mathbb{F}_{p^D}$ and let $\texttt{ct}_1, \ldots, \texttt{ct}_d \in R_q^2$ be alleged ciphertexts such that $\texttt{HE.Dec}(\texttt{ct}_i) = (m_{i,1}, \ldots, m_{i,\ell}) \in \prod_{j=1}^{\ell} \mathbb{F}_{p^D}$. Finally, let $f \in \mathbb{F}_{p^D}[\mathtt{X}]_{<|L|}$ be the polynomial such that $f(x_{i,j}) = m_{i,j} \in \mathbb{F}_{p^D}$ for every $x_{i,j} \in L$. We say that $\texttt{ct}_1, \ldots, \texttt{ct}_d \in R_q^2$ define an* encrypted polynomial (of f, at L) *if there exist admissible circuits such that,*

- *On input $\texttt{ct}_1, \ldots, \texttt{ct}_d \in R_q^2$ and any $(\alpha_1, \ldots, \alpha_\ell) \in \prod_{j=1}^{\ell} \mathbb{F}_{p^D}$, it returns a ciphertext $\texttt{ct}'$ such that, with overwhelming probability, $\texttt{HE.Dec}(\texttt{ct}') = (f(\alpha_1), \ldots, f(\alpha_\ell)) \in \prod_{j=1}^{\ell} \mathbb{F}_{p^D}$.*
- *On input $\texttt{ct}_1, \ldots, \texttt{ct}_d \in R_q^2$ to the $\texttt{HE-IOPP}$, all honestly produced messages within it decrypt correctly (with overwhelming probability).*

When we want to make the plaintext polynomial and evaluation domain explicit, we write $(\texttt{ct}_1, \ldots, \texttt{ct}_d) \in \mathsf{EncPoly}(f, L)$.

In other words, $(\texttt{ct}_1, \ldots, \texttt{ct}_d) \in \mathsf{EncPoly}(f, L)$ if, given those ciphertexts, it is possible both to compute $\mathsf{EncPoly}(f, \mathbb{F}_{p^D})$ and to show within the HE-IOPP that there exists such an f.

5.1 The HE-Batched-FRI protocol

The specific HE-IOPP we will employ is the HE transformation (see Definition 13) of the (Batched) FRI protocol. The batched FRI protocol allows a prover to prove the δ-correlated agreement of $f_1, \ldots, f_t$ by running the FRI protocol on $f = \sum_{i=1}^{t} \beta_i f_i$ for i.i.d. uniformly random[11] β_i. In fact,

[10] It could happen, for a malicious choice of the $\texttt{ct}_i \in R_q^2$, that $f(a) = \sum_{i=0}^{d} g_i a^i \ \forall a \in R_p$ for some $g_i \neq f_i$. In practice, this does not give any power to the adversary: it would be equivalent to putting a wrong polynomial of the right degree within the oracle, which should be caught by the IOP.

[11] We use i.i.d uniformly random coefficients, rather than powers of a single β, since otherwise we would incur an $O(n)$ soundness loss, see [BCI+20, BGK+23].

replacing FRI with another IOPP would still result in a δ-correlated agreement test and as we showed before (Theorems 3 and 4), we could use any other IOPP, to which we would previously apply our HE-transformation (Definition 13).

The $\texttt{HE-Batched-FRI}$ protocol

Setup: Agree on the following:

- A $\texttt{HE}$ scheme with ciphertext space R_q^2 and plaintext space $R_p \cong \prod_{j=1}^{\ell} \mathbb{F}_{p^D}, p \neq 2$, satisfying $2^n | (p^D - 1)$ for some positive integer n.
- A multiplicative group $L_0 = \{\omega, \ldots, \omega^{2^n}\} \subseteq \mathbb{F}_{p^D}^*$ of order 2^n, i.e. $L_0 = \langle \omega \rangle$.
- The rate of the Reed-Solomon code, $\rho = 2^{-R}$ for a positive integer R.
- A number of rounds $r < n - R$, each of which will use a domain $L_{i+1} = \{x^2 \mid x \in L_i\}$, i.e. $L_{i+1} = \langle \omega^{2^i} \rangle$.

Input: For $i = 1, \ldots, t$, alleged ciphertexts $(c_{i,1}, \ldots, c_{i,2^n/\ell}) \in R_q^2$ which allegedly satisfy $(c_{i,1}, \ldots, c_{i,2^n/\ell}) \in \mathsf{EncPoly}(f_i, L_0)$ for some $f_i(\mathtt{X}) \in \mathbb{F}_{p^D}[\mathtt{X}]_{<2^{n-R}}$. In other words, it should be that $f_i\big|_{L_0} \in \mathtt{RS}[\mathbb{F}_{p^D}, L_0, 2^{n-R}]$. $\mathcal{V}$ has oracles $\{[\![\mathsf{EncPoly}(f_i, L_0)]\!]\}_{i=1}^t$, whereas $\mathcal{P}$ has the underlying ciphertexts $\{\mathsf{EncPoly}(f_i, L_0)\}_{i=1}^t$.

Commit phase: $\mathcal{P}$ batches the encryptions of functions $f_1, \ldots, f_t$ into a single encryption of a function f. Afterwards, it sequentially constructs a series of oracles to "foldings" of encryptions of that function.

1. Obtain challenges $\beta_1, \ldots, \beta_t \xleftarrow{\$} \mathbb{F}_{p^D}$ from the verifier. Implicitly define the oracle $[\![f\big|_{L_0}]\!]_{\mathtt{HE}} = \sum_{i=1}^t \beta_i \cdot [\![f_i\big|_{L_0}]\!]_{\mathtt{HE}}$ from the oracles to the encrypted polynomials $\{\mathsf{EncPoly}(f_i, L_0)\}_{i=1}^t$.
2. For $0 \leq i < r$:
   - Obtain a challenge $\alpha_i \xleftarrow{\$} \mathbb{F}_{p^D}$ from the verifier.
   - $\mathcal{P}$ computes and sends the oracle $[\![\mathsf{EncPoly}(f^{(i+1)}, L_{i+1})]\!]$, where $L_{i+1} = \langle \omega^{2^{i+1}} \rangle$ and $f^{(i+1)} : L_{i+1} \to \mathbb{F}_{p^D}$ is allegedly such that (for $j \in [2^{n-i}]$):
$$f^{(i+1)}(\omega^{2^{i+1}j}) = \frac{f^{(i)}(\omega^{2^i j}) + f^{(i)}(-\omega^{2^i j})}{2} + \alpha_i \cdot \frac{f^{(i)}(\omega^{2^i j}) - f^{(i)}(-\omega^{2^i j})}{2 \cdot \omega^{2^i j}}.$$
3. The last oracle (to an encryption of) $f^{(r)} : L_r \to \mathbb{F}_{p^D}$, $L_r = \langle \omega^{2^{r-1}} \rangle$ is allegedly an encryption of a polynomial of degree strictly less than $\rho \cdot |L_r|$, so $\mathcal{P}$ can just directly send it as $\mathsf{EncPoly}(f^{(r)}, L_r)$ to $\mathcal{V}$.

Query phase: $\mathcal{V}$, using the $\mathtt{HE}$ secret key $\mathtt{sk}$ in order to decrypt the encrypted polynomials within the oracles, checks the consistency of the messages sent by the prover. If any of the following checks fails, reject and abort. Otherwise, accept.

1. For $0 \leq \ell < m$, $\mathcal{V}$ does the following in parallel:
   - Sample a random $\mu_\ell \xleftarrow{\$} L_0$.
   - For $0 \leq i < r$, using the oracles $\{[\![\mathsf{EncPoly}(f^{(j)}, L_j)]\!]\}_{j=0}^r$ and challenges $\{\alpha_j\}_{j=0}^{r-1}$ from the commit phase, check whether:
$$f^{(i+1)}(\mu_\ell^{2^{i+1}}) \stackrel{?}{=} \frac{f^{(i)}(\mu_\ell^{2^i}) + f^{(i)}(-\mu_\ell^{2^i})}{2} + \alpha_i \cdot \frac{f^{(i)}(\mu_\ell^{2^i}) - f^{(i)}(-\mu_\ell^{2^i})}{2 \cdot \mu_\ell^{2^i}}.$$
     Notice that the verifier needs to decrypt the values $f^{(i+1)}(\mu_\ell^{2^{i+1}})$, $f^{(i)}(\mu_\ell^{2^i})$ and $f^{(i)}(-\mu_\ell^{2^i})$ inside the oracles in order to do this.
2. Decrypt $\mathsf{EncPoly}(f^{(r)}, L_r)$ and check whether $f^{(r)}\big|_{L_r} \stackrel{?}{\in} \mathtt{RS}[\mathbb{F}_{p^D}, L_r, d]$.

Fig. 3: The $\texttt{HE-Batched-FRI}$ protocol.

Whereas there are new, concretely more efficient IOPs for circuit satisfiability every year [BCR+19, COS20], a series of variants of the FRI protocol have remained as the most practical choice for an IOPP until this day. Since this is the most stable component of our overall compilers, we provide our $\texttt{HE-Batched-FRI}$ protocol in Figure 3. We also adapt the results of [BGK+23] concerning round-by-round soundness of FRI to `HE-Batched-FRI`. Notice that we only need to consider RBR soundness, rather than RBR knowledge soundness, since the former is enough for their δ-correlated hIOP-to-SNARK compiler.

Theorem 5. *Let $\mathbb{F}$ be a finite field, $L_0 \subseteq \mathbb{F}^*$ a smooth multiplicative subgroup of size 2^n, $d_0 = 2^k, \rho = d_0/|L_0| = 2^{k-n}$ and ℓ a positive integer. For any integer $m \geq 3$, $\eta \in (0, \sqrt{\rho}/(2m))$, relative distance $\delta \in (0, 1 - \sqrt{\rho} - \eta)$ and functions $f_1^{(0)}, \dots, f_t^{(0)} : L_0 \to \mathbb{F}$ for $t \geq 2$ such that at least one of them is δ-far from $\mathtt{RS}^{(0)}$, the HE-Batched-FRI protocol (Figure 3) is complete and has round-by-round soundness error*

$$\epsilon = \max\left\{\frac{(m+1/2)^7 \cdot |L_0|^2}{3\rho^{3/2}|\mathbb{F}|}, (1-\delta)^\ell\right\}.$$

Furthermore, under Conjecture 1 (see Appendix A), the error can be further reduced to

$$\epsilon = \max\left\{\frac{|L_0|^{c_2}}{(\rho\eta)^{c_1}|\mathbb{F}|}, (1-\delta)^\ell\right\}.$$

Proof. Completeness follows from Theorem 2 and Definition 15. RBR soundness follows from Theorem 2 and [BGK+23, Theorem 4.2]. ■

6 Optimisations

To make our construction practical, we introduce several optimizations that may have independent value and applications beyond this work.

6.1 On the choice of HE Scheme

Our construction requires an HE scheme that supports finite fields as the plaintext space, making it compatible with nearly all modern HE schemes, such as TFHE [CGGI20], BGV [BGV12], and BFV [Bra12, FV12]. The notable exception is CKKS [CKKS17], which is inherently approximate (even for a fresh encryption, $\mathtt{HE.Dec}(\mathtt{HE.Enc}(\mathtt{pk}, m), \mathtt{sk}) \approx m$ rather than $\mathtt{HE.Dec}(\mathtt{HE.Enc}(\mathtt{pk}, m), \mathtt{sk}) = m$). This is incompatible with our verification approach, which requires exact arithmetic. Thus, our choices are BGV/BFV and TFHE, guided by practical performance and implementation availability, detailed in Section 7.

6.2 Tensoring

Recall the HE plaintext space structure from Section 2. Ideally, the plaintext ring R_p should split into ℓ copies of $\mathbb{F}_{p^D}$, where D meets FRI security requirements, i.e., $|\mathbb{F}_{p^D}| \approx 2^{256}$. This requires $\mathbb{F}_p$ to contain roots of unity of order at most $2N/D$, where N is the cyclotomic ring degree. Given the HE scheme's requirement that $\log(N) \in 11, \dots, 17$, p would have to be smaller than $2N/D$. Additionally, FRI requires $\mathbb{F}_{p^D}$ to have a 2^n-th root of unity, thus $pD > 2^n$. In practice, using roots of unity in $\mathbb{F}_p$ (requiring $p > 2^n$) allows the NTT (Section 6.3) to run D times faster.

Combining all requirements (FRI, HE, implementation), $2^n = d\rho^{-1} < p < 2N/D$, restricting input polynomial size to $d < 2N\rho/D$. This is feasible with parameter sets like $(\log(N), \rho, D) = (17, 1/2, 6)$, allowing d up to 2^{14}. However, better-performing parameters, such as $(\log(N), \rho, D) = (14, 1/16, 12)$, restrict d to around 2^6, almost entirely restricting the use of FRI use.

To address this, we use a field extension $\mathbb{F}_{p^D}$ of $\mathbb{F}_p$ in our HE-FRI protocol. When evaluating an encrypted polynomial on an element of $\mathbb{F}_{p^D}$, we emulate the arithmetic of $\mathbb{F}_{p^D}$ through a circuit. This results in D HE ciphertexts, which together encrypt a single value in $\mathbb{F}_{p^D}$. Alternatively, an intermediate approach uses a value d', with the plaintext space as $\mathbb{F}_{p^{d'}}$, and emulates $\mathbb{F}_{p^D}$ arithmetic with D/d' ciphertexts. The ideal value of d' depends on the application, since increasing the plaintext modulus increases all HE parameters.

6.3 Shallow Reed-Solomon encoding

Reed-Solomon encoding consists of interpreting data as a polynomial of degree $(d-1)$ and evaluating it at $2^n = d\rho^{-1}$ independent points. Polynomial evaluation is linear and simply evaluating it 2^n times would result in quadratic performance. The Fast Fourier Transform is a staple solution for this problem, enabling the evaluation in $O(2^n \log 2^n)$ operations. Commonly, however, it is implemented as a circuit with depth $\log 2^n$, which poses some challenges for its homomorphic evaluation. Fortunately, FFTs, and, more specifically, Number-Theoretic Transforms (NTTs), their generalization to finite fields, are ubiquitous in the FHE literature, and solutions for evaluating them with small depth are very well established [CG99, GPvL23]. In this work, we adopt a radix-k NTT of parametrizable depth for some $k \in [\![2, \sqrt{2^n}]\!]$ optimized based on practical performance.

The NTT can be especially costly when running batched FRI, as the RS codeword needs to be calculated for each polynomial individually. This cost can be minimized by exploiting the HE scheme packing to perform the NTT over several polynomials at once. We consider the following strategies for packing.

- **Single polynomial packing:** Let k be a single polynomial $k = \sum_{i=0}^{d-1} k_i \mathtt{X}^i \in \mathbb{F}_p[\mathtt{X}]$. We encrypt k in an array of d/N ciphertexts $\mathtt{ct}$, such that $\mathtt{ct}_i$ encrypts $\sum_{j=0}^{d-1} k_{i \cdot N + j} \mathtt{X}^j$. The main advantage of this approach is the significantly reduced memory usage, as we process one polynomial at a time. Conversely, evaluating the NTT algorithm with this packing requires performing permutations within each ciphertext [CG99], which is an expensive process compared to the other operations needed for evaluating the NTT.
- **Batched polynomial packing:** Let k be an N-sized list of polynomials with maximum degree $(d-1)$, and $k_{i,j}$ be the coefficient of degree j of the i-th polynomial. We encrypt k in an array of d ciphertexts $\mathtt{ct}_i$, such that the j-th slot of $\mathtt{ct}_i$ encrypts $k_{j,i}$. The main advantage of this approach is avoiding the aforementioned permutations, as each coefficient of a polynomial would be in different ciphertexts. For large polynomials, the memory requirements to run the NTT with this packing might be impractical, however.

In both cases, even though FRI is defined over $\mathbb{F}_{p^D}$, we perform the entire NTT in $\mathbb{F}_p$ by selecting roots of unity in $\mathbb{F}_p$, as roots of unity in $\mathbb{F}_{p^D}$ would bring negligible advantage compared to the size of p (as discussed in Section 6.2). Further, the goal of the NTT is to create a redundant representation of the polynomial (*i.e.*, the RS codeword), which consumes more memory to be stored. In this way, storing the codewords could represent a problem, even if storing the original polynomials was not. This is the main aspect we consider when choosing which type of packing we adopt. Mixed packing approaches could likely be a better solution for this problem, but developing them is not within our scope.

6.4 Shallow folding

In the commit phase of HE-FRI (Figure 3), $\mathcal{P}$ needs to compute a series of oracles $[\![\mathsf{EncPoly}(f^{(i)}, L_i)]\!]$. Let us denote $f^{(i+1)} = \texttt{Fold}(f^{(i)}, \alpha_i)$. The complexity of producing all such foldings as described there is $O(2^n)$, while the depth[12] is n. To reduce depth, we replace the FFT-like algorithm with a DFT-like algorithm. We compute the first layer of the Fold operation as usual, then pre-compute constants for the following layers, expressing each as a composition $\texttt{Fold} \circ \ldots \circ \texttt{Fold}$. Each layer can now be expressed as inner products of the original polynomial, reducing the depth to 1, while increasing the complexity to $O(2^n \log(2^n))$. This does not affect the overall FRI complexity (dominated by the NTT). The Verifier's side remains unchanged. The full algorithm is presented in Appendix D.

6.5 Fast decryption

In RLWE-based cryptography, decrypting small amounts of data may incur a significant overhead depending on the adopted parameters. An RLWE sample of dimension N encrypts up to N messages in $\mathbb{F}_p$, which can all be decrypted at once with cost $O(N \log N)$. However, if one wants to decrypt just a single message in $\mathbb{F}_p$, the cost would be at least $O(N)$, which represents a performance overhead of $N/\log(N)$ times compared to the amortized cost of decrypting all messages at once. During the commit phase of HE-FRI, the prover performs computation using RLWE samples of dimension N encrypting N messages in $\mathbb{F}_p$. During the query phase, however, the verifier only needs to learn two evaluation points in $\mathbb{F}_{p^D}$ per round for each linearity check. In this way, if the prover provides these points packed in a ciphertext of dimension N, it would impose a performance overhead of at least $N/(2D)$ times for the verifier compared to an optimal RLWE decryption (*i.e.*, it would be decrypting at least $N/(2D)$ more messages than necessary). To improve on this, we propose three possible approaches. We summarise them here and present the full details in Appendix D.

Repacking The simplest way of minimizing decryption costs is to reduce the ciphertext dimension. This can be achieved in several ways. In this work, since the prover also needs to arithmetically manipulate points individually, we choose to extract the points to LWE samples and repack them using a key-switching algorithm. We pack the two points needed per linearity check in the same RLWE samples, further minimizing decryption costs.

Decomposition and recomposing While the repacking already enables us to significantly reduce the ciphertext dimension, we are still limited by the value of p. Specifically, the ciphertext dimension depends on the ciphertext modulus for security, which, in turn, depends on the plaintext modulus p. To enable further reductions in the size of N, we introduce a decomposition and recomposing procedure based on techniques introduced in [CLOT21] and described in [CGGI20]. Our proposal starts from the observation that since the verifier does not compute any homomorphic operations on these samples (it only decrypts them), *we do not need to preserve any homomorphic properties.* In fact, once the commit phase is finished, the evaluation points can be treated simply as strings of bits, and the goal becomes to encrypt them in the smallest possible RLWE ciphertext. Considering this, we homomorphically decompose elements in $\mathbb{F}_p$ in digits of size $\log \bar{p}$ for some $\bar{p} < p$. This process enables us to repack the evaluation points in RLWE ciphertexts with smaller moduli and, hence, smaller dimensions. We present our full recomposition process in Algorithm 3.

[12] n is the logarithm of the codeword size. Folding is a linear algorithm.

Adding samples Adding samples appears as an alternative to the decomposition. The repacking procedure mentioned above allows us to reduce the dimension of the RLWE samples that are sent to the verifier, but we are still not using all the coefficients in the sample. Concretely, our implementation uses ciphertexts with $N = 512$ to only encrypt $2D = 32$ messages. Considering this, the verifier can further minimize decryption costs by adding rotations of ciphertexts together and decrypting messages from multiple ciphertexts at once. We note that rotation can be done considering *coefficient representation*, which is inexpensive. While this efficiently minimises the number of decryptions, it does not, contrary to the decomposition, reduce the cost of hashing operations or proof size.

7 Experimental Results

Operating over the plaintext space allows compatibility with nearly all HE schemes (except CKKS), enabling various implementation methods. For this proof of concept, we focus on a simple and efficient implementation that demonstrates practical execution times for both the prover and verifier. While we make specific choices of schemes and techniques, our construction remains compatible with most existing schemes. Appendix E provides a performance characterization based on the number of basic operations executed by each party, with and without the optimizations in Section 6. This broader view of HE-FRI performance helps extrapolate the results to other parameters or schemes.

7.1 Practical Parameters

For the security of FRI, we consider parameters established by previous literature [BGK+23], reproduced in Table 1. There are also practical parameters that are required for functionality, as we discussed in Section 6.2. Considering this, Table 2 presents the main choices of parameters according to the maximum size of the input polynomial that they support. We note that, for every parameter, the maximum size of the input polynomial can be increased by up to D times at the cost of D times more expensive NTT (as also discussed in Section 6.2).

Table 1: Security parameters we adopt for FRI, based on estimates of [BGK+23] using Conjecture 1. We approximate the size of the field for practical reasons.

Parameter	$\log_2(\lvert\mathbb{F}_{p^D}\rvert)$	ρ	m	δ
FRI_0	251-269	1/2	102	0.5
FRI_1		1/4	51	0.75
FRI_2		1/8	34	0.875
FRI_3		1/16	26	0.937

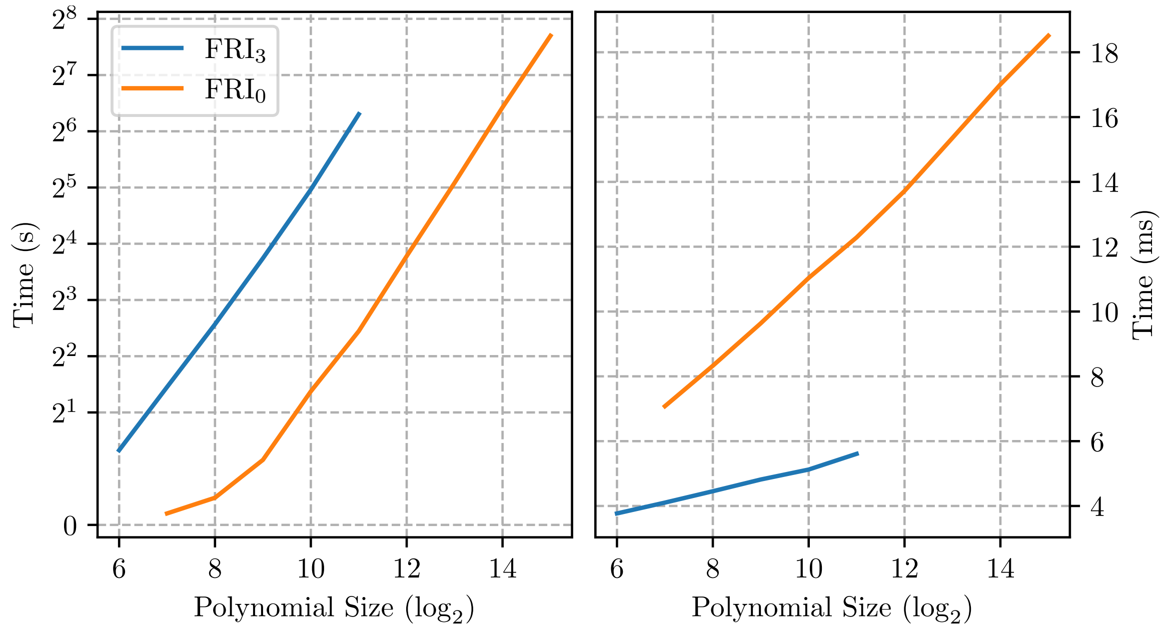
7.2 Proof-of-concept implementation

We build our proof of concept over the implementation of [S+21], a Python-implemented version of FRI for prime fields. We extend it to work over the extension field $\mathbb{F}_{p^D}$ and connect it to optimized FHE libraries to implement the encrypted arithmetic. We consider polynomials of degree up to 2^{15}

Table 2: Practical parameters for FRI based on the maximum size of the input polynomial d.

Maximum input size $\log_2(d)$	D	p	$\log_2(p)$	$\log_2(\lvert\mathbb{F}_{p^D}\rvert)$
15	16	65537	16.0	256.0
20	11	23068673	24.5	269.1
25	9	469762049	28.8	259.3
30	7	75161927681	36.1	252.9
35	7	206158430209	37.6	263.1
40	6	6597069766657	42.6	255.5
45	5	1337006139375617	50.2	251.2

and present results for parameter sets FRI_0 and FRI_3, which are optimized for the verifier and the prover, respectively. We run all the experiments in a `c6i.metal` instance (Intel Xeon 8375C at 3.5 GHz with 256 GiB of RAM) on AWS. The prover is parallelized to use up to 32 threads and the verifier is single-threaded.



(a) Prover (32 threads)

(b) Verifier (single-thread)

Fig. 4: Performance of HE-FRI using parameter sets FRI_3 and FRI_0 for $D = 16$ for 4096 polynomials (batched). Detailed results are provided in Appendix F.

Prover performance Figure 4a shows the results for the prover. It includes the execution time of the RS encoding, batching, and folding procedures. For the RS encoding, we implemented a generic RNS-based homomorphic NTT implementation built over Intel HEXL [BKS+21]. Its performance should be similar to most commonly used HE libraries, as HEXL is used for RNS implementations on libraries such as HELib [HS20] and OpenFHE [ABBB+22]. We use depth-2 NTTs with parallelized recursive calls (up to a maximum of 32 threads) and perform batched polynomial packing (Section 6.3) to compute it on $N = 4096$ polynomials at once. Our construction is flexible to the HE encoding, as we may later move to the required encodings during batching, as Appendix C details.

Our folding procedure, on the other hand, has fixed depth (Section 6.4) and is the last procedure to run before decryption. As such, it requires a much smaller ciphertext modulus and can be evaluated using single-precision (non-RNS) implementations. In this way, whenever possible, we evaluate it using implementation techniques from TFHE [CGGI20] using the MOSFHET library [GBA24]. Despite being asymptotically quasi-linear, we note that the execution time increases almost linearly with the polynomial size, which is a result of better parallel efficiency and the overhead introduced by repacking (which is linear). With FRI_0, it takes less than 0.5 seconds to run for polynomials of degree bound up to 2^8, going up to 207 seconds for polynomials of degree bound 2^{15}.

Verifier performance Figure 4b shows the verifier results. Verification in FRI is sublinear and typically runs in milliseconds. Hence, Python is not a good fit to showcase practical timings, and we implemented an optimized version of it in C. We only present a single-threaded version, but we note that it is trivially parallelizable and could be significantly accelerated for larger parameters. We show results for FRI_0 and FRI_3, noting that other parameters can also affect verifier performance, particularly decryption costs. To minimize these, we only considered repacking without decompositions, but Appendix D discusses further improvements. Our current setup allows for batched verification of 4096 codewords in as low as 6ms, with performance scaling linearly with the batch size. Future work includes optimizing for smaller batches and other parameters.

Acknowledgements

We would like to thank Zvika Brakerski for comments about our repacking optimization for the HE-Batched-FRI protocol. We also want to thank Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari and Michał Zajac for a useful conversation about their work [BGK+23] and confirming that their analysis does not require finite fields to be prime.

This work was partly done while A. Guimarães was a Ph.D. student at University of Campinas, Brazil. He was supported by the São Paulo Research Foundation under grants 2013/08293-7, 2019/12783-6, and 2021/09849-5. This work is partially funded by the European Union (GA 101096435). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

References

ABBB+22. Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 53–63, New York, NY, USA, 2022. Association for Computing Machinery.

ACY23. Gal Arnon, Alessandro Chiesa, and Eylon Yogev. IOPs with inverse polynomial soundness error. *Cryptology ePrint Archive*, 2023.

AHH$^+$24. Nuttapong Attrapadung, Goichiro Hanaoaka, Ryo Hiromasa, Yoshihiro Koseki, Takahiro Matsuda, Yutaro Nishida, Yusuke Sakai, Jacob C. N. Schuldt, and Satoshi Yasuda. Privacy-preserving verifiable CNNs. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 373–402, Cham, 2024. Springer Nature Switzerland.

APS15. Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

BCCW19. Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 45–56, 2019.

BCFK21. Alexandre Bois, Ignacio Cascudo, Dario Fiore, and Dongwoo Kim. Flexible and efficient verifiable computation on encrypted data. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pages 528–558. Springer, Heidelberg, May 2021.

BCI$^+$20. Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *61st FOCS*, pages 900–909. IEEE Computer Society Press, November 2020.

BCR$^+$19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60. Springer, Heidelberg, October / November 2016.

BGBE19. Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821. PMLR, 2019.

BGK$^+$23. Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michał Zając. Fiat-Shamir Security of FRI and Related SNARKs. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 3–40, Singapore, 2023. Springer Nature Singapore.

BGKS20. Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In Thomas Vidick, editor, *ITCS 2020*, volume 151, pages 5:1–5:32. LIPIcs, January 2020.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

BKS18. Eli Ben-Sasson, Swastik Kopparty, and Shubhangi Saraf. Worst-case to average case reductions for the distance to a code. In *33rd Computational Complexity Conference (CCC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

BKS$^+$21. Fabian Boemer, Sejun Kim, Gelila Seifu, Fillipe D.M. de Souza, and Vinodh Gopal. Intel HEXL: Accelerating Homomorphic Encryption with Intel AVX512-IFMA52. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '21, page 57–62, New York, NY, USA, 2021. Association for Computing Machinery.

BMMP18. Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 483–512. Springer, Heidelberg, August 2018.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Heidelberg, August 2012.

BV14. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on computing*, 43(2):831–871, 2014.

CCH$^+$19. Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In Moses Charikar and Edith Cohen, editors, *51st ACM STOC*, pages 1082–1090. ACM Press, June 2019.

CCL15. Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

CG99. Eleanor Chu and Alan George. *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms.* CRC Press, November 1999. Google-Books-ID: 30S3kRiX4xgC.

CGGI20. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

CKKS17. Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Heidelberg, December 2017.

CLOT21. Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Heidelberg, December 2021.

CMS19. Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part II*, volume 11892 of *LNCS*, pages 1–29. Springer, Heidelberg, December 2019.

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Heidelberg, May 2020.

CP19. Benjamin R Curtis and Rachel Player. On the feasibility and impact of standardising sparse-secret LWE parameter sets for homomorphic encryption. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 1–10, 2019.

DDGR20. Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2020.

DPSZ12. Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, Heidelberg, August 2012.

DS16. Léo Ducas and Damien Stehlé. Sanitization of FHE ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 294–310. Springer, Heidelberg, May 2016.

FGP14. Dario Fiore, Rosario Gennaro, and Valerio Pastro. Efficiently verifiable computation on encrypted data. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 844–855. ACM Press, November 2014.

FNP20. Dario Fiore, Anca Nitulescu, and David Pointcheval. Boosting verifiable computation on encrypted data. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 124–154. Springer, Heidelberg, May 2020.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. https://eprint.iacr.org/2012/144.

GBA24. Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized Software for FHE over the Torus. *Journal of Cryptographic Engineering*, July 2024.

Gen09a. Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

Gen09b. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

GGP10. Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 465–482. Springer, Heidelberg, August 2010.

GGW23. Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? Cryptology ePrint Archive, Paper 2023/1609, 2023. https://eprint.iacr.org/2023/1609.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

GNS23. Chaya Ganesh, Anca Nitulescu, and Eduardo Soria-Vazquez. Rinocchio: SNARKs for ring arithmetic. *Journal of Cryptology*, 36(4):41, October 2023.

GPvL23. Antonio Guimarães, Hilder V. L. Pereira, and Barry van Leeuwen. Amortized bootstrapping revisited: Simpler, asymptotically-faster, implemented. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 3–35, Singapore, 2023. Springer Nature Singapore.

GVW15. Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.

Hab22. Ulrich Haböck. A summary on the fri low degree test. Cryptology ePrint Archive, Paper 2022/1216, 2022. https://eprint.iacr.org/2022/1216.

HS15. Shai Halevi and Victor Shoup. Bootstrapping for HElib. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 641–670. Springer, Heidelberg, April 2015.

HS20. Shai Halevi and Victor Shoup. Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481, 2020. https://eprint.iacr.org/2020/1481.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

S+21. Szepieniec et al. Anatomy of a stark - tutorial for starks with supporting code in python, November 2021. https://github.com/aszepieniec/stark-anatomy.

SV14. Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *DCC*, 71(1):57–81, 2014.

A A conjecture on FRI

Our implementation makes use of the following Conjecture (Conjecture 5.12 from [BGK+23]). Before presenting it, we introduce the following notation. For an index $i \in \{0, \ldots, k\}$, we let $\mathtt{RS}^{(i)} := \mathtt{RS}[\mathbb{F}, L_i, d_i]$, where $L_0 \subset \mathbb{F}^*$ is a smooth multiplicative subgroup of size 2^n, $d_0 = 2^{n-R}$, $L_i := \{z^2 : z \in L_{i-1}\}$, and $d_i = d_{i-1}/2$. Note that this implies that for every $i \in \{0, \ldots, n-R\}$, L_i is a smooth multiplicative subgroup of size 2^{n-i}. For a function $f : L_i \to \mathbb{F}$ and $x \in \mathbb{F}$, we define an "algebraic hash function" [BKS18] as follows:

$$
\begin{gathered}
H_x[f] : L_{i+1} \to \mathbb{F} \\
H_x[s] := \frac{x - s'}{s'' - s'} \cdot f(s'') + \frac{x - s''}{s' - s''} \cdot f(s') \\
s, s' \in L_i,\; s' \neq s'', (s')^2 = (s'')^2 = s \in L_{i+1}.
\end{gathered}
$$

This hash function H_x has the property that if $f \in \mathtt{RS}^{(i)}$, then $H_x[f] \in \mathtt{RS}^{(i+1)}$ for any x. Additionally, for arbitrary $G_i : L_i \to \mathbb{F}$ and $G_{i+1} = H_x[G_i]$, then G_i and G_{i+1} will pass all verifier checks during the Query Phase of the FRI protocol, so long as $d_{i+1} > 1$.

Conjecture 1. Let $\mathbb{F}$ be a finite field, $L_0 \subset \mathbb{F}^*$ a smooth multiplicative subgroup of size 2^n, $d_0 = 2^{n-R}$ and $\rho = 2^{-R}$. There exist constants c_1 and c_2 such that for all $\nu > 0$ and any $\delta \leq 1 - \rho\nu$, for any function $G_i : L_i \to \mathbb{F}$ that is δ-far from $\mathtt{RS}^{(i)}$ we have that

$$
\Pr_{x \xleftarrow{\$} \mathbb{F}} \left[\Delta(H_x[G_i], \mathtt{RS}^{(i)}) \leq \delta \right] \leq \frac{|L_0|^{c_2}}{(\nu\rho)^{c_1} \cdot |\mathbb{F}|}.
$$

Moreover, for any $f_1, \ldots, f_t : L_0 \to \mathbb{F}$ such that at least one f_i is δ-far from $\mathtt{RS}^{(0)}$, we have

$$
\Pr_{\alpha_1, \ldots, \alpha_t \xleftarrow{\$} \mathbb{F}} \left[\Delta(G_0, \mathtt{RS}^{(0)}) \leq \delta | G_0 = \sum_i \alpha_i f_i \right] \leq \frac{|L_0|^{c_2}}{(\nu\rho)^{c_1} \cdot |\mathbb{F}|},
$$

$$
\Pr_{\alpha \xleftarrow{\$} \mathbb{F}} \left[\Delta(G_0, \mathtt{RS}^{(0)}) \leq \delta | G_0 = \sum_i \alpha^{i-1} f_i \right] \leq \frac{t \cdot |L_0|^{c_2}}{(\nu\rho)^{c_1} \cdot |\mathbb{F}|}.
$$

In their work, Ben-Sasson et al. [BCI+20] state the following: "To the best of our knowledge, nothing contradicts setting $c_1 = c_2 = 2$", and when the characteristic of $\mathbb{F}$ is greayer than d_0 state that they "are not aware of anything contradicting $c_1 = c_2 = 1$". Finally, they do note that if $\mathbb{F}$ has characteristic 2, then $c_1 = c_2 = 1$ is impossible, due to an attack of [BGKS20].

B HE definitions

B.1 Semantic Security

Definition 16 *(Semantic Security) Let* $\mathtt{HE} = (\mathtt{HE.KeyGen}, \mathtt{HE.Enc}, \mathtt{HE.Dec}, \mathtt{HE.Eval})$ *be a (public-key) homomorphic encryption scheme as defined above, let* $(\mathtt{pp}, \mathcal{C}) \leftarrow \mathtt{HE.Setup}(1^\lambda, \mathcal{M}, \widehat{\mathcal{C}\mathrm{irc}})$*, and let* $\mathcal{A}$ *be an adversary. The advantage of* $\mathcal{A}$ *with respect to* $\mathtt{HE}$ *is defined as follows, with* $(\mathtt{sk}, \mathtt{pk}, \mathtt{evk}) \leftarrow \mathtt{HE.KeyGen}(1^\lambda, \mathcal{C}, \mathtt{pp})$.

$$\begin{aligned}\mathbf{Adv}^{\mathtt{HE}}_{\mathcal{A}}(\lambda) := \big| &\Pr[\mathcal{A}(\mathtt{pk}, \mathtt{ct}) = 1 : \mathtt{ct} \leftarrow \mathtt{HE.Enc}(\mathtt{pk}, 1)] \\ &- \Pr[\mathcal{A}(\mathtt{pk}, \mathtt{ct}) = 1 : \mathtt{ct} \leftarrow \mathtt{HE.Enc}(\mathtt{pk}, 0)]\big|.\end{aligned}$$

$\mathtt{HE}$ *is semantically secure if* $\mathbf{Adv}^{\mathtt{HE}}_{\mathcal{A}} = \mathsf{negl}(\lambda)$ *for every PPT adversary* $\mathcal{A}$.

B.2 The BGV Scheme

Following [HS20], we define the BGV scheme [BGV12] as a levelled FHE scheme based on the RLWE problem [LPR10]. The ciphertext space is $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, where q is the ciphertext modulus. The plaintext space is $\mathcal{R}_p = \mathbb{Z}_p[x]/(x^n + 1)$, where p is the plaintext modulus. Messages and ciphertexts will be considered as polynomials in $\mathcal{R}_p$ and $\mathcal{R}_q$, respectively.

The BGV scheme is parametrised by the following:

– The length L of the moduli chain $Q_L \gg \ldots \gg Q_0$, where $Q_i | Q_{i+1}$ for $i \in \{0, \ldots, L-1\}$
– The decomposition base ω, where $h = \lfloor \log(q) \rfloor + 1$ is the number of elements in a decomposition of an element modulo q in base ω.
– The secret key distribution $\mathcal{S}$, typically ternary with a specified Hamming weight
– The error distribution χ, typically a discrete Gaussian of small standard deviation

BGV consists of the algorithms `HE.KeyGen`, `HE.Enc`, `HE.Dec`, `HE.Add`, `HE.PreMult`, `HE.Relinearize`, `HE.KeySwitch` and `HE.ModSwitch`, defined as follows. Below, we assume that a fresh encryption will be "at the top" modulus L, but note that `HE.Enc` can be defined at any level i.

`HE.KeyGen`(1^λ): Draw $s \leftarrow \mathcal{S}$ and set $(1, s) := \mathtt{sk}$ as the secret key. Sample $a \leftarrow \mathcal{R}_{Q_L}$ and $e \leftarrow \chi$. Set $\mathtt{pk} = (\mathtt{pk}[0], \mathtt{pk}[1]) := ([-as - pe]_{Q_L}, a)$ as the public key. The evaluation key is defined as follows. For $i \in \{0, \ldots, h\}$, sample $a_i \leftarrow R_q$ uniformly at random, and $e_i \leftarrow \chi$. Output $\mathtt{evk} = \{(-(a_i s + pe_i) + w_i s^2, a_i)\}_{i \in \{0,\ldots,h\}}$ in R_q. Return $(\mathtt{sk}, \mathtt{pk}, \mathtt{evk})$.

`HE.Enc`$(\mathtt{pk}, m)$: Let $m \in \mathcal{R}_p$ be a message. Let $Q_i, i \in \{0, \ldots, L\}$ be the modulus in the moduli chain of the current level. Sample $u \leftarrow \mathcal{S}$ and $e_1, e_2 \leftarrow \chi$. Return $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) := ([m + \mathtt{pk}[0]u + pe_1]_{Q_i}, [\mathtt{pk}[1]u + pe_2]_{Q_i})$.

`HE.Dec`$(\mathtt{sk}, \mathtt{ct})$: Let $\mathtt{ct} = (c_0, c_1)$ and $\mathtt{sk} = (1, s)$. Return $m' = [[c_0 - sc_1]_{Q_i}]_p$.

`HE.Add`$(\mathtt{ct}_0, \mathtt{ct}_1)$: Bring $\mathtt{ct}_0, \mathtt{ct}_1$ to the same level by modulus switching the ciphertext at the higher level to the lower level. Return $\mathtt{ct} := ([\mathtt{ct}_0[0] + \mathtt{ct}_1[0]]_{Q_i}, [\mathtt{ct}_0[1] + \mathtt{ct}_1[1]]_{Q_i})$.

$\texttt{HE.Mult}(\texttt{ct}_0, \texttt{ct}_1, \texttt{evk})$**:** Bring $\texttt{ct}_0, \texttt{ct}_1$ to the same level by modulus switching the ciphertext at the higher level to the lower level.
Calculate $\texttt{ct}_{\text{mult}} = (c_0, c_1, c_2) :=$
$([\texttt{ct}_0[0]\texttt{ct}_1[0]]_{Q_i}, [\texttt{ct}_0[0]\texttt{ct}_1[1] + \texttt{ct}_0[1]\texttt{ct}_1[0]]_{Q_i}, [\texttt{ct}_0[1]\texttt{ct}_1[1]]_{Q_i})$.

$\texttt{HE.ModSwitch}((\texttt{ct}, Q_i), Q_j)$**:** Let $\texttt{ct} = (\texttt{ct}[0], \texttt{ct}[1])$ be at level i. Return $\texttt{ct}^{ms} := \left(\left\lfloor \frac{Q_j}{Q_i}\texttt{ct}[0] \right\rceil_p, \left\lfloor \frac{Q_j}{Q_i}\texttt{ct}[1] \right\rceil_p \right)$.

$\texttt{HE.KeySwitch}(\texttt{ct}, \texttt{sk}', \texttt{evk})$**:** Let $\texttt{ct}[0] = c_0$, $\texttt{ct}[1] = c_1$ and $\texttt{ct}[2] = c_2$. Let $\texttt{evk}[i][0] = -(a_i s + pe_i) + w_i s^2$ and $\texttt{evk}[i][1] = a_i$, and express c_2 in base ω as

$$c_2 = \sum_{i=0}^{h} c_2^{(i)} \omega^i.$$

Set $c_0' = c_0 + \sum_{i=0}^{h} c_2^{(i)}\texttt{evk}[i][0]$ and $c_1' = c_1 + \sum_{i=0}^{h} c_2^{(i)}\texttt{evk}[i][1]$. Output $\texttt{ct}' = (c_0', c_1')$.

C Batching from slot encoding

Applications using FHE schemes such as BGV and BFV often employ the CRT encoding [SV14] to work with *"plaintext slots"*, which can be multiplied element-wise. HE-FRI doesn't require such slots as it doesn't perform multiplications between ciphertexts. One could choose to employ them nonetheless, but the LWE extraction procedure needed for the repacking becomes significantly more expensive when using slot encoding.

The simplest way of approaching this issue would be converting from slots to the standard *coefficient encoding* before evaluating FRI. Efficient methods for it are well defined in the bootstrapping literature [HS15] and are composed of linear combinations that are reasonably inexpensive to evaluate. Notwithstanding, we introduce a method for evaluating (batched) HE-FRI directly over the slot representation that produces evaluation points in coefficient representation at no additional cost.

This method comes from the observation that the first step of the folding is to perform a linear combination of several codewords, which are packed according to the batched polynomial packing described in Section 6.3. This process consists essentially of a linear combination of values in the same ciphertext, which can be obtained directly from the slot representation. More specifically, let $m \in \mathcal{R}_p$ be the polynomial encoding the plaintext $\boldsymbol{t} = [t_0, t_1, \ldots, t_{N-1}] \in \mathbb{F}_p \times \ldots \times \mathbb{F}_p$ in slot representation, which contains the k-th evaluation points of every input polynomial. The process for converting from slot to coefficients can be implemented as follows, where $\boldsymbol{z}_i$ are vectors of constants and $\hat{\mathbf{m}}$ is the vector of coefficients of m. We can recover $\boldsymbol{t}$ as a coefficient-wise encoded polynomial as follows.

$$\boldsymbol{t} = [\langle \hat{\mathbf{m}}, \boldsymbol{z}_0 \rangle, \langle \hat{\mathbf{m}}, \boldsymbol{z}_1 \rangle, \ldots, \langle \hat{\mathbf{m}}, \boldsymbol{z}_{N-1} \rangle]. \tag{2}$$

In practice, each $\boldsymbol{z}_i$ would be a different permutation of a vector of powers of some root of unity in R_p and the vector $\boldsymbol{t}$ would be encoded as a polynomial (coefficient encoding). For our purposes, however, the specific details of this process are unimportant. At the beginning of the folding, we would batch the codewords as follows, to obtain the k-th point of the initial codeword $f^{(0)}$.

$$f^{(0)}(k) = \sum_{i=0}^{N} \langle \hat{\mathbf{m}}, \boldsymbol{z}_i \rangle \cdot \beta_i. \tag{3}$$

Writing the inner product explicitly and reordering the summations, we have that:

$$f^{(0)}(k) = \sum_{i=0}^{N} \left(\sum_{j=0}^{N} \hat{\mathbf{m}}_j \cdot \boldsymbol{z}_{i,j} \right) \cdot \beta_i = \sum_{j=0}^{N} \hat{\mathbf{m}}_j \cdot \left(\sum_{i=0}^{N} \boldsymbol{z}_{i,j} \cdot \beta_i \right) \tag{4}$$

In this way, we can pre-compute the inner summation of constants, and obtain $f^{(0)}(k)$ from a single inner product with $\hat{\mathbf{m}}$, without having to calculate $\boldsymbol{t}$. As this is a single inner product, we can evaluate it with just a polynomial multiplication. Let $\boldsymbol{z}'_j = \left(\sum_{i=0}^{N} \boldsymbol{z}_{i,j} \cdot \beta_i \right)$, we have that:

$$f^{(0)}(k) = \textsc{ExtractLWE}(m \cdot \left(\boldsymbol{z}'_0 - \sum_{j=1}^{N-1} \boldsymbol{z}'_j x^{N-j} \right), 0). \tag{5}$$

D Further improvements for decryptions

D.1 Shallow `Fold`

Algorithm 1: Shallow `Fold`

```
Input  : $n' = 2^n$, $r$, $f^{(0)}, \omega$.
Output: codewords $f^{(i)}$, for $i \in [\![0, r-1]\!]$.
1  $c \leftarrow [0, \ldots, 0]$                                  // First pre-compute the constants
2  for $k \leftarrow 0$ to $r-1$ do
3      for $i \leftarrow 0$ to $n'/2 - 1$ do
4          for $j \leftarrow 0$ to $2^{k+1} - 1$ do
5              $c_{i+jn'/2} \leftarrow c_{i+jn'/2} \cdot \frac{1}{2} \cdot \left(1 + \frac{(-1)^j \alpha_j}{\omega^i}\right)$
6      for $i \leftarrow 0$ to $n'/2 - 1$ do
7          $f_i^{(k)} \leftarrow 0$                                  // Now compute the Fold
8          for $j \leftarrow 0$ to $2^{k+1} - 1$ do
9              $f^{(k)}(\omega^i) \leftarrow f^{(k)}(\omega^i) + c_{i+j\cdot(n'/2)} \cdot f(\omega^{i+j\cdot(n'/2)})^{(0)}$
10     $n' \leftarrow n'/2$
11     $\omega \leftarrow \omega^2$
12 return $f$
```

D.2 Decomposed Repacking

The verifier does not compute any homomorphic operations over ciphertexts in our construction, hence we do not need to preserve any homomorphisms for them. In fact, once the commit phase is finished, the evaluation points can be treated as just strings of bits, and our goal becomes to encrypt them in the smallest possible ciphertext. As described in Section 6.5, we define repacking and recompositions algorithms to improve performance by exploiting this fact. Algorithm 2 shows the repacking procedure, which requires the subprocedures listed in the following paragraph. We use techniques introduced in [CLOT21] for the decomposition and described in [CGGI20] for the other procedures. We note that, although most of these subprocedures are often used with the TFHE scheme [CGGI20] in previous literature, they are generic for RLWE-based FHE, and we do not rely on any specific property of TFHE.

Algorithm 2: Repacking

```
Input  : Two vectors of RLWE samples $\mathbf{x} = [x_0, x_1, \ldots, x_{d-1}]$ and $\mathbf{y} = [y_0, y_1, \ldots, y_{d-1}]$, encrypting $N$
         evaluation points in $\mathbb{F}_{p^D}$ each;
Input  : indices $i_x, i_y \in \{0, \ldots, N-1\}$ indicating the position of two evaluation points within each element
         of $\mathbf{x}$ and $\mathbf{y}$, respectively;
Input  : input and output parameter sets $(p, q, N)$ and $(\overline{p}, \overline{q}, \overline{N})$;
Input  : decomposition base $\mathfrak{b} = \log(\overline{p})$;
Input  : a packing key switching key ksk
Output: Repacked ciphertext
// Extract the evaluation points to LWE samples
1 for $j \leftarrow 0$ to $d$ do
2     $\hat{x}_j \leftarrow$ ExtractLWE$(x_j, i_x)$
3     $\hat{y}_j \leftarrow$ ExtractLWE$(y_j, i_y)$
// Decompose each LWE sample in $k$ new samples, each encrypting $\log(\overline{p})$ bits of the evaluation
   points
4 $k \leftarrow \lceil \log(p) / 2^{\mathfrak{b}} \rceil$
5 for $j \leftarrow 0$ to $D-1$ do
6     for $i \leftarrow 0$ to $k-1$ do
7         $\tilde{c}_{kj+i} \leftarrow$ Decompose$_{\overline{p},\overline{q}}(\hat{x}_j, i)$
8         $\tilde{c}_{k(j+D)+i} \leftarrow$ Decompose$_{\overline{p},\overline{q}}(\hat{y}_j, i)$
// Pack all LWE samples in a single RLWE
9 return PackingKeySwitching$([\tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{2Dk}]$, ksk$)$
```

- ExtractLWE: Given an RLWE sample c encrypting a polynomial $m = \sum_{i=0}^{N-1} m_i X^i$ under key s, ExtractLWE(c, i) produces an LWE sample encrypting m_i under key s', where s' is the vector interpretation (*i.e.* the array of coefficients) of s.
- Decompose: Given an LWE sample c encrypting a message $m \in \mathbb{Z}_p$ with ciphertext modulus q, the procedure Decompose$_{\overline{p},\overline{q}}(c, i)$ decomposes the message in base $\overline{p}$ and produces an LWE sample encrypting its i-th most significant digit with ciphertext modulus $\overline{q} < q$. This procedure is implemented as modular reduction followed by a real division (ModDown), as shown in Equation 6:

$$\textsc{Decompose}_{\overline{p},\overline{q}}(c, i) \mapsto \left[\left\lfloor [c]_{(q/\overline{p}^i)} \cdot \frac{\overline{q}}{(q/\overline{p}^i)} \right\rceil \right]_{\overline{q}}. \tag{6}$$

- PackingKeySwitching: Given a list of LWE samples c_i encrypting messages m_i, respectively, for $i \in [\![0, \overline{N}-1]\!]$ and a key switching key ksk, the PackingKeySwitching produces an RLWE sample C encrypting the polynomial $m = \sum_{i=0}^{\overline{N}-1} m_i X^i$. In this process, the dimension of C is defined by ksk.

We select $\overline{p}$ and its associated dimension $\overline{N}$ as the smallest possible values that allow the encryption of a string of $2\log(|\mathbb{F}_{p^D}|)$ bits while providing a 128-bit security level. Additionally, the decomposition algorithm requires the ciphertext modulus q to be divisible by $\overline{p}$, which, for simplicity, we achieve by mod-switching the ciphertext to a power-of-two modulus before the repacking. Table 3 presents the main practical choices for these parameters. We note that, depending on the size of p, some of them may not require decomposition but would still benefit from the repacking.

Table 3: Practical choices for FHE parameters for the repacking procedure. In this table, k is the module-LWE dimension and q is the ciphertext modulus. All parameters are estimated for the 128-bit security level, and the decryption cost is measured in the number of multiplications.

Parameter Set	k	N	$\log_2(q)$	Size (bytes)	Decryption Cost
$\mathfrak{P}_0$	1	512	12	8192	5120
$\mathfrak{P}_1$	2	512	25	12288	5632
$\mathfrak{P}_2$	1	1024		16384	11264
$\mathfrak{P}_3$	4	512	52	20480	6656
$\mathfrak{P}_4$	2	1024		24576	12288
$\mathfrak{P}_5$	1	2048		32768	24576

To minimize the noise generated by the repacking, we always run the PackingKeySwitching with a larger parameter set and perform another key switching to reduce the dimension at the end. For example, we could first perform the folding using $\mathfrak{P}_5$; then, the repacking would extract and decompose the samples still using $\mathfrak{P}_5$. PackingKeySwitching would repack them using $\mathfrak{P}_1$. Finally, before committing to the codeword, the prover would run another key switching to reduce from $\mathfrak{P}_1$ to $\mathfrak{P}_0$. This final reduction may not improve performance considerably (as decryption in $\mathfrak{P}_1$ is only 10% slower than in $\mathfrak{P}_0$), but it would reduce the size of the ciphertext by 33%, accelerating hash operations and reducing proof size.

D.3 Recomposition

The repacking process is fundamental for minimizing the impact of the HE overhead on FRI, but, if used with the decomposition, it also introduces some challenges for the verifier to recover the evaluation points in $\mathbb{F}_{p^D}$. Algorithm 3 shows the full recomposition process, and the next paragraphs describe the main challenges it addresses.

We first recall the definition of the phase function.

Definition 17 ([CGGI20]) *Let* `HE` *be a Homomorphic Encryption scheme as defined above. For a concrete instantiation, that is to say, for a fixed key pair* ($\mathtt{sk}, \mathtt{pk}, \mathtt{evk}$)*, and for a ciphertext* $\mathtt{ct} = (\mathtt{ct}_0, \mathtt{ct}_1)$ *that is the output of the* `HE.Enc` *algorithm (and has possibly been computed on) defined modulo some* q*. We define the phase function as*

$$\psi(\mathtt{ct})_{\mathtt{sk}} = \mathtt{ct}_1 - \mathtt{sk} \cdot \mathtt{ct}_0 \pmod{q}.$$

```
Algorithm 3: Recomposition
  Input  : Repacked RLWE ciphertext $\texttt{ct} = (\texttt{ct}_0, \texttt{ct}_1)$
  Input  : Secret key $\texttt{sk}$
  Input  : Decomposition base $\mathfrak{b} = \log(\overline{p})$, and message scaling factor $\Delta$
  Output: An array of points $\hat{v} \in \mathbb{F}_p^{2D}$
1  $v \leftarrow \frac{\texttt{ct}_1 - \texttt{ct}_0 \cdot \texttt{sk}}{2^{\mathfrak{b}+2}}$                    // Calculate the phase
2  $k \leftarrow \lceil \log(p) / 2^{\mathfrak{b}} \rceil$                    // Recompose each point in $\mathbb{F}_p$
3  for $j \leftarrow 0$ to $2D - 1$ do
4      $\hat{v}_j \leftarrow \left\lceil \frac{v_{kj+k-1}}{4} \right\rfloor$
5      $x \leftarrow \left\lfloor \frac{\hat{v}_j}{2^{\mathfrak{b}-2}} \right\rfloor$
6      for $i \leftarrow k-2$ to $0$ step $-1$ do
           // Calculate the noise propagated by the previous digit
7          $c \leftarrow \text{SignExtend}\left(x - [v_{kj+i}]_4\right)$
           // Remove the noise and scale
8          $\hat{v}_j \leftarrow \hat{v}_j + \left\lfloor \frac{v_{kj+i}+c}{4} \right\rfloor \cdot 2^{\mathfrak{b}(k-1-i)}$
           // Extract the two most significant bits to x
9          $x \leftarrow \left\lfloor \frac{(v_{kj+i}+c)}{2^{\mathfrak{b}}} \right\rfloor$
10     $\hat{v}_j \leftarrow \left\lfloor \frac{\hat{v}_j}{\Delta} \right\rceil$
11 return $\hat{v}$
1  Procedure SignExtend($x$)
2      if $x = 3$ then
3          return $-1$
4      else if $x = 1$ then
5          return 1
6      return 0
```

Message recomposition. During the repacking process, we extract digits relying on the fact that $\overline{p}$ divides q. Our plaintext space, on the other hand, is defined by a prime modulus p following FRI requirements, which is not divisible by $\overline{p}$. Therefore, we will interpret the extracted digits not as digits of the message, but as digits of the *phase function* (Definition 17). Recall that the phase function is computed modulo q, and we have that $\overline{p} \mid q$. At the time of decryption, the verifier needs therefore to first remove the noise and recompose the phase before dividing by the scaling factor Δ.

Digit decomposition noise. Considering the decomposition process presented in Equation 6, at the extraction of the i-th digit, the $(i+1)$-th digit remains in the sample and is treated as noise. This noise increases the probability of a decryption error significantly once added to the key-switching noise. To avoid this problem, we select parameters that allow us to encrypt at least two extra bits. To give a concrete example, if $\overline{p} = 2^8$, we select the HE parameters to encrypt a 10-bit message, so that we also preserve the bits following the least significant bit (LSB) of our message, as illustrated in Figure 5). At high level, one can view these two extra bits as "redundant", that allow for a higher noise growth, without causing a decryption error. Notice that, for the $(k-1)$-th digit, these *redundant bits* will be zero, as there is no k-th digit, counting from zero. For all other digits, the *redundant bits* of the i-th digit should be the two most significant bits (MSBs) of the $(i+1)$-th digit. Considering this, the verifier decrypts the digits sequentially from the least to the most significant and uses the MSBs of the previous digit and the redundant bits to correct for possibly propagated errors. Notice that these bits are supposed to be the same, and hence this process will not change the redundant bits if there is no propagated noise. Otherwise, if the noise propagates up to the redundant bits (also shown in Figure 5), it will correct the message to its original value.

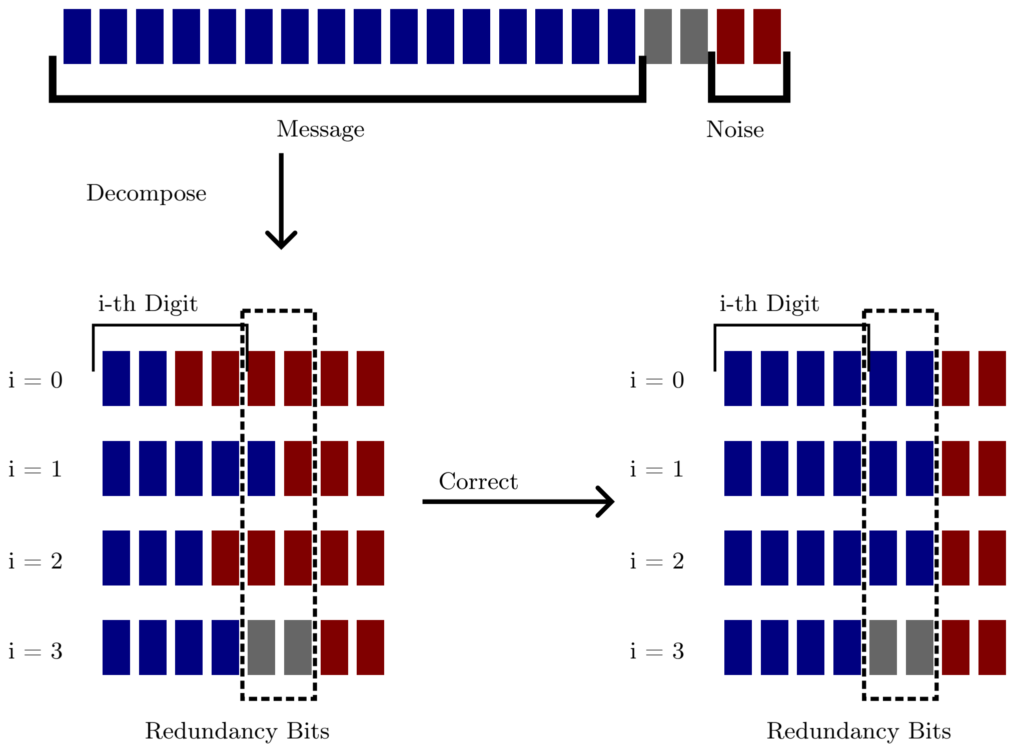


Fig. 5: Example of decomposition and error correction. Notice that the process for correcting using the redundant bit only occurs after the decryption.

D.4 Using bootstrapping

Besides reducing costs for the verifier, the repacking technique also allows us to produce ciphertexts with a small plaintext modulus, regardless of the size of the base field we adopt for FRI. This in turn makes it possible to bootstrap these ciphertexts, which would otherwise be prohibitively expensive for some of the sizes of base fields we consider for FRI (we consider $|\mathbb{F}_p|$ ranging from 2^{16} up to 2^{50}). Bootstrappings are particularly useful for our construction in two ways. First, they enable the prover to remove the digit decomposition noise, which leads to better parameters and hence better performance for the verifier. Bootstrapping also makes the error correction part of Algorithm 3 unnecessary. Second, it can also enable us to achieve the notion of Circuit Privacy defined in Definition 14. In particular, we know that we can use bootstrapping as a rerandomization algorithm [DS16].

E HE-FRI performance in the number of operations

Table 4 shows the impact of our optimizations for FRI with a constant depth equal to 4. While the number of operations grows for the prover, the growth for the verifier (Query and RS decode procedures) only depends on the RLWE decryption overhead, which is independent of the size of n. Furthermore, all our optimizations allow for tradeoffs between the number of operations and the depth of the circuit for the prover.

Table 4: Cost and depth comparison between FRI [BBHR18] and HE-FRI. We only consider the number of cost-dominant operations, which is the multiplication in $\mathbb{F}_{p^D}$ for all procedures except for the Merkle Tree and query phases, where the cost of committing and verifying the Merkle tree dominates. For HE-FRI, we also include the overhead of RLWE encryption and decryption operations, given by the factors $\mathfrak{P}_0$ and $\mathfrak{P}_1$ which refer to the decryption costs presented in Table 3. In turn, $\mathfrak{P}'_i = \mathfrak{P}_i \log_2(p^D)/\mathsf{b}$ is the amortized cost of decrypting each element in $\mathbb{F}_{p^D}$ using the repacking method (Section 6.5), where b is the number of bits this parameter set can encrypt. Finally, to simplify notation, let $m' = \lceil \log_2(m) \rceil$, where m is the number of linearity checks defined by the security parameters of FRI.

Procedure	Cost		Depth	
	FRI	HE-FRI	FRI	HE-FRI
RS encode	$2^n\,(n)$	$2^{n+1}\sqrt{2^n}$	n	2
Folding	$2^{n+1} - 2^{m'+1}$	$2^n\,(n - m')$	$(n - m')$	1
Merkle Tree	$2^{n+1} - 2^{m'+1}$	$\left(2^{n+1} - 2^{m'+1}\right)\mathfrak{P}_1$	0	1
Query	$2m\,(n - m')$	$m\,(n - m')\,\mathfrak{P}_0/D$	0	0
RS decode	$2^{m'}m'$	$2^{m'}\,(m' + \mathfrak{P}'_1)$	0	0

E.1 Operation counting

As we built HE-FRI over the FRI implementation of Szepieniec *et al.* [S+21], we extended it with our proposals and instrumented it to measure the number of operations. We only consider the number of cost-dominant operations in $\mathbb{F}_{p^D}$, and we measure them for each level of depth of the algorithm individually. We evaluate each phase of the protocol separately, leaving the tradeoffs between them to be addressed when considering practical instantiations.

Verifier Performance Figure 6-a shows the number of $\mathbb{F}_{p^D}$ elements received by the verifier during FRI, a key performance metric since each element requires a Merkle Tree check, the most expensive verifier operation. In HE-FRI, each element also needs decryption, adding significant overhead, as shown in Figure 6-b. We define our comparison baseline based on two scenarios:

– Outsourcing: A client (the verifier) wants to outsource a polynomial evaluation to a (usually more powerful) server. In this case, the baseline for comparison is a local evaluation, *i.e.*, one in which the verifier receives the entire polynomial encrypted from the server, decrypts it and evaluates it themselves. For this scenario, HE-FRI provides gains for polynomials larger than 2^{17}, as Figure 6-b shows.
– Zero-Knowledge: A client (verifier) wants to know the evaluation of a polynomial P at some points x, but the server does not want to reveal any information about P besides its evaluation at x. In this scenario, local evaluation is not an option, and HE-FRI, or some other verifiable polynomial evaluation method over encrypted data, must be used.

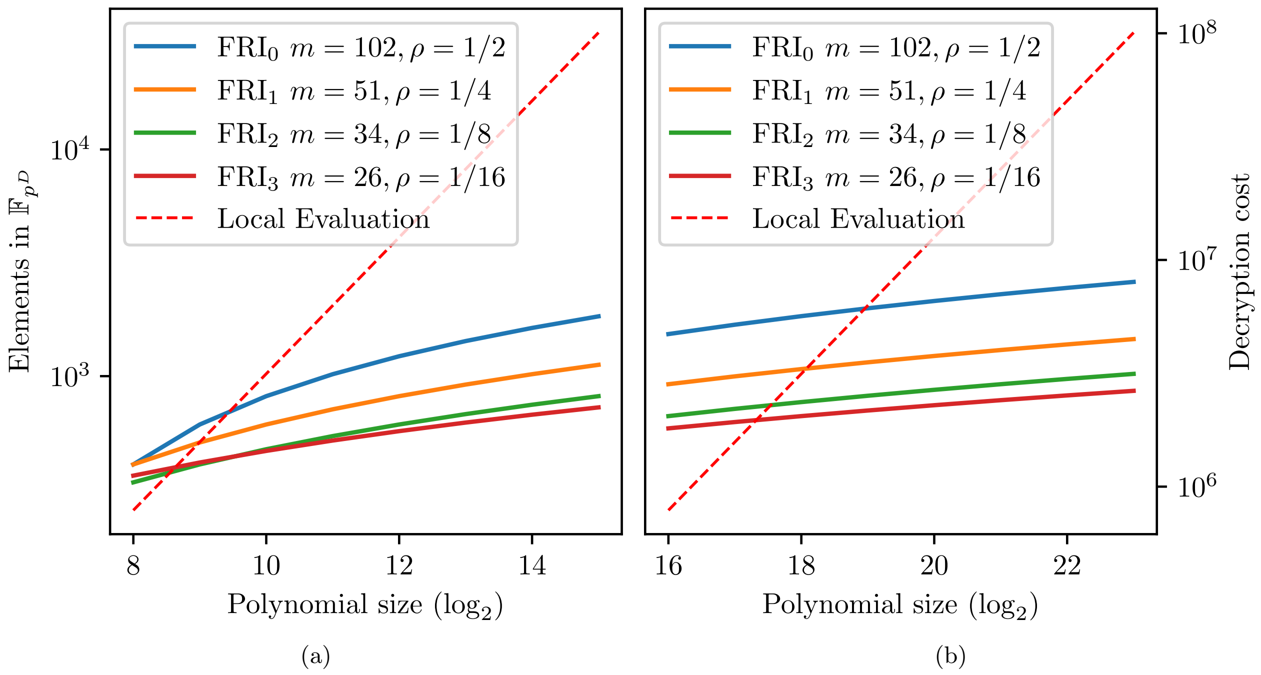


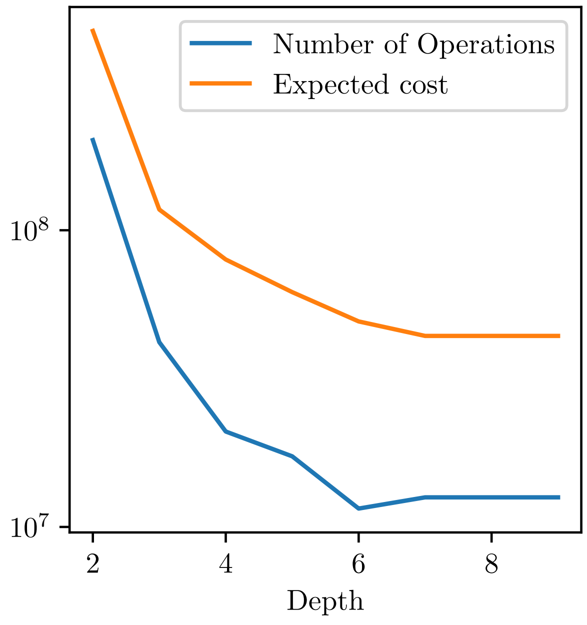
Fig. 6: Estimated cost for the Verifier in HE-FRI (a) in the number of elements in $\mathbb{F}_{p^D}$ received and (b) in the number of multiplications required to decrypt them.

Prover Performance Once we establish the minimum size for which gains with HE-FRI are possible, our goal becomes minimizing the cost for the prover. Taking, for example, a polynomial of size 2^{17} and $\rho = 1/2$, the RS encoding and the folding algorithm would have depths 18 and 17, respectively in the the original implementation of FRI. Even for a small value of p, let us suppose $p = 2^{20}$, depth $(18 + 17) = 35$ would require a ciphertext modulus with at the least $20 \cdot 35 = 700$ bits. In turn, this modulus would require dimension $N = 2^{15}$, and each ciphertext would weigh at least 7 megabytes (MB). If we consider batched FRI, memory could already be a problem for this size of ciphertext depending on the number of input polynomials. Nonetheless, even in a non-batched scenario (or one with a small number of polynomials), increasing the ciphertext modulus has typically a quadratic impact on the HE evaluation performance.

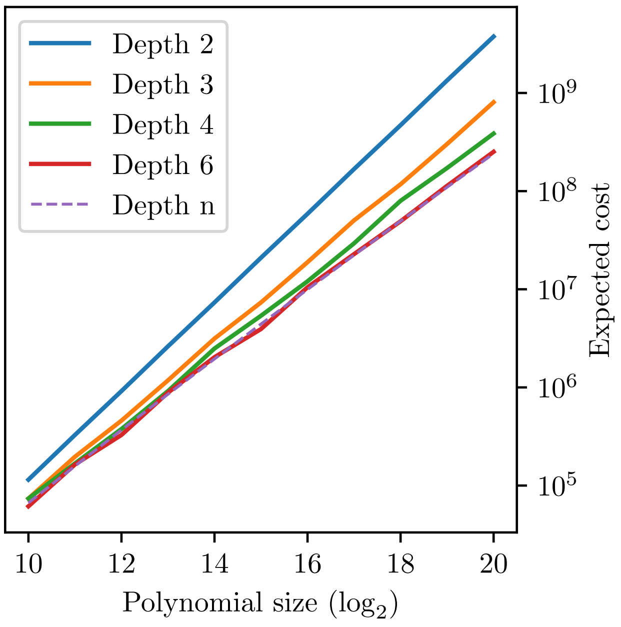
In Section 6, we showed how to solve this problem with a series of optimizations that enable to run FRI with significantly smaller, fixed depth. In principle, we could evaluate FRI at depth only 1, but decreasing depth comes at the expense of increasing the total number of operations. Considering this, we address several trade-offs between depth, performance, and memory use enabled by our techniques. We present several options of depth that should suit most applications, but choosing among them is ultimately an application-dependent choice.

For the Folding, as we discussed in Section 6, minimizing depth to 1 only increases complexity from $O(2^n)$ to $O(2^n \log 2^n)$, preserving the overall complexity of FRI. In practice, this increment is also negligible for the prover as the cost of the folding is dominated by the number of executions of the repacking, which remains linear in the input size. Therefore, for practical purposes, we always implement folding with depth 1 and focus our efforts on the trade-offs enabled by the shallow NTT during the RS encoding phase.

Reed-Solomon code encoding Increasing arity is a standard way of enabling shallower NTT circuits. Finding the optimal arity, however, depends both on the general goal as well as on the adopted cost model. For HE-FRI, while we want to minimize depth, we only use multiplications by constants, a procedure in which performance is linear in the depth. Therefore, we model the cost of our NTT as the number of multiplications weighted according to the depth in which they occur in the algorithm. Specifically, a multiplication at depth k' has cost $(k - k')$, where k is the total depth of HE-FRI. Notice that operations happening at a higher depth are considered linearly less expensive, as the ciphertext levels were already consumed by operations from shallower levels.



(a) NTT for an input of size 2^{18}



(b) Optimal expected cost

Fig. 7: Expected cost of our NTT implementation.

Figure 7-a shows the results for an NTT with an input of size 2^{18}. For each value of maximum depth, we choose arity to minimize the expected cost (even if it would increase the number of operations). While the cost decreases quickly with the depth at first, it stabilizes with depth as low as just 6. Based on this, we selected a few different values for the maximum depth and extended this analysis to different sizes of polynomials. Figure 7-b shows the results as well as the estimated cost for a standard depth-n (radix-2) NTT, using the same cost model. This second result explains depth 6 as a stability point for the cost since it reaches essentially the same performance level provided by the depth-n NTT.

F Complete results

This section presents the complete practical results. Execution times for the verifier are an average of 100 executions.

Table 5: Results for batches of 4096 polynomials, for up to 32 threads.

Polynomial Size ($\log_2$)	FRI_3				FRI_0			
	Prover (s)	Verifier (ms)	λ	Memory (GB)	Prover (s)	Verifier (ms)	λ	Memory (GB)
6	1.33	3.77	120	1.2				
7	2.74	4.10	120	2.0	0.20	7.08	120	0.5
8	5.93	4.46	120	3.5	0.48	8.33	120	0.8
9	13.38	4.82	120	6.4	1.16	9.64	120	1.2
10	31.21	5.12	118	12.2	2.59	11.04	120	2.1
11	78.98	5.61	116	23.7	5.45	12.29	120	3.7
12					13.74	13.72	120	6.9
13					33.92	15.36	118	13.2
14					85.96	17.01	116	25.9
15					207.76	18.51	114	51.0

Table 6: Results for batches of 8192 polynomials, for up to 32 threads.

Polynomial Size ($\log_2$)	FRI_3				FRI_0			
	Prover (s)	Verifier (ms)	λ	Memory (GB)	Prover (s)	Verifier (ms)	λ	Memory (GB)
6	2.97	6.66	191	2.4				
7	6.24	6.79	191	4.1	0.46	16.64	191	1.0
8	15.46	7.07	191	7.3	1.10	18.04	191	1.6
9	36.27	7.76	191	13.6	2.56	19.35	191	2.5
10	85.73	7.97	191	26.1	6.11	20.93	191	4.3
11	186.69	8.60	191	50.7	13.71	23.13	191	7.8
12					32.49	23.74	191	14.6
13					80.74	26.54	191	28.1
14					188.36	26.99	191	55.0
15					486.71	29.09	191	108.9