Constant-Round Secure Distributed Decoding and HQC Threshold Decryption

Pascal Giorgi[1], Fabien Laguillaumie[1], Lucas Ottow[1] and Damien Vergnaud[2]

[1] LIRMM, Univ. Montpellier, CNRS, Montpellier, France
[2] Sorbonne Université, CNRS, LIP6, Paris, France

Abstract. Threshold public-key encryption schemes enable decryption only with the participation of enough partial secret key holders. In this article, we propose the first dedicated protocol for distributed decryption of HQC ciphertexts. This protocol is perfectly correct and does not leak any information about the shared secret key. This leads to the first threshold cryptosystem based on HQC. To this end, we present protocols for securely decoding shared erroneous words of both Reed-Muller and Reed-Solomon codes. Such decodings require to develop innovative techniques for specific multiparty computations in fields of characteristic 2. For distributed Reed-Muller decoding, we investigate two approaches : one relying on a majority computing protocol and a second one relying on an exaustive search for closest codeword. For distributed Reed-Solomon decoding, we propose a novel protocol for securely solving Padé approximants over shared polynomials. Beyond their immediate application to HQC, our results enable new techniques in secure distributed computation over structured algebraic objects, and may find independent applications in advanced cryptographic protocols.

Keywords: HQC · Threshold cryptography · Multi-party computation · Reed-Solomon decoding · Reed-Muller decoding

1 Introduction

Threshold public key encryption (t-PKE) [DF90, Des93] is a variant of public key encryption (PKE) in which a number of `N` participants receive a partial secret key. It is required from such a primitive that at least a threshold number `T` of these participants collaborate in order to decrypt a given ciphertext. The need for threshold primitives is quite substantial, as highlighted by NIST's first official call for threshold schemes[1] that was published in early 2026.

A number of t-PKE instantiations already exist in the literature. Many of these rely on classical problems in pre-quantum cryptography such as discrete logarithm-based problems [FP01, LP01, SG02, GJKR07], RSA-based problems [DDFY94, GRJK00], composite residuosity [FPS01, DJ01, HMR+19], or class group-based problems [CLT22, BCD+24]. As such, these constructions would be vulnerable to quantum computers. Lattice-based cryptography has been used to build post-quantum schemes. In 2010, Bendlin and Damgård [BD10] designed the first threshold lattice-based encryption scheme. Since then, threshold lattice-based schemes with additional properties have been proposed, such as adaptive (or CCA) security [DLN+21], or fully homomorphic encryption [BS23]. In order

E-mail: pascal.giorgi@lirmm.fr (Pascal Giorgi), fabien.laguillaumie@lirmm.fr (Fabien Laguillaumie), lucas.ottow@lirmm.fr (Lucas Ottow), damien.vergnaud@lip6.fr (Damien Vergnaud)

[1] https://csrc.nist.gov/pubs/ir/8214/c/final

This work is licensed under a "CC BY 4.0" license.
Date of this document: 2026-10-19.

to diversify the range of hardness assumptions, we would like to study the feasibility of threshold schemes on code-based assumptions.

Code-based threshold cryptography. In the realm of cryptography based on error-correcting codes, threshold schemes are less numerous. Takahashi, Hashimoto and Ogata [THO24] proposed the first three threshold PKEs based on code-related assumptions. In order to design their schemes, the authors use the Niederreiter cryptosystem [Nie86], which is used in the Classic McEliece NIST submission [ABC+22], as a baseline. Then, they use different conversion techniques for each of their proposition in order to build threshold schemes achieving CCA security. Their first two propositions use general conversions that do not rely on the specific properties of the Niederreiter cryptosystem, and the sizes of the elements of their schemes depend on the number `N` of participants of the scheme. Their last contribution follows a different design; the authors build a dedicated multiparty computation (MPC) protocol for decoding Goppa codes, which is the main step in the decryption procedure of the Niederreiter cryptosystem. Naively, building an MPC protocol for decrypting a PKE ciphertext allows one to build a threshold scheme by first sharing a secret key of the base cryptosystem. Similar ideas were also used to build threshold multivariate-based digital signatures [CS19, CEN25]. Unfortunately, the MPC protocol of [THO24] would fail when run on incorrectly formed ciphertexts. This prevents their third scheme from achieving CCA security, even when using an additional conversion aiming at achieving this level of security.

In [GLOV26], the authors fix this issue by designing a conversion based on the Naor-Yung transform [NY90, Lin03], and by instantiating it using a custom-made Zero-Knowledge proof system for Niederreiter ciphertexts. This construction prevents the MPC protocol from being run on incorrect ciphertexts by requiring each party to publicly verify that a submitted ciphertext is correctly constructed, and this leads to a CCA construction. The authors of [GLOV26] also provide an additional improvement that allows reducing the partial secret key size compared to the third but erroneous scheme of [THO24].

All of these schemes rely on the Niederreiter cryptosystem [Nie86], which suffers from large public key sizes compared to other PKEs. It would therefore be preferable to also dispose of threshold PKE schemes that rely on other code-based constructions. In 2025, NIST selected the HQC cryptosystem [AAB+22] as a standard alternative to the lattice-based primitives that were previously selected to be standardized. As in the Niederreiter cryptosystem, the core of the HQC cryptosystem, called HQC.PKE, requires decoding an erroneous codeword as the main part of the decryption procedure. Moreover, these codes have symbols either over $\mathbb{F}_2$ or one of its extension fields. For the HQC.PKE cryptosystem, this code is a concatenated code with some variants of first order Reed-Muller codes and Reed-Solomon codes. However, main differences arise in the decoding procedures of Niederreiter and HQC.PKE. Decoding during Niederreiter decryption is done on a private Goppa code (defined by the secret key) and for which the number of errors is known in advance. Decoding for HQC.PKE is done on a public code (part of the parameters of the scheme), and only an upper bound on the number of errors is known. This calls for distinct techniques when trying to come up with an MPC protocol for decoding these codes, especially for the computation of the error polynomials in the case of either Goppa codes or Reed-Solomon codes. In the plain setting, decoding Reed-Solomon codewords usually relies on the Berlekamp-Massey algorithm or the extended Euclidean algorithm, which are challenging to translate into constant-round MPC protocols.

Constant round MPC protocols for computer algebra problems. As decoding error-correcting codes efficiently in the plain setting relies heavily on computer algebra techniques, it is necessary to understand to what extent these techniques are applicable in the MPC setting.

Multiparty computation as a field dates back to the seminal works of Yao [Yao82] and Goldreich, Micali, and Widgerson [BGW88]. Since then, a number of constant-round protocols for dedicated purposes have been published. The seminal work of Bar-Ilan and Beaver [BIB89] presents techniques for general constant-round computation of general arithmetic circuits. Since then, a multitude of works have been published in order to improve this generic construction through more dedicated constant-round protocols.

In the realm of linear algebra, the work of Cramer and Damgård [CD01] was the first to propose dedicated MPC protocols, but these protocols had some error probability that made them not perfectly correct, i.e. the protocols could sometimes leak information about the secret matrix. In the context of threshold decryption, such a leakage could help an adversary learn information about the secret key. Improved protocols in terms of efficiency were proposed in [CKP07] and [MW08], but still encompassing error probabilities. The work of Maire and Vergnaud [MV24] introduced the first protocol for computing the determinant of a shared matrix in a perfectly correct manner. However, their techniques require the coefficients of the shared matrix to be in a field of large characteristic, which does not correspond to the setting of decoding codewords over extension fields of $\mathbb{F}_2$.

A very recent ePrint paper [HLPS26] actually provides a constant-round protocol for computing determinant over a small characteristic field, but with a quite significant additional communication cost. In general, multi-party computations over fields of characteristic 2 are more challenging than over large prime fields.

Mohassel and Franklin [MF06] were the first to present dedicated protocols for unbounded shared polynomial multiplication, shared polynomial evaluation and interpolation over shared points. These protocols were later improved in [GLOV24] to achieve a quasi-linear communication complexity. While an MPC protocol for polynomial gcd has already been presented in [MF06], it does leak some information on the input polynomial, which is unacceptable in our setting. More advanced techniques with perfect correctness and privacy are thus required to tackle those challenging polynomial calculations arising within the decoding of some error-correcting codes. In particular, for Reed-Solomon decoding, one needs to provide a dedicated MPC protocol for solving Padé approximant problems, i.e., find a rational fraction $\frac{r}{t} = f \bmod X^n$ given a polynomial f and some degree constraints on the polynomials r and t.

Our contribution. In this paper, we design the first multi-party protocol **HQCdecryption** for HQC.PKE decryption. This protocol is built around novel protocols : **ReedMuller-R** and **ReedMuller-ML** for decoding duplicated Reed-Muller codewords as well as **ReedSolomon** for Reed-Solomon codes, which are used in the design of HQC. Our decryption protocol leads to the first threshold PKE based on HQC. This threshold PKE achieves CPA security and relies on the exact same assumptions as the non-threshold HQC cryptosystem in the UC framework. This MPC protocol is perfectly correct and does not reveal any information about the decrypted message or the shared secret key. This security property will be really crucial for designing a future CCA variant of this cryptosystem. Our protocols are constant-rounds, meaning the number of rounds of interactions between the participant is not dependent on the size of the shared inputs. As such, some of our protocols are parametrized by a constant τ, unrelated to the shared inputs, that dictates the number of rounds.

Building protocols for decoding Reed-Solomon and Reed-Muller codes is challenging and requires new MPC techniques over fields of characteristic 2. To decode Reed-Muller codewords, we propose two approaches. The first (**ReedMuller-R**), based on Reed's algorithm [Ree54], allows unique decoding as long as the number of errors is less than half the code's minimum distance. It requires the computation of the most common value among a list of shared boolean values. To this end, we design a new majority computation protocol **Majority** using techniques on shared polynomials from [GLOV24]. We believe

this protocol might also be of independent interest in other contexts.

The second approach (**ReedMuller-ML**) fits the decoder of the HQC specification, as it outputs a closest codeword even if the number of errors exceeds the previous bound. It consists of exhaustively computing distances between candidate codewords as integers encoded as an exponent of a public value. Comparisons and detection of a closest codeword then rely on elementary MPC protocols.

Even if the first approach does not apply to the HQC cryptosystem, it is of independent interest, since it is the first secure distributed variant of Reed algorithm, and its complexity is actually asymptotically better than the second one.

The main difficulty in the secure distributed decoding of Reed-Solomon codes is to compute the error locator polynomial without any information leakage. To this end, we design the first multi-party protocol **Padé** for solving a Padé approximant problem corresponding to a secret polynomial whose coefficients are shared. We believe that this protocol might be of independent interest and used in other settings. This protocol relies on properties of subresultant matrices, which were already used in [MF06] for computing the GCD of two secret polynomials. However, our solution is perfectly secure, in the sense that participants learn absolutely no information about the secret polynomial, be it the degree or values of the polynomials corresponding to the solution, or even the existence of a solution. Our protocol's main step boils down to computing the determinant of many subresultant matrices. As such, we present a new protocol **CharPoly** for computing characteristic polynomials of matrices (which, in particular, allows for determinant computation) over fields whose sizes are powers of two. The idea is inspired by the short discussion in [MV24] where the authors suggest to adapt their protocol for the high-characteristic case to the small characteristic one using techniques from Schönhage's work in [Sch93]. We build on this idea and pursue further its implications by designing all the necessary tools to achieve a complete specification of this protocol. In particular, because Schönhage's techniques heavily rely on computing over rings of the form $\mathcal{R} = \mathbb{F}_{2^m}[Y]/Y^{d+1}$, we provide in Section 3.1 the first MPC protocols dedicated to such operations. Although the techniques employed are not novel, we believe that a comprehensive description of these new protocols, along with complexity and security analysis, is of independent interest.

Our protocols rely exclusively on techniques that are independent of the underlying MPC instantiation. As a result, they are agnostic to the specific MPC implementation and can be readily adapted to a wide range of use cases.

Organization of the paper We first lay some background notions in Section 2. In Section 3, we detail the protocol for computing the characteristic polynomial of a shared matrix in a field of characteristic 2, as well as the required protocols for handling computations for specific polynomial matrices and bivariate polynomials. In Section 4, we detail our techniques for designing an MPC protocol for securely solving Padé approximants problems. The protocol for Majority computation as well as the two Reed-Muller decoding protocols are then detailed in Section 5. Our techniques for distributed Reed-Solomon decoding are discussed in Section 6. Finally, we present our full protocol for HQC decryption in Section 7, along with discussions regarding the sizes and efficiency of our new threshold cryptosystem, and some concluding remarks for future work.

2 Preliminaries

2.1 Notations

We denote log as the base-2 logarithm. For a ring $\mathbb{R}$ and $n \in \mathbb{N}$, we denote $\mathbb{R}[X]$ (resp. $\mathbb{R}[X]_{\leq n}$) the ring of polynomials with indeterminate X with coefficients in $\mathbb{R}$ (resp. the polynomials of $\mathbb{R}[X]$ of degree up to n). We let $\mathbb{R}(X)$ (resp. $\mathbb{R}[[X]]$) the set of rational

functions (resp. formal series) with coefficients in $\mathbb{R}$. For q a power of two, we denote by $\mathbb{F}_q$ the finite field with q elements. All computations in the MPC protocols in this article deal with computations on elements of $\mathbb{F}_q$ as base operations. Given a finite set S, we denote by $x \xleftarrow{\$} S$ the fact that x is uniformly distributed in S. We note by $\lambda \in \mathbb{N}$ the security parameter and we say that a positive function parametrized by λ is negligible if it is asymptotical smaller than any positive rational function in λ when λ goes to infinity.

2.2 Security and base protocol functionalities

UC Security and hybrid models. Our protocols are proven secure using the Universally Composable (UC) framework introduced in [Can01]. It allows for proving the security of our protocols, even when composed in other protocols and executed in parallel. This framework is defined using Probabilistic Polynomial Time (PPT) Turing machines that interact with other machines. Informally, this framework compares the behavior of an environment $\mathcal{Z}$ interacting with an adversary either in a "real-world" experiment or an "ideal-world" experiment. In the "real-world" experiment, the protocol Π for $\mathtt{N}$ parties is executed. The environment $\mathcal{Z}$ can choose the inputs of the parties and an adversary $\mathcal{A}$ can corrupt any subset of parties. The protocol is executed and $\mathcal{A}$ and $\mathcal{Z}$ can communicate during the execution. At the end of the protocol, $\mathcal{Z}$ receives the output of all parties and the adversary and outputs a decision bit. In the "ideal-world" experiment, the protocol Π is replaced by an ideal functionality $\mathcal{F}$. As before, $\mathcal{Z}$ can choose the input of the parties and interact with an adversary $\mathcal{B}$. Then, it receives the outputs and returns a decision bit. We say that Π UC-realizes $\mathcal{F}$ if for any adversary $\mathcal{A}$ in the "real-world", there exists a corresponding adversary $\mathcal{B}$ in the "ideal-world" such that any environment $\mathcal{Z}$ has a negligible advantage at differentiating between interacting with $\mathcal{A}$ in the "real-world" and with $\mathcal{B}$ in the "ideal-world".

In order to prove UC security for our protocol, we usually build a simulator $\mathcal{S}$ that simulates the views of the parties in an execution of Π by interacting with the ideal functionality $\mathcal{F}$. This way, an adversary $\mathcal{B}$ in the "ideal-world" can be built for any "real-world" adversary $\mathcal{A}$ using the simulator, as the behavior of $\mathcal{A}$ interacting with $\mathcal{S}$ and in the protocol Π is the same.

Hybrid models introduced in [CLOS02] are useful in order to compose our protocols securely in parallel. In the $\mathcal{F}$-hybrid model, parties executing a protocol Π also have an unrestricted access to an ideal functionality $\mathcal{F}$. If a protocol P UC-realizes the functionality $\mathcal{F}$, let us denote Π^P the protocol Π where every call to $\mathcal{F}$ is replaced by an execution of P. If Π realizes a functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, then Π^P UC-realizes $\mathcal{G}$.

The Arithmetic Black Box model. The MPC protocols presented here would typically be instantiated using secret-sharing techniques. These methods allow a set of parties to compute on secret data, composed of elements of a finite field $\mathbb{F}_q$, such that any set of less than t participants cannot learn anything about said data, but a set of at least $t+1$ participants can deduce the value of said data. While computing linear functions of the shared data only requires local computation from the participants, computing a sharing corresponding to the product of two previously shared secrets always requires some form of communication. The first result to achieve such functionality is the BGW protocol [BGW88] and nowadays many different secret-sharing techniques exist for different security guarantees. In order to abstract from any choice of the underlying technique, we present our protocols in the *Arithmetic Black-Box Model* (ABB model). The ABB model is standard in the literature and has been used in prior works to describe MPC protocol with perfect security [HIKC23, CEN25]. In this model, parties have the ability to share elements, reveal their values, and perform additions and multiplications. These are the usual operations expected from secret-sharing techniques for MPC.

The functionality stores values with IDs of the form $(x, \mathtt{id})$, with $x \in \mathbb{F}_q$. It aborts on inconsistent inputs or on illegal inputs (e.g. trying to read a non-existing ID or trying to replace an existing one).

- Sharing: Upon receiving $(\mathtt{in}, \mathtt{id}, P_i, x)$ from party i and $(\mathtt{in}, \mathtt{id}, P_i)$ from everyone else, the functionality stores $(x, \mathtt{id})$.
- Opening: Upon receiving $(\mathtt{out}, \mathtt{id}, P_i)$ from every party, the functionality looks at the corresponding entry $(x, \mathtt{id})$ and sends it to party i.
- Linear functions: Upon receiving $(\mathtt{lin}, \mathtt{id}, \mathtt{id}', a, b)$ from every party, the functionality searches for the entries $(x, \mathtt{id})$ and stores $(a \cdot x + b, \mathtt{id}')$.
- Addition: Upon receiving $(\mathtt{add}, \mathtt{id}_1, \mathtt{id}_2, \mathtt{id}')$ from every party, the functionality searches for the entries $(x, \mathtt{id}_1)$ and $(y, \mathtt{id}_2)$ and stores $(x + y, \mathtt{id}')$.
- Multiplication: Upon receiving $(\mathtt{mul}, \mathtt{id}_1, \mathtt{id}_2, \mathtt{id}')$ from every party, the functionality searches for the entries $(x, \mathtt{id}_1)$ and $(y, \mathtt{id}_2)$ and stores $(x \cdot y, \mathtt{id}')$.
- Random element: Upon receiving $(\mathtt{rand}, \mathtt{id})$ from every party, the functionality samples $x \leftarrow^{\$} \mathbb{F}_q$ and stores $(x, \mathtt{id})$.
- Inner product: Upon receiving $(\mathtt{inprod}, \mathtt{id}_{1,1}, \ldots, \mathtt{id}_{1,n}, \mathtt{id}_{2,1}, \ldots, \mathtt{id}_{2,n}, \mathtt{id}')$ from every party, the functionality searches for the n entries $(x_1, \mathtt{id}_{1,1}), \ldots, (x_n, \mathtt{id}_{1,n})$, the n entries $(y_1, \mathtt{id}_{2,1}), \ldots, (y_n, \mathtt{id}_{2,n})$, and stores $(\sum_{i=1}^{n} x_i y_i, \mathtt{id}')$.

Figure 1: The $\mathcal{F}_{ABB}$ model

We formally describe a modified version of the ABB model $\mathcal{F}_{ABB}$ in Fig. 1. Any of the protocols described in this article is realizing its corresponding functionality in the $\mathcal{F}_{ABB}$-Hybrid model. Our description presents the usual functionalities of the ABB model, as well as the functionality to sample random secret-shared values unknown to every party, and the ability to compute inner products of shared vectors. Sampling random shares is a standard functionality and can be achieved in most cases (for example, by asking every participant to sample a random element as a share). The inner product functionality can already be achieved by combining n calls to the multiplication functionality and $n-1$ calls to the addition functionality, but we distinguish it for communication complexity reasons. Indeed, some secret-sharing techniques allow one to compute the inner product of shared vectors with the same communication cost as one multiplication between shared elements. This is the case for all *multiplicative* linear secret-sharing. In essence, multiplicative linear secret sharing allows to compute shares of a product $[xy]$ from a linear combination of the product of each players' shares of $[x]$ and $[y]$. Shamir secret-sharing [Sha79], which is used in the BGW protocol, is multiplicative. Moreover, any secret-sharing scheme that allows for the base functionalities of the ABB model can be converted into a multiplicative secret sharing scheme [CDM00].

For protocol and functionalities presentation, we will always denote shared elements with brackets, i.e. $[x]$ for $x \in \mathbb{F}_q$, and elements known to some or all the parties without them. In order to simplify the presentation of the protocols, the IDs are omitted from the description of the protocol (i. e. we will say that $[x] \in \mathbb{F}_q$ is shared without dealing with $\mathtt{id}_x$). Moreover, we might use simplified notations for the different operations; $[x] \leftarrow^{\$} \mathbb{F}_q$ denotes the generation of a random shared value, $[z] \leftarrow [x] + [y]$, $[z] \leftarrow [x] \cdot [y]$, and $[z] \leftarrow a \cdot [x] + b$ denote addition, multiplication and linear mapping respectively. We also use the same notations for matrices, polynomials and polynomials matrices. These objects are shared in the obvious way: matrices in $\mathbb{F}_q^{m \times m}$ are shared as $m \times m$ fields elements, and polynomials of degree at most n are shared as $n+1$ elements. Polynomial matrices of size

$m \times m$ whose coefficients have degrees up to n are shared as $m \times m$ polynomials of degree up to n. When dealing with these objects for the description of the ideal functionalities, we will make an abuse of notation for the IDs. We will often denote as one ID the list of all the IDs corresponding to the coefficients of a given object. For example, we could denote $\mathtt{id}_A$ the ID of a shared matrix $A = (a_{i,j})_{1 \leq i,j \leq n} \in \mathbb{F}_q^{n \times n}$, i.e. the $n \times n$ list of all the IDs $(\mathtt{id}_{a_{i,j}})_{1 \leq i,j \leq n}$ corresponding to the coefficients. Of course, the functionalities abort on illegal inputs, for example when receiving IDs of inconsistent size.

For communication estimates, we only consider multiplication between two shares and inner products between shared vectors. Indeed, almost all secret sharing schemes only require actual communication for these functionalities, as well as sharing and opening. Moreover, we assume that the sharing scheme is multiplicative, so that the cost of multiplication and inner product is the same. For each protocol, we provide asymptotic estimates for the number of secure multiplications required. In section 7.1, we provide the exact number of secure multiplications and rounds required for our decryption protocol, using the concrete parameter sets of HQC.

2.3 Existing MPC protocols

We present below some very classical MPC protocols together with straightforward extensions that will be required in our main protocols. All of them realize their corresponding functionalities in the $\mathcal{F}_{ABB}$-Hybrid model. The summary of round and communication complexities of these protocols is given in Table 1 below.

Table 1: Summary of basic MPC protocols. The value of τ can be any integer constant.

Name	Input	Output	Mult.	Rounds
GenInv	-	$[x] \xleftarrow{\$} \mathbb{F}_q^*$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Inv	$[x] \in \mathbb{F}_q^*$	$[x^{-1}]$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Products	$([x_1], \ldots, [x_k]) \in (\mathbb{F}_q^*)^k$	$\left[\prod_{i=1}^k x_i\right]$	$\mathcal{O}(k)$	$\mathcal{O}(1)$
PowersNZ	$k \in \mathbb{N}$, $[x] \in (\mathbb{F}_q^*)$	$[x^2], [x^3], \ldots, [x^k]$	$\mathcal{O}(k)$	$\mathcal{O}(1)$
Powers	$k \in \mathbb{N}$,$[x] \in \mathbb{F}_q$	$[x^2], [x^3], \ldots, [x^k]$	$\mathcal{O}(k)$	$\mathcal{O}(1)$
Poly2Mul	$[f], [g] \in \mathbb{F}_q[X]_{\leq d}$	$[fg]$	$2d+1$	$\mathcal{O}(1)$
PolyFromRoots	$([x_1], \ldots, [x_k]) \in \mathbb{F}_q^k$	$\left[\prod_{i=1}^k (X - x_i)\right]$	$\mathcal{O}(\tau k^{1+1/\tau})$	$\mathcal{O}(\tau)$
ZeroTest	$[x] \in \mathbb{F}_q$ with $x \in \mathbb{F}_{q'} \subseteq \mathbb{F}_q$	$[x \neq 0] \in \mathbb{F}_q$	$\mathcal{O}(q')$	$\mathcal{O}(1)$
PrefixOR	$([x_1], \ldots, [x_n]) \in \mathbb{F}_q^n$ with $x_i \in \{0,1\}$	$[x_1], [x_1 \vee x_2], \ldots, [\vee_{i=1}^n x_i]$	$\mathcal{O}(\tau n^{1+1/2\tau})$	$\mathcal{O}(\tau)$
Mat2Mul	$[A] \in \mathbb{F}_q^{n \times k}, [B] \in \mathbb{F}_q^{k \times m}$	$[AB] \in \mathbb{F}_q^{n \times m}$	$\mathcal{O}(nm)$	$\mathcal{O}(1)$
MatInv	$[A] \in \mathbb{F}_q^{n \times n}$	$[A^{-1}]$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$
TraceMul	$[A], [B] \in \mathbb{F}_q^{n \times n}$	$[\mathrm{Tr}(AB)] \in \mathbb{F}_q$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

- **GenInv**: This protocol from [BIB89] produces a randomly shared element of $\mathbb{F}_q^*$. It requires on average 2 secure multiplications and a constant number of rounds of communication.
- **Inv**: This protocol also from [BIB89] produces the inverse $[x^{-1}]$ from a shared element $[x]$ such that $x \in \mathbb{F}_q^*$. It requires on average 2 secure multiplications and a constant number of rounds of communication. Note that if this protocol is used on a zero shared element $[x]$, then the parties learn that $x = 0$.
- **Products**: This protocol, adapted from [BIB89], computes products of shared elements $[x_1], \ldots, [x_\ell]$ such that $x_i \in \mathbb{F}_q^*$, namely either the final product $[\prod_{i=1}^\ell x_i]$ or all the prefix products $[\prod_{i=1}^t x_i]$ for $1 \leq t \leq \ell$. For $1 \leq i \leq \ell$, the parties generate

random shared elements $[r_i]$ and $[s_i]$, compute and reveal $r_i s_i$ (restarting if $r_i s_i = 0$), and set $[r_i^{-1}] = (r_i s_i)^{-1}[s_i]$. They then reveal $m_1 = x_1 r_1^{-1}$ and $m_i = r_{i-1} x_i r_i^{-1}$ for $2 \leq i \leq \ell$, which are independent and uniformly distributed in $\mathbb{F}_q^*$. Since the masks telescope, the parties obtain locally

$$\left[\prod_{i=1}^{t} x_i\right] = \left(\prod_{i=1}^{t} m_i\right)[r_t] \quad \text{for } 1 \leq t \leq \ell.$$

Computing $[x_i r_{i-1}]$ in parallel with the products $r_i s_i$, the protocol requires on average $3\ell - 1$ secure multiplications and 4 rounds of communication. Note that if some x_i is zero, then the parties learn it.

- **PowersNZ**: this protocol takes a non-negative integer k and a shared invertible field element $[x]$ and compute all the powers $[x^2], [x^3], \ldots, [x^k]$. They are the prefix products of k copies of $[x]$, and are thus computed with **Products**, with the same complexity.
- **Powers**: this protocol takes a non-negative integer k and a shared field element $[x]$ (not necessarily invertible) and compute all the powers $[x^2], [x^3], \ldots, [x^k]$. Such a protocol is given in [CKP07] as a subprotocol for evaluating a shared element on public polynomials. The authors use a protocol from Bar-Ilan and Beaver [BIB89] on well-chosen 2×2 invertible matrices that depend on $[x]$ in order to compute the evaluation on $[x]$ of the Tchebychev polynomials up to degree k. Then, the shared powers of $[x]$ are computed as publicly-known linear combination of the evaluations. This requires $\mathcal{O}(k)$ secure multiplications and a constant number of rounds. However, their protocol is not applicable in fields of characteristic 2 because of properties of the Tchebychev polynomials. Nevertheless, a slightly modified variant of their algorithm is presented in [MV24] (in the context of matrix powering), and this variant is applicable to all finite fields with the same communication complexity.
- **Poly2Mul**: Mohassel and Franklin proposed in [MF06] a protocol for multiplying two shared polynomials of degree at most d in $\mathbb{F}_q[X]$ that requires $2d + 1$ secure multiplications and one round of multiplications in $\mathbb{F}_q$. This protocol uses evaluation-interpolation techniques, i.e. it evaluates the shared polynomials on public points, securely multiplies the evaluations, and interpolates back the result. As such, it requires the field $\mathbb{F}_q$ to have at least $2d + 1$ elements.
- **PolyFromRoots**: At multiple occasions, our protocols need to perform the following operation: given k shared elements $[x_1], \ldots, [x_k]$, compute the shared polynomial $[g] = \left[\prod_{i=1}^{k}(X - x_i)\right]$. However, computing an arbitrary number k of shared polynomials in $\mathbb{F}_q[X]$ would not be possible in constant round using **Poly2Mul**. The best option to achieve the desired operation is to use the unbounded fan-in polynomial multiplication protocol of [GLOV24]. To multiply k polynomials of degree at most d, this MPC protocol requires $\mathcal{O}(\tau k^{1+1/\tau} d)$ secure multiplication and $\mathcal{O}(\tau)$ rounds, where τ is a constant. Applied to our case, this leads to a protocol requiring $\mathcal{O}(\tau)$ rounds and $\mathcal{O}(\tau k^{1+1/\tau})$ secure multiplications. As this protocol uses **Poly2Mul** as a subroutine, it also requires the field $\mathbb{F}_q$ to have more than k elements. As noted in [GLOV24], this enables to compute the product of $[x_1], \ldots, [x_k]$ even if some of the elements might be zero by using this protocol. Indeed, protocols for unbounded multiplications usually only allow for multiplications of nonzero shared elements. In our case, it is done by noting that $[x_1 \cdots x_k]$ is the constant coefficient of $[g]$.
- **ZeroTest**: For our Reed-Solomon decoding protocol, we need a way to test if a shared element $[x]$, which we possibly know to be in a subfield $\mathbb{F}_{q'} \subseteq \mathbb{F}_q$, is zero. This means producing a shared element $[x \neq 0] \in \mathbb{F}_q$ such that $(x \neq 0) = 0$ if $x = 0$ and $(x \neq 0) = 1$ otherwise. In [DFK+06], such a protocol was given for sharings in

prime fields. In our case, we compute $[x^{q'-1}]$ using **Powers** in order to compute this value. This requires $\mathcal{O}(q')$ secure multiplications and a constant number of rounds.

- **PrefixOR**: With boolean values computed with **ZeroTest**, we need to compute some "prefix OR" operation, i.e., given $[x_1], \ldots, [x_k]$ whose values are known to be 0 or 1, we wish to compute $[x_1], [x_1 \vee x_2], \ldots, [\vee_{i=1}^{k} x_i]$. In [DFK+06, Section 5.2], the authors present a protocol for the "prefix-OR" operation in prime fields. In order to compute the operation on n elements, they use a previous protocol for the OR operation in a "baby-steps giant-steps" manner. After a careful examination of their protocol, one can remark that their protocol is still correct on fields of characteristic 2 as long as a subprotocol for the OR operation for this type of field is used. Moreover, the basic OR operation (i.e. computing $[\vee_{i=1}^{k} x_i]$ only) can be computed as $1 - \prod_{i=1}^{k}(1 - [x_i])$, which can be achieved using **PolyFromRoots**. This leads to a protocol for the "prefix OR" operation requiring at most $\mathcal{O}(\tau)$ rounds and $\mathcal{O}(\tau n^{1+1/2\tau})$ secure multiplications.
- **Mat2Mul**: This protocol computes the shared product $[AB] \in \mathbb{F}_q^{n\times m}$ of two shared matrices $[A] \in \mathbb{F}_q^{n\times k}$ and $[B] \in \mathbb{F}_q^{k\times m}$. Using secure inner product computations, this protocol has the same cost as $\mathcal{O}(nm)$ secure multiplications. This protocol is used in both [MW08] and [MV24].
- **MatInv**: Using **Mat2Mul** as a base for matrix multiplication, inverting a shared matrix $[A] \in \mathbb{F}_q^{n\times n}$ whose value is known to be non-singular can be done using a similar protocol than one from [BIB89]. This is done by generating a random invertible shared matrix $[R] \in \mathbb{F}_q^{n\times n}$ along with its inverse $[R^{-1}]$ and revealing the value RA, which can then be inverted in the clear. This protocol requires $\mathcal{O}(n^2)$ secure multiplications and a constant number of rounds. Note that if this protocol is used on a singular matrix then the row space of A can be deduced. Notably, this reveals the rank of A.
- **TraceMul**: In [MV24], the authors remarked that, given two matrices $[A] = [(a_{i,j})] \in \mathbb{F}_q^{n\times n}$ and $[B] = [(b_{i,j})] \in \mathbb{F}_q^{n\times n}$, the shared value $[\mathrm{Tr}(AB)] \in \mathbb{F}_q$ can be computed using one inner product, as the equation $[\mathrm{Tr}(AB)] = \sum_{1\le k,l\le n}[a_{k,l}][b_{l,k}]$ can be seen as an inner product between two shared vectors of size n^2: $[A]$ written in row-major order and $[B]$ in column-major order.

2.4 The HQC cryptosystem

The HQC cryptosystem [AAB+22] is a code-based public key encryption scheme whose security relies on a quasi-cyclic variant of the syndrome decoding problem. It uses operations in the ring $\mathcal{R}^{HQC} = \mathbb{F}_2[X]/(X^n - 1)$, where $n \in \mathbb{N}$ is one of the parameters of the scheme. The message space is $\mathcal{M} = \{0,1\}^\lambda$, where $\lambda = 128$, 192 or 256 (depending on the required security level of the scheme). It also uses three parameters w, $w_{\mathbf{r}}$, and $w_{\mathbf{e}}$ corresponding to the weights of the "error vectors" generated. We will denote by $\mathcal{R}_w^{HQC}$ (resp. $\mathcal{R}_{w_{\mathbf{r}}}^{HQC}$, $\mathcal{R}_{w_{\mathbf{e}}}^{HQC}$) the set of all elements in $\mathcal{R}^{HQC}$ having Hamming weight w (resp. $w_{\mathbf{r}}$, $w_{\mathbf{e}}$). The cryptosystem also uses a public concatenated code $\mathcal{C}$ of length $n_e n_i \le n$ and dimension λ.

The encryption algorithm uses a matrix $G \in \mathbb{F}_2^{\lambda\times n_e n_i}$ for the public code to encode the message, and uses an instance of the Quasi-Cyclic variant of the syndrome decoding problem (related to the public key) as a one-time pad. For decryption, the secret key is first used on the ciphertext to obtain a noisy codeword of the public code that corresponds to the message. Then, the decoder is applied on the result to obtain back the message. The formal description is given in Fig. 2.

During the decryption, the decoding itself is done in two parts, as HQC uses a concatenated code. The concatenated code is built from an internal error correcting code $\mathcal{C}_i$ over $\mathbb{F}_2$ of length n_i and dimension k_i, and an external error correcting code $\mathcal{C}_e$ over $\mathbb{F}_{q'}$

- $\underline{\text{Setup}^{HQC}(1^\lambda)}$: Test if $\lambda = 128$, 192, or 256, and return the parameters $(n, w, w_{\mathbf{r}}, w_{\mathbf{e}}, n_e n_i, G, \mathcal{C})$ corresponding to the desired security level.
- $\underline{\mathcal{K}^{HQC}(1^\lambda)}$: Retrieve the parameters (n, w) from $\text{Setup}^{HQC}(1^\lambda)$. Sample $\mathbf{h} \leftarrow^{\$} \mathcal{R}^{HQC}$ and $(\mathbf{x}, \mathbf{y}) \leftarrow^{\$} \mathcal{R}_w^{HQC} \times \mathcal{R}_w^{HQC}$. Compute $\mathbf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and $\mathbf{sk} = (\mathbf{x}, \mathbf{y})$. Output $(\mathbf{pk}, \mathbf{sk})$.
- $\underline{\mathcal{E}^{HQC}(\mathbf{m}, \mathbf{pk})}$: Retrieve the parameters $(n, w_r, w_e, n_e n_i, G)$ from $\text{Setup}^{HQC}(1^\lambda)$. Sample $\mathbf{e} \leftarrow^{\$} \mathcal{R}_{w_e}^{HQC}$ and $(\mathbf{r}_1, \mathbf{r}_2) \leftarrow^{\$} \mathcal{R}_{w_r}^{HQC} \times \mathcal{R}_{w_r}^{HQC}$. Compute $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \text{Truncate}(\mathbf{m}G + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}, n - n_e n_i)$. Return $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
- $\underline{\mathcal{D}^{HQC}(\mathbf{c}, sk)}$: Retrieve the parameters $(n, n_e n_i, \mathcal{C})$ from $\text{Setup}^{HQC}(1^\lambda)$. Interpret $\mathbf{c}$ as $(\mathbf{u}, \mathbf{v}) \in \mathcal{R}^{HQC} \times \mathcal{R}^{HQC}$. Compute $\mathbf{c}' = \text{Truncate}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y}, n - n_e n_i)$, and return $\mathcal{C}.\text{Decode}(\mathbf{c}')$.

Figure 2: The HQC.PKE cryptosystem

of length n_e and dimension k_e, such that $q' = 2^{k_i}$, and $k_e k_i = \lambda$. First, the noisy codeword $\mathbf{c}' \in \mathbb{F}_2^{n_e n_i}$ is seen as n_e noisy codewords $\mathbf{c}'_1, \ldots, \mathbf{c}'_{n_e} \in \mathbb{F}_2^{n_i}$ of the internal code. These chunks are all passed through the decoder for $\mathcal{C}_i$ to obtain $\mathbf{c}'' = (\text{Decode}_{\mathcal{C}_i}(\mathbf{c}'_1), \ldots, \text{Decode}_{\mathcal{C}_i}(\mathbf{c}'_{n_e}))$, requiring n_e calls to the decoder for $\mathcal{C}_i$. Each of the decoded chunks of $\mathbf{c}''$ can be seen as an element of $\mathbb{F}_{q'}$ since $q' = 2^{k_i}$. In the second step, $\mathbf{c}''$ is thus seen as an element of $\mathbb{F}_{q'}^{n_e}$ and passed through the decoder for $\mathcal{C}_e$, resulting in a word $\mathbf{m}' \in \mathbb{F}_{q'}^{k_e} \simeq \mathbb{F}_2^{\lambda}$, which is expected to be the plaintext.

If $\mathcal{C}_i$ is able to decode δ_i errors and $\mathcal{C}_i$ is able to decode δ_e errors, then the concatenated code $\mathcal{C}$ can correct at least $\delta_i \delta_e$ errors. The designers of HQC gave in [AAB+22] a more detailed analysis of the probability of the decoding process to fail, and chose the internal code and external code carefully according to this failure probability and efficiency considerations. For the external code, shortened Reed-Solomon codes are used. For the internal code, duplicated Reed-Muller codes of the first order are used. In the next subsection we introduce the basic definitions for duplicated Reed-Muller codes and shortened Reed-Solomon codes, as well as basic properties. In section 5 and 6, we show how to decode these codes, both in the classical and MPC setting.

2.4.1 Duplicated Reed-Muller of first order

Let us give some basic definitions and properties of duplicated Reed-Muller codes of first order. Informally, such codewords correspond to evaluations of multivariate polynomials of degree one in $\mathbb{F}_2[X]$. The duplicated variant means that this codeword is repeated a fixed number of times. In the following, we consider $m, \gamma \in \mathbb{N}$. The integer m is number of variables of the polynomials, and γ denotes the number of times the codeword is duplicated. For the "evaluation vector", we index the coefficients of vectors by elements of $\mathbb{F}_2^m$ (which means the vector is of length 2^m). Similarly, we index the coefficients the duplicated evaluation vector by elements of $[\![1, \gamma]\!] \times \mathbb{F}_2^m$. For more details on first-order Reed-Muller codes, see [MS77, Ch. 14].

Definition 1. The duplicated Reed-Muller code $RM_\gamma(1, m)$ is defined as:

$$RM_\gamma(1,m) = \left\{ (c, \ldots, c) \;\middle|\; c = \left(w_0 + \sum_{i=1}^{m} w_i v_i \right)_{v \in \mathbb{F}_2^m},\ w = (w_0, \ldots, w_m) \in \mathbb{F}_2^{m+1} \right\} \subseteq (\mathbb{F}_2^{2^m})^\gamma$$

It is a well-known result that the length of $RM_\gamma(1,m)$ is $\gamma \cdot 2^m$, its dimension is $m+1$, and its minimal distance is $\gamma \cdot 2^{m-1}$. From Definition 1, the coding algorithm is straightforward. Given a message $w \in \mathbb{F}_2^{m+1}$, evaluate $c_v = w_0 + \sum_{i=1}^{m} w_i v_i \in \mathbb{F}_2$ for all $v \in \mathbb{F}_2^m$. Output the vector $c \in \mathbb{F}_2^{2^m}$ duplicated γ times. Given the minimal distance of the code, the uniqueness decoding bound is $\gamma \cdot 2^{m-2}$. We give a decoding algorithm in Section 5, which serves as a baseline for our MPC protocol for Reed-Muller decoding.

2.4.2 Shortened Reed-Solomon

Here, we give basic definitions and properties about Reed-Solomon codes (the variant that is sometimes called BCH codes). We define the codes over a subfield $\mathbb{F}_{q'}$ of $\mathbb{F}_q$. Informally, a Reed-Solomon codeword is a polynomial of bounded degree that vanishes in subsequent powers of an elements α. See [AAB+22, GG03, LC04] for additional details on Reed-Solomon codes.

Definition 2. Let $m' > 1$, and $\delta \in \mathbb{N}^*$. Denote $q' = 2^{m'}$, $n' = q' - 1$ and α a primitive element of $\mathbb{F}_{q'}$. Define $g(X) = (X-\alpha)(X-\alpha^2)\dots(X-\alpha^{2\delta}) \in \mathbb{F}_{q'}[X]$. Let us denote $\mathcal{C} = BCH(q', \delta_e, \alpha)$ as all the multiples of g in $\mathbb{F}_{q'}[X]/(X^{n'}-1) \cong \mathbb{F}_{q'}^{n'}$. Then $BCH(q', \delta_e, \alpha)$ is a cyclic code with symbols in $\mathbb{F}_{q'}$, with length $n' = q' - 1$, dimension $k' = n' - 2\delta = q' - 1 - 2\delta$ and minimum distance $2\delta + 1$.

One can remark that, for a word $c \in \mathbb{F}_{q'}[X]/(X^{n'}-1)$, $c \in BCH(q', \delta, \alpha)$ if and only if for $1 \le i \le 2\delta$, $c(\alpha^i) = 0$. To encode a message $u \in \mathbb{F}_{q'}^{k'}$, the message is seen as a polynomial in $\mathbb{F}_{q'}[X]$ of degree less than k'. The corresponding codeword is computed as $c = X^{2\delta} \cdot u(X) + (X^{2\delta} \cdot u(X) \bmod g(X))$. Note that c is indeed in $\mathcal{C} = BCH(q', \delta, \alpha)$ (it is of degree less than n' and $c \bmod g = 0$) and it uniquely determines u. Given the minimal distance of $BCH(q', \delta, \alpha)$, a noisy codeword of $\mathcal{C}$ can be decoded up to δ errors. We give a decoding algorithm in Section 6, which serves as a model for our MPC protocol for Reed-Solomon decoding.

In HQC, shortened Reed-Solomon codes are considered. This allows to reduce the length and dimension of the code by the same parameter (that we denote ϵ), namely the parameters $[n', k', 2\delta]$ of the full codes become $[n_e, k_e, 2\delta_e] = [n' - \epsilon, k' - \epsilon, 2\delta]$ for the shortened code. The shortened code is obtained by imposing ϵ coordinates of a codeword to be zero, therefore allowing them to be deleted. For decoding, one can restore those zero coordinates and use the decoding algorithm for the regular code. For the sake of simplicity, we will consider here that the zero coordinates corresponding to the shortened code are the last coordinates of the codewords in HQC decryption.

2.4.3 HQC parameters instanciantion

According to the design constraints of HQC, parameters of the duplicated Reed Muller and shortened Reed-Solomon codes are linked. Indeed, if we let m and $\gamma \in \mathbb{N}$ such $\mathcal{C}_i = RM_\gamma(1,m)$, then we have $n_i = \gamma 2^m$, $k_i = m+1$ and $\delta_i = \gamma 2^{m-1}$. Then, the shortened Reed-Solomon code $\mathcal{C}_e$ has symbols in $\mathbb{F}_{q'}$ where $q' = 2^{k_i} = 2^{m+1}$. This means that the corresponding full Reed-Solomon code is of length $n' = q' - 1$. Furthermore, $k_e k_i = \lambda$, this gives a value for k_e as k_i is fixed, and therefore for k' as the shortening parameter ϵ is defined by $n' = q' - 1$ and n_e. This gives the value of δ_e as $k' = q' - 1 - 2\delta_e$.

The designers of HQC proposed 3 parameter sets achieving the 3 different security levels given by NIST. We recall the values of parameters sets of HQC relevant to us in table 2.

Table 2: Values from the parameter sets for HQC.

Instance	Security	n_e	k_e	δ_e	n_i	k_i	δ_i	n
HQC1	NIST-1	46	16	15	348	8	87	17 669
HQC3	NIST-3	56	24	16	640	8	160	35 851
HQC5	NIST-5	90	32	29	640	8	160	57 637

3 Distributed characteristic polynomial computation over field of characteristic two

In this section, we present a protocol for characteristic polynomial computation of shared matrices in fields of characteristic two. This protocol is necessary to construct a constant-round protocol for solving Padé approximants problems. To build such a protocol, the authors of [MV24] briefly suggest to combine principles of their protocol for the same operation in fields of high characteristic with an algorithmic solution from Schönhage in [Sch93]. In this section, we provide a deeper analysis and solve any potential technical issues for building such a protocol. Notably, the solution from Schönhage relies heavily on computations on a well-chosen polynomial ring, namely $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$. Therefore, new protocols for dealing with operations on shared matrices and polynomials with coefficients in $\mathcal{R}$ are needed. We design all the ones we need in section 3.1. Then, using these protocols, we detail our techniques to securely compute the characteristic polynomial in section 3.2

3.1 MPC protocols for computation in polynomial rings

In this section, we present multiple MPC protocols to handle shared elements having coefficients in a polynomial ring $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, namely polynomials of $\mathcal{R}[X]$ and matrices of $\mathcal{R}^{n\times n}$ for any $n \in \mathbb{N}$. Functionality $\mathcal{F}_{ABB+\texttt{PolyRing}(\mathcal{R})}$ (see Fig. 3) describe formally the operations we want to achieve. To our knowledge, this is the first paper in which these protocols are formally described, although the techniques used are quite standard. While we go into the details of these protocols in the rest of this section, Table 3 summarizes all the operations enabled by the protocol discussed here, as well as their asymptotic communication complexities.

Table 3: Summary of our protocols for operations over $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$ as well as their communication complexity

Name	Input	Output	Mult.	Rounds
PolyMatMul	$[A] \in \mathcal{R}^{n\times k}$, $[B] \in \mathcal{R}^{k\times m}$	$[AB] \in \mathcal{R}^{n\times m}$	$\mathcal{O}(nmd)$	$\mathcal{O}(1)$
PolyTraceMul	$[A] \in \mathcal{R}^{n\times n}$, $[B] \in \mathcal{R}^{n\times n}$	$[\mathrm{Tr}(AB)] \in \mathcal{R}$	$\mathcal{O}(d)$	$\mathcal{O}(1)$
BiPoly2Mul	$[f] \in \mathcal{R}[X]_{\leq n}$, $[g] \in \mathcal{R}[X]_{\leq n}$	$[fg] \in \mathcal{R}[X]$	$\mathcal{O}(nd)$	$\mathcal{O}(1)$
PolyMatInv	$[A] \in \mathcal{R}^{n\times n}$, $C \in (\mathbb{F}_q^{n\times n})^\times$ s.t. $A \bmod Y = C$	$[A^{-1}] \in \mathcal{R}^{n\times n}$	$\mathcal{O}(n^2 d)$	$\mathcal{O}(1)$
PolyMatPowers	$[A] \in \mathcal{R}^{n\times n}$, $C \in (\mathbb{F}_q^{n\times n})^\times$ s.t. $A \bmod Y = C$	$[A^2], \ldots, [A^k] \in \mathcal{R}^{n\times n}$	$\mathcal{O}(kn^2 d)$	$\mathcal{O}(1)$
PolySeriesInv	$[f] \in \mathcal{R}[X]_{<n}$, $c \in \mathbb{F}_q^\times$ s.t. $f \bmod Y = c$	$[f^{-1} \bmod X^n] \in \mathcal{R}[X]_{<n}$	$\mathcal{O}(nd)$	$\mathcal{O}(1)$

The protocols in this section mostly rely on the following observation: for $[a] \in \mathcal{R}$, we denote $[\tilde{a}] \in \mathbb{F}_q[Y]$ the lifting of $[a]$, i.e. the shared polynomial whose value is the representative in $\mathbb{F}_q[Y]$ of degree at most d representing $a \in \mathcal{R}$. For $[a], [b] \in \mathcal{R}$, we have that $[ab] = [\tilde{a}\tilde{b}] \bmod Y^{d+1}$. Similarly, we can define liftings for polynomials of $\mathcal{R}[X]$ and matrices with coefficients in $\mathcal{R}$. Products of matrices or polynomials can be computed using

The functionality is the same as $\mathcal{F}_{ABB}$ with five more functionality that are parameterized by $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$:

- Matrix product in $\mathcal{R}$: Upon receiving $(\texttt{PolyMatMul}, \texttt{id}_1, \texttt{id}_2, \texttt{id}')$ from all the parties, the functionality fetches $A \in \mathcal{R}^{n\times k}$ at $\texttt{id}_1$, $B \in \mathcal{R}^{k\times m}$ at $\texttt{id}_2$, computes $AB \in \mathcal{R}^{n\times m}$ and stores it at $\texttt{id}'$.
- Trace of matrix product in $\mathcal{R}$: Upon receiving $(\texttt{PolyTraceMul}, \texttt{id}_1, \texttt{id}_2, \texttt{id}')$ from all the parties, the functionality fetches $A \in \mathcal{R}^{n\times n}$ at $\texttt{id}_1$, $B \in \mathcal{R}^{n\times n}$ at $\texttt{id}_2$, computes $\mathrm{Tr}(AB) \in \mathcal{R}$ and stores it at $\texttt{id}'$.
- Polynomial product in $\mathcal{R}$: Upon receiving $(\texttt{BiPoly2Mul}, \texttt{id}_1, \texttt{id}_2, \texttt{id}')$ from all the parties, the functionality fetches $f \in \mathcal{R}[X]_{\leq d}$ at $\texttt{id}_1$, $g \in \mathcal{R}[X]_{\leq d}$ at $\texttt{id}_2$, computes $fg \in \mathcal{R}[X]_{\leq 2d}$ and stores it at $\texttt{id}'$.
- Matrix inversion in $\mathcal{R}$: Upon receiving $(\texttt{PolyMatInv}, \texttt{id}, \texttt{id}', C)$, the functionality checks that $C \in (\mathbb{F}_q^{n\times n})^\times$. If so, it fetches $A' \in \mathcal{R}^{n\times n}$, set $A = C + YA$, computes A^{-1} and stores it at $\texttt{id}'$. Otherwise, it rejects.
- Matrix powering in $\mathcal{R}$: Upon receiving $(\texttt{PolyMatPower}, \texttt{id}, (\texttt{id}'_i)_{2\leq i\leq k}, C)$, the functionality checks $C \in (\mathbb{F}_q^{n\times n})^\times$. If so, it fetches $A' \in \mathcal{R}^{n\times n}$, sets $A = C + YA'$, computes $A^2, \ldots, A^k$ and stores them at $\texttt{id}'_2, \ldots, \texttt{id}'_k$. If not, it rejects.
- Series inversion in $\mathcal{R}$: Upon receiving $(\texttt{PolySeriesInv}, \texttt{id}, \texttt{id}'_i, c)$, the functionality checks that $c \in \mathbb{F}_q^*$. If so, it fetches $g \in \mathcal{R}[X]_{<n-1}$ at $\texttt{id}$, computes $f = c + Xg$, computes $f^{-1} \bmod X^n$ and stores it at $\texttt{id}'_i$.

Figure 3: $\mathcal{F}_{ABB+\texttt{PolyRing}(\mathcal{R})}$

products of liftings and reductions mod Y^{d+1}. These products of liftings are computed using evaluation-interpolation techniques. As an example, for two square shared matrices $[A]$ and $[B] \in \mathcal{R}^{n\times n}$, each coefficient of $[\tilde{A}\tilde{B}]$ is a polynomial of degree at most $2d$, and therefore can be interpolated back locally knowing $[\tilde{A}\tilde{B}(\alpha_i)]$, where $\alpha_1, \ldots, \alpha_{2d}$ are distinct points in $\mathbb{F}_q$. The value $[\tilde{A}\tilde{B}(\alpha_i)]$ can be computed with one secure matrix product in $\mathbb{F}_q^{n\times n}$, as $[\tilde{A}(\alpha_i)]$ and $[\tilde{B}(\alpha_i)]$ can be locally computed from $[A]$ and $[B]$ (remember that α_i is a public point). Moreover, we know that $[AB] = [\tilde{A}\tilde{B}] \bmod Y^{d+1}$, which means that $[AB]$ can be computed locally knowing $[\tilde{A}\tilde{B}]$. This technique is used in protocol **PolyMatMul**, which computes the product two shared matrices with coefficients in $\mathcal{R}$ (generalized to handle non-square matrices). Similar techniques are used in protocol **PolyTraceMul**, which computes $[\mathrm{Tr}(AB)] \in \mathcal{R}$ instead, and protocol **BiPoly2Mul**, which computes the product $[fg] \in \mathcal{R}[X]$ from two polynomials $[f]$ and $[g] \in \mathcal{R}[X]$. These protocols use a number of secure multiplications which is linear in the size of the shared input elements.

Protocol 1: PolyMatMul

Parameters: $n, k, m \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, $\alpha_1, \ldots, \alpha_{2d+1}$ distinct points of $\mathbb{F}_q$.
Input: $[A] \in \mathcal{R}^{n\times k}$, $[B] \in \mathcal{R}^{k\times m}$
Output: $[AB] \in \mathcal{R}^{n\times m}$

1 Parties locally compute the liftings $[\tilde{A}] \in \mathbb{F}_q[Y]^{n\times k}$ and $[\tilde{B}] \in \mathbb{F}_q[Y]^{k\times m}$ ▷ no com.
2 **for** $1 \leq i \leq 2d+1$ **in parallel do**
3 Parties locally compute $[\tilde{A}(\alpha_i)] \in \mathbb{F}_q^{n\times k}$ and $[\tilde{B}(\alpha_i)] \in \mathbb{F}_q^{k\times m}$ ▷ no com.
4 $[\tilde{A}\tilde{B}(\alpha_i)] \leftarrow [\tilde{A}(\alpha_i)][\tilde{B}(\alpha_i)]$ ▷ **Mat2Mul**
5 **end**
6 Parties locally interpolate $[\tilde{A}\tilde{B}] \in \mathbb{F}_q[Y]^{n\times m}$ from $[\tilde{A}\tilde{B}(\alpha_1)], \ldots, [\tilde{A}\tilde{B}(\alpha_{2d})]$ ▷ no com.
7 $[AB] \leftarrow [\tilde{A}\tilde{B}] \bmod Y^{d+1}$ ▷ no com.

Protocol 2: PolyTraceMul

Parameters: $n \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, $\alpha_1, \dots, \alpha_{2d+1}$ distinct points of $\mathbb{F}_q$.
Input: $[A], [B] \in \mathcal{R}^{n\times n}$
Output: $[\mathrm{Tr}(AB)] \in \mathcal{R}^{n\times n}$

1 Parties locally compute the liftings $[\tilde{A}]$ and $[\tilde{B}] \in \mathbb{F}_q[Y]^{n\times n}$ ▷ no com.
2 **for** $1 \le i \le 2d+1$ **in parallel do**
3 Parties locally compute $[\tilde{A}(\alpha_i)]$ and $[\tilde{B}(\alpha_i)]$ ▷ no com.
4 $[\mathrm{Tr}(\tilde{A}\tilde{B})(\alpha_i)] \leftarrow [\mathrm{Tr}(\tilde{A}(\alpha_i) \cdot \tilde{B}(\alpha_i))]$ ▷ **TraceMul**
5 **end**
6 Parties locally interpolate $[\mathrm{Tr}(\tilde{A}\tilde{B})]$ from $[\mathrm{Tr}(\tilde{A}\tilde{B})(\alpha_1)], \dots, [\mathrm{Tr}(\tilde{A}\tilde{B})(\alpha_{2d})]$ ▷ no com.
7 $[\mathrm{Tr}(AB)] \leftarrow [\mathrm{Tr}(\tilde{A}\tilde{B})] \bmod Y^{d+1}$ ▷ no com.

Protocol 3: BiPoly2Mul

Parameters: $n \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, $\alpha_1, \dots, \alpha_{2d+1}$ distinct points of $\mathbb{F}_q$.
Input: $[f] \in \mathcal{R}[X]$ and $[g] \in \mathcal{R}[X]$ of degree up to n
Output: $[fg] \in \mathcal{R}[X]$

1 Parties locally compute the liftings $[\tilde{f}(Y,X)]$ and $[\tilde{g}(Y,X)] \in \mathbb{F}_q[Y][X]$ ▷ no com.
2 **for** $1 \le i \le 2d+1$ **in parallel do**
3 Parties locally compute $[\tilde{f}(\alpha_i, X)]$ and $[\tilde{f}(\alpha_i, X)] \in \mathbb{F}_q[X]$ ▷ no com.
4 $[\tilde{f}\tilde{g}(\alpha_i, X)] \leftarrow [\tilde{f}(\alpha_i, X)][\tilde{g}(\alpha_i, X)] \in \mathbb{F}_q[X]$ ▷ **Poly2Mul**
5 **end**
6 Parties locally interpolate $[\tilde{f}\tilde{g}(Y,X)]$ from $[\tilde{f}\tilde{g}(\alpha_1, X)], \dots, [\tilde{f}\tilde{g}(\alpha_{2d+1}, X)]$ ▷ no com.
7 $[fg] \leftarrow [\tilde{f}\tilde{g}(Y,X)] \bmod Y^{d+1}$ ▷ no com.

Using protocol **PolyMatMul**, we can build MPC protocols for securely computing inversion and successive powering of a matrix in $\mathcal{R}^{n\times n}$ and whose constant matrix (relative to the variable Y) is invertible and publicly known. More precisely, given $[A'] \in \mathcal{R}^{n\times n}$ and $C \in \mathbb{F}_q^{n\times n}$ a public invertible matrix, we wish to compute either $[A^{-1}] \in \mathcal{R}^{n\times n}$, or $[A^2], \dots, [A^k] \in \mathcal{R}^{n\times n}$, where $[A] = C + Y[A']$. To this end, we adapt the usual protocols of Bar-Ilan and Beaver [BIB89], usually presented for inverting elements and computing unbounded multiplications of elements of a multiplicative group. Even though the set of matrices of $\mathcal{R}^{n\times n}$ that have C as their constant matrix is not a group, one can observe that the matrices of $\mathcal{R}^{n\times n}$ that have I_n as their constant matrix exert a group action on it. This allows to mask the matrix $[A]$ by matrices whose constant matrix is I_n, as any revealed matrix will just be a random matrix in $\mathcal{R}^{n\times n}$ whose constant matrix is C. Protocols **PolyMatInv** (for matrix inversion), and **PolyMatPowers** and (for matrix powering) details these operations.

Protocol 4: PolyMatInv

Parameters: $n \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, $C \in (\mathbb{F}_q^{n\times n})^\times$.
Input: $[A'] \in \mathcal{R}^{n\times n}$
Output: $[A^{-1}] \in \mathcal{R}^{n\times n}$ where $A = C + YA' \in (\mathcal{R}^{n\times n})^\times$

1 $[R'] \leftarrow^{\$} \mathcal{R}^{n\times n}$
2 $[R] \leftarrow I_n + Y[R']$ ▷ no com.
3 $[A] \leftarrow c + Y[A']$ ▷ no com.
4 $[RA] \leftarrow [R][A]$ ▷ **PolyMatMul**
5 Parties reveal RA and locally compute $(RA)^{-1} \in \mathcal{R}^{n\times n}$
6 $[A^{-1}] \leftarrow (RA)^{-1}[R]$ ▷ no com.

Protocol 5: PolyMatPowers

Parameters: $n, k \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, $C \in (\mathbb{F}_q^{n\times n})^\times$.
Input: $[A'] \in \mathcal{R}^{n\times n}$
Output: $[A^2], \ldots, [A^k] \in \mathcal{R}^{n\times n}$ where $A = C + YA' \in (\mathcal{R}^{n\times n})^\times$

$[A] \leftarrow C + Y[A']$ ▷ no com.
for $1 \le l \le k$ **in parallel do**
  $[R'_l] \leftarrow^{\$} \mathcal{R}^{n\times n}$
  $[R_l] \leftarrow I_n + Y[R'_l]$ ▷ no com.
  Parties compute $[R_l^{-1}]$ ▷ **PolyMatInv**
  $[B_l] \leftarrow [R_{l-1}^{-1}][A][R_l]$ ▷ **PolyMatMul**, $R_0 = I_n$
  Parties reveal B_l
end
for $2 \le l \le k$ **in parallel do**
  $[A^l] \leftarrow B_1 B_2 \ldots B_l [R_l^{-1}]$ ▷ no com.
end

Finally, using similar techniques as matrix inversion, we also build a protocol for inverting a shared polynomial modulo X^n in $\mathcal{R}[X]$ when the constant coefficient (according to X) is invertible in $\mathbb{F}_q$ and publicly known. Our protocol, named **PolySeriesInv**, is really similar to a protocol presented in [MF06] for inverting shared polynomials of $\mathbb{F}_q[X]$ modulo X^n.

Protocol 6: PolySeriesInv

Parameters: $n \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$, $c \in (\mathbb{F}_q^{n\times n})^*$
Input: $[g] \in \mathcal{R}[X]_{<n-1}$
Output: $[f^{-1} \bmod X^n] \in \mathcal{R}[X]$ where $f = c + Xg \in \mathcal{R}[X]_{<n}$

$[s] \leftarrow^{\$} \mathcal{R}[X]_{<n-1}$
$[r] \leftarrow 1 + X[s]$ ▷ no com.
$[f] \leftarrow 1 + X[g]$ ▷ no com.
$[rf \bmod X^n] \leftarrow [r][f] \bmod X^n$ ▷ **BiPoly2Mul** + truncation
Parties reveal $rf \bmod X^n$ and locally compute $(rf)^{-1} \bmod X^n$
$[f^{-1} \bmod X^n] \leftarrow (fr)^{-1}[r] \bmod X^n$ ▷ no com.

Theorem 1. *Let $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$. The protocols **PolyMatMul**, **PolyTraceMul**, **BiPoly2Mul**, **PolyMatInv**, **PolyMatPowers**, and **PolySeriesInv** instantiate the ideal functionality $\mathcal{F}_{ABB+\texttt{PolyRing}(\mathcal{R})}$ in the $\mathcal{F}_{ABB}$-hybrid model. Table 3 give the communication complexity of each of these protocols.*

3.2 Protocol for characteristic polynomial computation

We are ready to describe a protocol for characteristic polynomial computation for fields whose size is a power of two. In [MV24, Appendix 9], the authors suggest to adapt the algorithmic solution of Schönhage in [Sch93] to the MPC setting for that case. We undertake a thorough investigation of this idea and establish that it can indeed be developed into a protocol for characteristic polynomial computation over fields of characteristic two.

Given a shared matrix $[A] \in \mathbb{F}_q^{n\times n}$, we wish to compute the shared characteristic polynomial of $[A]$, which we denote as $[\chi_A]$. We denote this functionality as $\mathcal{F}_{ABB+\texttt{Charpoly}}$ (see Fig. 4).

The functionality is the same as $\mathcal{F}_{ABB}$ with one more functionality.

- Upon receiving $(\texttt{CharPoly}, \texttt{id}, \texttt{id}')$ from all the parties, the functionality fetches $A \in \mathbb{F}_q^{n\times n}$ at $\texttt{id}$ and computes the degree n polynomial $\chi_A = \det(XI_n - A)$. Then, it stores χ_A at $\texttt{id}'$.

Figure 4: $\mathcal{F}_{ABB+\texttt{Charpoly}}$

In [MV24], the authors present an MPC protocol for this computation in the case of fields of characteristic greater than n. The protocol is based on Leverrier's Lemma [MV24, Lemma 3] that relates the coefficients of χ_A and the traces of power of A. It encompasses two main steps. First, the shared values $[\mathrm{Tr}(A^l)]$ are computed securely for $1 \leq l \leq n$. This is done using the "Baby-Steps Giant-Steps" paradigm, for communication efficiency. Using these values, the shared characteristic polynomial is computed by solving specific linear system, which is called the Leverrier system. This system is always invertible for field with characteristic larger than n. Unfortunately, this is not true anymore if q is a power of two and their protocol is not applicable. This is where we begin to leverage the ideas from [Sch93] in this context.

Let us consider the case in which the size of $[A]$ is odd. If that's not the case, parties can replace $[A]$ by $\left[\begin{array}{c|c} \mathrm{A} & \\ \hline & 1 \end{array}\right]$, as its characteristic polynomial is $[\chi_A] \cdot (X-1)$. To always have a invertible system, the idea is to replace the original matrix A with a polynomial matrix related to $[A]$. More specifically, the matrix $[A]$ is replaced by $[B] = C + YC[A] \in \mathcal{R}^{n \times n}$, where $C \in \mathbb{F}_q^{n \times n}$ is the following permutation matrix:

$$C = \begin{pmatrix} & & & 1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{pmatrix} \tag{1}$$

This choice for C ensures that $\chi_C = X^n + 1$, and in particular $\det(C) = 1$, implying $[\det(B)] = [\det(C(I_n + YA))] = [\mathrm{rev}(\chi_A(Y))]$, the reverse polynomial of χ_A. This means that $[\chi_A(Y)]$ can be deduced from $[\chi_B] \in \mathcal{R}[X]$ (as $[\det(B)]$ is the constant coefficient in X of $[\chi_B]$ up to sign). The characteristic polynomial $[\chi_A]$ is then obtained by reversing its coefficients, since χ_A has degree exactly n. If A was replaced by $\mathrm{diag}(A, 1)$ (when n is even), the characteristic polynomial of A is finally obtained by dividing the result by $X+1$. Both operations are linear, and thus do not require any communication. In order to get $[\chi_B]$, the protocol first computes the shared values $[\mathrm{Tr}(B^l)]$ as in the large characteristic case [MV24]. Then, one can use the ideas of Schönhage [Sch93] and Pan [Pan00] to replace linear system solving with Padé approximant. Let $P = \mathrm{rev}(\chi_B) \in \mathcal{R}[X]$ be the reverse polynomial of χ_B. Following [Pan00], we split P according to odd and even monomial, i.e. we write $P = P_0(X^2) + XP_1(X^2)$ such that $\deg(P_0), \deg(P_1) \leq n_0 = (n-1)/2$, and $P_0(0) = 1$ (as χ_B is monic). We then denote $Q = P_1/P_0 \in \mathcal{R}(X)$ and $S = 1 + \sum_{i \geq 1} \mathrm{Tr}(B^k)X^k \in \mathcal{R}[[X]]$. This allows to replace the Leverrier's linear system with the following formal power series relation:

$$Q(X^2) = \frac{P_1(X^2)}{P_0(X^2)} = \frac{1 + S^{-1}}{X}. \tag{2}$$

Since this equation is also valid mod X^{2n}, one can compute the shared polynomial $[Q(X^2) \bmod X^{2n}]$. In particular, this amounts to invert $[S \bmod X^{2n+1}]$ in $\mathcal{R}[X]/X^{2n+1}$ (which can be handled by the protocol **PolySeriesInv**) and apply some linear operations (as adding 1 is linear and dividing by X amounts to shifting coefficients). Then, $[Q(X) \bmod X^n]$ is deduced from $[Q(X^2) \bmod X^{2n}]$. As shown in [Sch93], it is possible to find the coefficients of $[P_0]$ by solving a $n_0 \times n_0$ shared linear system deduced from the degree constraints on P_0 and P_1 as well as the equation

$$P_1 = QP_0 \bmod X^n.$$

Note that this is a special case of a Padé approximant problem. Then, $[P_1]$ can be computed with a call to **BiPoly2Mul** and $[\chi_B]$ can be set as $[\mathrm{rev}(P_0(X^2) + XP_1(X^2))]$. Let us detail the linear system to solve in order to recover the coefficients of P_0. We note

$Q \bmod X^n = \sum_{i=0}^{n} q_i X^i$ and $P_0 = \sum_{j=0}^{n_0} a_j$, with $a_0 = 1$. Since $\deg P_1 \leq n_0$, we have that for $n_0 + 1 \leq j \leq n - 1$, the coefficient corresponding to X^j of $QP_0 \bmod X^n$ is zero, i.e.:

$$0 = \sum_{i=0}^{j} a_i q_{j-i}, \text{ i.e. } q_j = \sum_{i=1}^{n_0} a_i q_{j-i} \text{ , since } \deg(P_0) \leq n_0.$$

Since $2n_0 = n - 1$, this yields the following $n_0 \times n_0$ Toeplitz linear system:

$$M \cdot a = \begin{pmatrix} q_{n_0} & q_{n_0-1} & \cdots & q_1 \\ q_{n_0+1} & q_{n_0} & \cdots & q_2 \\ \vdots & \vdots & \vdots & \vdots \\ q_{n-2} & q_{n-3} & \cdots & q_{n_0} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n_0} \end{pmatrix} = \begin{pmatrix} q_{n_0+1} \\ q_{n_0+2} \\ \vdots \\ q_{n-1} \end{pmatrix} = b \tag{3}$$

Note that the coefficients of $[M]$ and $[b]$ are coefficients of $[Q(X) \bmod X^n]$. Therefore, parties can deduce $[M]$ and $[b]$ from $[Q(X) \bmod X^n]$ without any communication. Let us look at the value of the constant coefficient of M according to Y (remember that $M \in \mathcal{R}^{n_0 \times n_0}$ and $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$). Note that $\chi_B \bmod Y = \chi_C = X^n + 1$. Therefore, $Q(X) \bmod Y = X^{n_0}/1 = X^{n_0}$. This means that for $1 \leq i \leq n$, $q_i \bmod Y = 1$ if $i = n_0$ and 0 otherwise. Thus we know that even though $[M]$ is a shared matrix in $\mathcal{R}^{n \times n}$, its constant coefficient according to Y is always equal to I_{n_0}. This means $[M]$ is invertible over $\mathcal{R}$, and it can be inverted using **PolyMatInv**. Note that, in the general case of trying to solve Padé approximants, the matrix M might be singular and therefore make MPC Padé approximant solving harder than matrix inversion. The choice of C made the problem easier to solve than in the general case.

The most communication intensive operations are linked to the powering steps of the matrix $[B] \in \mathcal{R}^{n \times n}$ used to compute the shared values $[\mathrm{Tr}(B^l)]$. We provide a detailed description of the protocol **CharPoly** below, and Theorem 2 states its security and complexity.

Protocol 7: CharPoly

Parameters: $n \in \mathbb{N}$, $\mathcal{R} = \mathbb{F}_q[Y]/Y^{n+1}$, $k = \lceil \sqrt{2n} \rceil$, $n_0 = \lfloor n/2 \rfloor$, $C \in \mathbb{F}_q^{n \times n}$ as defined in equation 1
Input: $[A] \in \mathbb{F}_q^{n \times n}$
Output: $[\chi_A] \in \mathbb{F}_q[X]$ where $\chi_A = \det(XI_n - A)$

1 If n is even, replace n by $n+1$ and $[A]$ by $[\mathrm{diag}(A, 1)]$ ▷ no com.
2 Parties set $[B] \leftarrow C + YC[A] \in \mathcal{R}^{n \times n}$ ▷ no com.
3 Parties compute $[B], [B^2], \ldots, [B^k]$ ▷ **PolyMatPowers**
4 Parties compute $[B^k], [B^{2k}], \ldots, [B^{k^2}]$ ▷ **PolyMatPowers**
5 **for** $0 \leq i < k$, $0 \leq j \leq k$ **in parallel do**
6 $\quad [\mathrm{Tr}(B^{jk+i})] \leftarrow [\mathrm{Tr}(B^{jk} \cdot B^i)]$ ▷ **PolyTraceMul**
7 **end**
8 $[S] \leftarrow 1 + \sum_{l=1}^{2n} [\mathrm{Tr}(B^l)] X^l \in \mathcal{R}[X]$ ▷ no com.
9 Parties compute $[S^{-1} \bmod X^{2n+1}]$ ▷ **PolySeriesInv**
10 $[Q(X^2)] \leftarrow (1 + [S^{-1} \bmod X^{2n+1}])/X$ ▷ no com.
11 Parties deduce $[Q(X)]$ from $[Q(X^2)]$ ▷ no com.
12 Parties deduce $[M] \in \mathcal{R}^{n_0 \times n_0}$ and $[b] \in \mathcal{R}^{n_0}$ from $[Q(X)]$ ▷ no com., see eq. 3
13 Parties compute $[M^{-1}] \in \mathcal{R}^{n_0 \times n_0}$ ▷ **PolyMatInv**
14 $[a] \leftarrow [M^{-1}][b] \in \mathcal{R}^{n_0}$ ▷ **PolyMatMul**
15 $[P_0] \leftarrow 1 + \sum_{l=1}^{n_0} [a_l] X^l$ ▷ no com.
16 $[P_1] \leftarrow [P_0][Q(X)] \bmod X^n$ ▷ **BiPoly2Mul**
17 $[\chi_B] \leftarrow \mathrm{rev}([P_0(X^2)] + X[P_1(X^2)]) \in \mathcal{R}[X]$ ▷ no com.
18 $[\mathrm{rev}(\chi_A)] \leftarrow [\chi_B] \bmod X \in \mathbb{F}_q[Y]$ ▷ no com.
19 $[\chi_A] \leftarrow \mathrm{rev}([\mathrm{rev}(\chi_A)]) \in \mathbb{F}_q[X]$ ▷ no com.
20 If $[A]$ was replaced by $[\mathrm{diag}(A, 1)]$, $[\chi_A] \leftarrow [\chi_A]/(X + 1)$ ▷ no com.

Theorem 2. *Given a shared matrix $[A] \in \mathbb{F}_q^{n\times n}$, the Protocol* ***CharPoly*** *instantiates $\mathcal{F}_{ABB+\texttt{CharPoly}}$ (see Fig. 4) in the $\mathcal{F}_{ABB}$-hybrid model, and requires at most $\mathcal{O}(n^{3.5})$ secure multiplications and a constant number of rounds of multiplications.*

Proof. Since every protocol instantiate their corresponding functionality, one can build a simulator emulating the correct views of the environment from the ideal functionality $\mathcal{F}_{ABB+\texttt{CharPoly}}$. Notably, it is publicly known that $[B]$ and $[B^k]$ have C and C^k respectively as constant coefficient according to Y. Therefore, **PolyMatPowers** can be correctly invoked at steps 3 and 4. Similarly, it is publicly know that the constant coefficient of $[M]$ is I_{n_0}, and so **PolyMatInv** is correctly invoked.

For the communication complexity estimate, one must notice that $k = \mathcal{O}(\sqrt{n})$ and $\mathcal{R} = \mathbb{F}_q[Y]/Y^{d+1}$ with $d = n$, and it is sufficient to deduce the values from Table 3. More precisely, Steps 3 and 4 (**PolyMatPowers**) require $\mathcal{O}(n^2kd) = \mathcal{O}(n^{3.5})$ secure multiplications and a constant number of rounds. The $k\times k = \mathcal{O}(n)$ calls to **PolyTraceMul** are done in parallel, still yielding a constant number of round and a total number of secure multiplications that is $\mathcal{O}(n^2)$. Steps 9 (**PolySeriesInv**) and Step 16 **BiPoly2Mul**) concern a single constant round operation on polynomials of degree $\mathcal{O}(n)$ in $\mathcal{R}[\mathcal{X}]$ which both cost $\mathcal{O}(nd) = \mathcal{O}(n^2)$. Steps 13 (**PolyMatInv**) and Step 14 (**PolyMatMul**) are also constant round and both manipulate matrices of $\mathcal{R}^{n\times n}$, implying a cost of $\mathcal{O}(n^2 \times d) = \mathcal{O}(n^3)$. As claimed, the whole protocol exhibits a constant number of rounds and $\mathcal{O}(n^{3.5})$ secure multiplications.

□

4 Distributed Padé approximant solving

One of the crucial step while decoding Reed-Solomon codewords is finding the error-locator polynomial. Using the syndrome of the erroneous codeword, one can find such a polynomial by solving a Padé approximant problem. In the classical setting, the latter problem may be solved using the Berlekamp-Massey algorithm [Mas69] or an adaptation of extended euclidean algorithm. However, adapting these algorithms in the MPC setting is challenging, as these methods often leak information about the data (for example, the steps of the extended euclidean algorithm depends on the degree sequence of the input polynomials). Moreover, they do not lead to constant-round protocols straightforwardly.

In this section, we present a novel constant-round MPC protocol for solving such Padé approximant problems. As our protocol makes use of linear algebra, it relies on properties of subresultants and Padé approximants. We introduce the required background in section 4.1. Then, we give a detailed description of the protocol as well as the techniques used in section 4.2

4.1 Background on Padé approximants and sub-resultants

We first present some useful background on Padé approximants and on subresultants. The use of subresultants matrices allows to make use of linear algebra tools for MPC protocols (especially the protocol for characteristic polynomial of section 3), which allow to build constant-round solutions. These matrices in an MPC setting were first used in [MF06] for computing the gcd of two shared polynomials. For an extensive presentation on Padé approximants and subresultants, we suggest reading [GG03].

Definition 3. [GG03, Ch. 5.9] Let $\mathbb{F}$ be a field and $f \in \mathbb{F}[X]$ a polynomial of degree less than $n \in \mathbb{N}$. Let $0 \leq k \leq n$. The $(k, n-k)$-Padé approximant problem aims to find two polynomials $(r, t) \in \mathbb{F}[X]^2$ such that

$$\frac{r}{t} = f \mod X^n, \ \deg r < k, \ \deg t \leq n-k, \text{ and } t(0) \neq 0. \tag{4}$$

If such polynomials do not exist, then we say that the Padé approximant problem is not solvable. All pairs forming solutions are equal up to a nonzero scalar. In our use cases, we usually have the additional constraint that $t(0) = 1$, which ensures uniqueness of the solution. This problem is deeply tied with the Extended Euclidean Algorithm (EEA), as its intermediary results allow to recover a solution if one exists.

Lemma 1. *[GG03, Theorem 5.16] Let $\mathbb{F}$ be a field and $f \in \mathbb{F}[X]$ a polynomial of degree less than $n \in \mathbb{N}$. Let $0 \leq k \leq n$. Let $(r_j, s_j, t_j) \in \mathbb{F}[X]^3$ be the polynomials computed at the jth step of the EEA between X^n and f (notably, $r_j = s_j X^n + t_j f$). Let j_0 be the first index for which $\deg r_{j_0} < k$. Then (r_{j_0}, t_{j_0}) is a solution to the $(k, n-k)$-Padé approximant problem for f if and only if $t_{j_0}(0) \neq 0$. If so, every solution is equal to (r_{j_0}, t_{j_0}) up to a factor in $\mathbb{F}^*$. If not, the $(k, n-k)$-Padé approximant problem for f is not solvable.*

In order to retrieve the polynomials at the correct step of the EEA without actually going through the computation steps of the EEA, it is possible to look at the properties of the subresultants matrices linked to both X^n and f. This technique is used in [MF06] to design a constant-round MPC protocol to compute polynomial GCD, and also in [THO24] to compute a rational reconstruction of a known degree. Note that the following definition is specific to the Padé approximant problem (i.e. one of the polynomial is X^n) and that our definition gets rids of any dependence in the degree of f. Indeed, in MPC setting, one cannot assume that the actual degree of f is known. Nevertheless, the required properties still holds for our definition.

Definition 4. Let $\mathbb{F}$ be a field and $f = \sum_{j=0}^{n-1} f_j X^j \in \mathbb{F}[X]$ a polynomial of degree less than $n \in \mathbb{N}$. The square $n-i$ Toeplitz matrix T_i and the rectangular $(n-i-1) \times (n-i)$ matrix U_i associated to the subresultant of order i between f and X^n are the following matrices, where $f_j = 0$ for any $j < 0$:

$$T_i = \begin{pmatrix} f_i & f_{i+1} & \cdots & f_{n-1} \\ f_{i-1} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ f_{2i-n+1} & \cdots & \cdots & f_i \end{pmatrix}, \; U_i = \begin{pmatrix} f_{n-1} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ f_{i+1} & \cdots & f_{n-1} & 0 \end{pmatrix} \tag{5}$$

The subresultant matrix S_i between X^n and f for $0 \leq i \leq n-1$ is the $2n-2i-1$ square matrix defined as the following matrix:

$$S_i = \left(\begin{array}{c|c} I_{n-i-1} & U_i \\ \hline 0 & T_i \end{array} \right). \tag{6}$$

One can note that S_i has the same determinant as the smaller matrix T_i, which allows to reduce matrix dimension for determinant computation.

The following lemma links theses subresultant matrices with entries of the EEA according to the degree. Notably, it is possible to invert S_i (if possible) in order to obtain the Bézout coefficients corresponding to the step of the EEA where the remainder is of degree i.

Lemma 2. *[GG03, Corollary 6.49] Let f and S_i $(0 \leq i \leq n-1)$ as in Def. 4. Then $\det(T_i) = \det(S_i) \neq 0$ if and only if i appears in the degree sequence of the EEA between X^n and f. If so, let $x = S_i^{-1}[0, \ldots, 1]^\top$, and note $x = [a_{n-i-2}, \ldots, a_0 | b_{n-i-1}, \ldots, b_0]^\top$, $s = \sum_{l=0}^{n-i-2} a_l X^l$, $t = \sum_{l=0}^{n-i-1} b_l X^l$. Then s and t are the coefficients corresponding to the step of the EEA of degree i.*

Combining lemmas 1 and 2 allows to compute a solution to a Padé approximant problem relying only on linear algebra. This will be our main approach in section 4.2 to design a constant-round MPC protocol for Padé approximants. The following corollary summarizes the approach.

Corollary 1. *Let f and $(S_i)_{0\leq i\leq n-1}$ as in Definition 4. Let $0 \leq k \leq n$ and let, if it exists:*

$$i_0 = \max\{0 \leq i < k \mid \det(T_i) \neq 0\}.$$

Let the vector $x = S_{i_0}^{-1}[0,\dots,1]^\top$, and note $x = [a_{n-i_0-2},\dots,a_0|b_{n-i_0-1},\dots,b_0]^\top$. Let $s = \sum_{l=0}^{n-i_0-2} a_l X^l$, $t = \sum_{l=0}^{n-i_0-1} b_l X^l$, and $r = sX^n + tf$. Then the $(k, n-k)$-Padé approximant problem for f is solvable if and only if $t(0) \neq 0$. If so, (r,t) is a solution to the Padé approximant problem.

4.2 Protocol for Padé approximant solving

We now present our MPC protocol for solving the Padé approximant problem (see Definition 3) for a shared polynomial $[f]$. As mentioned previously, our protocol always output a shared solution $([\tilde{r}],[\tilde{t}])$ such that $\tilde{t}(0) = 1$. In order to not leak any information about $[f]$, the protocol will set the value of $([\tilde{r}],[\tilde{t}])$ to the default value $(0,1)$ in the case that the Padé approximant problem has no solution. A formal description of the ideal functionality for Padé approximant is given in Fig. 5. Of course, since k and n are public values, this functionality aborts if k is not a non-negative integer smaller than n.

The functionality is the same as $\mathcal{F}_{ABB}$ with one more functionality.

- Upon receiving $(\texttt{Pade}, \texttt{id}, \texttt{id}'_1, \texttt{id}'_2, n, k)$ from all the parties, the functionality fetches $f \in \mathbb{F}_q[X]$ at $\texttt{id}$. It computes a solution (if it exists) to the $(k, n-k)$-Padé approximant for f, $(\tilde{r},\tilde{t}) \in \mathbb{F}_q[X]^2$, such that $\tilde{t}(0) = 1$. If no solution exists, it sets $(\tilde{r},\tilde{t}) = (0,1)$. Then, it stores $\tilde{r}$ at $\texttt{id}'_1$ and $\tilde{t}$ at $\texttt{id}'_2$.

Figure 5: $\mathcal{F}_{ABB+\texttt{Pade}}$

One main difficulty to provide a protocol for this functionality is to guarantee privacy about $[f]$. Notably, the index i_0 of the subresultant matrix S_{i_0} to invert (see Corollary 1) must be kept hidden, and the knowledge of the existence of a solution should not be revealed either. The first successful use of subresultants matrices in an MPC settings is in [MF06] for computing the gcd of two shared polynomials. However, this protocol additionally reveal the degree of the gcd, which in our context would lead to revealing i_0, which, in the case of HQC decryption, would be a problem for a future threshold CCA version of HQC. Let us describe our new techniques in detail to circuvent the latter problem.

Solvable Padé approximant. First, let's consider that the Padé approximant problem has a solution. To invert the correct matrix in a oblivious manner, we rely on the booleans values $[e_i] = [\det(T_i) \neq 0]$ corresponding to whether T_i (and therefore S_i) is singular (see Definition 4). This reduces to using protocol **CharPoly** for determinant computation and protocol **ZeroTest** for zero testing. Then, we can compute from $([e_i])_{0\leq i\leq k-1}$ a shared vector $([c_i])_{0\leq i\leq k-1}$ which is such that $c_i = 1$ if $i = i_0$ and 0 otherwise. This corresponds to a prefix-or of the vector $([e_i])_{0\leq i\leq k-1}$, and can be done securely on a shared vector using **PrefixOR** protocol. Then, in order to solve the proper linear system, we need to hide the size of the system to invert. To this purpose, we define the following $(2n-1)\times(2n-1)$ matrices $\tilde{S}_i$ that are always invertible when S_i are invertible, for $0 \leq i \leq k-1$:

$$\tilde{S}_i = \left(\begin{array}{c|c} S_i & 0 \\ \hline 0 & I_{2i} \end{array}\right). \tag{7}$$

We can compute the shared matrix $[\tilde{S}] = \sum_{i=0}^{k-1}[c_i][\tilde{S}_i]$ which encodes the matrix S_{i_0} in an oblivious way. By Definition 4, coefficients of S_i are either 0, 1, or coefficients of f. Therefore, one can notice that, for $0 \leq i \leq k-1$, having the shared polynomial $[c_i f]$ along with $[c_i]$ is enough to deduce $[c_i \tilde{S}_i]$ without any communication. Thus, we compute $[c_i f]$ for $0 \leq i \leq k-1$ so that parties can deduce the sharing $[\tilde{S}]$ locally.

Of course, by computing $[\tilde{S}]$, the actual value of i_0 stays hidden. Nevertheless, note that, by the discussion above, $\tilde{S} = \tilde{S}_{i_0}$. Therefore, $\tilde{S}$ is invertible, and $\tilde{S}^{-1}$ is a shared block diagonal matrix whose blocks are $S_{i_0}^{-1}$ and I_{2i_0}. Again, note that the size of the blocks of $[\tilde{S}^{-1}]$ are hidden to the players, and the shared matrix $[\tilde{S}^{-1}]$ looks like a regular dense shared matrix. To solve a linear system involving $[\tilde{S}]$, one can easily rely on the protocol **MatInv**.

We know that $S_{i_0}^{-1} \cdot (0, \ldots, 1)^\top = (a|b)^\top$, with a and b being vectors corresponding to the reverse of polynomials (s, t) that are solution of the Padé problem (Corollary 1). We would like to use this fact to compute these coefficients from $[\tilde{S}^{-1}]$. For $0 \leq i \leq k-1$, we denote δ_i as the $2n-1$ column vector that is equal to zero except at index $2n-2i-2$ where it is one, and we compute the vector $[\delta] = \sum_{i=0}^{k-1}[c_i]\delta_i$. Note that $\delta = \delta_{i_0}$ and thus $\tilde{S}^{-1}\delta_{i_0} = (a|b|0_{2i_0})^\top$. However, i_0 is unknown to the parties, so it is not possible to know if a given coefficients of $[x]$ belongs to $[a]$, $[b]$ or is a sharing of zero.

One may remark that transforming $[x]$ to $[x'] = [0_{i_0}|a|0_{i_0}|b]^\top$ (i.e. the vector containing i_0 zeroes, then a, and then the same for b) is sufficient. Indeed, the first $n-1$ coefficients of $[x']$ can be considered by the parties as the shared coefficients of the reverse of $[s]$ (we know that in all cases we always have $\deg s < n-1$) and the last n as the reverse of $[t]$. Both polynomial could be shared as polynomials of degree less than $n-1$ and n, with the highest order coefficients being possibly 0. In order to compute $[x']$ from $[x]$, consider the $(2n-1) \times (2n-1)$ permutation matrices P_i for $0 \leq i \leq k-1$:

$$P_i = \left(\begin{array}{c|c|c|c} 0 & 0 & I_i & 0 \\ \hline I_{n-i-1} & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & I_i \\ \hline 0 & I_{n-i} & 0 & 0 \end{array}\right) \tag{8}$$

Note that $x' = P_{i_0}x$. Moreover, if we denote $[P] = \sum_{i=0}^{k-1}[c_i]P_i$, notice that $[P]$ is computable locally from the shared vector $[c]$, and that $P = P_{i_0}$. Therefore, to obtain $[x']$ (and therefore $[s]$ and $[t]$), we compute it as $[P][x]$, using **Mat2Mul** protocol. This allows us to obtain $[r] = [s]X^n + [t][f]$ with the protocol **Poly2Mul**. In order to obtain the sought shared solution $([\tilde{r}], [\tilde{t}])$ such that $\tilde{t}(0) = 1$, we divide $([\tilde{r}], [\tilde{t}])$ by the coefficient $[t(0)]$ using protocol **Inv**.

Padé approximant with no solution. Let us now consider the behavior of this protocol when the problem has no solution. This can happen in two cases: either $t(0) = 0$ (see Corollary 1), or there is no integer between 0 and $k-1$ in the degree sequence of the EEA between f and X^n, which means that i_0 is not defined. Note that, according to the ideal functionality, parties should not be able to distinguish these two cases from the case in which the problem has a solution.

One can remark that in the first case, inverting $[t(0)]$ using **Inv** would result in parties noticing that $t(0) = 0$. In order to hide the real value of $t(0)$, parties can compute $[\pi] = [t(0) \neq 0]$ and replace $([r], [t])$ by $([\hat{r}], [\hat{t}]) = [\pi] \cdot ([r], [t]) + (1 - [\pi]) \cdot (0, 1)$. Note that if the problem has a solution, then $(\hat{r}, \hat{t}) = (r, t)$ and the protocol runs as described above. If $t(0) = 0$, then $(\hat{r}, \hat{t}) = (0, 1)$, and so using **Inv** does not reveal a zero value. Moreover, we would have $(\tilde{r}, \tilde{t}) = (0, 1)$, which correspond to the desired functionality. This trick is thus sufficient to handle the case $t(0) = 0$.

In the second case, we want to make sure that $\tilde{S}$ is still invertible, so that the protocol does not reveal a singular matrix when using **MatInv**. Indeed, in that case,

the above definition of $\tilde{S}$ implies $\tilde{S} = 0_{2n-1}$. Remark that the value of $[\vee_{i=0}^{k-1} e_i]$ is 0 while in all other cases it is 1. Therefore, it is then sufficient to replace $\tilde{S}$ by $\tilde{S} = \sum_{i=0}^{k-1} c_i \tilde{S}_i + (1 - \vee_{i=0}^{k-1} e_i) \cdot I_{2n-1}$.

Using this new definition of $\tilde{S}$, we either have I_{2n-1} when i_0 is not defined or $\tilde{S} = S_{i_0}$ in all previous cases. This implies that inverting this new $\tilde{S}$ with protocol **MatInv** always work, without leaking any information, and the behaviour remain unchanged for all cases where i_0 is well defined.

Assuming i_0 is not defined, the rest of the protocol behaves the same as in case $t(0) = 0$. Indeed, in this case, $P = 0_{2n-1}$, and from $[x']$, parties will compute the shared polynomials $[s]$ and $[t]$ such that $s = t = 0$. Therefore, the parties will compute $[\tilde{r}]$ and $[\tilde{t}]$ such that $(\tilde{r}, \tilde{t}) = (0, 1)$.

The Protocol **Padé** below gives a step by step description of our protocol. Its communication complexity and security are detailed in theorem 3.

Protocol 8: Padé

Parameters: $n \in \mathbb{N}$, $0 \leq k \leq n$
Input: $[f] \in \mathbb{F}_q[X]$ of degree $< n$
Output: $[\tilde{r}], [\tilde{t}]$, shared as a polynomial of degree $< n$, such that $(\tilde{r}, \tilde{t})$ is a solution for the $(k, n-k)$-Padé approximation problem and $\tilde{t}(0) = 1$.

1 **for** $0 \leq i \leq k-1$ **in parallel do**
2 Parties locally deduce $[T_i]$ using $[f]$ ▷ no com., see Definition 4
3 $[e_i] \leftarrow [\det(T_i)]$ ▷ **CharPoly**
4 Parties compute $[e_i \neq 0]$ ▷ **ZeroTest**
5 **end**
6 $[d_i] \leftarrow [\vee_{j=i}^{k-1}(e_j \neq 0)]$ for $0 \leq i \leq k-1$ ▷ **PrefixOR**
7 $[c_i] \leftarrow [d_i] - [d_{i+1}]$ for $0 \leq i \leq k-1$ ▷ $d_k = 0$, no com.
8 Parties compute $[c_i][f]$ for $0 \leq i \leq k-1$
9 Parties locally deduce $[c_i \tilde{S}_i]$ from $[c_i f]$ and $[c_i]$ for $0 \leq i \leq k-1$ ▷ no com.
10 $[\tilde{S}] \leftarrow (1 - [d_0]) I_{2n-1} + \sum_{i=0}^{k-1} [c_i \tilde{S}_i]$ ▷ no com.
11 Parties compute $[\tilde{S}^{-1}]$ ▷ **MatInv**
12 $[\delta] \leftarrow \sum_{i=0}^{k-1} [c_i] \delta_i$ ▷ $\delta_i = 0^{2n-1}$ plus one at index $2n-2i-2$, no com.
13 $[x] \leftarrow [\tilde{S}^{-1}][\delta]$ ▷ **Mat2Mul**
14 $[P] \leftarrow \sum_{i=0}^{k-1} [c_i] P_i$ ▷ no com., see equation 8
15 $[s|t]^\top \leftarrow [P][x]$ ▷ **Mat2Mul**
16 $[r] \leftarrow [s] X^n + [t][f]$ ▷ **Poly2Mul**
17 $[\pi] \leftarrow [t(0) \neq 0]$ ▷ **ZeroTest**
18 $([\hat{r}], [\hat{t}]) \leftarrow [\pi] \cdot ([r], [t]) + (1 - [\pi]) \cdot (0, 1)$
19 Parties compute $[\hat{t}(0)^{-1}]$ ▷ **Inv**
20 $([\tilde{r}], [\tilde{t}]) \leftarrow ([\hat{t}(0)^{-1}][\hat{r}], [\hat{t}(0)^{-1}][\hat{t}])$

Theorem 3. *Given a shared polynomial $[f] \in \mathbb{F}_q[X]$ of degree less than n, and a public integer k satisfying $0 \leq k \leq n$, the Protocol **Padé** instantiates $\mathcal{F}_{ABB+\texttt{Padé}}$ (see Fig. 5) in the $\mathcal{F}_{ABB}$-hybrid model to compute the $(k, n-k)$-Padé approximant for f, it it exists. It requires at most $\mathcal{O}(kn^{3.5} + kq)$ secure multiplications and $\mathcal{O}(1)$ rounds of multiplications.*

Proof. Let us build a simulator $\mathcal{S}$ that has access to the ideal functionality and simulates to the adversary an execution of the protocol. Because all the called protocols in **Padé** instantiates their corresponding ideal functionality, $\mathcal{S}$ can correctly simulate all the steps of the protocol. Notably, note that by the discussion above, the call to **MatInv** always reveals a random invertible matrix, and the call to **Inv** always reveal a nonzero field element. Since $\mathcal{S}$ perfectly simulates the views of the adversary in a real execution of the protocol, **Padé** instantiates $\mathcal{F}_{ABB+\texttt{Padé}}$ in the $\mathcal{F}_{ABB}$-hybrid model.

Let us detail the communication complexity of the protocol. Step 3 requires k calls to **CharPoly** on matrices of size at most n, requiring $k\mathcal{O}(n^{3.5}) = \mathcal{O}(kn^{3.5})$ secure multiplications. Step 4 requires k calls to **ZeroTest**, requiring $k\mathcal{O}(q) = \mathcal{O}(qk)$ secure

multiplications. Step 6 requires a call to **PrefixOR**, amounting for $\mathcal{O}(\tau k^{1+1/2\tau})$ secure multiplications and $\mathcal{O}(\tau)$ rounds for any constant $\tau > 0$. As this cost is not dominant, we can take $\tau = 1$. Step 8 requires n secure multiplications per index, i.e. nk secure multiplications in total. Step 11 is a call to **MatInv** for $\mathcal{O}(n^2)$ secure multiplications. Step 13 and 15 are two calls to **Mat2Mul** for $\mathcal{O}(n^2)$ secure multiplications. At step 16 parties invoke **Poly2Mul** for $\mathcal{O}(n)$ secure multiplications. Step 17 requires a call to **ZeroTest** amounting to $\mathcal{O}(q)$ multiplications. Step 18 requires $\mathcal{O}(n)$ secure multiplications. **GenInv** at step 19 consists of a constant number of secure multiplications. This leads to the desired communication complexity as $k \leq n$. Since all the called protocols are constant rounds, and they are leveraged in parallel, this yields a constant round protocol. □

5 Distributed Reed-Muller decoding

In this section, we present MPC protocols for decoding shared noisy first-order duplicated Reed-Muller codewords. This type of code is used for the inner code in the final proposition for HQC.PKE. Therefore, these protocols will be used as subprotocols for our shared decryption of an HQC ciphertext.

The decoder of the HQC implementation (as well as the one of [MS77]) uses a fast Hadamard transform in order to find a closest codeword to the input. We first consider the original decoding algorithm by Reed [Ree54], which relies on majority votes and recovers the transmitted codeword as long as the number of errors is less than half the minimum distance of the code. We introduce this algorithm, and then present our novel MPC techniques to design the corresponding MPC protocol, namely a subprotocol to securely compute the majority among a set of shared values. In Section 5.4, we present a second MPC protocol, which securely computes a closest codeword to the input, and therefore exactly matches the decoder of the HQC implementation (and hence its decryption failure rate). We believe that both MPC protocols remains of interest outside the thresholdization of HQC and thus motivate their detailed presentation.

5.1 Plain Reed-Muller decoding

We first start with the decoding algorithm of duplicated Reed-Muller codes, based on the original algorithm by Reed [Ree54]. It decodes each component of the message individually: starting with the components from 1 to m of the original message, then lastly, the one at the index 0. We recall that basic definition and properties on Reed-Muller codes are given in section 2.4.1.

Let $y = (y_1, \ldots, y_\gamma) \in (\mathbb{F}_2^{2^m})^\gamma$ be a noisy codeword of $RM_\gamma(1, m)$, that is $y = c' + e$ with $c' = (c_1, \ldots, c_\gamma) \in RM_\gamma(1, m)$ and $\mathtt{wt}(e) \leq \gamma \cdot 2^{m-2}$. We note $c = c_1 = \cdots = c_\gamma = (w_0 + \sum_{i=1}^m w_i v_i)_{v \in \mathbb{F}_2^m}$. First, the algorithm computes the value w_j for $1 \leq j \leq m$ in the following way: let $\mathbf{1}_j \in \mathbb{F}_2^m$ the zero vector except at index j. Notice the following for $v \in \mathbb{F}_2^m$ and $1 \leq k \leq \gamma$:

$$\begin{aligned}
\delta_{k,v,j} := y_{k,v} + y_{k,v+\mathbf{1}_j} &= c_v + e_{k,v} + c_{v+\mathbf{1}_j} + e_{k,v+\mathbf{1}_j} \\
&= w_0 + \sum_{i=1}^m w_i v_i + e_{k,v} + w_0 + \sum_{i=1}^m w_i v_i + w_j + e_{k,v+\mathbf{1}_j} \\
&= w_j + e_{k,v} + e_{k,v+\mathbf{1}_j}.
\end{aligned}$$

Therefore, if $e_{k,v} + e_{k,v+\mathbf{1}_j} = 0$, then $\delta_{k,v,j}$ gives out the correct value for w_j. Because $\mathtt{wt}(e) \leq \gamma \cdot 2^{m-2}$, then a majority of these values will actually correspond to w_j. Indeed:

$$\begin{aligned}\left|\{(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m \mid e_{k,v} + e_{k,v+\mathbf{1}_j} = 1\}\right| &\leq \left|\{(k,v) \mid e_{k,v} = 1 \vee e_{k,v+\mathbf{1}_j} = 1\}\right| \\ &\leq 2 \cdot \mathtt{wt}(e) \leq \gamma \cdot 2^{m-1}\end{aligned}$$

Thus if more than $\gamma \cdot 2^{m-1}$ values are equal to 1 among all the $\gamma \cdot 2^m$ values $\delta_{k,v,j}$, then $w_j = 1$. Otherwise, $w_j = 0$. Once the values $w_1, \ldots, w_m$ are found, the value w_0 can be found using a similar approach. Indeed, $v \in \mathbb{F}_2^m$ and $1 \leq k \leq \gamma$, we have that $\delta_{k,v,0} := y_{k,v} + \sum_{i=1}^m w_i v_i = w_0 + e_{k,v}$. Since $\mathtt{wt}(e) \leq \gamma \cdot 2^{m-2}$, then there are a minority of incorrect values for w_0 among all the values $\delta_{k,v,0}$. Therefore, computing the majority can be used again to compute w_0. Algorithm 9 gives a detailed description of the decoding algorithm.

Algorithm 9: Duplicated Reed-Muller Decoding

Parameters: $m, \gamma \in \mathbb{N}$
Input: $y = (y_1, \ldots, y_\gamma) \in (\mathbb{F}_2^{2^m})^\gamma$ such that $y = c' + e$ with $c' = (c_1, \ldots, c_\gamma) \in RM_\gamma(1,m)$ and $\mathtt{wt}(e) \leq \gamma \cdot 2^{m-2}$.
Output: $w \in \mathbb{F}_2^{m+1}$ such that $c_1 = \cdots = c_\gamma = (\sum_{i=0}^m w_i v_i)_{v \in \mathbb{F}_2^m}$

1 **for** j **from** 1 **to** m **do**
2 **for** $(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m$ **do**
3 Compute $\delta_{k,v,j} := y_{k,v} + y_{k,v+\mathbf{1}_j}$
4 **end**
5 Set w_j to the value that appears at least $\gamma \cdot 2^{m-1}$ times in $(\delta_{k,v,j})_{(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m}$
6 **end**
7 **for** $(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m$ **do**
8 Compute $\delta_{k,v,0} := y_{k,v} + \sum_{i=1}^m w_i v_i$
9 **end**
10 Set w_0 to the value that appears at least $\gamma \cdot 2^{m-1}$ times in $(\delta_{k,v,0})_{(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m}$
11 Return $w = (w_0, \ldots, w_m)$

In the MPC setting, the addition steps (steps 3 and 8) are straightforward to adapt. However, the steps for determining the majority value (steps 5 and 10) are not. Therefore, we first present a protocol for this task, and then use it as a building block of our decoding protocol for Reed-Muller codewords.

5.2 Majority computation

In this section, we present an MPC protocol for majority computation. Given N (we'll consider N to be odd) shared booleans $[a_1], \ldots, [a_N] \in \mathbb{F}_q$ (i.e. the shared elements lie in $\mathbb{F}_2$ but their shares lie in $\mathbb{F}_q$), we want to compute $[M]$ where $M = 0$ if there is a majority of 0 and 1 otherwise. To design an MPC protocol for this task, let us do the following observations: let $A_0 = \{1 \leq i \leq N \mid a_i = 0\}$, and notice that $M = 0$ if and only if $|A_0| > \lfloor N/2 \rfloor$. Next, let g be the polynomial $g = \prod_{i=1}^N (X - a_i) = \sum_{i=0}^N g_i X^i \in \mathbb{F}_q[X]$. Notice that $g = X^{|A_0|}(X-1)^{N-|A_0|}$, and that the coefficients of g lie in $\mathbb{F}_2$. We can therefore check the value of $|A_0|$ by computing the multiplicity of 0 in g. This can be done by checking the coefficients of lowest order of g; if $g_0 = g_1 = \cdots = g_{\lfloor N/2 \rfloor} = 0$ then $|A_0| > \lfloor N/2 \rfloor$ and therefore $M = 0$. Otherwise, $M = 1$. Thus, we notice that $M = \bigvee_{i=0}^{\lfloor N/2 \rfloor} g_i$. Since $g \in \mathbb{F}_2[X]$, this expression can be written in terms of products: $M = 1 - \prod_{i=0}^{\lfloor N/2 \rfloor} (1 - g_i)$

Because the protocol does not check if the input values are actually in $\mathbb{F}_2$, we describe in Fig. 6 the desired functionality in terms of the algebraic operations as described above. When the input values are actually in $\mathbb{F}_2$, these algebraic operations indeed correspond to computing the majority. For the description of the protocol itself, we will consider that it is only used on shared inputs whose values lie in $\mathbb{F}_2$.

The functionality is the same as $\mathcal{F}_{ABB}$ with one more functionality.

- Upon receiving $(\texttt{Majority}, (\texttt{id}_i)_{1\leq i\leq N}, \texttt{id}')$ from all the parties, the functionality fetches $(a_i, \texttt{id}_i)_{1\leq i\leq N}$, computes $g = \prod_{i=1}^{N}(X - a_i) = \sum_{i=0}^{N} g_i X^i$, and then computes $M = 1 - \prod_{i=0}^{\lfloor N/2\rfloor}(1 - g_i)$. Finally, it stores $(M, \texttt{id}')$.

Figure 6: $\mathcal{F}_{ABB+\texttt{Majority}}$

To instantiate this functionality, our MPC protocol proceeds as follows: first, using protocol **PolyFromRoots**, we compute the shared polynomial $[g]$. Then, the protocol needs to compute $[M]$ itself. As stated above, this operation can be computed as an unbounded multiplication of potentially zero field elements. As the protocol for invertible elements of [BIB89] is not applicable to zero field elements, the authors of [CD01, GLOV24] proposed an idea to compute this operation as the constant coefficients of a polynomial. In our case, this means computing the shared polynomial $[h] = \left[\prod_{i=0}^{\lfloor N/2\rfloor}(X - 1 + g_i)\right]$. Again, this can be done using **PolyFromRoots**. Taking the constant coefficient of $[h]$ (up to sign) allows to obtain the product $\left[\prod_{i=0}^{\lfloor N/2\rfloor}(1 - g_i)\right]$, and therefore obtain $[M]$. Overall, this leads to a MPC protocol requiring a number of secure multiplications in $\mathbb{F}_q$ which is sub-quadratic in N (i.e. in $\mathcal{O}(\tau N^{1+1/\tau})$), and a number of rounds of secure multiplication that is proportional to τ (τ being any constant).

One thing to note here: the value τ is independent of the size of the shared input, and only serves in **PolyFromRoots** to parameterize the trade-off between the number of secure multiplications and rounds. As such, this protocol and **Majority** have a number of rounds independent of the size of the inputs and are constant-round protocols.

The protocol is described in detail in protocol **Majority**. Theorem 4 states the security and the communication complexity of the protocol.

Protocol 10: Majority

Parameters: $N \in \mathbb{N}$ odd, $l = \lceil \log(N) \rceil$, $l' = \lceil \log(\lfloor N/2 \rfloor + 1) \rceil$
Input: $[a_1], \ldots, [a_N] \in \mathbb{F}_q^N$ such that each value is either 0 or 1
Output: $[M]$ such that $M = 0$ if there is a majority of 0 in $a_1, \ldots, a_N$, 1 otherwise

1 $\left[g = \sum_{i=0}^{N} g_i X^i\right] \leftarrow [\prod_{i=1}^{N}(X - a_i)]$ ▷ **PolyFromRoots**
2 $\left[h = \sum_{i=0}^{\lfloor N/2\rfloor+1} h_i X^i\right] \leftarrow [\prod_{i=0}^{\lfloor N/2\rfloor}(X - 1 + g_i)]$ ▷ **PolyFromRoots**
3 $[M] \leftarrow 1 - (-1)^{\lfloor N/2\rfloor+1}[h_0]$ ▷ no com.

Theorem 4. *Given $[a_1], \ldots, [a_N] \in \mathbb{F}_q$, where a_i's value lie in $\mathbb{F}_2$, and a integer $\tau \in \mathbb{N}^*$ (independent of N), the Protocol **Majority** instantiates $\mathcal{F}_{ABB+\texttt{Majority}}$ (see Fig. 6) in the $\mathcal{F}_{ABB}$-hybrid model, and requires at most $\mathcal{O}(\tau N^{1+1/\tau})$ secure multiplications and $\mathcal{O}(\tau)$ rounds of multiplications.*

Proof. By the discussion above, this protocol instantiates functionality $\mathcal{F}_{ABB+\texttt{Majority}}$ in the $\mathcal{F}_{ABB+\texttt{MultPoly}}$-hybrid model. Moreover, protocol **PolyFromRoots** instantiates functionality $\mathcal{F}_{ABB+\texttt{Majority}}$ in the $\mathcal{F}_{ABB}$-hybrid model. This means that **Majority** indeed instantiates $\mathcal{F}_{ABB+\texttt{Majority}}$ (see Fig. 6) in the $\mathcal{F}_{ABB}$-hybrid model. Step 1 requires at most $\mathcal{O}(\tau N^{1+1/\tau})$ secure multiplications and $\mathcal{O}(\tau)$ rounds of multiplications. Step 2 has the same asymptotic communication complexity. Step 3 does not require any communication. This leads to the desired communication complexity. □

The functionality is the same as $\mathcal{F}_{ABB}$ with one more functionality.

- Upon receiving $(\texttt{ReedMuller}, \texttt{id}, \texttt{id}')$ from all the parties, the functionality fetches $(y_{k,v})_{(k,v)\in[\![1,\gamma]\!]\times\mathbb{F}_2^m}$ at $\texttt{id}$. If at least one of the coefficient is not in $\mathbb{F}_2$, then the functionality computes $y_{k,v}^2 - y_{k,v}$ for all $(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m$ and sends all these values to all the parties, and then aborts. Otherwise, it runs algorithm 9 on $y = (y_{k,v})_{(k,v)\in[\![1,\gamma]\!]\times\mathbb{F}_2^m}$. The algorithm outputs $w = (w_j)_{0\le j\le m} \in \mathbb{F}_q^{m+1}$ and the functionality stores w at $\texttt{id}'$.

Figure 7: $\mathcal{F}_{ABB+\texttt{ReedMuller}}$

5.3 Our Reed decoding protocol for Reed-Muller codes

We now present the MPC protocol for duplicated Reed-Muller decoding, using Algorithm 9 as a baseline. The corresponding ideal functionality $\mathcal{F}_{ABB+\texttt{ReedMuller}}$ is presented in Fig. 7. While the addition steps are straightforward, the majority computation steps are handled using protocol **Majority** as a subroutine. This leads to protocol **ReedMuller-R** which requires a number of secure multiplications that is sub-quadratic in the length of the Reed-Muller code (i. e. $\gamma 2^m$). This protocol induces a relatively small overhead compared to the number of operations for decoding in the clear. The security and exact complexity of the protocol are stated in Theorem 5.

As the MPC protocols are designed for shared elements of $\mathbb{F}_q$, the functionality might be asked to perform the decoding procedure on elements of $\mathbb{F}_q$ that are actually not elements in $\mathbb{F}_2$. Therefore, the MPC protocol tests if the input values are in $\mathbb{F}_2$, and abort if it is not the case, and this is reflected in the ideal functionality of Fig. 7.

Protocol 11: ReedMuller-R

Parameters: $m, \gamma \in \mathbb{N}^2$ such that γ is odd, $\tau \in \mathbb{N}^*$ independent of m and γ
Input: $[y] = ([y_1], \dots, [y_\gamma]) \in (\mathbb{F}_q^{2^m})^\gamma$
Output: $[w] \in \mathbb{F}_q^{m+1}$ being the result of algorithm 9 if $y \in (\mathbb{F}_2^{2^m})^\gamma$, otherwise abort

for $(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m$ **in parallel do**
  $[z_{k,v}] \leftarrow [y_{k,v}][y_{k,v}] - [y_{k,v}]$
  Parties reveal the value $z_{k,v}$, and abort if $z_{k,v} \neq 0$
end
for j **from** 1 **to** m **in parallel do**
  for $(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m$ **do**
    $[\delta_{k,v,j}] \leftarrow [y_{k,v}] + [y_{k,v+\mathbf{1}_j}]$ ▷ no com.
  end
  $[w_j] \leftarrow \text{Majority}(([\delta_{k,v,j}])_{(k,v)\in[\![1,\gamma]\!]\times\mathbb{F}_2^m})$ ▷ **Majority**
end
for $(k,v) \in [\![1,\gamma]\!] \times \mathbb{F}_2^m$ **do**
  $[\delta_{k,v,0}] \leftarrow [y_{k,v}] + \sum_{i=1}^m [w_i] v_i$ ▷ no com.
end
$[w_0] \leftarrow \text{Majority}(([\delta_{k,v,0}])_{(k,v)\in[\![1,\gamma]\!]\times\mathbb{F}_2^m})$ ▷ **Majority**
$[w] \leftarrow ([w_0], \dots, [w_m])$ ▷ no com.

Theorem 5. *Let $m, \gamma \in \mathbb{N}^*$ be the parameters of a concatenated Reed-Muller code $RM_\gamma(1,m)$ (as in Definition 1) and let $\tau \ge 1$ be any integer independent of m and γ. Given a shared vector $[y] \in (\mathbb{F}_q^{2^m})^\gamma$ such that $y = c + e$ where $c \in RM_\gamma(1,m)$. and $\texttt{wt}(\texttt{e}) \le \gamma \cdot 2^{\texttt{m}-2}$. The Protocol **ReedMuller-R** instantiates functionality $\mathcal{F}_{ABB+\texttt{ReedMuller}}$ (see Fig. 7) in the $\mathcal{F}_{ABB}$-hybrid model to get the shared value over $\mathbb{F}_q$ of the decoding of $c + e$. Moreover, it requires at most $\mathcal{O}(m\tau(\gamma 2^m)^{1+1/\tau})$ secure multiplications and $\mathcal{O}(\tau)$ rounds of multiplications.*

Proof. Let us build a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{ABB+\texttt{ReedMuller}}$ and the adversary, that emulates honest parties and tries to convince the adversary that it is running the protocol. First, $\mathcal{S}$ asks the ideal functionality. It receives either nothing (meaning that the functionality correctly ran algorithm 9 on the shared inputs), or the leakage $(y_{k,v}^2 - y_{k,v})_{(k,v)\in[\![1,\gamma]\!]\times\mathbb{F}_2^m}$ and an abort signal. Then, $\mathcal{S}$ runs the simulators for multiplication and addition to simulate steps 1 to 4 of the protocol. If $\mathcal{S}$ received a leakage and an abort signal from the ideal functionality, then $\mathcal{S}$ forwards these to the adversary, as it would have received in the protocol when revealing shares. Otherwise, $\mathcal{S}$ emulates the revealing of shares of value 0 to the adversary, and proceeds to runs multiple instances of the simulator corresponding to protocol **Majority** (it exists because of Theorem 4) in order to simulate the views of the adversary in the rest of the protocol. Therefore, protocol **ReedMuller-R** instantiates functionality $\mathcal{F}_{ABB+\texttt{ReedMuller}}$ (see Fig. 7) in the $\mathcal{F}_{ABB}$-hybrid model.

The only steps that require secure multiplications in protocol **ReedMuller-R** are steps 2, 9, and 14. Step 2 requires one round of communication and $\gamma 2^m$ secure multiplications. Step 9 consists of m parallel calls of protocol **Majority** on $\gamma 2^m$ elements each. Step 14 consists of one call of the same protocol on the same number of elements. This leads to the desired communication complexity. □

5.4 Closest codeword decoding

Algorithm 9 is guaranteed to recover the transmitted codeword as long as there are fewer than $\gamma 2^{m-2}$ errors. The decoder of the HQC specification [AAB+22] instead outputs a closest codeword to its input, which may succeed well beyond this bound, and the decryption failure rate analysis of HQC relies on this decoder. In this section, we present an MPC protocol that computes exactly such a closest codeword, in a constant number of rounds (independent of m and γ). Since Hamming distances are integers, they cannot be directly computed on values shared in a field of characteristic 2. Instead, we compute the distance between the input and each candidate codeword *in the exponent* of a public field element ρ, using products of non-zero elements. Two distances are then compared by evaluating a public bivariate polynomial on the corresponding powers of ρ, which amounts to a single inner product and directly outputs the result of the comparison in the exponent. Finally, the position of the minimum is obtained with an unbounded `AND` of these results, computed as a product.

Decoding function. We let $N = \gamma 2^m$, and we denote $u \cdot v = \sum_{i=1}^m u_i v_i$ for $u, v \in \mathbb{F}_2^m$. The codeword of $RM_\gamma(1, m)$ corresponding to a message $w = (w_0, u) \in \mathbb{F}_2 \times \mathbb{F}_2^m$ is $c = (c_v)_{v\in\mathbb{F}_2^m}$ duplicated γ times, where $c_v = w_0 + u \cdot v$. For $y \in \mathbb{F}_2^N$ and $w \in \mathbb{F}_2^{m+1}$, we denote

$$D_w = \#\{(k, v) \in \{1, \ldots, \gamma\} \times \mathbb{F}_2^m \mid y_{k,v} \neq c_v\}$$

the distance between y and the codeword c associated to the message w. Note that $D_{(1,u)} = N - D_{(0,u)}$.

To obtain a uniquely defined output, ties are broken with a public total order $\prec$ on $\mathbb{F}_2^{m+1}$. For $y \in \mathbb{F}_2^N$, we denote by $\mathrm{Dec}_\prec(y)$ the smallest $w \in \mathbb{F}_2^{m+1}$ according to $\prec$ such that $D_w = \min_{w'\in\mathbb{F}_2^{m+1}} D_{w'}$. For HQC, we order the candidates according to u, following the order in which the reference implementation examines them. The order between $(0, u)$ and $(1, u)$ is irrelevant since, for odd γ, these their associated codewords are never both at minimal distance from y (indeed, let $F(u) = N - 2D_{(0,u)} = \sum_v \sigma_v (-1)^{u\cdot v}$ with $\sigma_v = \sum_k (-1)^{y_{k,v}}$; every σ_v is odd, so Parseval's identity $\sum_u F(u)^2 = 2^m \sum_v \sigma_v^2 > 0$ gives $\min_w D_w < N/2$, whereas $D_{(0,u)} = D_{(1,u)}$ implies $D_{(0,u)} = N/2$).

Distances in the exponent. Let $\rho \in \mathbb{F}_q$ be an element of multiplicative order larger than $\max(N, 2^{m+1}-1)$ (i.e. larger than N as soon as $\gamma \geq 2$), so that the elements $\rho^0, \ldots, \rho^{\max(N,2^{m+1}-1)}$ are pairwise distinct, and let $z = (\rho+1)^{-1}$, so that $\rho = (z+1)/z$ and $z \notin \{0,1\}$. For $v \in \mathbb{F}_2^m$ and $w \in \mathbb{F}_2^{m+1}$, we have $c_v = w_0 + \sum_{i=1}^m w_i v_i \in \mathbb{F}_2$, and we define

$$P_v = \prod_{k=1}^{\gamma} (z + y_{k,v}) \in \mathbb{F}_q, \qquad Q_v = \prod_{k=1}^{\gamma} (z + 1 + y_{k,v}) \in \mathbb{F}_q,$$

$$\theta_w = z^{-N} \prod_{v \in \mathbb{F}_2^m} R_{w,v} \in \mathbb{F}_q, \quad \text{where } R_{w,v} = Q_v \text{ if } c_v = 1 \text{ and } R_{w,v} = P_v \text{ otherwise.}$$

When $y \in \mathbb{F}_2^N$, the elements P_v, Q_v and θ_w are non-zero since $z \notin \{0,1\}$. We have $z + y_{k,v} + c_v = z$ if $y_{k,v} = c_v$ and $z + y_{k,v} + c_v = z + 1$ otherwise. Therefore

$$\prod_v R_{w,v} = \prod_{(k,v)} (z + y_{k,v} + c_v) = z^{N - D_w} (z+1)^{D_w} = z^N \rho^{D_w},$$

and thus $\theta_w = \rho^{D_w}$ encodes the distance D_w in the exponent.

Comparisons with a public polynomial. Let $H_{\leq}, H_{<} \in \mathbb{F}_q[X,Y]$ be the unique polynomials of degree at most N in each variable such that, for $0 \leq a, b \leq N$,

$$H_{\leq}(\rho^a, \rho^b) = \begin{cases} 1 & \text{if } a \leq b, \\ \rho & \text{otherwise,} \end{cases} \qquad H_{<}(\rho^a, \rho^b) = \begin{cases} 1 & \text{if } a < b, \\ \rho & \text{otherwise} \end{cases}.$$

They can be computed publicly by Lagrange interpolation in each variable.

Given two distinct message w and w', we say that w *beats* w' if and only if $(D_w < D_{w'} \wedge w' \prec w)$ or $(D_w \leq D_{w'} \wedge w \prec w')$. Therefore, if we denote $\varphi_{w,w'} = H(\theta_w, \theta_{w'})$ where H is equal to $H_{\leq}$ if $w \prec w'$ and to $H_{<}$ otherwise, we have $\varphi_{w,w'} = 1$ if w beats w', and $\varphi_{w,w'} = \rho$ otherwise.

Let $H_{\leq} = \sum_{s,t=0}^N \eta_{s,t}^{\leq} X^s Y^t$ and $H_{<} = \sum_{s,t=0}^N \eta_{s,t}^{<} X^s Y^t$, whose coefficients are public. We have

$$\varphi_{w,w'} = H(\theta_w, \theta_{w'}) = \sum_{s=0}^N \theta_w^s \, \zeta_{w',s}, \quad \text{with } \zeta_{w',s} = \sum_{t=0}^N \eta_{s,t} \, \theta_{w'}^t,$$

where $\eta_{s,t} = \eta_{s,t}^{\leq}$ if $w \prec w'$ and $\eta_{s,t} = \eta_{s,t}^{<}$ otherwise. Once the powers θ_w^t and $\theta_{w'}^t$ (for $0 \leq t \leq N$) are known, the values $\zeta_{w',s}$ are linear combinations with public coefficients of the powers of $\theta_{w'}$, and each comparison $\varphi_{w,w'}$ is an inner product of length $N+1$.

Selecting the minimum. The candidate $\mathrm{Dec}_{\prec}(y)$ is the only one that beats all the others. Since $\varphi_{w,w'} \in \{1, \rho\}$, for a given message w, the product $U_w = \prod_{w' \neq w} \varphi_{w,w'}$ is equal to ρ^ℓ, where ℓ is the number of candidates that w does not beat. To identify $\mathrm{Dec}_{\prec}(y)$, one needs to find the unique w for which this product is equal to 1 (i.e. with $\ell = 0$).

Let $L' = \sum_{t=0}^{2^{m+1}-1} \Lambda'_t Y^t$ be the polynomial of degree less than 2^{m+1} which is equal to 1 on ρ^0 and vanishes on $\rho^1, \ldots, \rho^{2^{m+1}-1}$. If we define $R_w = L'(U_w)$, then $R_w = 1$ if $w = \mathrm{Dec}_{\prec}(y)$ and 0 otherwise.

Our protocol. The corresponding ideal functionality $\mathcal{F}_{ABB+\texttt{ReedMullerML}}$ is the same as $\mathcal{F}_{ABB+\texttt{ReedMuller}}$ (see Fig. 7), parameterized by the order $\prec$, except that it computes $w = \mathrm{Dec}_{\prec}(y)$ instead of running Algorithm 9.

The resulting protocol is given in protocol **ReedMuller-ML**. The products defining P_v, Q_v, $\theta_{(0,u)}$ and U_w are computed with **Products** (all their factors are non-zero, so

these calls do not leak any information). To improve the efficiency, we remark that, since $D_{(1,u)} = N - D_{(0,u)}$ for any $u \in \mathbb{F}_2^m$, we have $\theta_{(1,u)} = \rho^N \theta_{(0,u)}^{-1}$ and $\theta_{(1,u)}^t = \rho^{Nt}\theta_{(0,u)}^{-t} = \rho^{Nt}\theta_{(0,u)}^{-N}\theta_{(0,u)}^{N-t}$ for all exponents $t \in \{0, \dots, N\}$, which only requires inverting $\theta_{(0,u)}^N \in \mathbb{F}_q^*$ with **Inv** and $N-1$ multiplications.

Protocol 12: ReedMuller-ML

Parameters: $m, \gamma \in \mathbb{N}$, $N = \gamma 2^m$, $\prec$, ρ, z, $(\eta_{s,t}^{\leq})$, $(\eta_{s,t}^{<})$ and (Λ'_t) as in Section 5.4
Input: $[y] = ([y_1], \dots, [y_\gamma]) \in (\mathbb{F}_q^{2^m})^\gamma$
Output: $[w] \in \mathbb{F}_q^{m+1}$ such that $w = \mathrm{Dec}_\prec(y)$ if $y \in (\mathbb{F}_2^{2^m})^\gamma$, otherwise abort

Parties check that $y \in (\mathbb{F}_2^{2^m})^\gamma$ and abort otherwise ▷ as in **ReedMuller-R**
for $v \in \mathbb{F}_2^m$ **in parallel do**
  $[P_v] \leftarrow \prod_{k=1}^{\gamma}(z + [y_{k,v}])$ and $[Q_v] \leftarrow \prod_{k=1}^{\gamma}(z + 1 + [y_{k,v}])$ ▷ **Products**
end
for $u \in \mathbb{F}_2^m$ **in parallel do**
  $[\theta_{(0,u)}] \leftarrow z^{-N} \prod_{v \in \mathbb{F}_2^m} [R_{(0,u),v}]$ ▷ **Products**
  Parties compute $[\theta_{(0,u)}^2], \dots, [\theta_{(0,u)}^N]$ ▷ **PowersNZ**
  Parties compute $[\theta_{(0,u)}^{-N}]$ from $[\theta_{(0,u)}^N]$ ▷ **Inv**
  $[\theta_{(0,u)}^{-t}] \leftarrow [\theta_{(0,u)}^{-N}][\theta_{(0,u)}^{N-t}]$ for $1 \leq t < N$
  $[\theta_{(1,u)}^t] \leftarrow \rho^{Nt}[\theta_{(0,u)}^{-t}]$ for $0 \leq t \leq N$ ▷ no com.
end
for $w \in \mathbb{F}_2^{m+1}$ **in parallel do**
  $[\zeta_{w,s}^{\leq}] \leftarrow \sum_{t=0}^{N} \eta_{s,t}^{\leq}[\theta_w^t]$ and $[\zeta_{w,s}^{<}] \leftarrow \sum_{t=0}^{N} \eta_{s,t}^{<}[\theta_w^t]$ for $0 \leq s \leq N$ ▷ no com.
end
for $w, w' \in \mathbb{F}_2^{m+1}$ *such that* $w \neq w'$ **in parallel do**
  $[\varphi_{w,w'}] \leftarrow \sum_{s=0}^{N} [\theta_w^s][\zeta_{w',s}]$, with $\zeta = \zeta^{\leq}$ if $w \prec w'$ and $\zeta = \zeta^{<}$ otherwise ▷ inner product
end
for $w \in \mathbb{F}_2^{m+1}$ **in parallel do**
  $[U_w] \leftarrow \prod_{w' \neq w} [\varphi_{w,w'}]$ ▷ **Products**
  Parties compute $[U_w^2], \dots, [U_w^{2^{m+1}-1}]$ ▷ **PowersNZ**
  $[R_w] \leftarrow \sum_{t=0}^{2^{m+1}-1} \Lambda'_t [U_w^t]$ ▷ no com.
end
$[w] \leftarrow \sum_{w \in \mathbb{F}_2^{m+1}} [R_w]\, c$ ▷ no com.

Theorem 6. *Let $m, \gamma \in \mathbb{N}^*$ be the parameters of a duplicated Reed-Muller code $RM_\gamma(1,m)$ (as in Definition 1). Given a shared vector $[y] \in (\mathbb{F}_q^{2^m})^\gamma$ such that $y = c + e$ where $c \in RM_\gamma(1,m)$. The Protocol **ReedMuller-ML** instantiates $\mathcal{F}_{ABB+\texttt{ReedMullerML}}$ in the $\mathcal{F}_{ABB}$-hybrid model to get the shared value over $\mathbb{F}_q$ of the decoding of the closest codeword of y. It requires $\mathcal{O}(\gamma 4^m)$ secure multiplications and $\mathcal{O}(1)$ rounds of multiplications. γ.*

Proof. The correctness of the protocol follows from the discussion above.

Security. Let us build a simulator $\mathcal{S}$ that interacts with the adversary and the ideal functionality $\mathcal{F}_{ABB+\texttt{ReedMullerML}}$. First, $\mathcal{S}$ calls the ideal functionality. It receives either nothing, or the leakage $(y_{k,v}^2 - y_{k,v})_{(k,v)}$ and an abort signal. As in the proof of Theorem 5, $\mathcal{S}$ runs the simulator of multiplication for the input check, and then either forwards the leakage and aborts, or simulates the revealing of zeroes. In the latter case, $y \in \mathbb{F}_2^N$, and the rest of the protocol only consists of local operations, multiplications and inner products of $\mathcal{F}_{ABB}$, and calls to **Products**, **PowersNZ** and **Inv**. All the calls to **Products** are made on non-zero elements: the factors $z + y_{k,v}$ and $z + 1 + y_{k,v}$ lie in $\{z, z+1\}$, the factors $R_{(0,u),v}$ lie in $\{P_v, Q_v\}$, which are non-zero, and the factors $\varphi_{w,w'}$ lie in $\{1, \rho\}$. Similarly, **Inv** is only called on $\theta_{(0,u)}^N = \rho^{N D_{(0,u)}} \neq 0$. Hence, none of these calls leaks any information, and $\mathcal{S}$ simulates them by running their respective simulators (using fresh

randomness for each call to **Products**). The views of the adversary are thus perfectly simulated, independently of y (in particular, independently of the number of errors in y). Therefore, protocol **ReedMuller-ML** instantiates $\mathcal{F}_{ABB+\texttt{ReedMullerML}}$ in the $\mathcal{F}_{ABB}$-hybrid model.

Complexity. The input check requires $\gamma 2^m$ secure multiplications. The computation of $[P_v]$ and $[Q_v]$ consists of 2^{m+1} calls to **Products** on γ elements, i.e. $\mathcal{O}(\gamma 2^m)$ secure multiplications. For each $u \in \mathbb{F}_2^m$, the computation of $[\theta_{(0,u)}]$ requires one call to **Products** on 2^m elements, its positive powers one call to **PowersNZ** up to the power N, and its negative powers one call to **Inv** and $N-1$ secure multiplications, i.e. $\mathcal{O}(N) = \mathcal{O}(\gamma 2^m)$ secure multiplications, and $\mathcal{O}(\gamma 4^m)$ in total. The computation of the values $[\zeta^{\leq}_{w,s}]$ and $[\zeta^{<}_{w,s}]$ is local. The comparisons require $2^{m+1}(2^{m+1}-1)$ inner products, which have the same cost as $\mathcal{O}(4^m)$ secure multiplications. For each $w \in \mathbb{F}_2^{m+1}$, the computation of $[U_w]$ requires one call to **Products** on $2^{m+1}-1$ elements and one call to **PowersNZ** up to the power $2^{m+1}-1$, i.e. $\mathcal{O}(4^m)$ secure multiplications in total. The remaining steps are local. This amounts to $\mathcal{O}(\gamma 4^m)$ secure multiplications. Regarding the number of rounds, the random elements used in **Products**, **PowersNZ** and **Inv** (and the products between them) do not depend on the input, and are computed in parallel with the first three rounds of the protocol. Then, the input check, the computation of $[P_v]$ and $[Q_v]$, of $[\theta_{(0,u)}]$, of its positive powers, of $[\theta^{-N}_{(0,u)}]$ and of $[U_w]$ and its powers each require one round of multiplications followed by one round of openings, while the computation of the negative powers and of the comparisons each require one round of multiplications. Since the input check can be run in parallel with the computation of $[P_v]$ and $[Q_v]$, the protocol requires a constant number of rounds (i.e. 9 rounds of multiplications and 7 round of openings). □

6 Distributed Reed-Solomon decoding

In this section, we present a protocol for decoding a noisy codeword of a publicly known Reed-Solomon code. Our MPC protocol does not leak any information about the codeword or the error. As in Section 5, this protocol will also serve for shared decryption of HQC ciphertext. We first give a decoding algorithm for Reed-Solomon codes that serves as a baseline for our MPC protocol. Then, we present the decoding protocol itself as well as its corresponding functionality.

6.1 Plain Reed-Solomon decoding

The decoding algorithm presented here follows a similar principle as the one presented in the specification of the HQC cryptosystem [AAB+22]. As most decoding algorithms for Reed-Solomon codes, this algorithm first computes the error polynomials which, notably, encodes the indices at which the error vector is non-zero. In particular, this can be done by solving a Padé approximant. Once the error locator is known, we use similar techniques as in Forney's algorithm [For65] to find the values of the error vector.

Let us consider a Reed-Solomon code $\mathcal{C}$ (as in Definition 2), and a noisy codeword $y = c + e \in \mathbb{F}_{q'}[X]/(X^{n'}-1)$, such that $c \in \mathcal{C}$ and $\texttt{wt}(e) \leq \delta$. Let us denote by $M = \{0 \leq k \leq n'-1 \mid e_k \neq 0\}$ the indices at which the error symbols are non-zero.

The syndrome polynomial associated to y is the polynomial $S = \sum_{i=0}^{2\delta-1} s_{i+1} X^i$ where $s_i = y(\alpha^i) = e(\alpha^i)$ for $1 \leq i \leq 2\delta$. The error-locator polynomial, that is $\sigma = \prod_{k \in M}(1 - \alpha^k X)$, can be computed from the syndrome polynomial S.

Indeed, let $Z = \sigma S \bmod X^{2\delta}$ and observe that:

$$Z = \sigma \cdot \left(\sum_{i=0}^{2\delta-1} \sum_{k=0}^{n'-1} e_k (\alpha^{i+1})^k X^i \right)$$
$$= \sigma \cdot \left(\sum_{k \in M} e_k \alpha^k \sum_{i=0}^{2\delta-1} (\alpha^k)^i X^i \right)$$

$$Z = \sigma \cdot \left(\sum_{k \in M} \frac{e_k \alpha^k}{1 - \alpha^k X} \right) \mod X^{2\delta}$$
$$= \sum_{k \in M} e_k \alpha^k \prod_{j \in M, j \neq k} (1 - \alpha^j X) \mod X^{2\delta} \quad (9)$$

Therefore, $\deg Z < \delta$. Moreover, $\deg \sigma \leq \delta$, $\sigma(0) = 1 \neq 0$ and $Z/\sigma = S \bmod X^{2\delta}$. Therefore, (Z, σ) is a solution to the (δ, δ)-Padé approximant problem for S. From σ, the indices M can be computed by evaluation of σ over the points $1, \alpha^{-1} \ldots, \alpha^{-n'+1}$, as $k \in M$ iff $\sigma(\alpha^{-k}) = 0$.

Forney's algorithm [For65] uses Equation 9 above to recover the value of e_k for $k \in M$. When evaluating Z on α^{-k}, we have $Z(\alpha^{-k}) = e_k \alpha^k \prod_{j \in M, j \neq k} (1 - \alpha^{j-k})$. Moreover, one can note that the derivative of σ is such that $\sigma'(\alpha^{-k}) = -\alpha^k \prod_{j \in M, j \neq k} (1 - \alpha^{j-k}) \neq 0$. Therefore, one can easily deduce the value $e_k = -Z(\alpha^{-k})/\sigma'(\alpha^{-k})$. When $\mathtt{wt}(e) > \delta$, the computed polynomial σ may have multiple roots, in which case $\sigma'(\alpha^{-k}) = 0$ for some roots α^{-k} of σ; we then set $e_k = 0$.

In order to derive an MPC protocol that does not leak any information from the input, one must consider the case where Padé approximant has no solution. This may occur for instance when an "incorrect" HQC ciphertext is submitted for decryption. Unfortunately, aborting the protocol will offer a vulnerability that must be avoided. Instead, we will simply set $(Z, \sigma) = (0, 1)$, which corresponds to correcting no errors. Algorithm 13 gives a detailed description of the decoding algorithm for Reed-Solomon codewords that will be compliant to our MPC setting.

Algorithm 13: Reed-Solomon Decoding

Parameters: q' a power of two, $\delta \in \mathbb{N}$, $n' = q' - 1$, α a primitive element of $\mathbb{F}_{q'}$.
Input: $y \in \mathbb{F}_{q'}[X]/(X^{n'} - 1)$ s.t. $y = c + e$, where $c \in BCH(q', \delta, \alpha)$, $\mathtt{wt}(\mathtt{e}) \leq \delta$
Output: $u \in \mathbb{F}_{q'}[X]$ s.t. c is the encoding of u in $BCH(q', \delta, \alpha)$

for $1 \leq i \leq 2\delta$ **do**
  $s_i \leftarrow y(\alpha^i)$
end
$S \leftarrow \sum_{i=0}^{2\delta-1} s_{i+1} X^i$
Compute the solution of the (δ, δ)-Padé approximant of S as (Z, σ) such that $\sigma(0) = 1$. If the Padé approximant problem has no solution, set $(Z, \sigma) \leftarrow (0, 1)$.
for $0 \leq k \leq n' - 1$ **do**
  if $\sigma(\alpha^{-k}) = 0$ *and* $\sigma'(\alpha^{-k}) \neq 0$ **then**
    $e_k \leftarrow -Z(\alpha^{-k})/\sigma'(\alpha^{-k})$
  end
  else
    $e_k \leftarrow 0$
  end
end
$e \leftarrow \sum_{i=0}^{n'-1} e_k X^k$, and $c \leftarrow y - e$
Return $u \leftarrow c$ quo $X^{2\delta}$

6.2 Protocol for Reed-Solomon decoding in MPC

Here, we present our MPC protocol for decoding Reed Solomon codes, using algorithm 13 as a baseline. Our protocol instantiates the ideal functionality $\mathcal{F}_{ABB+\texttt{ReedSolomon}}$, presented in Fig. 8. As for our decoding protocol for Reed-Muller, we need to ensure that the protocol is running on an input whose coefficients are in $\mathbb{F}_{q'}$, as the field $\mathbb{F}_q$ used for the MPC instanciation might be bigger than $\mathbb{F}_{q'}$. To this purpose, our protocol leaks the values $y_i^{q'} - y_i$ on input $[y]$, which should all be 0 on valid inputs.

The functionality is the same as $\mathcal{F}_{ABB}$ with one more functionality.

- Upon receiving $(\texttt{ReedSolomon}, \texttt{id}, \texttt{id}', q', \delta, \alpha)$ from all the parties, the functionality fetches $y \in \mathbb{F}_q[X]_{<n'}$ at $\texttt{id}$. It checks if $y_i \in \mathbb{F}_{q'}$ for $0 \leq i \leq n'-1$. If not, then the functionality leaks the values $y_i^{q'} - y_i$ for $0 \leq i \leq n'-1$ and sends an abort signal to all parties. Otherwise, it runs algorithm 13 on y. The result u is stored at $\texttt{id}'$.

Figure 8: $\mathcal{F}_{ABB+\texttt{ReedSolomon}}$

Let us detail how to build the protocol. First, computing the polynomial $[S]$ from $[y]$ does not require any communication, as computing $[s_i] = [e(\alpha^i)] = [y(\alpha^i)]$ for $1 \leq i \leq 2\delta$ amounts to evaluating $[y]$ on public points, which is linear. Next, we compute $[\sigma]$ and $[Z]$ as a particular solution of the (δ, δ)-Padé approximant problem, using protocol **Padé**. Note that using this protocol does not reveal any information about $[S]$ as it sets $[Z], [\sigma]$ such that $(Z, \sigma) = (0, 1)$ if no solution to the Padé problem exists, just as in Algorithm 13.

Then, we need to encode the non-zero indices in a shared form without revealing them. To do that, we compute $[\sigma(\alpha^{-k})]$ for $0 \leq k \leq n'-1$ locally from $[\sigma]$ (again, no communication is needed), and we do zero testing using Protocol **ZeroTest** to obtain the shared boolean $[b_k] = [\sigma(\alpha^{-k}) = 0]$.

Finally, we will use the following equality to compute $[e]$:

$$\forall 0 \leq k \leq n'-1,\ e_k = b_k\, Z(\alpha^{-k})\, \xi_k,\ \text{where } \xi_k = \sigma'(\alpha^{-k})^{q'-2}.$$

Indeed, since $\sigma'(\alpha^{-k}) \in \mathbb{F}_{q'}$, $\xi_k = \sigma'(\alpha^{-k})^{-1}$ if $\sigma'(\alpha^{-k}) \neq 0$, and $\xi_k = 0$ otherwise. Hence, e_k is exactly the value computed in Algorithm 13.

In the MPC setting, the shared values $[Z(\alpha^{-k})]$ and $[\sigma'(\alpha^{-k})]$ can be computed without any communication, as α is a public point, and $[\sigma']$ can be computed linearly from $[\sigma]$. The value $[\xi_k]$ is computed using protocol **Powers**, which, contrary to protocol **Inv**, does not reveal whether $\sigma'(\alpha^{-k}) = 0$ (this may happen when $\texttt{wt}(e) > \delta$, e.g. for an incorrect ciphertext). The value $[e_k]$ is finally computed as $[e_k] = [b_k][Z(\alpha^{-k})][\xi_k]$.

All these steps are detailed in protocol **ReedSolomon**. Theorem 7 states the security and the communication complexity of the protocol.

Theorem 7. *Let $\mathbb{F}_{q'} \subseteq \mathbb{F}_q$ be a field of characteristic 2, $\mathcal{C}$ be a BCH code of parameters (q', δ, α) (as in Definition 2) and define $n' = q' - 1$. Given a shared vector $[y] \in \mathbb{F}_q^{n'}$ such that $y = c + e$ where $c \in \mathcal{C}$ and $\texttt{wt}(\texttt{e}) \leq \delta$. The protocol **ReedSolomon** instantiates functionality $\mathcal{F}_{ABB+\texttt{ReedSolomon}}$ (see Fig. 8) in the $\mathcal{F}_{ABB}$-hybrid model to get the shared value over $\mathbb{F}_q$ of the decoding of $c + e$. Moreover, it requires at most $\mathcal{O}(\delta^{4.5} + q'^2)$ secure multiplications and $\mathcal{O}(1)$ rounds of multiplications.*

Proof. Let us build a simulator that interacts with $\mathcal{F}_{ABB+\texttt{ReedSolomon}}$ and simulates the views of an adversary in a real execution of the protocol. First, $\mathcal{S}$ calls the ideal functionality. Then, two cases are possible; either $\mathcal{S}$ receives nothing for the ideal functionality, meaning that the functionality correctly ran algorithm 13 on the shared input, or it receives the

leakage $(y_k^{q'} - y_k)_{0 \le k \le n'-1}$ and an abort signal. If $\mathcal{S}$ receives nothing, then $\mathcal{S}$ simulates the entire protocol as if no abort happens using the simulator corresponding to **Powers**, **Padé**, and **ZeroTest**. Notably, the revealing of values at step 3 are simulated as revealings of zero values. If $\mathcal{S}$ receives the leakage and an abort signal, then it simulates the calls to **Powers** of step 2 and then simulates revealings of values corresponding to step 3 as revealing of the values $(y_k^{q'} - y_k)_{0 \le k \le n'-1}$ which were sent by the ideals functionalities. In both cases, $\mathcal{S}$ simulates perfectly the views of parties as in a real execution of the protocol. Therefore, the protocol indeed instantiates the corresponding functionality.

Let us detail the communication complexity. The n' calls to **Powers** at step 2 require $n' \cdot \mathcal{O}(q') = \mathcal{O}(n'q')$ secure multiplications. The call to **Padé** at step 9 requires $\mathcal{O}(\delta \cdot \delta^{3.5} + \delta q')$ secure multiplications and $\mathcal{O}(1)$ rounds of multiplications. Indeed, one can note that even if the sharing is done over F_q, all the shared elements have a value lying in $\mathbb{F}_{q'}$. This could be leveraged into the calls to **ZeroTest** within the protocol **Padé**, replacing the dependence in q by one in q' in the complexity.

Similarly, steps 11 and 12 consist of n' calls to **ZeroTest** and n' calls to **Powers**, requiring $n' \cdot \mathcal{O}(q') = \mathcal{O}(n'q')$ secure multiplications. Step 13 requires a constant number of secure multiplications per coordinate, which amounts to $\mathcal{O}(n')$ secure multiplications in total. This leads to the expected communication complexity, as $n' = q' - 1$ and $\delta \le n'$. The number of rounds of multiplication is obviously in $\mathcal{O}(1)$, concluding the proof. □

Protocol 14: ReedSolomon

Parameters: q' such that $\mathbb{F}_{q'} \subseteq \mathbb{F}_q$, $\delta \in \mathbb{N}$, $n' = q' - 1$, $\alpha \in \mathbb{F}_{q'}$ primitive.
Input: $[y] = [y_0, \dots, y_{n'-1}] \in \mathbb{F}_q^{n'} \simeq \mathbb{F}_q[X]_{<n'}$
Output: $[u] \in \mathbb{F}_q[X]$ such that u is the output of algorithm 13 on y

1 **for** $0 \le k \le n' - 1$ **in parallel do**
2 $\quad [z_k] \leftarrow [y_k]^{q'} - [y_k]$ ▷ **Powers**
3 $\quad$ Parties reveal the value z_k, and abort if $z_k \neq 0$
4 **end**
5 **for** $1 \le i \le 2\delta$ **in parallel do**
6 $\quad [s_i] \leftarrow [y(\alpha^i)]$ ▷ no com.
7 **end**
8 $[S] \leftarrow \sum_{i=0}^{2\delta-1} [s_{i+1}] X^i$ ▷ no com.
9 Parties compute $[Z], [\sigma]$ as the (δ, δ)-Padé sol. for $[S]$ ▷ **Padé**, $(Z, \sigma) = (0, 1)$ if no sol.
10 Parties locally compute $[\sigma']$ ▷ no com.
11 **for** $0 \le k \le n' - 1$ **in parallel do**
12 $\quad$ Parties locally compute $[\sigma(\alpha^{-k})]$, $[\sigma'(\alpha^{-k})]$, and $[Z(\alpha^{-k})]$ ▷ no com.
13 $\quad [b_k] \leftarrow [\sigma(\alpha^{-k}) = 0]$ ▷ **ZeroTest**
14 $\quad$ Parties compute $[\xi_k] = [\sigma'(\alpha^{-k})^{q'-2}]$ ▷ **Powers**
15 $\quad [e_k] \leftarrow [b_k][Z(\alpha^{-k})][\xi_k]$
16 **end**
17 $[e] \leftarrow \sum_{i=0}^{n'-1} [e_i] X^i$ ▷ no com.
18 $[c] \leftarrow [y] - [e]$ ▷ no com.
19 $[u] \leftarrow [c] \text{ quo } X^{2\delta} \in \mathbb{F}_q[X]_{<n'-2\delta}$ ▷ no com.

7 Threshold variant of HQC.PKE

In this section, we present our full MPC protocol for decrypting a HQC ciphertext, assuming only a sharing of the secret key. This leads to the first threshold IND-CPA PKE based on HQC. We then estimate the sizes of the keys and ciphertext of this PKE instantiated on the parameters sets of the original HQC.PKE, and we compare it to the other code-based threshold schemes of [GLOV26].

The functionality is the same as $\mathcal{F}_{ABB}$ with one more functionality, parameterized by the total order $\prec$ on $\mathbb{F}_2^{m+1}$ of Section 5.4.

- Upon receiving $(\texttt{HQC}, \texttt{id}, \texttt{id}', \mathbf{c} = (\mathbf{u}, \mathbf{v}))$ from all the parties, the functionality fetches $\mathbf{y} \in \mathbb{F}_q[X]/(X^n - 1)$. It first checks if $\mathbf{y}_i \in \mathbb{F}_2$ for all $0 \leq i \leq n-1$. If not, the functionality aborts and sends the values $(\mathbf{y}_i^2 - \mathbf{y}_i)_{0 \leq i \leq n-1}$ to all parties. Otherwise, the functionality runs the decryption of $(\mathbf{u}, \mathbf{v})$; it computes $\mathbf{c}' = \text{Truncate}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y}, n - n_e n_i) \in \mathbb{F}_2^{n_e n_i}$. It sets $\mathbf{c}' = (\mathbf{c}'_1, \ldots, \mathbf{c}'_{n_e})$ and computes the closest codeword decoding $\mathbf{c}''_i = \text{Dec}_{\prec}(\mathbf{c}'_i)$ (see Section 5.4) for $1 \leq i \leq n_e$, with $\mathbf{c}''_i \in \mathbb{F}_2^{k_i}$. Then, it reinterprets $\mathbf{c}''_i$ as an element of $\mathbb{F}_{q'}$. It sets $\mathbf{c}'' = (\mathbf{c}''_1, \ldots, \mathbf{c}''_{n_e}, 0, \ldots, 0) \in \mathbb{F}_{q'}^{q'-1}$ and runs the Reed Solomon decoder on $\mathbf{c}''$ (i.e. algorithm 13). It obtains $\mathbf{m} \in \mathbb{F}_{q'}^{k_e}$ and it stores it in $\texttt{id}'$.

Figure 9: $\mathcal{F}_{ABB+\texttt{HQC}}$

7.1 MPC protocol for HQC.PKE decryption

In this section, we present a complete protocol to securely decrypt a ciphertext $\mathbf{c} = (\mathbf{u}, \mathbf{v})$ of the PKC version of the HQC cryptosystem [AAB+22] using part of a shared secret key $[\mathbf{y}]$. As for the classical decryption (see Fig. 2), the shared decryption is composed of two main steps. First, an algebraic operation (i.e. $\mathbf{v} - \mathbf{u} \cdot [\mathbf{y}]$) between the ciphertext $(\mathbf{u}, \mathbf{v})$ and part of the secret key $[\mathbf{y}]$ is performed. This operation doesn't require any communication, as it is linear in the shared secret key. Then, the errors are corrected using the MPC protocols detailed in sections 5 and 6 to recover the message in shared form, following the concatenated code construction: the internal Reed-Muller codewords are decoded with protocol **ReedMuller-ML** (Section 5.4), which outputs a closest codeword exactly as the decoder of the HQC specification, and the external Reed-Solomon codeword with protocol **ReedSolomon**. As explained in section 2.4.3, the shortened Reed-Solomon code $\mathcal{C}_e$ has symbols in $\mathbb{F}_{q'}$ where $q' = 2^{k_i} = 2^{m+1}$.

The ideal functionality corresponding to the decryption is presented in Fig. 9. As for the shared Reed-Muller decoding, we ensure that the coefficients of $[\mathbf{y}]$ actually lie in $\mathbb{F}_2$. This is done by computing the values $[\mathbf{y}_i^2 - \mathbf{y}_i]$ and revealing them.

One can notice that this protocol never fails on a shared input $[\mathbf{y}]$ such that $\mathbf{y} \in \mathbb{F}_2[X]/(X^n - 1)$. This means that if a ciphertext is used on the MPC protocol for decryption for which the decryption fails or is incorrect, then the shared output $[\mathbf{m}]$ will have an incorrect value, but the behavior of parties during the protocol would be the same as if a correct ciphertext was decrypted. This ensures that, if an adversary participates in a decryption query (as long as the number of corrupted parties is below the threshold for the considered secret sharing scheme), this adversary does not learn additional information than the eventual reception of the decrypted message. This property could be useful for a future threshold IND-CCA construction based on this decryption protocol.

To perform outer decoding (with Reed-Solomon decoding) after inner decoding (with Reed-Muller decoding), parties need to reinterpret elements of $\mathbb{F}_2^{k_i}$ as elements of $\mathbb{F}_{q'}$ (with $\mathbb{F}_{q'} \subseteq \mathbb{F}_q$). However, because the MPC protocol's base field is $\mathbb{F}_q$, the outputs of the inner decoding step are shared as elements of $\mathbb{F}_q^{k_i}$. To achieve this conversion, parties denote $\eta \in \mathbb{F}_{q'}$ the representative of $(0, 1, 0, \ldots, 0) \in \mathbb{F}_2^{k_i}$. Then, given $[a] \in \mathbb{F}_q^{k_i}$, they compute $[\tilde{a}] \leftarrow \sum_{j=1}^{k_i} a_j \eta^{j-1}$. If $a \in \mathbb{F}_2^{k_i}$, then this operation indeed reinterprets a as an element $\tilde{a} \in \mathbb{F}_{q'}$ correctly. This leads to the protocol **HQCdecryption** given below. Theorem 8 states the security and the communication complexity of the protocol.

Theorem 8. *Protocol **HQCdecryption** instantiates functionality $\mathcal{F}_{ABB+\texttt{HQC}}$ (see Fig. 9) in*

the $\mathcal{F}_{ABB}$-hybrid model. It requires at most $\mathcal{O}(n+n_e\gamma 4^m+\delta_e^{4.5}+2^{2m})$ secure multiplications and $\mathcal{O}(1)$ rounds of multiplications.

Proof. Similarly as in Theorem 6, we build a simulator that calls on the ideal functionality, and simulates the corresponding revealings according to the answer of the functionality (i.e. revealing the values sent by the functionality if it aborts, and zeroes otherwise). If the functionality did not abort, the simulator can simulate the rest of the protocol using the simulators corresponding to **ReedMuller-ML** and **ReedSolomon** without any abort. Therefore, the protocol indeed instantiates the functionality in the $\mathcal{F}_{ABB}$-hybrid model.

Steps 1 through 4 require n secure multiplications. Step 7 requires n_e parallel calls to **ReedMuller-ML**, amounting to $\mathcal{O}(n_e\gamma 4^m)$ secure multiplications and a constant number of rounds of multiplications. Step 13 requires one call to **ReedSolomon**, amounting to $\mathcal{O}(\delta_e^{4.5}+q'^2)=\mathcal{O}(\delta_e^{4.5}+2^{2m})$ secure multiplications and $\mathcal{O}(1)$ rounds. □

Protocol 15: HQCdecryption

Parameters: The relevant HQC parameters: $n\in\mathbb{N}$, $\lambda\in\mathbb{N}$, $\delta_e\in\mathbb{N}$, $m\in\mathbb{N}$, $\gamma\in\mathbb{N}$, $n_i=\gamma 2^m$, $k_i=m+1$, $q'=2^{m+1}=2^{k_i}$, $n_e\in\mathbb{N}$, $k_e=\lambda/k_i$, $\eta\in\mathbb{F}_{q'}$ being the representative of $(0,1,0,\dots,0)$, $\prec$ the total order on $\mathbb{F}_2^{m+1}$ of Section 5.4

Input: An HQC ciphertext $\mathbf{c}=(\mathbf{u},\mathbf{v})\in(\mathcal{R}^{HQC})^2$

Input: $[\mathbf{y}]\in\mathbb{F}_q[X]/(X^n-1)$ such that $\mathbf{y}$ corresponds to the secret key of the public key used to compute $\mathbf{c}$

Output: $[\mathbf{m}]$ such that $\mathbf{m}\in\mathbb{F}_2^\lambda$ is the plaintext corresponding to $\mathbf{c}$

1 **for** $0\le l\le n-1$ **in parallel do**
2 $\quad[z_l]\leftarrow[\mathbf{y}_l][\mathbf{y}_l]-[\mathbf{y}_l]$
3 $\quad$Parties reveal the value z_l, and abort if $z_l\neq 0$
4 **end**
5 $[\mathbf{c}']\leftarrow\text{Truncate}(\mathbf{v}-[\mathbf{y}]\cdot\mathbf{u},n-n_en_i)$ ▷ no com.
6 $([\mathbf{c}'_1],\dots,[\mathbf{c}'_{n_e}])\leftarrow[\mathbf{c}']\in(\mathbb{F}_q^{n_i})^{n_e}$ ▷ no com.
7 **for** $1\le i\le n_e$ **in parallel do**
8 $\quad$Parties decode $[\mathbf{c}'_i]$ as a duplicated Reed-Muller codeword to obtain $[\mathbf{c}''_i]$,
9 $\quad\quad$with $\mathbf{c}''_i=\text{Dec}_\prec(\mathbf{c}'_i)$ ▷ **ReedMuller-ML**
10 $\quad[\tilde{\mathbf{c}}''_i]\leftarrow\sum_{j=1}^{k_i}[\mathbf{c}''_{i,j}]\eta^{j-1}$ ▷ no com.
11 **end**
12 $[\tilde{\mathbf{c}}'']\leftarrow([\tilde{\mathbf{c}}''_1],\dots,[\tilde{\mathbf{c}}''_{n_e}],[0],\dots,[0])$
13 Parties decode $[\tilde{\mathbf{c}}'']$ as a noisy Reed-Solomon code, obtain $[\mathbf{m}]\in\mathbb{F}_q^{k_e}$ ▷ **ReedSolomon**

Assuming a trusted dealer for generating and sharing HQC.PKE secret keys, Theorem 8 leads to a novel CPA threshold PKE based on the HQC.PKE cryptosystem. As this threshold PKE does not modify HQC.PKE to achieve thresholdizing, this threshold variant only relies on the same hardness assumption as the original cryptosystem, aside from the additional assumptions for enabling MPC (this would depend on the chosen instantiation for the MPC protocols). Moreover, since protocol **ReedMuller-ML** computes exactly the same decoding as the reference implementation of HQC, our threshold variant uses the same parameter sets and has the same decryption failure rate as HQC.PKE.

7.2 Efficiency analysis

In this section, we provide some evaluation of the efficiency of our threshold variant of the HQC.PKE cryptosystem, as well as comparisons with the McEliece-based threshold cryptosystem of [GLOV26].

Choice of the MPC field.

The field $\mathbb{F}_q$ used for the MPC protocols must contain $\mathbb{F}_{q'}=\mathbb{F}_{2^{m+1}}$ for the Reed-Solomon decoding, i.e. $m+1$ must divide $\log_2 q$. Moreover, it must be large enough to use the

evaluation-interpolation techniques of protocol **Poly2Mul** and of the protocols of Section 3.1, as well as protocol **ReedMuller-ML**, which requires an element of multiplicative order larger than $\max(\gamma 2^m, 2^{m+1} - 1)$. For all the parameter sets of HQC, we have $m + 1 = 8$ and $\gamma 2^m \in \{384, 640\}$, so that the smallest suitable field is $\mathbb{F}_{2^{16}}$.

Efficiency estimates for our HQC decryption protocol.

We estimate the number of secure multiplications and of rounds of the decryption protocol from the complexities of the subprotocols given in Section 2.3, an inner product being counted as one secure multiplication. In order to compare our protocol with the protocol of [GLOV26] in the same way, the rounds of the subprotocols are added sequentially, without taking into account the possible overlapping of independent steps (for instance, this leads to 25 rounds for protocol **ReedMuller-ML** instead of the 16 rounds of Theorem 6). Since all the elements whose powers are computed in protocol **ReedMuller-ML** are non-zero, these powers are computed with protocol **PowersNZ**, and protocol **PrefixOR** is used with $\tau = 1$ in the Reed-Solomon decoding.

Table 4 gives these estimates for each HQC parameter set, together with the number of secure multiplications due to the decoding of the n_e internal Reed-Muller (RM) codewords and to the decoding of the external Reed-Solomon (RS) codeword (the remaining ones being due to the input check on the secret key).

Table 4: Estimated number of secure multiplications and of rounds of our decryption protocol for the HQC parameter sets.

Parameter Set	Secure Mult.	RM decoding	RS decoding	Rounds
HQC1	$5.18 \cdot 10^7$	$3.24 \cdot 10^7$ (63%)	$1.94 \cdot 10^7$ (37%)	87
HQC3	$7.21 \cdot 10^7$	$4.69 \cdot 10^7$ (65%)	$2.52 \cdot 10^7$ (35%)	87
HQC5	$4.05 \cdot 10^8$	$7.53 \cdot 10^7$ (19%)	$3.30 \cdot 10^8$ (81%)	87

For HQC1 and HQC3, about two thirds of the secure multiplications are due to the Reed-Muller decoding, the dominant steps of protocol **ReedMuller-ML** being the computation of the products U_w and of their powers (which select the closest codeword among the 2^{m+1} candidates, and amount to about half of its cost) and the computation of the powers of $\theta_{(0,u)}$. For HQC5, the Reed-Solomon decoding is dominant, because of the larger correction capacity $\delta_e = 29$, which increases the number and the size of the determinants computed in protocol **Padé**.

Comparison with [GLOV26].

While the main design principle behind both cryptosystems (i.e. designing a MPC protocol for decryption) is the same, their comparison is not straightforward. First, the schemes from [GLOV26] achieve a different security notion, so we compare only to the IND-CPA variant (i.e. before applying their generic conversion from IND-CPA to IND-CCA security). Moreover, as explained below, the constraints on the underlying MPC field are not the same. Thus, the communication induced by a given number of secure multiplications might be different.

We compare threshold PKEs based on both McEliece and HQC, using parameter sets for NIST security levels (1, 3 and 5), on two main aspects: the size of the different elements of the cryptosystem itself (i.e. ciphertext size, public and partial secret key size), and communication estimates for the MPC protocols (i.e. size of the field, number of secure multiplications, number of rounds).

For [GLOV26], the dominant constraint on the base field for MPC is that it needs to contain the field corresponding to the Goppa code construction, which means that this field can be picked for the MPC protocol.

While the sizes of ciphertexts and public keys are identical as for HQC.PKE, i.e. the size of the public key is n bits and the size of one ciphertext is $n + n_e n_i$ bits, the partial secret keys size has a dependence in $\log q$. Indeed, the secret key is shared as an element of $\mathbb{F}_q[X]/(X^n - 1)$, since the base field used of MPC protocols is $\mathbb{F}_q$. This means that each partial secret key is of size $\log(q) \cdot n$ bits.

Table 5: Parameter choices and sizes for instantiations of our t-PKE scheme as well as [GLOV26]. "mc" is short for "mceliece".

Scheme	Security	MPC field	Public Key	Partial Secret Key	Ciphertext
[GLOV26], mc348864	NIST-1	2^{12}	261 KB	5.52 KB	128 B
Ours, HQC1		2^{16}	2.21 KB	35.3 KB	4.42 KB
[GLOV26], mc460896	NIST-3	2^{13}	524 KB	7.96 KB	188 B
Ours, HQC3		2^{16}	4.48 KB	71.7 KB	8.96 KB
[GLOV26], mc6960119	NIST-5	2^{13}	1.04 MB	11.9 KB	226 B
Ours, HQC5		2^{16}	7.20 KB	115 KB	14.4 KB

Comparing both approaches, one may notice that McEliece-based PKEs have a lower secret key and ciphertext sizes while our HQC-based constructions still preserve the advantage of a much smaller public key, as in the non-threshold variants. The efficiency estimates of both decryption protocols are given in Table 6. Since the communication induced by a secure multiplication grows linearly with the size of the field elements, we also give the product of the number of secure multiplications and $\log_2 q$, as a rough estimate of the total communication volume.

Table 6: Efficiency comparison of the decryption protocols of our t-PKE scheme (with protocol **ReedMuller-ML**) as well as [GLOV26]. "mc" is short for "mceliece".

Scheme	Security	MPC field	Secure Mult.	Mult. $\times \log_2 q$	Rounds
[GLOV26], mc348864	NIST-1	2^{12}	$2.38 \cdot 10^6$	$2.86 \cdot 10^7$	53
Ours, HQC1		2^{16}	$5.18 \cdot 10^7$	$8.29 \cdot 10^8$	87
[GLOV26], mc460896	NIST-3	2^{13}	$4.72 \cdot 10^6$	$6.14 \cdot 10^7$	53
Ours, HQC3		2^{16}	$7.21 \cdot 10^7$	$1.15 \cdot 10^9$	87
[GLOV26], mc6960119	NIST-5	2^{13}	$8.74 \cdot 10^6$	$1.14 \cdot 10^8$	53
Ours, HQC5		2^{16}	$4.05 \cdot 10^8$	$6.48 \cdot 10^9$	87

Our decryption protocol requires between 15 and 46 times more secure multiplications than the one of [GLOV26], on a slightly larger field, and 87 rounds instead of 53. The values in Tables 4 and 6 must be understood as rough estimates, and might not represent practical communication costs. While direct comparison could help to highlight the advantage of each approach, it should not be used to derive which construction would be the most efficient. Practical implementations would be needed to reach a definitive conclusion, but this is a non-trivial task, which is a research direction on its own and is out of the scope of our work. We shall recall that the actual communication cost, both in terms of total message size and number of communication rounds, highly depends on the chosen sharing technique, which again calls for thorough implementations.

In terms of security comparison, two aspects need to be considered: the security related to the decryption protocols, and the security of the base schemes. Our decryption protocols as well as the one in [GLOV26] are built in the ABB model. Therefore, in order to compare the assumptions required for the schemes, we would have to compare the security assumptions and guarantees that would be needed for the MPC protocols, which depend on the chosen instantiation of the protocols. Furthermore, thresholdization does not add any new assumption compared to the respective base schemes, nor modify the parameter sets (since protocol **ReedMuller-ML** computes exactly the same decoding as the reference implementation of HQC, our threshold variant has the same decryption failure rate as

HQC.PKE). Therefore, the advantages of using our scheme over a McEliece-based one are mainly the same as using regular HQC over regular McEliece: smaller public keys, and the fact that recent progress has improved the cryptanalysis of the problems on which Classic McEliece relies [Ran25, Hem26, BLRT26, GIJS26].

Efficiency improvements. One could circumvent the possible higher cost of threshold HQC in several ways. First, one could tweak the parameters sets of HQC to find a better balance between Reed-Muller and Reed-Solomon decoding (i.e. allowing for greater correction capacity for Reed-Solomon when the cost of Reed-Muller is dominant, and vice-versa). The security of these new parameters would need to be thoroughly analyzed. Next, one could consider using protocols that have a non-constant number of round in theory, but might have better communication costs in practice when applied to HQC decryption. This would probably lead to very different MPC techniques than those presented in this article.

CCA security. Achieving CCA security would also be desirable, but is highly non trivial and could be the topic of future work. This could be achieved by applying some kind of variants of either the Fujisaki-Okamoto transform [FO13, HHK17] or the Naor-Yung transform [NY90, Lin03], adapted to the threshold context. Both solutions would have challenges to overcome. On one hand, a FO-like transform would require evaluating hash functions, and a re-encryption step in MPC, which would degrade the efficiency of the threshold decryption protocol even more. On the other hand, a Naor-Yung-style transform (the route [GLOV26] took) would have two problems to solve. First, it would require the design of a Zero-Knowledge proof system tailored to well-formed HQC ciphertexts, which does not currently exist and is non-trivial to build because of the quasi-cyclic properties of the scheme. Second, HQC has a non-zero decryption failure rate, which would have to be folded into the analysis of any such transform.

Acknowledgements

This work was supported by the French National Research Agency with the grants: 22-PECY-0010, 22-PECY-003, ANR-21-CE39-0006.

References

[AAB+22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[ABC+22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

[BCD+24] Lennart Braun, Guilhem Castagnos, Ivan Damgård, Fabien Laguillaumie, Kelsey Melissaris, Claudio Orlandi, and Ida Tucker. An improved threshold homomorphic cryptosystem based on class groups. In Clemente Galdi and

Duong Hieu Phan, editors, *SCN 24, Part II*, volume 14974 of *LNCS*, pages 24–46. Springer, Cham, September 2024. doi:10.1007/978-3-031-71073-5_2.

[BD10] Rikke Bendlin and Ivan Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 201–218. Springer, Berlin, Heidelberg, February 2010. doi:10.1007/978-3-642-11799-2_13.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. doi:10.1145/62212.62213.

[BIB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In Piotr Rudnicki, editor, *8th ACM PODC*, pages 201–209. ACM, August 1989. doi:10.1145/72981.72995.

[BLRT26] Pierre Briaud, Axel Lemoine, Hugues Randriambololona, and Jean-Pierre Tillich. A heuristic subexponential attack on the McEliece cryptosystem. Cryptology ePrint Archive, Paper 2026/1232, 2026. URL: https://eprint.iacr.org/2026/1232.

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 371–404. Springer, Singapore, December 2023. doi:10.1007/978-981-99-8721-4_12.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001. doi:10.1109/SFCS.2001.959888.

[CD01] Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, August 2001. doi:10.1007/3-540-44647-8_7.

[CDM00] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 316–334. Springer, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_22.

[CEN25] Sofía Celi, Daniel Escudero, and Guilhem Niot. Share the MAYO: Thresholdizing MAYO. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part I*, pages 165–198. Springer, Cham, April 2025. doi:10.1007/978-3-031-86599-2_6.

[CKP07] Ronald Cramer, Eike Kiltz, and Carles Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 613–630. Springer, Berlin, Heidelberg, August 2007. doi:10.1007/978-3-540-74143-5_34.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002. doi:10.1145/509907.509980.

[CLT22] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Threshold linearly homomorphic encryption on $\mathbf{Z}/2^k\mathbf{Z}$. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 99–129. Springer, Cham, December 2022. doi:10.1007/978-3-031-22966-4_4.

[CS19] Daniele Cozzo and Nigel P. Smart. Sharing the LUOV: Threshold post-quantum signatures. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 128–153. Springer, Cham, December 2019. doi:10.1007/978-3-030-35199-1_7.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994. doi:10.1145/195058.195405.

[Des93] Yvo Desmedt. Treshold cryptosystems (invited talk). In Jennifer Seberry and Yuliang Zheng, editors, *AUSCRYPT'92*, volume 718 of *LNCS*, pages 3–14. Springer, Berlin, Heidelberg, December 1993. doi:10.1007/3-540-57220-1_47.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990. doi:10.1007/0-387-34805-0_28.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Berlin, Heidelberg, March 2006. doi:10.1007/11681878_15.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, February 2001. doi:10.1007/3-540-44586-2_9.

[DLN+21] Julien Devevey, Benoît Libert, Khoa Nguyen, Thomas Peters, and Moti Yung. Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 659–690. Springer, Cham, May 2021. doi:10.1007/978-3-030-75245-3_24.

[FO13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013. doi:10.1007/s00145-011-9114-1.

[For65] G. Forney. On decoding bch codes. *IEEE Transactions on Information Theory*, 11(4):549–557, 1965. doi:10.1109/TIT.1965.1053825.

[FP01] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 351–368. Springer, Berlin, Heidelberg, December 2001. doi:10.1007/3-540-45682-1_21.

[FPS01] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In Yair Frankel, editor, *FC 2000*, volume 1962 of *LNCS*, pages 90–104. Springer, Berlin, Heidelberg, February 2001. doi:10.1007/3-540-45472-1_7.

[GG03] Joachim Gathen and Jürgen Gerhard. *Modern computer algebra (2. ed.)*. Cambridge University Press, 01 2003.

[GIJS26] Ashrujit Ghoshal, Yuval Ishai, Aayush Jain, and Nuozhou Sun. Quasipolynomial cryptanalysis of the McEliece cryptosystem (or: PIR meets McEliece). Cryptology ePrint Archive, Paper 2026/1630, 2026. URL: https://eprint.iacr.org/2026/1630.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007. doi:10.1007/s00145-006-0347-3.

[GLOV24] Pascal Giorgi, Fabien Laguillaumie, Lucas Ottow, and Damien Vergnaud. Fast secure computations on shared polynomials and applications to private set operations. In Divesh Aggarwal, editor, *ITC 2024*, volume 304 of *LIPIcs*, pages 11:1–11:24. Schloss Dagstuhl, August 2024. doi:10.4230/LIPIcs.ITC.2024.11.

[GLOV26] Pascal Giorgi, Fabien Laguillaumie, Lucas Ottow, and Damien Vergnaud. Threshold Niederreiter: chosen-ciphertext security and improved distributed decoding. *Designs, Codes and Cryptography*, 94(3):64, February 2026. doi:10.1007/s10623-025-01795-6.

[GRJK00] Rosario Gennaro, Tal Rabin, Stanislaw Jarecki, and Hugo Krawczyk. Robust and efficient sharing of RSA functions. *Journal of Cryptology*, 13(2):273–300, March 2000. doi:10.1007/s001459910011.

[Hem26] Tobias Hemmert. Key recovery for the McEliece cryptosystem using higher-order vanishing. Cryptology ePrint Archive, Paper 2026/1339, 2026. URL: https://eprint.iacr.org/2026/1339.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 341–371. Springer, Cham, November 2017. doi:10.1007/978-3-319-70500-2_12.

[HIKC23] Koki Hamada, Dai Ikarashi, Ryo Kikuchi, and Koji Chida. Efficient decision tree training with new data structure for secure multi-party computation. *PoPETs*, 2023(1):343–364, January 2023. doi:10.56553/popets-2023-0021.

[HLPS26] Seungwoo Han, Jooyoung Lee, Seungmin Park, and Mincheol Son. Secure matrix invertibility testing over fields of small order or characteristics. Cryptology ePrint Archive, Paper 2026/514, 2026. URL: https://eprint.iacr.org/2026/514.

[HMR+19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32(2):265–323, April 2019. doi:10.1007/s00145-017-9275-7.

[LC04] S. Lin and D.J. Costello. *Error Control Coding: Fundamentals and Applications*. Computer applications in electrical engineering series. Pearson-Prentice Hall, 2004.

[Lin03] Yehuda Lindell. A simpler construction of cca2-secure public-key encryption under general assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 241–254. Springer, Berlin, Heidelberg, May 2003. doi:10.1007/3-540-39200-9_15.

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Berlin, Heidelberg, December 2001. doi:10.1007/3-540-45682-1_20.

[Mas69] J. Massey. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969. doi:10.1109/TIT.1969.1054260.

[MF06] Payman Mohassel and Matthew Franklin. Efficient polynomial operations in the shared-coefficients setting. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 44–57. Springer, Berlin, Heidelberg, April 2006. doi:10.1007/11745853_4.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. North Holland, 1977.

[MV24] Jules Maire and Damien Vergnaud. Secure multi-party linear algebra with perfect correctness. *CiC*, 1(1):29, 2024. doi:10.62056/avzojbkrz.

[MW08] Payman Mohassel and Enav Weinreb. Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 481–496. Springer, Berlin, Heidelberg, August 2008. doi:10.1007/978-3-540-85174-5_27.

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990. doi:10.1145/100216.100273.

[Pan00] Victor Y. Pan. New techniques for the computation of linear recurrence coefficients. *Finite Fields and Their Applications*, 6(1):93–118, 2000. URL: https://www.sciencedirect.com/science/article/pii/S1071579799902678, doi:https://doi.org/10.1006/ffta.1999.0267.

[Ran25] Hugues Randriambololona. The syzygy distinguisher. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VI*, volume 15606 of *LNCS*, pages 324–354. Springer, Cham, May 2025. doi:10.1007/978-3-031-91095-1_12.

[Ree54] I. Reed. A class of multiple-error-correcting codes and the decoding scheme. *Transactions of the IRE Professional Group on Information Theory*, 4(4):38–49, 1954. doi:10.1109/TIT.1954.1057465.

[Sch93] Arnold Schönhage. Fast parallel computation of characteristic polynomials by leverrier's power sum method adapted to fields of finite characteristic. In Andrzej Lingas, Rolf Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming*, pages 410–417, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[SG02] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, March 2002. doi:10.1007/s00145-001-0020-9.

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. doi:10.1145/359168.359176.

[THO24] Kota Takahashi, Keitaro Hashimoto, and Wakaha Ogata. Chosen-ciphertext secure code-based threshold public key encryptions with short ciphertext. *DCC*, 92(2):277–301, 2024. doi:10.1007/s10623-023-01309-2.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. doi:10.1109/SFCS.1982.38.