Monoidic Codes in Cryptography

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

July 11, 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 keysizes. 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 Berstein, 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, classical public-key cryptography was shown to be subject 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 only their generator matrix in systematic 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 keysizes 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], Berstein, 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 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 a new class of quasi-dyadic codes, whose members have very compact descriptions,

*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.

‡Escola Politécnica, Universidade de São Paulo, Brasil. E-Mail: rmisoczki@larc.usp.br.

and 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.

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.

We show that this new class of codes is very useful in practice. For example, we find that many wild Goppa codes are in fact quasi-monoidic. Using quasi-monoidic Goppa codes for the McEliece cryptosystems and CFS signature scheme, one can achieve smaller keysizes than before, as exemplified by Tables 1 and 3 in Section 4.

Organization. In Section 2, 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 und signature schemes with this family in Section 3. Afterwards, in Section 4 we give preliminary parameters and discuss their security against known attacks. Finally, in Section 5 we briefly argue why the matrix-vector products for quasi-monoidic matrices can be computed efficiently using a discrete Fourier transform.

2 Quasi-Monoidic Codes

We start with the definitions of monoidic and Cauchy (power) matrices. 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 $g(x)$. 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. Afterwards, we show how to construct random monoidic Cauchy matrices algorithmically. In the final segment, we put everything together and construct quasi-monoidic Goppa codes.

Monoidic matrices.

Definition 2.1. Let R be a commutative ring, $A = \{g_0, \cdots, g_{N-1}\}$ a finite abelian group of size $|A| = N$ with neutral element $g_0 = 0$, and $\alpha\colon A \longrightarrow R$ a sequence indexed by A. The A-adic matrix $M(\alpha)$ associated with this sequence is one for which $M_{i,j} = \alpha(g_i - g_j)$ holds, i.e.,

$$M = \begin{pmatrix} \alpha(0) & \alpha(-g_1) & \cdots & \alpha(-g_{N-1}) \\ \alpha(g_1) & \alpha(0) & \cdots & \alpha(g_1 - g_{N-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha(g_{N-1}) & \alpha(g_{N-1} - g_1) & \cdots & \alpha(0) \end{pmatrix}.$$

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 monoids, in which case one would 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.

Cauchy matrices.

Definition 2.2. Let $\mathbb{F}$ be a finite field, and $\beta = (\beta_0, \beta_1 \ldots, \beta_{t-1}), \gamma = (\gamma_0, \gamma_1, \ldots, \gamma_{n-1})$ 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 number $r > 0$, 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 Berstein, 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 capablities surpassing general alternant codes and make them particularly interesting for various application including cryptography.

Conditions for which monoidic implies Cauchy.

Theorem 2.3. *Let $M(\alpha)$ be A-adic for a sequence α of length N over $\mathbb{F}$. Then M is Cauchy iff*

(1) *$\alpha(g_i)$ are distinct and invertible in $\mathbb{F}$* *for all $0 \leq i < N$, and*

(2) *$(\alpha(g_i - g_j))^{-1} = (\alpha(g_i))^{-1} + (\alpha(-g_j))^{-1} - (\alpha(0))^{-1}$* *for all $0 \leq i, j < N$.*

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

Proof. We start by showing that our conditions indeed imply that M is Cauchy. Since all elements of α are distinct, so are those of β and γ. For the disjointness, assume that there are indices i and j, such that $\beta(g_i) = \gamma(g_j)$. In this case we get $0 = \beta(g_i) - \gamma(g_j) = 1/\alpha(g_i - g_j)$, which is a contradiction. Finally we compare the matrices $M(\alpha)$ and $C(\beta, \gamma)$ resulting in the equality

$$M_{i,j} = \alpha(g_i - g_j) = 1/(1/\alpha(g_i) + 1/\alpha(-g_j) - 1/\alpha(0)) = 1/(\beta(g_i) - \gamma(g_j)) = C_{i,j}.$$

We continue by showing that if M is Cauchy, i.e., $M(\alpha) = C(\beta', \gamma')$, 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 $\alpha(g_i) = 1/\beta'(g_i)$. By the properties of β' this gives us condition (1), i.e., that all $\alpha(g_i)$ are distinct and invertible, as well as $\beta' = \beta$. We use similarly that $M_{0,i} = C_{0,i}$ which implies $\alpha(-g_i) = 1/(\beta(0) - \gamma'(i))$. Solving for γ' reveals that it equals γ. Since $\beta = \beta'$ and $\gamma = \gamma'$, we get that $M(\alpha) = C(\beta, \gamma)$ implying condition (2). □

Note that if the A-adic matrix of a sequence α is also Cauchy, then the sequence of r-th powers, i.e., $\alpha^r = (\alpha_0^r, \alpha_{g_1}^r, \dots, \alpha_{g_{n-1}}^r)$ yields the corresponding Cauchy power matrix. In other words, for any number $r > 0$ we have $M(\alpha) = C(\beta, \gamma) \implies M(\alpha^r) = C(\beta, \gamma, r)$.

Construction of monoidic Cauchy matrices.

Corollary 2.4. *Let A be a finite, abelian group with basis $b_1, \ldots, b_d$ and $M(\alpha)$ be A-adic and Cauchy for a sequence α over $\mathbb{F}$, then for all $c_1, \ldots, c_d \in \mathbb{Z}$,*

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

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

Proof. By Theorem 2.3, we know that for all $g, g' \in A$ the following holds

$$(\alpha(g + g'))^{-1} = (\alpha(g))^{-1} + (\alpha(g'))^{-1} - (\alpha(0))^{-1}.$$

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

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

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

$$\begin{aligned}
\alpha(0)^{-1} = \alpha(kg)^{-1} &= k\alpha(g)^{-1} - (k-1)\alpha(0)^{-1} \\
k(\alpha(0)^{-1} - \alpha(g)^{-1}) &= 0
\end{aligned}$$

Since g is not the neutral element, all elements of α 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 $b_1, \ldots, b_d$ be group elements that form an $\mathbb{F}_p$ basis, then the sequence elements $\alpha(0), \alpha(b_1), \ldots, \alpha(b_d)$ completely determine the sequence. We call these values the essence of the sequence α.

For example, if $A = \mathbb{F}_p^d$, then such a basis $b_1, \ldots, b_d$ is given by the generators of the d distinct copies of $\mathbb{F}_p$ in A. For a given basis, we can sample a monoidic sequence uniformly at random with the algorithm in Figure 1.

We will briefly argue why the algorithm in Figure 1 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

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

Now, not all coefficients of $\alpha(0), \alpha(b_1), \ldots, \alpha(b_d)$ 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 $\alpha(b_i)$ are chosen, no such dependency can exist.

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

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

$\textsc{MonoidCauchy}(p, Q, d)$:

$F \longleftarrow \mathbb{F}_Q \setminus \{0\}$
$\alpha(0) \longleftarrow U(F)$
For $i = 1, \ldots, d$:
$F \longleftarrow \mathbb{F}_Q \setminus (\mathbb{F}_p\,\alpha(0) + \mathbb{F}_p\,\alpha(b_1) + \cdots + \mathbb{F}_p\,\alpha(b_{i-1}))$
$\alpha(b_i) \longleftarrow U(F)$
For $c_1, \ldots, c_d \in \mathbb{F}_p$:
$\alpha(c_1 b_1 + \cdots + c_d b_d) \longleftarrow c_1\alpha(b_1) + \cdots + c_d\alpha(b_d) - (c_1 + \cdots + c_d - 1)\alpha(0)$
Output $(\alpha(0)^{-1}, \alpha(g_1)^{-1}, \ldots, \alpha(g_{p^d-1})^{-1})$

Figure 1: Choosing A-adic Cauchy sequences, where $A = \{0, g_1, \ldots, g_{p^d-1}\}$ has basis $b_1, \ldots, b_d$.

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

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

Figure 2: 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).

$\textsc{QuasiMonoidic}(\ldots)$:

$\alpha \longleftarrow \textsc{MonoidCauchy}(p, Q, d)$; $\omega \longleftarrow U(\mathbb{F}_Q)$
For $i = 0, \ldots, t-1$: $\beta_i \longleftarrow (\alpha(g_i))^{-1} + \omega$ — "Goppa roots"
For $i = 0, \ldots, N-1$: $\gamma_i \longleftarrow (\alpha(0))^{-1} - (\alpha(-g_i))^{-1} + \omega$ — "Goppa support"
$\tau \longleftarrow U(S_{N/b})$ — "Block permutation"
$\pi_0, \ldots, \pi_{\ell-1} \longleftarrow U(\{0, \ldots, b-1\})$ — "Support permutations"
$\sigma_0, \ldots, \sigma_{\ell-1} \longleftarrow U(\mathbb{F}_q^*)$ — "Scaling"
For $i = 0, \ldots, \ell-1$: $\widehat{\gamma}_i \longleftarrow (\gamma_{\tau(i)b}, \ldots, \gamma_{\tau(i)b+b-1})$ — "Select blocks"
For $i = 0, \ldots, \ell-1$: $\widehat{\gamma}_i \longleftarrow \widehat{\gamma}_i M(\chi_{g_{\pi_i}})$ — "Permute support"
$H \longleftarrow [CL(\beta, \widehat{\gamma}_0, r)\sigma_0 \mid \cdots \mid CL(\beta, \widehat{\gamma}_{\ell-1}, r)\sigma_{\ell-1}]$ — "Parity-check matrix"
$H \longleftarrow \mathrm{QMT}\textsc{race}(q, b, H)$
$H \longleftarrow \mathrm{QMG}\textsc{auss}(b, H)$
$H \longleftarrow \mathrm{QMS}\textsc{ignature}(b, H)$
Output private $\beta, \widehat{\gamma}_0, \ldots, \widehat{\gamma}_{\ell-1}, \sigma_0, \ldots, \sigma_{\ell-1}$; public H

Figure 3: Choosing quasi-monoidic GS codes with private and public description. Here, $S_{N/b}$ is the group of permutations on $\{0, \ldots, N/b-1\}$ and $\chi_{g_{\pi_i}}$ is the characteristic function of the group element g_{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 2 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 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 $M(\chi_{g_\pi})$ 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 $tr/b \times \ell$ monoidic blocks of size 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. 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 should 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.

The first subroutine QMTrace will generate an 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 factor 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. 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 b-th column. This concludes our description of the algorithm.

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

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, so long as they don't end up in the submatrix we actually use.

This relaxation is realized by omitting line 4 of MonoidCauchy (Fig. 1), 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 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.

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.

3 Monoidic Encryption and Signatures

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

3.1 McEliece encryption scheme

Key Generation: Let λ be the security level wanted and 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 the Figure 3. Let $k = n - mt$. The choice of parameters is guided so that the cost to decode a $[n, k, 2t+1]$-code be at least 2^λ steps. 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 $c - e$.

3.2 Parallel-CFS

To sign a document with the standard CFS signature schemes we should hash the document into a syndrome and then decoding 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. However, an unpublished attack by D. Bleichenbacher and described in [FS09] showed 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 h) from a document D and signing it, one can produce i hashes (using i different functions $h_1, \ldots, h_i$) and sign all $h_1(D), \ldots, h_i(D)$ 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 Goppa code $\Gamma(g, L)$, where g is th Goppa 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 $s_{j,i} = h_i(D) + H \times \phi_\delta(j)^T$. Once a decodable syndrome is found for an $j_{0,i}$, then there exists a plain text $p'_{j_0,i}$, such that $H \times \phi_t(p'_{j_0,i})^T = s_{j_0,i} = h_i(D) + H \times \phi_\delta(j_0)^T$.

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 $H \times e_i^T = h_i(D)$, 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 $H \times \phi_{t+\delta}(p_i)^T \stackrel{?}{=} h_i(D)$.

4 Parameters of Cryptographic Interest

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

In estimating *concrete* security (rather than asymptotic behavior only), we adopt the following criteria, which were discussed and analyzed by Finiasz e Sendrier [FS09] and by Peters [Pet11, Observation 6.9], whereby directly decoding a code of length n, dimension k, and generic error patterns of weight w (which may or may not match the design number of errors t) over $\mathbb{F}_q$, without using the trapdoor, has a *workfactor* at least WF_q measured in bit operations. Typically $\wp \approx w/2$ e $\ell \gtrsim \log_q \binom{k/2}{\wp} + \wp \log(q-1)$:

$$\mathrm{WF}_2 = \min_{\wp,\ell} \left\{ \begin{array}{c} \frac{1}{2}(n-k)^2(n+k) + \left(k/2 - \wp + 1 + \binom{\lfloor k/2 \rfloor}{\wp} + \binom{\lceil k/2 \rceil}{\wp}\right)\ell \\ +(w - 2\wp + 1)4\wp \binom{\lfloor k/2 \rfloor}{\wp}\binom{\lceil k/2 \rceil}{\wp} \end{array} \right\} \tag{4.1}$$

$$\mathrm{WF}_q = \min_{\wp,\ell} \left\{ \begin{array}{c} (n-k)^2(n+k) + \left(k/2 - \wp + 1 + \left(\binom{\lfloor k/2 \rfloor}{\wp} + \binom{\lceil k/2 \rceil}{\wp}\right)(q-1)^\wp\right)\ell \\ +\dfrac{q}{q-1}(w - 2\wp + 1)2\wp\left(1 + \dfrac{q-2}{q-1}\right)\dfrac{\binom{\lfloor k/2 \rfloor}{\wp}\binom{\lceil k/2 \rceil}{\wp}(q-1)^{2\wp}}{q^\ell} \end{array} \right\} \tag{4.2}$$

When it is known beforehand that all errors have equal magnitude and $q > 2$, we simplify Equation 4.2 accordingly:

$$\mathrm{WF}'_q = \min_{\wp,\ell} \left\{ \begin{array}{c} (n-k)^2(n+k) + \left(k/2 - \wp + 1 + \left(\binom{\lfloor k/2 \rfloor}{\wp} + \binom{\lceil k/2 \rceil}{\wp}\right)(q-1)\right)\ell \\ +\dfrac{q}{q-1}(w - 2\wp + 1)2\wp\left(1 + \dfrac{q-2}{q-1}\right)\dfrac{\binom{\lfloor k/2 \rfloor}{\wp}\binom{\lceil k/2 \rceil}{\wp}(q-1)^{2}}{q^\ell} \end{array} \right\} \tag{4.3}$$

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 up to t errors of equal magnitude, not necessarily general errors. This is possible using e.g. the decoding method for square-free Goppa codes proposed in [BLM10].

The entries on Table 1 describe codes suitable for McEliece or Niederreiter encryption. To the best of our knowledge, structural attacks (based on converting the public code into a multivariate nonlinear system and then trying to solve it with Gröbner basis techniques) are as ineffective against quasi-monoidic codes in characteristic $p > 2$ as they are against the binary case (i.e. against binary quasi-dyadic codes). Nevertheless, even if they were successful, the Goppa polynomial would have to be interpolated from the alternant trapdoor extracted from the public code, since otherwise most of the errors would not be correctable in general. However, this attack only retrieves n/t variables y_i and n variables x_i such that $y_i = g(x_i)^{-1}$. One way to protect against interpolation of the monic Goppa polynomial g of degree t is to impose $t > n/t$, i.e. $t^2 > n$. The codes suggested on Table 1 satisfy this condition, making interpolation inviable.

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 (including protection against structural interpolation attacks), 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.

Table 1: Encryption quasi-monoidic codes.

level	p	m	n	k	t	key	syndrome
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	239	3	956	239	239	5665	5665
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	syndrome
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

Table 3: Parallel CFS quasi-monoidic codes.

level	p	m	n	k	t	key (bits / KiB)	sigs	δ	sig bits
82.5	2	15	32580	32400	12	1458000 / 178	2	4	326
79.5	3	11	177048	176949	9	3085033 / 377	3	2	375
89.8	13	4	28509	28457	13	421214 / 52	2	4	342
114.4	2	20	1048332	1048092	12	62885520 / 7677	3	3	636
114.8	11	6	1771495	1771429	11	36768825 / 4489	3	2	558
118.6	13	5	371228	371163	13	6867332 / 839	3	3	624
132.6	2	23	8388324	8388048	12	578775312 / 70652	3	2	684
131.4	5	8	390495	390375	15	21754145 / 2656	3	4	759
131.6	13	6	4826731	4826653	13	107164431 / 13082	2	3	514

5 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 2.4, 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 $R[A]$. In the following lemma, we show that this has the structure of a multivariate polynomial quotient ring.

Lemma 5.1. *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-modul isomorphism on the whole ring. It only remains to check that ψ respects multiplication. It suffices to check for the basis, 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.

References

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

[BCMN10] P. S. L. M. Barreto, P.-L. Cayrel, R. Misoczki, and R. Niebuhr. Quasi-dyadic CFS signatures. In *Inscrypt 2010*, LNCS. Springer, 2010. On pp. 1, 2, and 4.

[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/`. On pp. 2, 2, 4, and B.

[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. ISBN 978-3-642-19573-0. On pp. 1, 2, and 2.

[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. ISBN 3-540-42987-5. On p. 1.

[Fin10] M. Finiasz. Parallel-CFS. In *Selected Areas in Cryptography – SAC 2010*, volume 6544 of *Lecture Notes in Computer Science*, pages 161–172. Springer, Waterloo, Canada, 2010. On pp. 3.2 and 4.

[FS09] M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In *Advances in Cryptology – Asiacrypt 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, Tōkyō, Japan, 2009. On pp. 3.2 and 4.

[Lan02] S. Lang. *Algebra. Revised Third Edition.* Springer-Verlag, 2002. On p. 2.

[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. ISBN 978-3-642-05443-3. On pp. 1 and 5.

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

[Pet11] C. Peters. *Information-Set Decoding for Linear Codes over* $\mathbb{F}_q$. Ph.D. thesis, Technische Universiteit Eindhoven, 2011. On p. 4.

[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. 2.

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 $\mathbb{F}_{3^3} = \mathbb{F}_3[u]/\langle u^4+2u^3+2\rangle$, the group $A = \mathbb{Z}_3^3$ of size $N = p^d = 27$, with basis $b_1 = (1,0,0)$, $b_2 = (0,1,0)$, $b_3 = (0,0,1)$.

We randomly select the $\mathbb{F}_3$-linearly independent

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

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 $\gamma = (\gamma_0, \ldots, \gamma_8)$ 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 $0 = (0,0,0), g_1 = (1,0,0), g_2 = (2,0,0), \ldots, g_{p^d-1} = (2,2,2)$. Our blocksize is $b = \gcd(t, N) = 3$ 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 $n = b\ell = 18$.

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 $M(\chi_{g_{\pi_i}})$, where

$$M(\chi_{g_0}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad M(\chi_{g_1}) = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \qquad M(\chi_{g_2}) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

We compute

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

Afterwards, we set all scaling factors to $\sigma_i = 1$, so we can be sure to end up with a Goppa code and will be able to use the superior error-correction capabilities of "equal magnitude" decoding described in Appendix B. Now, we have everything the layered parity-check matrix and compute the corresponding H for the subfield subcode via QMTRACE

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

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

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

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 up to $(2/p)t$ 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.

This algorithm was proposed for Goppa codes, which can always be described by a parity-check in the form:

$$\begin{aligned} H \;=\; & \operatorname{toep}(g_1, \dots, g_t) \\ \cdot\; & \operatorname{vdm}_t(L_0, \dots L_{n-1}) \\ \cdot\; & \operatorname{diag}(g(L_0)^{-1}, \dots, g(L_{n-1})^{-1}) \end{aligned} \tag{B.1}$$

Where $\operatorname{toep}(g_1, \dots, g_t)$ is a $t \times t$ Toeplitz matrix (defined by $T_{ij} := g_{t-i+j}$ for $j \leqslant i$ and $T_{ij} := 0$ otherwise), $\operatorname{vdm}_t(L_0, \dots L_{n-1})$ is a $t \times n$ Vandermonde matrix (defined by $V_{ij} := L_j^i$, $0 \leqslant i < t$, $0 \leqslant j < n$), and $\operatorname{diag}(g(L_0)^{-1}, \dots, g(L_{n-1})^{-1})$ is an $n \times n$ diagonal matrix.

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

$$A = \begin{bmatrix} g & 0 & 0 & \dots & 0 \\ -v_1 & 1 & 0 & \dots & 0 \\ -v_2 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -v_{p-1} & 0 & 0 & \dots & 1 \end{bmatrix}, \tag{B.2}$$

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

$s^{\mathrm{T}} \leftarrow Hc'^{\mathrm{T}} \in \mathbb{F}_q^n$, $s_e(x) \leftarrow \sum_i s_i x^i$. $\triangleright$ N.B. $Hc'^{\mathrm{T}} = He^{\mathrm{T}}$.
if $\nexists\, s_e^{-1}(x) \bmod g(x)$ **then**
  return $\varnothing$ $\triangleright$ $g(x)$ is composite
end if
$S \leftarrow \varnothing$
for $\phi \leftarrow 1$ **to** $p-1$ **do** $\triangleright$ guess the correct scale factor ϕ
  for $k \leftarrow 1$ **to** $p-1$ **do**
    $u_k(x) \leftarrow x^k + \phi k x^{k-1}/s_e(x) \mod g(x)$
    if $\nexists \sqrt[p]{u_k(x)} \bmod g(x)$ **then**
      try next ϕ $\triangleright$ $g(x)$ is composite
    end if
    $v_k(x) \leftarrow \sqrt[p]{u_k(x)} \bmod g(x)$
  end for
  Build the lattice basis A defined by Equation B.2.
  Apply WEAKPOPOVFORM (Algorithm 2) to reduce the basis of $\Lambda(A)$.
  for $i \leftarrow 1$ **to** p **do**
    $a \leftarrow A_i$ $\triangleright$ with a_j indices in range $0 \dots p-1$
    for $j \leftarrow 0$ **to** $p-1$ **do**
      if $\deg(a_j) > \lfloor (t-j)/p \rfloor$ **then**
        try next i $\triangleright$ not a solution
      end if
    end for
    $\sigma(x) \leftarrow \sum_j x^j a_j(x)^p$
    Compute the set J such that $\sigma(L_j) = 0$, $\forall j \in J$.
    for $j \in J$ **do**
      Compute the multiplicity μ_j of L_j.
      $e_j \leftarrow \phi \mu_j$
    end for
    if $He^{\mathrm{T}} = s^{\mathrm{T}}$ **then**
      $S \leftarrow S \cup \{c' - e\}$
    end if
  end for
  return S
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.2.
Output: weak Popov form of A.

▷ Compute I^A:
for $j \leftarrow 1$ **to** p **do**
    $I^A_j \leftarrow$ **if** $\deg(A_{j,1}) > 0$ **then** 1 **else** j
end for
▷ Put A in weak Popov form:
while $\mathrm{rep}(I^A) > 1$ **do**
    ▷ Find suitable k and ℓ to apply simple transform of first kind:
    for $k \leftarrow 1$ **to** p **such that** $I^A_k \neq 0$ **do**
        for $\ell \leftarrow 1$ **to** p **such that** $\ell \neq k$ **do**
            while $\deg(A_{\ell, I^A_k}) \geqslant \deg(A_{k, I^A_k})$ **do**
                $c \leftarrow \mathrm{lead}(A_{\ell, I^A_k}) / \mathrm{lead}(A_{k, I^A_k})$
                $e \leftarrow \deg(A_{\ell, I^A_k}) - \deg(A_{k, I^A_k})$
                $A_\ell \leftarrow A_\ell - c x^e A_k$
            end while
            ▷ Update I^A_ℓ and hence $\mathrm{rep}(I^A)$ if necessary:
            $d \leftarrow \max\{\deg(A_{\ell,j}) \mid j = 1, \dots, p\}$
            $I^A_\ell \leftarrow \max\{j \mid \deg(A_{\ell,j}) = d\}$
        end for
    end for
end while
return A