Monoidic Codes in Cryptography

Paulo S. L. M. Barreto[*] Richard Lindner [†] Rafael Misoczki [‡]

September 13, 2011

Abstract

At SAC 2009, Misoczki and Barreto proposed a new class of codes, which have parity-check matrices that are quasi-dyadic. A special subclass of these codes were shown to coincide with Goppa codes and those were recommended for cryptosystems based on error-correcting codes. Quasi-dyadic codes have both very compact representations and allow for efficient processing, resulting in fast cryptosystems with small key sizes. In this paper, we generalize these results and introduce quasi-monoidic codes, which retain all desirable properties of quasi-dyadic codes. We show that, as before, a subclass of our codes contains only Goppa codes or, for a slightly bigger subclass, only Generalized Srivastava codes. Unlike before, we also capture codes over fields of odd characteristic. These include wild Goppa codes that were proposed at SAC 2010 by Bernstein, Lange, and Peters for their exceptional error-correction capabilities. We show how to instantiate standard code-based encryption and signature schemes with our codes and give some preliminary parameters.

Keywords: post-quantum cryptography, codes, efficient algorithms.

1 Introduction

In 1996, conventional public-key cryptography deployed in practice was shown to be susceptible to feasible attacks, if sufficiently large quantum computers were ever built. In order to counter such attacks preemptively, several computational problems resistant to quantum computer attacks have been studied for their usage as foundation of cryptographic security [BBD08].

One promising candidate of such computational problems is the syndrome decoding problem. McEliece showed in 1978 how to construct a public-key encryption scheme based on the problem of decoding binary Goppa codes to their full error-correction capability when given their generator matrix in a disguised form [McE78]. At ASIACRYPT 2001, Courtois, Finiasz, and Sendrier showed that a signature scheme can be based on the same problem [CFS01].

So far, no algorithm is capable of decoding Goppa codes, or the closely related Generalized Srivastava (GS) codes, better than completely random linear codes. And the problem of decoding random linear codes is widely believed to be very hard. The main drawback of cryptographic schemes which use Goppa/GS codes is that their keys are several orders of magnitude bigger than those of classical schemes with comparable practical security. This issue of big key sizes is directly related to the size of the code description. This is the main problem which we will address.

Related Work. The problem of finding Goppa/GS codes with small descriptions is not new.

In [BLP10], Bernstein, Lange, and Peters find that Goppa codes over $\mathbb{F}_q$, where the Goppa polynomial has t roots of multiplicity $r-1$ and r divides q, have the capability of correcting $\lfloor rq/2 \rfloor$ errors instead of the usual $\lfloor (r-1)q/2 \rfloor$ errors they can correct with an alternant decoder. These codes are called wild Goppa codes and due to their increased correction capability, one can use codes with smaller descriptions for the same level of practical security.

Another major breakthrough in saving description size has been achieved in [MB09] by Barreto and Misoczki. They define the new class of quasi-dyadic codes, which have very compact descriptions, and

[*]Escola Politécnica, Universidade de São Paulo, Brasil. E-Mail: pbarreto@larc.usp.br. Supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under research productivity grant 303163/2009-7.

[†]Technische Universität Darmstadt, Germany. E-Mail: rlindner@cs.tu-darmstadt.de.

[‡]Project SECRET, INRIA-Rocquencourt, France. E-Mail: rafael.misoczki@inria.fr.

show that this has a non-empty intersection with the class of binary Goppa codes. They also show how to generate codes in this intersection efficiently and give some preliminary parameters. Later, in [BCMN10], they are joined by Cayrel and Niebuhr and go on to show that quasi-dyadic Goppa codes can be generated in such a way that they are dense enough to be usable with the CFS signature scheme.

More generally, other proposals aimed at key reduction not restricted to Goppa codes were proposed [BC07, Gab05, MRS00] but subsequently broken [OTD10]; in special, [FOPT10a] and [GL10] presented structural attacks against McEliece variants with compact keys, being effective against quasi-cyclic codes [BCGO09]. With respect to the binary quasi-dyadic Goppa codes, this attack was not successful and, focused on increasing the effort of this attack, Persichetti proposed a construction using quasi-dyadic Srivastava codes [Per11], instead of Goppa ones, providing keys with similar size to the keys presented in [MB09].

Most attempts at decreasing key sizes deal with codes in characteristic 2, in spite of evidence [Pet10] that odd characteristics may offer security advantages.

Our Contribution. In this paper we introduce a new class of codes which allow for an extremely small representation and efficient processing. Our so called quasi-monoidic codes are a generalization of quasi-dyadic codes to finite fields of odd characteristics.

Using quasi-monoidic Goppa codes for the McEliece cryptosystems and Parallel-CFS signature scheme, one can potentially obtain smaller key sizes than before, as exemplified by Tables 1 and 3 in Section 6. For example, we find that many wild Goppa codes are in fact quasi-monoidic.

Organization. In Section 2, we introduce the basic concepts of coding theory, which are relevant to our proposal. In Section 3, we introduce our new class of quasi-monoidic codes and show how to construct Goppa/GS codes that are quasi-monoidic. Next, we describe how to instantiate the standard code-based encryption and signature schemes with this family in Section 4. Afterwards, in Section 5 we assess the security properties of our proposal, and in Section 6 we suggest a few actual parameters to encourage further analysis. Finally, in Section 7 we briefly argue why the matrix-vector products for quasi-monoidic matrices can be computed efficiently using a discrete Fourier transform.

2 Coding Theory

Basic concepts. We will start with some matrix descriptions. For both descriptions, t is an integer greater than zero. Given a sequence $L = (L_0, \ldots, L_{n-1}) \in \mathbb{F}_q^n$, the *Vandermonde matrix* $\mathrm{vdm}(t, L)$ is the $t \times n$ matrix with elements $V_{ij} = L_j^i$. Given a polynomial g with coefficients $(g_1, \ldots, g_t) \in \mathbb{F}_q^n$, the *Toeplitz matrix* $\mathrm{toep}(g_1, \ldots, g_t)$ is the $t \times t$ matrix with elements $T_{ij} := g_{t-i+j}$ for $j \leqslant i$ and $T_{ij} := 0$ otherwise. The following are the *GRS*, *alternant* and *Goppa codes* definitions.

Definition 2.1. Given a sequence $L = (L_0, \ldots, L_{n-1}) \in \mathbb{F}_q^n$ of distinct elements and a sequence $D = (D_0, \ldots, D_{n-1}) \in \mathbb{F}_q^n$ of nonzero elements, the *Generalized Reed-Solomon code* $GRS_r(L, D)$ is the $[n, k, r]$ linear error-correcting code defined by the parity-check matrix

$$H = \mathrm{vdm}(r-1, L) \cdot \mathrm{diag}(D).$$

An *alternant code* is a subfield subcode of a Generalized Reed-Solomon code.

Definition 2.2. Given a prime power p, $q = p^d$ for some d, a sequence $L = (L_0, \ldots, L_{n-1}) \in \mathbb{F}_q^n$ of distinct elements and a polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t such that $g(L_i) \neq 0$ for $0 \leqslant i < n$, the *Goppa code* $\Gamma(L, g)$ over $\mathbb{F}_p$ is defined by the parity-check matrix

$$\begin{aligned} H \;=\; & \mathrm{toep}(g_1, \ldots, g_t) \\ \cdot\; & \mathrm{vdm}(t, L_0, \ldots L_{n-1}) \\ \cdot\; & \mathrm{diag}(g(L_0)^{-1}, \ldots, g(L_{n-1})^{-1}) \end{aligned} \tag{2.1}$$

By *[MS77][Ch. 12, §3]*, we can omit the matrix $\mathrm{toep}(g_1, \ldots, g_t)$ from this construction, making it easy to see that *Goppa codes* are also *alternant codes* over $\mathbb{F}_p$ corresponding to $GRS_t(L, D)$ where $D = (g(L_0)^{-1}, \ldots, g(L_{n-1})^{-1})$.

Goppa codes have minimum distance at least $t+1$. Binary Goppa codes improve this to at least $2t+1$. It turns out that, although this improvement does not hold in general for larger characteristics, codewords that differ by vectors whose components are all equal are on average much more sparsely distributed. Thus, while the unambiguous correction of general errors in odd characteristics can in general not proceed beyond about $t/2$ errors, correction of error patterns of homogeneous (all-equal) error magnitudes can probabilistically reach as much as t errors [BLM10].

Goppa codes in Tzeng-Zimmermann form. It was shown by Tzeng and Zimmermann [TZ75], that all Goppa codes with Goppa polynomial $g(x) = h(x)^r$, for some square-free $h(x)$ and number $r > 0$, admit a parity-check matrix consisting solely of Cauchy power matrices over the splitting field of $g(x)$.

Definition 2.3. Let $\mathbb{F}$ be a finite field, and $\beta = (\beta_0, \beta_1 \dots, \beta_{t-1}), \gamma = (\gamma_0, \gamma_1, \dots, \gamma_{n-1})$ be two disjoint sequences of distinct elements in $\mathbb{F}$. The *Cauchy matrix* $C(\beta, \gamma)$ associated with these sequences is one where $C_{i,j} = (\beta_i - \gamma_j)^{-1}$, i.e.,

$$C = \begin{pmatrix} (\beta_0 - \gamma_0)^{-1} & \cdots & (\beta_0 - \gamma_{n-1})^{-1} \\ \vdots & & \vdots \\ (\beta_{t-1} - \gamma_0)^{-1} & \cdots & (\beta_{t-1} - \gamma_{n-1})^{-1} \end{pmatrix}.$$

For any additional integer $r > 0$, the associated *Cauchy power matrix* $C(\beta, \gamma, r)$ is a Cauchy matrix, where each coordinate is raised to the r-th power, i.e., $C_{i,j} = (\beta_i - \gamma_j)^{-r}$.

Finally, the *Cauchy layered matrix* $CL(\beta, \gamma, r)$ consists of all Cauchy power matrices with exponents up to r, i.e.,

$$CL(\beta, \gamma, r) = \begin{pmatrix} C(\beta, \gamma) \\ C(\beta, \gamma, 2) \\ \vdots \\ C(\beta, \gamma, r) \end{pmatrix}.$$

There is an ambivalence in this definition, i.e., there is no bijection from all sequences β and γ to all Cauchy matrices. Specifically, for any $\omega \in \mathbb{F}$, we have $C(\beta, \gamma) = C(\beta + \omega, \gamma + \omega)$.

In terms of properties, Cauchy matrices are very similar to Vandermonde matrices. For example, there are efficient algorithms to compute matrix-vector products, submatrices of Cauchy matrices are again Cauchy, all Cauchy matrices have full-rank, and there are closed formulas for computing their determinant.

As mentioned before, Tzeng and Zimmermann showed that all Goppa codes, where the Goppa polynomial is the r-th power of an square-free polynomial, admit a parity-check matrix which is a Cauchy layered matrix. This parity-check matrix is in TZ form. Specifically, the parity-check matrix H in TZ form of the Goppa code with support $L = \{\gamma_0, \dots, \gamma_{n-1}\}$ and Goppa polynomial $g(x) = \prod_{i=0}^{t-1}(x - \beta_i)^r$ is $H = CL(\beta, \gamma, r)$.

This is particularly interesting for the case of wild Goppa codes as introduced by Bernstein, Lange, and Peters [BLP10]. They show that if r divides the field characteristic, then the rows of this TZ parity-check matrix are not linearly independent, but the rows of $H' = CL(\beta, \gamma, r-1)$, where we omit the last Cauchy block, are already a parity-check matrix of the full code. This allows wild Goppa codes to achieve error-correcting capabilities surpassing general alternant codes and make them particularly interesting for various application including cryptography.

Generalized Srivastava codes.

Definition 2.4. Let $(\alpha_1, \dots, \alpha_n)$, $(\omega_1, \dots, \omega_s)$ are $n + s$ distinct elements of $\mathbb{F}_{q^m}$, and $(z_1, \dots, z_n)$ are nonzero elements of $\mathbb{F}_{q^m}$. The *Generalized Srivastava code* is an $[n, k \geq n - mst, d \geq st + 1]$ code over $\mathbb{F}_q$, is also an alternant code, and is defined by the parity-check matrix

$$H = \begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_s \end{pmatrix}$$

where

$$H_l = \begin{pmatrix} \frac{z_1}{\alpha_1-\omega_l} & \frac{z_2}{\alpha_2-\omega_l} & \cdots & \frac{z_n}{\alpha_n-\omega_l} \\ \frac{z_1}{(\alpha_1-\omega_l)^2} & \frac{z_2}{(\alpha_2-\omega_l)^2} & \cdots & \frac{z_n}{(\alpha_n-\omega_l)^2} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{z_1}{(\alpha_1-\omega_l)^t} & \frac{z_2}{(\alpha_2-\omega_l)^t} & \cdots & \frac{z_n}{(\alpha_n-\omega_l)^t} \end{pmatrix}$$

for $l = 1, \ldots, s$. The original Srivastava codes are the case $t = 1$, $z_i = \alpha_i^\mu$ for some μ.

For more details about Generalized Srivastava codes, see [MS77][Ch. 12, §6].

3 Quasi-Monoidic Codes

Monoidic matrices.

Definition 3.1. Let R be a commutative ring, $A = \{a_0, \cdots, a_{N-1}\}$ a finite abelian group of size $|A| = N$ with neutral element $a_0 = 0$, and $h\colon A \longrightarrow R$ a sequence indexed by A. The A-adic matrix $M(h)$ associated with this sequence is one for which $M_{i,j} = h(a_i - a_j)$ holds, i.e.,

$$M = \begin{pmatrix} h(0) & h(-a_1) & \cdots & h(-a_{N-1}) \\ h(a_1) & h(0) & \cdots & h(a_1 - a_{N-1}) \\ \vdots & \vdots & \ddots & \vdots \\ h(a_{N-1}) & h(a_{N-1} - a_1) & \cdots & h(0) \end{pmatrix}.$$

All A-adic matrices form a ring that is isomorphic to the monoid ring $R[A]$, which is studied in abstract algebra [Lan02]. We use the additive notation for the finite abelian group A here for practical purposes, but the definition can be generalized to all groups, in which case one might prefer the multiplicative notation.

Some A-adic matrices have special names, for example the $\mathbb{Z}_2^d$-adic matrices are dyadic and the $\mathbb{Z}_3^d$-adic matrices are triadic. If we do not want to specify the group A explicitly, we will say the matrix is monoidic. So, to identify all Goppa codes with a monoidic representation, we continue by giving necessary and sufficient conditions for Cauchy matrices to be monoidic and show that the case for Cauchy power matrices follows from that.

Conditions for which monoidic implies Cauchy.

Theorem 3.2. *Let $M(h)$ be A-adic for a sequence h of length N over $\mathbb{F}$. Then M is Cauchy iff*

(1) *$h(a_i)$ are distinct and invertible in $\mathbb{F}$ for all $0 \le i < N$, and*

(2) *$(h(a_i - a_j))^{-1} = (h(a_i))^{-1} + (h(-a_j))^{-1} - (h(0))^{-1}$ for all $0 \le i, j < N$.*

In this case $M(h) = C(\beta, \gamma)$, where $\beta(a_i) = (h(a_i))^{-1}$ and $\gamma(a_i) = (h(0))^{-1} - (h(-a_i))^{-1}$.

Proof. We start by showing that our conditions indeed imply that M is Cauchy. For the disjointness, assume that there are indices i and j, such that $\beta(a_i) = \gamma(a_j)$. In this case we get $0 = \beta(a_i) - \gamma(a_j) = 1/h(a_i - a_j)$, which is a contradiction. Finally we compare the matrices $M(h)$ and $C(\beta, \gamma)$ resulting in the equality

$$M_{i,j} = h(a_i - a_j) = 1/(1/h(a_i) + 1/h(-a_j) - 1/h(0)) = 1/(\beta(a_i) - \gamma(a_j)) = C_{i,j}.$$

We continue by showing that if M is Cauchy, i.e., $M(h) = C(\beta', \gamma')$, then indeed our conditions must hold. Since $C(\beta', \gamma') = C(\beta' + \omega, \gamma' + \omega)$ for any $\omega \in \mathbb{F}$, we can choose the sequences in such a way that $\gamma'(0) = 0$. Now, $M_{i,0} = C_{i,0}$ for all i, which means $h(a_i) = 1/\beta'(a_i)$. By the properties of β' this gives us condition (1), i.e., that all $h(a_i)$ are distinct and invertible, as well as $\beta' = \beta$. We use similarly that $M_{0,i} = C_{0,i}$ which implies $h(-a_i) = 1/(\beta(0) - \gamma'(a_i))$. Solving for γ' reveals that it equals γ. Since $\beta = \beta'$ and $\gamma = \gamma'$, we get that $M(h) = C(\beta, \gamma)$ implying condition (2). □ □

Note that if the A-adic matrix of a sequence h is also Cauchy, then the sequence of r-th powers, i.e., $h^r = (h_0^r, h_{a_1}^r, \ldots, h_{a_{n-1}}^r)$ yields the corresponding Cauchy power matrix. In other words, for any number $r > 0$ we have $M(h) = C(\beta, \gamma) \implies M(h^r) = C(\beta, \gamma, r)$.

Now, we will show how to construct random monoidic Cauchy matrices.

Construction of monoidic Cauchy matrices.

Corollary 3.3. *Let A be a finite, abelian group with set of generators $b_1, \dots, b_d$ and $M(h)$ be A-adic and Cauchy for a sequence h over $\mathbb{F}$, then for all $c_1, \dots, c_d \in \mathbb{Z}$,*

$$(h(c_1 b_1 + \cdots + c_d b_d))^{-1} = c_1(h(b_1))^{-1} + \cdots + c_d(h(b_d))^{-1} - (c_1 + \cdots + c_d - 1)(h(0))^{-1}.$$

Furthermore, the field characteristic $\operatorname{char}(\mathbb{F})$ *divides the order of any element in* $A \setminus \{0\}$.

Proof. By Theorem 3.2, we know that for all $a, a' \in A$ the following holds

$$(h(a + a'))^{-1} = (h(a))^{-1} + (h(a'))^{-1} - (h(0))^{-1}.$$

By repeatedly using this equation, we prove the first claim.

$$\begin{aligned}
(h(c_1 b_1 + \cdots + c_d b_d))^{-1} &= (h(\underbrace{b_1 + \cdots + b_1}_{c_1 \text{ times}} + \cdots + \underbrace{b_d + \cdots + b_d}_{c_d \text{ times}}))^{-1} \\
&= (h(b_1))^{-1} + (h(\underbrace{b_1 + \cdots + b_1}_{(c_1 - 1) \text{ times}} + \cdots + \underbrace{b_d + \cdots + b_d}_{c_d \text{ times}}))^{-1} - (h(0))^{-1} \\
&= c_1(h(b_1))^{-1} + (h(\underbrace{b_2 + \cdots + b_2}_{(c_2) \text{ times}} + \cdots + \underbrace{b_d + \cdots + b_d}_{c_d \text{ times}}))^{-1} - c_1(h(0))^{-1} \\
&= c_1(h(b_1))^{-1} + \cdots + c_d(h(b_d))^{-1} - (c_1 + \cdots + c_d - 1)(h(0))^{-1}.
\end{aligned}$$

For the second claim, let $a \in A \setminus \{0\}$ be a non-neutral group element and $k = \operatorname{ord}(a)$, i.e., $ka = 0$. By the equation we have just shown, we know that

$$\begin{gathered}
h(0)^{-1} = h(ka)^{-1} = kh(a)^{-1} - (k-1)h(0)^{-1} \\
k(h(0)^{-1} - h(a)^{-1}) = 0
\end{gathered}$$

Since a is not the neutral element, all elements of h are distinct, and the field characteristic is prime, the second claim follows. □ □

Since the field characteristic p divides the order of any element, only groups of size $N = p^d$ can be used. Conversely, let $b_1, \dots, b_d$ be group elements that form an $\mathbb{F}_p$ set of generators, then the sequence elements $h(0), h(b_1), \dots, h(b_d)$ completely determine the sequence. We call these values the essence of the sequence h.

For example, if $A = \mathbb{F}_p^d$, then such a set of generators $b_1, \dots, b_d$ is given by the generators of the d distinct copies of $\mathbb{F}_p$ in A. For a given set of generators, we can sample a monoidic sequence uniformly at random with the algorithm in Figure 2.

We will briefly argue why the algorithm in Figure 2 is correct. Assume that it is not. The only situation resulting in an error is in line 7, if the computed quantity is not invertible, so let us assume this to be the case. Since only zero is not invertible, we have

$$0 = c_1 h(b_1) + \cdots + c_d h(b_d) - (c_1 + \cdots + c_d - 1)h(0).$$

Now, not all coefficients of $h(0), h(b_1), \dots, h(b_d)$ can be zero simultaneously, so there is an $\mathbb{F}_p$-linear dependency among them. However, by our choice of F in line 4, from which all $h(b_i)$ are chosen, no such dependency can exist.

As a consequence of our algorithm, the total number of possible sequences is

$$|\{h \colon \mathbb{F}_p^d \to \mathbb{F}_Q \mid M(h) \text{ is monoidic and Cauchy}\}| = (Q-1) \cdots (Q - p^d).$$

Description	Parameter	Restriction
Field char	p	prime
Base field	q	p^s
Extension field	Q	q^m
Group order	N	$p^d \leq Q/p$

Description	Parameter	Restriction
Goppa roots	t	$< n/m$
Goppa multiplicity	r	$< n/(tm)$
Blocksize	$\mathtt{b}$	$\gcd(t, N)$
Code length	n	$\mathtt{b}\ell < N$

Figure 1: Parameters for quasi-monoidic GS codes. Let $s, m, \ell > 0$. For brevity, we will focus on the case where $s = 1$ (smallest base field size), $r = p - 1$ (wild case).

```
MonoidCauchy($p, Q, d$):
  1. $F \longleftarrow \mathbb{F}_Q \setminus \{0\}$
  2. $h(0) \longleftarrow U(F)$
  3. For $i = 1, \dots, d$:
  4.     $F \longleftarrow \mathbb{F}_Q \setminus (\mathbb{F}_p\, h(0) + \mathbb{F}_p\, h(b_1) + \cdots + \mathbb{F}_p\, h(b_{i-1}))$
  5.     $h(b_i) \longleftarrow U(F)$
  6. For $c_1, \dots, c_d \in \mathbb{F}_p$:
  7.     $h(c_1 b_1 + \cdots + c_d b_d) \longleftarrow c_1 h(b_1) + \cdots + c_d h(b_d) - (c_1 + \cdots + c_d - 1)h(0)$
  8. Output $(h(0)^{-1}, h(a_1)^{-1}, \dots, h(a_{p^d-1})^{-1})$
```

Figure 2: Choosing A-adic Cauchy sequences, where $A = \{0, a_1, \dots, a_{p^d-1}\}$ has set of generators $b_1, \dots, b_d$.

```
QuasiMonoidic(...):
  1. $h \longleftarrow$ MonoidCauchy($p, Q, d$); $\omega \longleftarrow U(\mathbb{F}_Q)$
  2. For $i = 0, \dots, t-1$: $\beta_i \longleftarrow (h(a_i))^{-1} + \omega$                              "Goppa roots"
  3. For $i = 0, \dots, N-1$: $\gamma_i \longleftarrow (h(0))^{-1} - (h(-a_i))^{-1} + \omega$            "Goppa support"
  4. $\tau \longleftarrow U(S_{N/\mathtt{b}})$                                                            "Block permutation"
  5. $\pi_0, \dots, \pi_{\ell-1} \longleftarrow U(\{0, \dots, \mathtt{b}-1\})$                              "Support permutations"
  6. $\sigma_0, \dots, \sigma_{\ell-1} \longleftarrow U(\mathbb{F}_q^*)$                                    "Scaling"
  7. For $i = 0, \dots, \ell-1$: $\widehat{\gamma}_i \longleftarrow (\gamma_{\tau(i)\mathtt{b}}, \dots, \gamma_{\tau(i)\mathtt{b}+\mathtt{b}-1})$   "Select blocks"
  8. For $i = 0, \dots, \ell-1$: $\widehat{\gamma}_i \longleftarrow \widehat{\gamma}_i M(\chi_{a_{\pi_i}})$          "Permute support"
  9. $H \longleftarrow [CL(\beta, \widehat{\gamma}_0, r)\sigma_0 \mid \cdots \mid CL(\beta, \widehat{\gamma}_{\ell-1}, r)\sigma_{\ell-1}]$   "Parity-check matrix"
 10. $H \longleftarrow$ QMTrace($q, \mathtt{b}, H$)
 11. $H \longleftarrow$ QMGauss($\mathtt{b}, H$)
 12. $H \longleftarrow$ QMSignature($\mathtt{b}, H$)
 13. Output private $\beta, \widehat{\gamma}_0, \dots, \widehat{\gamma}_{\ell-1}, \sigma_0, \dots, \sigma_{\ell-1}$; public $H$
```

Figure 3: Choosing quasi-monoidic GS codes with private and public description. Here, $S_{N/\mathtt{b}}$ is the group of permutations on $\{0, \dots, N/\mathtt{b} - 1\}$ and $\chi_{a_{\pi_i}}$ is the characteristic function of the group element a_{π_i}.

Quasi-monoidic Generalized Srivastava codes. Our final goal is to describe a way of disguising the Cauchy block structures of the code that is used for error-correction, while simultaneously keeping much of the monoidic structure intact in the form of small monoidic blocks. This will allow us to obtain code-based public-key schemes with small keys.

The relevant parameters used for this process are described in Figure 1 and the corresponding algorithm is presented in Figure 3. We will continue by explaining some details including the QM-subroutines used therein and conclude with a clarifying example.

We start the generation process by choosing a random fully monoidic Goppa code of length N. Then we split the support in blocks of length $\mathtt{b}$ and select ℓ such blocks at random to comprise the support of our quasi-monoidic code. To each chosen support block, we apply a random monoidic permutation, i.e., we multiply with the matrix $M(\chi_{a_\pi})$, where χ_{a_π} is the characteristic function of the group element a_π, for a randomly chosen π. Since χ is a characteristic function, this matrix will have a single non-zero coefficient being 1 per row and column, so it is a permutation matrix. Furthermore, this transformation preserves the monoidic structure of the block and indeed all monoidic permutations have this form.

We continue by creating the parity-check matrix H of our code consisting of the ℓ scaled Cauchy layered matrices corresponding to each block. The resulting matrix consists of $tr/\mathtt{b} \times \ell$ monoidic blocks of size $\mathtt{b}$ and we will keep this quasi-monoidic structure intact for the remainder. Note that if $q > 2$, i.e., we have non-trivial scaling factors, then the code defined via our parity-check matrix need not be Goppa anymore, but it is always a Generalized Srivastava code.

The first subroutine QMTRACE will generate a parity-check matrix for the corresponding subfield subcode over the base field $\mathbb{F}_q$. Recall that $\mathbb{F}_Q = \mathbb{F}_q[x]/\langle f \rangle$ for some irreducible polynomial f of degree m. We can identify each matrix coefficient $h_{i,j}$ with its representative polynomial $h_{i,j,0} + h_{i,j,1}x + \cdots + h_{i,j,m-1}x^{m-1}$ of smallest degree. We expand the matrix rows by a factor of m and distribute the entries as follows $h^{\text{new}}_{kt+i,j} \longleftarrow h_{i,j,k}$,i.e., in order to keep the block structure intact, we first take all constant terms of coefficients in a block then all linear terms and so on.

The second subroutine QMGAUSS will compute the quasi-monoidic systematic form of the parity-check matrix. It does so by identifying each monoidic block with an element of the corresponding ring of monoidic matrices and performing the usual Gauss algorithm on those elements. Since this ring is not necessarily an integral domain, the algorithm may find that a pivot element is not invertible. In this case, the systematic form we seek does not exist and the algorithm has to loop back to the "Block permutation" step. Fortunately, the chance of this is small. The probability that the matrix is nonsingular is $\prod_{j=0}^{k-1} 1 - 1/p^{k-j}$, which approaches a constant (to be determined numerically) for large k. This constant is different for each p but tends to 1 for large p. In order to avoid redundancy, this subroutine omits those columns of the systematic form, which we know to be the identity matrix.

The third and final subroutine QMSIGNATURE will simply extract the monoidic signature of each block, i.e., its first column. For the whole quasi-monoidic matrix, this simply amounts to extracting each $\mathtt{b}$-th column. This concludes our description of the algorithm.

In Appendix A, we give a detailed example of the generation process that illustrates some subtleties of the algorithm.

Decoding quasi-monoidic Goppa codes. In [BLM10], an efficient decoding algorithm for square-free (irreducible or otherwise) Goppa codes over $\mathbb{F}_p$ for any prime p is presented. Since it fits perfectly to decode quasi-monoidic Goppa codes, we will provide a brief description of this method in Appendix B.

Decoding GS codes is less studied than the case of Goppa codes, though both are closely related. Let D be a diagonal matrix containing the scaling factors, then adding an error pattern e to a GS codeword c amounts to adding the pattern eD to the codeword cD of the associated unscaled Goppa code. So, if the Goppa decoder capability depends only on the weight of the error pattern (like the wild decoder [BLP10]), then it can be used equally well for GS codes and scaling could be used. On the other hand, if the Goppa decoder capability is best if all error magnitudes coincide (like the "equal magnitude" decoder [BLM10]), then scaling must not be used. It turns out that, in the latter case, keys also get potentially smaller due to the larger number of correctable errors.

4 Monoidic Encryption and Signatures

In this section we provide the basic description about the McEliece encryption scheme [McE78] and the Parallel-CFS signature scheme [Fin10]. Both of them can be instantiated with our monoidic codes.

4.1 McEliece encryption scheme

Let the security level be λ. The parameters for the code below are assumed to be chosen so that the cost of the best attack against it is at least 2^{λ} operations (see [Pet11] for a recent survey).

Key Generation: Choose a prime p, a finite field $\mathbb{F}_q$ with $q = p^m$ for some $m > 0$ and a Quasi-Monoidic code $\Gamma(L, g)$ with support $L = (L_0, \ldots, L_{n-1}) \in (\mathbb{F}_q)^n$ of distinct elements and a square-free generator polynomial $g \in \mathbb{F}_q[x]$ of degree t, satisfying $g(L_j) \neq 0$, $0 \leqslant j < n$, both provided by the algorithm of Figure 3. Let $k = n - mt$. Compute a systematic generator matrix $G \in \mathbb{F}_p^{k \times n}$ for $\Gamma(L, g)$, i.e. $G = [I_k \mid -M^T]$ for some matrix $M \in \mathbb{F}_p^{mt \times k}$ and I_k an identity matrix of size k. The private key is $sk := (L, g)$ and the public key is $pk := (M, t)$.

Encryption: To encrypt a plain text $d \in \mathbb{F}_p^k$, choose an error-vector $e \in \{0, 1\}^n \subseteq \mathbb{F}_p^n$ with weight $\mathsf{wt}(e) \leqslant t$, and compute the cipher text $c \leftarrow dG + e \in \mathbb{F}_p^n$.

Decryption: To decrypt a cipher text $c \in \mathbb{F}_p^n$ knowing L and g, compute the decodable syndrome of c, apply a decoder to determine the error-vector e, and recover the plain text d from the first k columns of $c - e$.

4.2 Parallel-CFS

To sign a document with the standard CFS signature schemes [CFS01] we should hash the document into a syndrome and then decode it to an error vector of certain weight t. Since not all syndromes are decodable, a counter is hashed with the message, and the signer tries successive counter values until a decodable syndrome is found. The signature consists of both the error pattern of weight t corresponding to the syndrome and the counter value yielding this syndrome. In [FS09] is described an unpublished attack by D. Bleichenbacher showing that the usual parameters are insecure and the improved parameters result in a signature scheme with excessive cost of signing time or key length.

To address this problems, M. Finiasz proposed in [Fin10] the Parallel-CFS, which can be described as follows: instead of producing one hash (using a function $\mathcal{H}$) from a document D and signing it, one can produce i hashes (using i different functions $\mathcal{H}_1, \ldots, \mathcal{H}_i$) and sign all $\mathcal{H}_1(D), \ldots, \mathcal{H}_i(D)$ in parallel. Then Parallel-CFS can be described by the following algorithms.

Key Generation: Choose parameters m, t and let $n = 2^m$. Select δ such that $\binom{2^m}{t+\delta} > 2^{mt}$. Choose a Quasi-Monoidic code $\Gamma(g, L)$, where g is a polynomial of degree t in $\mathbb{F}_{2^m}[X]$ and a support $L = (L_0, \ldots, L_{n-1}) \in \mathbb{F}_{2^m}^n$. Let H be a $mt \times n$ systematic parity-check matrix of Γ. H is the public verification key and $\Gamma(g, L)$ represents the private signature key.

Signature: For i signatures in parallel (see Table 2 column "sigs", based on [Fin10], for this estimation), the signer tries to guess δ errors, searching all error patterns $\phi_\delta(j)$ of weight δ, and then applies the decoding algorithm to the resulting syndrome $s_{j,i} = \mathcal{H}_i(D) + H \cdot \phi_\delta(j)^T$. Once a decodable syndrome is found for an $j_{0,i}$, then there exists a plain text $p'_{j_{0},i}$, such that $H \cdot \phi_t(p'_{j_{0},i})^T = s_{j_0,i} = \mathcal{H}_i(D) + H \cdot \phi_\delta(j_0)^T$.

With the error patterns $e_i = \phi_t(p'_{j_{0},i}) + \phi_\delta(j_0)$ of weight at most $t + \delta$, it holds that $H \cdot e_i^T = \mathcal{H}_i(D)$, for i signatures. The signature is $(\phi_{t+\delta}^{-1}(e_1) \| \ldots \| \phi_{t+\delta}^{-1}(e_i))$.

Verification: Given a signature $(p_1 \| \ldots \| p_i)$ for a document D, the verification step consists of checking the i equalities $H \cdot \phi_{t+\delta}(p_i)^T \stackrel{?}{=} \mathcal{H}_i(D)$.

Table 1: Encryption quasi-monoidic codes.

level	p	m	n	k	t	key(bits)	syndrome(bits)
80	2	12	3840	768	256	9216	3072
80	3	8	2430	486	243	6163	3082
80	5	5	1000	375	125	4354	1452
80	167	3	668	167	167	3700	3700
112	2	12	2944	1408	128	16896	1536
112	3	8	2673	729	243	9244	3082
112	11	5	1089	484	121	8372	2093
112	241	3	964	241	241	5722	5722
128	2	12	3200	1664	128	19968	1536
128	3	9	3159	972	243	13866	3467
128	5	5	5000	625	625	10159	10159
128	373	3	1492	373	373	9560	9560
192	2	14	6144	2560	256	35840	3584
192	3	10	4131	1701	243	26961	3852
192	29	6	5887	841	841	24514	24514
192	547	4	2735	547	547	19901	19901
256	2	15	11264	3584	512	53760	7680
256	7	9	5145	2058	343	51998	8667
256	37	6	9583	1369	1369	42791	42791
256	907	4	4535	907	907	35645	35645

Table 2: Encryption quasi-monoidic codes yielding short syndromes.

level	p	m	n	k	t	key(bits)	syndrome(bits)
80	2	11	1792	1088	64	11968	704
80	7	5	735	490	49	6879	688
80	41	3	451	328	41	5272	659
128	2	12	3200	1664	128	19968	1536
128	3	9	2106	1377	81	19643	1156
128	7	6	1813	1519	49	25587	826
192	2	14	5376	3584	128	50176	1792
192	3	11	4536	3645	81	63550	1413

CFS-friendly quasi-monoidic Goppa codes. There is a simple extension to the construction of quasi-dyadic codes that applies to our quasi-monoidic codes as well. These CFS-friendly codes were proposed in [BCMN10] and we will briefly describe the idea. Recall that MonoidCauchy constructs a full monoidic $N \times N$ parity-check matrix of which, after some scaling and permuting, we will use only a $t \times n$ submatrix. The idea is to relax the construction of the full matrix, allowing for some undefined entries, as long as they do not end up in the submatrix we actually use.

This relaxation is realized by omitting line 4 of MonoidCauchy (Fig. 2), i.e., the condition of linear independence of the essential entries in the inverted monoidic sequence. This may cause some entries in the sequence to be 0, so we cannot invert them in the final step of the algorithm and just leave them at 0, since no legal entry can have that value. Now, after selecting the submatrix in QuasiMonoidic (Fig. 3), i.e., after line 7, we need to check that all matrix coefficients are non-zero and restart if there are any. This is unlikely since the submatrix is usually small.

The whole relaxation allows us to work with smaller extension fields $\mathbb{F}_Q$, because we now need only $t + n$ distinct elements in $\mathbb{F}_Q^*$, where before we needed $2N$. So the codes we produce will be denser and thus more suited for the CFS signature scheme.

Table 3: Parallel CFS quasi-monoidic codes.

level	p	m	n	k	t	key (bits / KiB)	sigs	δ	sigbits
80	2	15	32580	32400	12	1458000 / 178	2	4	326
80	3	11	177048	176949	9	3085033 / 377	3	2	375
80	13	4	28509	28457	13	421214 / 52	2	4	342
112	2	20	1048332	1048092	12	62885520 / 7677	3	3	636
112	11	6	1771495	1771429	11	36768825 / 4489	3	2	558
112	13	5	371228	371163	13	6867332 / 839	3	3	624
128	2	23	8388324	8388048	12	578775312 / 70652	3	2	684
128	5	8	390495	390375	15	21754145 / 2656	3	4	759
128	13	6	4826731	4826653	13	107164431 / 13082	2	3	514

5 Security assessment

Decoding attacks. In estimating *concrete* security (rather than asymptotic behavior only), we adopt the following criteria, which were discussed and analyzed by Finiasz and Sendrier [FS09] and by Peters [Pet11, Observation 6.9] (see also [BLP11]), whereby directly decoding a code of length n, dimension k, and generic error patterns of weight w over $\mathbb{F}_q$, without using the trapdoor, has a *workfactor* at least WF_q measured in bit operations. Typically $\wp \approx w/2$ and $\ell \gtrsim \log_q \binom{k/2}{\wp} + \wp \log_q(q-1)$:

$$\mathrm{WF}_2 = \min_{\wp,\ell} \left\{ \begin{array}{c} \frac{1}{2}(n-k)^2(n+k) + \Big(k/2 - \wp + 1 + \binom{\lfloor k/2 \rfloor}{\wp} + \binom{\lceil k/2 \rceil}{\wp}\Big)\,\ell \\ +(w - 2\wp + 1)4\wp\binom{\lfloor k/2 \rfloor}{\wp}\binom{\lceil k/2 \rceil}{\wp} \end{array} \right\} \tag{5.1}$$

$$\mathrm{WF}_q = \min_{\wp,\ell} \left\{ \begin{array}{c} (n-k)^2(n+k) + \Big(k/2 - \wp + 1 + \Big(\binom{\lfloor k/2 \rfloor}{\wp} + \binom{\lceil k/2 \rceil}{\wp}\Big)(q-1)^{\wp}\Big)\,\ell \\ +\dfrac{q}{q-1}(w - 2\wp + 1)2\wp\left(1 + \dfrac{q-2}{q-1}\right)\dfrac{\binom{\lfloor k/2 \rfloor}{\wp}\binom{\lceil k/2 \rceil}{\wp}(q-1)^{2\wp}}{q^{\ell}} \end{array} \right\} \tag{5.2}$$

When it is known beforehand that all errors have equal magnitude and $q > 2$, we simplify Equation 5.2 accordingly:

$$\mathrm{WF}'_q = \min_{\wp,\ell} \left\{ \begin{array}{c} (n-k)^2(n+k) + \Big(k/2 - \wp + 1 + \Big(\binom{\lfloor k/2 \rfloor}{\wp} + \binom{\lceil k/2 \rceil}{\wp}\Big)(q-1)\Big)\,\ell \\ +\dfrac{q}{q-1}(w - 2\wp + 1)2\wp\left(1 + \dfrac{q-2}{q-1}\right)\dfrac{\binom{\lfloor k/2 \rfloor}{\wp}\binom{\lceil k/2 \rceil}{\wp}(q-1)}{q^{\ell}} \end{array} \right\} \tag{5.3}$$

The results of this estimations are provided in Tables 1, 2, and 3 and discussed in Section 6.

Structural attacks. Structural attacks against families of codes that yield compact keys McEliece have also been proposed. In [FOPT10a], the idea is to convert the public code into a multivariate nonlinear system and then trying to solve it with Gröbner basis techniques. A related technique inspired by the Sidelnikov-Shestakov attack [SS92] is described in [GL10].

The former attack recovers variables x_i and y_i which denote respectively the diagonal and the support of the code, i.e. the x_i define the Vandermonde matrix V, and the y_i define the diagonal matrix D, which compose the parity-check matrix $H = VD$ in the alternant case. In the Goppa case, these variables are coupled by a more complex relationship, namely $y_i = g(x_i)^{-1}$. In both cases, the result is a multivariate system, with equations of degree up to t, namely, $H_{ij} = x_j^i y_j$ for $0 \le i < t$ and $0 \le j < n$.

For generic codes, this system is too complex to be feasibly solved with Gröbner bases. However in the dyadic case (and, by extension, the monoidic case), many equations are redundant, due to relation (2) of Theorem 3.2. Furthermore, for subcodes defined over extension fields (but not over the base field itself), it turns out that only linear and quadratic equations are enough to specify variables x_i and y_i. This feature yields a simpler multivariate system that can be tackled with, and in fact the corresponding quasi-dyadic codes over extension fields in [MB09] can be broken this way.

However, for the case of a subcode defined over the base field, the associated Gröbner basis is trivial if only linear and quadratic equations are used to define the x_i and the y_i variables, and the attack fails [FOPT10b]. Although those results were obtained for characteristic 2, at the time of writing there does not appear to be any way to take advantage of larger characteristics to improve this attack. Exploring this line of attack is thus left as an open problem for followup research.

Regarding the attack in [GL10], a 'small' extension degree m could lead to a successful break, but it is unclear how small m must be so that such an attack would become feasible. Just how small m should be for the attack to be successful in each characteristic $p > 2$ is unclear, though. In this sense, the parameters listed in this paper deliberately use relatively small values of m, in the hope that they stimulate further cryptanalysis research. While we stress that these parameters are not designed for effective deployment, the indicated security levels correspond to the best known generic decoding attacks so as to give a realistic impression of what practical might look like.

6 Parameters of Cryptographic Interest

We now assess the efficiency of the proposed codes in possible practical cryptographic scenarios.

Tables 1, 2, and 3 compare some of the best quasi-monoidic codes achievable for each characteristic at several security levels. These figures only assume the ability to correct t errors of equal magnitude, already taking into account that this choice of introduced errors decreases the WF to break it. Correcting such error patterns is possible using e.g. the decoding method for square-free Goppa codes proposed in [BLM10]. The design minimum distance is at least $t + 1$ when differences between codewords are allowed to assume any pattern, but codewords that differ by patterns where all magnitudes are equal are much more sparse than that; the distribution is much more similar to what holds for binary codes, since the difference patterns only fail to be binary because of the overall magnitude. The error correcting strategy described in [BLM10] (algorithm 1) benefits from this observation, which allows for the correction of t errors with high probability as long as all error magnitudes are equal. The entries on Table 1 describe codes suitable for McEliece or Niederreiter encryption [Nie86].

One can argue that minimizing keys may not be the best way to reduce bandwidth occupation. After all, usually one expects to exchange encrypted messages considerably more often than certified keys, so it pays to minimize the encryption overhead per message instead. This is particularly easy to achieve using the Niederreiter cryptosystem, as long as the adopted codes yield short syndromes. Table 2 lists suggestions for codes that satisfy these requirements, without incurring unduly long keys. One sees that the choice for short syndromes often implies longer codes for larger characteristics.

Table 3 describes codes suitable for Parallel-CFS digital signatures [Fin10, BCMN10]. The signature size is slightly smaller than the product of the the syndrome size by the number of parallel signatures, and signing times are $O(t!)$. Quasi-monoidic codes in larger characteristics yield either shorter keys and signatures than in the binary case, or else considerably shorter signing times due to smaller values of t.

7 Efficiency

We will show that for all groups A relevant to cryptography, the matrix-vector products involving A-adic matrices can be computed in $\widetilde{O}(N)$ operations with a multidimensional discrete Fourier transform. As we have seen in Corollary 3.3, all relevant groups have the form $A = \mathbb{Z}_p^d$. Recall that the ring of A-adic matrices over R is isomorphic to the monoid ring $R[A]$ (hence the name, 'monoidic' matrices and codes). In the following lemma, we show that this has the structure of a multivariate polynomial quotient ring.

Lemma 7.1. *Let R be a commutative ring, then $R[\mathbb{Z}_p^d] \cong R[x_1, \ldots, x_d] \,/ \langle x_1^p - 1, \ldots, x_d^p - 1\rangle$.*

Proof. Let $A = \mathbb{Z}_p^d$. Consider the following R-bases for the left and right ring respectively, left we have $[\chi_{(a_1,\ldots,a_d)}]_{a\in A}$ and right $[x_1^{a_1} \cdots x_k^{a_d}]_{a\in A}$, where a ranges through all d-tuples in A for each ring.

We define ψ for all basis elements of the left ring to be $\psi(\chi_{(a_1,\ldots,a_k)}) = x_1^{a_1} \cdots x_k^{a_k}$. This can be extended canonically to an R-module isomorphism on the whole ring. It only remains to check that ψ respects multiplication. It suffices to check this for the generators, so let $a, b \in A$ then

$$\begin{aligned}\psi(\chi_a) \cdot \psi(\chi_b) &= (x_1^{a_1} \cdots x_k^{a_d}) \cdot (x_1^{b_1} \cdots x_k^{b_d}) \bmod x_1^p - 1, \ldots, x_d^p - 1 \\ &= x_1^{a_1+b_1 \bmod p} \cdots x_d^{a_d+b_k \bmod p} \\ &= \psi(\chi_{(a_1+b_1 \bmod p, \ldots, a_d+b_d \bmod p)}) = \psi(\chi_a \cdot \chi_b)\end{aligned}$$

□ □

We propose to compute the polynomial products by means of the several size-p fast discrete Fourier transform (DFT). This requires that the ring we work over has an element ω of order p and its characteristic is not p. One way to achieve this, is to lift our field $\mathbb{F}_q$ into a ring R of characteristic 0 that has been extended with a primitive p-th root of unity. Now, we can perform the operation in R, and project the results back.

The DFT itself works like the Walsh-Hadamard transform in [MB09], except that the matrices describing the transformation and its inverse are H_d and H_d^{-1}, which are recursively defined as

$$H_1 = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \cdots & \omega^{p-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(p-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{p-1} & \omega^{2(p-1)} & \cdots & \omega^{(p-1)(p-1)} \end{pmatrix}, \quad H_1^{-1} = \frac{1}{p} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(p-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(p-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(p-1)} & \omega^{-2(p-1)} & \cdots & \omega^{-(p-1)(p-1)} \end{pmatrix},$$

$$H_k = H_1 \otimes H_{k-1}, \qquad H_k^{-1} = H_1^{-1} \otimes H_{k-1}^{-1},$$

where $\otimes$ is the Kronecker product.

Acknowledgements

We thank Rafael Dahmen for his invaluable advice; Christiane Peters for her always patient comments, and the anonymous reviewers for helping in the improvement of this paper.

References

[BBD08] D. J. Bernstein, J. Buchmann, and E. Dahmen, editors. *Post-Quantum Cryptography*. Springer-Verlag, 2008. On p. 1.

[BC07] M. Baldi and F. Chiaraluce. Cryptanalysis of a new instance of McEliece cryptosystem based on QC-LDPC code. In *IEEE International Symposium on Information Theory – ISIT'2007*, pages 2591–2595. IEEE, 2007. On p. 2.

[BCGO09] T. P. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani. Reducing key length of the McEliece cryptosystem. In B. Preneel, editor, *Progress in Cryptology – Africacrypt'2009*, volume 5580 of *Lecture Notes in Computer Science*, pages 77–97. Springer, 2009. On p. 2.

[BCMN10] P. S. L. M. Barreto, P.-L. Cayrel, R. Misoczki, and R. Niebuhr. Quasi-dyadic CFS signatures. In X. Lai, M. Yung, and D. Lin, editors, *Inscrypt*, volume 6584 of *Lecture Notes in Computer Science*, pages 336–349. Springer, 2010. On pp. 2, 9, and 11.

[BLM10] P. S. L. M. Barreto, R. Lindner, and R. Misoczki. Decoding square-free Goppa codes over $\mathbb{F}_p$. Cryptology ePrint Archive, Report 2010/372, 2010. `http://eprint.iacr.org/2010/372.pdf`. On pp. 3, 7, 11, and 15.

[BLP10] D. J. Bernstein, T. Lange, and C. Peters. Wild McEliece. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2010. On pp. 1, 3, and 7.

[BLP11] D. J. Bernstein, T. Lange, and C. Peters. Smaller decoding exponents: Ball-collision decoding. In P. Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011. On p. 10.

[CFS01] N. Courtois, M. Finiasz, and N. Sendrier. How to achieve a McEliece-based digital signature scheme. In C. Boyd, editor, *ASIACRYPT*, volume 2248 of *Lecture Notes in Computer Science*, pages 157–174. Springer, 2001. On pp. 1 and 8.

[Fin10] M. Finiasz. Parallel-CFS. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography – SAC 2010*, volume 6544 of *Lecture Notes in Computer Science*, pages 161–172. Springer, 2010. On pp. 8 and 11.

[FOPT10a] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic cryptanalysis of McEliece variants with compact keys. In H. Gilbert, editor, *Advances in Cryptology – Eurocrypt'2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer, 2010. On pp. 2 and 10.

[FOPT10b] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tilllich. Algebraic cryptanalysis of compact McEliece's variants – toward a complexity analysis. In *International Conference on Symbolic Computation and Cryptography – SCC'2010*, pages 45–56. 2010. On p. 10.

[FS09] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Advances in Cryptology – Asiacrypt 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009. On pp. 8 and 10.

[Gab05] P. Gaborit. Shorter keys for code based cryptography. In *International Workshop on Coding and Cryptography – WCC'2005*, pages 81–91. ACM Press, Bergen, Norway, 2005. On p. 2.

[GL10] V. Gauthier Umaña and G. Leander. Practical key recovery attacks on two McEliece variants. In C. Cid and J.-C. Faugère, editors, *International Conference on Symbolic Computation and Cryptography – SCC'2010*, pages 27–44. 2010. On pp. 2, 10, and 11.

[Lan02] S. Lang. *Algebra.* Springer-Verlag, revised third edition, 2002. On p. 4.

[MB09] R. Misoczki and P. S. L. M. Barreto. Compact McEliece keys from Goppa codes. In M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2009. On pp. 1, 2, 10, and 12.

[McE78] R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. *Deep Space Network Progress Report*, 44:114–116, January 1978. On pp. 1 and 8.

[MRS00] C. Monico, J. Rosenthal, and A. Shokrollahi. Using low density parity check codes in the McEliece cryptosystem. In *IEEE International Symposium on Information Theory – ISIT'2000*, page 215. IEEE, Sorrento, Italy, 2000. On p. 2.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*, volume 16. North-Holland Mathematical Library, 1977. On pp. 2 and 4.

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986. On p. 11.

[OTD10] A. Otmani, J.-P. Tillich, and L. Dallot. Cryptanalysis of two McEliece cryptosystems based on quasi-cyclic codes. *Mathematics in Computer Science*, 3(2):129–140, 2010. On p. 2.

[Per11] E. Persichetti. Compact McEliece keys based on quasi-dyadic Srivastava codes. Cryptology ePrint Archive, Report 2011/179, 2011. `http://eprint.iacr.org/2011/179.pdf`. On p. 2.

[Pet10] C. Peters. Information-set decoding for linear codes over $\mathbb{F}_q$. In N. Sendrier, editor, *PQCrypto*, volume 6061 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2010. On p. 2.

[Pet11] C. Peters. *Curves, Codes, and Cryptography.* Ph.D. thesis, Technische Universiteit Eindhoven, the Netherlands, 2011. `http://alexandria.tue.nl/extra2/711052.pdf`. On pp. 8 and 10.

[SS92] V. Sidelnikov and S. Shestakov. On the insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications*, 1(4):439–444, 1992. On p. 10.

[TZ75] K. K. Tzeng and K. Zimmermann. On extending Goppa codes to cyclic codes. *IEEE Transactions on Information Theory*, 21:712–716, 1975. On p. 3.

A An Exemplary Quasi-Monoidic Srivastava Code

For our example, let $p = 3, s = 1, m = 4, d = 3, t = 3$. We use the extension field $\mathbb{F}_{3^4} = \mathbb{F}_3[u]/\langle u^4+2u^3+2\rangle$, the group $A = \mathbb{Z}_3^3$ of size $N = p^d = 27$, with set of generators $b_1 = (1, 0, 0)$, $b_2 = (0, 1, 0)$, $b_3 = (0, 0, 1)$.

We randomly select the images of the $\mathbb{F}_3$-linearly dependent

$$h(0)^{-1} = u^3 + u^2 + u + 2, \qquad h(b_1)^{-1} = u^2 + 2u + 1,$$
$$h(b_2)^{-1} = u^3 + 2u^2 + u + 1, \qquad h(b_3)^{-1} = u^2 + 1.$$

We also select a shift $\omega = u^3 + 2u + 2$, compute $\beta = (2u^3 + u^2 + 1, u^3 + u^2 + u, u^2 + 2u + 2)$, and $\gamma = (\gamma_0, \dots, \gamma_8)$ with

$$\gamma_0 = (u^3 + 2u + 2, 1, 2u^3 + u), \qquad \gamma_1 = (u^3 + u^2 + 2u + 1, u^2, 2u^3 + u^2 + u + 2),$$
$$\gamma_2 = (u^3 + 2u^2 + 2u, 2u^2 + 2, 2u^3 + 2u^2 + u + 1), \qquad \gamma_3 = (u + 1, 2u^3 + 2u, u^3 + 2),$$
$$\gamma_4 = (u^2 + u, 2u^3 + u^2 + 2u + 2, u^3 + u^2 + 1), \qquad \gamma_5 = (2u^2 + u + 2, 2u^3 + 2u^2 + 2u + 1, u^3 + 2u^2),$$
$$\gamma_6 = (2u^3, u^3 + u + 2, 2u + 1), \qquad \gamma_7 = (2u^3 + u^2 + 2, u^3 + u^2 + u + 1, u^2 + 2u),$$
$$\gamma_8 = (2u^3 + 2u^2 + 1, u^3 + 2u^2 + u, 2u^2 + 2u + 2).$$

where the group indices are ordered $0 = (0, 0, 0), a_1 = (1, 0, 0), a_2 = (2, 0, 0), \dots, a_{p^d-1} = (2, 2, 2)$. Our blocksize is $\mathtt{b} = \gcd(t, N) = 3$ and we randomly choose the permutation $\tau = \binom{012345678}{567834012}$. We use only the first $\ell = 6$ blocks chosen by the permutation, i.e., blocks $5, 6, 7, 8, 3, 4$, resulting in a code of length $n = \mathtt{b}\ell = 18$.

We continue and select the support permutations

$$\pi_0 = 0, \qquad \pi_1 = 2, \qquad \pi_2 = 1, \qquad \pi_3 = 2, \qquad \pi_4 = 0, \qquad \pi_5 = 1.$$

corresponding to the monoidic permutation matrices $M(\chi_{a_{\pi_i}})$, where

$$M(\chi_{a_0}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad M(\chi_{a_1}) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \qquad M(\chi_{a_2}) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

We compute

$$\widehat{\gamma}_0 = (2u^2 + u + 2, 2u^3 + 2u^2 + 2u + 1, u^3 + 2u^2), \qquad \widehat{\gamma}_1 = (u^3 + u + 2, 2u + 1, 2u^3),$$
$$\widehat{\gamma}_2 = (u^2 + 2u, 2u^3 + u^2 + 2, u^3 + u^2 + u + 1), \qquad \widehat{\gamma}_3 = (u^3 + 2u^2 + u, 2u^2 + 2u + 2, 2u^3 + 2u^2 + 1),$$
$$\widehat{\gamma}_4 = (u + 1, 2u^3 + 2u, u^3 + 2), \qquad \widehat{\gamma}_5 = (u^3 + u^2 + 1, u^2 + u, 2u^3 + u^2 + 2u + 2).$$

Afterwards, we have to set the scaling factors σ and to compute the layered parity-check matrix from the sequence β and $\widehat{\gamma}$. Since we would like to end up with a Goppa code (to be able to use the superior error-correction capabilities of "equal magnitude" decoding described in Appendix B), we will set all $\sigma_i = 1$ and the Cauchy layered exponent to be $r = 1$.

Finally, using the QMTrace step, we can produce the subfield subcode

$$H = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc}
2&0&2&1&1&1&0&2&1&1&1&0&2&1&0&2&0&0\\
2&2&0&1&1&1&1&0&2&0&1&1&0&2&1&0&2&0\\
0&2&2&1&1&1&2&1&0&1&0&1&1&0&2&0&0&2\\ \hline
2&0&2&0&2&2&2&0&2&0&1&1&2&1&0&1&1&0\\
2&2&0&2&0&2&2&2&0&1&0&1&0&2&1&0&1&1\\
0&2&2&2&2&0&0&2&2&1&1&0&1&0&2&1&0&1\\ \hline
2&2&1&2&1&2&0&0&2&1&0&2&1&1&1&0&1&0\\
1&2&2&2&2&1&2&0&0&2&1&0&1&1&1&0&0&1\\
2&1&2&1&2&2&0&2&0&0&2&1&1&1&1&1&0&0\\ \hline
0&0&1&1&0&1&2&0&2&0&0&0&0&1&1&0&0&1\\
1&0&0&1&1&0&2&2&0&0&0&0&1&0&1&1&0&0\\
0&1&0&0&1&1&0&2&2&0&0&0&1&1&0&0&1&0
\end{array}\right), \qquad
H_{\text{sys}} = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc}
\mathbf{2}&1&0&\mathbf{2}&1&1&1&0&0&0&0&0&0&0&0&0&0&0\\
\mathbf{0}&2&1&\mathbf{1}&2&1&0&1&0&0&0&0&0&0&0&0&0&0\\
\mathbf{1}&0&2&\mathbf{1}&1&2&0&0&1&0&0&0&0&0&0&0&0&0\\ \hline
\mathbf{0}&0&0&\mathbf{1}&2&0&0&0&0&1&0&0&0&0&0&0&0&0\\
\mathbf{0}&0&0&\mathbf{0}&1&2&0&0&0&0&1&0&0&0&0&0&0&0\\
\mathbf{0}&0&0&\mathbf{2}&0&1&0&0&0&0&0&1&0&0&0&0&0&0\\ \hline
\mathbf{0}&2&2&\mathbf{1}&2&2&0&0&0&0&0&0&1&0&0&0&0&0\\
\mathbf{2}&0&2&\mathbf{2}&1&2&0&0&0&0&0&0&0&1&0&0&0&0\\
\mathbf{2}&2&0&\mathbf{2}&2&1&0&0&0&0&0&0&0&0&1&0&0&0\\ \hline
\mathbf{1}&2&2&\mathbf{1}&0&2&0&0&0&0&0&0&0&0&0&1&0&0\\
\mathbf{2}&1&2&\mathbf{2}&1&0&0&0&0&0&0&0&0&0&0&0&1&0\\
\mathbf{2}&2&1&\mathbf{0}&2&1&0&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right).$$

The systematic form H_{sys} can be computed by inverting the matrix consisting of the last trm columns. From the systematic parity-check matrix, QMSignature extracts those entries marked in boldface, which are sufficient to describe the public generator matrix

$$G_{\text{sys}} = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc}
1 & 0 & 0 & 0 & 0 & 0 & \mathbf{2} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{2} & \mathbf{1} & \mathbf{2} & \mathbf{2} \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 2 & 2 & 1 & 2 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 2 & 2 & 0 & 2 & 2 & 1 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 & \mathbf{2} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{2} & \mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 2 & 1 & 2 & 1 & 0 & 2 & 1 & 2 & 0 & 1 & 2 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 2 & 0 & 2 & 1 & 2 & 2 & 1 & 2 & 0 & 1
\end{array}\right).$$

Note that for the parity-check matrix, the signature of each monoidic block is its first column, but for the generator, which contains transposed monoidic blocks, the signature is the first row.

B Decoding Square-Free Goppa Codes

For codes with degree t and its average distance at least $(4/p)t + 1$, the proposed decoder can uniquely correct $(2/p)t$ errors, with high probability. The correction capability is higher if the distribution of error magnitudes is not uniform, approaching or reaching t errors when any particular error value occurs much more often than others or exclusively. The parity-check matrix used by this algorithm is in the form (2.1).

At some point of this algorithm, we will call the WeakPopovForm algorithm (also present in [BLM10] and described below) to find the short vectors in the lattice spanned by the rows of

$$A = \begin{bmatrix} g & 0 & 0 & \dots & 0 \\ -v_1 & 1 & 0 & \dots & 0 \\ -v_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -v_{p-1} & 0 & 0 & \dots & 1 \end{bmatrix}, \tag{B.1}$$

Where g is the Goppa polynomial and the v_i's values will be computed through the execution of Algorithm 1.

Algorithm 1 Decoding p-ary square-free Goppa codes

```
INPUT: $\Gamma(L, g)$, a Goppa code over $\mathbb{F}_p$ where $g$ is square-free.
INPUT: $H \in \mathbb{F}_q^{r \times n}$, a parity-check matrix in the form of Equation 2.1.
INPUT: $c' = c + e \in \mathbb{F}_p^n$, the received codeword with errors.
OUTPUT: set of corrected codeword $c \in \Gamma(L, g)$ ($\varnothing$ upon failure).
 1: $s^{\mathrm{T}} \leftarrow Hc'^{\mathrm{T}} \in \mathbb{F}_q^n$, $s_e(x) \leftarrow \sum_i s_i x^i$. ▷ N.B. $Hc'^{\mathrm{T}} = He^{\mathrm{T}}$.
 2: if $\nexists\, s_e^{-1}(x) \bmod g(x)$ then
 3:     return $\varnothing$ ▷ $g(x)$ is composite
 4: end if
 5: $S \leftarrow \varnothing$
 6: for $\phi \leftarrow 1$ to $p - 1$ do ▷ guess the correct scale factor $\phi$
 7:     for $k \leftarrow 1$ to $p - 1$ do
 8:         $u_k(x) \leftarrow x^k + \phi k x^{k-1}/s_e(x) \mod g(x)$
 9:         if $\nexists \sqrt[p]{u_k(x)} \bmod g(x)$ then
10:             try next $\phi$ ▷ $g(x)$ is composite
11:         end if
12:         $v_k(x) \leftarrow \sqrt[p]{u_k(x)} \bmod g(x)$
13:     end for
14:     Build the lattice basis $A$ defined by Equation B.1.
15:     Apply WEAKPOPOVFORM (Algorithm 2) to reduce the basis of $\Lambda(A)$.
16:     for $i \leftarrow 1$ to $p$ do
17:         $a \leftarrow A_i$ ▷ with $a_j$ indices in range $0 \dots p - 1$
18:         for $j \leftarrow 0$ to $p - 1$ do
19:             if $\deg(a_j) > \lfloor (t - j)/p \rfloor$ then
20:                 try next $i$ ▷ not a solution
21:             end if
22:         end for
23:         $\sigma(x) \leftarrow \sum_j x^j a_j(x)^p$
24:         Compute the set $J$ such that $\sigma(L_j) = 0$, $\forall j \in J$.
25:         for $j \in J$ do
26:             Compute the multiplicity $\mu_j$ of $L_j$.
27:             $e_j \leftarrow \phi \mu_j$
28:         end for
29:         if $He^{\mathrm{T}} = s^{\mathrm{T}}$ then
30:             $S \leftarrow S \cup \{c' - e\}$
31:         end if
32:     end for
33:     return $S$
34: end for
```

Algorithm 2 (WeakPopovForm) Computing the weak Popov form

```
Input: $A \in \mathbb{F}_q[x]^{p \times p}$ in the form of Equation B.1.
Output: weak Popov form of $A$.
1: ▷ Compute $I^A$:
2: for $j \leftarrow 1$ to $p$ do
3:     $I^A_j \leftarrow$ if $\deg(A_{j,1}) > 0$ then $1$ else $j$
4: end for
5: ▷ Put $A$ in weak Popov form:
6: while $\mathrm{rep}(I^A) > 1$ do
7:     ▷ Find suitable $k$ and $\ell$ to apply simple transform of first kind:
8:     for $k \leftarrow 1$ to $p$ such that $I^A_k \neq 0$ do
9:         for $\ell \leftarrow 1$ to $p$ such that $\ell \neq k$ do
10:            while $\deg(A_{\ell,I^A_k}) \geqslant \deg(A_{k,I^A_k})$ do
11:                $c \leftarrow \mathrm{lead}(A_{\ell,I^A_k}) / \mathrm{lead}(A_{k,I^A_k})$
12:                $e \leftarrow \deg(A_{\ell,I^A_k}) - \deg(A_{k,I^A_k})$
13:                $A_\ell \leftarrow A_\ell - cx^e A_k$
14:            end while
15:            ▷ Update $I^A_\ell$ and hence $\mathrm{rep}(I^A)$ if necessary:
16:            $d \leftarrow \max\{\deg(A_{\ell,j}) \mid j = 1, \ldots, p\}$
17:            $I^A_\ell \leftarrow \max\{j \mid \deg(A_{\ell,j}) = d\}$
18:        end for
19:    end for
20: end while
21: return $A$
```