Decoding McEliece with a Hint – Secret Goppa Key Parts Reveal Everything

Elena Kirshanova[1],[2] and Alexander May[3]

[1] Immanuel Kant Baltic Federal University, Kaliningrad, Russia
[2] Technology Innovation Institute, UAE
[3] Ruhr University Bochum, Bochum, Germany
elenakirshanova@gmail.com, alex.may@rub.de

Abstract. We consider the McEliece cryptosystem with a binary Goppa code $C \subset \mathbb{F}_2^n$ specified by an irreducible Goppa polynomial $g(x) \in \mathbb{F}_{2^m}[X]$ and Goppa points $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{2^m}^n$. Since $g(x)$ together with the Goppa points allow for efficient decoding, these parameters form McEliece secret keys. Such a Goppa code C is an $(n - tm)$-dimensional subspace of $\mathbb{F}_2^n$, and therefore C has co-dimension tm. For typical McEliece instantiations we have $tm \approx \frac{n}{4}$.

We show that given more than tm entries of the Goppa point vector $(\alpha_1, \ldots, \alpha_n)$ allows to recover the Goppa polynomial $g(x)$ and the remaining entries in polynomial time. Hence, in case $tm \approx \frac{n}{4}$ roughly a fourth of a McEliece secret key is sufficient to recover the full key efficiently.

Let us give some illustrative numerical examples. For ClassicMcEliece with $(n, t, m) = (3488, 64, 12)$ on input $64 \cdot 12 + 1 = 769$ Goppa points, we recover the remaining $3488 - 769 = 2719$ Goppa points in $\mathbb{F}_{2^{12}}$ and the degree-64 Goppa polynomial $g(x) \in \mathbb{F}_{2^{12}}[x]$ in 60 secs.

For ClassicMcEliece with $(n, t, m) = (8192, 128, 13)$ on input $128 \cdot 13 + 1 = 1665$ Goppa points, we recover the remaining $8192 - 1665 = 6529$ Goppa points in $\mathbb{F}_{2^{13}}$ and the degree-128 Goppa polynomial $g(x) \in \mathbb{F}_{2^{13}}[x]$ in 288 secs.

Our results also extend to the case of erroneous Goppa points, but in this case our algorithms are no longer polynomial time.

Keywords: Classic McEliece · code-based cryptogrpaphy · cryptanalysis · partial key exposure

1 Introduction

Partial Key Exposure Attacks. Some cryptosystems are known to allow for full key recovery from only a fraction of the secret key. As an example, let (N, e) be an RSA public key with corresponding secret key (d, p, q). A famous result of Coppersmith [Cop97] shows that N can be factored efficiently if half of the bits of p are given, thereby revealing the complete secret key. Boneh, Durfee, and Frankel [BDF98] showed that for small e a quarter of the bits of d also suffices to reconstruct the complete secret key. This kind of attacks are often

referred to as *Partial Key Exposure* attacks, and there is a long line of research for RSA [EJMdW05,STK20,MNS21,MNS22].

However, the existence of polynomial time Partial Key Exposure attacks is usually considered a typical RSA vulnerability. It is widely believed that for discrete logarithm problems g^x leakage of a constant fraction of bits of x does not degrade the problem [Gen05,PS98].

The situation seems to be similar for post-quantum cryptosystem. For most schemes, no vulnerabilities are known in the sense that leakage of a constant fraction of the secret key leads to full secret key recovery. In fact, some cryptosystems are believed to be somewhat robust against Partial Key Exposure attacks [DGKS21].

Among the literature on post-quantum Partial Key Exposures is an NTRU attack by Paterson and Villanueva-Polanco [PV17], and attacks of Villanueva-Polanco on BLISS [Vil19a] and LUOV [VP20]. The PhD thesis of Villanueva-Polanco [Vil19b] contains a more systematic study of partial key exposure attacks, including also McEliece. However, none of these attacks is polynomial time for a known constant fraction of the secret key bits. To the best of our knowledge the only exception is a recent result of Esser, May, Verbel, Wen [EMVW22] that recovers BIKE keys from a constant fraction of their secret key.

McEliece Cryptanalysis. Since the McEliece cryptosystem was proposed more than 40 years ago, it faced a lot of significant cryptanalysis efforts [BLP08,EMZ22]. However, most cryptanalysis concentrated on *information set decoding* algorithms [Pra62,BLP11a,MMT11,BJMM12,MO15], basically trying to decoding McEliece instances as if they where instances of random codes with Goppa code parameters, thereby completely ignoring the Goppa code structure hidden in McEliece public keys.

When McEliece is instantiated with other codes, e.g. generalized Reed-Solomon codes, there have been devastating attacks breaking the scheme in polynomial time [SS92]. However, for the originally suggested binary subfield Goppa codes very little structural attacks are known, besides for distinguishers for high-rate Goppa codes [FGO$^+$11] and for very special choices of Goppa code parameters [LS06], both cases being far off typical cryptographic parameter selection.

Our results. We show surprisingly elementary facts about McEliece keys that strongly exploit the Goppa code structure. Our attacks imply a clear warning that one should well protect McEliece secret keys, e.g. against side-channels, since leaking even a small fraction of the key in *any* known positions already allows to efficiently recover the complete secret key.

Let us be a bit more precise what we show in this work. Binary Goppa codes $C \in \mathbb{F}_2^n$ of length n and co-dimension tm are defined via an irreducible *Goppa polynomial* $g(x) \in \mathbb{F}_{2^m}[x]$ of degree-t with $tm < n$, and *Goppa points* $(\alpha_1, \ldots, \alpha_n) \in (\mathbb{F}_{2^m})^n$. In fact, the parity check matrix defining a Goppa code C is a function of a Goppa polynomial and a tuple of Goppa points. Therefore, a McEliece secret key consists of $g(x)$ and $(\alpha_1, \ldots, \alpha_n)$, whereas a McEliece public key H^{pub} is a scrambled form of a parity check matrix.

It is a folklore result that $(\alpha_1, \ldots, \alpha_n)$ allows to efficiently recover $g(x)$. We give a more formal proof of this folklore result in Section 3.1, since it is the starting point for our more involved algorithms.

1. We show in Section 3.2 that any $tm+1$ points α_i allow to efficiently recover $g(x)$.
2. We show in Algorithm 3.3 that any $tm+1$ points α_i together with $g(x)$ allow to recover in polynomial time all the remaining Goppa points, provided that the submatrix formed by the columns of H^{pub} indexed by α_i has full rank.

Both results together imply that with constant probability any $tm+1$ Goppa points suffice to recover the complete McEliece secret key in polynomial time.

We support our claims by implementing our algorithms, and successfully running them on McEliece parameters proposed in [ARBC+20]. Our non-optimized implementations are available at `https://github.com/ElenaKirshanova/leaky_goppa_in_mceliece`.

The results are provided in Table 1. On input of $tm+1$ Goppa points, our algorithm Key-Recovery for all 2000 instances succeeded to recover the degree-t Goppa polynomial $g(x)$ in $\mathbb{F}_{2^m}[x]$ and all remaining $n-(tm+1)$ Goppa points in averaged run times between 1 and 5 mins.

(n, t, m)	$tm+1$	Time
$(3488, 64, 12)$	769	**60 sec**
$(4608, 96, 13)$	1249	**184 sec**
$(6960, 119, 13)$	1548	**258 sec**
$(8192, 128, 13)$	1665	**288 sec**

Table 1: Experimental results for Classic McEliece Key-Recovery (Algorithm 3.4), averaged over 2000 instances. The middle column refers to the number of Goppa points the algorithm receives as input.

Typical McEliece instantiations have $tm+1 \approx \frac{n}{4}$ showing that knowledge of only a quarter of the secret key suffices to efficiently recover the whole. Somewhat interestingly, current McEliece instantiations explicitly choose small co-dimension $tm \ll n$ to guard against information set decoding attacks. Our results in turn benefit from small co-dimension.

Technically, our results heavily use the structure imposed by Goppa codes, and thus can be considered as one of the very few structural non-generic McEliece attacks. At the heart of our algorithms lies a simple routine that constructs — with the help of the McEliece public key— codewords with at most the Hamming weight tm of the co-dimension.

Impact on Classic McEliece. Classic McEliece [ARBC+20, Section 2.5.2] proposes to store the secret key as a so-called *in-place Beneš network* in which neither $g(x)$, nor $(\alpha_1, \ldots, \alpha_n)$ are stored explicitly. It is an open question whether our attacks also apply to this setting.

2 Preliminaries

Notation. We let $\mathbb{F}_2$ denote the binary field and let $\mathbb{F}_{2^m}$ be a finite extension of $\mathbb{F}_2$ of degree $m > 1$. We denote by γ a primitive element of $\mathbb{F}_{2^m}$, i.e., field elements of $\mathbb{F}_{2^m}$ are of the form $\sum_{i=0}^{m-1} a_i \gamma^i$, $a_i \in \mathbb{F}_2$. We further let $n \leq 2^m$ denote some positive integer, and let $L = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{2^m}^n$ be our list of *Goppa points* with distinct points $\alpha_i \neq \alpha_j$ for $i \neq j$. We denote by $g(x)$ our *Goppa polynomial* – an irreducible polynomial of degree t in $\mathbb{F}_{2^m}[x]$.

Let $H = (\mathbf{h}_1 \ldots \mathbf{h}_n) \in \mathbb{F}_2^{tm \times n}$ be a matrix with n columns $\mathbf{h}_i \in \mathbb{F}_2^{tm}$. Let $I \subset \{1, \ldots, n\}$ be an index set. Then we denote by $H[\mathcal{I}]$ the projection of H's to the columns defined by $\mathcal{I} = \{i_1, \ldots, i_\ell\}$, i.e.,

$$H[\mathcal{I}] = (\mathbf{h}_{i_1}, \ldots, \mathbf{h}_{i_\ell}).$$

Analogous for a code $C \subseteq \mathbb{F}_2^n$ and some $\mathcal{I} \subseteq \{1, \ldots, n\}$ we denote by $C[\mathcal{I}]$ the projection to the coordinates in $\mathcal{I} = \{i_1, \ldots, i_\ell\}$, i.e.,

$$C[\mathcal{I}] = \{(c_{i_1}, \ldots, c_{i_\ell}) \in \mathbb{F}_2^\ell \mid (c_1, \ldots, c_n) \in C\}.$$

For a matrix A, we denote its transpose by A^{t}.

Definition 1 (Goppa code). *Let $L = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{2^m}$ be Goppa points and $g(x) \in \mathbb{F}_{2^m}[x]$ be an irreducible, degree-t Goppa polynomial. Then we define a Goppa code*

$$C(L, g) = \left\{ \mathbf{c} \in \mathbb{F}_2^n \ : \ \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \equiv 0 \mod g(x) \right\}. \tag{1}$$

For a codeword $\mathbf{c}$, we define its support as $supp(\mathbf{c}) := \{i \in \{1, \ldots, n\} \mid c_i \neq 0\}$.

Let us consider $\overline{H}_{\mathsf{Goppa}}(L, g) \in \mathbb{F}_{2^m}^{t \times n}$ of the form

$$\overline{H}_{\mathsf{Goppa}}(L, g) = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1 & \alpha_2 & \ldots & \alpha_n \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \ldots & \alpha_n^{t-1} \end{pmatrix} \cdot \begin{pmatrix} g^{-1}(\alpha_1) & 0 & \ldots & 0 \\ 0 & g^{-1}(\alpha_2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & g^{-1}(\alpha_n) \end{pmatrix}.$$

From $\overline{H}_{\mathsf{Goppa}}(L, g) \in \mathbb{F}_{2^m}^{t \times n}$, we construct the parity-check matrix $H_{\mathsf{Goppa}}(L, g) \in \mathbb{F}_2^{mt \times n}$ by applying the bijection $V : \mathbb{F}_{2^m} \to \mathbb{F}_2^m$, that represents $\mathbb{F}_{2^m}$ as an m-dimensional vector space over $\mathbb{F}_2$, i.e, $\sum_{i=0}^{m-1} a_i \gamma^i \mapsto [a_0, \ldots, a_{m-1}]$.

With constant probability $H_{\mathsf{Goppa}}(L,g)$ has rank full rank tm. Throughout this paper we only consider full rank parity check matrices —the standard cryptographic case. For full rank $H_{\mathsf{Goppa}}(L,g)$ the Goppa code $C(L,g) \subset \mathbb{F}_2^n$ is a binary code of co-dimension tm and therefore of dimension $n-tm$.

In Classic McEliece [ARBC+20], the echelon form of H_{Goppa} defines the public key H^{pub}, while (L,g) is the secret key. Knowledge of (L,g) allows for efficient decoding of up to t errors [Gop70,Cho17].

Definition 2 (Syndrome). *For $\mathbf{y} \in \mathbb{F}_2^n$, the* syndrome *of y is defined as*

$$s_{\mathbf{y}} := \sum_{i=1}^{n} \frac{y_i}{x-\alpha_i} = \sum_{i=1}^{n} y_i \prod_{j\neq i}(x-\alpha_j) \bmod g(x). \tag{2}$$

From Definition 1 and Definition 2 we see that $\mathbf{y} \in C(L,g)$ iff we have syndrome $s_{\mathbf{y}} = 0 \mod g(x)$.

The following lemma, see [Gop70,EOS07,BLP11b], shows that a Goppa polynomial $g(x)$ and its square $g^2(x)$ define the same code. We will use this property in our algorithms for recovering the correct Goppa polynomial. We include a proof for completeness.

Lemma 1. *[Gop70] The binary irreducible Goppa code $C(L,g)$ satisfies*

$$C(L,g) = C(L,g^2).$$

Proof. Since $s_{\mathbf{c}} \equiv 0 \mod g(x)^2$ we have $s_{\mathbf{c}} \equiv 0 \bmod g(x)$. The inclusion $C(L,g^2) \subset C(L,g)$ follows.

To show $C(L,g) \subset C(L,g^2)$, for any $\mathbf{c} \in C(L,g)$ define

$$f_{\mathbf{c}}(x) := \prod_{i\in \mathrm{supp}(\mathbf{c})} (x-\alpha_i) \text{ with derivative } f'_{\mathbf{c}}(x) = \sum_{i\in\mathrm{supp}(\mathbf{c})} \prod_{j\neq i}(x-\alpha_j).$$

From Definition 2 it follows that $s_{\mathbf{c}} \equiv f'_{\mathbf{c}}(x)/f_{\mathbf{c}}(x) \bmod g(x)$. Since $g(x)$ is irreducible of degree $t>1$, we have $\gcd(f_{\mathbf{c}}(x), g(x)) = 1$ and hence $f_{\mathbf{c}}(x)$ is invertible modulo $g(x)$. Therefore

$$\mathbf{c} \in C(L,g) \Leftrightarrow s_{\mathbf{c}} \equiv 0 \bmod g(x) \Leftrightarrow f'_{\mathbf{c}}(x) \equiv 0 \bmod g(x).$$

Notice that $f'_{\mathbf{c}}(x) = \sum_{i=1}^{n} i f_i x^{i-1}$ and thus for even i we have $i f_i x^{i-1} = 0 \bmod 2$. Therefore, only x^i-terms with even degree remain. Thus,

$$f'_{\mathbf{c}}(x) = \sum_{i\equiv 0 \bmod 2}^{n} f_i(x^{i/2})^2 = \left(\sum_{i\equiv 0 \bmod 2}^{n} f_i(x^{i/2})\right)^2 \bmod 2.$$

This implies that $f'_{\mathbf{c}}(x)$ is a square. Hence every irreducible divisor of $f'_{\mathbf{c}}(x)$ has to appear with even multiplicity, implying that $g^2(x)$ divides $f'_{\mathbf{c}}(x)$. Therefore, $\mathbf{c} \in C(L,g^2)$. □

Algorithm 2.1 TEST-GOPPA-POLYNOMIAL

Input: $f(x)$, Goppa points $\alpha_i \in \mathbb{F}_{2^m}$ with $i \in \mathcal{I} \subset \{1, \ldots, n\}$,
index set $\mathcal{I}$ with $|\mathcal{I}| := \ell$, generator matrix $G \in \mathbb{F}_2^{j \times \ell}$ of $C(L, g)[\mathcal{I}]$
Output: 1 indicating that $f(x)$ might be $C(L, g)$'s Goppa polynomial, or 0

1: **for** all j rows $\mathbf{g}$ of G **do**
2: Compute $s_{\mathbf{g}}(x) = \sum_{i \in \mathcal{I}} \frac{g_i}{x - \alpha_i} \bmod f(x)$ from Equation (2).
3: **if** $s_{\mathbf{g}}(x) \not\equiv 0 \bmod f(x)$ **then**
4: Return 0
5: Return 1

The following algorithm TEST-GOPPA-POLYNOMIAL (Algorithm 2.1) tests whether a potential Goppa polynomial satisfies Equation (1) for all codewords in the span of some projected Goppa code. We shall make use of this algorithm in the next section.

Throughout the paper, we need that some projected random submatrices have full rank. The following lemma states the probability for this event.

Lemma 2. *Suppose we obtain $\mathbf{u}_1, \ldots, \mathbf{u}_k \in \mathbb{F}_2^\ell$, $\ell > k$ drawn independently at uniform from $\mathbb{F}_2^\ell$. Then $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are linearly independent with probability $\prod_{i=0}^{k-1} 1 - 2^{i-\ell}$.*

Proof. Let E_i, $0 \leq i \leq k$ be the event that the first i vectors $\mathbf{u}_1, \ldots, \mathbf{u}_i$ form an i-dimensional space. Define $\Pr[E_0] := 1$.

Let $p_1 = \Pr[E_1]$ and $p_i = \Pr[E_i \mid E_{i-1}]$ for $2 \leq i \leq k$. Then $p_1 = 1 - \frac{1}{2^\ell}$, since we only have to exclude $\mathbf{u}_1 = 0^\ell \in \mathbb{F}_2^\ell$. Moreover for $1 < i \leq k$, we have

$$p_i = 1 - \frac{2^{i-1}}{2^\ell},$$

since $\mathbf{u}_i$ should not lie in the $(i-1)$-dimensional span $\langle \mathbf{u}_1, \ldots, \mathbf{u}_{i-1} \rangle$. We obtain

$$\begin{aligned} \Pr[E_k] &= \Pr[E_k \mid E_{k-1}] \cdot \Pr[E_{k-1}] = \ldots = \prod_{i=1}^{k} \Pr[E_i \mid E_{i-1}] \\ &= \prod_{i=1}^{k} p_i = \prod_{i=1}^{k} 1 - 2^{i-1-\ell} = \prod_{i=0}^{k-1} 1 - 2^{i-\ell}. \end{aligned}$$

□

3 Some Parts of a Secret Goppa Key Reveal Everything

Our first result states that the knowledge of all Goppa points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{2^m}$ together with the public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$ allows to recover the secret Goppa polynomial $g(x)$. This result seems to be folklore knowledge and is mentioned e.g. in [OS09, Section 4.3] and in [ARBC+20]. However, we are not aware of an

algorithm, let alone a formal proof, showing such a result. We will close this gap for completeness, and also because recovery of the full secret key from *all* Goppa points is the starting point for our more advanced results that recover the secret key from only a *small fraction* of all Goppa points.

3.1 Key Recovery from ALL Goppa points

Idea of Goppa Polynomial Recovery Algorithm for All Points. Recall from Equation (1) that

$$C(L,g) = \left\{ \mathbf{c} \in \mathbb{F}_2^n \ : \ \sum_{i=1}^{n} \frac{c_i}{x-\alpha_i} \equiv 0 \mod g(x) \right\}.$$

Thus, for every codeword $\mathbf{c} = (c_1, \ldots, c_n) \in C(L,g)$ we have

$$\sum_{i=1}^{n} c_i \prod_{1 \le j \le n, j \ne i} (x - \alpha_j) \equiv 0 \bmod g(x). \tag{3}$$

Observe that the left-hand side of Equation (3) is a multiple of the desired Goppa polynomial $g(x)$.

The public key H^{pub} allows to easily compute a generator matrix of the code, from which we can sample random codewords $\mathbf{c} \in C(L,g)$. Our algorithm Basic-Goppa (Algorithm 3.1) now computes from a random $\mathbf{c}$ the left-hand side of Equation (3), factors the resulting polynomial in irreducible factors, and in case there are several degree-t factors, chooses the correct Goppa polynomial using the test from Algorithm 2.1.

Algorithm 3.1 Basic-Goppa

Input: public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$, Goppa points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{2^m}$
Output: Goppa polynomial $g(x)$

1: Compute a generator matrix $G \in \mathbb{F}_2^{n \times (n-tm)}$ of C as the right kernel of H^{pub}.
2: Generate $\mathbf{c} \in C \setminus \{0\}$: for some non-zero $\mathbf{m} \in \mathbb{F}_2^{n-tm}$ set $\mathbf{c} = \mathbf{m}G^{\mathsf{t}} \in \mathbb{F}_2^n$.
3: Compute $f(x) = \sum_{i=1}^{n} c_i \prod_{1 \le j \le n, j \ne i} (x - \alpha_j) \in \mathbb{F}_{2^m}[x]$, see Equation (3).
4: Factor $f(x)$ into irreducible factors over $\mathbb{F}_{2^m}$.
5: **for** all irreducible degree-t factors $\hat{g}(x)$ such that $\hat{g}(x)^2$ divides $f(x)$ **do**
6: **if** Test-Goppa-Polynomial$(\hat{g}(x), \{1, \ldots, n\}, G^{\mathsf{t}}) = 1$ **then** output $\hat{g}(x)$.

Theorem 1. *On input of $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$ and all Goppa points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{2^m}$, algorithm* Basic-Goppa *recovers the Goppa polynomial $g(x)$ in $\widetilde{\mathcal{O}}(n^3)$ operations in $\mathbb{F}_{2^m}$.*

Proof. From the discussion before we know that the polynomial $f(x)$ in line 3 of Basic-Goppa is a multiple of the Goppa polynomial $g(x)$. By Lemma 1 we

know that $C(L,g) = C(L,g^2)$, and thus not only $g(x)$, but also $g^2(x)$ divides $f(x)$.

Among all potential irreducible candidates $\hat{g}(x)$ of degree t whose square divide $f(x)$, we look for a Goppa polynomial that generates our code $C(L,g)$. To this end we use Test-Goppa-Polynomial that checks whether all codewords generated by the basis G^{t} are in $C(L,\hat{g})$ from Equation (1), which implies $C(L,\hat{g}) = C(L,g)$. This in turn means that $\hat{g}(x)$ defines the desired Goppa code.

This completes correctness of Basic-Goppa, it remains to show the run time. Using Gaussian elimination, the generator matrix G can be computed in time $\mathcal{O}(n^3)$.

The polynomial $f(x) \in \mathbb{F}_{2^m}[x]$ is of degree $n-1$. Thus, $f(x)$ can be factored in time $\widetilde{\mathcal{O}}(n^3 + n^2 \log |\mathbb{F}|) = \widetilde{\mathcal{O}}(n^3)$ operations in $\mathbb{F}_{2^m}$ [Sho05, Section 20]. □

3.2 Goppa Polynomial Recovery from only $tm+1$ Goppa points

In this section, we show that only $tm+1$ Goppa points together with the public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$ suffice to recover a list of candidate polynomials that contain the Goppa polynomial $g(x)$. Since $tm+1 \ll n$ this improves significantly over the results of the previous Section 3.1. For typical McEliece parameters we have $tm+1 \approx \frac{n}{4}$, i.e. only a quarter of the Goppa points suffice to recover the Goppa polynomial. In the subsequent Section 3.3, we further show how to efficiently compute the remaining Goppa points, thereby identifying the correct $g(x)$ and recovering the complete McEliece secret key.

Idea of Goppa Polynomial Recovery Algorithm from $tm+1$ Points. Recall from Definition 1 of a Goppa code and Equation (1) that all Goppa codewords $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{F}_2^n$ satisfy

$$\sum_{i=1}^{n} \frac{c_i}{x - a_i} = \sum_{i \in \mathrm{supp}(\mathbf{c})}^{n} \frac{c_i}{x - a_i} \equiv 0 \bmod g(x),$$

where $\mathrm{supp}(\mathbf{c}) = \{i \in \{1,\ldots,n\} \mid c_i \neq 0\}$ denotes the index set of non-zero coordinates in $\mathbf{c}$, called $\mathbf{c}$'s support. We conclude that

$$\sum_{i \in \mathrm{supp}(\mathbf{c})} c_i \prod_{j \in \mathrm{supp}(\mathbf{c}) \setminus \{i\}} (x - \alpha_j) \equiv 0 \bmod g(x). \tag{4}$$

Assume now that we know the Goppa points α_j within an index set $j \in \mathcal{I} \subseteq \{1,\ldots,n\}$. If we succeed to construct a codeword $\mathbf{c}$ with $\mathrm{supp}(\mathbf{c}) \subseteq \mathcal{I}$, then we can efficiently compute the left-hand side of Equation (4).

Our main observation is that for any index set $\mathcal{I}$ with $|\mathcal{I}| > tm$ we can easily construct a non-zero codeword $\mathbf{c}$ with $\mathrm{supp}(\mathbf{c}) \in \mathcal{I}$. In a nutshell, we project the Goppa code $C(L,g)$ to the coordinates in $\mathcal{I}$. The details are provided in Advanced-Goppa (Algorithm 3.2).

Algorithm 3.2 ADVANCED-GOPPA

Input: public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$, index set $\mathcal{I} \subset \{1, \ldots, n\}$ with $\ell := |\mathcal{I}| > tm$, Goppa points $\alpha_i \in \mathbb{F}_{2^m}$ with $i \in \mathcal{I}$

Output: list $\mathcal{L}$ of all potential Goppa polynomials $\hat{g}(x)$ with $g(x) \in \mathcal{L}$

1: Let $H^{\mathsf{pub}}[\mathcal{I}] \in \mathbb{F}_2^{tm \times \ell}$ be the projection of $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$ to the ℓ columns from $\mathcal{I}$.
2: Compute $G[\mathcal{I}] \in \mathbb{F}_2^{\ell \times (\ell - \mathrm{rank}(\bar{H}))}$ as the right kernel of $H^{\mathsf{pub}}[\mathcal{I}]$.
3: For some non-zero $\mathbf{m} \in \mathbb{F}_2^{\ell - \mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}])}$ set $\bar{\mathbf{c}} = \mathbf{m}(G[\mathcal{I}])^{\mathsf{t}} \in \mathbb{F}_2^{\ell}$.
4: Construct $\mathbf{c}$ by appending to $\bar{\mathbf{c}}$ zeros in all positions $\{1, \ldots, n\} \setminus \mathcal{I}$.
5: Compute $f(x) = \sum_{i \in \mathrm{supp}(\mathbf{c})} c_i \prod_{j \in \mathrm{supp}(\mathbf{c}) \setminus \{i\}} (x - \alpha_j) \in \mathbb{F}_{2^m}[x]$, see Equation (4).
6: Factor $f(x)$ into irreducible factors over $\mathbb{F}_{2^m}$. Set $\mathcal{L} = \emptyset$.
7: **for** all irreducible degree-t factors $\hat{g}(x)$ such that $\hat{g}(x)^2$ divides $f(x)$ **do**
8: **if** TEST-GOPPA-POLYNOMIAL$(\hat{g}(x), \mathcal{I}, \bar{G}[I]^{\mathsf{t}}) = 1$ **then** $\mathcal{L} := \mathcal{L} \cup \hat{g}(x)$.

Theorem 2 (Goppa polynomial). *On input of $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$, an index set $\mathcal{I} \subset \{1, \ldots, n\}$ with $\ell := |\mathcal{I}| > tm$, and Goppa points $\alpha_i \in \mathbb{F}_{2^m}, i \in \mathcal{I}$, algorithm* ADVANCED-GOPPA *computes a list $\mathcal{L}$ with the Goppa polynomial $g(x) \in \mathcal{L}$ in $\widetilde{\mathcal{O}}(n^3)$ operations in $\mathbb{F}_{2^m}$.*

Proof. The correctness and run time proof for ADVANCED-GOPPA follows mostly the reasoning in the proof of Theorem 1. In addition, we have to show that ADVANCED-GOPPA constructs a non-zero codeword $\mathbf{c} \in C(L, g)$ with $\mathrm{supp}(\mathbf{c}') \in \mathcal{I}$.

Since $\ell > tm$ we have $\ell - \mathrm{rank}(\bar{H}) \geq \ell - tm > 0$. Thus, there exists some non-zero $\mathbf{m} \in \mathbb{F}_2^{\ell - \mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}])}$, and in turn some non-zero $\bar{\mathbf{c}} \in \mathbb{F}_2^{\ell}$. Since $\mathbf{c}$ is constructed from $\bar{\mathbf{c}}$ by appending zeros in positions outside $\mathcal{I}$, we have $\mathrm{supp}(\mathbf{c}) \subseteq \mathcal{I}$. Since $\bar{\mathbf{c}}$ is from the right kernel of $H^{\mathsf{pub}}[\mathcal{I}]$ we have $H^{\mathsf{pub}}[\mathcal{I}]\bar{\mathbf{c}} = 0^{tm}$, and since we append only zeros, also $H^{\mathsf{pub}}\mathbf{c} = 0^{tm}$. This shows that $\mathbf{c} \in C(L, g)$ is indeed a codeword with $\mathrm{supp}(\mathbf{c}) \in \mathcal{I}$. □

Remark 1 (less Goppa points). In the proof of Theorem 2 we construct a polynomial $\bar{\mathbf{c}}$ with Hamming weight at most ℓ, and expected Hamming weight only $\frac{\ell}{2}$. Assume that we are given an oracle $\mathcal{O}(i)$ that returns α_i. Then we could ask $\mathcal{O}(\cdot)$ on $\bar{\mathbf{c}}$'s support, i.e., on expectation only $\frac{tm+1}{2}$ Goppa points would be sufficient.

Experiments. In Table 2 we show the results of running the ADVANCED-GOPPA algorithm on Classic McEliece parameter sets, implemented in SageMath (version 9.4). For each parameter set we generated 20 different McEliece public keys, and for each key, we ran ADVANCED-GOPPA on 100 different index sets $\mathcal{I}$.

Observe from Table 2 (columns 3 and 5) that ADVANCED-GOPPA already for the minimal $\ell = tm + 1$ usually only outputs the desired $g(x)$. When we increased to $\ell = tm + 2$ we never found a polynomial $\hat{g}(x) \neq g(x)$.

Larger ℓ helps TEST-GOPPA-POLYNOMIAL (Algorithm 2.1) to exclude faulty $\hat{g}$. For $\ell = tm + 1$ we have $\mathrm{rank}((G[\mathcal{I}])^{\mathsf{t}}) = 1$ with high probability, and hence there is only a single non-zero $\mathbf{c}$ in the code generated by $(G[\mathcal{I}])^{\mathsf{t}}$. In this case

(n, t, m)	$\ell = tm + 1$	$\lvert\mathcal{L}\rvert = 1$	$\ell = tm + 2$	$\lvert\mathcal{L}\rvert = 1$	Av. time
$(3488, 64, 12)$	769	97%	770	100%	18 sec
$(4608, 96, 13)$	1249	99%	1250	100%	54 sec
$(6960, 119, 13)$	1548	99%	1549	100%	91 sec
$(8192, 128, 13)$	1665	99%	1666	100%	105 sec

Table 2: Experimental results for ADVANCED-GOPPA(Algorithm 3.2).

TEST-GOPPA-POLYNOMIAL cannot exclude any false positive $\hat{g}$. However, for $\ell = tm+2$ we might have $\text{rank}((\bar{G})) = 2$, which lets TEST-GOPPA-POLYNOMIAL exclude faulty $\hat{g}$'s.

Our run time (last column) is averaged over all 2000 runs. Our single-threaded experiments were conducted on Intel Xeon(R) E-2146G CPU 3.50GHz×12, 64GiB, Ubuntu 20.04. We see that our non-optimized implementation finds the Goppa polynomial $g(x)$ for all parameter sets in a matter of seconds.

3.3 Reconstruction of the Remaining Goppa Points

In Section 3.2, we showed that $\ell > tm$ Goppa points are sufficient to efficiently reconstruct the Goppa polynomial $g(x)$ (or a list $\mathcal{L}$ containing $g(x)$). In this section, we show that $g(x)$ together with $\ell > tm$ Goppa points are sufficient to recover all n Goppa points from H^{pub}. This in turn implies that $\ell > tm$ Goppa points are sufficient to efficiently recover the complete McEliece secret key.

Idea of Goppa Points Recovery Algorithm from $g(x)$ and $tm+1$ Points. Assume that we know α_i for an index set $\mathcal{I}$ of size $\ell := |\mathcal{I}| > tm$. Our goal is to compute α_r for some $r \notin \mathcal{I}$.

Our idea is to construct a codeword $\mathbf{c} \in C(L, g)$ with $\text{supp}(\mathbf{c}) \setminus \mathcal{I} = \{r\}$, i.e., $\mathbf{c}$ has all but a single 1-coordinate $c_r = 1$ inside $\mathcal{I}$.

From the definition of a Goppa code in Equation (1) we obtain

$$\sum_{i \in \mathcal{I}} \frac{c_i}{x - \alpha_i} \equiv \frac{1}{x - \alpha_r} \bmod g(x). \tag{5}$$

Knowing the left-hand side and $g(x)$ enables us to compute α_r.

The high-level idea of constructing $\mathbf{c} \in C(L, g)$ with $\text{supp}(\mathbf{c}) \setminus \mathcal{I} = \{r\}$ is to express the r-th column of H^{pub} as an $\mathbb{F}_2$-sum of the ℓ columns in $H^{\mathsf{pub}}[\mathcal{I}]$. This amounts to solving a system of linear equations. The details are given in GOPPA POINTS (Algorithm 3.3) and the analysis in Theorem 3

Theorem 3 (Goppa points). *On input $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$, the Goppa polynomial $g(x)$, an index set $\mathcal{I} \subset \{1, \ldots, n\}$ with $\ell := |\mathcal{I}| > tm$ such that $\text{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$, and Goppa points $\alpha_i \in \mathbb{F}_{2^m}, i \in \mathcal{I}$, algorithm GOPPA-POINTS outputs in time $\mathcal{O}\left(n^4\right)$ all Goppa points $\alpha_1, \ldots, \alpha_n$.*

Algorithm 3.3 GOPPA-POINTS

Input: public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$, Goppa polynomial $g(x)$,
index set $\mathcal{I}$ with $\ell := |\mathcal{I}| > tm$ and $\operatorname{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$,
Goppa points $\alpha_i \in \mathbb{F}_{2^m}$ with $i \in \mathcal{I} \subset \{1, \ldots, n\}$

Output: all Goppa points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{2^m}$ or FAIL

1: **while** $\mathcal{I} \neq \{1, \ldots, n\}$ **do**
2: Pick $r \in \{1, \ldots, n\} \setminus \mathcal{I}$.
3: Find $\mathbf{c} \in \mathbb{F}_2^{|\mathcal{I}|}$ that solves the linear equation system $H^{\mathsf{pub}}[J]\mathbf{c} = H^{\mathsf{pub}}[\{r\}]$.
4: Compute $f(x) = \left(\sum_{i \in \mathcal{I}} \frac{c_i}{x - \alpha_i}\right)^{-1} \bmod g(x)$ using Equation (5).
5: **if** $f(x)$ is of the form $x - \alpha_r$ **then** output α_r,
6: **else** output FAIL.
7: Set $\mathcal{I} \leftarrow \mathcal{I} \cup \{r\}$.

Proof. Let us first address the correctness of GOPPA-POINTS. Since we require $\operatorname{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$ the linear equation equation system $H^{\mathsf{pub}}[J]\mathbf{c} = H^{\mathsf{pub}}[\{r\}]$ in line 3 is always solvable.

Thus, GOPPA-POINTS constructs a solution $\mathbf{c} \in \mathbb{F}_2^{|\mathcal{I}|}$. Define $\mathbf{c}'$ by appending to $\mathbf{c}$ a 1-coordinate in the r position, and 0-coordinates in all positions from $\{1, \ldots, n\} \setminus \{\mathcal{I} \cup \{r\}\}$. By construction $H^{\mathsf{pub}}\mathbf{c}' = 0^{tm}$, and therefore $\mathbf{c}' \in C(L, g)$ with $\operatorname{supp}(\mathbf{c}') \in \mathcal{I} \cup \{r\}$. This allows us to solve for α_r using Equation (5) in line 4.

By Equation (5) we always have $f(x) = x - \alpha_r$, and thus we output another Goppa point in line 5. The purpose of the else-Statement in line 6 is to identify incorrect inputs, either incorrect $\hat{g}(x)$ or faulty Goppa points $\tilde{\alpha}_i$. We come back to this issue in Sections 3.4 and 4.

GOPPA-POINTS's runtime is dominated by running Gaussian elimination in line 3 for computing $\mathbf{c}$. Gaussian elimination runs in $\mathcal{O}(n^3)$ steps in each of the $n - \ell$ iterations, resulting in total run time $\mathcal{O}(n^4)$. □

(n, t, m)	$\ell = tm + 1$	time
$(3488, 64, 12)$	769	42 sec
$(4608, 96, 13)$	1249	130 sec
$(6960, 119, 13)$	1548	167 sec
$(8192, 128, 13)$	1665	183 sec

Table 3: Experimental results for GOPPA-POINTS (Algorithm 3.3).

Experiments. Table 3 shows how Algorithm 3.3 performs in practice.

Analogous to the experiments in Section 3.2, we generated 20 different McEliece public key, and for each of them we ran 100 experiments with different index sets $\mathcal{I}$. We averaged the run time over all 2000 experiments.

Again, we see that recovering all remaining $n - \ell$ Goppa points is with our (non-optimized) implementation realized in a matter of seconds.

Remark 2 (Non-full rank). The condition $\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$ in Goppa-Points is required to solve the equation system in line 3. However, we would like to stress that Goppa-Points allows to successfully recover some Goppa points even in the case $\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) < tm$. In this case, the equation system in line 3 is solvable by the famous Rouché-Capelli theorem iff

$$\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = \mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I} \cup \{r\}]). \tag{6}$$

Thus, we can modify Line 2 such that we choose only r satisfying Equation (6). All corresponding α_r can still be computed by Goppa-Points. E.g. for $\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm-1$ we expect that Goppa-Points still computes $\frac{n-\ell}{2}$, i.e., half of all remaining Goppa points.

Remark 3 (Probability of full rank). Assume that we obtain an index set $\mathcal{I}$ of size $\ell \geq tm + 1$ chosen uniformly at random from $\{1, \ldots, n\}$. Under the assumption that H^{pub} behaves like a random matrix over $\mathbb{F}_2$, Lemma 2 shows that we are in the full-rank case $\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$ with probability at least

$$\prod_{i=0}^{tm-1} 1 - 2^{i-\ell} \geq \prod_{i=0}^{tm-1} 1 - 2^{i-tm-1} = \prod_{i=2}^{tm+1} 1 - 2^{-i} \geq \lim_{n\to\infty} \left(\prod_{i=2}^{n} 1 - 2^{-i} \right) \approx 0.58.$$

3.4 Full Key Recovery from $tm + 1$ Goppa Points

Combining Theorem 2 and Theorem 3, we obtain a full key recovery algorithm from at least $tm + 1$ Goppa points. The algoritm Key-Recovery that successively runs Advanced-Goppa and Goppa-Points is described in Algorithm 3.4.

Theorem 4 (Key Recovery). *On input of $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm\times n}$, an index set $\mathcal{I} \subset \{1, \ldots, n\}$ with $\ell := |\mathcal{I}| > tm$ such that $\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$, and Goppa points $\alpha_i \in \mathbb{F}_{2^m}, i \in \mathcal{I}$, algorithm Key-Recovery outputs in time $\mathcal{O}\left(\frac{n^5}{t}\right) = \mathcal{O}(n^5)$ the Goppa polynomial $g(x)$ and all Goppa points $\alpha_1, \ldots, \alpha_n$.*

Proof. Let us first show correctness. Key-Recovery calls Advanced-Goppa Algorithm 3.2 and recovers a list $\mathcal{L}$ that contains the correct Goppa polynomial $g(x)$. Then for every candidate $\hat{g}(x)$ in line 3 Key-Recovery tries to recover all remaining Goppa points. Usually, Goppa-Points immediately fails on incorrect $\hat{g}(x)$.

Notice that the correct polynomial $g(x)$ is always contained in $\mathcal{L}$, thus by Theorems 2 and 3 the loop in line 3 recovers at least one key candidate k_i. Thus,

Algorithm 3.4 KEY-RECOVERY

Input: public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$,
index set $\mathcal{I}$ with $\ell := |\mathcal{I}| > tm$ and $\operatorname{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$,
Goppa points $\alpha_i \in \mathbb{F}_{2^m}$ with $i \in \mathcal{I} \subset \{1, \ldots, n\}$

Output: Goppa polynomial $g(x)$, all Goppa points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{2^m}$

1: Run $\mathcal{L} \leftarrow$ADVANCED-GOPPA$(H^{\mathsf{pub}}, \mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}})$
2: $i := 0$
3: **for** every $\hat{g}(x) \in \mathcal{L}$ **do**
4: **if** GOPPA-POINTS$(H^{\mathsf{pub}}, \hat{g}(x), \mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}}) \neq$ FAIL **then**
5: $i \leftarrow i + 1$
6: $k_i \leftarrow (\hat{g}(x), \alpha_1, \ldots, \alpha_n)$
7: **if** $i = 1$ **then** output $k_1 = (\hat{g}(x), \alpha_1, \ldots, \alpha_n)$.
8: **else** check $k_1, \ldots, k_i$ via a transformation to public key and comparison with H^{pub}.

we either output the correct key in line 7, or find among more than one candidate k_i the correct key in line 8. Our check in line 8 uses McEliece's deterministic transformation from secret to public key, and thereby assures that we output the correct key.

Let us consider run time. ADVANCED-GOPPA factors in Equation (4) polynomials of degree at most $\ell \leq n$. A candidate polynomial $\hat{g}(x)$ from $\mathcal{L}$ must have degree t, and appear as a square in the factorization. Thus, $|\mathcal{L}| = \mathcal{O}(n/t)$. KEY-RECOVERY's run time is dominated by the loop in line 3, running GOPPA-POINTS in time $\mathcal{O}(n^4)$ for $|\mathcal{L}|$ iterations. The run time follows. □

Experiments. Run times of our KEY-RECOVERY (Algorithm 3.4) experiments are provided in Table 1. Since almost always $|\mathcal{L}| = 1$, i.e., ADVANCED-GOPPA finds only the correct Goppa polynomial, KEY-RECOVERY's runtime is mainly the sum of the runtimes of ADVANCED-GOPPA and of GOPPA-POINTS (compare with Tables 2 and 3).

We would like to stress that we never found an example of more than a single key k_1, thus we never had to apply the key check in line 8 of KEY-RECOVERY.

4 Correcting Faulty Goppa Points

Error Model. In practice, one might be able (e.g. via some side-channel) to obtain erroneous Goppa points. Assume the following simple error model. An attacker obtains erroneous Goppa points $\tilde{\alpha}_1, \ldots \tilde{\alpha}_n$, where each $\tilde{\alpha}_i \in \mathbb{F}_{2^m}$ is correct with probability $1 - p$, and faulty with probability p for some constant $0 < p < 1$. In case $\tilde{\alpha}_i$ is faulty, we assume that $\tilde{\alpha}_i$ is uniformly distributed (among all incorrect values). More precisely, for all $1 \leq i \leq n$ we have

$$\Pr[\tilde{\alpha}_i = \alpha_i] = 1 - p \text{ and } \Pr[\tilde{\alpha}_i = y \mid \tilde{\alpha}_i \neq \alpha_i] = \frac{1}{2^m - 1} \text{ for all } y \in \mathbb{F}_{2^m} \setminus \{\alpha_i\}.$$

We now show that our algorithm Key-Recovery nicely extends to the error scenario, but we have to sacrifice polynomial run time.

Idea of Faulty Goppa Point Correction. Recall that Key-Recovery from Section 3.4 requires only $tm + 1 \ll n$ correct Goppa points to recover the secret key. The basic idea of our correction algorithm is to guess a size-$(tm+1)$ subset of $\tilde{\alpha}_1, \ldots \tilde{\alpha}_n$ that contains only correct Goppa points. Thus, our algorithm is reminiscent of Prange's *information set decoding* [Pra62].

To this end we have to be cautious, since the correctness proof of Key-Recovery only guarantees that Key-Recovery outputs the correct key when run on error-free α_i's. Therefore, we modify Key-Recovery to Faulty-Key-Recovery that also handles erroneous inputs.

Faulty-Key-Recovery (see Algorithm 4.1) provides the following additional checks. Line 2 aborts, when Advanced-Goppa does not find a candidate for the Goppa polynomial. This usually happens for faulty α_i, since Equation (4) only holds for correct Goppa points. Moreover, line 5 aborts, when Goppa-Points fails, because Equation (5) does not hold for incorrect Goppa points. We build in additional checks in lines 8 and 9 in order to prove correctness of our Goppa point correction algorithm. However, we experimentally observe that lines 2 and 5 seem to capture already all faults in practice.

Algorithm 4.1 Fault-Key-Recovery

Input: public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$,
index set $\mathcal{I}$ with $\ell := |\mathcal{I}| > tm$ and $\mathrm{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$,
Goppa points $\alpha_i \in \mathbb{F}_{2^m}$ with $i \in \mathcal{I} \subset \{1, \ldots, n\}$

Output: Goppa polynomial $g(x)$, all Goppa points $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{2^m}$ or FAIL

1: Run $\mathcal{L} \leftarrow$Advanced-Goppa$(H^{\mathsf{pub}}, \mathcal{I}, \{\alpha_i\}_{i\in\mathcal{I}})$
2: **if** $|\mathcal{L}| = 0$ **then** output FAIL.
3: $i := 0$
4: **for** every $\hat{g}(x) \in \mathcal{L}$ **do**
5: **if** Goppa-Points$(H^{\mathsf{pub}}, \hat{g}(x), \mathcal{I}, \{\alpha_i\}_{i\in\mathcal{I}}) \neq$ FAIL **then**
6: $i \leftarrow i + 1$
7: $k_i \leftarrow (\hat{g}(x), \alpha_1, \ldots, \alpha_n)$
8: **if** $i = 0$ **then** output FAIL.
9: **else** check $k_1, \ldots, k_i$ via a transformation to public key and comparison with H^{pub}. If none of $k_1, \ldots, k_i$ is the correct key, output FAIL.

Our algorithm Faulty-Goppa (Algorithm 4.2) now calls Fault-Key-Recovery to check for error-freeness of the chosen size-$(tm + 1)$ subset of Goppa points, and recovers the key for an error-free subset.

Theorem 5. *On input $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm\times n}$, and erroneous Goppa points $\tilde{\alpha}_1, \ldots \tilde{\alpha}_n$, where pn α_i are faulty and $n(1-p) \geq tm+1$,* Faulty-Goppa *outputs the Goppa*

Algorithm 4.2 FAULTY-GOPPA

Input: public key $H^{\mathsf{pub}} \in \mathbb{F}_2^{tm \times n}$,
erroneous Goppa points $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_n \in \mathbb{F}_{2^m}$

Output: Goppa polynomial $g(x)$, Goppa points $\alpha_1, \ldots, \alpha_n$

1: **repeat**
2: Choose uniformly $\mathcal{I} \subset \{1, \ldots, n\}$, $|\mathcal{I}| = tm + 1$ s.t. $\operatorname{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$.
3: **until** FAULTY-KEY-RECOVERY$(H^{\mathsf{pub}}, \mathcal{I}, \{\alpha_i\}_{i \in \mathcal{I}}) \neq$ FAIL.

polynomial $g(x)$ and the Goppa points $\alpha_1, \ldots, \alpha_n$ in expected time

$$T = \mathcal{O}\left(n^5 \cdot \frac{\binom{n}{tm+1}}{\binom{n(1-p)}{tm+1}} \right).$$

The run time can only be proven under the assumption that H^{pub} behaves like a random matrix over $\mathbb{F}_2$.

Proof. Let us first show correctness. Theorem 4 ensures that KEY-RECOVERY and therefore also FAULTY-KEY-RECOVERY outputs the correct key when run on an error-free Goppa point subset $\{\alpha_i\}_{i \in \mathcal{I}}$, since the additional checks in lines 2, 5, 8 and 9 never apply. Moreover, these checks guarantee that FAULTY-KEY-RECOVERY either outputs FAIL, or the correct secret key.

Let us now prove FAULTY-GOPPA's expected run time. The input contains $n(1-p)$ correct Goppa points, and the probability that we choose an index set $\mathcal{I}$ of size $tm + 1$ with only error-free Goppa points is

$$p_0 := \Pr[\{\alpha_i\}_{i \in \mathcal{I}} \text{ error-free}] = \frac{\binom{n(1-p)}{tm+1}}{\binom{n}{tm+1}}.$$

By Lemma 2, we have $\operatorname{rank}(H^{\mathsf{pub}}[\mathcal{I}]) = tm$ with probability at least $\frac{1}{2}$, if H^{pub} behaves like a random matrix. Thus, we have to run an expected number of $\mathcal{O}(p_0^{-1})$ iterations, until we discover an error-free Goppa point subset. Since in each iteration we run FAULTY-KEY-RECOVERY, and FAULTY-KEY-RECOVERY has the same asymptotic run time $\mathcal{O}(n^5)$ as KEY-RECOVERY, the run time follows. □

Run Time Discussion. Assume that $tm + 1 = cn$ for some constant c, typically $c = \frac{1}{4}$ for McEliece instantiations. Using Stirling's formula and the binary entropy function $\mathcal{H}(\cdot)$, one can express FAULTY-GOPPA's run time T in Theorem 5 (neglecting polynomial factors) as the *exponential run time*

$$2^{\left(\mathcal{H}(c) - \mathcal{H}\left(\frac{c}{1-p}\right)(1-p)\right)n}.$$

Experiments. In our experiments, we wanted to understand which checks of Faulty-Key-Recovery lead to FAIL. To this end, for each Classic McEliece parameter set we started with error-free Goppa points $\alpha_1, \ldots, \alpha_n$, chose a size-$(tm+1)$ subset thereof, and injected a single faulty $\tilde{\alpha}_i$ in this subset. We consider this the hardest case for letting Faulty-Key-Recovery fail. We ran Faulty-Key-Recovery on 100 of these injected faulty instances. We then repeated the experiments with two injected faults.

Our results are presented in Table 4. We provide the percentages of FAIL events caused by either line 2 or line 5 of Faulty-Key-Recovery. All faulty keys were detected by these two checks, the additional checks of lines 8 and 9 were never applied.

From Table 4 we observe that if we run Faulty-Key-Recovery with a single injected fault, it still recovers the correct Goppa polynomial $g(x)$ with probability roughly $1/2$. This happens, since a codeword $\mathbf{c}$ constructed inside Advanced-Goppa may have a 0-coordinate on the position of the faulty $\tilde{\alpha}_i$. This probability drops to at most 29%, when the input set has two faulty Goppa points, since now Advanced-Goppa needs a $\mathbf{c}$ with 0-coordinates on these two faulty positions. However, our subroutine Goppa-Points inside Faulty-Key-Recovery eventually detected all faulty inputs via Eq. (5) in our experiments.

(n, t, m)	1 fault in $tm+1$ points		2 faults in $tm+1$ points	
	line 2	line 5	line 2	line 5
$(3488, 64, 12)$	46%	54%	71%	29%
$(4608, 96, 13)$	50%	50%	80%	20%
$(6960, 119, 13)$	51%	49%	83%	17%
$(8192, 128, 13)$	52%	48%	84%	16%

Table 4: Occurrences of two FAIL events in Faulty-Key-Recovery, when either 1 (left part), or 2 (right part) faulty α's are injected in the index set.

Acknowledgments. Elena Kirshanova is supported by the Young Russian Mathematics scholarship and by the Russian Science Foundation grant N 22-41-04411, `https://rscf.ru/project/22-41-04411/`. Alexander May is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – grants 465120249; 390781972.

References

ARBC+20. Martin Albrecht R., Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Mis-

oczki, Ruben Niederhagen, Kenneth G. Paterson, Persichetti Edoardo, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece: conservative code-based cryptography. `https://classic.mceliece.org/nist/mceliece-20201010.pdf`, 2020. 3, 4, 5, 6

BDF98. Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 25–34. Springer, 1998. 1

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20: How 1 + 1 = 0 improves information set decoding. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012. 2

BLP08. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the McEliece cryptosystem. In *PQCrypto*, volume 5299 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2008. 2

BLP11a. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. 2

BLP11b. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Wild McEliece. In *Selected Areas in Cryptography*, pages 143–158, 2011. 5

Cho17. Tung Chou. Mcbits revisited. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 213–231, 2017. 5

Cop97. Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptol.*, 10(4):233–260, 1997. 1

DGKS21. Dana Dachman-Soled, Huijing Gong, Mukul Kulkarni, and Aria Shahverdi. (In)security of ring-lwe under partial key exposure. *J. Math. Cryptol.*, 15(1):72–86, 2021. 2

EJMdW05. Matthias Ernst, Ellen Jochemsz, Alexander May, and Benne de Weger. Partial key exposure attacks on RSA up to full size exponents. In *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 371–386. Springer, 2005. 2

EMVW22. Andre Esser, Alexander May, Javier Verbel, and Weiqiang Wen. Partial key exposure attacks on BIKE, Rainbow and NTRU. In *CRYPTO*, Lecture Notes in Computer Science. Springer, 2022. 2

EMZ22. Andre Esser, Alexander May, and Floyd Zweydinger. McEliece needs a break–solving McEliece-1284 and quasi-cyclic-2918 with modern ISD. In *EUROCRYPT*, Lecture Notes in Computer Science. Springer, 2022. 2

EOS07. D. Engelbert, R. Overbeck, and A. Schmidt. A summary of McEliece-type cryptosystems and their security. *Journal of Mathematical Cryptology*, 1(2):151–199, 2007. 5

FGO$^+$11. Jean-Charles Faugère, Valérie Gauthier-Umaña, Ayoub Otmani, Ludovic Perret, and Jean-Pierre Tillich. A distinguisher for high rate McEliece cryptosystems. In *ITW*, pages 282–286. IEEE, 2011. 2

Gen05. Rosario Gennaro. An improved pseudo-random generator based on the discrete logarithm problem. *J. Cryptol.*, 18(2):91–110, 2005. 2

Gop70. Valerii Denisovich Goppa. A new class of linear correcting codes. pages 207–212, 1970. 5

LS06. P. Loidreau and N. Sendrier. Weak keys in the McEliece public-key cryptosystem. *IEEE Trans. Inf. Theor.*, 47(3):1207–1211, sep 2006. 2

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124. Springer, 2011. 2

MNS21. Alexander May, Julian Nowakowski, and Santanu Sarkar. Partial key exposure attack on short secret exponent CRT-RSA. In *ASIACRYPT (1)*, volume 13090 of *Lecture Notes in Computer Science*, pages 99–129. Springer, 2021. 2

MNS22. Alexander May, Julian Nowakowski, and Santanu Sarkar. Approximate divisor multiples - factoring with only a third of the secret crt-exponents. In *EUROCRYPT*, Lecture Notes in Computer Science. Springer, 2022. 2

MO15. Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228. Springer, 2015. 2

OS09. Raphael Overbeck and Nicolas Sendrier. *Code-based cryptography*, pages 95–145. Springer Berlin Heidelberg, 2009. 6

Pra62. Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Inf. Theory*, 8(5):5–9, 1962. 2, 14

PS98. Sarvar Patel and Ganapathy S. Sundaram. An efficient discrete log pseudo random generator. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 1998. 2

PV17. Kenneth G. Paterson and Ricardo Villanueva-Polanco. Cold boot attacks on NTRU. In *INDOCRYPT*, volume 10698 of *Lecture Notes in Computer Science*, pages 107–125. Springer, 2017. 2

Sho05. Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, USA, 2005. 8

SS92. Vladimir Michilovich Sidelnikov and Sergey O Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. 1992. 2

STK20. Kaichi Suzuki, Atsushi Takayasu, and Noboru Kunihiro. Extended partial key exposure attacks on RSA: improvement up to full size decryption exponents. *Theor. Comput. Sci.*, 841:62–83, 2020. 2

Vil19a. Ricardo Villanueva-Polanco. Cold boot attacks on BLISS. In *LATINCRYPT*, volume 11774 of *Lecture Notes in Computer Science*, pages 40–61. Springer, 2019. 2

Vil19b. Ricardo Villanueva-Polanco. *Cold boot attacks on post-quantum schemes*. PhD thesis, Royal Holloway, University of London, Egham, UK, 2019. 2

VP20. Ricardo Villanueva-Polanco. Cold boot attacks on LUOV. *Applied Sciences*, 10(12):4106, 2020. 2