Efficient Implementation of a CCA2-secure Variant of McEliece Using Generalized Srivastava Codes

Pierre-Louis Cayrel[1], Gerhard Hoffmann[2], and Edoardo Persichetti[3]

[1] Université Jean Monnet, Saint-Etienne, France
[2] Technische Universität Darmstadt, Germany
[3] University of Auckland, New Zealand

Abstract. In this paper we present efficient implementations of McEliece variants using quasi-dyadic codes. We provide secure parameters for a classical McEliece encryption scheme based on quasi-dyadic generalized Srivastava codes, and successively convert our scheme to a CCA2-secure protocol in the random oracle model applying the Fujisaki-Okamoto transform. In contrast with all other CCA2-secure code-based cryptosystems that work in the random oracle model, our conversion does not require a constant weight encoding function. We present results for both 128-bit and 80-bit security level, and for the latter we also feature an implementation for an embedded device.

1 Introduction

The McEliece and Niederreiter public-key encryption schemes are based on error-correcting codes. One drawback are the large public keys. There have been few implementations reported; we cite for instance [29] and [30] for 32-bit software implementations. An alternative scheme, called HyMES (Hybrid McEliece cryptosystem), was implemented by Sendrier and Biswas [11], combining ideas from both the previous schemes.
Recently, implementations of the McEliece and Niederreiter cryptosystems for embedded devices have been presented, respectively by Eisenbarth et al. in [13] and by Heyse in [18], with the disadvantage of an external memory requirement for storing the key. A first proposal to deal with this issue from an implementational point of view is to make use of the quasi-dyadic variant of Misoczki and Barreto [25]. This was done by Heyse in [19], along with the extension to a CCA2-secure protocol. Unfortunately, the fields underlying the Goppa codes chosen are still too big to fit on the flash memory of the embedded device and this has repercussions in the speed of the implementation, since the use of tower field arithmetic becomes necessary.
In our paper, we provide an alternative construction based on the more general framework of generalized Srivastava codes described by Persichetti in [27]. We then convert the encryption scheme into a CCA2-secure protocol with the help of the Fujisaki-Okamoto transform [17]. To the best of our knowledge, a scheme

based on this family of codes has never been implemented before; moreover, we use McEliece with a twist, and we don't require any constant weight encoding function [32] for our conversion. This is also a novelty, and it allows to simplify the construction and save computational costs at the same time. The finite fields in use are much smaller than previous proposals, and fit completely on the flash memory, with the result that our implementation is much faster.
We note that there exist schemes, such as Dowsley et al. [12] and Freeman et al. [22], that provide CCA2-secure encryption based on coding theory in the standard model, but these schemes are completely impractical.
The paper is organized as follows: in Section 2 the McEliece and Niederreiter encryption schemes are introduced, along with an overview of constructions based on structured matrices. Security definitions such as IND-CCA2 and their instantiations are discussed in Section 3, and the technical details about the implementations with the respective timings are provided in Section 4, both for a C++ code, and for implementation on an embedded device. Finally, we conclude in Section 5.

2 Code-based public-key encryption schemes

2.1 The McEliece cryptosystem

The first cryptosystem based on coding theory was introduced in 1978 by Robert J. McEliece [23] and, for an appropriate choice of parameters, is still unbroken. In the original proposal, binary Goppa codes are used as a basis for the construction, and the security comes from the hardness of the General Decoding Problem (GDP).

Definition 1 (GDP). *Let C be an $[n, k]$ linear code over $\mathbb{F}_q$ and let y be a vector of $\mathbb{F}_q^n$.*
Find the codeword closest to y, i.e. find $c \in \mathsf{C}$ such that $d(c, y)$ is minimal.

This corresponds to correcting a certain number of errors occurred on the codeword c, represented by an error vector e, that is $y = c + e$. A unique solution exists if the weight of e is less than or equal to $w = \lfloor \frac{d-1}{2} \rfloor$, where d is the *minimum distance* of the code C.
This problem is well known and was proved to be NP-complete [7]. Moreover, GDP is believed to be hard on average, and not just on the worst-case instances. The general framework proceeds as follows:

Key Generation: Pick a $k \times n$ generator matrix G for a w-error correcting linear code with an efficient decoding algorithm over the finite field $\mathbb{F}_q$, a $k \times k$ invertible matrix S and an $n \times n$ permutation matrix P at random, then compute $G' = SGP$, which is another valid generator matrix. The private key consists of G, S, P, and the public key is G'.The system parameters n, k, w are also public.

Encryption: To encrypt a plaintext $x \in \mathbb{F}_q^k$, compute the corresponding codeword xG' and add a random error vector e of weight at most w, obtaining the ciphertext $y = xG' + e$.

Decryption: Given a ciphertext y, calculate $yP^{-1} = xG'P^{-1} + eP^{-1} = xSG + eP^{-1}$, and since the weight of eP^{-1} is still the same, it is enough to apply the decoding algorithm for the code to retrieve xS and consequently x.

The other computational assumption underlying the security is that the $k \times n$ matrix G' so obtained is computationally indistinguishable from a uniform matrix of the same size, hence an attacker that does not know the private key is faced with solving GDP.

Remark The encryption process is dominated by the cost of computing xG', which requires at most $k \times n$ field multiplications. Hence this is fast. On the other hand, decryption requires performing a decoding algorithm and is not usually so fast. Therefore, McEliece is most suitable for applications where encryption is required to be fast. This is analogous to RSA using small encryption exponents.

2.2 The Niederreiter cryptosystem

A first alternative version of the McEliece cryptosystem has been proposed by Niederreiter [26] in 1986, and has been proved to be equivalent in terms of security. It is often considered as a "dual" version, as the trapdoor is given by the parity-check matrix rather than the generator matrix. The underlying hard problem is the Syndrome Decoding Problem.

Definition 2 (SDP). *Let H be an $r \times n$ matrix over $\mathbb{F}_q$, s a vector of $\mathbb{F}_q^r$ and $w > 0$.*
Find a vector e in $\mathbb{F}_q^n$ of weight $\leq w$ such that $He^T = s$.

If H is the parity-check matrix for an $[n, k]$ linear code C, then $r = n - k$ and it is immediate to see that the two problems are equivalent: in fact, for $y = c + e$ we have $Hy^T = Hc^T + He^T$ but $Hc^T = 0$ since c is a codeword so $Hy^T = He^T = s$, which means that SDP in this case corresponds, again, to finding an error vector of weight less or equal to w.
This is a description of Niederreiter's scheme:

Key Generation: Pick an $(n - k) \times n$ parity-check matrix H for a w-error correcting linear code with an efficient decoding algorithm over the finite field $\mathbb{F}_q$, an $(n - k) \times (n - k)$ invertible matrix S and an $n \times n$ permutation matrix P at random, then evaluate $H' = SHP$, which is another valid parity-check matrix. The private key consists of H, S, P, and the the public key is H'.The system parameters n, k, w are also public.

Encryption: A plaintext here is a vector $e \in \mathbb{F}_q^n$ of weight at most w; to encrypt, compute the corresponding syndrome, obtaining the ciphertext $y = H'e^T$.

Decryption: Given a ciphertext y, calculate first $S^{-1}y = HPe^T$, and then apply the decoding algorithm for the code to retrieve Pe^T and consequently e.

2.3 Structured matrices

Definition 3. *Given a ring R (in our case the finite field $\mathbb{F}_{q^m}$) and a vector $\bar{h} = (h_0, \ldots, h_{n-1}) \in \mathsf{R}^n$, the dyadic matrix $\Delta(\bar{h}) \in \mathsf{R}^{n \times n}$ is the symmetric matrix with components $\Delta_{ij} = h_{i \oplus j}$, where $\oplus$ stands for bitwise exclusive-or on the binary representations of the indices. The sequence $\bar{h}$ is called its signature. Moreover, $\Delta(t, \bar{h})$ denotes the matrix $\Delta(\bar{h})$ truncated to its first t rows. Finally, we call a matrix quasi-dyadic if it is a block matrix whose component blocks are $t \times t$ dyadic submatrices.*

If n is a power of 2, then every $2^k \times 2^k$ dyadic matrix can be described recursively as

$$M = \begin{pmatrix} A & B \\ B & A \end{pmatrix}$$

where each block is a $2^{k-1} \times 2^{k-1}$ dyadic matrix (and where any 1×1 matrix is dyadic).

Definition 4. *Given two sequences $\bar{x} = (x_1, \ldots, x_n)$, $\bar{y} = (y_1, \ldots, y_n) \in \mathbb{F}_q^n$, a* Generalized Reed-Solomon (GRS) code *of order ℓ is defined by a parity-check matrix related to the Vandermonde form, i.e. the matrix with components $H_{ij} = y_j x_j^{i-1}$:*

$$H = \begin{pmatrix} y_1 & \cdots & y_n \\ y_1 x_1 & \cdots & y_n x_n \\ \vdots & \vdots & \vdots \\ y_1 x_1^{\ell-1} & \cdots & y_n x_n^{\ell-1} \end{pmatrix}.$$

If the resulting code is then restricted to $\mathbb{F}_q$ it is called an Alternant code.

Definition 5. *For $m, n, s, t \in \mathbb{N}$ and a prime power q, let $\bar{\alpha} = (\alpha_1, \ldots, \alpha_n)$, $\bar{w} = (w_1, \ldots, w_s)$ be $n + s$ distinct elements of $\mathbb{F}_{q^m}$, and $(z_1, \ldots, z_n)$ be nonzero elements of $\mathbb{F}_{q^m}$. The* Generalized Srivastava (GS) code *of order st and length n is defined by a parity-check matrix of the form:*

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

where each block is

$$H_i = \begin{pmatrix} \dfrac{z_1}{\alpha_1 - w_i} & \cdots & \dfrac{z_n}{\alpha_n - w_i} \\ \dfrac{z_1}{(\alpha_1 - w_i)^2} & \cdots & \dfrac{z_n}{(\alpha_n - w_i)^2} \\ \vdots & \vdots & \vdots \\ \dfrac{z_1}{(\alpha_1 - w_i)^t} & \cdots & \dfrac{z_n}{(\alpha_n - w_i)^t} \end{pmatrix}.$$

The parameters for such a code are the length $n \leq q^m - s$, dimension $k \geq n - mst$ and minimum distance $d \geq st + 1$.
GS codes are part of the family of Alternant codes, and therefore benefit of an efficient decoding algorithm. More information about this class of codes can be found in [21, Ch. 12, §6].

2.4 Secure parameters

Both the previous schemes share some common traits: a very fast and efficient encryption procedure, and very big public keys. Our proposal to deal with these issues is to use structured codes, and in particular, quasi-dyadic codes. See Appendix B for a summary of the key generation process.
Misoczki and Barreto in [25] give an assessment of the hardness of decoding quasi-dyadic codes, providing a reduction to the Syndrome Decoding Problem.
Keeping in mind the scope of the paper, the parameters proposed in [27, Table 3] seem to fit our proposal best; we report the table here for completeness.

Table 1: Quasi-dyadic GS codes [27, Table 3]. The column "Size" indicates the size of the public key, while in the column "Security level" are reported the approximate cost of general decoding attacks ($\log_2$ of binary operations).

Base Field	m	n	k	s	t	Errors	Size (bytes)	Security level[1]
$\mathbb{F}_{2^5}$	2	992	416	2^5	9	144	4680	128
$\mathbb{F}_{2^4}$	3	768	432	2^4	7	56	4536	80
$\mathbb{F}_{2^5}$	2	512	256	2^4	2^3	64	2560	80

3 CCA-secure schemes

Until now, we have been considering only the weakest notion of security for a public-key encryption scheme, that is, One-Way Encryption (OWE). The following are formal definitions of public-key encryption and one-way security.

Definition 6. *A Public-Key Encryption (PKE) scheme consists of a 6-tuple* $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{G}, \mathcal{E}, \mathcal{D})$ *defined as follows:*

- $\mathcal{K} = \mathcal{K}_{publ} \times \mathcal{K}_{priv}$ *is the* key space.
- $\mathcal{P}$ *is the set of messages to be encrypted, or* plaintext space.
- $\mathcal{C}$ *is the set of the messages transmitted over the channel, or* ciphertext space.
- $\mathcal{G}$ *is a probabilistic key generation algorithm that takes as input a security parameter* 1^δ *and outputs a public key* $pk \in \mathcal{K}_{publ}$ *and a private key* $sk \in \mathcal{K}_{priv}$.

[1] http://www2.mat.dtu.dk/people/C.Peters/isdfq.html

- $\mathcal{E}$ *is a (possibly probabilistic) encryption algorithm that receives as input a public key* $pk \in \mathcal{K}_{publ}$ *and a plaintext* $x \in \mathcal{P}$ *and returns a ciphertext* $\psi \in \mathcal{C}$.
- $\mathcal{D}$ *is a deterministic decryption algorithm that receives as input a private key* $sk \in \mathcal{K}_{priv}$ *and a ciphertext* $\psi \in \mathcal{C}$ *and outputs either a plaintext* $x \in \mathcal{P}$ *or the failure symbol* $\perp$.

Definition 7 (One-Way). *A One-Way adversary is a polynomial-time algorithm* $\mathcal{A}$ *that takes as input a public key* $pk \in \mathcal{K}_{publ}$ *and a ciphertext* $\psi \in \mathcal{C}$. *We say that a PKE is* One-Way Secure *if the probability of success of any adversary* $\mathcal{A}$ *is negligible in the security parameter, i.e.*

$$Pr[pk \leftarrow \mathcal{K}_{publ}, x \leftarrow \mathcal{P} : \mathcal{A}(pk, \mathcal{E}_{pk}(x)) = x] \in negl(\delta)$$

The standard definitions for *Indistinguishability*, and the attack models CPA and CCA2 are omitted here due to space requirements.

3.1 CCA2 security conversions

There are standard ways to obtain an IND-CCA2 secure encryption scheme from one that only has OW-CPA, for example the Fujisaki-Okamoto transform [17]. The construction achieves CCA2-security by integrating an asymmetric encryption scheme with a symmetric scheme.

Definition 8. *A Symmetric Encryption (SE) scheme consists of a 5-tuple* $(\mathcal{K}, \mathcal{P}, \mathcal{C}, \mathcal{E}, \mathcal{D})$ *defined as follows:*

- $\mathcal{K}$ *is the* key space.
- $\mathcal{P}$ *is the set of messages to be encrypted, or* plaintext space.
- $\mathcal{C}$ *is the set of the messages transmitted over the channel, or* ciphertext space.
- $\mathcal{E}$ *is a deterministic encryption algorithm that receives as input a key* $\chi \in \mathcal{K}$ *and a plaintext* $x \in \mathcal{P}$ *and returns a ciphertext* $\psi \in \mathcal{C}$.
- $\mathcal{D}$ *is a deterministic decryption algorithm that receives as input a key* $\chi \in \mathcal{K}$ *and a ciphertext* $\psi \in \mathcal{C}$ *and outputs a plaintext* $x \in \mathcal{P}$.

The Fujisaki-Okamoto conversion requires an additional property of the encryption scheme called γ-$uniformity$. We define it here.

Definition 9. *Let* Π *be a PKE defined as above and let's call* $\mathcal{R}$ *the set where the randomness to be used in the (probabilistic) encryption is chosen. For given* $(pk, sk) \in \mathcal{K}$, $x \in \mathcal{P}$ *and a string* y, *we define*

$$\gamma(x, y) = Pr[r \xleftarrow{\$} \mathcal{R} : y = \mathcal{E}_{pk}(x, r)]$$

where the notation $\mathcal{E}_{pk}(x, r)$ *makes explicit the role of the randomness* r. *We say that* Π *is* γ**-uniform** *if, for any* $(pk, sk) \in \mathcal{K}$, *any* $x \in \mathcal{P}$ *and any* y, $\gamma(x, y) \leq \gamma$ *for a certain* $\gamma \in \mathbb{R}$.

Table 2: The Fujisaki-Okamoto conversion. $\mathcal{H}_1$ and $\mathcal{H}_2$ are hash functions.

Encryption of x	Decryption of ψ
$\sigma \xleftarrow{\$} \mathcal{P}^{PKE}$	$\psi := (\psi_1 \| \psi_2)$
$r := \mathcal{H}_1(\sigma, x)$	$\hat{\sigma} := \mathcal{D}_{sk}^{PKE}(\psi_1)$ (return $\perp$ if decryption fails)
$\psi_1 := \mathcal{E}_{pk}^{PKE}(\sigma, r)$	$\hat{x} := \mathcal{D}_{\mathcal{H}_2(\hat{\sigma})}^{SE}(\psi_2)$ (return $\perp$ if decryption fails)
$\psi_2 := \mathcal{E}_{\mathcal{H}_2(\sigma)}^{SE}(x)$	$\hat{r} := \mathcal{H}_1(\hat{\sigma}, \hat{x})$
	if $\mathcal{E}_{pk}^{PKE}(\hat{\sigma}, \hat{r}) == \psi_1$ return $x := \hat{x}$
return $\psi := (\psi_1 \| \psi_2)$	else return $\perp$

In a successive paper [20], Kobara and Imai proposed three alternative constructions in a similar fashion, tailored specifically for the McEliece cryptosystem rather than a general OWE encryption scheme. The biggest contribution of the new constructions is that the amount of overhead data (i.e. difference between the bit-length of the ciphertext and the bit-length of the plaintext) is considerably reduced.
While this is certainly an important issue for some applications, in the common cryptographic practice it will never constitute a serious concern. In fact, the aim of public key cryptography is not to encrypt a whole, large plaintext, but rather to encrypt just a small (e.g. 128 or 256 bits) key for a more efficient symmetric scheme, that will be then used to encrypt the message. From a computational point of view the Kobara-Imai encryption process seems to be more expensive; in fact, the whole construction is rather complex.

Table 3: The Kobara-Imai hybrid conversion γ for the McEliece (McE) public-key encryption scheme. $\mathcal{H}$ is a hash function, Gen a random number generator, $Conv$ a constant weight encoding function and $Const$ a (predetermined) public constant.

Encryption of x	Decryption of ψ
$r \xleftarrow{\$} \{0,1\}^*$	$\psi := (y_5 \| y')$
$y_1 := Gen(r) \oplus (x \| Const)$	$y_3 := \mathcal{D}_{sk}^{McE}(y')$
$y_2 := r \oplus \mathcal{H}(y_1)$	$y_3 G' \oplus y'$
$(y_5 \| y_4 \| y_3) := (y_2 \| y_1)$	$y_4 := Conv^{-1}(z)$
$z := Conv(y_4)$	$(y_2 \| y_1) := (y_5 \| y_4 \| y_3)$
	$r := y_2 \oplus \mathcal{H}(y_1)$
	$(\hat{x} \| Const') := y_1 \oplus Gen(r)$
	if $Const' == Const$ return $x := \hat{x}$
return $\psi := (y_5 \| \mathcal{E}_{pk}^{McE}(y_3, z))$	else return $\perp$

Note that the Fujisaki-Okamoto decryption process includes an encoding operation in the final check. This makes decryption slower. The cost of the

process, though, is still dominated by the decoding operation rather than the matrix-vector multiplication. Moreover, as we already remarked, we argue that the distinctive feature of the McEliece scheme is the fast encryption process, and the Fujisaki-Okamoto conversion preserves fast encryption better than the Kobara-Imai approach.

3.2 Applying Fujisaki-Okamoto to McEliece

We give here a new way to use McEliece together with the Fujisaki-Okamoto transform. Previous approaches always needed a constant weight encoding function to convert $\mathcal{H}_1(\sigma, x)$ into an error vector. Our idea is to swap the message and the error in the McEliece scheme, with a technique similar to the one used by Micciancio in [24]. This means that we interpret $\mathcal{E}_{G'}^{McE}(x, r) = rG' + x$, encoding the message in the error vector rather than in the codeword. This is possible because, unlike other PKE's, the decryption process of McEliece, consisting mainly of decoding, returns both x and r, allowing to recover, in addition to the plaintext, also the randomness used. With this simple trick, we avoid having to use a (costly) constant weight encoding function and we simplify the encryption process considerably.
For simplicity we take the symmetric encryption scheme to be the one-time pad with an ephemeral key generated as $\mathcal{H}_2(\sigma)$ where $\mathcal{H}_2$ is a random oracle with arbitrary length output. This symmetric encryption scheme satisfies the Find-Guess security property. In practice, one might use a block cipher in CBC mode.

Table 4: The Fujisaki-Okamoto transform applied to McEliece.

Encryption of x	Decryption of ψ
$\sigma \xleftarrow{\$} \mathcal{W}_{n,w}$	$\psi := (\psi_1 \| \psi_2)$
$r := \mathcal{H}_1(\sigma \| x)$	$\hat{\sigma} := \mathcal{D}_G^{McE}(\psi_1)$ (return $\perp$ if decoding fails)
$\psi_1 := rG' + \sigma$	$\hat{x} = \mathcal{H}_2(\hat{\sigma}) \oplus \psi_2$
$\psi_2 := \mathcal{H}_2(\sigma) \oplus x$	$\hat{r} := \mathcal{H}_1(\hat{\sigma} \| \hat{x})$
	if $\hat{r}G' + \hat{\sigma} == \psi_1$ return $x := \hat{x}$
return $\psi := (\psi_1 \| \psi_2)$	else return $\perp$

The following lemma is fundamental to prove that our scheme enjoys the γ-uniformity required by the conversion.

Lemma 1. *The McEliece encryption scheme described above is γ-uniform for* $\gamma = \dfrac{1}{q^k}$.

Proof. Let G' be a public key that is a generator matrix for the code C; in our setting, y is a generic string in $\mathbb{F}_q^n$. Then clearly:

$$\gamma(\sigma, y) = Pr[r \xleftarrow{\$} \mathbb{F}_q^k : y = rG' + \sigma] = \begin{cases} 0 \text{ if } y - \sigma \notin \mathsf{C} \\ \\ \dfrac{1}{q^k} \text{ if } y - \sigma \in \mathsf{C} \end{cases}$$

and that concludes the proof. □

Theorem 1. *If the assumptions of indistinguishability and decoding hardness of the McEliece PKE hold, the encryption scheme described in Table 4 is IND-CCA2 secure.*

Proof. The scheme enjoys one-way security because of the computational assumptions in the hypothesis. Moreover, Lemma 1 provides the γ-uniformity as required. Finally, the symmetric scheme used (one-time pad) satisfies the required security property (Find-Guess). It is then possible to apply [17, Th. 12]. □

4 Efficient implementation

The implementation was done in C++ and is based on the library *SBCrypt* (Syndrome-Based Cryptography Library) by Barreto, Misoczki and Villas Boas [3]. We subsequently converted our code to run on an embedded device, namely the microcontroller ATxmega256A3 from the AVR XMEGA family. It has 264 Kbytes of Flash memory, 16 Kbytes of SRAM memory and is running at a clock frequency of 32 MHz.

To represent the finite fields we used exponential/antilog tables [21, Ch. 4, §5], which is possible as our extension fields are small enough to fit completely in the available memory (apart from the first code, for which the private trapdoor would be too big). This is a key feature of our scheme and one of the main reasons to choose GS codes over Goppa codes. In fact, when using GS codes, it is possible to choose secure parameters even for codes defined over relatively small extension fields. See Appendix C for a summary of the security discussion. More information can be found in [27].

As for the hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$, we opted for the Keccak family [10], one of the five remaining SHA-3 finalists, with assigned output length equal to k, in the first instance, or equal to the plaintext length (128 bits in our case), in the second. Its flexibility also allows for using it as stream cipher, and we deployed it for randomly choosing error vectors of weight w.

The procedure to generate error vectors for encryption is as follows: at first, the error vector is initialized to zero. Next, we ask Keccak for $\beta = \lceil \log_2 n \rceil$ bits and interpret the result as an index into the error vector. If the interval is greater than n then we reject and re-sample. Now, in case this index is still a zero entry, we ask Keccak for additional bits to be read as a field element. Otherwise, we ask Keccak for the next bits to be interpreted as the next index to be examined. This simple procedure is iterated until the error vector has the desired weight. It is clear that this process samples uniformly from $\mathcal{W}_{n,w}$.

The test results for the C++ code have been executed on an Intel(R) Core(TM) 2 Duo CPU E8400@3.00GHz running Ubuntu/Linux 2.6.32, where the source has been compiled with gcc 4.4.3. Similar results have been obtained using the Intel compiler icpc/icc. As for the embedded microcontroller, the code has been simulated on AVR Studio 5.0 [1].

McEliece based on GS codes We have measured two different operations: the encoding step $xG + e$ for $x \in \mathbb{F}_q^k$ and the decoding of a ciphertext $y \in \mathbb{F}_q^n$. Results are presented in Table 5 (timings expressed in milliseconds (ms)).

Table 5: Profiling results for McEliece using GS codes.

Code Name	Base Field	m	n	k	s	t	Errors	Encoding	Decoding
$\mathfrak{A}$	$\mathbb{F}_{2^5}$	2	992	416	2^5	9	144	0.287	5.486
$\mathfrak{B}$	$\mathbb{F}_{2^4}$	3	768	432	2^4	7	56	0.179	1.578
$\mathfrak{C}$	$\mathbb{F}_{2^5}$	2	512	256	2^4	2^3	64	0.093	1.234

It is easy to see that the decoding process dominates the runtime.
The following tables report the results obtained when running the same operations on the microcontroller, for the last two codes. The costs displayed are in clock cycles; for a conversion to the standard time units, keep in mind that the device runs at 32MHz, hence we have 32 million cycles per second.

Table 6: Details of the costs of encryption and decryption steps for codes $\mathfrak{B}$ and $\mathfrak{C}$.

Operation	**Code $\mathfrak{B}$**	**Code $\mathfrak{C}$**
Generate error vector e	313,114	316,568
Load the plaintext x	4,313	2,553
Encode xG	3,418,292	1,603,854
Add e	8,818	5,944
Encoding total	3,744,537	1,928,919

Operation	**Code $\mathfrak{B}$**	**Code $\mathfrak{C}$**
Compute syndrome Hy^T	6,910,742	5,440,245
Solve key equation	955,597	1,192,400
Compute error positions	2,061,066	1,571,689
Compute error values	611,898	794,463
Correct the errors	8,641	5,121
Decoding total	10,547,944	9,003,918

Note on decoding In our scheme, we have implemented a standard alternant decoder (see for example [21, Ch. 12, §9]). That consists of extrapolating the key equation from the syndrome and then solve it and compute the error positions as the roots of the error locator polynomial. To find the roots, we use the Horner scheme in the sense that we directly evaluate the polynomial on the support. More sophisticated root-finding algorithms are available, for instance Berlekamp's trace algorithm [6]. However, our codes are punctured codes, and, as also stated in [19], Berlekamp's trace algorithm is not designed for such a case. Moreover, although Berlekamp's algorithm does find the roots of the polynomial, there is an additional step necessary to find them in the support sequence, which is not the case when using the Horner scheme and direct evaluation. Finally, one can see from the timings of the decoding operation, that the by far dominating part is the syndrome computation. For the time being, we therefore refrained from implementing Berlekamp's algorithm, opting for the much simpler Horner scheme instead.

CCA2-McEliece based on GS codes The performances of the scheme are given in Table 7 and Table 8, respectively for the C++ code and for the microcontroller.

Table 7: Profiling results for CCA2-McEliece using GS codes.

Code Name	Base Field	m	n	k	s	t	Errors	Encryption	Decryption
$\mathfrak{A}$	$\mathbb{F}_{2^5}$	2	992	416	2^5	9	144	0.323	5.914
$\mathfrak{B}$	$\mathbb{F}_{2^4}$	3	768	432	2^4	7	56	0.213	1.814
$\mathfrak{C}$	$\mathbb{F}_{2^5}$	2	512	256	2^4	2^3	64	0.114	1.382

Table 8: Details of the costs of the encryption and decryption steps of CCA2-McEliece.

Operation	**Code $\mathfrak{B}$**	**Code $\mathfrak{C}$**
Generate error vector σ	322,109	321,812
Load the plaintext x	1,019	1,019
Hash $r = \mathcal{H}(\sigma, x)$	282,285	281,497
Encode rG	3,426,700	1,591,031
Add σ	1,103	1,314
Hash $\mathcal{K}(\sigma)$	137,704	137,720
Pad $\mathcal{K}(\sigma) \oplus x$	1,814	1,811
Encryption total	4,171,734	2,336,204

Operation	**Code $\mathfrak{B}$**	**Code $\mathfrak{C}$**
Compute syndrome $H\psi_1^T$	7,029,985	5,425,696
Solve key equation	954,522	1,202,032
Compute error positions	2,031,514	1,561,946
Compute error values	611,944	794,524
Correct the errors	1,108	5,112
Hash $\mathcal{K}(\hat{\sigma})$	147,822	144,768
Pad $\mathcal{K}(\hat{\sigma}) \oplus \psi_2$	1,585	1,586
Hash $\hat{r} = \mathcal{H}(\hat{\sigma}, \hat{x})$	282,066	282,278
Encode $\hat{r}G$	3,426,721	1,591,049
Add $\hat{\sigma}$	1,113	1,273
Check equality	9,207	6,135
Decryption total	14,497,587	11,016,399

Comparing the results in Table 5 and Table 7 (as well as Table 6 and Table 8), we see that indeed the computational overhead is quite low.
For simplicity, the comparison of the total timings for both cases is reported in Tables 9 and 10.

Table 9: Summary of the timings (ms) for the C++ code.

Code	Encoding	CCA2 Encryption	Decoding	CCA2 Decryption
$\mathfrak{A}$	0.287	0.323	5.486	5.914
$\mathfrak{B}$	0.179	0.213	1.578	1.814
$\mathfrak{C}$	0.093	0.114	1.234	1.382

Table 10: Summary of the timings (clock cycles) for the embedded device.

Code	Encoding	CCA2 Encryption	Decoding	CCA2 Decryption
$\mathfrak{B}$	3,744,537	4,171,734	10,547,944	14,497,587
$\mathfrak{C}$	1,928,919	2,336,204	9,003,918	11,016,399

5 Conclusions

In this paper we propose the implementation of a construction based on quasi-dyadic generalized Srivastava codes. We first implement a plain McEliece encryption scheme, and then convert it to a CCA2-secure scheme using the Fujisaki-Okamoto transform. The results are initially given for a C++ implementation, and successively for an embedded device.
An independent work proposing a CCA2-secure scheme based on quasi-dyadic Goppa codes has been recently presented at PQCrypto 2011 by Stefan Heyse

[19]. The performance indicated for encryption and decryption on the embedded device are slower than our results (the simulator program is the same, AVR Studio, although in a slightly older version). Part of the reason is due to the use a constant weight encoding function (more than three times as costly as hashing) that we avoid thanks to the particular configuration of our scheme. However, the major difference comes from the fact that our vector-matrix multiplication, despite performing operations over non-binary fields, is at least two times faster, and this is the dominating part in the encryption process and is also a very high cost in the decryption process. This is a direct consequence of the structure of the scheme. In fact, the construction in [19] makes use of binary Goppa codes, which for security reasons [14] need to be defined over the extension field $\mathbb{F}_{2^{16}}$: this is too big to fit the corresponding log/antilog tables on the flash memory of the device. The result is that, in order to avoid using additional, external memory, the tables for $\mathbb{F}_{2^8}$ are represented instead, and operations are performed using tower field arithmetic, which is much slower. For example, a multiplication over a tower $\mathbb{F}_{(2^8)^2}$ is equivalent to performing 5 multiplications over $\mathbb{F}_{2^8}$.
Another disadvantage is constituted by the fact that the public key G' is computed as SG like in the original McEliece (P is supposed to be implicit into the support of the code), and the scramble matrix S occupies a great amount of memory (131,072 bytes, see [19, Table 3]). This is completely redundant, as the reduction to the systematic form is enough to mask the trapdoor and provide one-way security [11].
On the other hand, the length of the encrypted plaintext is about 10 times the length of our plaintext (1288 bits, as opposed to 128 bits); however, we stress again that, in a "real-world" scenario, public-key encryption would only be used for encrypting a small amount of data, for obvious reasons. So if a large number of bits needs to be encrypted, with every probability a PKE would be used to exchange a small key (usually 128 or 256 bits) and then the plaintext would be encrypted with a symmetric encryption scheme.
If we follow this approach in our case, the timings that we obtain strongly support our claim. The latest benchmark speed indicated for AES-128 is about 16 cycles per byte[2]. Hence, if we want to encrypt, for a comparison, a plaintext of length 1288 bits = 161 bytes, it would take only 2,576 clock cycles; even on an embedded device, this number is very small compared to the rest of the encryption process. In total, our encryption process ranges from around 1.5 to 2.7 times faster than [19].

Table 11: Cost of encrypting a plaintext of length 1288 bits

Code	**Cost (clock cycles)**
Goppa + Kobara-Imai	6,358,952
Code $\mathfrak{B}$	4,174,310
Code $\mathfrak{C}$	2,338,780

A similar argument holds for decryption.

Finally, we would like to highlight that we are using Keccak to represent both our hash functions and a random number generator; the flexibility that it provides is evident. Other SHA-3 competitors like the function Blue Midnight Wish (BMW) used in [19] have been proved to be faster [16], but do not reach the same level of security, and for this have been discarded: although, as noted in the announcement of the finalists, "none of these candidates was clearly broken", several attacks have been presented[3].
Further investigation is certainly still required, but for a totally detailed analysis probably even a comparison at source code level would become necessary, and that falls beyond the scope of this paper.

6 Acknowledgments

We would like to thank Steven Galbraith for many fruitful discussions and his constant support throughout the development of the paper.

References

1. Atmel Corporation, "AVR Studio 5.0". *www.atmel.com/avrstudio*.
2. P. S. L. M. Barreto, P.-L. Cayrel, R. Misoczki, and R. Niebuhr, "Quasi-dyadic CFS signatures", volume 6584 of *LNCS*, pages 336-349, Springer, October 2010.
3. P. S. L. M. Barreto and R. Misoczki and L. B. Villas Boas, "SBCRYPT - Syndrome-Based Cryptography Library".
4. T. P. Berger, P. L. Cayrel, P. Gaborit and A. Otmani, "Reducing key length of the McEliece cryptosystem". In Bart Preneel, editor, *Progress in Cryptology - Second International Conference on Cryptology in Africa (AFRICACRYPT 2009)*, volume 5580 of *LNCS*, pages 77-97, Gammarth, Tunisia, June 21-25, 2009.
5. T. P. Berger and P. Loidreau, "How to mask the structure of codes for a cryptographic use". In *Design, Codes and Cryptography*, volume 35, pages 63-79, 2005.
6. E. R. Berlekamp, "Factoring polynomials over finite fields", volume 46 of *Bell System Technical Journal*, pages 1853–1859, 1967.
7. E. R. Berlekamp, R. J. McEliece and H. C. A. van Tilborg, "On the inherent intractability of certain coding problems". In *IEEE Transactions on Information Theory*, volume 24, pages 384-386, 1978.
8. D. J. Bernstein, T. Lange and C. Peters, "Attacking and defending the McEliece cryptosystem". In J. Buchman and J. Ding, editors, *Post-Quantum Cryptography-Second International Workshop (PQCrypto 2008)*, volume 5299 of *LNCS*, pages 31-46, Springer, Berlin, 2008.
9. D. J. Bernstein, T. Lange, C. Peters and H. C. A. van Tilborg, "Explicit bounds for generic decoding algorithms for code-based cryptography". In *Pre-proceedings of WCC 2009*, pages 168-180, 2009.

[2] http://www.cryptopp.com/benchmarks.html
[3] http://ehash.iaik.tugraz.at/wiki/Blue_Midnight_Wish

10. G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, "The Keccak sponge function family". *http://keccak.noekeon.org/*
11. B. Biswas and N. Sendrier, "McEliece Cryptosystem Implementation: Theory and Practice". In *PQCrypto 2008*, pages 47-62, 2008.
12. R. Dowsley, J. Müller-Quade, and A. C. A. Nascimento, "A CCA2 secure public key encryption scheme based on the McEliece assumptions in the standard model". In *Topics in Cryptology - CT-RSA 2009*, LNCS, volume 5473, pages 240-251, 2009.
13. T. Eisenbarth, T. Güneysu, S. Heyse and C. Paar, "Microeliece: McEliece for embedded devices". In *CHES '09: Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 49-64, Berling, Heidelberg, 2009. Springer-Verlag.
14. J. C. Faugère, A. Otmani, L. Perret and J. P. Tillich, "Algebraic Cryptanalysis of McEliece Variants with Compact Keys". In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 279-298, French Riviera, May 30 - June 3, 2010.
15. J. C. Faugère, A. Otmani, L. Perret and J. P. Tillich, "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.
16. E. Fleischmann, C. Forler, M. Gorski, "Classification of the SHA-3 Candidates". *http://drops.dagstuhl.de/volltexte/2009/1948/pdf/09031.ForlerChristian.Paper.1948.pdf*
17. E.Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes". In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, volume 6110 of *LNCS*, Springer-Verlag, pages 537-554, London, 1999.
18. S. Heyse, "Low-reiter: Niederreiter encryption scheme for embedded microcontrollers". In *Post-Quantum Cryptography, Third International Workshop, (PQCrypto 2010)*, Springer, 2010.
19. S. Heyse, "Implementation of McEliece Based on Quasi-dyadic Goppa Codes for Embedded Devices". In *Post-Quantum Cryptography, Fourth International Workshop, (PQCrypto 2011)*, Springer, 2011.
20. K. Kobara and H. Imai, "Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC". In *PKC '01: Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, Springer-Verlag, pages 19-35, London, 2001.
21. F. J. MacWilliams and N. J. Sloane, "The theory of error-correcting codes". North Holland, Amsterdam, 1977.
22. D. Mandell Freeman, O. Goldreich, E. Kiltz, A. Rosen, and G. Segev, "More constructions of lossy and correlation-secure trapdoor functions". In *Public Key Cryptography - PKC 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 279-295, 2010.
23. R. J. McEliece, "A Public-Key System Based on Algebraic Coding Theory". In *DSN Progress Report 44*, pages 114-116, Jet Propulsion Lab, 1978.
24. D. Micciancio, "Improving Lattice Based Cryptosystems Using the Hermite Normal Form". In *CaLC '01*, pages 126-145, 2001.
25. R. Misoczki and P. S. L. M. Barreto, "Compact McEliece keys from Goppa codes". In *Selected Areas in Cryptography (SAC 2009)*, Calgary, Canada, August 13-14, 2009.
26. H. Niederreiter, "A public-key cryptosystem based on shift register sequences". In *EUROCRYPT*, volume 219 of *LNCS*, pages 35-39, 1985.

27. E. Persichetti, "Compact McEliece keys based on Quasi-Dyadic Srivastava codes". In IACR Cryptology ePrint Archive, preprint, 2011.
28. C. Peters, "Information-set decoding for linear codes over $\mathbb{F}_q$". In *Post-Quantum Cryptography, Third International Workshop, (PQCrypto 2010)*, volume 6061 of *LNCS*, pages 81-94, Darmstadt, Germany, May 25-28, 2010.
29. B. Preneel, A. Bosselaers, R. Govaerts and J. Vandewalle, "A software implementation of the McEliece public-key cryptosystem". In *Proceedings of the 13th Symposium on Information Theory in the Benelux, Werkgemeenschap voor Informatieen Communicatietheorie*, pages 119-126, Springer-Verlag, 1992.
30. "Prometheus. Implementation of McEliece cryptosystem for 32-bit microprocessors (c-source)". *http://www.eccpage.com/.*
31. S. Schechter, "On the inversion of certain matrices". In *Mathematical Tables and Other Aids to Computation*, volume 13, issue 66, pages 73-77, 1959.
32. N. Sendrier, "Encoding information into constant weight words". In *IEEE Conference, ISIT 2005*, pages 435-438, September 2005.

A Additional definitions

We present here some additional definitions needed for the key generation process.

Definition 10. *Given two disjoint sequences* $\bar{v} = (v_1, \ldots, v_\ell) \in \mathbb{F}_q^\ell$ *and* $\bar{L} = (L_1, \ldots, L_n) \in \mathbb{F}_q^n$*, the* Cauchy matrix $C(\bar{v}, \bar{L})$ *is the matrix with components* $C_{ij} = \dfrac{1}{v_i - Lj}$*, i.e.*

$$C(\bar{v}, \bar{L}) = \begin{pmatrix} \dfrac{1}{v_1 - L_1} & \cdots & \dfrac{1}{v_1 - L_n} \\ \vdots & \vdots & \vdots \\ \dfrac{1}{v_\ell - L_1} & \cdots & \dfrac{1}{v_\ell - L_n} \end{pmatrix}.$$

Cauchy matrices have the property that all of their submatrices are invertible [31].

Definition 11. *Fix a finite field* $\mathbb{F}_q$ *and an integer* $m > 1$*. Choose a polynomial* $g(z)$ *in* $\mathbb{F}_{q^m}[z]$ *of degree* $t < n/m$ *and a sequence of distinct elements* $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{q^m}$ *such that* $g(\alpha_i) \neq 0$ *for all* i*. The polynomial* $g(z)$ *is called the* Goppa polynomial*. The set of words* $\bar{c} = (c_1, \ldots, c_n) \in \mathbb{F}_{q^m}^n$ *with* $\sum_{i=1}^n \frac{c_i}{z - \alpha_i} \equiv 0 \pmod{g(z)}$ *defines an* $[n, n-t]$ *linear code over* $\mathbb{F}_{q^m}$*. The corresponding* Goppa code *is the restriction of this code to* $\mathbb{F}_q$*, i.e. the set of elements* $\bar{c} = (c_1, \ldots, c_n) \in \mathbb{F}_q^n$ *which satisfy the above condition.*

Alternatively (and usually) a Goppa code is defined by means of its parity-check matrix, which is of the form:

$$H = \begin{pmatrix} \frac{1}{g(\alpha_1)} & \cdots & \frac{1}{g(\alpha_n)} \\ \vdots & \vdots & \vdots \\ \frac{\alpha_1^{t-1}}{g(\alpha_1)} & \cdots & \frac{\alpha_n^{t-1}}{g(\alpha_n)} \end{pmatrix}$$

It is clear then that a Goppa code has dimension $k \geq n - mt$. The minimum distance is $t + 1$, or $2t + 1$ in the special binary case ($q = 2$).
Goppa codes are a particular instance of Alternant codes, with $x_i = \alpha_i$, $y_i = 1/g(\alpha_i)$.

B Quasi-dyadic key generation

Misoczki and Barreto in [25] first introduced a scheme based on quasi-dyadic Goppa codes, making use of codes simultaneously in dyadic [25, Th. 2] and Cauchy form [21, Ch. 12, Pr. 5]. Necessary conditions are that the generator polynomial has to be monic and without multiple zeros, and that the code needs to be defined over a field of characteristic 2, with a dyadic signature satisfying

$$\frac{1}{h_{i \oplus j}} = \frac{1}{h_i} + \frac{1}{h_j} + \frac{1}{h_0}. \tag{1}$$

The scheme was subsequently extended and generalized to the case of GS codes [27], with multiple benefits including security improvements (described in the next section). Since it can be easily proved that every generalized Srivastava code with $t = 1$ is a Goppa code, the two cases are in fact just two instances of the same scheme. For the construction, we follow the steps presented in [27, Section 4].

Equation (1) is the core of the key generation algorithm. The procedure takes input parameters n, s, t such that $n = n_0 s$, $mst < n$ for s a power of 2 and a finite field $\mathbb{F}_{q^m} = \mathbb{F}_{2^u}$ where $q = 2^\lambda$, $u = m\lambda$, then assigns distinct values at random to the elements h_{2^j} for $j = 1, \ldots, \log_2(n-1)$, in the meantime fixing the elements between h_{2^j} and $h_{2^{j+1}}$ by using (1).
An initial block in dyadic form is formed from the signature $\bar{h}$ just built; this is equivalent to a Goppa code. In case $t > 1$, the other blocks are computed by successive powering, up to the power of t. The parity-check matrix eventually obtained is projected onto the base field and finally, we retain the non-trivial part of its systematic form to be used as trapdoor.

We refer to [27] for a fully detailed description of the construction process.

C Resistance to structural attacks

The main threat against quasi-dyadic schemes is represented by the so-called FOPT attack [14]. It relies on the fundamental property $H \cdot G^T = 0$ to build an algebraic system, using then Gröbner bases techniques to solve it. The special properties of codes in quasi-dyadic form are of key importance, as they contribute to considerably reduce the number of unknowns of the system. Also, the parameters m and t come into account as they define the dimension of the solution space.

The aim is to find a valid parity-check matrix for the code, that is, a matrix H in Alternant form, $H = \{y_j x_j^i\}$; these elements are represented by two sets of unknowns $\{X_i\}$ and $\{Y_i\}$. The first step of the attack is then generating the following system of equations:

$$\left\{ g_{i,0} Y_0 X_0^j + \cdots + g_{i,n-1} Y_{n-1} X_{n-1}^j = 0 \mid i = 0, \ldots, k-1, j = 0, \ldots, \ell - 1 \right\}. \quad (2)$$

As is easy to see, the case $j = 0$ produces a set of linear equations involving only the Y_i. These can be further reduced with the help of some properties derived from the dyadicity and the key-generation algorithm [14, Pr. 5]; in particular, we have that $Y_{is+j} = Y_{is}$ for each block, i.e. $i = 0, \ldots, n_0 - 1$, $j = 1, \ldots s$ (a proof is given for the case $t = 1$; for the adaptation to the case $t > 1$ see [27]). This results in having only $n_0 - 1$ unknowns Y_i, since we can arbitrarily choose one of them. Moreover, the linear equations are identical for all the rows of each dyadic block, hence only $n_0 - mt$ distinct equations remain after eliminating the redundant ones.

As in any linear system, the difference between these two numbers gives the number of *free variables* of the system: in this case, $mt - 1$. If it is possible to recover the free variables (if the number of those is very small, even just by guessing) it is possible to reduce (2) to a simplified system involving only the X_i. Once the reduction is done, a linearization trick is used to solve and retrieve the remaining unknowns.

Hence, it is crucial to keep the dimension of the solution space (number of free variables) high enough to prevent the attack to succeed; the authors in [15] indicate that this number should be not smaller than 20. In this case in fact, the computational effort required to solve the system is too high: experimental results indicate a complexity of approximately 2^{128} bit operations.

Additional security comes from another phenomenon that occurs when the base field is $\mathbb{F}_2$. In this case the Gröbner basis necessary to solve the system is easy to compute, but somehow "trivial" (reduced to one equation) and doesn't provide enough information, hence the attack cannot be completed.