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 in Python and show that, for an encrypted Reed Solomon codeword with degree bound 2^{11} and rate $1/16$ in a (plaintext) field of size 2^{256}, we can run FRI's commit phase in just 43 minutes on a single thread on a `c6i.metal` instance (which could be reduced to less than a minute in a multi-threaded implementation in a large server). Verification takes less than 0.2 seconds, and, based on micro-benchmarks of the employed techniques, we show it could be up to 100 times faster in a fully optimized implementation.

1 Introduction

Cloud computing has changed the way citizens and businesses store, access and process their data, including very sensitive information. Whereas there are numerous usability and economic benefits to the outsourcing of these tasks to remote servers, this paradigm introduces 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 computation. Since the first scheme was 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). VC 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.

Early on, the cryptographic community realized that VC and HE complement each other very well, since the limitations of one technique 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 to this problem was proposed by Gennaro, Gentry and Parno [GGP10] in 2010, and employs a heavy combination of Yao's garbled circuit for a 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 as well as it introduces the need to keep a secret verification key hidden from the prover, hence eliminating the possibility for public verifiability.

The work of [FNP20] improves the expressiveness of [FGP14], by allowing to efficiently compute circuits of (arbitrary) constant depth. Nevertheless, they are restricted to deal with a very narrow subset of inefficient HE schemes: a variant of the BV [BV14] scheme, where the integer ciphertext modulus q has to match the (prime) order of the source groups in the underlying pairing-based SNARK.

Both [GNS23] and [BCFK21] overcome the barrier on 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, which introduces large overheads. While the addition of ciphertexts can be easily

emulated (as an addition of elements in the ciphertext space R_q^2), the product of ciphertext is not as algebraically structured and becomes hard to emulate. This is 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 the value of 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 quickly becomes prohibitive 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 according to that. 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 be able to evaluate.

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 $\mathcal{O}_{\texttt{HE}}$, which are oracles to data that is homomorphically encrypted. While the prover $\mathcal{P}$ does not know *what* are the actual plaintexts they are computing on, $\mathcal{P}$ knows how to arrange

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

them into oracles (i.e. $\mathcal{P}$ can place $\texttt{HE.Enc}(x)$ into an oracle $\mathcal{O}_{\texttt{HE}}$, rather than x into $\mathcal{O}$).

Whereas the modified IOP (which we refer to as 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, but it is also very simple to reduce the security of an HE-IOP to that of its corresponding IOP (Theorem 2). Furthermore, we also adapt several results in the literature compiling different variants of (zk)IOPs into (zk)SNARKs [COS20, BGK+23]. Even more, our resulting zkSNARKs are plausibly post-quantum, since so are the BCS transformation [BCS16, CMS19] and all the efficient HE schemes that 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 to Reed-Solomon Codes, which is interpreted either as a low degree test or a correlated agreement test [BCI+20, BGK+23]. As is 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 provide experimental data (Section 7).

Aligning security parameters. The first obstacle that we encounter is how to enable 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 for the HE scheme would result in unmanageable parameters and significantly affect the performance. We circumvent this by emulating the arithmetic of $\mathbb{F}_{p^D}$ from that of $\mathbb{F}_p$ by using D ciphertexts, each of which encrypts elements from $\mathbb{F}_p$.

Reducing depth and exploiting HE packing. A second challenge to the homomorphic evaluation of FRI is in the multiplicative depth of the standard techniques used to implement it. Even though it only performs multiplications between plaintexts and ciphertexts, the size of these plaintexts increases the noise almost as much as a ciphertext multiplication for some choices of $\mathbb{F}_p$. A typical implementation of FRI would have depth $2n$ for an input of size 2^n, which represents a challenge for HE schemes, as performance degrades quickly with the depth. We solve this problem by introducing low-depth versions of every sub-routine required to evaluate FRI. Particularly, we show how to perform Reed-Solomon codeword encoding (RS encoding) with small fixed depth, as opposed to the traditional methods that require depth n. We also propose a "Shallow Fold" algorithm to replace FRI's standard Fold operation, which reduces the depth to 1

[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 constructions for such primitive are very inefficient.

(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 asymptotics of the overall procedure. Additionally, we exploit the packing capabilities of the HE scheme to further reduce the cost of the RS encoding. In more detail, we consider packing methods that trade off memory consumption and execution time to accelerate both single and batched polynomial commitments.

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. Here, if the ciphertexts are fully packed, an overhead of at least $N/(2D)$ is introduced. In order to avoid this, we introduce a repacking procedure that extracts the evaluation points from the RLWE samples of dimension N and repacks them in another RLWE sample, but of a much smaller dimension, reducing decryption costs in 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 input size 2^n or of the HE parameters adopted by the prover.

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 in Python 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 implementing the BGV and TFHE schemes. While we provide efficient implementations for the homomorphic evaluated techniques, we maintain all the plaintext arithmetic as well as the high-level procedures in Python. This gives our implementation flexibility and allows one to easily extend and test our techniques. At the same time, however, it also introduces a significant overhead compared to a dedicated (fully optimized) implementation and limits our ability to control computational resources such as memory and parallelization.

Even with these restrictions, our implementation runs FRI's commit phase in 43 minutes for encrypted codewords of size 2^{15} with degree bound $d = 2^{11}$. This execution time is, however, in a single-threaded execution, whereas all our techniques enable trivial parallelization, which could lower the execution time to less than a minute in a large server. Verification is much faster and takes only

0.2 seconds. In this case, we also show that the execution is dominated by the (Python-implemented) plaintext arithmetic and that it could be accelerated by up to two orders of magnitude in a dedicated fully optimized implementation.

1.2 Comparison with concurrent work

In concurrent work [GGW23], Garg, Goel and Wang provide 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 `HE`, their work considers more general ways to hide these values from the prover, such as homomorphic commitments and group exponentiation. They group all of these hiding mechanisms into 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 `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 constitute a significant obstacle for noise management in practice (see Section 6).

Whereas the focus of [GGW23] is more theoretical and does not include an implementation, our work comes from a more practical stance in the current state of the art for verifiable computation on encrypted data. Besides different technical optimizations and trade-offs informed by our experimentation, there are two important aspects that we consider and [GGW23] does not: our compilers include scenarios where the prover provides some inputs in zero knowledge and we use FRI more directly (e.g. by compiling δ-correlated IOPs into (zk)SNARKs [BGK+23]) rather than by going through a polynomial commitment abstraction. The former greatly improves the parameters of the `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 `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

Throughout this document, when we refer to a ring R, we assume that it is finite, commutative and that it has a multiplicative identity element $1_R \neq 0_R$.

For ease of notation, we will write 1 to mean 1_R, and 0 to mean 0_R whenever the context is clear. We use $R[\mathtt{X}]_{\leq d}$ to refer to polynomials with coefficients in 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 $a \leftarrow A$ to denote sampling an element a from a distribution A. When A is a set rather than a distribution, we write $a \xleftarrow{\$} A$ to mean that we are 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 use the abbreviation PPT for Probabilistic Polynomial Time. We let λ denote a computational security parameter. We denote computational indistinguishability by $\overset{\text{c}}{\approx}$ 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 (mod 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 (HE) scheme* $\mathtt{HE}$ *over a set of admissible circuits* $\widehat{\mathcal{C}\mathrm{irc}}$ *consists of the following algorithms.*

- $(\mathtt{pp}, \mathcal{C}) \leftarrow \mathtt{HE.Setup}(1^\lambda, \mathcal{M}, \widehat{\mathcal{C}\mathrm{irc}})$: *Given a message space* $\mathcal{M}$*, a set of admissible circuits* $\widehat{\mathcal{C}\mathrm{irc}}$*, 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* $m \in \mathcal{M}$*, the public parameters* $\mathtt{pp}$ *and the ciphertext space* $\mathcal{C}$*, output an encryption* $\mathtt{ct}$ *of* m
- $m' \leftarrow \mathtt{HE.Dec}(\mathtt{sk}, \mathtt{ct})$: *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* $\hat{C} \in \widehat{\mathcal{C}\mathrm{irc}}$*, 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. We note that the above definition applies to both exact and approximate HE schemes.

Definition 2. *Let* $\mathtt{HE}$ *be a homomorphic scheme as in Definition 1. We say that* $\mathtt{HE}$ *is* correct *if the following equation*

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

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 $\mathtt{HE}$ 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.

Definition 3. *(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.*

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

We say that $\mathtt{HE}$ *is semantically secure if* $\mathbf{Adv}^{\mathtt{HE}}_{\mathcal{A}}$ *is negligible in* λ*, for every PPT adversary* $\mathcal{A}$*.*

Finally, we define the phase function, defined in [CGGI20].

Definition 4. *Let* $\mathtt{HE}$ *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* $\mathtt{HE.Enc}$ *algorithm (and has possibly been computed on) defined modulo some* q*. We define the phase function as*

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

2.4 Reed Solomon Codes

Reed-Solomon ($\mathtt{RS}$) 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 $\Delta(\boldsymbol{u}, \boldsymbol{v}) \coloneqq |\{u_i \neq v_i | i \in \{1, \ldots, n\}|/n$. For a set of vectors $S \subset \mathbb{F}^n$ and any vector $\boldsymbol{u} \in \mathbb{F}^n$, we define $\Delta(\boldsymbol{u}, S) = \Delta(S, \boldsymbol{u}) \coloneqq \min_{\boldsymbol{v} \in S}\{\Delta(\boldsymbol{u}, \boldsymbol{v})\}$. 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 refer to it as being within the *unique decoding radius*, and when $\delta < 1 - \sqrt{\rho}$, we say that δ is within the Johnson bound.

Definition 5 (Correlated agreement). *Let* $V = \mathtt{RS}[\mathbb{F}, L, d]$ *and let* $W = \{w_1, \ldots, w_k\} \subseteq \mathbb{F}^{|L|}$*. Let* $\delta \in (0, 1)$*. 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* $v_1, \ldots, v_k \in V$ *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 6. *An indexed relation $\mathcal{R}$ is a set of triples $(\mathtt{i}, \mathbb{x}; \mathbb{w}) \in \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$. The string $\mathbb{x}$ is the called input, statement or instance, the string $\mathbb{w}$ is called the witness and the string $\mathtt{i}$ is an index. The index can be thought as something that is fixed at setup time, and chooses among a universe of binary relations $\mathcal{R}_{\mathtt{i}} = \{(\mathbb{x}; \mathbb{w}) : (\mathtt{i}, \mathbb{x}; \mathbb{w}) \in \mathcal{R}\}$.*

A good example for indexed relations, in the setting of holographic proofs, is an indexed relation for circuit satisfiability, where the index $\mathtt{i}$ 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 7 ([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 $(\mathtt{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the index $\mathtt{i}$ 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 8. *A μ-round* holographic interactive oracle proof (hIOP) *for an indexed relation $\mathcal{R}$ is a tuple of PPT interactive algorithms $\Pi = (\mathcal{P}, \mathcal{V})$, together with a deterministic polynomial-time algorithm* $\mathtt{Ind}$ *(the indexer). It proceeds in two phases:*

- *In an offline phase, given an index $\mathtt{i}$, the indexer* $\mathtt{Ind}$ *computes an encoding of it,* $\mathtt{Ind}(\mathtt{i})$.
- *In an online phase, $\mathcal{P}(\mathtt{Ind}(\mathtt{i}), \mathbb{x}, \mathbb{w})$ and $\mathcal{V}^{\mathtt{Ind}(\mathtt{i})}(\mathbb{x})$ 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*

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

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 $\mathcal{P}^*$, we have that*

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

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 5). On top of checking all the received oracles correspond indeed to δ-correlated maps (which we capture by the relation in Definition 9), it is necessary to verify some algebraic equalities involving some evaluations of those maps. These are made concrete in Definition 10.

Definition 9 ([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:*

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

Definition 10 (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 $(\mathtt{x}, \tau)$ generated during Π, let $\mathtt{Words}(\mathtt{x}, \tau)$ be the words from $\mathbb{F}^L$ that fully describe the oracles appearing in $(\mathtt{x}, \tau)$. 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 $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$:*
  - *In the last round of interaction between $\mathcal{P}(\mathtt{Ind}(\mathtt{i}), \mathtt{x}, \mathtt{w})$ and $\mathcal{V}^{\mathtt{Ind}(\mathtt{i}),\mathtt{CoAgg}(\delta)}(\mathtt{x})$, 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:*

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

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

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

    Check 2 *Assert whether the maps*

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

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

1. *If $\mathtt{x} \notin \mathcal{L}_{\mathcal{R}_\mathtt{i}}$, then $(\mathtt{x}, \emptyset) \in \mathcal{D}(\mathtt{i})$, 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* $\Pr_{c \xleftarrow{\$} C_i}[(\mathbb{x}, \tau, m, c) \notin \mathcal{D}(\mathbb{i})] \leq \epsilon(\mathbb{i})$ *for every possible next prover message* m.

Definition 12. *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 $\mathtt{Ext}(\mathbb{i}, \mathbb{x}, \tau, m)$ *outputs a valid witness for* $\mathbb{x}$.

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

Definition 13. *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* $\mathcal{V}^* = \mathcal{V}^{\mathtt{Ind}(\mathbb{i})}(\mathbb{x})$.

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 14. *Let* $\mathtt{RS}$ *denote the family of Reed Solomon codes* $\mathtt{RS}[\mathbb{F}, L, d]$*. 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* $\mathtt{RS}$ *is an IOP with the following modifications*

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

The following 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 (such as batched FRI) to produce a 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. $\delta < (1 - \rho)/2$.

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

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

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 intended to be used in practice. In the following (Definition 15), we tweak their definition and syntax to one which will better capture 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 $\mathbb{w}_\mathcal{P}$ is an input to be specified by the prover. Notice that $\mathbb{w}_\mathcal{P}$ could consist of any

combination of plaintexts and ciphertexts, which would be revealed to $\mathcal{V}$ during verification. Unless there is some threshold decryption method to stop $\mathcal{V}$ from performing unspecified decryptions during the protocol execution, this means that all values within $\mathtt{w}_{\mathcal{P}}$ are learned by $\mathcal{V}$. Hence, the most sensible approach for an outsourcing scenario with a single client would be to have $\mathtt{w}_{\mathcal{P}}$ consist only of plaintexts.

If $\mathtt{w}_{\mathcal{P}}$ has to remain hidden, we can instead think about it as some private witness, which we want to mix with the encryptions of $\mathbb{x}$ in a zero-knowledge fashion. For example, one could think about a private Machine-Learning-as-a-Service, where the client sends encrypted queries $\mathtt{HE.Enc}(\mathbb{x})$ to the server, who applies their private model $\mathtt{w}_{\mathcal{P}}$. This approach, which was also pursued in [BCFK21, GNS23], has the advantage that the product between plaintexts (such as the values within $\mathtt{w}_{\mathcal{P}}$) and ciphertexts (the encryptions of $\mathbb{x}$ and the outputs that result from operating on them) is much cheaper than the product of ciphertexts.

Definition 15 (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*.*
- $(\boldsymbol{\sigma_x}, \mathsf{VK_x}) \leftarrow \mathsf{VC.ProbGen}(\mathsf{PK}, \boldsymbol{x})$*: 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* $\mathsf{VK_x}$*.*
- $\boldsymbol{\eta_y} \leftarrow \mathsf{VC.Compute}(\mathsf{PK}, \boldsymbol{\sigma_x}, \boldsymbol{w}, C)$*: Given a public key* PK *for a circuit* C*, the encoded input* $\boldsymbol{\sigma_x}$ *and input* $\boldsymbol{w}$*,* $\mathcal{P}$ *computes* $\eta_{\boldsymbol{y}}$*, which consists of an encoded version* $\boldsymbol{\sigma_y}$ *of the circuit's output* $\boldsymbol{y} = C(\boldsymbol{x}, \boldsymbol{w})$ *and data to answer challenges about that statement.*
- $(\mathtt{acc}, \boldsymbol{\sigma_y}) \leftarrow \mathsf{VC.Ver}\langle \mathcal{P}(\mathsf{PK}, \boldsymbol{\eta}_y), \mathcal{V}(\mathsf{SK}, \mathsf{VK_x})\rangle(C)$ *: The interactive verification algorithm uses the input-specific verification key* $\mathsf{VK_x}$*, the setup secret key* SK *and a proof* $\eta_{\boldsymbol{y}}$ *to return* $\boldsymbol{\sigma_y}$ *together with a bit* $\mathtt{acc} \in \{0,1\}$ *such that* $\mathtt{acc} = 1$ *if* $\mathsf{VC.Decode}(\boldsymbol{\sigma_y}, \mathsf{SK}) = C(\boldsymbol{x}, \boldsymbol{w})$ *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 $\mathtt{VC.}$ prefix in the different algorithms for ease of readability. In our work, we will always be interested in all of the following ones whenever $\boldsymbol{w}$ 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 $\boldsymbol{x}, \boldsymbol{w}$ of C the following probability equals $1 - \mathsf{negl}(\lambda)$.

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

– *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 $C(\boldsymbol{x}, \boldsymbol{w})$.
– 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 $\mathsf{Exp}^{Ver}_{\mathcal{A}}$ defined as $\Pr\left(\mathsf{Exp}^{Ver}_{\mathcal{A}}[VC, C, \lambda] = 1\right)$ is ϵ.

---

procedure $\underline{\textsc{Game}\ \mathsf{Exp}^{Ver}_{\mathcal{A}}}(\underline{VC, C, \lambda})$
  $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, C)$
  $\boldsymbol{x} \leftarrow \mathcal{A}(\mathsf{PK}, C)$
  $(\boldsymbol{\sigma_x}, \mathsf{VK}_{\boldsymbol{x}}) \leftarrow \mathsf{ProbGen}(\mathsf{PK}, \boldsymbol{x})$
  $\boldsymbol{\eta_y} \leftarrow \mathcal{A}(\mathsf{PK}, \boldsymbol{\sigma_x}, C)$
  $(\mathsf{acc}, \boldsymbol{\sigma_y}) \leftarrow \mathsf{Ver}\langle\mathcal{P}(\mathsf{PK}, \boldsymbol{\eta}_y), \mathcal{V}(\mathsf{SK}, \mathsf{VK}_{\boldsymbol{x}})\rangle(C)$
  $\boldsymbol{y} \leftarrow \mathsf{Decode}(\boldsymbol{\sigma_y}, \mathsf{SK})$
  If $\mathsf{acc} = 1 \wedge \nexists \boldsymbol{w} : C(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{y}$, **return** 1
  Else return 0.
end procedure

– *Verifier-Privacy.* For any valid circuit C, $\Pr\left[\mathsf{Exp}^{\mathcal{V}.Priv}_{\mathcal{A}}[VC, C, \lambda] = 1\right] \leq 1/2 + \mathsf{negl}(\lambda)$.

---

procedure $\underline{\textsc{Game}\ \mathsf{Exp}^{\mathcal{V}.Priv}_{\mathcal{A}}}(\underline{VC, C, \lambda})$
  $b \xleftarrow{\$} \{0, 1\}$
  $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{Setup}(1^\lambda, C)$
  $(\boldsymbol{x}_0, \boldsymbol{x}_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{PK}, C)$
  $(\boldsymbol{\sigma}_{\boldsymbol{x}_b}, \mathsf{VK}_{\boldsymbol{x}_b}) \leftarrow \mathsf{ProbGen}(\mathsf{PK}, \boldsymbol{x}_b)$
  $\hat{b} \leftarrow \mathcal{A}(\mathsf{state}, \boldsymbol{\sigma}_{\boldsymbol{x}_b}, \mathsf{VK}_{\boldsymbol{x}_b})$
return $b \stackrel{?}{=} \hat{b}$
end procedure

In previous works on verifiable computation over encrypted data, the goal of keeping $\boldsymbol{w}_{\mathcal{P}}$ 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 $\boldsymbol{w}_{\mathcal{P}}$ 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 16. *We say that a* `VC` *protocol is* honest-verifier prover-private *(HVPP) if there exists a PPT simulator $\mathcal{S}$ such that for every admissible circuit C:*

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

where $(\mathsf{SK}, \mathsf{PK}) \leftarrow \mathsf{VC.Setup}(1^\lambda, C)$ *and* $\mathbf{View}_\mathcal{V}(\mathsf{SK}, \mathsf{PK}, \mathbf{x}, \mathbf{w}_\mathcal{P}, \lambda)$ *denotes the view of $\mathcal{V}$ during an execution of the protocol on inputs $(\boldsymbol{x}, \boldsymbol{w}_\mathcal{P})$ and security parameter λ, that is* $(\mathsf{SK}, \mathsf{PK}, \boldsymbol{x}, \boldsymbol{r}; m_1, \ldots, m_e, \texttt{out})$ *where $\boldsymbol{r}$ is $\mathcal{V}$'s random tape, each m_i value is the i-th message $\mathcal{V}$ receives and* `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 $\mathsf{ProbGen}$ 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.

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 homomorphic encryption in Definition 17.

Definition 17 (HE-transformation). *Let $\langle \mathcal{P}(\mathbb{w}, \mathbb{x}), \mathcal{V}(\mathbb{x}) \rangle$ 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* `HE` *scheme as follows:*

- *There is a trusted setup* $(\texttt{pp}, \mathcal{C}) \leftarrow \texttt{HE.Setup}(1^\lambda, R_p, \widehat{\mathcal{C}\text{irc}})$*, where R_p splits into copies of $\mathbb{F}$ and $\widehat{\mathcal{C}\text{irc}}$ is a family of circuits that captures all necessary computation within the IOP as well as any preceding/posterior one (such as e.g. coming up with parts of the witness, or using an IOPP and the BCS transformation to compile into a SNARK).*
- *$\mathcal{V}$ has an additional input* $\texttt{sk} \leftarrow \texttt{HE.KeyGen}(1^\lambda, \mathcal{C}, \texttt{pp})$*, where $\mathcal{C} \in \widehat{\mathcal{C}\text{irc}}$.*
- *$\mathcal{P}$'s input $\mathbb{x}$ is replaced with its encryption* $\texttt{HE.Enc}(\mathbb{x})$*. Parts of $\mathbb{w}$ could be equally replaced by their homomorphic encryption.*
- *As a result, some oracles in the HE-IOP might now contain ciphertexts. We refer to then as* HE-oracles *or* encrypted oracles*. $\mathcal{V}$ has to decrypt the ciphertexts obtained through HE-oracles and perform the same checks it would have done in $\mathbb{F}$.*

Verifiable Computation over encrypted data

Let $\texttt{IOP}$ be an holographic Interactive Oracle Proof. Let $\texttt{HE}$ 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 $L_x = \lceil inp/\ell \rceil$, $L_w = \lceil wit/\ell \rceil$.

$(\texttt{pk}, \texttt{evk}, \texttt{sk}, K) \leftarrow \mathsf{Setup}(1^\lambda, C)$: Run $\texttt{HE}$ 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 $(\texttt{pp}, R_q^2) \leftarrow \texttt{HE.Setup}(1^\lambda, R_p, \widehat{\mathcal{C}irc})$.
3. $\mathcal{V}$ runs $(\texttt{pk}, \texttt{sk}, \texttt{evk}) \leftarrow \texttt{HE.KeyGen}(1^\lambda, \mathcal{C}, \texttt{pp})$.
4. Given an index $\texttt{i}$ for the circuit C, the indexer $\texttt{Ind}$ computes an encoding of it, $\texttt{Ind}(\texttt{i})$.

$\boldsymbol{\sigma}_{\mathbb{x}} \leftarrow \mathsf{ProbGen}(\mathbb{x}, \texttt{pk})$: $\mathcal{V}$ 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 $\boldsymbol{\sigma}_{\mathbb{x}} = \{\texttt{HE.Enc}(\texttt{pk}, m_{\mathbb{x},i})\}_{i=0}^{L_x-1}$. Set $\texttt{vk}_{\mathbb{x}} = \texttt{sk}$.

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

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

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

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

If we need to refer explicitly to the HE-IOP, we denote it as $\langle \mathcal{P}(\texttt{HE.Enc}(\mathbb{x}), \mathbb{w}), \mathcal{V}(\texttt{sk}, \mathbb{x})\rangle$*, 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, only with some negligible degradation due to the use of homomorphic encryption.

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

Proof. Completeness follows from the evaluation correctness of the 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 HE-IOP can be reduced to that of IOP as follows. Let $\mathcal{A}$ be an adversary against HE-IOP which an advantage bigger than an adding a factor $\mathsf{negl}(\lambda)$ to the one for the corresponding notion of the IOP ($\epsilon, \epsilon_{\texttt{rbr}}, \epsilon_k$ respectively). We will build an adversary $\mathcal{A}'$ against 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 HE scheme, obtaining in particular sk. Given any input, $\mathcal{A}'$ encrypts it under 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 $[\![f]\!]_{\texttt{HE}}$ he receives from $\mathcal{A}$ at every point, decrypts the answers using sk so as to recover f, and then sends the oracle $[\![f]\!]$ to $\mathcal{V}$. Clearly, if $\mathcal{A}$ succeeds, then 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 HE-friendly SNARKs using an HE-friendly low degree test (such as the 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 16), 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 $\mathtt{HE}$ 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[7] about $\boldsymbol{w}_{\mathcal{P}}$. We provide our own definition of the circuit-privacy notion that is best aligned with our HVPP goal.

Definition 18. *A homomorphic encryption scheme $\mathtt{HE}$ (see Definition 1) is* circuit-private *if there exists a rerandomization algorithm $\mathtt{HE.Rerand}(\mathtt{evk}, \mathtt{pk}, C, \mathtt{ct})$ and a simulator $\mathcal{S}_{\mathtt{HE}}$ such that, for any admissible circuit 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):*

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

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 to be given either a zero-knowledge RS-hIOP or a δ-correlated hIOP. In order 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 reduce the degree of the linear combination (since the coefficients of it are sampled afterwards), so this does not hurt soundness. On the other hand, the input to the LDT is now a *random* codeword, so we do not need to worry about its internals beyond what are 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 we discussed in Consideration #2 and the one we will present 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 for it. But, most importantly, the increase in the amount of queries

[7] 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 $\{C_{\boldsymbol{w}_{\mathcal{P}}}(\boldsymbol{x})\}_{\boldsymbol{w}_{\mathcal{P}}}$.

through the introduction of the LDT also degrades the query bound for zero knowledge (see Definition 13). 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 such 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, $\mathcal{V}$ also 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.[8] Packing becomes then as devastating (or 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 practical solution for this issue in Section 6.5.

Malicious verifier We will only briefly mention how to deal with a malicious verifier. At this point, rather than coming up with an ad-hoc "malicious-verifier prover-private" notion it would be best to model overall security as maliciously secure 2-party computation protocol, for which we refer to reader to e.g. [CCL15]. Besides all of the cautions that we employed for the honest verifier case, we need to ensure honest behaviour in the Setup and ProbGen steps. We can either assume a trusted setup or, otherwise, enforce the right behavior of the establishing party through the use of zero-knowledge proofs. That is also what we will do for the ProbGen step, and one can think about this solution as a GMW-style compiler [GMW87] from passive to active security.

In particular, for the ProbGen step, it is necessary to ensure that the verifier is providing valid ciphertexts, which in the context of lattice-based HE means that the noise must be within bounds. Using a zero-knowledge proof of knowledge (ZKPoK) ensures the right bounds for the cleartext and encryption randomness (see [DPSZ12, Figure 9] for an example).

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* HE-IOP *be its HE-transformation. Let $(\mathcal{P}_{LDT}, \mathcal{V}_{LDT})$ be an IOPP for the* RS *code $\mathtt{RS}[\mathbb{F}, L, d_c]$ 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* HE-IOPP *be its HE-transformation.*

Proceed as follows:

[8] 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.

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

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

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

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 following compiler allows to set the proximity parameter up to the Johnson bound, which improves its practical efficiency. It is the result of adapting one of the central theorems in [BGK+23] through the application of the HE transformation (Definition 17). The overall compiler that would use this theorem 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*

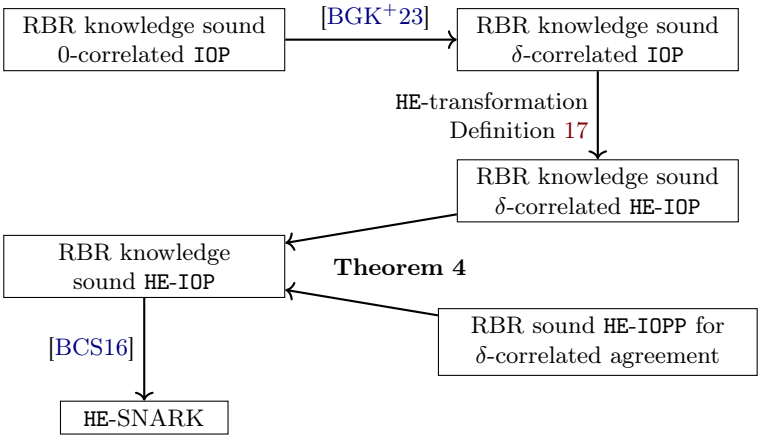


Fig. 2: Summary of our compilation flow for δ-correlated IOPs.

$\delta = 1 - \sqrt{\rho} - \eta$ 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 an SNARK (over encrypted data) with knowledge error $Q \cdot \max\{\epsilon/(2\eta\sqrt{\rho}), \epsilon_{\mathtt{CA}}\} + O(Q^2/2^\lambda)$.

Proof. This is a consequence of combining Theorem 2 and Theorem 1. ∎

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 test should exactly verify (δ-correlated agreement or merely closeness to a RS code). First of all, we need to think about how polynomials mix with $\mathtt{HE}$. For example, the following map

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

only corresponds to a degree-d polynomial $f \in R_p[\mathtt{X}]$ as long as $f(a) = \sum_{i=0}^{d} f_i a^i$ $\forall a \in R_p$, i.e. as long as it preserves evaluation correctness[9].

In this work, as it is common in the IOP literature, we think about polynomials as being given in a point-value representation, which matches the definition of a Reed Solomon codeword. There are a series of operations that both the prover and 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. In order to achieve this, we introduce the notion of *encrypted polynomials.*

Definition 19. *Let* $\mathtt{HE}$ *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* $\mathtt{HE\text{-}IOPP}$ *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* $\mathtt{ct}_1, \ldots, \mathtt{ct}_d \in R_q^2$ *be alleged ciphertexts such that* $\mathtt{HE.Dec}(\mathtt{ct}_i) = (m_{i,1}, \ldots, m_{i,\ell}) \in \prod_{j=1}^{\ell} \mathbb{F}_{p^D}$*. 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* $\mathtt{ct}_1, \ldots, \mathtt{ct}_d \in R_q^2$ *define an* encrypted polynomial (of f, at L) *if there exist admissible circuits such that,*

- *On input* $\mathtt{ct}_1, \ldots, \mathtt{ct}_d \in R_q^2$ *and any* $(\alpha_1, \ldots, \alpha_\ell) \in \prod_{j=1}^{\ell} \mathbb{F}_{p^D}$*, it returns a ciphertext* $\mathtt{ct}'$ *such that, with overwhelming probability,*

$$\mathtt{HE.Dec}(\mathtt{ct}') = (f(\alpha_1), \ldots, f(\alpha_\ell)) \in \prod_{j=1}^{\ell} \mathbb{F}_{p^D}.$$

- *On input* $\mathtt{ct}_1, \ldots, \mathtt{ct}_d \in R_q^2$ *to the* $\mathtt{HE\text{-}IOPP}$*, 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 $(\mathtt{ct}_1, \ldots, \mathtt{ct}_d) \in \mathsf{EncPoly}(f, L)$.

In other words, $(\mathtt{ct}_1, \ldots, \mathtt{ct}_d) \in \mathsf{EncPoly}(f, L)$ 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 f.

5.1 The HE-Batched-FRI protocol

The specific HE-IOPP we will employ is the HE transformation (see Definition 17) 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[10] β_i. In fact, replacing FRI

[9] It could happen, for a malicious choice of the $\mathtt{ct}_i \in R_q^2$, 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.

[10] 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].

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 17).

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 seemingly the most stable component of our overall compilers, we provide our `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 $f_1^{(0)}, \ldots, f_t^{(0)} : L_0 \to \mathbb{F}$ 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) 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. We will reduce the security of our protocol to that of batched FRI. Let $\mathcal{A}$ be an adversary who can break the round-by-round knowledge soundness of the `HE-Batched-FRI` (Figure 3). We define an adversary $\mathcal{A}'$ who breaks the security of Batched FRI as follows. $\mathcal{A}'$ runs $(\mathtt{sk}, \mathtt{pk}, \mathtt{evk}) \leftarrow \mathtt{HE.KeyGen}(1^\lambda, \mathcal{C}, \mathtt{pp})$ for the relevant `HE` scheme, and in particular recovers `sk`. Given any input, $\mathcal{A}'$ encrypts it under `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 oracles $[\![f]\!]_{\mathtt{HE}}$ he receives from $\mathcal{A}$ at every point, decrypts the answers using `sk` so as to recover f, and then sends the oracle $[\![f]\!]$ to $\mathcal{V}$.

As long as the parameters of the `HE` scheme provide evaluation correctness for all the operations that $\mathcal{A}$ has to perform, the above strategy implies that if $\mathcal{A}$ breaks the round-by-round knowledge soundness of the `HE-Batched-FRI` (Figure 3), then so does $\mathcal{A}'$ for batched FRI. The security of the latter, under the same RBR soundness error, was shown in [BGK+23, Theorem 4.2]. ∎

6 Optimisations

In order to make our construction practical, we introduce a number of optimisations. We note that these may be of independent interest, and may have applications outside of what is presented in this work.

The HE-Batched-FRI protocol

Setup: Agree on the following:

- A $\mathtt{HE}$ 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.

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 $f^{(r)}\Big|_{L_r} \stackrel{?}{\in} \mathtt{RS}[\mathbb{F}_{p^D}, L_r, d]$. If this or any of the previous checks fail, reject and abort. Otherwise, accept.

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

6.1 On the choice of the HE scheme

At its core, our construction only requires an HE scheme supporting prime fields as the plaintext space, which makes it compatible with nearly all modern HE schemes, such as TFHE [CGGI20], BGV [BGV12], and BFV [Bra12, FV12]. The notable exception here is CKKS [CKKS17]. The reason for this is twofold: first of all, CKKS doesn't have a plaintext space in the "traditional" way, as the other schemes mentioned above do. Secondly, the CKKS scheme is, by nature, approximate – this means that, even for a fresh encryption, we always have $\texttt{HE.Dec}(\texttt{HE.Enc}(\texttt{pk}, m), \texttt{sk}) \approx m$ and never $\texttt{HE.Dec}(\texttt{HE.Enc}(\texttt{pk}, m), \texttt{sk}) = m$. This makes it incompatible with our verification approach, since IOPs work with exact arithmetic. This leaves us with the choices of BGV/ BFV and TFHE, and this choice is mostly guided by practical performance and availability of implementations, as we further detail in Section 7.

6.2 Tensoring

Recall the structure of the HE plaintext space as shown in Section 2, in Equation 1. Ideally, we would want the plaintext ring R_p to split into ℓ copies of $\mathbb{F}_{p^D}$, where D would satisfy the FRI security requirements, i.e. such that $|\mathbb{F}_{p^D}| \approx 2^{256}$. For this to happen, we would need $\mathbb{F}_p$ to contain roots of unity of order at most $2N/D$, where N is the degree of the cyclotomic ring. Given the requirements of the HE scheme, i.e. that $\log(N) \in \{11, \ldots, 17\}$, this would restrict the value of p to be smaller than $2N/D$. On the other hand, FRI also requires $\mathbb{F}_{p^D}$ to have an 2^n-th root of unity, which therefore requires $pD > 2^n$. In practice, we would prefer to further restrict this and only use roots of unity in $\mathbb{F}_p$ (which requires $p > 2^n$), as it enables some procedures (*e.g.*, the NTT in Section 6.3) to run D times faster while only slightly reducing the maximum order of the roots of unity.

Therefore, once we factor in all the requirements (FRI, HE and implementation), we have that $2^n = d\rho^{-1} < p < 2N/D$, which restricts the size of the input polynomial to $d < 2N\rho/D$. This is doable in practice if we consider some of our most comprehensive parameter sets. For example, with $(\log(N), \rho, D) = (17, 1/2, 6)$, we could have d to be at most 2^{14}. More often, however, we would prefer better-performing parameters, such as $(\log(N), \rho, D) = (14, 1/16, 12)$, which restricts d to just around 2^6, which would almost entirely restrict the use of FRI.

To circumvent this, we use the following trick. For our HE-FRI protocol, we will make use of a field extension $\mathbb{F}_{p^D}$ of $\mathbb{F}_p$. Whenever we want to evaluate an encrypted evaluation map on an element of $\mathbb{F}_{p^D}$, we emulate the arithmetic of the field extension $\mathbb{F}_{p^D}$ of $\mathbb{F}_p$ as a homomorphic circuit. This results in D `HE` ciphertexts, which we interpret as (together) encrypting a single value in $\mathbb{F}_{p^D}$. Any further evaluation proceeds as an emulation of the arithmetic of $\mathbb{F}_{p^D}$ via an `HE` circuit. Alternatively, one can also choose a "halfway" approach: instead of using D ciphertexts encrypting values in $\mathbb{F}_p$to emulate the arithmetic of $\mathbb{F}_{p^D}$, one can pick an intermediate value d'. Then, the plaintext space of the HE scheme

can be $\mathbb{F}_{p^{d'}}$, and from there we emulate the arithmetic of $\mathbb{F}_{p^D}$, this time only with D/d' ciphertexts. The ideal value of d' will typically depend on the exact application; as mentioned above, increasing the plaintext modulus will incur an increase in all remaining `HE` parameters, and this needs to be carefully balanced.

6.3 Shallow Reed-Solomon encoding

Encoding a Reed-Solomon codeword consists of interpreting the input data as a polynomial of degree $(d-1)$ and evaluating it at $2^n = d\rho^{-1}$ independent points. Polynomial evaluation is a linear procedure 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. In its original format, however, it is typically 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 the most expensive procedure of FRI in general, but it is especially costly when running batched polynomial commitments, 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 $k = \sum_{i=0}^{d-1} k_i X^i \in \mathbb{F}_p[X]$. We encrypt k in an array of d/N ciphertexts $\mathtt{ct}$, such that $\mathtt{ct}_i$ encrypts $\sum_{j=0}^{d-1} k_{i \cdot N + j} X^j$. 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 previously mentioned permutations, as each coefficient of a polynomial would be in different ciphertexts. For large polynomials, however, the memory requirements to run the NTT with this packing might make it impractical.

Notice that, 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). Notice further that the goal of the NTT is to create a redundant representation of the polynomial (*i.e.*, the RS codeword), which will take more

memory to be stored. In this way, storing the codewords could represent a problem even if storing the original polynomials was not one. 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 FOLD optimisation

Recall the commit phase of our HE-FRI presented in Section 5.1. The complexity of the folding algorithm is $O(2^n)$ and the depth is n[11]. In order to decrease the depth, we will evaluate the `Fold` function in a DFT-like manner. Instead of evaluating the protocol as presented, we will express each $\texttt{Fold}(f^{(i)}, \alpha_i)$ as a function of the first $[\![f^{(0)}]\!]_{\texttt{HE}}$.

In particular, on the Prover side, we will replace the FFT-like algorithm with a DFT-like algorithm, thereby reducing the depth to 1. We compute the first layer of the `Fold` operation as is, and then for the following layers, we proceed as follows. We first pre-compute the constants, then express all the following layers as the composition $\texttt{Fold} \circ \ldots \circ \texttt{Fold}$. It follows that each layer is represented as several inner products of the original polynomial, as Algorithm 1 shows. This reduces the depth to 1, at the cost of increasing the complexity to $O(2^n \log(2^n))$, which does not change the overall complexity of FRI (dominated by the NTT). Nothing is changed on the Verifier's side.

6.5 Repacking and Recomposing

Repacking 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 may encrypt 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 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 avoid this, we introduce a repacking procedure that extracts the evaluation points from the RLWE samples of dimension N and repacks them in RLWE samples of much smaller dimensions. Since the verifier does not compute any

[11] Remember that n is the logarithm of the size of the codeword. The folding is a linear algorithm.

Algorithm 1 Shallow `Fold`

Input: $n' = 2^n$, r, $f^{(0)}, \omega$
Output codewords $f^{(i)}$, for $i \in [\![0, r-1]\!]$.

$c \leftarrow [0, \dots, 0]$
for $k \leftarrow 0$ **to** $r-1$ **do**
  ▷ First we pre-compute the constants
  for $i \leftarrow 0$ **to** $n'/2 - 1$ **do**
    for $j \leftarrow 0$ **to** $2^{k+1} - 1$ **do**
      $c_{i+jn'/2} \leftarrow c_{i+jn'/2} \cdot \frac{1}{2} \cdot \left(1 + \frac{(-1)^j \alpha_j}{\omega^i}\right)$
    end for
  end for
  ▷ Now we compute the `Fold`
  for $i \leftarrow 0$ **to** $n'/2 - 1$ **do**
    $f_i^{(k)} \leftarrow 0$
    for $j \leftarrow 0$ **to** $2^{k+1} - 1$ **do**
      $f^{(k)}(\omega^i) \leftarrow f^{(k)}(\omega^i) + c_{i+j\cdot(n'/2)} \cdot f(\omega^{i+j\cdot(n'/2)})^{(0)}$
    end for
  end for
  $n' \leftarrow n'/2$
  $\omega \leftarrow \omega^2$
end for

homomorphic operation over the ciphertext, we do not need to preserve any homomorphisms. 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. Algorithm 2 shows the repacking procedure, which requires the subprocedures listed in the following. 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.

- 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 2:

$$\text{Decompose}_{\overline{p},\overline{q}}(c, i) \mapsto \left\lfloor \left\lceil [c]_{(q/\overline{p}^i)} \cdot \frac{\overline{q}}{(q/\overline{p}^i)} \right\rceil \right\rfloor_{\overline{q}}. \tag{2}$$

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 $\mathfrak{b} = \log(\overline{p})$; and
Input: a packing key switching key `ksk`
Output Repacked ciphertext

▷ Extract the evaluation points to LWE samples
for $j \leftarrow 0$ **to** d **do**
  $\hat{x}_j \leftarrow \text{ExtractLWE}(x_j, i_x)$
  $\hat{y}_j \leftarrow \text{ExtractLWE}(y_j, i_y)$
end for
▷ Decompose each LWE sample in k new samples, each encrypting $\log(\overline{p})$ bits of the evaluation points
$k \leftarrow \lceil \log(p) / 2^{\mathfrak{b}} \rceil$
for $j \leftarrow 0$ **to** $D-1$ **do**
  for $i \leftarrow 0$ **to** $k-1$ **do**
    $\tilde{c}_{kj+i} \leftarrow \text{Decompose}_{\overline{p},\overline{q}}(\hat{x}_j, i)$
    $\tilde{c}_{k(j+D)+i} \leftarrow \text{Decompose}_{\overline{p},\overline{q}}(\hat{y}_j, i)$
  end for
end for
▷ Pack all LWE samples in a single RLWE
return PackingKeySwitching $([\tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{2Dk}], \texttt{ksk})$

– 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 1 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.

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 dimension at the end. For example, in our practical instantiation (Section 7), we first perform the folding using $\mathfrak{P}_5$; then, the repacking extracts and decomposes the samples still using $\mathfrak{P}_5$, and the PackingKeySwitching repacks them using $\mathfrak{P}_1$. Finally, before committing to the codeword, the prover

Table 1: 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		20480	6656
$\mathfrak{P}_4$	2	1024	52	24576	12288
$\mathfrak{P}_5$	1	2048		32768	24576

runs 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 reduces the size of the ciphertext in 33%.

Recomposition The repacking process is fundamental for minimizing the impact of the HE overhead on FRI, but 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 following paragraphs describe the main challenges it addresses.

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 4). 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 2, 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 4). At a high level, one can view these two extra bits as "redundant" bits, 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

Algorithm 3 Recomposition

Input Repacked RLWE ciphertext $\mathtt{ct} = (\mathtt{ct}_0, \mathtt{ct}_1)$
Input Secret key $\mathtt{sk}$
Input Decomposition base $\mathfrak{b} = \log(\overline{p})$, and message scaling factor Δ
Output An array of points $\hat{v} \in \mathbb{F}_p^{2D}$

$v \leftarrow \frac{\mathtt{ct}_1 - \mathtt{ct}_0 \cdot \mathtt{sk}}{2^{\mathfrak{b}+2}}$ ▷ Calculate the phase
▷ Recompose each point in $\mathbb{F}_p$
$k \leftarrow \lceil \log(p) / 2^{\mathfrak{b}} \rceil$
for $j \leftarrow 0$ **to** $2D - 1$ **do**
  $\hat{v}_j \leftarrow \lceil \frac{v_{kj+k-1}}{4} \rfloor$
  $x \leftarrow \left\lfloor \frac{\hat{v}_j}{2^{\mathfrak{b}-2}} \right\rfloor$
  for $i \leftarrow k-2$ **to** 0 **step** -1 **do**
    ▷ Calculate the noise propagated by the previous digit
    $c \leftarrow \textsc{SignExtend}\,(x - [v_{kj+i}]_4)$
    ▷ Remove the noise and scale
    $\hat{v}_j \leftarrow \hat{v}_j + \left\lfloor \frac{v_{kj+i}+c}{4} \right\rfloor \cdot 2^{\mathfrak{b}(k-1-i)}$
    ▷ Extract the two most significant bits to x
    $x \leftarrow \left\lfloor \frac{(v_{kj+i}+c)}{2^{\mathfrak{b}}} \right\rfloor$
  end for
  $\hat{v}_j \leftarrow \left\lfloor \frac{\hat{v}_j}{\Delta} \right\rceil$
end for
return $\hat{v}$

Procedure $\textsc{SignExtend}(x)$:
if $x = 3$ **then**
  return -1
else if $x = 1$ **then**
  return 1
else
  return 0
end if

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 illustrated in Figure 4), it will correct the message to its original value.

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

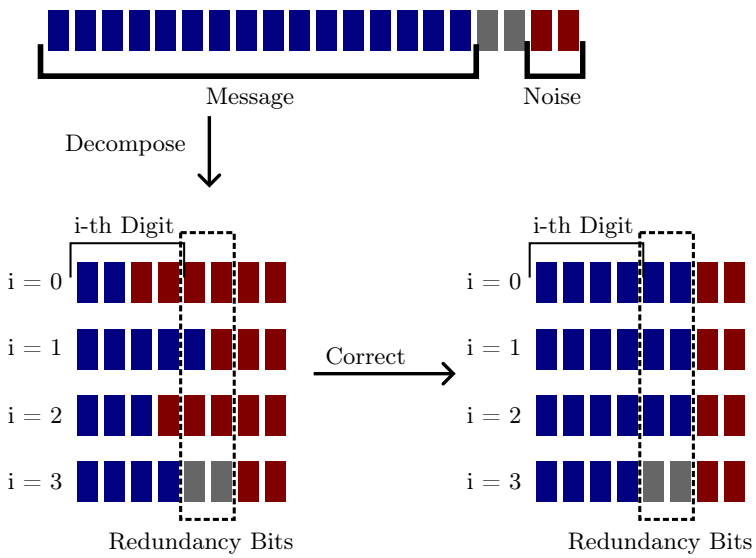


Fig. 4: Example of decomposition and error correction. Notice that the process for correcting using the redundant bit only occurs after the decryption.

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 18. In particular, we know that we can use bootstrapping as a rerandomization algorithm [DS16].

7 Experimental Results

Operating over the plaintext space enables our construction to work with nearly all modern HE schemes (CKKS being the notable exception). Considering this, we evaluate the practical impact of our proposals based on two performance metrics. First, we provide a full performance characterization of HE-FRI based on the number of basic arithmetic operations executed by each party with and without the optimizations and designing choices described in Section 6. This evaluation model allows us to present a broader view of HE-FRI performance behavior, clearly defining the impact of each of our proposals independently of the HE scheme. It also enables one to easily estimate practical performance for different schemes and parameter choices.

Then, based on conclusions from our operation counting, we instantiate the construction using implementations of the BGV and TFHE schemes, both in

leveled mode. This proof-of-concept implementation not only serves to validate our estimates but also to address practical implementation aspects.

7.1 Practical Parameters

For the security of FRI, we consider parameters established by previous literature [BGK+23] and reproduced in Table 2. There are also practical parameters that are required for functionality. For example, as we discussed in Section 6.2, $\mathbb{F}_p$ needs to provide a root of unity of order 2^n, which limits the maximum size of the input polynomial based on the value of n. Considering this, Table 3 presents the main choices of parameters according to the maximum size of the input polynomial that they support. We note that, for every parameter in Table 3, 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 2: Security parameters we adopt for FRI, based on the security estimates of [BGK+23] using Conjecture 1. We approximate the size of the field for practical reasons.

Parameter	$\lvert\mathbb{F}_{p^d}\rvert$	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

Table 3: 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

7.2 Number of Operations

We start building HE-FRI over the FRI implementation of Szepieniec *et al.* [S+21], which we extend with our proposals and instrument 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 the practical instantiation. Table 4 summarizes our results.

Table 4: Cost and depth comparison between FRI 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 1. 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 [BBHR18]	HE-FRI	FRI [BBHR18]	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

Verifier Performance Figure 5-a presents the number of elements in $\mathbb{F}_{p^D}$ received by the verifier during the execution of FRI. We take this number as representative of FRI's performance, as each element needs to be checked in the Merkle Tree, which is the most expensive operation for the verifier. In the case of HE-FRI, each element also needs to be decrypted, which introduces a significant overhead leading to the results presented in Figure 5-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

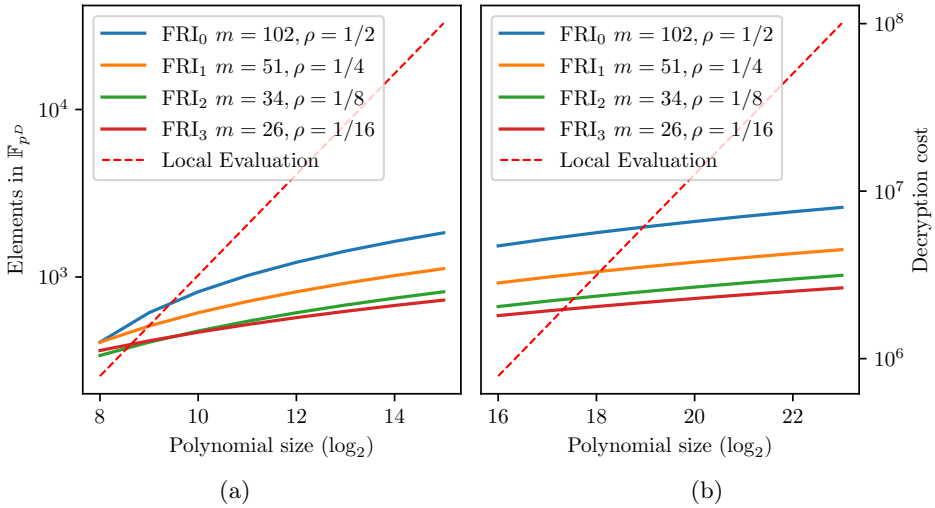


Fig. 5: Estimated cost for the Verifier in HE-FRI (a) in the number of elements in $\mathbb{F}_{p^D}$ received and (b) in the number of multiplications required to decrypt them.

themselves. For this scenario, HE-FRI provides gains for polynomials larger than 2^{17}, as Figure 5-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.

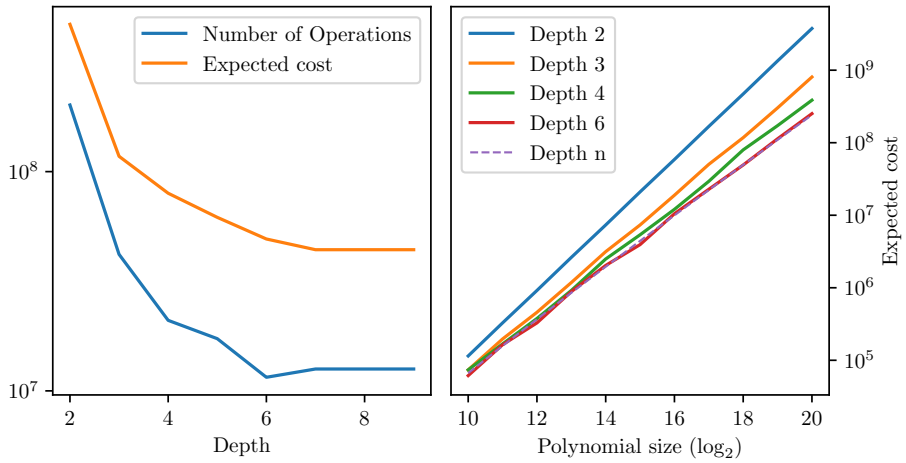
Prover Performance Once we establish the minimum size for which gains with HE-FRI are possible, our goal becomes minimizing the cost for the prover. Taking, for example, a polynomial of size 2^{17} and $\rho = 1/2$, the RS encoding and the folding algorithm would have depths 18 and 17, respectively in 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 a batched polynomial commitment, 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 a typically quadratic impact on the HE evaluation performance.

In Section 6, we showed how to solve this problem with a series of optimizations that enable FRI to be evaluated with a 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. 6: Expected cost of our NTT implementation.

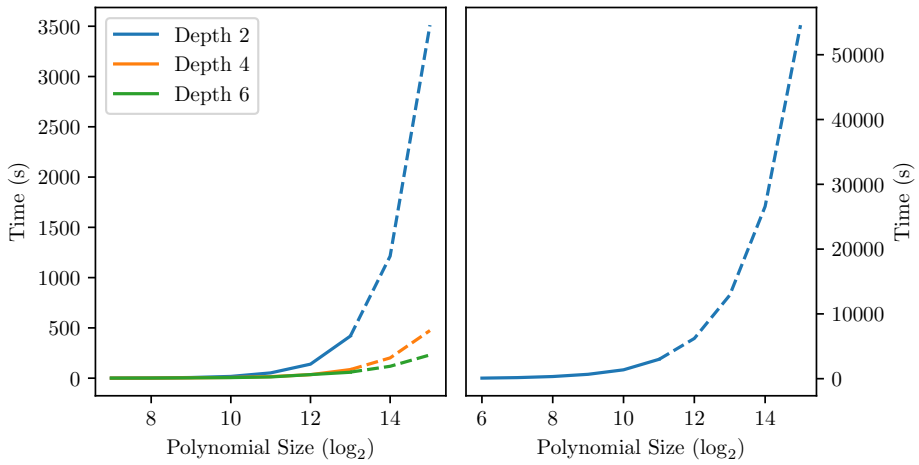
Figure 6-a shows the results for an NTT with an input of size 2^{18}. For each value of maximum depth, we choose arity to minimize the expected cost (even if it would increase the number of operations). While the cost decreases quickly with the depth at first, it stabilizes with depth as low as just 6. Based on this, we selected a few different values for the maximum depth and extended this analysis to different sizes of polynomials. Figure 6-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.

7.3 Proof-of-concept Implementation

When designing a proof-of-concept implementation, most of our project decisions (including, *e.g.*, the choice of HE libraries and schemes) were taken aiming at convenience and applicability rather than specific requirements from our construction. As in Section 7.2, we also start by building over the FRI implementation of [S+21], which is 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 FHE libraries to implement the encrypted arithmetic. More precisely, all cleartext operations are run using SageMath [The22] while operations over ciphertexts are delegated to the FHE libraries HELib [HS20] and MOSFHET [GBA22] using Python Ctypes. We also evaluate a cleartext version of our encrypted operations (also using SageMath and Python's standard libraries) to validate the correctness of our implementation at every step. We include the execution time of all these operations in the results presented in this section, as one would expect the homomorphic libraries to dominate the cost in all procedures.

For the purposes of this proof-of-concept, we consider only the smallest parameter set of Table 3, as we are limited by specific aspects of our current implementation. We note that in a dedicated (fully optimized) implementation, larger values of p would have only a minor impact on the prover's performance while almost not changing performance at all for the verifier. We further discuss the limitations of our current implementation in Section 7.4. We run all the experiments in a `c6i.metal` instance (Intel Xeon 8375C at 3.5 GHz with 256 GiB of RAM) on AWS. We measure time using the standard `time` library from Python, and we report both measured results (solid curves) as well as curve fitting estimates (dashed curves) based on the asymptotic complexity of each specific operation.

Reed-Solomon code encoding Based on the results of our estimates (Section 7.2), we consider NTTs of depth up to 6 for encoding RS codewords. In conjunction with the size of our plaintext space (which varies from 16 up to 50 bits), this requires us to use a large ciphertext modulus and, hence, a multi-precision implementation of the RLWE scheme. Considering this, we adopt the RNS-based implementations of the BGV scheme for evaluating this part. Figure 7-a shows the performance results for our proof-of-concept implementation.



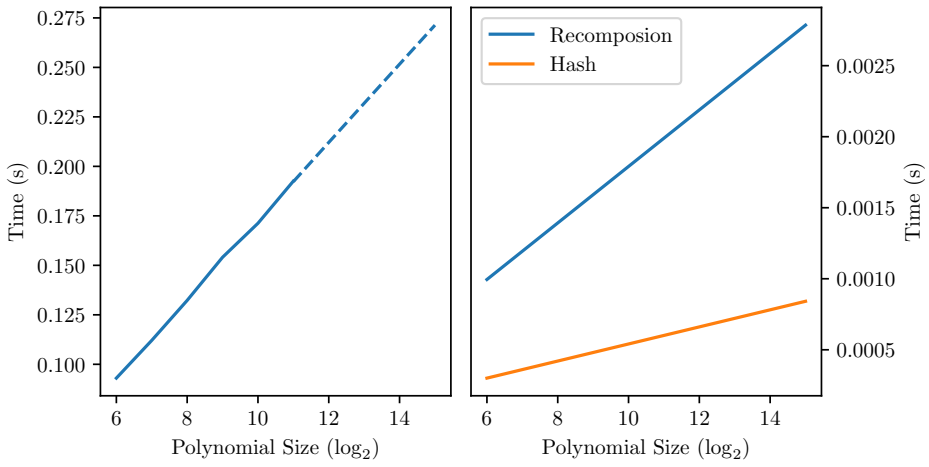
(a) NTT (b) Folding

Fig. 7: Performance of HE-FRI for the prover using the parameter set FRI_3 for $D = 16$.

Folding Contrary to the NTT, the folding procedure has a fixed multiplicative depth equal to 1 (thanks to the optimizations we introduce in Section 6.4) and is the last procedure to be run before decryption. As such, it requires a much smaller ciphertext modulus and, for some of our parameter sets, can be evaluated using single-precision (non-RNS) implementations. In this way, whenever it is possible, we evaluate it using implementation techniques from TFHE. Figure 7-b presents its performance.

Verification Figure 8-a shows the execution time for the verification procedure. In this procedure, while the performance achieved by our proof-of-concept implementation is already good, we note it should be significantly better in a fully optimized implementation. As we mentioned at the beginning of Section 7.3, we would at first expect the cost of homomorphic operations to prevail in all procedures, but this was not the case in the verification. By micro-benchmarking the operations evaluated by the FHE library, we noticed the execution time of the full verification is two orders of magnitude higher than all the time spent by the FHE library. Specifically, the decryption and recomposition operations (Algorithm 3) take just 200 microseconds per round of FRI while the hash operations take only 60 microseconds, both for m = 26 and including the cost of calling a shared library from the Python code. Considering that the cost of these operations per round of FRI is independent of the size of the input polynomial, we estimate the full cost of these operations in Figure 8-b.

The rest of the cost of our current implementation of the verification process arises from the SageMath-implemented arithmetic for the linearity checks, which could be significantly accelerated in a fully optimized implementation. Partic-



(a) Full verification (b) HE operations

Fig. 8: Performance of HE-FRI for the verifier using the parameter set FRI_3 for $D = 16$.

ularly, in terms of asymptotic complexity, the linearity checks should be much faster than the decryption process, which enables us to conclude that a fully optimized implementation could achieve gains of up to 100 times over the current one. It is also important to note that the performance of the recomposition procedure (Algorithm 3) only depends on the size of the field $\mathbb{F}_{p^D}$, but not on the values of p or n, which makes the results of this section also valid for other parameter sets.

7.4 Further improvements

Two main aspects limit the results our proof-of-concept implementation is capable of providing. First, while we use optimized libraries for the FHE part of the protocol, our plaintext arithmetic and the integration between parts are still entirely implemented in Python, which not only adds significant overhead but also restricts our ability to control aspects such as memory usage, which become very important for larger parameters. It also introduces some challenges for efficiently parallelizing the procedures, which could greatly improve the overall performance, as most of our algorithms are trivially parallelizable.

The second aspect is the integration between cryptosystems themselves. As it stands, our implementation does not rely on specific properties of any HE scheme (other than providing a prime plaintext space). Our only motivation to work with implementations of different schemes (a BGV implementation for the NTT and a TFHE one for the folding) is the convenience of using already optimized implementations that fit well in the procedures we want to evaluate. Integrating these two schemes, however, would require complex scheme-switching

techniques, which are outside of our scope in this work. Instead, the ideal solution for integrating these two parts is simply to extend one of the implementations with the techniques from the other. Notice that all procedures we use in this work are fully compatible with both schemes, and this extension is only a matter of implementation (we do not need, for example, ciphertext-ciphertext multiplications, which could require changes to the encoding adopted by each scheme). For the final version of this paper, we plan to implement this extension in a dedicated fully optimized implementation of HE-FRI, which we expect to achieve much better performance compared to our current results.

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

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

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.

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.

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.

GBA22. Antonio Guimarães, Edson Borin, and Diego F. Aranha. MOSFHET: Optimized software for FHE over the torus. Cryptology ePrint Archive, Report 2022/515, 2022. https://eprint.iacr.org/2022/515.

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.

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. *Designs, codes and cryptography*, 71:57–81, 2014.

The22. The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.5)*, 2022. https://www.sagemath.org.

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:

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

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 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 $\mathcal{R}_t = \mathbb{Z}_t[x]/(x^n+1)$, where t is the plaintext modulus. Messages and ciphertexts will be considered as polynomials in $\mathcal{R}_t$ 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 $\mathbf{D} = \{D_1^\star, \ldots, D_L^\star\}$, with $D_j^\star = \prod_{h=1}^{j-1} D_h$, where the D_h are such that $Q_i = \prod_{h=1}^{\ell}$, for a decomposition parameter ℓ
– 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 $(1, s) \coloneqq \mathtt{sk}$ as the secret key. Sample $a \leftarrow \mathcal{R}_{Q_L}$ and $e \leftarrow \chi$. Set $\mathtt{pk} = (\mathtt{pk}[0], \mathtt{pk}[1]) \coloneqq ([-as - te]_{Q_L}, a)$ as the public key. Let $\mathtt{sk}' = (s_0', \ldots, s_k')$ be another secret key. The evaluation key for switching a ciphertext with respect to $\mathtt{sk}'$ to a ciphertext with respect to $\mathtt{sk} = (1, s)$ is defined as follows. For $j \in \{0, \ldots, k\}$ and $i \in \{0, \ldots, \ell\}$ sample $a_{ij} \leftarrow \mathcal{R}_{Q_L}$ and $e_{ij} \leftarrow \chi$ and set $\mathtt{evk}_j \coloneqq ([-a_{ij}s - te_{ij} + D_i^\star s_i']_{Q_L}, a_{ij})$. Return $(\mathtt{sk}, \mathtt{pk}, \mathtt{evk})$.

`HE.Enc`$(\mathtt{pk}, m)$: Let $m \in \mathcal{R}_t$ 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 $\mathtt{ct} = (\mathtt{ct}[0], \mathtt{ct}[1]) \coloneqq ([m + \mathtt{pk}[0]u + te_1]_{Q_i}, [\mathtt{pk}[1]u + te_2]_{Q_i})$.

`HE.Dec`$(\mathtt{sk}, \mathtt{ct})$: Return $m' = [[< \mathtt{ct}, \mathtt{sk} >]_{Q_i}]_t$.

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

`HE.KeySwitch`$(\mathtt{ct}, \mathtt{sk}', \mathtt{evk})$: Let $\mathtt{ct}'$ be a ciphertext with respect to $\mathtt{sk}' = (s_0, \ldots, s_k)$. Recall $\mathtt{sk} = (1, s)$. Define $T \subseteq \{1, \ldots, k\}$ to be the set such that for $i \in T$ $s_i' = 1$ or $s_i' = s$. Define $\mathtt{ct}_j'[i]$ via

$$\mathtt{ct}'[i] = \sum_{j=1}^{\ell} \mathtt{ct}_j'[i] D_j^\star.$$

For $i \in \{0, \ldots, k\} \setminus T$ compute

$$(\mathtt{ct}^{(i)}[0], \mathtt{ct}^{(i)}[1]) = \left[\sum_{j=1}^{\ell} (\mathtt{ct}_j'[i]\mathtt{evk}_j[i][0], \mathtt{ct}_j'[i]\mathtt{evk}_j[i][1])\right]_{kQ_i}.$$

For $i \in T$ compute

$$(\mathtt{ct}^{(i)}[0], \mathtt{ct}^{(i)}[1]) = (k\mathtt{ct}'[i], 0) \qquad \text{if } s'_i = 1$$
$$(\mathtt{ct}^{(i)}[0], \mathtt{ct}^{(i)}[1]) = (0, k\mathtt{ct}'[i]) \qquad \text{if } s'_i = s.$$

Return $(\mathtt{ct}[0], \mathtt{ct}[1]) = \left[\left(\sum_{i=0}^{k} \mathtt{ct}^{(i)}[0], \mathtt{ct}^{(i)}[1]\right)\right]_{kQ_i}$.

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 $\mathtt{ct} \coloneqq ([\mathtt{ct}_0[0] + \mathtt{ct}_1[0]]_{Q_i}, [\mathtt{ct}_0[1] + \mathtt{ct}_1[1]]_{Q_i})$.

HE.Mult$(\mathtt{ct}_0, \mathtt{ct}_1, \mathtt{evk})$: Bring $\mathtt{ct}_0, \mathtt{ct}_1$ to the same level by modulus switching the ciphertext at the higher level to the lower level.
Calculate $\mathtt{ct}_{\text{pre-mult}} = (\mathtt{ct}_{\text{pre-mult}}[0], \mathtt{ct}_{\text{pre-mult}}[1], \mathtt{ct}_{\text{pre-mult}}[2]) \coloneqq ([\mathtt{ct}_0[0]\mathtt{ct}_1[0]]_{Q_i}, [\mathtt{ct}_0[0]\mathtt{ct}_1[1]+\mathtt{ct}_0[1]\mathtt{ct}_1[0]]_{Q_i}, [\mathtt{ct}_0[1]\mathtt{ct}_1[1]]_{Q_i})$. Define $\mathtt{ct}_j^{\text{pre-mult}}[2]$ such that

$$\mathtt{ct}_{\text{pre-mult}}[2] = \sum_{j=1}^{\ell} \mathtt{ct}_j^{\text{pre-mult}}[2] D_j^{\star}.$$

Compute

$$\mathtt{ct} \coloneqq k(\mathtt{ct}_{\text{pre-mult}}[0], \mathtt{ct}_{\text{pre-mult}}[1]) + \sum_{j=1}^{\ell} (\mathtt{ct}_j^{\text{pre-mult}}[2]\mathtt{evk}_2[j][0], \mathtt{ct}_j^{\text{pre-mult}}[2]\mathtt{evk}_2[j][1]).$$

Output

$$\mathtt{ct}_{\text{mult}} = \mathtt{HE.ModSwitch}((\mathtt{ct}, Q_{sp}), Q_{i-1}).$$