IACR Communications in Cryptology
Vol. 0, No. 0, 31 pages.

# On the $s$-projected Goppa Codes and their Application to the Interactive Oracle Proofs of Proximity

Abhraneel Dutta[1], Veronika Kuchta[2] and Edoardo Persichetti[2]

[1] Department of Mathematics, University of Miami, USA
[2] Department of Mathematics and Statistics, Florida Atlantic University, USA

**Abstract.** Error-correcting codes are a useful tool in cryptography, being used both to design schemes whose hardness relies on coding theory problems (the area typically known as code-based cryptography) but also as a building block in other contexts, such as proof systems and associated cryptographic argument systems. Among all codes used in cryptography, Goppa codes stand out, notably in the well-known McEliece encryption scheme, where they have withstood the test of time as one of very few classes of codes to be considered secure, for over 40 years. Therefore, a natural question arises whether Goppa codes can also be used in the construction of proof systems or cryptographic argument systems. In the context of probabilistically checkable proofs (PCPs) [AS92], codes are used to transform proofs into a format that requires only a constant number of queries for verification. This efficiency is often achieved using low-degree polynomial codes, which enable the verifier to check the consistency through interpolation, ensuring correctness with minimal effort. This application underscores the importance of good codes in enhancing the robustness and verifiability of proof systems. Given their critical role, we take a closer look at how error-correcting codes are used in proof systems to find a new way of obtaining an interactive oracle proof system of proximity (IOPP) from Goppa codes.

**Keywords:** Goppa Codes · Interactive Oracle Proofs of Proximity

## 1 Introduction

Efficient proof systems have evolved significantly since the introduction of Probabilistically Checkable Proofs (PCPs) [AS92], which allow a verifier to check the validity of a proof by inspecting only a small portion of it. This innovation laid the groundwork for Interactive Oracle Proofs (IOPs) [BCS16], which extend the PCP model by incorporating interaction from Interactive Proofs (IPs) [GMR85]. IOPs maintain the expressive power of PCPs while improving efficiency through interaction, and their optimization has been a focus of recent research.

Interactive oracle proofs (IOPs) were introduced by Ben-Sasson, Chiesa, and Spooner [BCS16], and independently by Reingold, Rothblum, and Rothblum [RRR16]. They generalize both PCPs and interactive proofs to a multi-round setting. [BCS16] also showed

E-mail: abdutta@miami.edu (Abhraneel Dutta), vkuchta@fau.edu (Veronika Kuchta), epersichetti@fau.edu (Edoardo Persichetti)

This work is licensed under a "CC BY 4.0" license.

how to compile any public-coin IOP into a non-interactive argument in the random oracle model. In this BCS transformation, the prover commits to each oracle message with a Merkle tree, the verifier's challenges come from hashing the transcript, and each query is answered with an authentication path. Because of the extra interaction, IOP provers can be much more efficient than classical PCP provers. For example, IOPs with quasilinear-time provers were later built for the Rank-1 Constraint Satisfaction (R1CS) problem [BCR+19, COS20].A further refinement is the polynomial IOP model [BFS20], which simplifies the representation of prover messages as univariate polynomials over finite fields. Each prover message $m_i$ is encoded as a polynomial $p_i$, with degree $\deg(p_i) \leq d_i$. Verifiers then query evaluations $p_i(r)$, and the prover provides responses $v_i$, which the verifier validates using cryptographic tools like polynomial commitment schemes. This framework transforms standard IOPs into succinct arguments, employing techniques such as the Merkle tree-based "in-the-head" simulation introduced in [BCS16].

A polynomial IOP is typically paired with a polynomial commitment scheme, which enables verifiable computation of polynomial evaluations. For example, the Fast Reed-Solomon Interactive Oracle Proofs of Proximity (FRI) protocol [BSBHR18] is a polynomial commitment scheme that also serves as an Interactive Oracle Proof of Proximity (IOPP). It is particularly efficient in terms of the prover's effort and proof size. Alternatively, Ligero [AHIV17] offers faster evaluation but produces larger proofs compared to FRI. Both protocols rely heavily on error-correcting codes, such as Reed-Solomon codes, which are instrumental in encoding data and enabling efficient algebraic operations.

At the core of many proof systems, including IOPs, is the concept of arithmetization: an algebraic technique that transforms binary relations into structured representations using low-degree polynomials. These representations map instance-witness pairs $(x, w)$ into codewords of specific error-correcting codes $\mathscr{C}$. More specifically, when $(x, w) \in R$, the transformation places the pair into $\mathscr{C}$; otherwise, the resulting data is far from any valid codeword. This principle, foundational to PCPs [AS92, BFLS91], underpins modern IOPs and polynomial commitment schemes like FRI and Ligero, enabling robust and scalable verifications.

These developments illustrate the ongoing interplay between theoretical advancements in proof systems and practical applications, with tools like Reed-Solomon codes and commitment schemes at the forefront of cryptographic innovation. We will delve deeper into the specific properties of FRI as an IOPP in the following section.

**Interactive Oracle Proofs of Proximity.** An Interactive Oracle Proof of Proximity (IOPP), as introduced by Ben-Sasson et al. [BSBHR18], is technically an extension of an Interactive Oracle Proof (IOP). An IOPP is defined for a specific relation $R$ and is characterized by round complexity $k$, soundness error $\epsilon$, and proximity parameter $\delta$. This proof system involves a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, and it adheres to two key properties:

- *Completeness:* For every instance-witness pair $(x, w)$ in the relation $R$, the proof system $\langle \mathcal{P}(x, w), \mathcal{V}^w(x) \rangle$ is a $k$-round interactive oracle proof with accepting probability 1.

- *Soundness:* For every instance-witness pair $(x, w)$ in the relation $R$ with relative Hamming distance $\Delta(w, R|_x) \geq \delta(n)$ and unbounded malicious prover $\mathcal{P}^*$ holds: $\langle \mathcal{P}^*, V^w(x) \rangle$ is a $k(n)$-round interactive oracle protocol with an accepting probability of at most $\epsilon(n)$, where $n$ denotes the security parameter.

**IOPP from Reed-Solomon Codes.** In [BSBHR18], the authors present an IOPP for Reed-Solomon codes, named the Fast Reed-Solomon IOPP (FRI). This construction enables a prover with linear complexity and a verifier with logarithmic complexity, making it highly efficient. A Reed-Solomon (RS) code, denoted by $RS[\mathbb{F}, S, \rho]$ can be seen as a space of functions $f : S \to \mathbb{F}$ representing evaluations of polynomials with degree $d < \rho N$ where $N$ is the size of $S$. The proximity testing problem for Reed-Solomon codes requires a verifier, with access to a random oracle, to determine whether a function $f$ is a codeword of $RS[\mathbb{F}, S, \rho]$ or whether $f$ is $\delta$-far (in Hamming distance) from any codeword.

The fundamental challenge addressed by an interactive oracle proof of proximity is as follows: given access to a function $f : S \to \mathbb{F}_q$, determine whether $f$ is part of an RS code. This task is non-trivial because verifying membership requires evaluating $f$ at multiple points. Specifically, if the degree of $f$ is bounded by $d$, then at least $d$ points must be checked to determine if $f$ is consistent with an RS code. Thus, solving this problem requires a minimum of $d$ queries, with a time complexity that scales linearly with $n$, where $n$ represents the size of the subset $S$.

An efficient approach to this problem is to split $f$ into two pieces, $f_1$ and $f_2$, each of length $n/2$, and verify that both pieces belong to RS codes. This method reduces the complexity and makes the verification process more manageable, since one claim is reduced to 2 claims of smaller size. Then both of these claims will be combined into one single claim that a random linear combination of $f_1, f_2$ is in the RS code, i.e. $f_1 + Zf_2 \in RS[\mathbb{F}_q, S', d']$, for a random variable $Z \in \mathbb{F}_q$ where $f_1, f_2 : S' \to \mathbb{F}_q$ and $d' = d/2$.

**Goppa Codes.** Goppa codes play a pivotal role in cryptography due to their long security history, particularly in the context of encryption (e.g. the McEliece cryptosystem), where several other code families have instead proved to be too fragile [SS92, MS07, Wie10, LT13, BCD+16]. From a high-level perspective, Goppa codes, which are built using notions from algebraic geometry, offer a variety of tools to the cryptographer, including access to a very powerful, specialized decoder [Pat75]. Compared to Reed-Solomon codes, they can be seen as a more sophisticated choice, since in fact Goppa codes are defined as *subfield subcodes* of *Generalized* Reed-Solomon codes.

For our purposes, we explore Goppa codes as a possible building block in the construction of an IOPP, exactly due to their algebraic properties and efficient decoding algorithm. Indeed, IOPPs require codes with strong dual-distance properties to ensure the soundness of proofs while maintaining proximity testing efficiency. Goppa codes, defined over finite fields with carefully chosen irreducible polynomials, inherently provide these properties, making them ideal for constructing proximity testing schemes. Their capacity for error correction allows them to handle noisy or approximate queries effectively, a critical feature in IOPP frameworks. Moreover, the algebraic properties of Goppa codes enable the design of succinct and verifiable proofs, aligning well with the computational and communication efficiency goals of IOPPs. These features, combined with their proven resilience in cryptographic settings, highlight their versatility and reliability in interactive proof systems.

## 1.1 Our Contribution

We present a novel construction of an interactive oracle proof of proximity (IOPP) based on binary Goppa codes. To the best of our knowledge, existing IOPP constructions are instantiated almost exclusively from Reed-Solomon codes or algebraic geometry codes

[BSBHR18, BSGKS19, BLNR22]. Unlike these approaches, our construction is based on the syndrome representation of Goppa codes rather than on low-degree polynomial evaluations.

Our approach builds upon the family of $s$-projected Goppa codes introduced by Loidreau [Loi01a]. The key idea is to exploit the Frobenius symmetry that arises when the Goppa polynomial has coefficients in a proper subfield. This symmetry yields a Frobenius-invariant subcode together with a natural decomposition of the support into Frobenius orbits, allowing us to work over a reduced evaluation domain while preserving the parity-check equation required for membership testing.

**$s$-Projected Codes.** In [Loi01a], the author presents a framework for deriving $s$-projected codes from Goppa codes, leveraging the structure of a special automorphism group. The key observation is that if the coefficients of the generating polynomial $g(x)$ lie in a proper subfield $\mathbb{F}_{2^s}$, then the automorphism group of $\Gamma$ contains a cyclic subgroup generated by the Frobenius automorphism $x \mapsto x^{2^s}$. This first gives the invariant subcode $I(L, g)$, consisting of codewords fixed by the Frobenius action. The $s$-projected Goppa code is then obtained by projecting these invariant codewords onto one representative from each Frobenius orbit.

More precisely, an $s$-projected Goppa code is derived by reducing the dimensionality of the original Goppa code via a projection onto a lower-dimensional space, while preserving its core algebraic structure. For a Goppa code $\Gamma(L, g)$ defined over a (large) finite field $\mathbb{F}_q$, an $s$-projection restricts the evaluation set $L$ to an $s$-dimensional subfield $\mathbb{F}_{q'} \subseteq \mathbb{F}_q$, where $q' = q^{1/s}$. The projection ensures that the resulting code inherits key properties of Goppa codes, such as a basis for error correction, while reducing the computational complexity associated with decoding.

The construction in [Loi01a] introduces $s$-projected Goppa codes and studies their algebraic properties, but it does not address proximity testing. In this work, we show how these codes can be used to construct an interactive oracle proof of proximity. The key observation is that membership in an $s$-projected Goppa code can be expressed by a the full parity-check relation. This allows the verifier to test whether a committed word belongs to the projected code without checking each Frobenius orbit separately.

**Use of structure.** The algebraic structure of the $s$-projected Goppa code is used to partition the support into Frobenius orbits and to derive the parity-check equations of the projected code. These parity-check equations form the basis of our proximity test and allow the verifier to certify code membership by verifying a single parity-check relation, rather than checking separate conditions for each Frobenius orbit.

Unlike McEliece-type cryptosystems, our construction does not rely on hiding the structure of a Goppa code. Instead, the code is public, and its algebraic structure is exploited to obtain an efficient interactive oracle proof of proximity. The soundness of the protocol follows from the correctness of the parity-check test, the soundness of the sum-check protocol, and the binding property of the polynomial commitment scheme.

**Polynomial Commitment Scheme.** A key building block of our IOPP construction is a polynomial commitment scheme, which allows the prover to commit succinctly to the multilinear extension of the projected word while allowing the verifier to query its value at a randomly chosen point. More precisely, let $\tilde{a} \in \mathbb{F}_2^N$ be the projected codeword indexed by the Frobenius orbit representatives, and let $\hat{w} : \{0,1\}^\ell \to \mathbb{F}$, with $\ell = \lceil \log_2 N \rceil$, be its multilinear extension over a sufficiently large field. The prover first computes a commitment

$\mathsf{com} = \mathsf{Commit}(\hat{w})$, which serves as the oracle accessed by the verifier throughout the protocol.

Finally, the verifier checks the single parity-check equation

$$\sum_{j=1}^{N} \widetilde{w}_j h_j(x) \equiv 0 \pmod{g(x)},$$

where $h_j(x)$ denotes the orbit syndrome polynomial associated with the $j$-th Frobenius orbit $\mathcal{O}_j$. The verifier first reduces this condition to a single randomly batched linear relation by sampling a random challenge and combining all syndrome equations into one. This relation is then verified using the sum-check protocol.

During the query phase, the verifier receives messages of the sum-check protocol. At the end of the protocol, the verifier obtains a random evaluation point $z \in \mathbb{F}^{\ell}$ and requests an opening of the commitment at $z$. The verifier accepts only if the sum-check protocol is successful and the polynomial commitment opening proves that the claimed value $\hat{w}(z)$ is consistent with the original commitment.

**Roadmap.** The structure of this paper is organized as follows. Section 2 introduces the definitions of all the tools and primitives utilized throughout the paper. Section 3 details the process of deriving $s$-projected codes from Goppa codes. Finally, Section 4 presents our construction of an interactive oracle proof of proximity, along with its security and complexity analysis.

## 2 Preliminaries

**Definition 1** (Interactive Oracle Proof [BCS16])**.** An interactive oracle proof for a relation $R$ with round complexity $\kappa : \{0,1\}^* \to \mathbb{N}$ and soundness $\varepsilon : \{0,1\}^* \to [0,1]$ is a pair $(\mathcal{P}, \mathcal{V})$ of probabilistic algorithms $\mathcal{P}$ and $\mathcal{V}$. Let $\mathsf{out}$ denote the output of the verifier. The IOP satisfies the following properties:

- *Completeness:* For every statement-witness pair $(x, w) \in R$:

$$\Pr[\mathsf{out}(\mathcal{P}(x, w), \mathcal{V}(x)) = \mathsf{accept}] = 1.$$

- *Soundness:* For every statement $x \notin \mathcal{L}(R)$ (where $\mathcal{L}$ is a language for the relation $R$) and a prover's strategy $\mathcal{P}^*$:

$$\Pr[\mathsf{out}(\mathcal{P}^*, \mathcal{V}(x)) = \mathsf{accept}] \leq \varepsilon,$$

  where $\varepsilon$ is called *soundness error*.

**Definition 2** (Interactive Oracle Proof of Proximity (IOPP) [BCF+16])**.** An $r$-round Interactive Oracle Proof of Proximity (IOPP) is a tuple $(\mathcal{P}, \mathcal{V})$ which can be seen as a $(r+1)$-round IOP. We say that an IOPP runs in $r$ rounds for the error correcting code $\mathscr{C} = \{f : S \to \Sigma\}$, with soundness $\varepsilon : (0,1] \to [0,1]$, with respect to distance measure $\Delta$, and $S$ being a finite set, if the following conditions hold:

- First message format: the first prover message, denoted $f^{(0)}$, is a purported codeword of $\mathscr{C}$, i.e., $f^{(0)} : S \to \Sigma$, where $S$ is a finite set and $\Sigma$ is an alphabet.

- Completeness: $\Pr[\langle \mathcal{P} \leftrightarrow \mathcal{V} \rangle = \mathsf{accept} | \Delta(f^{(0)}, \mathscr{C}) = 0)] = 1$; this means that for every $f^{(0)} \in \mathscr{C}$ the protocol terminates in acceptance.
- Soundness: For any $\mathcal{P}^*$, $\Pr[\langle \mathcal{P}^* \leftrightarrow \mathcal{V} \rangle = \mathsf{reject} | \Delta(f^{(0)}, \mathscr{C}) = \delta] \geq 1 - \varepsilon(\delta)$.

The total length of all prover messages, excluding $f^{(0)}$, is referred to as the IOPP proof length. The time required to generate all prover messages, apart from $f^{(0)}$, is defined as the prover complexity. The query complexity in an IOPP protocol is given by the total number of queries to all messages, including $f^{(0)}$.

An IOPP is a proof system proving that a committed vector over a given field $\mathbb{F}$ is close to a codeword of some error-correcting code $\mathscr{C} \subseteq \mathbb{F}$. IOPP protocols can be used to obtain efficient constructions of polynomial commitment schemes [KPV22, GLS+23, CBBZ23, BCHO22].

In the following definition we describe the univariate version of the polynomial commitment scheme given in [BFS20, GLS+23].

**Definition 3** (Polynomial Commitment Scheme)**.** A polynomial commitment scheme $PC$ over a finite field $\mathbb{F}$ is given by the following four algorithms: $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ :

- $\mathsf{Setup}(1^\lambda)$: On input a security parameter $1^\lambda$, outputs public parameters $\mathsf{pp}$.
- $\mathsf{Commit}(\mathsf{pp}, f)$: On input public parameters $\mathsf{pp}$ and a polynomial $f \in \mathbb{F}[x]$, outputs a commitment $\mathsf{com}$.
- $\mathsf{Open}(\mathsf{pp}, \mathsf{com}, f)$: On input public parameters $\mathsf{pp}$, a commitment $\mathsf{com}$ and a polynomial $f$, outputs a bit $b$.
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{com}, z, y; f)$: On input public parameters $\mathsf{pp}$, a commitment $\mathsf{com}$, an evaluation point $z \in \mathbb{F}$ and a value $y \in \mathbb{F}$, the prover wants to convince a verifier that $f$ is an opening to $\mathsf{com}$ and that $f(z) = y$. The verifier outputs a bit $b$ at the end of the protocol.

This polynomial commitment scheme satisfies the following three properties.

**Completeness:** $PC$ is complete if for any polynomial $f \in \mathbb{F}[x]$ and any point $z \in \mathbb{F}$ the following holds:

$$\Pr\left[\mathsf{Eval}(\mathsf{pp}, \mathsf{com}, z, y; f) = 1 \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{com} \leftarrow \mathsf{Commit}(\mathsf{pp}, f) \end{array}\right] = 1.$$

**Binding:** $PC$ is binding if for any PPT adversary $\mathcal{A}$ the following holds:

$$\Pr\left[b_0 = b_1 = 1 \wedge f_0 \neq f_1 \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f_1) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

us

**Knowledge Soundness:** $PC$ satisfies knowledge soundness if $\mathsf{Eval}$ is an argument of knowledge for the relation

$$R_{\mathsf{Eval},\mathsf{pp}} = \{[(\mathsf{com}, z, y); f] : f(z) = y \wedge \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f) = 1\},$$

where $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. That is, for any PPT adversary $\mathcal{A}$ and any malicious $\mathcal{P}^*$, there exists a polynomial-time extractor $\mathsf{Ext}$ s.t. for any randomness $r$, the following holds:

$$\Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)\\ (\mathsf{com}, z, y) \leftarrow \mathcal{A}(\mathsf{pp}, r)\\ \mathsf{Eval}(\mathsf{pp}, \mathsf{com}, z, y)\langle P^*(r), \mathcal{V}\rangle = 1\end{array}\right] \approx \Pr\left[\begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)\\ (\mathsf{com}, z, y) \leftarrow \mathcal{A}(\mathsf{pp}, r)\\ f \leftarrow \mathsf{Ext}(\mathsf{pp}, r)\\ ((\mathsf{com}, z, y); f) \in R_{\mathsf{Eval},\mathsf{pp}}\end{array}\right].$$

## 2.1 Linear Codes

A linear code of *length* $n$ and *dimension* $k$ is a vector subspace $\mathscr{C} \subset \mathbb{F}_q^n$. A basis for this vector space is typically expressed via a matrix $\mathbf{G}$, known as *generator matrix*, containing the basis vectors as rows, i.e. $\mathbf{G} \in \mathbb{F}_q^{k\times n}$. Note that, since the mapping $\mathbf{m} \mapsto \mathbf{c} = \mathbf{mG}$ is injective, we have that each code has $q^k$ codewords. Alternatively, the code can be defined as the kernel of a matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$, known as *parity-check matrix*, i.e. $\mathbf{c} \in \mathscr{C} \iff \mathbf{Hc}^T = 0$. The *Hamming weight* of a codeword corresponds to the number of its non-zero entries; we denote this by $\|\mathbf{c}\|$. This naturally induces a distance between codewords, calculated as the weight of their difference. Then, one can define the *minimum (Hamming) distance* of a code is the smallest number of different entries between any two codewords $\mathbf{c}_1, \mathbf{c}_2 \in \mathscr{C}$. A linear code $\mathscr{C}$ with minimum distance $d \in [0, n]$ is denoted by $[n, k, d]$ and $\Delta_{\mathscr{C}}$ is used to denote the fraction $d/n$, which is the relative minimum distance of the code.

**Definition 4** ($q$-ary Goppa Code)**.** Let $q$ be a prime power and let $\mathbb{F}_{q^m}/\mathbb{F}_q$ be a finite field extension. Let

$$L = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q^m}$$

be a set of distinct elements, and let $g(x) \in \mathbb{F}_{q^m}[x]$ be a polynomial of degree $t$ such that

$$g(\alpha_i) \neq 0 \qquad \text{for all } i = 1, \ldots, n.$$

The $q$-ary Goppa code associated with $L$ and $g$ is

$$\Gamma_q(L, g) = \left\{\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{F}_q^n : \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \equiv 0 \pmod{g(x)}\right\}.$$

The congruence is taken in $\mathbb{F}_{q^m}[x]/\langle g(x)\rangle$. Since $g(\alpha_i) \neq 0$, each $x - \alpha_i$ is invertible modulo $g(x)$.

Equivalently, $\Gamma_q(L, g)$ is characterized by the parity-check matrix

$$\mathbf{H} = \begin{pmatrix} \frac{1}{g(\alpha_1)} & \cdots & \frac{1}{g(\alpha_n)} \\ \frac{\alpha_1}{g(\alpha_1)} & \cdots & \frac{\alpha_n}{g(\alpha_n)} \\ \vdots & \ddots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{pmatrix} \in \mathbb{F}_{q^m}^{t\times n},$$

namely,

$$\mathbf{c} \in \Gamma_q(L, g) \iff \mathbf{Hc}^T = 0 \quad \text{over } \mathbb{F}_{q^m}.$$

In the binary case used later in this paper, we take $q = 2$. For simplicity, we denote the corresponding binary Goppa code by $\Gamma(L, g)$. Thus

$$\Gamma(L, g) \subseteq \mathbb{F}_2^n, \qquad L \subseteq \mathbb{F}_{2^m}.$$

For the $s$-projected construction, we further assume

$$g(x) \in \mathbb{F}_{2^s}[x], \qquad s \mid m.$$

Goppa codes are *alternant* codes, meaning that they can be seen as subfield subcodes[1] of Generalized Reed-Solomon codes. As such, they have access to generic decoders such as the famous Berlekamp-Massey [Ber68, Mas69]. Furthermore, *binary* Goppa codes, where the base field is $\mathbb{F}_2$, admit a specialized decoder, due to Patterson [Pat75], which exhibits nearly double the error correction capacity.

It is also possible to use *list decoders* for Goppa codes. A list decoder is a special type of algorithm which returns as output a (short) list of candidate codewords - hence the name. As stated in [ABC11], binary Goppa codes are list-decodable up to the binary Johnson bound.

# 3 Codes Derived from Binary Goppa Codes

In Section 3 of [Loi01a], Loidreau introduces $s$-projected binary codes derived from classical Goppa codes using Frobenius-invariant subcodes and orbit representatives. The approach involves projecting codes, derived from binary Goppa codes, called $s$-projected codes, to take advantage of their automorphism properties, thereby improving their effectiveness in error correction and cryptography. By projecting invariant code vectors through the Frobenius automorphism, this method reduces the code length while preserving key attributes such as dimension and minimum distance. This innovation simplifies decoding processes and enhances understanding of the structural properties of Goppa codes, offering potential for bolstering cryptographic systems like the McEliece cryptosystem against structural weaknesses.

The purpose of this section is twofold. First, we recall the construction of $s$-projected Goppa codes introduced by Loidreau, including the use of Frobenius automorphisms, invariant subcodes, orbit representatives, and trace-based parity-check matrices. These structural ingredients are not new and serve as the algebraic foundation for our construction. Second, we introduce the new results needed for our IOPP application. In particular, Theorem 2 gives an exact orbit-count formula for the Frobenius action in a power-of-two extension setting, improving on the approximate length estimate

$$N \approx \frac{s}{m} n$$

used in Loidreau's work. Corollary 1 then shows how the number of orbits changes monotonically with the projection parameter. Finally, Section 3.2 provides a bit-operation complexity analysis for constructing the $s$-projected Goppa code and its trace parity-check matrix. These results are used later in the complexity bounds for the IOPP.

[1] This construction simply restricts the code to codewords having elements on the base field only.

**What is recalled and what is new.** The construction of $s$-projected Goppa codes follows Loidreau [Loi01b, Loi01a]. To make the contribution of this section explicit, Table 1 separates the recalled background from the new ingredients used in our IOPP construction.

**Table 1:** Comparison between the results recalled from Loidreau and the ingredients added or adapted in this paper.

| Recalled from Loidreau | New or adapted in this paper |
|---|---|
| Frobenius-invariant subcodes of binary Goppa codes | Adapted to the notation and parameter regime needed for the IOPP construction. |
| Projection onto representatives of Frobenius orbits | Used as the reduced evaluation domain for the IOPP protocol. |
| Trace-based parity-check matrix for the projected code | Used to derive an explicit preprocessing and matrix-construction cost. |
| Approximate projected length $N \approx \frac{s}{m}n$ | Introduced an exact orbit-count formula using Burnside's lemma. |
| General structural properties of $s$-projected codes | A monotonicity consequence is derived to guide the choice of the projection parameter $s$. |

## 3.1 Construction of $s$-Projected Codes

The Frobenius automorphism is a fundamental concept in finite-field theory and plays a central role in the construction of $s$-projected codes. We distinguish the absolute Frobenius

$$\phi : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, \qquad \phi(x) = x^2,$$

from the relative Frobenius used in the $s$-projected construction. For a positive integer $s$ with $s \mid m$, the $s$-projected construction uses the Frobenius automorphism associated with the extension $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$, defined as follows.

**Definition 5.** The Frobenius automorphism associated with the field extension $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$ is the map

$$\sigma : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, \qquad \beta \mapsto \beta^{2^s},$$

equivalently, $\sigma = \phi^s$. This map fixes every element of $\mathbb{F}_{2^s}$; that is, for all $\beta \in \mathbb{F}_{2^s}$, $\sigma(\beta) = \beta$.

For the construction of an $s$-projected code, let $\Gamma(L, g) \subseteq \mathbb{F}_2^n$ be a binary Goppa code with support

$$L = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}_{2^m}.$$

We assume that the Goppa polynomial satisfies

$$g(x) \in \mathbb{F}_{2^s}[x], \qquad g(\alpha_i) \neq 0 \quad \text{for all } i = 1, \ldots, n.$$

Since the coefficients of $g(x)$ lie in the fixed field of $\sigma$, the Frobenius action preserves the Goppa-code structure. Therefore, the action of $\langle\sigma\rangle$ partitions $L$ into Frobenius orbits, and the $s$-projected code is obtained by restricting invariant codewords to one representative from each orbit. In Lemma 1 it is shown that the automorphism group of a Goppa code includes the group generated by the Frobenius automorphism of the extension field $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$, i.e.

$$\langle\sigma\rangle \subset Aut(\Gamma(L, g)).$$

Let $\mathbf{a} = (a_{\alpha_1}, \ldots, a_{\alpha_n})$ be a codeword of $\Gamma(L, g)$ labeled by the elements of $L = \{\alpha_1, \ldots, \alpha_n\}$. The Frobenius automorphism acts on such a codeword as a permutation of its components as follows:

$$\sigma(\mathbf{a}) = (a_{\sigma^{-1}(\alpha_1)}, \ldots, a_{\sigma^{-1}(\alpha_n)}).$$

We now report some results from literature which will aid the reader understand some crucial properties of Goppa codes. In the interest of space, the proofs of these technical lemmas are omitted, and we point the reader to the source (indicated in the Lemma's statement) for further details.

**Lemma 1** (Lemma 1 of [Loi01a])**.** *Let $\Gamma(L, g)$ be a Goppa code over $\mathbb{F}_{2^m}$, where the Goppa polynomial $g(x)$ has coefficients in the subfield $\mathbb{F}_{2^s}$. Then, the code is invariant under the action of the Frobenius automorphism $\sigma : \beta \mapsto \beta^{2^s}$ associated with the field extension $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$. That is, if $\mathbf{a} = (a_{\alpha_1}, \ldots, a_{\alpha_n}) \in \Gamma(L, g)$, then the permuted vector*

$$\sigma(\mathbf{a}) = (a_{\sigma^{-1}(\alpha_1)}, \ldots, a_{\sigma^{-1}(\alpha_n)})$$

*also belongs to $\Gamma(L, g)$. This implies that the subgroup $\langle\sigma\rangle$ generated by the Frobenius automorphism is contained in the automorphism group of the code:*

$$\langle\sigma\rangle \subseteq Aut(\Gamma(L, g)).$$

Let $g(x) \in \mathbb{F}_{2^s}[x]$ be a polynomial. The set of its roots in $\mathbb{F}_{2^m}$ is *invariant* under the action of the group $\langle\sigma\rangle$ generated by the Frobenius automorphism $\sigma$, where $\sigma$ fixes $\mathbb{F}_{2^s}$. Explicitly, if $\beta \in \mathbb{F}_{2^m}$ satisfies $g(\beta) = 0$, then $g(\sigma(\beta)) = 0$ as well. Now, consider the set $L = \{\alpha_1, \ldots, \alpha_n\}$ consisting of the elements of $\mathbb{F}_{2^m}$ that are *not* roots of $g(x)$. Under the action of $\langle\sigma\rangle$, the set $L$ is partitioned into disjoint orbits of the form

$$\mathcal{O}_{\alpha_i} = \{\alpha_i, \sigma(\alpha_i), \ldots, \sigma^{|\mathcal{O}_{\alpha_i}|-1}(\alpha_i)\},$$

and thus $L$ can be expressed as a disjoint union of these orbits:

$$L = \bigcup_{i=1}^{N} \mathcal{O}_{\alpha_i},$$

where $N$ denotes the number of distinct orbits.

We next define the invariant code, consisting of all codewords that remain fixed under the Frobenius action.

**Definition 6.** The invariant code is the set of vectors of the Goppa code $\Gamma(L, g)$ that are invariant under the action of $\sigma$.

$$I(L, g) = \{\mathbf{a} \in \Gamma(L, g) | \sigma(\mathbf{a}) = \mathbf{a})\}.$$

**Definition 7** ($s$-Projected Goppa Code)**.** Let $\Gamma(L, g) \subseteq \mathbb{F}_2^n$ be a binary Goppa code with $L \subseteq \mathbb{F}_{2^m}$, $g(x) \in \mathbb{F}_{2^s}[x]$, and $s \mid m$. Let $\sigma : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ be the Frobenius automorphism defined by $\sigma(\beta) = \beta^{2^s}$. Let

$$I(L, g) = \{a \in \Gamma(L, g) : \sigma(a) = a\}$$

be the invariant subcode. Let

$$L = \bigcup_{j=1}^{N} \mathcal{O}_{o_j}$$

be the decomposition of $L$ into $\langle\sigma\rangle$-orbits, and let $R = \{o_1, \ldots, o_N\}$ be a set of orbit representatives. Define

$$P : \mathbb{F}_2^n \to \mathbb{F}_2^N, \qquad P(a) = (a_{o_1}, \ldots, a_{o_N}).$$

The $s$-projected Goppa code associated with $\Gamma(L, g)$ is

$$\widetilde{I}(R, g) := P(I(L, g)) = \{(a_{o_1}, \ldots, a_{o_N}) : a \in I(L, g)\} \subseteq \mathbb{F}_2^N.$$

Since every word in $I(L, g)$ is constant on each $\langle\sigma\rangle$-orbit, the projection map $P$ is injective on $I(L, g)$. Hence $\widetilde{I}(R, g)$ has the same dimension as $I(L, g)$, while its length is the number $N$ of Frobenius orbits. We next recall the classical binary case, corresponding to $s = 1$. In this setting, the relevant Frobenius map is the absolute Frobenius

$$\tau : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, \qquad \tau(x) = x^2.$$

The following orbit-support property, stated as Property 2 in Section 3 of [Loi01a], is recalled here in the binary setting $s = 1$.

**Lemma 2** ( [Loi01b])**.** *Let $\Gamma(L, g)$ be a binary Goppa code with $g(x) \in \mathbb{F}_2[x]$, and let $\tau : x \mapsto x^2$ be the Frobenius automorphism of $\mathbb{F}_{2^m}/\mathbb{F}_2$. For a codeword $a = (a_\alpha)_{\alpha \in L} \in \Gamma(L, g)$, the following are equivalent:*

1. *$a$ is invariant under the Frobenius action, that is,*

$$\tau(a) = a.$$

2. *The support of $a$ is a union of $\tau$-orbits in $L$.*

*Equivalently, $a_\alpha = a_\beta$ whenever $\alpha$ and $\beta$ lie in the same $\tau$-orbit.*

For the $s$-projected codes used in this paper, we work in the more general relative setting of Lemma 1, namely $g(x) \in \mathbb{F}_{2^s}[x]$ and $\sigma_s : x \mapsto x^{2^s}$. In this setting, the same orbit-constancy principle is applied with respect to the $\sigma_s$-orbits. Thus, if $a = (a_\alpha)_{\alpha \in L}$ is invariant under $\sigma_s$, then its coordinates are constant on the $\sigma_s$-orbits, and its support is a union of such orbits. The projected code is then obtained by selecting one representative from each $\sigma_s$-orbit.

For every $\beta \in \mathbb{F}_{2^m}$, $\mathbb{F}_2(\beta)$ denotes the smallest subfield of $\mathbb{F}_{2^m}$ containing $\beta$. Then the trace operator for $\beta$, denoted by $\mathsf{Tr}_\beta$ is defined as follows:

**Definition 8** (Trace map for the $s$-projected construction)**.** Let $\Gamma(L, g) \subseteq \mathbb{F}_2^n$ be a binary Goppa code with $L \subseteq \mathbb{F}_{2^m}$, $g(x) \in \mathbb{F}_{2^s}[x]$, and $s \mid m$. Let

$$\sigma : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}, \qquad \sigma(\beta) = \beta^{2^s}$$

be the Frobenius automorphism of $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$. For $\beta \in L$, let

$$\mathcal{O}_\beta = \{\beta, \sigma(\beta), \ldots, \sigma^{|\mathcal{O}_\beta|-1}(\beta)\}$$

be its orbit under $\langle\sigma\rangle$. The trace map associated with $\mathcal{O}_\beta$ is

$$\mathsf{Tr}_\beta : \mathbb{F}_2(\beta) \to \mathbb{F}_{2^s}, \qquad \mathsf{Tr}_\beta(\xi) = \sum_{i=0}^{|\mathcal{O}_\beta|-1} \sigma^i(\xi).$$

Thus $\mathsf{Tr}_\beta$ is the relative orbit trace to the fixed field $\mathbb{F}_{2^s}$. For $s = 1$, its image is $\mathbb{F}_2$.

An $s$-projected code is a subcode over the field $\mathbb{F}_2$ of a code over $\mathbb{F}_{2^s}$, where the parity-check matrix is directly derived from the parity-check matrix of the parent Goppa code. Lemma 3 says that by projecting the codewords of the invariant code $I(L, g)$ onto orbit representatives $R$, we obtain the projected invariant subcode $\tilde{I}(R, g)$, which maintains the same dimension but has a shorter length due to grouping elements within orbits. When $g$ is binary, the derived subcode $\tilde{I}(R, g)$ remains non-trivial, preserving useful algebraic properties.

We now recall the trace-based parity-check construction for $s$-projected Goppa codes given in Section 4 of [Loi01a]. In Lemma 3 below, the Frobenius map is the relative Frobenius $\sigma : x \mapsto x^{2^s}$. Hence the trace $\mathsf{Tr}_{o_j}$ has image in $\mathbb{F}_{2^s}$. When $s = 1$, this reduces to the usual trace into $\mathbb{F}_2$.

**Lemma 3** (Section 4 of [Loi01a])**.** *Let $\Gamma(L, g)$ be a Goppa code with Goppa polynomial $g(x) \in \mathbb{F}_{2^s}[x]$ of degree $t$. Let $R = \{o_1, \ldots, o_N\} \subset L$ be the set of orbit representatives. Then a parity-check matrix for the $s$-projected code $\widetilde{I}(R, g)$ is given with the entries as follows*

$$\mathbf{H}^{(\widetilde{I})}_{i,j} = \mathsf{Tr}_{o_j}\left(\frac{o_j^i}{g(o_j)}\right), \qquad 0 \le i < t, \quad 1 \le j \le N.$$

*Equivalently, the matrix can be written explicitly as*

$$\mathbf{H}^{(\widetilde{I})} = \begin{pmatrix} \mathsf{Tr}_{o_1}\left(\frac{1}{g(o_1)}\right) & \cdots & \mathsf{Tr}_{o_N}\left(\frac{1}{g(o_N)}\right) \\ \mathsf{Tr}_{o_1}\left(\frac{o_1}{g(o_1)}\right) & \cdots & \mathsf{Tr}_{o_N}\left(\frac{o_N}{g(o_N)}\right) \\ \vdots & \ddots & \vdots \\ \mathsf{Tr}_{o_1}\left(\frac{o_1^{t-1}}{g(o_1)}\right) & \cdots & \mathsf{Tr}_{o_N}\left(\frac{o_N^{t-1}}{g(o_N)}\right) \end{pmatrix}.$$

*This matrix characterizes the parity conditions defining the projected invariant subcode $\widetilde{I}(R, g)$, which has length $N$.*

Lemma 4 now gives the corresponding kernel characterization of the $s$-projected code using the parity-check matrix established in Lemma 3.

**Lemma 4** (Lemma 2 of [Loi01a])**.** *For an $s$-projected code $\tilde{I}(R, g)$ the codewords $\tilde{a}$ satisfy $\mathbf{H}^{(\tilde{I})}\tilde{\mathbf{a}} = 0$, i.e.:*

$$\tilde{I}(R, g) = \{\tilde{\mathbf{a}} = (a_{o_1}, \ldots, a_{o_N}) \in \mathbb{F}_2^N | \mathbf{H}^{(\tilde{I})}\tilde{\mathbf{a}} = 0\}.$$

**Structured Construction of $s$-Projected Goppa Codes.** The following is presented as an outline of the systematic procedure for constructing $s$-projected Goppa codes. The methodology is also formalized in Algorithm 1.

1. Begin by defining a binary Goppa code $\Gamma(L, g)$, where $g(x) \in \mathbb{F}_{2^s}[x]$ is a monic polynomial of degree $t$, and $L = \{\alpha_1, \ldots, \alpha_n\} \subset \mathbb{F}_{2^m}$ is a support set such that $g(\alpha_i) \neq 0$ for all $i$.

2. Introduce the Frobenius automorphism associated with the field extension $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$, defined by $\sigma(\beta) = \beta^{2^s}$ for all $\beta \in \mathbb{F}_{2^m}$.

3. Using this automorphism, define the invariant subcode $I(L, g)$ by Definition 6.

4. The group $\langle\sigma\rangle$ acts on $L$, partitioning it into disjoint orbits of the form $\mathcal{O}_{o_i} = \{o_i, \sigma(o_i), \ldots, \sigma^{|\mathcal{O}_{o_i}|-1}(o_i)\}$. Consequently, the set $L$ can be written as:

$$L = \bigcup_{i=1}^{N} \mathcal{O}_{o_i},$$

   where $N$ is the number of such orbits.

5. Select a set of orbit representatives $R = \{o_1, \ldots, o_N\} \subset L$, where each $o_i = \alpha_j$ for some $j \in \{1, \ldots, n\}$.

6. Define the projection map[2] $P : \mathbb{F}_2^n \to \mathbb{F}_2^N$, which extracts from a word indexed by $L$ the coordinates indexed by $R$:

$$P(\mathbf{a}) = (a_{o_1}, \ldots, a_{o_N}), \quad \text{for } \mathbf{a} = (a_{\alpha_1}, \ldots, a_{\alpha_n}).$$

7. Applying $P$ to each word in the invariant subcode $I(L, g)$ yields the $s$-projected code:

$$\tilde{I}(R, g) := \{P(\mathbf{a}) \mid \mathbf{a} \in I(L, g)\}.$$

   This code has length $N$ and the same dimension as $I(L, g)$.

   It is referred to as the *$s$-projected code* derived from the Goppa code $\Gamma(L, g)$.

8. Every word $\tilde{\mathbf{a}} \in \tilde{I}(R, g)$ uniquely corresponds to a word $\mathbf{a} \in I(L, g)$, via the bijective projection and its inverse.

9. The parity-check matrix of the $s$-projected code can be constructed using trace evaluations. Specifically, each entry of the matrix is given by:

$$\mathsf{Tr}_{o_i}\left(\frac{o_i^j}{g(o_i)}\right), \quad \text{for } i = 1, \ldots, N,\ j = 0, \ldots, t-1,$$

   where $\mathsf{Tr}_{o_i}$ denotes the trace from $\mathbb{F}_{2^m}$ to $\mathbb{F}_{2^s}$, defined by:

$$\mathsf{Tr}_{o_i}(\beta) = \sum_{k=0}^{|\mathcal{O}_{o_i}|-1} \sigma^k(\beta).$$

**Algorithm 1** Generation of $s$-projected code from a Goppa Code

**Input:** Goppa polynomial $g(x) \in \mathbb{F}_{2^s}[x]$, generating set $L \subset \mathbb{F}_{2^m}$, degree $t$ of $g(x)$
**Output:** $s$-projected code $\tilde{I}(R, g)$

**Initialize**
Define the Frobenius automorphism $\sigma : \beta \mapsto \beta^{2^s}$.
Partition $L$ into orbits $\mathcal{O} = \{\mathcal{O}_{\alpha_1}, \mathcal{O}_{\alpha_2}, \ldots, \mathcal{O}_{\alpha_N}\}$ under the action of $\sigma$.
**Compute Invariant Subcode**
Define $I(L, g) \subseteq \Gamma(L, g)$ as the set of vectors $\mathbf{a} = (a_{\alpha_1}, \ldots, a_{\alpha_n})$ such that $\sigma(\mathbf{a}) = \mathbf{a}$.
**Choose Representatives**
**for** each orbit $\mathcal{O}_{\alpha_i} \in \mathcal{O}$ **do**
    Select a representative $o_i \in \mathcal{O}_{\alpha_i}$.
Construct the set $R = \{o_1, o_2, \ldots, o_N\}$, where $N = |\mathcal{O}|$.
**Construct $s$-projected code**
**for** each vector $\mathbf{a} \in I(L, g)$ **do**
    Project the coordinates of $a$ indexed by $L$ to the coordinates indexed by $R$:
    For each $\mathcal{O}_{o_i}$, map all elements of $\mathcal{O}_{o_i}$ to their representative $o_i$ in $R$.
    Create the projected vector $\tilde{\mathbf{a}} = (a_{o_1}, a_{o_2}, \ldots, a_{o_N})$.
**Generate Parity-Check Matrix**
Construct the parity-check matrix $\mathbf{H}^{(\tilde{I})}$ for $\tilde{I}(R, g)$ using the trace operator:
**for** each $o_i \in R$ **do**
    **for** $j = 0$ to $t - 1$ **do**
        Compute trace terms $\mathsf{Tr}\left(\frac{o_i^j}{g(o_i)}\right)$.
**Return**
The linear $s$-projected code $\tilde{I}(R, g)$ with its parity-check matrix $\mathbf{H}^{(\tilde{I})}$.

We next analyze how the projection parameter affects the length of the $s$-projected code. Since the projected code is indexed by orbit representatives, its length is determined by the number of Frobenius orbits in the support set. We therefore first record the order of the relative Frobenius automorphism and the size of the fixed-point sets of its powers.

**Lemma 5.** *Let $\phi$ be the Frobenius automorphism on the finite field $\mathbb{F}_{2^m}$ and $\sigma = \phi^s$ be the Frobenius automorphism associated with the field extension $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$. Then the order of $\sigma$, denoted by $|\sigma|$, is given by*

$$|\sigma| = \frac{m}{\gcd(m, s)}.$$

**Definition 9.** Let $G$ be a finite group that acts on a finite set $X$. The set of elements of $x$ fixed by $g \in G$ (also known as fixed-point set) is defined as $Fix(g) := \{x \in X : g \cdot x = x\}$.

The following lemma determines the number of elements in the fixed-point set of the Frobenius automorphism. Specifically, it characterizes the structure of the fixed-point set and provides a precise count of its elements, which plays a crucial role in understanding the length of the obtained $s$-projected code with respect to the projection factor $s$.

**Lemma 6.** *Let $\sigma : x \to x^{2^s}$ be the Frobenius automorphism on $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$. Then $|Fix(\sigma^j)| = 2^{\gcd(sj,m)}$ where $j = 1, 2, \ldots, |\sigma| - 1$*

[2]The inverse projection $P^{-1}(\mathbf{b})$, for $\mathbf{b} = (b_{o_1}, \ldots, b_{o_N}) \in \mathbb{F}_2^N$, reconstructs an invariant word by replicating each symbol $b_{o_i}$ to all positions in its corresponding orbit $\mathcal{O}_{o_i}$. Formally $P(P^{-1}(\mathbf{b})) = \mathbf{b}$.

We now recall Burnside's lemma, which expresses the number of orbits of a finite group action in terms of fixed-point sets. This result will be applied to the action of the Frobenius subgroup $\langle\sigma\rangle$ on $\mathbb{F}_{2^m}$ in order to compute the number of orbit representatives appearing in the $s$-projected code.

**Theorem 1.** ***Burnside's Lemma (Orbit Counting Theorem)*** ***[Bur97]*** *Let $G$ be a finite group that acts on a finite set $X$. The number of distinct orbits of $X$ under the action of $G$ is given by:*

$$Orb_G(X) = \frac{1}{|G|} \sum_{g \in G} |Fix(g)|,$$

*where:*

- *$Orb_G(X)$ is the number of distinct orbits.*
- *$|G|$ is the order (size) of the group.*
- *$Fix(g)$ is the set of elements in $X$ that are fixed by $g$, i.e.,*

$$Fix(g) = \{x \in X \mid g \cdot x = x\}.$$

Theorem 2 provides an exact orbit count in our setting, obtained by applying Burnside's lemma together with the fixed-point count from Lemma 6. Corollary 1 then gives a monotonicity consequence by showing how the number of orbits varies with the subfield parameter.

Recall that the Frobenius automorphism $\sigma : x \mapsto x^{2^s}$ acts on $\mathbb{F}_{2^m}$ by mapping elements to their $2^s$-th powers. The group $\langle\sigma\rangle$ generated by this automorphism partitions $\mathbb{F}_{2^m}$ into orbits under repeated application of $\sigma$. Loidreau's construction uses the approximation

$$N \approx \frac{s}{m} n$$

for the number of Frobenius orbits. For the purposes of our IOPP complexity analysis, however, we need a more explicit relationship between the projection parameter and the number of orbit representatives. The next theorem provides such an exact orbit count in the case where $m$ and $s$ are powers of two. This is one of the new technical ingredients of the paper.

**Theorem 2.** *Let $\mathbb{F}_{2^m}/\mathbb{F}_{2^s}$ be a finite field extension, where $m = 2^{m'}$ and $s = 2^{s'}$, with $s' < m'$. Let $\phi : x \mapsto x^2$ be the Frobenius automorphism on $\mathbb{F}_{2^m}$, and let $\sigma = \phi^s$. Then*

$$Orb_{\langle\sigma\rangle}(\mathbb{F}_{2^m}) = 2^{m-m'+s'} + 2^{2^{s'}-1} + \frac{1}{2} \sum_{k=1}^{m'-s'-1} 2^{2^{s'+k}-k}.$$

*Proof.* By Lemma 5,

$$|\langle\phi^s\rangle| = 2^{m'-s'}.$$

Hence, by Burnside's lemma,

$$Orb_{\langle\phi^s\rangle}(\mathbb{F}_{2^m}) = \frac{1}{2^{m'-s'}} \left( 2^m + \sum_{j=1}^{2^{m'-s'}-1} 2^{\gcd(sj,m)} \right),$$

where the term $2^m$ corresponds to $j = 0$, i.e., the identity automorphism. Its contribution is therefore

$$\frac{2^m}{2^{m'-s'}} = 2^{m-m'+s'}.$$

For odd $j$, $\gcd(sj, m) = 2^{s'}$, and there are $2^{m'-s'-1}$ such values. Hence their total contribution is

$$\frac{2^{m'-s'-1}2^{2^{s'}}}{2^{m'-s'}} = 2^{2^{s'}-1}.$$

For even $j$, write $j = 2^k\ell$, where $\ell$ is odd and $1 \le k \le m' - s' - 1$. For each fixed $k$, there are $2^{m'-s'-k-1}$ such integers $j$, and

$$\gcd(sj, m) = 2^{s'+k}.$$

Thus the even terms contribute

$$\frac{1}{2^{m'-s'}} \sum_{k=1}^{m'-s'-1} 2^{m'-s'-k-1}2^{2^{s'+k}} = \frac{1}{2} \sum_{k=1}^{m'-s'-1} 2^{2^{s'+k}-k}.$$

Combining the identity, odd, and even contributions gives

$$Orb_{\langle\phi^s\rangle}(\mathbb{F}_{2^m}) = 2^{m-m'+s'} + 2^{2^{s'}-1} + \frac{1}{2} \sum_{k=1}^{m'-s'-1} 2^{2^{s'+k}-k}.$$

Since $\sigma = \phi^s$, the result follows. □

This exact orbit count is useful because the number of orbits determines the length of the projected code, the number of columns in the projected parity-check matrix, and the number of orbit representatives used in the IOPP protocol. The following corollary gives a useful monotonicity property of the orbit count: choosing a smaller subfield parameter produces fewer orbits, and hence a shorter projected code. This monotonicity result is useful when selecting projection parameters for the IOPP, because the orbit count directly affects the verifier's query complexity and the size of the projected parity-check matrix.

**Corollary 1.** *Let $\mathbb{F}_{2^m}/\mathbb{F}_{2^{s_1}}$ and $\mathbb{F}_{2^m}/\mathbb{F}_{2^{s_2}}$ be two finite field extensions, where $m = 2^{m'}$, $s_1 = 2^{s_1'}$, and $s_2 = 2^{s_2'}$, with $s_2' < s_1' < m'$. Then*

$$Orb_{\langle\phi^{s_2}\rangle}(\mathbb{F}_{2^m}) < Orb_{\langle\phi^{s_1}\rangle}(\mathbb{F}_{2^m}).$$

*Proof.* From Theorem 2, for $s = 2^{s'}$,

$$Orb_{\langle\phi^s\rangle}(\mathbb{F}_{2^m}) = 2^{m-m'+s'} + 2^{2^{s'}-1} + \frac{1}{2} \sum_{k=1}^{m'-s'-1} 2^{2^{s'+k}-k}.$$

Define

$$S(t) := 2^{m-m'+t} + 2^{2^t-1} + \frac{1}{2} \sum_{k=1}^{m'-t-1} 2^{2^{t+k}-k}.$$

Then

$$Orb_{\langle\phi^{s_i}\rangle}(\mathbb{F}_{2^m}) = S(s_i'), \qquad i \in \{1, 2\}.$$

It therefore suffices to show that $S(t)$ is strictly increasing.

Let

$$A_t := 2^{m-m'+t} + 2^{2^t-1}, \qquad a_{t,k} := 2^{2^{t+k}-k}.$$

For $1 \leq t \leq m' - 3$,

$$S(t) = A_t + \frac{1}{2} \sum_{k=1}^{m'-t-1} a_{t,k},$$

while, since

$$2^{2^{t+k+1}-k} = 2a_{t,k+1},$$

we have

$$S(t+1) = A_{t+1} + \sum_{j=2}^{m'-t-1} a_{t,j}.$$

Hence

$$S(t+1) - S(t) = (A_{t+1} - A_t) + \frac{1}{2} \left( \sum_{j=2}^{m'-t-1} a_{t,j} - a_{t,1} \right) > 0,$$

because $A_{t+1} > A_t$ and

$$a_{t,2} > a_{t,1}.$$

For the endpoint $t = m' - 2$, direct substitution gives

$$S(t+1) - S(t) = 2^{m-m'+t} + 2^{2^{t+1}-2} - 2^{2^t-1} > 0.$$

Thus $S(t)$ is strictly increasing. Since $s'_2 < s'_1$, we obtain

$$Orb_{\langle \phi^{s_2} \rangle}(\mathbb{F}_{2^m}) < Orb_{\langle \phi^{s_1} \rangle}(\mathbb{F}_{2^m}).$$

□

The following example will present a detailed, step-by-step explanation of the construction process for an $s$-projected code derived from a given Goppa code $\Gamma(L, g)$. We describe the projection mechanism, which systematically reduces the dimensionality of the code while preserving its key structural properties. Specifically, we define the selection criteria for the smaller evaluation set and analyze how this transformation affects the generator matrix.

**A Worked Example of a 2-Projected Goppa Code.**

**Example 1.** Let $\sigma : x \to x^4$ be the Frobenius automorphism on $\mathbb{F}_{2^4}$ over $\mathbb{F}_{2^2}$. Let $g(x) = x^3 + x + 1$ be the Goppa polynomial and $g(\beta) \neq 0$ for all $\beta \in \mathbb{F}_{2^2}$. Let $\alpha$ be the primitive element of $\mathbb{F}_{2^4}$ with $\alpha^4 + \alpha + 1 = 0$. The group generated by $\langle \sigma \rangle$ acts on $\mathbb{F}_{2^4} = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{14}\}$. The orbits generated by this action are the following:

$$\{0\}, \{1\}, \{\alpha, \alpha^4\}, \{\alpha^2, \alpha^8\}, \{\alpha^3, \alpha^{12}\}, \{\alpha^5\}, \{\alpha^6, \alpha^9\}, \{\alpha^{10}\}, \{\alpha^7, \alpha^{13}\}, \{\alpha^{11}, \alpha^{14}\} \quad (1)$$

Now let us compute the evaluation of the representatives of the orbits on $g(x) = x^3 + x + 1$ as follows

$$\begin{aligned} &g(0) = 1,\ g(1) = 1,\ g(\alpha) = \alpha^7,\ g(\alpha^2) = \alpha^{14},\ g(\alpha^3) = \alpha^4,\ g(\alpha^5) = \alpha^5 \\ &g(\alpha^6) = \alpha^8,\ g(\alpha^7) = \alpha^5,\ g(\alpha^{10}) = \alpha^{10},\ g(\alpha^{11}) = \alpha^{10} \end{aligned} \quad (2)$$

For each $\beta \in \mathbb{F}_{2^4}$, define the orbit trace

$$\mathsf{Tr}_\beta : \mathbb{F}_{2^2}(\beta) \longrightarrow \mathbb{F}_{2^2}, \qquad \mathsf{Tr}_\beta(y) = \sum_{i=0}^{|\mathcal{O}_\beta|-1} \sigma^i(y).$$

In the following table we computed all the entries of the parity check matrix for the 2-projected code. The first row computes the quotient $\frac{o_i^j}{g(o_i)}$ where $o_i$ is the representative of each of the 10 orbits that we derived in (1) with $i = 1, 2, \ldots, 10$ and $j = 0, 1, 2$. In the second row we calculate $\frac{o_i^j}{g(o_i)}$ as a power of $\alpha$ that can be obtained by multiplying $o_i^j$ and $g(o_i)^{-1}$ deduced with the help of the evaluated values from (2). Thus, the last row gives us the trace of each of these quotient values.

**Table 2:** The entries of the parity check matrix

| $\frac{o_i^j}{g(o_i)}$ | $\alpha^i$ | $\mathsf{Tr}_{o_i}(\frac{o_i^j}{g(o_i)})$ | $\frac{o_i^j}{g(o_i)}$ | $\alpha^i$ | $\mathsf{Tr}_{o_i}(\frac{o_i^j}{g(o_i)})$ |
|---|---|---|---|---|---|
| $\frac{1}{g(0)}$ | 1 | 1 | $\frac{\alpha^{10}}{g(\alpha^5)}$ | $\alpha^5$ | $\alpha^5$ |
| $\frac{1}{g(1)}$ | 1 | 1 | $\frac{1}{g(\alpha^{10})}$ | $\alpha^5$ | $\alpha^5$ |
| $\frac{1}{g(\alpha)}$ | $\alpha^8$ | 1 | $\frac{\alpha^{10}}{g(\alpha^{10})}$ | 1 | 1 |
| $\frac{\alpha}{g(\alpha)}$ | $\alpha^9$ | $\alpha^5$ | $\frac{\alpha^{20}}{g(\alpha^{10})}$ | $\alpha^{10}$ | $\alpha^{10}$ |
| $\frac{\alpha^2}{g(\alpha)}$ | $\alpha^{10}$ | $\alpha^{10}$ | $\frac{1}{g(\alpha^{11})}$ | $\alpha^5$ | $\alpha^5$ |
| $\frac{1}{g(\alpha^2)}$ | $\alpha$ | 1 | $\frac{\alpha^{11}}{g(\alpha^{11})}$ | $\alpha$ | 1 |
| $\frac{\alpha^2}{g(\alpha^2)}$ | $\alpha^3$ | $\alpha^{10}$ | $\frac{\alpha^{22}}{g(\alpha^{11})}$ | $\alpha^{12}$ | $\alpha^{10}$ |
| $\frac{\alpha^4}{g(\alpha^2)}$ | $\alpha^5$ | $\alpha^5$ | $\frac{1}{g(\alpha^6)}$ | $\alpha^7$ | $\alpha^5$ |
| $\frac{1}{g(\alpha^3)}$ | $\alpha^{11}$ | $\alpha^{10}$ | $\frac{\alpha^6}{g(\alpha^6)}$ | $\alpha^{13}$ | $\alpha^5$ |
| $\frac{\alpha^3}{g(\alpha^3)}$ | $\alpha^{14}$ | $\alpha^{10}$ | $\frac{\alpha^{12}}{g(\alpha^6)}$ | $\alpha^{10}$ | $\alpha^{10}$ |
| $\frac{\alpha^6}{g(\alpha^3)}$ | $\alpha^2$ | 1 | $\frac{1}{g(\alpha^7)}$ | $\alpha^{10}$ | $\alpha^{10}$ |
| $\frac{1}{g(\alpha^5)}$ | $\alpha^{10}$ | $\alpha^{10}$ | $\frac{\alpha^7}{g(\alpha^7)}$ | $\alpha^{17}$ | 1 |
| $\frac{\alpha^5}{g(\alpha^5)}$ | 1 | 1 | $\frac{\alpha^{14}}{g(\alpha^7)}$ | $\alpha^9$ | $\alpha^5$ |

Now from Lemma 3 and Table 2 we deduce the parity check matrix for the 2-projected code as follows:

$$\mathbf{H}^{(\tilde{I})} = \begin{pmatrix} 1 & 1 & 1 & 1 & \alpha^{10} & \alpha^{10} & \alpha^5 & \alpha^5 & \alpha^5 & \alpha^{10} \\ 0 & 1 & \alpha^5 & \alpha^{10} & \alpha^{10} & 1 & 1 & 1 & \alpha^5 & 1 \\ 0 & 1 & \alpha^{10} & \alpha^5 & 1 & \alpha^5 & \alpha^{10} & \alpha^{10} & \alpha^{10} & \alpha^5 \end{pmatrix}$$

This matrix is then used to generate $s$-projected codewords.

## 3.2 Complexity of Constructing the $s$-Projected Goppa Code

The structural construction of $s$-projected Goppa codes follows Sections 3–4 of [Loi01a]. However, for our IOPP application, it is not sufficient to know only that the projected code exists, we must also quantify the cost of constructing the orbit decomposition, evaluating the Goppa polynomial at orbit representatives, computing trace entries, and assembling the projected parity-check matrix. This subsection provides that complexity analysis. It is later used as the preprocessing term $T_{\text{proj}}(n, t, m)$ in the IOPP complexity bounds.

This subsection presents a detailed study of the computational complexity involved in constructing the $s$-projected Goppa code $\widetilde{I}(R, g)$ from a classical Goppa code $\Gamma(L, g)$ over $\mathbb{F}_{2^m}$. The following notation is used:

- $n = |L|$ denotes the length of the original code (number of support elements),
- $t = \deg g$ denotes the degree of the Goppa polynomial,
- $m$ denotes the extension degree of the field $\mathbb{F}_{2^m}$,
- $s$ denotes the subfield degree for the Frobenius automorphism, and
- $N \approx \frac{s}{m}\, n$ denotes the number of Frobenius orbits of $L$.

The construction consists of five main phases: orbit finding, polynomial evaluation at orbit representatives, field inversion and multiplication, trace computation, and matrix assembly. Each phase is analyzed in turn, expressing time complexities in bit operations, assuming the usual cost of a field multiplication in $\mathbb{F}_{2^m}$ is $O(m^2)$ and that of a squaring (and XOR) is $O(m)$. Whenever possible, intermediate results are reused to minimize redundant work.

**a) Orbit Finding.**

The support set $L$ is partitioned into orbits under the Frobenius map

$$\sigma : x \longmapsto x^{2^s}.$$

Starting from an unvisited element $\alpha \in L$, the algorithm performs:

1. Initialize an empty list $\mathcal{O}$ for the current orbit.
2. Repeat:
   - Append $\alpha$ to $\mathcal{O}$ and mark $\alpha$ as visited.
   - Compute $\alpha \leftarrow \sigma(\alpha)$ by performing $s$ squarings in $\mathbb{F}_{2^m}$ (total cost $O(m)$ bit operations).
3. Until the new $\alpha$ is already marked visited, indicating the cycle is complete.
4. Store $\mathcal{O}$ and its first element as the orbit representative.

Every element of $L$ is visited exactly once in this process, and there is one additional $\sigma$-call per orbit to detect the closure. Therefore the number of Frobenius steps is

$$\sum_{j=1}^{N} (|\mathcal{O}_j| + 1) \;=\; n + N \leq 2n.$$

Since each Frobenius step costs $O(m)$ bit operations, the total cost of orbit finding is

$$O\big(n\, m\big) \quad \text{bit operations.}$$

A boolean visited array of size $n$ is maintained to ensure already processed elements are skipped in $O(1)$ time.

**b) Polynomial Evaluation at Orbit Representatives.**

Let $R = \{o_1, \ldots, o_N\}$ be the set of orbit representatives. Each evaluation of

$$g(o_j) = \sum_{i=0}^{t} a_i \, o_j^i,$$

with $a_i \in \mathbb{F}_{2^s} \subset \mathbb{F}_{2^m}$, is performed using Horner's rule:

$$g(o_j) = (((a_t \, o_j + a_{t-1}) \, o_j + a_{t-2}) \cdots) o_j + a_0.$$

This requires exactly $t$ multiplications and $t$ additions in $\mathbb{F}_{2^m}$. A field multiplication costs $O(m^2)$ bit operations, whereas an addition (XOR) costs $O(m)$. Therefore, each evaluation costs

$$O(t\,m^2) + O(t\,m) = O(t\,m^2).$$

Repeating for all $N$ representatives gives

$$O\big(N\,t\,m^2\big) = O\big(n\,t\,m^2\big)$$

bit operations. The computed values $g(o_j)$ are cached for use in the inversion step.

**c) Field Inversion and Multiplication.**

For each representative $o_j$, the multiplicative inverse $g(o_j)^{-1}$ in $\mathbb{F}_{2^m}$ is computed. For modular inversion a fast extended GCD performs one inversion in $O(m^2)$ bit operations. A table of powers

$$o_j^0, o_j^1, \ldots, o_j^{t-1}$$

is then constructed by successive multiplication ($t-1$ multiplications), each costing $O(m^2)$. Finally, for each exponent $i$, the value $v_{j,i} = o_j^i \cdot g(o_j)^{-1}$ is computed, requiring one further multiplication. Summing these costs yields per-representative work of

$$O(m^2)_{\text{inv}} + O(t\,m^2)_{\text{powers}} + O(t\,m^2)_{\text{mults}} = O(t\,m^2),$$

and across all representatives:

$$O\big(N\,t\,m^2\big) = O\big(n\,t\,m^2\big).$$

**d) Trace Computation.**

For each $v_{j,i}$, the orbit-trace is computed as $\mathrm{Tr}_j(v_{j,i}) = \sum_{k=0}^{|O_j|-1} v_{j,i}^{2^{sk}}$. The Frobenius conjugates are generated via repeated squaring ($O(m)$ each), and each is XORed into an accumulator ($O(m)$). Since $\sum_j |O_j| = n$, and there are $t$ traces per orbit,

$$\sum_{j=1}^{N} \sum_{i=0}^{t-1} O(|\mathcal{O}_j|\, m) = O(n\,t\,m)$$

bit operations. Previously computed Frobenius powers are reused to minimize overhead.

**e) Parity-Check Matrix Assembly.**

The $t \times N$ matrix $\mathbf{H}^{(I)}$ over $\mathbb{F}_{2^s}$ is assembled by storing each trace entry in its cell $O(tN)$ constant-time symbol writes. When a binary matrix is needed, each of the $tN$ symbols (an $s$-bit word) is unpacked into $s$ bits, yielding a $(t\,s) \times N$ matrix. This step incurs a cost of

$$O(tN\,s) = O(n\,t\,m)$$

bit writes, which remains lower order compared to $O(n\,t\,m^2)$. Efficient data layout strategies (e.g., row-major or column-major) ensure fast syndrome computations during decoding.

**Overall Complexity.** Adding all dominant terms yields a total cost in bit operations of

$$O(n\,m) + 2\,O(n\,t\,m^2) + O(n\,t\,m) + O(n\,t\,m) = O(n\,t\,m^2).$$

Thus, constructing the $s$-projected code $\widetilde{I}(R, g)$ requires asymptotically the same computational effort as constructing a classical Goppa code, while offering advantages such as reduced code length $N \approx \frac{s}{m}n$, smaller parity-check matrices, and improved performance for operations over $\mathbb{F}_{2^s}$. We denote this one-time preprocessing cost by

$$T_{\text{proj}}(n, t, m) := O(ntm^2).$$

It includes the orbit decomposition, evaluation and inversion of the Goppa polynomial, trace computations, and assembly of the projected parity-check data.

# 4 IOPP from Goppa Codes

We begin by outlining the construction of our Interactive Oracle Proof of Proximity (IOPP) based on Goppa codes. The input to the protocol is a vector in $\mathbb{F}_q^N$, where $\mathbb{F}_q$ is a finite field over which the Goppa code is defined. Let $\Gamma(L, g)$ be a Goppa code, as defined in Section 2.1. This code is specified by a Goppa polynomial $g(x)$ and an evaluation set $L \subseteq \mathbb{F}_q$. A Goppa code encodes polynomials under constraints imposed by $g(x)$, ensuring that the resulting codeword provides algebraic structure beneficial for error correction.

**Proximity Testing.** The verifier aims to determine whether a received codeword $\tilde{a}$ is close to a valid codeword $c$ in a Goppa code $\Gamma(L, g)$. To facilitate this, the prover provides an oracle that allows the verifier to query specific portions of $\tilde{a}$, the corresponding codeword $c$, and additional relevant data. The received codeword can be represented by a polynomial $w$, and the prover's main goal is to prove that the distance $\Delta(\tilde{w}, \Gamma(L, g)) \leq \delta N$ with $N = |L|$. The IOPP protocol consists of the following two phases:

**Commit Phase:** The prover commits to the encoded information, ensuring that the verifier has a reference for structured queries.

**Query Phase:** The verifier performs structured queries to validate the proximity of the received word to the Goppa code.

## 4.1 IOPP Protocol

In this section we provide the full description of the IOPP protocol, consisting of a commit and a query phase. Let $\tilde{I}(R,g) \subseteq \mathbb{F}_2^N$ be an $s-$projected Goppa code which is obtained from the Frobenius orbit decomposition $L = \bigsqcup_{j=1}^{N} \mathcal{O}_j$ with orbit representatives $R = \{o_1, \ldots, o_N\}$. For each orbit $\mathcal{O}_j$ we define the orbit syndrome polynomial $h_j(x) = \sum_{\beta \in O_j} \frac{1}{x-\beta} \pmod{g(x)}$. and for a word $\tilde{w} = (\tilde{w}_1, \ldots, \tilde{w}_N) \in \mathbb{F}_2^N$, we define the parity-check equation $\Phi_{\tilde{w}}(x) = \sum_{j=1}^{N} \tilde{w}_j h_j(x) \pmod{g(x)}$. Then we have the following equivalence

$$\tilde{w} \in \tilde{I}(R,g) \quad \Longleftrightarrow \quad \Phi_{\tilde{w}}(x) \equiv 0 \pmod{g(x)}.$$

We note that the verifier does not check the orbit-wise conditions $\tilde{w}_j h_j(x) \equiv 0 \pmod{g(x)}$ for each index $j$. These conditions are too strong and are false for general $s$-projected codewords. The verifier checks only the single parity-check condition

$$\sum_{j=1}^{N} \tilde{w}_j h_j(x) \equiv 0 \pmod{g(x)}.$$

**Public input.** The verifier receives the public description of the projected code:

$$g(x), \quad R = \{o_1, \ldots, o_N\}, \quad \mathcal{O}_1, \ldots, \mathcal{O}_N,$$

together with the precomputed orbit syndrome polynomials $h_1(x), \ldots, h_N(x)$.

Equivalently, the verifier receives the projected parity-check matrix $H^{(\tilde{I})} \in \mathbb{F}_{2^s}^{t \times N}$.

**Prover input.** The prover receives a word $\tilde{w} \in \mathbb{F}_2^N$ and wants to prove that $\tilde{w} \in \tilde{I}(R,g)$.

The protocol follows the standard IOPP structure consisting of a commit and a query phase.

**Commit Phase:**

- Let $\ell = \lceil \log_2 N \rceil$, and embed the vector $\tilde{w} = (\tilde{w}_1, \ldots, \tilde{w}_N)$ into a vector of length $2^\ell$ by padding with zeros if necessary.
- Let $\hat{w} : \{0,1\}^\ell \to \mathbb{F}$ be its multilinear extension over a sufficiently large field $\mathbb{F}$.
- The prover commits to $\hat{w}$ by setting $\mathsf{com} = \mathsf{Commit}(\hat{w})$.
- The verifier has commitment access to $\hat{w}$, and can request evaluation openings of the form $\hat{w}(z)$, where $z \in \mathbb{F}^\ell$.
- **Batched parity-check.**
  - Write $h_j(x) = \sum_{r=0}^{t-1} h_{r,j} x^r$.
  - The equation $\Phi_{\tilde{w}}(x) \equiv 0 \pmod{g(x)}$ is equivalent to the $t$ linear equations $\sum_{j=1}^{N} h_{r,j} \tilde{w}_j = 0, \qquad 0 \le r < t$.
  - The verifier samples a random challenge $\rho \leftarrow \mathbb{F}$ and forms the random linear combination
  $$\sum_{r=0}^{t-1} \rho^r \sum_{j=1}^{N} h_{r,j} \tilde{w}_j = 0.$$
  - Let the public coefficients be $\lambda_j = \sum_{r=0}^{t-1} \rho^r h_{r,j}$. Then the single parity-check equation is equal to $\sum_{j=1}^{N} \lambda_j \tilde{w}_j = 0$.

- **Sum-check protocol.**
  - Let $\hat{\lambda} : \{0,1\}^\ell \to \mathbb{F}$ be the multilinear extension of the public coefficient vector $(\lambda_1, \ldots, \lambda_N, 0, \ldots, 0)$.
  - The prover and verifier run the sum-check protocol for the claim

$$\sum_{u \in \{0,1\}^\ell} \hat{\lambda}(u)\hat{w}(u) = 0.$$

**Query Phase:** The verifier performs the following steps:

- During the sum-check protocol, the verifier receives univariate polynomial messages from the prover.
- At the end of the protocol, the verifier obtains a random point $z \in \mathbb{F}^\ell$ and must check the claimed value $y = \hat{w}(z)$.
- The verifier queries the polynomial commitment by asking the prover to open $\hat{w}(z) = y$.
- The verifier accepts if and only if the following two steps are satisfied:
  1. the sum-check protocol accepts the claimed sum

$$\sum_{u \in \{0,1\}^\ell} \hat{\lambda}(u)\hat{w}(u) = 0;$$

  2. the commitment opening proves that $\hat{w}(z) = y$.

## 4.2 Complexity of the IOPP

We first recall the one-time cost of constructing the $s$-projected Goppa code. As shown in Section 3.2,

$$T_{\text{proj}}(n, t, m) = O(ntm^2)$$

bit operations. This preprocessing includes the orbit decomposition, Goppa-polynomial evaluations and inversions, trace computations, and construction of the projected parity-check data. Since the orbit syndrome data are assumed to be precomputed in the protocol below, $T_{\text{proj}}(n, t, m)$ is a setup cost and is separated from the online prover and verifier costs.

The complexity of the IOPP is obtained by combining the cost of several steps in the protocol as highlighted in the following theorem.

**Theorem 3** (Complexity of the new IOPP protocol)**.** *Let $\mathcal{C} = \tilde{I}(R, g) \subseteq \mathbb{F}_2^N$ be an $s$-projected Goppa code, where $N$ is the number of Frobenius orbits, and let $t = \deg(g)$, and $\ell = \lceil \log_2 N \rceil$. Assume that the orbit syndrome polynomials $h_i(x)$ for $i \in [1, N]$ are precomputed, and let $T_{\text{com}}(N)$ and $T_{\text{open}}(N)$ denote the cost of one commitment and one evaluation opening in the chosen polynomial commitment scheme. Then the IOPP protocol has the following complexities:*

- *Preprocessing time: The one-time construction of the $s$-projected code costs*

$$T_{\text{proj}}(n, t, m) = O(ntm^2)$$

  *bit operations,*

- *Round complexity: The number of rounds is* $\ell + 2$,
- *Prover time: The prover runs in* $O\big(T_{\mathrm{com}}(N) + Nt + N\big)$,
- *Verifier time: The verifier runs in time:* $O\big(Nt + \ell + T_{\mathrm{open}}(N)\big)$,
- *Proof length: The length of the proof is computed as follows:*

$$|\mathsf{com}| + 3\ell + k,$$

*where* $k$ *denotes the length of one evaluation-opening proof and this size depends on the polynomial commitment scheme that is used in the instantiation of our protocol.*

*Furthermore, the query complexity is equal to the query complexity of one evaluation opening.*

*Proof.* The prover first commits to the multilinear extension $\hat{w} : \{0,1\}^\ell \to \mathbb{F}$ of the projected word $\tilde{w} \in \mathbb{F}_2^N$. This costs $T_{\mathrm{com}}(N)$.

The verifier samples a random batching challenge $\rho \leftarrow \mathbb{F}$. For each orbit $\mathcal{O}_j$, the orbit syndrome polynomial has degree less than $t$ and is defined as

$$h_j(x) = \sum_{r=0}^{t-1} h_{r,j} x^r.$$

Therefore, when computing $\lambda_j = h_j(\rho) = \sum_{r=0}^{t-1} \rho^r h_{r,j}$ for all $j = 1, \ldots, N$, the cost consists of $O(Nt)$ field operations.

The verifier then reduces the parity-check condition to the single linear relation $\sum_{j=1}^{N} \lambda_j \tilde{w}_j = 0$. After padding to length $2^\ell$, this is written as the sum-check claim

$$\sum_{u \in \{0,1\}^\ell} \hat{\lambda}(u)\hat{w}(u) = 0.$$

The summand is a product of two multilinear polynomials, and therefore has individual degree at most 2. Therefore, in each of the $\ell$ rounds of sum-check, the prover sends a univariate polynomial of degree at most 2, which can be represented by 3 field elements.

Thus the sum-check proof contributes $3\ell$ field elements and has $\ell$ rounds. Together with the commitment phase and the final evaluation-opening phase, the total number of rounds is $\ell + 2$. The prover's sum-check work is linear in $N$, i.e. $O(2^\ell) = O(N)$. Therefore the total prover time is

$$O\big(T_{\mathrm{com}}(N) + Nt + N\big).$$

The verifier performs $O(\ell)$ field operations for the sum-check consistency checks, computes the public coefficients $\lambda_j$, and verifies one commitment opening. Hence the verifier time is

$$O\big(Nt + \ell + T_{\mathrm{open}}(N)\big),$$

Finally, the verifier makes only one query to the committed polynomial, namely the final sum-check point $z \in \mathbb{F}^\ell$. Therefore the query complexity is exactly the query complexity of one evaluation opening in the chosen polynomial commitment scheme. □

### 4.3 Security Analysis of the IOPP

We analyze the correctness and the soundness of our IOPP protocol and summarize our results in the following theorems.

**Theorem 4** (Completeness)**.** *If $\tilde{w} \in C$, then the honest prover convinces the verifier to accept with probability* 1*, assuming perfect completeness of the sum-check protocol and the polynomial commitment evaluation protocol.*

*Proof.* Since $\tilde{w} \in C$, its syndrome vanishes:

$$\sum_{j=1}^{N} \widetilde{w}_j h_j(x) \equiv 0 \pmod{g(x)}.$$

Hence every coefficient equation is zero, and therefore every random linear combination of these equations is zero. The sum-check statement is true, so the honest sum-check prover is accepted. Finally, the commitment opening is valid because the prover committed to the multilinear extension of $\tilde{w}$. □

**Theorem 5** (Soundness of the IOPP from $s$-projected Goppa codes)**.** *Let $\mathcal{C} = \tilde{I}(R, g) \subseteq \mathbb{F}_2^N$ be an $s$-projected Goppa code obtained from the Frobenius orbit decomposition $L = \bigsqcup_{j=1}^{N} \mathcal{O}_j$, with orbit representatives $R = \{o_1, \ldots, o_N\}$. Let $g(x)$ be the Goppa polynomial of degree $t = \deg(g)$, and let*

$$h_j(x) = \sum_{\beta \in \mathcal{O}_j} \frac{1}{x - \beta} \pmod{g(x)}$$

*be the orbit syndrome polynomials. Let $\tilde{w} = (\tilde{w}_1, \ldots, \tilde{w}_N) \in \mathbb{F}_2^N$ be a received projected word such that $\Delta(\tilde{w}, \mathcal{C}) > \delta N$. Assume that the verifier samples the batching challenge $\rho \leftarrow \mathbb{F}$, and that the polynomial commitment scheme has evaluation soundness error $\epsilon_{\mathrm{PC}}$. Then the protocol described in Section 4.1 is sound with soundness error*

$$\epsilon \leq \frac{t - 1 + 2\lceil \log_2 N \rceil}{|\mathbb{F}|} + \varepsilon_{\mathrm{PC}}.$$

*Proof.* Let $\ell = \lceil \log_2 N \rceil$. The prover commits to the multilinear extension $\hat{w} : \{0,1\}^\ell \to \mathbb{F}$ of the projected word $\tilde{w}$. We note that the verifier does not check orbit-wise conditions. Instead, we use a membership protocol in the projected Goppa code which is characterized by the the full parity-check relation

$$\Phi_{\tilde{w}}(x) = \sum_{j=1}^{N} \tilde{w}_j h_j(x) \equiv 0 \pmod{g(x)}.$$

Since $\Delta(\tilde{w}, \mathcal{C}) > \delta N$, we have in particular $\tilde{w} \notin \mathcal{C}$. Therefore $\Phi_{\tilde{w}}(x) \not\equiv 0 \pmod{g(x)}$. Let $h_j(x) = \sum_{r=0}^{t-1} h_{r,j} x^r$ denote the orbit syndrome polynomials. Then it holds that $\Phi_{\tilde{w}}(x) \equiv 0 \pmod{g(x)}$ is equivalent to the t scalar equations $\sum_{j=1}^{N} h_{r,j} \tilde{w}_j = 0$, for $0 \leq r < t$. And since $\Phi_{\tilde{w}}(x) \not\equiv 0 \pmod{g(x)}$, at least one of these scalar equations is false. The verifier samples and considers the batched expression

$$B(\rho) = \sum_{r=0}^{t-1} \rho^r \sum_{j=1}^{N} h_{r,j} \tilde{w}_j,$$

for a random $\rho \leftarrow \mathbb{F}$. Since at least one coefficient $\sum_{j=1}^{N} h_{r,j}\tilde{w}_j$ is nonzero, $B(\rho)$ is a nonzero polynomial in $\rho$ of degree at most $t-1$. Hence, by the Schwartz-Zippel lemma [DL78, Zip79, Sch80],

$$\Pr_{\rho \leftarrow \mathbb{F}}[B(\rho) = 0] \leq \frac{t-1}{|\mathbb{F}|}.$$

Since $B(\rho) \neq 0$, the claimed parity-check linear relation

$$\sum_{j=1}^{N} \lambda_j \tilde{w}_j = 0, \qquad \lambda_j = \sum_{r=0}^{t-1} \rho^r h_{r,j},$$

is false. The verifier then runs the sum-check protocol for the claim

$$\sum_{u \in \{0,1\}^\ell} \hat{\lambda}(u)\hat{w}(u) = 0,$$

where $\hat{\lambda}$ is the multilinear extension of the public vector $(\lambda_1, \ldots, \lambda_N, 0, \ldots, 0)$. The summand $\hat{\lambda}(u)\hat{w}(u)$ has degree at most 2, since it is the product of two multilinear polynomials. The standard soundness analysis of the sum-check protocol therefore implies that a malicious prover can make the verifier accept a false sum with probability at most

$$\frac{2\ell}{|\mathbb{F}|} = \frac{2\lceil \log_2 N \rceil}{|\mathbb{F}|}.$$

Finally, at the end of the sum-check protocol, the verifier obtains a random point $z \in \mathbb{F}^\ell$ and a claimed value $y = \hat{w}(z)$.

The verifier checks this value using the polynomial commitment opening protocol. By the evaluation soundness of the polynomial commitment scheme, the probability that the verifier accepts an incorrect opening is at most $\epsilon_{\mathrm{PC}}$.

Combining the three possible failure events by the union bound gives

$$\epsilon \leq \frac{t-1}{|\mathbb{F}|} + \frac{2\lceil \log_2 N \rceil}{|\mathbb{F}|} + \epsilon_{\mathrm{PC}}.$$

This proves the theorem. □

**Discussion on the role of the orbits.** While the orbits do not improve the soundness directly instead, they improve the representation of the code and therefore the efficiency of the IOPP. In particular, they reduce the dimension of the object that the prover commits to and the verifier queries.

Our IOPP protocol runs one parity-check test and one sum-check protocol. Therefore the soundness error does not contain a multiplicative factor depending on the number of orbits or on the maximum orbit size.

The orbit decomposition affects the protocol only through the projected length $N = |R|$, which determines the number of variables in the multilinear extension: $\ell = \lceil \log_2 N \rceil$.

Thus the orbit structure contributes logarithmically to the sum-check soundness term. The verifier checks the parity-check equation $\sum_{j=1}^{N} \tilde{w}_j h_j(x) \equiv 0 \pmod{g(x)}$, not the separate equations $\tilde{w}_j h_j(x) \equiv 0 \pmod{g(x)}$.

# 5 Conclusion

In this work, we used the $s$-projected Goppa codes introduced by Loidreau as the algebraic foundation for a new interactive oracle proof of proximity. We first showed how Frobenius-invariant Goppa codes can be projected onto a shorter representation indexed by Frobenius orbit representatives, while preserving the algebraic characterization of code membership. We then constructed an IOPP based on a polynomial commitment to the multilinear extension of the projected word, a randomly batched parity check equation, a sum-check protocol, and a final evaluation-consistency check.

Our key contributions include:

1. An exact orbit-count analysis for the Frobenius action in the power-of-two extension setting, together with a monotonicity result describing how the number of orbits varies with the projection parameter.

2. A preprocessing complexity analysis for constructing the $s$-projected Goppa code, including the orbit decomposition, Goppa-polynomial evaluations, trace computations, and construction of the projected parity-check matrix.

3. A new IOPP for $s$-projected Goppa codes based on the full projected parity-check relation, randomized batching, the sum-check protocol, and a polynomial commitment scheme.

4. Completeness, soundness, and complexity analyses of the proposed IOPP, including explicit bounds for prover complexity, verifier complexity, proof length, and query complexity.

These results contribute to the broader study of error-correcting codes in proof systems, with potential applications in zero-knowledge protocols, such as ZK-SNARKs, cryptographic commitments, and complexity theory.

**Comparison with prior constructions.** A natural question is what advantage the present construction offers over IOPPs based on general AG codes, such as [BLNR22], or over a direct construction using Goppa codes without projection.

The principal advantage of our approach is the algebraic compression provided by the Frobenius orbit decomposition. Instead of operating on the original evaluation set, the protocol works directly with the orbit representatives and their associated orbit syndrome polynomials. This preserves the parity-check equation of the code while reducing the size of the committed object.

In contrast, a naive construction based on Goppa codes treats the full evaluation set at once and does not provide such a decomposition, while general AG-code-based constructions such as [BLNR22] do not naturally expose an orbit structure that can be exploited in this way. FRI [BSBHR18] and AG-code-based constructions [BLNR22] are unconditional IOPPs whose soundness degrades with the proximity parameter and which pay for it with repeated queries, whereas our protocol makes one evaluation opening and has $\delta$-independent soundness, at the price of inheriting the assumption of a polynomial commitment scheme. What is specific to Goppa codes is the source of the compression: neither RS codes nor general AG codes come with a canonical Frobenius orbit decomposition of the evaluation set that (i) preserves an exact parity-check characterization of membership and (ii) shrinks the evaluation domain by a factor $m/s$. Curve automorphisms in the AG setting can shorten a code, but they do not in general leave an explicit trace-based parity-check matrix

of the shortened code, which is what makes the single batched parity-check equation (and so the constant query complexity) possible here.

Building proof systems from the same family of codes used by the McEliece cryptosystem, proposed almost 50 years ago, allows us to work within a single, well-understood algebraic and implementation stack. We explicitly note that, unlike McEliece, our construction does not rely on hiding the code structure, so its security does not depend on the hardness of distinguishing Goppa codes.

## References

[ABC11] Daniel Augot, Morgan Barbier, and Alain Couvreur. List-decoding of binary Goppa codes up to the binary Johnson bound. In *2011 IEEE Information Theory Workshop, ITW 2011, Paraty, Brazil, October 16-20, 2011*, pages 229–233. IEEE, 2011. doi:10.1109/ITW.2011.6089384.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *ACM SIGSAC CCS 2017*, pages 2087–2104. ACM, 2017. doi:10.1145/3133956.3134104.

[AS92] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs; A new characterization of NP. In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 2–13. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267824.

[BCD+16] Magali Bardet, Julia Chaulet, Vlad Dragoi, Ayoub Otmani, and Jean-Pierre Tillich. Cryptanalysis of the McEliece public key cryptosystem based on polar codes. In *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings 7*, pages 118–143. Springer, 2016. doi:10.1007/978-3-319-29360-8_9.

[BCF+16] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. On probabilistic checking in perfect zero knowledge. *Electron. Colloquium Comput. Complex.*, TR16-156, 2016. URL: https://eccc.weizmann.ac.il/report/2016/156, arXiv:TR16-156.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 427–457. Springer, 2022. doi:10.1007/978-3-031-07085-3_15.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019. doi:10.1007/978-3-030-17653-2_4.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016. doi:10.1007/978-3-662-53644-5_2.

[Ber68] E. Berlekamp. Nonbinary BCH decoding. *IEEE Transactions on Information Theory*, 14(2):242, march 1968. doi:10.1109/TIT.1968.1054109.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *ACM STOC 1991*, pages 21–31. ACM, 1991. doi:10.1145/103418.103428.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020. doi:10.1007/978-3-030-45721-1_24.

[BLNR22] Sarah Bordage, Mathieu Lhotel, Jade Nardi, and Hugues Randriam. Interactive oracle proofs of proximity to algebraic geometry codes. In Shachar Lovett, editor, *37th Computational Complexity Conference, CCC 2022, July 20-23, 2022, Philadelphia, PA, USA*, volume 234 of *LIPIcs*, pages 30:1–30:45. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.CCC.2022.30, doi:10.4230/LIPICS.CCC.2022.30.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.14.

[BSGKS19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. Deep-fri: Sampling outside the box improves soundness, 2019. preprint, arxiv:1903.12243.

[Bur97] William Burnside. *Theory of Groups of Finite Order*. Cambridge University Press, 1897.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023. doi:10.1007/978-3-031-30617-4_17.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT 2020*,

volume 12105 of *LNCS*, pages 769–793. Springer, 2020. doi:10.1007/978-3-030-45721-1_27.

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

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023. doi:10.1007/978-3-031-38545-2_7.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *ACM STOC 1985*, pages 291–304. ACM, 1985. doi:10.1145/3335741.3335750.

[KPV22] Assimakis A. Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1725–1737. ACM, 2022. doi:10.1145/3548606.3560657.

[Loi01a] Pierre Loidreau. Codes derived from binary Goppa codes. *Probl. Inf. Transm.*, 37(2):91–99, 2001. doi:10.1023/A\%3A1010406807141.

[Loi01b] Pierre Loidreau. Étude et optimisation de cryptosystèmes à clé publique fondés sur la théorie des codes correcteurs. *Thèse de doctorat de l'Ecole Polytechnique*, 2001.

[LT13] Grégory Landais and Jean-Pierre Tillich. An efficient attack of a McEliece cryptosystem variant based on convolutional codes. In *Post-Quantum Cryptography: 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings 5*, pages 102–117. Springer, 2013. doi:10.1007/978-3-642-38616-9_7.

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

[MS07] L. Minder and A. Shokrollahi. Cryptanalysis of the Sidelnikov cryptosystem. In M. Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 347–360. Springer, 2007. doi:10.1007/978-3-540-72540-4_20.

[Pat75] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203 – 207, March 1975. doi:10.1109/TIT.1975.1055350.

[RRR16] Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. Constant-round interactive proofs for delegating computation. In *ACM STOC 2016*, pages 49–62. ACM, 2016. doi:10.1145/2897518.2897652.

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

[SS92] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 2(4):439 – 444, 1992. doi:10.1515/dma.1992.2.4.439.

[Wie10] Christian Wieschebrink. Cryptanalysis of the Niederreiter public key scheme based on grs subcodes. In *Post-Quantum Cryptography: Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings 3*, pages 61–72. Springer, 2010. doi:10.1007/978-3-642-12929-2_5.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation (EUROSAM '79)*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979. doi:10.1007/3-540-09519-5_73.