An Algebraic Attack Against McEliece-like Cryptosystems Based on BCH Codes

Freja Elbro and Christian Majenz
Technical University of Denmark, Denmark;
{freel,chmaj}@dtu.dk

***Abstract*—We present an algebraic attack on a McEliece-like scheme based on BCH codes (BCH-McEliece), where the Goppa code is replaced by a suitably permuted BCH codes. Our attack continues the line of work devising attacks against McEliece-like schemes with Goppa-like codes, with the goal of getting a better understanding of why Goppa codes are so intractable. Our starting point is the work of Faugère, Perret and Portzamparc (Asiacrypt 2014). We take their algebraic model and adapt and improve their attack algorithm so that it can handle BCH-McEliece. We demonstrate experimentally that our attack is practical for high rate codes over non-prime fields for parameters where generic attacks suggest cryptographic security.**

I. Introduction

The design of quantum-safe public key encryption schemes is a task of critical importance. This importance is highlighted by the recent NIST process to standardize quantum-resistant public-key cryptographic algorithms. One of the oldest cryptosystems not affected by known quantum computer attacks is the McEliece public key encryption scheme [1], a code-based cryptosystem based on binary Goppa codes. The main advantage of this cryptosystem is its very fast encryption/decryption, and the fact that despite decades of research, no attacks have been developed that exploit the structure of the Goppa codes. Generic decoding algorithms still provide the best known attacks. Furthermore, despite many improvements to these generic decoding algorithms, the generic decoder from before McEliece's time (due to Prange [2]) still has the best asymptotic runtime [3].

Although the McEliece cryptosystem has withstood cryptanalysis for almost 45 years, its security raises fundamental questions about the difficulty of recovering the secret algebraic structure of a (binary) Goppa code solely from an arbitrary basis. This is all the more intriguing as Goppa codes are linear subspaces of generalized Reed-Solomon codes for which it is famously easy to recover the algebraic structure from any basis [4]. This surprising constellation motivate the study of mathematical approaches to attacking McEliece-like schemes.

Algebraic cryptanalysis is a framework that can in principle be applied to any cryptographic scheme. It has for instance been used successfully against several stream ciphers and cryptographic schemes based on the rank-decoding problem. The idea is to associate to a cryptographic primitive a set of polynomial equations such that the solutions of this system allow the recovery of secret information (like the secret key).

It was shown in [5] that it is possible to construct a polynomial system of equations for the McEliece cryptosystem that a private key has to satisfy. This algebraic modeling works for McEliece-like cryptosystem as long as the underlying codes are alternant codes (of which Goppa codes form a subclass). In general, the resulting polynomial system has equations of high degree and many variables, rendering it intractable. However, it has lead to attacks when alternant codes with symmetries like cyclicity or dyadicity are used [6], [7]. Here, the symmetries can be used to reduce the number of variables, making the polynomial system tractable.

More recently, an attack on the cryptosystem Wild McEliece (a McEliece variant with wild Goppa codes) has been developed, which experimentally is shown to beat generic decoding algorithms when the field is non-prime and the extension degree is two or three [8]. Here they also use the modeling of [5], but solve the problem of high degree equations and many variables in another way: exploiting multiplicities in the Goppa polynomial, more polynomial equations can be added. Clever manipulations then allow halving the number of variables of the system. For the resulting system, it turns out that using only equations of low degree extends the solution set to a union of vector spaces with a nice structure. The structure of these vector spaces make it possible to find the secret key, given bases for the vector spaces. Such bases can be found by solving the system with low degree polynomials multiple times, each time fixing a number of variables equal to the dimension of the vector spaces.

The aim of this paper is to expand on the results in [8] to devise an attack on BCH-McEliece. This case is interesting for the following reasons:

1) The polynomial system associated to BCH codes by [5] is simpler, and hence might provide some insight. Possibly also on the security of the (standard) McEliece cryptosystem.
2) Narrow-sense BCH codes are a class of Goppa codes, and they might give slightly better parameters if used in McEliece-like schemes. Hence breaking a scheme based on masked BCH codes has been issued as a challenge in [9].
3) To the best of our knowledge, there is only one known better-than-generic attack against BCH-McEliece, which only applies for very specific parameters.

The non-generic attack, which applies to BCH McEliece for some parameters, is a polynomial attack on Wild McEliece over quadratic extensions [10]. This attack applies for Wild

Goppa codes with extension degree 2, and this class of codes has a a small, but not trivial, intersection with the class of BCH codes.

The generic attacks, which are relevant to BCH-McEliece, are information-set decoding [2], and the support splitting algorithm [11]. Information-set decoding is a generic decoding algorithm, which was originally developed by Prange [2]. Since its development, it has seen many improvements, but for errors with a sublinear Hamming weight, none of these improvements have led to asymptotic improvements in the running time of the algorithm [3]. The support splitting algorithm takes as input two codes which are equal up to a fixed permutation on the codeword coordinates, and gives as output the permutation. The complexity of the support splitting algorithm is polynomial in the length of the code and exponential in the dimension of the hull (the intersection of the code with its dual). For BCH codes, this dimension is large in general, so for BCH-McEliece, we consider information set decoding the most relevant generic attack.

A. Our contribution

We present a key recovery attack against the BCH-McEliece encryption scheme, obtained by adapting and improving the attack methodology developed in [8]. To the best of our knowledge, this is the first algebraic attack against BCH-McEliece for BCH codes, which are not also quadratic-extension wild Goppa codes.

We provide an implementation of the attack (in Magma [12]). Experimental results obtained with this implementation show that our attack is practical for high rate codes over non-prime fields for parameters where generic attacks suggest cryptographic security. In contrast to [8] and [10], we do not limit ourselves to extension degrees 2 or 3, and we find that this does not structurally influence the attack. By definition of BCH-McEliece, our attack tackles 4 out of 6 defense strategies suggested in the challenge of [9] (scaling, permutation, subfield and wildness[1]) in the context of BCH codes[2]. While we describe our attack in the "plain" BCH-McEliece setting, we provide experimental evidence that the attack defeats all 6 defenses mentioned in [9].

On the technical side,

1) we demonstrate that we can take the technique of [8] a step further by reducing the degrees of the polynomials in the system even more. We thus increase the dimension of the vector spaces in the solution set and thereby increase number of variables we can fix compared to [8].
2) In addition we replace an exhaustive search step in the attack algorithm of [8] by an algorithm which iteratively narrows down the search space, improving the efficiency of the attack.

[1] In [9] these are described as defenses of RS-codes. Therefore, we understand scaling, permutation and wildness to be included in the definition of BCH codes. Hence the only additional defense to BCH codes is permutation.

[2] [9] states that the four defenses we chose to focus on are "immediately" broken for BCH codes by the support splitting algorithm [11], but we find this to be untrue, as the support splitting algorithm has runtime exponential in the dimension of the hull of the code, and this dimension is large for BCH codes.

B. Organization

In section II and III we briefly present some notation and background information on algebraic coding theory. In section IV we present the McEliece cryptosystem and outline the algebraic modeling of [5]. In section V we describe our attack on permuted narrow sense primitive BCH-codes. In section VI and VII we first give a theoretical analysis of the runtime of our attack and then we provide experimental results on the runtime for specific parameters. In the latter of these two sections, we will additionally provide experimental results describing what happens when we include all defense strategies of [9]. In section VIII, we round the article off with a conclusion, where we outline possible further works.

II. Notation

The symbol $\triangleq$ is used to define the left-hand side object. $\mathbb{F}_q$ is the field with q elements where q is a power of a prime integer. When $a \leqslant b$ are integers, $[\![a, b]\!]$ is the integer interval $[a, b] \cap \mathbb{Z}$. When $b < a$, $[\![a, b]\!] \triangleq \emptyset$. $\mathbb{Z}_n \triangleq \mathbb{Z}/n\mathbb{Z}$ denotes the ring $\{0, 1, .., n-1\}$ with addition and multiplication modulo n. Vectors $\boldsymbol{a} = (a_1, \ldots, a_n)$ will be regarded as row vectors by default. The (Hamming) weight of $\boldsymbol{a} \in \mathbb{F}^n$ denoted by $\mathrm{wt}_H(\boldsymbol{a})$ is the number of non-zero coordinates. For a function $f : \mathbb{F}^n \to \mathbb{F}^n$, we write $f(\boldsymbol{a}) \triangleq (f(a_1), .., f(a_n))$. Given two fields $\mathbb{K} \subseteq \mathbb{F}$ and $\mathbf{A} \in \mathbb{F}^{r \times n}$, we define

$$\ker_{\mathbb{K}} \mathbf{A} \triangleq \left\{ \boldsymbol{v} \in \mathbb{K}^n \;\middle|\; \mathbf{A}\boldsymbol{v}^{\mathsf{T}} = \mathbf{0} \right\}.$$

$\mathbb{K}[X]$ is the ring of univariate polynomials with coefficients in the field $\mathbb{K}$. We also denote the ring of multivariate polynomials over $\mathbb{K}$ with variables $\boldsymbol{X} = (X_1, \ldots, X_n)$ by $\mathbb{K}[\boldsymbol{X}]$. Given $\mathcal{P} = \{P_1, \ldots, P_k\} \subseteq \mathbb{K}[\boldsymbol{X}]$ and a field $\mathbb{F} \supseteq \mathbb{K}$, we define

$$\mathcal{Z}_{\mathbb{F}}(\mathcal{P}) \triangleq \{\boldsymbol{x} \in \mathbb{F}^n \mid \forall i \in [\![1, k]\!], P_i(\boldsymbol{x}) = 0\}.$$

Given two n-tuples $\boldsymbol{x}$ and $\boldsymbol{y}$ from $\mathbb{F}^n$, we define the Vandermonde matrix $\mathbf{V}_t(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{F}^{t \times n}$ of order $t \in \mathbb{N}$

$$\mathbf{V}_t(\boldsymbol{x}, \boldsymbol{y}) \triangleq \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_1 x_1 & y_2 x_2 & \cdots & y_n x_n \\ \vdots & \vdots & & \vdots \\ y_1 x_1^{t-1} & y_2 x_2^{t-1} & \cdots & y_n x_n^{t-1} \end{bmatrix}.$$

III. Algebraic Coding Theory

This section is devoted to recalling classical notions on algebraic coding theory. We refer to [13] for a detailed treatment.

A linear *code* $\mathscr{C}$ of length n and dimension k over a field $\mathbb{F}$ is a k-dimensional subspace of $\mathbb{F}^n$ where $k < n$ are natural numbers. Any matrix such that its rows form a basis of $\mathscr{C}$ is called a *generator matrix*. A generator matrix $\mathbf{G}$ is said to be

on *standard form* if $\mathbf{G} = \begin{bmatrix}\mathbf{I}_k & | & \mathbf{A}\end{bmatrix}$ with $\mathbf{A} \in \mathbb{F}^{k\times n-k}$. The *dual* of $\mathscr{C} \subseteq \mathbb{F}^n$ is the linear space

$$\mathscr{C}^\perp \triangleq \left\{ \boldsymbol{z} \in \mathbb{F}^n \;\middle|\; \forall \boldsymbol{u} \in \mathscr{C},\ \sum_i u_i v_i = 0 \right\}.$$

We always have $\dim \mathscr{C}^\perp = n - \dim \mathscr{C}$, and a generator matrix of $\mathscr{C}^\perp$ is called a *parity check* matrix of $\mathscr{C}$. Moreover, if $\begin{bmatrix}\mathbf{I}_k & | & \mathbf{A}\end{bmatrix}$ is a generator matrix for $\mathscr{C}$ then $\begin{bmatrix}\mathbf{A}^\mathsf{T} & | & \mathbf{I}_{n-k}\end{bmatrix}$ is a parity-check matrix for $\mathscr{C}$. An algorithm Φ is *t-correcting* if $\Phi(\boldsymbol{u}+\boldsymbol{e}) = \boldsymbol{u}$ for every $(\boldsymbol{u}, \boldsymbol{e})$ in $\mathscr{C} \times \mathbb{F}^n$ when $\mathrm{wt}_H(e) \leqslant t$.

We now recall definitions and key properties of the linear codes, which are relevant for this article.

Definition 1 (Generalized Reed-Solomon code)**.** *Let k and n be integers such that $1 \leqslant k < n \leqslant q$ where q is a power of a prime number. The* generalized Reed-Solomon *code $\mathsf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ of dimension k, where $\boldsymbol{x}$ is an n-tuple of distinct elements of $\mathbb{F}_q$ and $\boldsymbol{y}$ is an n-tuple of nonzero elements in $\mathbb{F}_q$, is defined as the code with generator matrix $\mathbf{V}_k(\boldsymbol{x}, \boldsymbol{y})$.*

One of the key properties of GRS codes is that they can be efficiently decoded.

Proposition 2. *For every generalized Reed-Solomon code $\mathsf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$ there exists a polynomial-time algorithm that corrects all errors of weight at most $(n-k)/2$ whenever $\boldsymbol{x}$ and $\boldsymbol{y}$ are known. We denote this decoder by $\mathsf{decGRS}_{k,\boldsymbol{x},\boldsymbol{y}}$*

Another key property is that the dual of a GRS code is again a GRS code.

Proposition 3. *$\mathsf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})^\perp = \mathsf{GRS}_{n-k}(\boldsymbol{x}, \boldsymbol{z})$ where for every $i \in [\![1, n]\!]$,*

$$z_i^{-1} \triangleq y_i \prod_{j=1 j\neq i}^{n} (x_j - x_i)$$

We turn now to subfield subcodes of GRS codes.

Definition 4 (Alternant code)**.** *Given a natural number $m > 1$ the q-ary alternant code $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$, where $\boldsymbol{x}$ is an n-tuple of distinct elements of $\mathbb{F}_{q^m}$ and $\boldsymbol{y}$ is an n-tuple of nonzero elements in $\mathbb{F}_{q^m}$, is the following (subfield) subcode*

$$\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) = \mathsf{GRS}_r(\boldsymbol{x}, \boldsymbol{y})^\perp \cap \mathbb{F}_q^n$$

Remark 5. We note that $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) \subset \mathsf{GRS}_{n-r}(\boldsymbol{x}, \boldsymbol{z})$ with $\boldsymbol{z}$ defined in proposition 3, so $\mathsf{decGRS}_{n-r,\boldsymbol{x},\boldsymbol{z}}$ can be used as a decoding algorithm for $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$. It can correct up to $r/2$ errors whenever $\boldsymbol{x}$ and $\boldsymbol{z}$ are known – or equivalently whenever $\boldsymbol{x}$ and $\boldsymbol{y}$ are known.

Another key property of alternant codes is the following:

Corollary 6. *Given $\boldsymbol{x}$ that is an n-tuple of distinct elements of $\mathbb{F}_{q^m}$ and $\boldsymbol{y}$ which is an n-tuple of nonzero elements in $\mathbb{F}_{q^m}$, it holds that*

$$\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) = \ker_{\mathbb{F}_q} \mathbf{V}_r(\boldsymbol{x}, \boldsymbol{y}).$$

We turn now to specific alternant codes.

Definition 7 (Goppa code)**.** *A q-ary Goppa code $\mathscr{G}(\boldsymbol{x}, \gamma)$ of length n, where $\boldsymbol{x}$ is an n-tuple of distinct elements from $\mathbb{F}_{q^m}$ with $m > 1$ and γ is a polynomial in $\mathbb{F}_{q^m}[X]$ of degree t such that $\boldsymbol{x}$ does not contain any root of γ, is the alternant code $\mathscr{A}_t(\boldsymbol{x}, \boldsymbol{y})$ where $y_j \triangleq \gamma(x_j)^{-1}$ for every $j \in [\![1, n]\!]$.*

A key property of Goppa codes is the following:

Proposition 8 ([14])**.** *Let $\gamma \in \mathbb{F}_{q^m}[X]$ be a monic and square free polynomial of degree t. Let $\boldsymbol{x}$ be an n-tuple of distinct elements of $\mathbb{F}_{q^m}$ satisfying $\gamma(x_i) \neq 0$ for every $i \in [\![1, n]\!]$. Then*

$$\mathscr{G}(\boldsymbol{x}, \gamma^{q-1}) = \mathscr{G}(\boldsymbol{x}, \gamma^q).$$

The last code-class we introduce, is the BCH codes, which are also a subclass of alternant codes.

Definition 9 (BCH code)**.** *Let us assume that $q = p^s$ where p is a prime number and $s \geqslant 1$. Let b, m and n be a natural numbers such that n divides $q^m - 1$. Let α be an arbitrary primitive n-th root of unity in $\mathbb{F}_{q^m}$.*

A q-ary BCH code, of length n denoted by $\mathsf{BCH}_{b,r}(\alpha)$ is the q-ary alternant code $\mathscr{A}_r(\boldsymbol{\alpha}, \boldsymbol{\alpha}^b)$, where $\boldsymbol{\alpha} \triangleq (1, \alpha, \alpha^2, \ldots, \alpha^{n-1})$.

When $b = 1$ the BCH code is called narrow-sense and when $n = q^m - 1$ it is called primitive.

It is more standard to define a BCH code in terms of its designed distance δ. It is then defined as the cyclic code with zeros $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}$. If we set $\delta = r + 1$, then the two definitions are equivalent. We chose definition 9, as all the properties of BCH codes that will be relevant to us follow directly from it and we want to emphasize the connection to Goppa codes.

Proposition 10. *A narrow-sense q-ary BCH code $\mathsf{BCH}_{1,r}(\alpha)$ is the q-ary Goppa code $\mathscr{G}\left(\boldsymbol{\alpha}^{-1}, X^r\right)$ where $\boldsymbol{\alpha}^{-1} = \left(1, \alpha^{-1}, \ldots, \alpha^{-(n-1)}\right)$.*

Proof. By definition of a narrow-sense BCH code, any codeword is in the right kernel of the following matrix

$$\begin{bmatrix} 1 & \alpha^1 & \alpha^2 & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{(n-1)2} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^r & \alpha^{2r} & \cdots & \alpha^{(n-1)r} \end{bmatrix}. \tag{1}$$

For every $i \in [\![1, r]\!]$ and $j \in [\![1, n]\!]$ the entry located at position (i, j) in the matrix (1) is equal to $\alpha^{i(j-1)}$. But by letting $\ell \triangleq r - i$ and setting $a_j \triangleq \alpha^{-(j-1)}$ we then have

$$\begin{aligned}\alpha^{i(j-1)} = \alpha^{(r-\ell)(j-1)} &= \alpha^{(r)(j-1)}\alpha^{-\ell(j-1)} \\ &= a_j^{-(r)} a_j^\ell = \frac{a_j^\ell}{a_j^r}\end{aligned}$$

As $\ell \in [\![0, r-1]\!]$ this proves that a narrow-sense BCH code is a Goppa code defined with the polynomial X^r. □

Theorem 11 ([15])**.** *The dimension of a narrow-sense BCH code $\mathsf{BCH}_{1,r}(\alpha)$ is at least $n - m\lceil r - r/q \rceil$.*

IV. The McEliece cryptosystem

The McEliece cryptosystem [1] is a public key encryption scheme. We recall its key generation, encryption and decryption functions in algorithm 1, 2 and 3 respectively. We give the description for a generalization of the McEliece scheme based on alternant codes. To get the original McEliece cryptosystem, you just have to specialize to $q = 2$ and $\boldsymbol{y} = \gamma(\boldsymbol{x})^{-1}$. Then apply proposition 8 to double the number of errors that can be corrected.

In the algorithms we will write $x \xleftarrow{\$} \mathcal{S}$ to express that x is sampled according to the uniform distribution over a set $\mathcal{S}$.

Algorithm 1 $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}\left(1\!\!1^{\lambda}\right)$

1: $(q, m, n, r) \leftarrow \mathsf{param}(1\!\!1^{\lambda})$
2: $\boldsymbol{x} \xleftarrow{\$} \mathbb{F}_{q^m}^n$ composed of pairwise distinct entries
3: $\boldsymbol{y} \xleftarrow{\$} (\mathbb{F}_{q^m}^*)^n$
4: Pick at random a generator matrix $\mathbf{G}$ of $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$
5: **return** Return $(\mathsf{sk}, \mathsf{pk})$ with $\mathsf{sk} \triangleq (\boldsymbol{x}, \boldsymbol{y})$ and $\mathsf{pk} \triangleq \left(\mathbf{G}, \frac{r}{2}\right)$

Algorithm 2 $\boldsymbol{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \boldsymbol{m})$

1: Parse pk to obtain $\mathbf{G}$, r
2: $\boldsymbol{e} \xleftarrow{\$} \mathbb{F}_q^n$ of weight $\mathrm{wt}_H(\boldsymbol{e}) = \lfloor r/2 \rfloor$
3: **return** Return $\boldsymbol{c} \triangleq \boldsymbol{m}\mathbf{G} + \boldsymbol{e}$

Algorithm 3 $\boldsymbol{m} \leftarrow \mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})$

1: Parse sk to get $\boldsymbol{x}$ and $\boldsymbol{y}$
2: Define $\boldsymbol{z}$ as in prop. 3 from $\boldsymbol{x}$ and $\boldsymbol{y}$
3: **return** Return $\boldsymbol{m} \triangleq \mathsf{decGRS}_{n-r,\boldsymbol{x},\boldsymbol{z}}(\boldsymbol{c})$

A. Algebraic Modeling of the Key Recovery Problem

We here explain how recovering the secret key of any McEliece-like cryptosystem based on alternant codes is connected to solving a polynomial system. The adversary has access to the (public) generator matrix $\mathbf{G} = (g_{i,j})$ belonging to $\mathbb{F}_q^{k \times n}$. This matrix contains a basis for an alternant code $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y})$ of length $n \leqslant q^m$ and dimension $k \geqslant n - rm$.

The algebraic modeling given in [5] introduces $2n$ variables $\boldsymbol{X} \triangleq (X_1, \ldots, X_n)$ and $\boldsymbol{Y} \triangleq (Y_1, \ldots, Y_n)$ satisfying the following,

$$\sum_{j=1}^{n} g_{i,j} Y_j X_j^u = 0, \quad 1 \leqslant i \leqslant k, \quad 0 \leqslant u \leqslant r-1, \tag{2}$$

Substituting $\boldsymbol{X} \leftarrow \boldsymbol{x}$ and $\boldsymbol{Y} \leftarrow \boldsymbol{y}$ into the system yields a valid solution, because

$$\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) = \ker_{\mathbb{F}_q} \mathbf{V}_r(\boldsymbol{x}, \boldsymbol{y}),$$

see Cor. 6.

As Goppa codes are alternant codes, we can use the same modeling for them. We do have the additional information that $\boldsymbol{y} = \gamma(\boldsymbol{x})^{-1}$, but we do not know of a way to leverage this information without increasing massively the degree of the equations. So, like [5], we do not give a separate modeling for Goppa codes.

A classical approach for solving such a system is to use algorithms relying on the computation of Gröbner bases. But in a cryptographic setting, the number of unknowns can be really high, as n is larger than $1,000$. Unless the number of unknowns is very low (less than 100 as in [5]), no existing algorithms can deal with the current cryptographic parameters.

These practical obstacles do not change the fact that key recovery problem of the McEliece public-key cryptosystem has a strong connection to the following problem

Problem 1. *Considering the polynomial ring $\mathbb{F}_q[\boldsymbol{X}, \boldsymbol{Y}]$ with $\boldsymbol{X} \triangleq (X_1, \ldots, X_n)$ and $\boldsymbol{Y} \triangleq (Y_1, \ldots, Y_n)$. Compute $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P})$ where*

$$\mathcal{P} \triangleq \left\{ \sum_{j=1}^{n} g_{i,j} Y_j X_j^{\ell} \;\middle|\; i \in [\![1, k]\!],\ \ell \in [\![0, r-1]\!] \right\} \tag{3}$$

and $\mathbf{G} = (g_{i,j}) \in \mathbb{F}_q^{k \times n}$ is a generator matrix of a q-ary alternant code $\mathscr{A}_r(\boldsymbol{x}, \boldsymbol{y}) \subset \mathbb{F}_q^n$ of dimension k.

However, the two are not equivalent, as $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P})$ might contain more than just the secret key $(\boldsymbol{x}, \boldsymbol{y})$. In fact, we know that the solution set contains at least m elements as the zeros are sought in $\mathbb{F}_{q^m}$, but each polynomial in $\mathcal{P}$ has its coefficients in $\mathbb{F}_q$. Hence we can apply the Frobenius mapping defined for $z \in \mathbb{F}_{q^m}$ by $z \mapsto z^{q^e}$ with $e \in [\![0, m-1]\!]$ to get other solutions in $\mathbb{F}_{q^m}$.

Lemma 12. *If $(\boldsymbol{x}, \boldsymbol{y})$ is in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P})$, then so is $(\boldsymbol{x}^{q^e}, \boldsymbol{y}^{q^e})$ for every $e \in [\![1, m-1]\!]$.*

Furthermore, Theorem 4 in [16] shows that even though $(\boldsymbol{x}', \boldsymbol{y}') \neq (\boldsymbol{x}, \boldsymbol{y})$, it might still be the case that $\mathsf{GRS}_k(\boldsymbol{x}', \boldsymbol{y}') = \mathsf{GRS}_k(\boldsymbol{x}, \boldsymbol{y})$. If this is the case, then $(\boldsymbol{x}', \boldsymbol{y}')$ is also in the zero set of $\mathcal{P}$.

These two groups of extra solutions are not a problem though, as they all generate the same alternant code, so regardless of which one you choose, you can get a decoding algorithm for the code generated by G.

However, $\mathcal{P}$ also has zeros which does not generate the same alternant code. In fact, these extra solutions often do not generate an alternant code at all, as they do not consist of distinct/non-zero elements. These solutions are not well characterized, but it has been found experimentally that if we include extra polynomials which encode the information that $\boldsymbol{x}$ must consist of n distinct elements and all entries of $\boldsymbol{y}$ must be non-zero, then the solution set becomes manageable, so finding the secret key becomes equivalent to solving problem 1 with the inclusion of those extra polynomials [5]. The drawback of this approach is of course the extra variables added to an already intractable system.

V. Cryptanalysis of the BCH-McEliece Cryptosystem

A. Preliminaries

In this section we describe a key-recovery attack on the BCH-McEliece cryptosystem, which uses permuted narrow-sense primitive BCH codes. We will introduce additional notation throughout this section, and therefore provide an overview of notation in appendix A. We hope that this can serve as an index in case our readers want to go back and look up a definition.

Algorithm 4 describes the key generation process of the BCH-McEliece cryptosystem. The encryption and decryption functions do not change from the previous section, so they can be found in Algorithm 2 and Algorithm 3 respectively.

Algorithm 4 $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}\left(\mathbb{1}^\lambda\right)$

$(p, s, m, r) \leftarrow \mathsf{param}(\mathbb{1}^\lambda)$
$q \leftarrow p^s$
$n \leftarrow q^m - 1$
Pick at random in $\mathbb{F}_{q^m}$ a primitive n-th root of unity α
Pick at random a permutation π on $[\![0, n-1]\!]$
$\boldsymbol{x} \leftarrow \left(\alpha^{\pi(0)}, \alpha^{\pi(-1)}, \ldots, \alpha^{\pi(-(n-1))}\right)$
Pick at random a generator matrix $\mathbf{G}$ of $\mathscr{G}(\boldsymbol{x}, X^r)$
return $(\mathsf{sk}, \mathsf{pk})$ with $\mathsf{sk} \triangleq \boldsymbol{x}$ and $\mathsf{pk} \triangleq (\mathbf{G}, r/2)$

We recall from Prop. 10 that a narrow-sense BCH code is the Goppa code $\mathscr{G}(\boldsymbol{\alpha}, X^r)$ where $\boldsymbol{\alpha} \triangleq \left(1, \alpha^{-1}, \ldots, \alpha^{-(n-1)}\right)$.

The key recovery problem we want to solve is

Problem 2. *Given a public key $\mathbf{G}$ that is obtained from the BCH-McEliece cryptosystem instantiated with a k-dimensional q-ary permuted narrow-sense primitive BCH code $\mathsf{BCH}_r(\boldsymbol{x})$. Assuming that $q = p^s$ where p is a prime number and $s \geqslant 1$, find $\boldsymbol{x}$.*

Following [8], we will only use equations of low degree in the algebraic modeling of [5]. This makes the solution set of the system larger, naively increasing the difficulty of the remaining search problem. However, the extra solutions are helpful as they form a vector space with a very particular structure. We can therefore 1) repeatedly fix variables when solving the system, which means that it is easier to solve, and 2) "detangle" the vector spaces to obtain the correct solution.

B. Algebraic modeling

We specialize the algebraic modeling of [5] to the BCH-case and thus see that $\boldsymbol{x}$ is a zero of

$$\mathcal{P} \triangleq \left\{ \sum_{j=1}^{n} g_{i,j} X_j^\ell \;\middle|\; i \in [\![1,k]\!],\;\; \ell \in [\![1,r]\!] \right\}. \tag{4}$$

We can thus solve Problem 2 via finding $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P})$. However, we were unable to get very far by trying to solve this system directly. In stead we chose to follow in the footsteps of [8] and focus a set of low degree polynomials.

We first observe that $\mathcal{P}$ contains k linear polynomials. Following [8], we further observe that for every $a \in [\![0, sm]\!]$, the mapping from $\mathbb{F}_{q^m}$ to $\mathbb{F}_{q^m}$ given by $z \mapsto z^{p^a}$ is $\mathbb{F}_p$-linear and invertible. This enables us to find more linear equations that $\boldsymbol{x}$ satisfies. If we define $\sqrt[p^a]{\cdot} : \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$ to be the inverse of $z \mapsto z^{p^a}$, then the following is straight forward to prove

Lemma 13. *For any $a \in [\![0, sm]\!]$ such that $p^a \in [\![1, r]\!]$, and for every $i \in [\![1, k]\!]$ it holds that*

$$\sum_{j=1}^{n} \sqrt[p^a]{g_{i,j}}\, x_j = 0.$$

The secret key $\boldsymbol{x}$ is therefore additionally a zero of

$$\mathcal{L} \triangleq \left\{ \sum_{j=1}^{n} \sqrt[p^a]{g_{i,j}} X_j \;\middle|\; i \in [\![1,k]\!],\;\; p^a \in [\![1,r]\!] \right\}. \tag{5}$$

Solving linear equations is fast, but unfortunately we were not able to detangle $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{L})$ to obtain the secret key. However, we found that adding the following equations

$$\mathcal{P}_u \triangleq \left\{ \sum_{j=1}^{n} g_{i,j} X_j^u \;\middle|\; i \in [\![1,k]\!] \right\}. \tag{6}$$

where

$$u \triangleq \begin{cases} 3 & \text{for } p = 2 \\ 2 & \text{else} \end{cases}$$

gave a zero-set that we could detangle. [3]

An outline of our attack is as follows: We solve Problem 2 by identifying a particular vector space $V \subseteq \mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ such that $\boldsymbol{x} \in V$ and such that it is possible to recover $\boldsymbol{x}$ from the knowledge of V.

Finding a basis for V is much easier than finding zeros of $\mathcal{P}$ directly. In part because we can fix $\dim(V)$ many variables when we solve the system, and in part because we only have polynomials of low degree in our system. This means that we can break the BCH-McEliece cryptosystem for many parameters. See Section VII)

C. Identifying a vector space in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$.

An important feature of the attack given in [8] is to identify a relatively large vector space that is included in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$. The idea is to find sets $L \subseteq \mathbb{Z}_{q^m-1}$ which guarantees that

$$\operatorname{span}_{\mathbb{F}_{q^m}} \left\{ \boldsymbol{x}^\ell \;\middle|\; \ell \in L \right\} \subset \mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L}).$$

We now introduce a conditions that will help us to find such sets L. In the following, $\mathcal{C}_q(c)$ are the cyclotomic cosets,

$$\mathcal{C}_q(c) \triangleq \left\{ cq^e \pmod{q^m - 1} \;\middle|\; e \in [\![0, m-1]\!] \right\},$$

[3] The systems of [8] includes more non-linear equations than ours. There is a trade-off between the benefit of more equations and the dimension of the vector spaces in the solution set. Both our experiments and those of [8] indicate that it is more important to raise the dimension of the solution set than it is to keep more equations. With that in mind, we include the smallest possible number of non-linear equations necessary to obtain a solution space with a structure that allows finding the actual desired solution.

and $\mathcal{C}_q \triangleq \bigcup_{c\in[\![1,r]\!]} \mathcal{C}_q(c)$ is their union. Note that by the same reasoning as in lemma 12, $\cup_{c\in[\![1,r]\!]}\mathcal{C}_q(c)$ contains powers c for which we know that

$$\sum_{j=1}^{n} g_{i,j} x_j^c = 0 \text{ for all } i \in [\![1,r]\!]. \tag{7}$$

Definition 14. *For $u \in [\![1,r]\!]$, $L \subseteq \mathbb{Z}_{q^m-1}$ is said to satisfy the u-sum condition, if*

U1: for every $\ell \in L$ and any $p^a \in [\![1,r]\!]$, $p^a \cdot \ell \in \mathcal{C}_q$, and
U2: for any choice of $\ell_1, \ldots, \ell_u \in L$, $\ell_1 + \cdots + \ell_u \in \mathcal{C}_q$.

Proposition 15. *If $L \subseteq \mathbb{Z}_{q^m-1}$ satisfies the u-sum condition then*

$$\text{span}_{\mathbb{F}_{q^m}} \left\{ \boldsymbol{x}^\ell \;\middle|\; \ell \in L \right\} \;\subseteq\; \mathcal{Z}_{\mathbb{F}_{q^m}} \left(\mathcal{P}_u \cup \mathcal{L} \right).$$

Proof. We will prove the more precise statements:

1) U1 $\Rightarrow \text{span}_{\mathbb{F}_{q^m}} \left\{ \boldsymbol{x}^\ell \;\middle|\; \ell \in L \right\} \;\subseteq\; \mathcal{Z}_{\mathbb{F}_{q^m}} \left(\mathcal{L} \right)$.
2) U2 $\Rightarrow \text{span}_{\mathbb{F}_{q^m}} \left\{ \boldsymbol{x}^\ell \;\middle|\; \ell \in L \right\} \;\subseteq\; \mathcal{Z}_{\mathbb{F}_{q^m}} \left(\mathcal{P}_u \right)$.

For the first of these: Let $\boldsymbol{z} \in \text{span}_{\mathbb{F}_{q^m}} \{\boldsymbol{x}^\ell \mid \ell \in L\}$ be given for some L, which satisfies the U1 condition. Then

$$\boldsymbol{z} = \sum_{\ell \in L} c_\ell \boldsymbol{x}^\ell \tag{8}$$

for some c_ℓ's in $\mathbb{F}_{q^m}$. We now have to prove that

$$\sum_{j=1}^{n} \sqrt[p^a]{g_{ij}} z_j = 0 \tag{9}$$

for all $p^a \in [\![1,r]\!]$. But since raising to the power p^a is $\mathbb{F}_p$-linear for all elements in $\mathbb{F}_{q^m}$, this is equivalent to

$$\sum_{j=1}^{n} g_{ij} z_j^{p^a} = 0 \tag{10}$$

If we now plug in z_j from equation (8), and again use that raising to the power p^a is $\mathbb{F}_p$-linear, we see that

$$\sum_{j=1}^{n} g_{ij} z_j^{p^a} = \sum_{j=1}^{n} g_{ij} \sum_{\ell \in L} c_\ell^{p^a} x_j^{\ell \cdot p^a} = \sum_{\ell \in L} c_\ell^{p^a} \sum_{j=1}^{n} g_{ij} x_j^{\ell \cdot p^a}$$

Which is equal to zero by (7) as $\ell \cdot p^a \in \bigcup_{c\in[\![1,r]\!]} \mathcal{C}_q(c)$ for all l and p^a. This finishes the proof.

The proof of the second statement is similar. Except here, raising to the power u is not linear in general, so the last step becomes

$$\begin{aligned}
\sum_{j=0}^{n} g_{ij} z_j^u &= \sum_{j=0}^{n} g_{ij} \sum_{\ell_1,\ldots,\ell_u \in L} b_{\ell_1,\ldots,\ell_u} x_j^{\ell_1+\cdots+\ell_u} \\
&= \sum_{\ell_1,\ldots,\ell_u \in L} b_{\ell_1,\ldots,\ell_u} \sum_{j=0}^{n} g_{ij} x_j^{\ell_1+\cdots+\ell_u} \\
&= 0
\end{aligned}$$

□

Remark 16. Note that $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ might contain more than one vector space. In fact, for most parameters, it contains many vector spaces, which may or may not have overlapping subspaces. For example for $q = 8$, $m = 2$ and $r = 20$, all of the following sets satisfy the 3-sum condition:

- $\{1, 2, 3, 4, 6\}$ and $\{8, 16, 24, 32, 48\}$
- $\{1, 2, 3, 4, 8\}$ and $\{8, 16, 24, 32, 1\}$
- $\{1, 2, 8, 16\}$

The following lemma shows that some of the sets we found in remark 16 were no coincidence

Lemma 17. *If $L \subseteq \mathbb{Z}_{q^m-1}$ satisfies the u-sum criteria, then qL does as well.*

Proof. Relies on the simple observation that $a \in \mathcal{C}_q \Rightarrow qa \in \mathcal{C}_q$. □

For the attack, we need the size of the largest L as this tells us the number of variables we can fix. Once we fix this number of variables, we are unlikely to find solutions in the smaller vector spaces. Hence in the example of remark 16, if we fix 5 variables, then we are most likely to only find $2m$ solutions.

For the remainder of this article, we will denote by L a subset of $\mathbb{Z}_{q^m-1}$ of maximal size which fulfills the u-sum condition.

We were not able to find a closed formula for the size of L, but we developed two algorithms to determine L (next section). Furthermore, we found the following upper and lower limits:

Lemma 18. *Any L that fulfills the u-sum criteria has size at most rm. Conversely, the largest L's that fulfill the u-sum criteria, have size at least $\left\lfloor \frac{r}{\max U} \right\rfloor$, where*

$$U \triangleq \{u\} \cup \{p^a \mid a \in [\![0, s-1]\!], p^a \in [\![1,r]\!]\}$$

Proof. For the first statement, note that $L \subseteq \mathcal{C}_q$, and $\mathcal{C}_q$ has size at most rm. For the second, we claim that $\{1, .., \left\lfloor \frac{r}{\max U} \right\rfloor\}$ fulfills the u-sum criteria. The central step of proving the claim is to realize that

$$p^a \cdot l \in \mathcal{C}_q \Rightarrow q \cdot p^a \cdot l \in \mathcal{C}_q.$$

Hence, to check U1 of definition 14 it is sufficient to check p^a where $a \in [\![0, s-1]\!]$. □

D. Algorithms to find L's which fulfill the u-sum criteria

Here we present two algorithms to find L's of maximal size that fulfill the u-sum criteria. The first algorithm is fast but may not always find an L of maximal size. The second is guaranteed to find all L's of maximal size, but may have runtime exponential in r.

The first step for both of the algorithms is to create the set

$$S_U \triangleq \{l \in \mathbb{Z}_{q^m-1} \mid l \cdot u \in \mathcal{C}_q \text{ for all } u \in U\}$$

where U is defined in lemma 18. We create this set to limit the search space for L, as it is clear that $L \subseteq S_U$. Furthermore, S_U can be created efficiently by running through all elements l of $\mathbb{Z}_{q^m-1}$ and for each test if $l \cdot u \in \mathcal{C}_q$ for all $u \in U$.

In the fast algorithm (Algorithm 5), the idea is simply create L element by element. To initialize, we set $L = \{\}$, and then

for each element l of S_U, we check if $L \cup \{l\}$ fulfills the u-sum criteria. If it does, then we grow L and if not, then we leave out that particular element, and move onto the next. Based on our experience with how the sets look, we decided to sort the elements of S_U and start by adding small elements. We can check if a set L fulfills the u-sum criteria in time $O(|L|^3)$ (we need to calculate a sum for each choice of two or three elements from L). As we need to check a u-sum criteria for each $l \in S_U$, the total runtime of the algorithm is $O(|S_U|^4)$, which is $O(m^4 r^4)$ by the same argument as in lemma 18.

Algorithm 5 Find L Fast

Input: S_U and u
Output: A set $L \subseteq \mathbb{Z}_{q^m-1}$, which fulfills the u-sum criteria
Algorithm:
$L := \{\}$
for $l \in S_U$ **do**
  if $L \cup \{l\}$ fulfills the u-sum criteria **then**
    $L := L \cup \{l\}$
  end if
end for
return L

The second algorithm (Algorithm 6), which is guaranteed to find all L's of maximal size, leverages the maximal clique finding algorithms from graph theory. The idea is to construct a hypergraph where each vertex is an element of S_U, and a set of vertices is connected by an edge, if the set fulfills the u-sum criteria. Once this graph is constructed, finding L boils down to finding the maximal cliques of the graph. When $p \neq 2$, it is sufficient to consider edges of size two (the hypergraph is just a normal graph). This is because a set L fulfills the 2-sum criteria if and only if all subsets of L of size two fulfill the criteria. For $p = 2$, we need to consider hypergraphs, where all edges connect three vertices.

Algorithm 6 Find L Accurately for $u = 3$

Input: S_U
Output: All sets $L \subseteq \mathbb{Z}_{q^m-1}$ of maximal size, which fulfill the 3-sum criteria
Algorithm:
Initialize hypergraph $\mathcal{G}$ with nodeset S_U.
for $\{a, b, c\} \subseteq S_U$ **do**
  if $\{a, b, c\}$ fulfills the 3-sum criteria **then**
    add edge $\{a, b, c\}$ to $\mathcal{G}$
  end if
end for
Run algorithm for finding maximal cliques in $\mathcal{G}$
return Maximal cliques

In our code, we implemented the fast but inaccurate algorithm for $p = 2$, as magma does not support hypergraphs, and we did not run into any cases where the algorithm failed. For $p \neq 2$, we implemented the slow but accurate algorithm, and we used the clique finding algorithm of Magma, which is based on [17]. The runtime of this algorithm is not well understood in general, but we note that it is at most exponential in the number of vertices $|S_U|$, i.e. in r and m.

E. Repeatedly solving restrictions of the polynomial system

We use Magma's [12] Gröbner basis algorithm to find vector spaces of maximal dimension in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ as follows. The set $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ contains at least one vector space of dimension $|L|$. We can thus fix $|L|$ variables, and still find a solution in each vector space of maximal dimension. We run the Gröbner basis algorithm $|L|$ times, each time fixing $|L|$ variables differently, to find $|L|$ sets of solutions.

In the first run, we fix the first $|L|$ variables to

$$(1, \omega, \omega^2, .., \omega^{|L|-1})$$

where ω is a primitive n'th root of unity, that we choose. In the second run, we fix the variables to

$$(1, \omega^2, \omega^4, .., \omega^{2(|L|-1)})$$

etc. This choice of fixed variables is different from the choice of [8]. Our choice forces the solutions to have many different entries, which we found to be helpful when trying to avoid additional solutions outside of the vector spaces that we understand.

Let us denote the solution sets outputted by the Gröbner basis algorithm by $V_1, .., V_{|L|}$. If we denote by ν the number of vector spaces of maximal dimension in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$, then with high probability, V_i has exactly ν elements – one from each of these vector spaces. To obtain a basis for each of the maximal vector spaces, what is left to do is to "match up" the elements of V_i, i.e., to find sets W_j, $j = 1, ..., \nu$ such that $W_j \cap W_{j'} = \emptyset$ for $j \neq j'$, $|V_i \cap W_j| = 1$, and $\text{span}_{\mathbb{F}_{q^m}} W_j \subseteq \mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$. This is achieved via *Fröbenius Alignment*.

F. Step 4: Fröbenius Alignment

In [8], the above task is solved using a brute-force search, a routine with runtime exponential in $|L|$. This is possible since their L is not too big. However, as r grows, $|L|$ grows too rendering this step infeasible.

We we able to construct a faster solution by leveraging the following insight: if two vectors, $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ belong to the same vector space of zeros, then $\boldsymbol{v} + \boldsymbol{w}$ is also in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$, and if they don't belong to the same vector space, then most likely, $\boldsymbol{v} + \boldsymbol{w}$ is not in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$. Hence, we can build a basis W iteratively. First we choose $\boldsymbol{w}_1 \in V_1$. Then, for each i, we check whether $\boldsymbol{w}_1 + \boldsymbol{w}_i$ is a zero of $\mathcal{P}_u \cup \mathcal{L}$ for all $\boldsymbol{w}_i \in V_i$ and add the successful vector to W.

For each choice of $\boldsymbol{w}_1$ this gives a basis for a vector space in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$. If we loop through all the different choices of $\boldsymbol{w}_1$, we find bases for all the vector spaces in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ of maximal dimension.

Note that this algorithm in not fail safe: it is possible that $\boldsymbol{v} + \boldsymbol{w} \in \mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$, but $\boldsymbol{v}$ and $\boldsymbol{w}$ are not in the same vector space of maximal dimension. There are ways to make our Fröbenius Alignment algorithm more dependable, but for

all the parameters we tried, the simple version of the algorithm sufficed.

The algorithm is summarized in Algorithm 7. The runtime is $O(|L| \cdot \nu)$, where ν is the number of results given by the Gröbner basis calculation.

Algorithm 7 Fast Fröbenius Alignment

Inputs:

- $|L|$ lists of zeros of $\mathcal{P}_u \cup \mathcal{L}$: $V_1, ..., V_{|L|}$
- Index $j \in \{1, .., |V_1|\}$ (deciding which solution to pick from the first set)

Output: A set of vectors: $W := \{\boldsymbol{w}_1, .., \boldsymbol{w}_{|L|}\}$, such that

- $\boldsymbol{w}_i \in V_i$ for $i = 1, .., |L|$ and
- $\{\boldsymbol{w}_1, .., \boldsymbol{w}_{|L|}\}$ forms a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L\}$ for some L of maximum dimension fulfilling the u-sum condition.

Algorithm:

$\boldsymbol{w}_1 := V_1[j]$ (the j'th element on the list V_1)
for $i \in \{2, .., |L|\}$ **do**
  for $\boldsymbol{v} \in V_i$ **do**
    $b := 0$
    for $f \in \mathcal{P}_u \cup \mathcal{L}$ **do**
      if $f(\boldsymbol{w}_1 + \boldsymbol{v}) \neq 0$ **then**
        ($\boldsymbol{w}_1$ and $\boldsymbol{v}$ are not in the same vector space)
        $b := 1$
        break f-loop
      end if
    end for
    if $b = 0$
      $\boldsymbol{w}_i := \boldsymbol{v}$
      break $\boldsymbol{v}$-loop (go to next $i \in \{2, .., |L|\}$)
    end if
  end for
end for
return $\{\boldsymbol{w}_1, .., \boldsymbol{w}_{|L|}\}$

G. Step 5: Detanglement

Here we assume that we have a basis for all vector spaces of maximal dimension in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ and that we know all the corresponding L's. We further assume that we know which L belongs to which basis (we can achieve this by trying all combinations and seeing which produce the expected results in the upcoming calculations[4]).

In order to use this input to find the secret key, $\boldsymbol{x}$, we use the same idea as [8]. Their idea relies on the following two propositions[5] :

Proposition 19. *Let $a \in [\![0, sm-1]\!]$ and $W = \{\boldsymbol{w}_1, .., \boldsymbol{w}_{|L|}\}$ be one of the sets returned by Fröbenius alignment. If W is a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in L\}$ then $W^{p^a} \triangleq \{\boldsymbol{w}_1^{p^a}, .., \boldsymbol{w}_{|L|}^{p^a}\}$ is a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in p^a L\}$, where $p^a L \triangleq \{p^a \ell \mid \ell \in L\}$.*

Proof. To ease notation, we just show the proposition for $a = 1$. The idea generalizes directly to other a's.

That $\text{span}_{\mathbb{F}_{q^m}} W^p \subseteq \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in pL\}$ follows easily from $z \mapsto z^p$ being $\mathbb{F}_p$-linear for all $z \in \mathbb{F}_{q^m}$. To prove that the vectors of W^p are linearly independent, we note that for $w_i \in W$, the first entries of w_i were chosen to be $(1, \omega^i, \omega^{2i}, .., \omega^{|L|i})$ in section V-E, so the first entries of w_i^p are $(1, \omega^{ip}, \omega^{2ip}, .., \omega^{i|L|p})$. Furthermore, the matrix

$$\begin{bmatrix} 1 & \omega^p & \omega^{2p} & \dots & \omega^{|L|p} \\ 1 & \omega^{2p} & \omega^{4p} & \dots & \omega^{2|L|p} \\ \vdots & & & & \vdots \\ 1 & \omega^{|L|p} & \omega^{|L|p} & \dots & \omega^{|L|^2 p} \end{bmatrix}$$

has full rank, as it is a Vandermonde-matrix. Hence the vectors of W^p are linearly independent □

Proposition 20. *For $L_1, L_2 \subseteq \mathbb{Z}_{q^m-1}$*

$$\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L_1\} \ \cap \ \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L_2\}$$
$$= \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L_1 \cap L_2\}$$

Proof. First, note that the inclusion "$\supseteq$" is trivial. To prove the other inclusion, we show that the two vector spaces have the same dimension. As $\boldsymbol{x}^0, ..., \boldsymbol{x}^{n-1}$ are linearly independent per Lemma 26, the dimension of the right hand side is $|L_1 \cap L_2|$. To find the dimension of the left hand side, we use the formula $\dim(V_1 + V_2) = \dim(V_1) + \dim(V_2) - \dim(V_1 \cap V_2)$. In our case, the sum of vector spaces is equal to

$$\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L_1\} + \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L_2\}$$
$$= \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L_1 \cup L_2\}$$

So it has dimension $|L_1 \cup L_2|$. From this we deduce that the left hand side has dimension

$$|L|_1 + |L|_2 - |L_1 \cup L_2| = |L_1 \cap L_2|$$

Exactly like the right hand side. This finishes our proof. □

Using these two ideas and the attack on GRS codes by Sidelnikov and Shestakov [4], we can detangle the solution space for many parameters. We first illustrate the idea with an example, and then we will describe the idea in more detail.

Example 21. For $q = 25$, $m = 2$ $r = 100$, we are given $W = \{w_1, .., w_{26}\}$, which is a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L\}$ with

$$L = [\![1, 20]\!] \cup \{25, 30, 35, 40, 45, 50\}.$$

Using proposition 19 and 20 along with the fact that $5^{-1} = 5^3$ in $\mathbb{Z}_{q^m-1}$, we see that

$$(\text{span}_{\mathbb{F}_{q^m}} W) \cap (\text{span}_{\mathbb{F}_{q^m}} W^{5^3})$$
$$= \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \ : \ \ell \in L \cap 5^{-1}L\}$$

with $L \cap 5^{-1}L = [\![1, 10]\!]$. Note that calculating a basis for $(\text{span}_{\mathbb{F}_{q^m}} W) \cap (\text{span}_{\mathbb{F}_{q^m}} W^{5^3})$ is simple linear algebra given

[4]The number of vector spaces of maximal dimension is not very big. The largest we saw was $3m$, so this step of exhaustive search is not a problem

[5]The authors of [8] were not able to prove the second proposition in their setting. Our proof does not apply in their setting, either, as it is not clear whether their vectors are linearly independent.

W. Hence we are able to find a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell : \ell \in [\![1, 10]\!]\}$.

Now we just repeat the process. If we again raise to the power 5^3 and take the intersection, we find a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^1, \boldsymbol{x}^2\}$. This is a basis for the GRS code $\mathsf{GRS}_2(\boldsymbol{x}, \boldsymbol{x})$, so we can use the Sidelnikov-Shestakov attack [4] to find $\boldsymbol{x}$.

From this example we already see the outline of how detangle the solution sets. However, the idea used in example 21 relies on L having a very particular structure. We will call any L that has this structure well formed.

Definition 22. *$L \subseteq \mathbb{Z}_{q^m-1}$ is called* well-formed of length *$\sigma \in [\![1, sm-1]\!]$ if*

- $L \subseteq [\![1, p^\sigma - 1]\!]$
- $p^a \in L$ *for all* $a \in [\![0, \sigma - 1]\!]$
- *For every* $c \in [\![2, p-1]\!]$*:*

$$\forall a \in [\![0, \sigma-1]\!] : p^a \cdot c \in L \Rightarrow$$
$$\forall a \in [\![0, \sigma-1]\!], \forall b \in [\![2, c-1]\!] : p^a \cdot b \in L$$

Note that if such a σ exists, it is unique.

The remainder of this section will be used to prove that the strategy of example 21 works whenever L is well formed. In appendix B, we give experimental evidence to show that it is always possible to find an L which is well-formed when r is small enough. In section VII, we will see that the Gröbner basis step of our attack is the limiting factor, and this step has a runtime which increases in r. In the appendix we will show that it is possible to find well-formed L's for r's which are much bigger than the largest r's for which we are able to run the Gröbner basis step of our attack. In other words, the problem of finding well-formed L's is, in practice, not the limiting factor of our attack.[6]

To prove that the detanglement-idea of example 21 always works when L is well-formed, we start out by formalizing the step we repeated twice in the example. Here and throughout the rest of this section, we will use the notation of proposition 19:

Definition 23. *A* detanglement step *is an algorithm, which takes as input (L, W), where $L \subseteq \mathbb{Z}_{q^m-1}$ and $W \subseteq \mathbb{F}_{q^m}^n$, and outputs (L', W') where $L' = L \cap p^{-1}L$ and W' is a basis for $\text{span}_{\mathbb{F}_{q^m}} W \cap \text{span}_{\mathbb{F}_{q^m}} W^{p^{sm-1}}$.*

Note that the output is efficiently computable from the input.

Now for the key lemma of detanglement:

Lemma 24. *If the input (L, W) of a detanglement step satisfies*

- *L is well formed with length $\sigma > 1$*
- *W is a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in L\}$*

Then the output (L', W') will satisfy

- *L' is well formed with length $\sigma - 1$*
- *W' is a basis for $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in L'\}$*

Proof. As $p^{-1} = p^{sm-1}$ in $\mathbb{Z}_{q^m-1}$, the statement about W' follows from proposition 19 and 20.

We now turn our attention to $L' = L \cap p^{-1}L$. Throughout this proof, we will use the fact that

$$b \in L' \Leftrightarrow b \in L \text{ and } pb \in L \tag{11}$$

which follows directly from the definition of L'.

To show $L' \subseteq [\![1, p^{\sigma-1} - 1]\!]$, we first note that $L' \subseteq L \subseteq [\![1, p^\sigma - 1]\!]$. We claim that $L' \cap [\![p^{\sigma-1}, p^\sigma - 1]\!] = \emptyset$. To prove this claim, it is sufficient to show that for every $b \in [\![p^{\sigma-1}, p^\sigma - 1]\!]$, pb is not in L. First observe that

$$b \in [\![p^{\sigma-1}, p^\sigma - 1]\!] \Rightarrow bp \in [\![p^\sigma, p^{\sigma+1} - p]\!]$$

Next, observe that $[\![p^\sigma, p^{\sigma+1} - p]\!] \cap L = \emptyset$ (note that here we need $\sigma \leqslant sm - 1$ as it implies $p^{\sigma+1} - p \leqslant q^m - 1$, so nothing "wraps around"). These two observations together give a proof of the claim, and we can thus conclude that L' fulfills the first criteria of well formed.

For the second criteria of well formed, we have to show that $p^a \in L'$ for all $a \in [\![0, \sigma - 2]\!]$. Per Equation (11), we just have to show that both $p^a \in L$ and $p \cdot p^a \in L$, but this is clear.

For the third criteria, first note that if $p = 2$, the statement is vacuously true. Hence for this proof, we assume $p > 2$. Now let $c \in [\![2, p-1]\!]$ be given such that

$$\forall a \in [\![0, \sigma - 2]\!] : p^a c \in L'.$$

By Equations 11 this implies

$$\forall a \in [\![0, \sigma - 1]\!] : p^a c \in L.$$

As L is well-formed, we can conclude that

$$\forall a \in [\![0, \sigma - 1]\!], \forall b \in [\![2, c-1]\!] : p^a b \in L.$$

Again by Equations 11 this implies

$$\forall a \in [\![0, \sigma - 2]\!], \forall b \in [\![2, c-1]\!] : p^a b \in L',$$

which is what we wanted to prove. □

If we repeat the detanglement step $\sigma - 1$ times, we end out with an L which is well formed with length 1. I.e. we will have a basis W of $\text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell : \ell \in L'\}$, where $L' = [\![1, |L'|]\!]$. I.e. W is a basis for the GRS code $\mathsf{GRS}_{|L|}(\boldsymbol{x}, \boldsymbol{x})$, so we can apply the Sidelnikov-Shestakov attack [4] to find $\boldsymbol{x}$.

The algorithm for detanglement is summarized in Algorithm 8 Detanglement.

For each W, the algorithm has to perform the detanglement step at most $sm-2$ times. The detanglement step is polynomial in n. Furthermore, the Sidelnikov-Shestakov attack has to be run. This also have runtime polynomial in n. As there are ν different W's, the total runtime of the algorithm is $poly(n) \cdot \nu$.

[6] We suspect that even when no L's are well-formed, we would still be able to detangle most solution spaces through a clever combination of the ideas in example 21 and multiple runs of a Gröbner basis algorithm with different polynomial inputs.

Algorithm 8 Detanglement

Input:

- $Ws = \{W_1, .., W_\nu\}$, a set of bases of all vector spaces in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$ of maximal dimension.
- $L \subseteq \mathbb{Z}_{q^m-1}$, which is well-formed and such that $\text{span}_{\mathbb{F}_{q^m}} W_i = \text{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in L\}$ for some $i \in [\![1, \nu]\!]$.

Output: A vector $\boldsymbol{v}$, that can be used as secret key to decrypt messages

for $W \in Ws$ **do**
  $\sigma \leftarrow$ length of L (see Def. 22)
  $L' \leftarrow L$
  while $\sigma > 1$ **do**
    $L' \leftarrow L' \cap p^{-1}L'$
    $\sigma \leftarrow$ length of L'
    $W \leftarrow$ basis for $(\text{span}_{\mathbb{F}_{q^m}} W) \cap (\text{span}_{\mathbb{F}_{q^m}} W^{p^{sm-1}})$
    if $|L'| \neq |W|$ **then**
      (W is not a basis of $\text{span}\{\boldsymbol{x}^\ell \mid \ell \in L'\}$)
      continue W-loop (go to next $W \in Ws$)
    end if
  end while
  Run the Sidelnikov Shestakov attack with input W
  $\boldsymbol{v} \leftarrow$ output of Sidelnikov Shestakov attack
  if G is a generator matrix for $\mathscr{A}_r(\boldsymbol{v}, \boldsymbol{v})$ **then**
    return $\boldsymbol{v}$
  end if
end for

VI. Theoretical analysis of the runtime

When we analyze the runtime of the attack, it is helpful to split the attack into four parts:

1) Find L. The runtime of this part is not well understood, but it is potentially exponential in r.
2) Find a Gröbner basis in the grevlex order for the ideal generated by $\mathcal{P}_u \cup \mathcal{L}$. The runtime is well understood for random systems, but our results in the next section will show that our system does not behave like a random system. Hence, analyzing the precise complexity of the Gröbner basis step is a hard task. One difficulty is to predict the number of solutions to the system (see next section). In any case, even if the complexity of this step remains exponential in the parameters, it is practical for some cases, and that is what we want to emphasize in this article.
3) Transform the Gröbner basis in the grevlex order into a Gröbner basis in the lex-order using the FGLM algorithm [18]. This algorithm has complexity $O(ND^3)$ where D is the number of solutions to the system counted with multiplicities and N is the number of variables. D varies greatly with parameters (see next section), and therefore we cannot estimate the runtime analytically. However, once we have a Gröbner basis in the lex order, it is easy to compute the solutions of the system.
4) Fröbenius align and detangle the set of solutions to find one vector, which can be used like the secret key to decrypt messages. The runtime of this part is polynomial in n and ν, the number of solutions of the system (without multiplicities).

In practice we find that part 2 and 3 are the deciding factors of whether the attack is feasible or not. Analyzing analytically their running times is a hard task, on which we are working. In the next section we have collected our initial findings on the running time of part 2.

A. Running time of part 2

When analyzing the runtime of the Gröbner basis algorithm, the first step is often to count the number of polynomials and the number of variables. In one way, this is easy: We have $s \cdot k$ linear polynomials, k polynomials of degree 2 or 3 and $n - |L|$ variables (as we can fix $|L|$ of the n variables). However, not all of the polynomials are linearly independent.

Proposition 25. *The number of linearly independent polynomials in $\mathcal{L}$ is less than or equal to $n - |S_p|$, where*

$$S_p := \{\ell \in \mathbb{Z}_{q^m-1} \mid \forall p^a \in [\![1, r]\!] : \ell \cdot p^a \in \mathcal{C}_q\}$$

and $\mathcal{C}_q$ is defined in section V-C.

Experimentally, we find that the inequality is an equality for all parameters we tried. To prove the proposition, we need a short lemma

Lemma 26. *The vectors $\boldsymbol{x}^0, ..., \boldsymbol{x}^{n-1}$ are linearly independent.*

Proof. $\boldsymbol{x}$ is a vector of length n. Hence the $n \times n$ matrix, whose columns are $\boldsymbol{x}^i$ for $i = 0, .., n-1$ is a square Vandermonde matrix. We know that a Vandermonde matrix has full rank iff all entries in the generating vector are different. $\boldsymbol{x}$ is a permutation of $(1, \alpha, \alpha^2, ..., \alpha^{n-1})$. So all entries in $\boldsymbol{x}$ are different. □

Now we are ready to prove the proposition

Proof of proposition 25. If we define $G_a := \sqrt[p^a]{G}$, where the root is applied to all entries of G, then by the rank-nullity theorem, the number of linearly independent polynomials in $\mathcal{L}$ is equal to

$$n - \dim(\bigcap_{p^a \in [\![1,r]\!]} N(\sqrt[p^a]{G}))$$

where $N(\sqrt[p^a]{G})$ is the null-space of $\sqrt[p^a]{G}$. We will now show that if $\ell \in \mathbb{Z}_{q^m-1}$ satisfies $\ell \cdot p^a \in \mathcal{C}_q$ for all $p^a \in [\![1, r]\!]$ then $\boldsymbol{x}^\ell \in N(\sqrt[p^a]{G})$. To do this, we first note that $\ell \cdot p^a \in \mathcal{C}_q$ implies that

$$\sum_{j=1}^{n} g_{ij} x_i^{\ell \cdot p^a} = 0$$

for all $i = 1, .., k$ and $p^a \in [\![1, r]\!]$. But by taking the p^a'th root, we see that this implies

$$\sum_{j=1}^{n} \sqrt[p^a]{g_{ij}} x_i^{\ell} = 0$$

for all $i = 1, .., k$ and $p^a \in [\![1, r]\!]$. Which is equivalent to $\boldsymbol{x}^\ell \in N(\sqrt[p^a]{G})$. Hence

$$\{\boldsymbol{x}^\ell \mid \forall p^a \in [\![1, r]\!] : \ell \cdot p^a \in \mathcal{C}_q\} \subseteq \bigcap_{p^a \in U_p} N(\sqrt[p^a]{G})$$

Per Lemma 26, powers of $\boldsymbol{x}$ are linearly independent, so the dimension of $\bigcap_{p^a \in [\![1,r]\!]} N(\sqrt[p^a]{G})$ is at least $|S_p|$. Hence the number of linearly independent equations is at most $n - |S_p|$, which finishes the proof. □

Of course, it would be nice, if we could find a closed formula for the number of linear equations, but we were unable to do this. However, we can say that

Lemma 27. *The size of the set in proposition 25 satisfies*

$$\left\lfloor \frac{r}{\max U} \right\rfloor \leqslant |S_p| \leqslant mr,$$

where

$$U \triangleq \{u\} \cup \{p^a \mid a \in [\![0, s-1]\!], p^a \in [\![1, r]\!]\}$$

Proof. Same idea as the proof of lemma 18. □

Now, if we use the linear equations of $\mathcal{L}$ to eliminate variables from the equations of degree 2 or 3, we find that the total number of variables that the Gröbner basis algorithm has to deal with is

$$n - |L| - (n - |S_p|) = |S_p| - |L| \leqslant mr. \tag{12}$$

However, once we eliminate variables, the k equations of degree u are no longer linearly independent. Unfortunately we were not able to make much progress on how many linearly independent equations the Gröbner basis algorithm has access to once $|L|$ variables are fixed and $(n - |S_p|)$ are eliminated using the linear equations.

VII. Results from experiments

We split this section into five subsections. In the first, we look at the number of solutions with and without multiplicities that the Gröbner basis algorithm returns. The large variability of these numbers support our claim from the previous section that our system does not behave like a random system. In the second subsection, we report on experimental results for the runtime of the Gröbner basis algorithm. In the third we add more polynomials of low degree to our system and give experimental evidence that this increases the runtime of the attack. This supports our choice of adding as few non-linear equations to the system as possible. In the fourth and fifth subsections, we tackle the remaining two defenses of [9]: "puncturing" and "subcode". We choose to add them one at a time, in order to better understand how they individually influence our attack. Experimentally we find, that when we combine them, the effects are simply combined.

We run magma V2.26-9 on a virtual machine with a 4x Single Core CPU running at 1996 MHz. We capped calculations at 10 GB ram, and stopped them if they ran for more than five hours.[7]

[7]The code for our experiments is available on https://github.com/FrejaElbro/Cryptanalysis-of-BCH-McEliece.

A. The number of solutions to the Gröbner basis algorithm

In general each run of the Gröbner basis algorithm produces as many results as there are vector spaces of maximal dimension in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L})$. For most parameters, there are m vector spaces of maximal dimension, but we have also observed other possibilities. In rare instances the Gröbner Basis algorithm finds fewer solutions than there are vector spaces of maximal dimension. This can happen, as the vector spaces of maximal dimension are not disjoint[8]. We can also find more solutions than there are vector spaces of maximal dimension. This happens if we fix the variables in a way that allows for a solution from a vector space of lower dimension[9]. Furthermore, we sometimes find a lot of extra solutions that we have no control over. This is particularly true for $q = 2$. Here the runtime is dominated by part 3 in section VI. For most other parameters, part 2 of section VI is the dominating part.[10]

Aside from the many possibilities for the number of solutions, there is also a huge variability in the multiplicities of the solutions. The largest multiplicity we saw was for $q = 29$, $m = 2$ and $r = 27$. Here we found just two solutions, but each of them had multiplicity 85. A parameter set, which we though should behave fairly similarly is $q = 29$, $m = 2$ and $r = 26$, but here each solution only had multiplicity 5. And close to this is $q = 29$, $m = 2$ and $r = 30$, where each solution had multiplicity 1. We were not able to explain these large variations.

Also the multiplicities varied for different runs of the Gröbner basis algorithm. In Table I we have recorded the number of solutions with and without multiplicities for different parameter sets and ten independent runs of the Gröbner basis algorithm. For all other results than $q = 4$, $r = 3, 4, 5$, the number of results is equal to the number of vector spaces of maximal dimension in $\mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P} \cup \mathcal{L})$.

B. The runtime of the Gröbner basis algorithm

In this section, we provide examples, where our attack is practical, but both information set decoding (ISD) and the support splitting algorithm suggest cryptographic security. The support splitting algorithm is a relevant attack, as there are only $\phi(n)$ BCH codes for any given choice of (p, s, m, r)[11], so the security of BCH-McEliece relies crucially on the permutation in the key generation algorithm 4. This permutation can be found using the support splitting algorithm. However,

[8]For example for $q = 7$, $m = 4$ and $r = 11$, both $\{1, 2, 3, 4, 5\}$ and $\{1, 2, 3, 4, 7\}$ fulfill the u-sum criteria. When we fix 5 variables during the Gröbner basis computation, we are essentially picking an element at random in each of the corresponding vector spaces. With probability $1/n$ that vector also lies in the intersection of the vector spaces, so instead of one solution from each of the two vector spaces, we will find one solution, which happens to lie in both.

[9]See for example the sets in remark 16. With probability $1/n$ we find a solution in the vector space $\mathrm{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^8, \boldsymbol{x}^{16}\}$ when we fix 5 variables.

[10]The other parameter sets where we found extra solutions that we have no control over all have small values of r and $q = 3, 4$ or 8.

[11]The only free variable the primitive n'th root of unity, and there are $\phi(n)$ of them

r	solutions	with mult.
2	3	3
3	3	9
4	3	3
5	3	12
6	3	3
7	3	15
8	3	3
9	3	24
10	3	6 or 9
11	3	63
12	3	39
13	3	39
14	3	3
15	3	39
16	3	3
17	3	39
18	3	6
19	3	48
20	3	6
21	3	63
22	3	9
23	6	129
24	6	45
25	3	6
26	3	6
27	3	54

r	solutions	with mult.
3	132	136
4	132	136
5	132	152
6	4	16 or 20
7	4	64
8	4	64
9	4	8 or 12
10	8	160
11	4	16
12	4	16 or 20
13	4	48

TABLE I
NUMBER OF SOLUTIONS WITH AND WITHOUT MULTIPLICITIES FOR TEN RUNS OF THE GRÖBNER BASIS ALGORITHM. ON THE LEFT FOR $q = 13$ AND $m = 3$ AND ON THE RIGHT FOR $q = 4$ AND $m = 4$

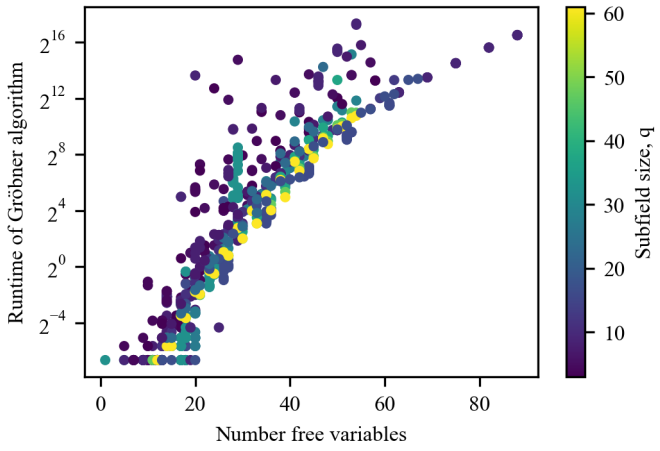

Fig. 1. The runtime of the Gröbner basis part of our attack in seconds (part 2 and 3 of section VI) as a function of number of free variables and the size of the subfield, q.

TABLE II
THE INFLUENCE OF s ON HOW ISD-SECURE SYSTEMS WE CAN BREAK

p	s	n	r	rate	pols	vars	GB	ISD	Hull
2	2	4095	13	0.99	390	32	2^{40}	2^{71}	60
2	3	4095	35	0.97	596	39	2^{42}	2^{121}	124
2	4	4095	113	0.92	1317	61	2^{47}	2^{247}	318
3	1	728	12	0.93	398	42	2^{44}	2^{51}	48
3	2	728	47	0.83	426	46	2^{49}	2^{87}	126
3	3	728	168	0.60	365	40	2^{43}	2^{150}	216

"rate" is the code-rate $= k/n$.
"pols" is the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$ once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"vars" is the number of variables in the system once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"GB" is the number of clock cycles expended on the Gröbner basis part of our attack including all $|L|$ runs and both part 2 and 3 of section VI). The remaining parts of our attack are insignificant in comparison.
"ISD" is the expected number of q'ary operations used by Prange's alg. (asymptotically the best generic decoding algorithm for BCH codes [3]).
"Hull" is the dimension of the hull of the code.

this algorithm is infeasible when the dimension of the hull is large.

We organize this section as an investigation into the influence of r, s, m and p respectively. In turn, we will vary one parameter and keep the remaining fixed to the greatest extend possible. Our aim is to understand how each of the parameters influences the runtime of the Gröbner basis algorithm.

The easiest parameter to understand the influence of is r. r influences the runtime in two different ways. Increasing r leads both to an increase in the number of variables (Equation (12)), and a decrease in the dimension of the BCH code (Theorem 11), and hence the number of non-linear equations in $\mathcal{P}_u$. Predictably, the effect of increasing the number of variables is to increase the runtime of the Gröbner basis algorithm. In Fig. 1 we have plotted all our results in one graph. We see that the runtime of the Gröbner basis algorithm seems exponential in the number of variables.

To visually compare our results with information-set decoding, we also graph our results as a function of the rate (which is closely connected to r through proposition 11). The results are in Fig. 2. We conclude that our attack works best for high rates, which was to be expected as high rate corresponds to low r and hence low number of variables and many equations. The precise cutoff point between when our attack is better than information-set decoding depends on the other parameters.

The easiest parameter after r to understand is s, that is the extension degree of the base field $\mathbb{F}_q$. The larger the extension degree, the more linear equations can be added by taking p'th roots, which leads to fewer variables and hence lower runtime of our attack. We ran some experiments, where we kept n constant, but varied s. Our experiment consisted of running the Gröbner basis part of our attack for larger and larger r, and stopping when either the runtime exceeded 5 hours or when memory exceeded 10 GB. The results are in Table II. We see that for both $p = 2$ and $p = 3$, the effect of s is clearly to bring down runtime of the Gröbner basis algorithm, and hence bring more parameter sets into the range we can attack with our limited resources.

We now explore the effect of m on the runtime. We ran similar experiments as for s to find the results of Table III. We find that increasing m leads to us being able to break systems with higher ISD security. This is in part because increasing m increases n, which increases the expected number of rounds the Prange Algorithm [2] has to run to find the error vector. However, we can also see that when we increase m, we are

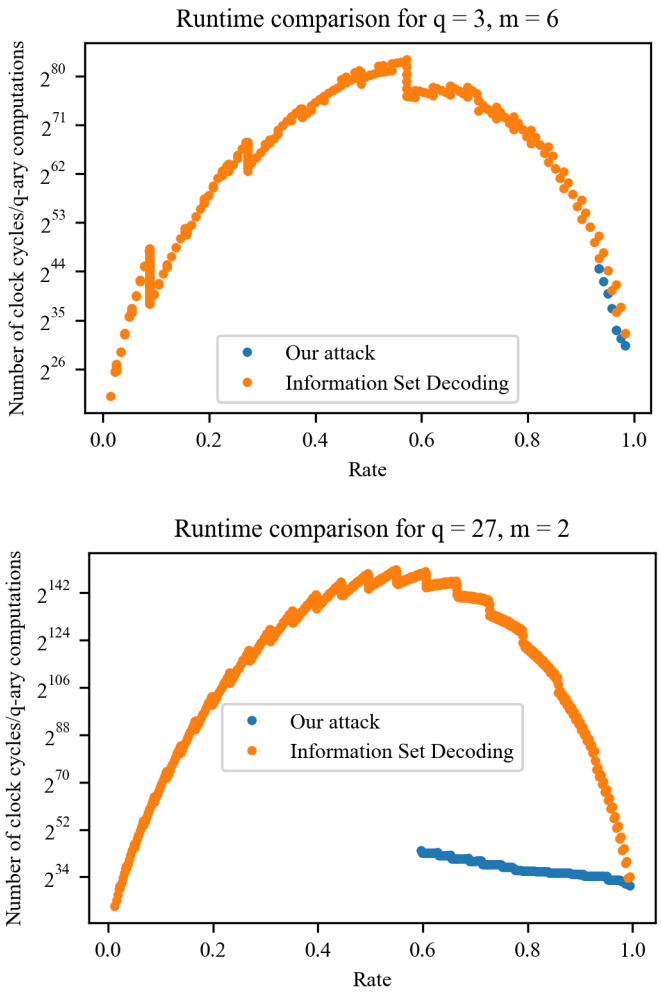

Fig. 2. Comparison between the runtime of our attack and the best previously known attack (information-set decoding). We estimate the runtime of information-set decoding as the expected number of q-ary computations that the Prange algorithm [2] needs to find the error-vector. For our attack, the runtime is estimated as the number of clock cycles used to run part 2 and 3 of our attack $|L|$ times (section VI). The remaining steps of our attack are insignificant in comparison. As the rate decreases, r increases, so the number of free variables increases. This explains why the runtime of our attack increases for decreasing rates, and why we were only able to run our attack for high rates.

TABLE III
THE INFLUENCE OF m ON HOW ISD-SECURE SYSTEMS WE CAN BREAK

q	m	n	r	k	pols	vars	GB	ISD	Hull
3	5	242	12	202	170	34	2^{45}	2^{38}	40
3	6	728	12	680	398	42	2^{44}	2^{51}	48
3	7	2186	12	2130	714	50	2^{45}	2^{63}	56
3	8	6560	12	6496	1086	58	2^{47}	2^{76}	64
5	3	124	13	91	83	27	2^{45}	2^{31}	33
5	4	624	16	572	404	44	2^{48}	2^{55}	52
5	5	3124	15	3064	840	53	2^{47}	2^{73}	60
9	2	80	32	31	31	20	2^{48}	2^{28}	29
9	3	728	47	602	426	46	2^{49}	2^{87}	126
9	4	6560	50	6380	1686	69	2^{49}	2^{168}	180

"pols" is the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$ once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"vars" is the number of variables in the system once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"GB" is the number of clock cycles expended on the Gröbner basis part of our attack including all $|L|$ runs and both part 2 and 3 of section VI). The remaining parts of our attack are insignificant in comparison.
"ISD" is the expected number of q'ary operations used by Prange's alg. (asymptotically the best generic decoding algorithm for BCH codes [3]).
"Hull" is the dimension of the hull of the code.

TABLE IV
THE INFLUENCE OF p ON HOW ISD-SECURE SYSTEMS WE CAN BREAK

p	s	n	r	k	pols	vars	GB	ISD	Hull
3	1	728	12	680	398	42	2^{44}	2^{51}	48
3	1	2186	12	2130	714	50	2^{45}	2^{63}	56
5	1	624	16	572	404	44	2^{48}	2^{55}	52
5	1	3124	15	3064	840	53	2^{47}	2^{73}	60
13	1	2196	26	2124	793	59	2^{47}	2^{97}	72
29	1	840	36	771	771	51	2^{46}	2^{95}	67
37	1	1368	34	1300	1069	51	2^{45}	2^{106}	68

"pols" is the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$ once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"vars" is the number of variables in the system once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"GB" is the number of clock cycles expended on the Gröbner basis part of our attack including all $|L|$ runs and both part 2 and 3 of section VI). The remaining parts of our attack are insignificant in comparison.
"ISD" is the expected number of q'ary operations used by Prange's alg. (asymptotically the best generic decoding algorithm for BCH codes [3]).
"Hull" is the dimension of the hull of the code.

able to solve systems with more variables within the same time window. We suspect this is because we are increasing k, which gives us more polynomials for the Gröbner basis algorithm to work with.

Finally, we look at the effect of p. This effect was the most difficult to isolate, as we cannot keep all the other parameters constant. Here we picked parameters so that $n \in [600, 4000]$. The results are in Table IV. We see that the effect of increasing p is to increase the ISD security of the largest parameters we are able to break in our setup.

C. Adding more non-linear equations of low degree

In section V-B, we took p^a'th roots and found a set of linear polynomials that has $\boldsymbol{x}$ as a zero. This idea is not limited to linear polynomials. We can similarly show that $\boldsymbol{x}$ is a zero of

$$\mathcal{L}_u \triangleq \left\{ \sum_{j=1}^{n} \sqrt[p^a]{g_{i,j}} X_j^u \;\middle|\; i \in [\![1,k]\!],\;\; u \cdot p^a \in [\![1,r]\!] \right\}. \quad (13)$$

for any $u \in [\![1,r]\!]$, and in particular for $u = 2$ or 3. This gives extra non-linear equations of low degree which can be added to the system. However, adding extra equations to the system limits the solution set. We find that

Proposition 28. *If $L \subseteq \mathbb{Z}_{q^m-1}$ satisfies the u-sum condition and furthermore*

TABLE V
THE INFLUENCE OF INCLUDING POLYNOMIALS FROM $\mathcal{L}_u$ ON THE RUNTIME OF THE ATTACK

q	r	n	$\mathcal{L}_u$	$\lvert L\rvert$	vars	pols	1GB time	Total time
25	50	624	no	13	19	160	0.4	15
			yes	5	27	223	7	43
25	100	624	no	26	38	376	89	2390
			yes	10	54	495	>18000	
27	50	728	no	11	13	103	0.01	10
			yes	3	21	168	0.7	9
27	100	728	no	22	28	276	7	205
			yes	6	44	411	7824	46971
16	50	255	no	10	14	74	0.06	2
			yes	2	22	130	1134	2291

$\mathcal{L}_u$ indicates whether the polynomials of $\mathcal{L}_u$ are included or not.
"vars" is the number of variables in the system once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"pols" is the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$ once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
1GB is the time in seconds to run one round of the Gröbner basis algorithm (including both part 2 and 3 of section VI).
"Total time" is the total time in seconds to run our entire attack.

U3: for any $a \in [\![0, sm]\!]$ such that $up^a \in [\![1, r]\!]$ and any choice of $\ell_1, \ldots, \ell_u \in L$ it holds that $p^a(\ell_1 + \cdots + \ell_u) \in \mathcal{C}_q$.

then

$$\operatorname{span}_{\mathbb{F}_{q^m}} \left\{ \boldsymbol{x}^\ell \;\middle|\; \ell \in L \right\} \subseteq \mathcal{Z}_{\mathbb{F}_{q^m}} \left(\mathcal{P}_u \cup \mathcal{L} \cup \mathcal{L}_u \right).$$

Proof. To prove U3 $\Rightarrow \operatorname{span}_{\mathbb{F}_{q^m}} \left\{ \boldsymbol{x}^\ell \;\middle|\; \ell \in L \right\} \subseteq \mathcal{Z}_{\mathbb{F}_{q^m}} \left(\mathcal{L}_u \right)$ we use the same ideas as in the proof of 15 □

Hence by including more equations, we are effectively decreasing the dimension of the vector spaces in the solution set. This means that we can fix fewer variables, when we solve the system. However, we also have to solve the system fewer times. In table V we have recorded results from experiments where we either included the polynomials of $\mathcal{L}_u$ or not. We find that for small runtimes of the Gröbner basis algorithm, there is a slight advantage to including more polynomials. This advantage comes from the fact that when the runtime of the Gröbner basis algorithm is low, the other steps of the attack dominate. And some of these have runtimes which increase in $|L|$. For large runtimes of the Gröbner basis algorithm, however, there is a clear advantage to not include the polynomials from $\mathcal{L}_u$.

D. How puncturing (defense 3 in the challenge of [9]) affects our attack

Here we consider subfield subcodes of the codes

$$\mathsf{GRS}_r \left(\boldsymbol{x}, \boldsymbol{x} \right)^\perp$$

where the only restriction on $\boldsymbol{x}$ is that it must consist of non-zero distinct elements. Note that these codes are in general not BCH codes, but they are still Goppa codes on the form $\mathscr{G}(\boldsymbol{x}^{-1}, X^r)$. Note also that attacking a McEliece-like scheme based on these codes correspond to breaking the challenge of [9] with 5 out of 6 defense strategies (scaling, permutation, subfield, wildness and puncturing).

Theoretically, our attack works just as well for these subfield subcodes as it does for permuted BCH codes. The u-sum condition only depends on p, s, m and r, so puncturing the GRS code does not change L. The only limitations we have to set is $\max(L) \leqslant n$, as otherwise we can no longer guarantee that the vectors in the proof of proposition 20 are linearly independent (however, the proposition might still be true) and $\max(S_p) \leqslant n$ for the same reason, but now in the proof of proposition 25 (again the proposition may still be true).

However, even for the permuted BCH codes, we have not been able to prove that all steps of our attack work. Hence we rely on the following experimental facts

(A1) We can explain almost all of the zeroes of $\mathcal{P}_u \cup \mathcal{L}$
(A2) Finding zeros of $\mathcal{P}_u \cup \mathcal{L}$ is practically feasible when we fix $|L|$ variables.
(A3) The inequality of proposition 10 is an equality. This not necessary for our attack to work, but it is relevant if we want to explain the number of free variables in the system (equation (12)).

To check whether puncturing the GRS code breaks any of these assumptions, we ran a number of experiments. We present a selection of the findings in table VI. The first row in each section corresponds to the non-punctured case. As we go down the lines, the codes are punctured at an increasing number of (randomly chosen) positions. At least for the parameters we test, we find that we can reduce n quite a bit without breaking the assumptions above. As we puncture the GRS code on more and more positions, we reduce both n and k. Once we puncture on so many positions that k becomes the limiting factor for the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$, then the runtime of the Gröbner basis algorithm starts to increase. Before that, the effect of puncturing on the total runtime of our attack is actually positive as many steps of our attack are polynomial in n.

E. How taking a random subcode (defense 4 in the challenge of [9]) affects our attack

Here we consider a McEliece-like scheme based on a randomly chosen subcode of a permuted, narrow sense, primitive BCH-code. Attacking this cryptosystem corresponds to breaking the challenge of [9] with 5 out of 6 defense strategies (scaling, permutation, subfield, wildness and subcode). Note further that randomly chosen subcodes of BCH codes are in general not BCH codes, and neither are they Goppa codes. The idea of masking a code in this way stems from [19].

Like in the previous section, we start by noting that none of our theoretical results break down when adding this additional defense. Here not even the proofs of proposition 20 and 25 are affected. However, like before, we still rely on assumptions A1-A3 of section VII-D. To check whether taking a random subcode breaks any of these assumptions, we ran a number of experiments. We present a selection of our findings in table VII. The first row in each section corresponds to the biggest

TABLE VI
THE INFLUENCE OF PUNCTURING ON THE RUNTIME OF OUR ATTACK

q	n	k	vars	pols	sols	Success	1GB	Total
27	728	631	13	103	2 (2)	10/10	2^{23}	2^{35}
27	228	131	13	103	2 (2)	10/10	2^{23}	2^{32}
27	168	71	13	71	2 (2)	10/10	2^{26}	2^{31}
27	138	41	13	41	2 (2)	10/10	2^{26}	2^{31}
9	728	674	17	138	3 (3)	10/10	2^{29}	2^{34}
9	198	144	17	138	3 (3)	10/10	2^{29}	2^{32}
9	128	74	17	74	3*(3*)	10/10	2^{29}	2^{32}
9	98	44	17	44	3 (3*)	10/10	2^{34}	2^{36}
13	168	150	14	87	2 (16)	10/10	2^{25}	2^{30}
13	108	90	14	87	2 (16)	10/10	2^{25}	2^{29}
13	68	50	14	50	2 (16*)	10/10	2^{26}	2^{29}
13	38	20	14	20	2 (16)	10/10	2^{36}	2^{38}

"vars" is the number of variables in the system once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"pols" is the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$ once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"sols" is the number of solutions that the GB algorithm finds. In parenthesis is the number of solutions counted with multiplicities.
"Success" reports on the number of times we repeated the experiment and for how many of those runs, the our attack succeeded.
1GB is the average number of clock cycles used in one round of the GB algorithm (including both part 2 and 3 of section VI).
"Total" is the total number of clock cycles used when running our entire attack.
* For one or more runs, the computation produced another number of solutions/multiplicities (19 instead of 16 or 4 instead of 3).

TABLE VII
THE INFLUENCE OF TAKING A SUBCODE ON THE RUNTIME OF OUR ATTACK

q	n	k	vars	pols	sols	Success	1GB	Total
27	728	631	13	103	2 (2)	10/10	2^{23}	2^{35}
27	728	431	13	103	2 (2)	10/10	2^{23}	2^{35}
27	728	231	24	231	2 (2)	10/10	2^{33}	2^{37}
9	728	674	17	138	3 (3)	10/10	2^{29}	2^{34}
9	728	574	17	138	3 (3)	10/10	2^{29}	2^{35}
9	728	474	17	138	3 (3)	10/10	2^{29}	2^{35}
9	728	374	17	138	3 (3)	10/10	2^{29}	2^{35}
13	168	150	14	87	2 (16)	10/10	2^{25}	2^{30}
13	168	140	24	140	2 (16*)	10/10	2^{31}	2^{33}
13	168	130	34	130	2 (16)	5/5	2^{43}	2^{45}

"vars" is the number of variables in the system once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"pols" is the number of $\mathbb{F}_{q^m}$-linearly independent polynomials in $\mathcal{P}_u$ once the linear equations of $\mathcal{L}$ have been used to eliminate variables and $|L|$ variables have been fixed.
"sols" is the number of solutions that the GB algorithm finds. In parenthesis is the number of solutions counted with multiplicities.
"Success" reports on the number of times we repeated the experiment and for how many of those runs, the our attack succeeded.
1GB is the average number of clock cycles used in one round of the GB algorithm (including both part 2 and 3 of section VI).
"Total" is the total number of clock cycles used when running our entire attack.
* For one or more runs, the computation produced 19 instead of 16 solutions with multiplicities.

possible subcode (namely the code itself). As we go down the lines, we take random subcodes of decreasing dimension. At least for the parameters we test, we find that for non-prime q we can reduce k quite a bit without breaking the assumptions above. A3 must break down when $s \cdot k < n - |S_p|$, as the number of linearly independent polynomials in $\mathcal{L}$ is upper bounded by $s \cdot k$.[12] For non-prime q, A3 breaks down for any non-trivial subcode as the upper bound here is just k. Hence, the number of variables is equal to $n-k-|L|$, so as k decreases the number of variables increase by the same amount. This causes the runtime of the Gröbner basis algorithm to increase.

VIII. CONCLUSION AND FURTHER WORK

We build on the attack by [8] to develop an algebraic attack on q-ary BCH-McEliece, which demonstrates that in particular high rate codes with non-prime q are unsafe cryptographic choices. We further show how our attack can be expanded to cover randomly chosen subcodes of subfield subcodes of $\mathsf{GRS}_r(\boldsymbol{x}, \boldsymbol{x})^{\perp}$, where $\boldsymbol{x}$ has distinct non-zero entries. That is, we break the BCH-challenge of [9] for high rate codes over non-prime fields.

The attack of [8] focuses on Goppa codes with large multiplicities. BCH codes in a way have the ultimate multiplicities, which motivated us to try to extend the work of [8] to BCH codes. However, through our work, we found that actually the main idea of [8], which is to throw away equations to extend the solution set, does not rely on multiplicities of the Goppa polynomial. In fact, it also applies to Classic McEliece (the NIST submission), which uses Goppa codes with irreducible Goppa polynomials. Breaking these codes is strongly related to finding zeros of [5]

$$\left\{ \sum_{j=1}^{n} g_{ij} Y_j X_j^u \ : \ i \in \{1,..,k\}, u \in \{0,..,r-1\} \right\} \quad (14)$$

I.e. here we do not have n unknowns but $2n$ unknowns, and we cannot get linear polynomials by applying p'th roots (we would have to apply the p'th root to both $\boldsymbol{x}$ and $\boldsymbol{y}$-variables).

However, the main idea [8] still applies. If we for example set $U = \{0, 1, 2\}$, then we can prove that the zero set of

$$\left\{ \sum_{j=1}^{n} g_{ij} Y_j X_j^u \ : \ i \in \{1,..,k\}, u \in U \right\},$$

contains the vector space spanned by $\left\{(\boldsymbol{y}, \boldsymbol{1}), (\boldsymbol{y}, \boldsymbol{x}), .., (\boldsymbol{y}, \boldsymbol{x}^{\lfloor \frac{r-1}{2} \rfloor})\right\}$. I.e. we can fix $\lfloor \frac{r+1}{2} \rfloor$ $\boldsymbol{x}$-variables, when we try to solve the system.

But even with this approach, we are not able to bring down the number of variables sufficiently for the system to be tractable by our polynomial system solver. In fact, for many parameter-sets, the reduced system has more variables than equations, even after we take advantage of the known extra solutions to fix variables. In other words, the reduced system has more solutions than we can explain.

[12] In equation (5) it may look like $\mathcal{L}$ contains $\lfloor \log_p(r) \rfloor \cdot k$ polynomials, but $g_{i,j} \in \mathbb{F}_q$ implies that $\sqrt[q]{\sqrt[p]{g_{i,j}}} = \sqrt[p]{\sqrt[q]{g_{i,j}}}$, so there can be maximally $s \cdot k$ polynomials in $\mathcal{L}$.

Inspired by the work of [8], we would like to understand the additional "wrong" solutions. In [8], it is shown that even though the extra solutions of the system cannot be used directly as a secret key to decrypt messages, they can still aid in finding the secret key. It would be interesting to study whether this idea could also apply to Goppa codes with irreducible Goppa polynomials or BCH codes with $q = 2$.

IX. Acknowledgments

We are indebted to Magali Bardet and Ayoub Otmani for having the idea for this research project and their invaluable support during the whole process. We thank Jean-Pierre Tillich for establishing the collaboration and for the helpful discussions in the early phases of the project.

C.M. was funded by a NWO VENI grant (Project No. VI.Veni.192.159) and F.E. was funded by Dencrypt in collaboration with the Danish Ministry of Defense Acquisition and Logistics Organization.

References

[1] R. J. McEliece, *A Public-Key System Based on Algebraic Coding Theory*. Jet Propulsion Lab, 1978, pp. 114–116, dSN Progress Report 44.

[2] E. Prange, "The use of information sets in decoding cyclic codes," *IRE Transactions on Information Theory*, vol. 8, no. 5, pp. 5–9, 1962. [Online]. Available: http://dx.doi.org/10.1109/TIT.1962.1057777

[3] R. Canto Torres and N. Sendrier, "Analysis of information set decoding for a sub-linear error weight," in *Post-Quantum Cryptography*. Springer, 2016, pp. 144–161.

[4] V. M. Sidelnikov and S. Shestakov, "On the insecurity of cryptosystems based on generalized Reed-Solomon codes," *Discrete Math. Appl.*, vol. 1, no. 4, pp. 439–444, 1992.

[5] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich, "Algebraic cryptanalysis of McEliece variants with compact keys," in *Advances in Cryptology - EUROCRYPT 2010*, ser. LNCS, vol. 6110, 2010, pp. 279–298.

[6] T. P. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani, "Reducing key length of the McEliece cryptosystem," in *Progress in Cryptology - AFRICACRYPT 2009*, ser. LNCS, B. Preneel, Ed., vol. 5580, Gammarth, Tunisia, Jun. 21-25 2009, pp. 77–97.

[7] R. Misoczki and P. Barreto, "Compact McEliece keys from Goppa codes," in *Selected Areas in Cryptography*, Calgary, Canada, Aug. 13-14 2009.

[8] J.-C. Faugère, L. Perret, and F. de Portzamparc, "Algebraic attack against variants of McEliece with Goppa polynomial of a special form," in *Advances in Cryptology - ASIACRYPT 2014*, ser. LNCS, vol. 8873. Kaoshiung, Taiwan, R.O.C.: Springer, Dec. 2014, pp. 21–41.

[9] D. J. Bernstein, "Cryptography for the paranoid," Presentation at Yet Another Conference on Cryptography (YACC), 2012, https://cr.yp.to/talks/2012.09.24/slides.pdf.

[10] A. Couvreur, A. Otmani, and J.-P. Tillich, "Polynomial time attack on wild McEliece over quadratic extensions," in *Advances in Cryptology - EUROCRYPT 2014*, ser. LNCS, P. Q. Nguyen and E. Oswald, Eds., vol. 8441. Springer Berlin Heidelberg, 2014, pp. 17–39.

[11] N. Sendrier, "Finding the permutation between equivalent linear codes: The support splitting algorithm," *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1193–1203, 2000.

[12] W. Bosma, J. Cannon, and C. Playoust, "The Magma algebra system. I. The user language," *J. Symbolic Comput.*, vol. 24, no. 3-4, pp. 235–265, 1997, computational algebra and number theory (London, 1993). [Online]. Available: http://dx.doi.org/10.1006/jsco.1996.0125

[13] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, 5th ed. Amsterdam: North–Holland, 1986.

[14] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "Further results on Goppa codes and their applications to constructing efficient binary codes," *IEEE Trans. Inform. Theory*, vol. 22, pp. 518–526, 1976.

[15] H. Liu, C. Ding, and C. Li, "Dimensions of three types of bch codes over gf(q)," *Discrete Mathematics*, vol. 340, no. 8, pp. 1910–1927, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0012365X1730105X

[16] A. Dür, "The automorphism groups of Reed-Solomon codes," *Journal of Combinatorial Theory, Series A*, vol. 44, pp. 69–82, 1987.

[17] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph (algorithm 457)," *Commun. ACM*, vol. 16, no. 9, pp. 575–576, 1973.

[18] J.-C. Faugère, P. M. Gianni, D. Lazard, and T. Mora, "Efficient computation of zero-dimensional gröbner bases by change of ordering," *J. Symbolic Comput.*, vol. 16, no. 4, pp. 329–344, 1993.

[19] T. P. Berger and P. Loidreau, "How to mask the structure of codes for a cryptographic use," *Des. Codes Cryptogr.*, vol. 35, no. 1, pp. 63–79, 2005.

Appendix

A. Overview of notation which is introduced section V

In section V we introduce a lot of additional notation. We provide the following list as an index in case our readers need to go back to look up a definition.

- q, the size of $\mathbb{F}_q$, which is the alphabet of the BCH code. Introduced in problem 2
- p, the characteristic of the field $\mathbb{F}_q$. Introduced in problem 2
- s, chosen such that $p^s = q$. Introduced in problem 2
- m, the extension degree of the code. I.e. the BCH code is a subfield subcode of a GRS code over $\mathbb{F}_{q^m}$. Introduced in problem 2
- $n = q^m - 1$, the length of the BCH code. Introduced in problem 2
- r, the dimension of the "parent" GRS code. Introduced in problem 2
- k, the dimension of the BCH code. Introduced in problem 2
- G, the generator matrix of the BCH code. Introduced in problem 2
- $\boldsymbol{x}$, the secret key of the BCH McEliece encryption scheme. Introduced in problem 2
- $\mathcal{L}$, the linear equations in the system we will solve. Introduced in equation (5).
- $\mathcal{P}_u$, the non-linear equations in the system we will solve. Introduced in equation (6).
- $\mathcal{C}_q$ the union of cyclotomic cosets $\bigcup_{c\in[\![1,r]\!]} \mathcal{C}_q(c)$, where

$$\mathcal{C}_q(c) \triangleq \left\{cq^e \pmod{q^m-1} \;\middle|\; e \in [\![0, m-1]\!]\right\},$$

  Introduced in section V-C.
- u-sum condition, a condition which will guarantee that a subset $L \subseteq \mathbb{Z}_{q^m-1}$ satisfies

$$\operatorname{span}_{\mathbb{F}_{q^m}}\{\boldsymbol{x}^\ell \mid \ell \in L\} \subseteq \mathcal{Z}_{\mathbb{F}_{q^m}}(\mathcal{P}_u \cup \mathcal{L}).$$

  Introduced in definition 14.
- L, a subset of $\mathbb{Z}_{q^m-1}$ of maximal size which satisfies the u-sum condition. Introduced in section V-C
- ν, the number of subsets of $\mathbb{Z}_{q^m-1}$ of maximal size which satisfy the u-sum condition. Introduced in section V-E.

TABLE VIII

COMPARISON BETWEEN THE LARGEST r FOR WHICH WE ARE ABLE TO RUN THE GRÖBNER BASIS ALGORITHM (WITH THE LIMITATIONS GIVEN IN SECTION VII) AND THE LARGEST r FOR WHICH WE ARE ABLE TO FIND A WELL-FORMED L.

q	m	Max r for Gröbner Basis	Max r with well-formed L
3	5	12	76
3	6	12	$\geqslant 157^*$
5	3	13	48
5	4	16	$\geqslant 213^*$
9	2	32	50
9	3	52	$\geqslant 303^*$
25	2	119	218
27	2	171	$\geqslant 424^*$

*For these parameters, it took longer than five hours to find L for larger r, so we stopped calculations.

B. *Experimental evidence on well-formed L's*

Here we provide experimental evidence to support our claim that finding well-formed L's is not the limiting factor of our attack. More precisely, the results in table VIII indicates that for cryptographically relevant parameters, the maximal r for which we can find a well-formed L is bigger than the maximal r for which we can run the Gröbner basis step of our attack. For larger m this gap seems to increase.

The code used to find these values and a collection of all the L's involved can be found on https://github.com/FrejaElbro/Cryptanalysis-of-BCH-McEliece.