Math.Comput.Sci. (2020) 14:457–469
https://doi.org/10.1007/s11786-019-00444-1

Mathematics in Computer Science

Computer Algebra Tales on Goppa Codes and McEliece Cryptography

Narcís Sayols · Sebastià Xambó-Descamps

Received: 29 November 2018 / Accepted: 12 April 2019 / Published online: 17 December 2019

Abstract The 40-year old McEliece public-key crypto-system is revisited with the help of recently developed resources: an improved Peterson–Gorenstein–Zierler decoder for alternant error-correcting codes; PyECC, a purely Python CAS; a package of PyECC functional utilities for the computations involved in defining, coding and decoding error-correcting codes; and a Web page with free-access to companion materials. The last two tales trace the evolution of the McEliece systems and provide an account about their current security levels.

Keywords Symbolic computation · Error-correcting codes · Post-quantum cryptography

Mathematics Subject Classification Primary 68W30 · 11T71 · 94Bxx · 94B35 · 94A60; Secondary 01-02 · 01-08 · 01A65

Foreword

One of the motivations for this work was the development of a purely Python CAS environment to cover the computational needs of a book such as [62]. That prompted its extension to meet other related purposes, such as, among others, the computational tasks of the forthcoming [61] and [60].

Much encouragement came from the implementation of efficient decoders, as for instance the improved version [23] of the old Peterson-Gorenstein-Zierler decoder [27,40,41], and from other enthralling computations, like those in [33]. Further developments led to the CAS system PyECC (see [48]). The revisiting of the McEliece public-key crypto-system [30], which is based on a class of binary classical Goppa codes, was a further interesting test for these tools.

In celebration of the 50th aniversary of [4] and the 40th of [30].

N. Sayols
Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial Universitat Politècnica de Catalunya, Jordi Girona, 1-3. K2M, 08034 Barcelona, Spain
e-mail: narcissb@gmail.com

S. Xambó-Descamps (✉)
Department de Matemàtiques, Universitat Politècnica de Catalunya, Jordi Girona, 1-3. Omega, 08034 Barcelona, Spain
e-mail: sebastia.xambo@upc.edu

One friendly feature of our system is the availability of the source code as Python files or as Jupyter notebooks. [1] In the case of this paper, see [49]. As the reader will see, most source code is not provided in listings, but rather by free links to the companion materials. The Python files by themselves, with the included comments, amount to an advantageous replacement of the listings, even if the user does not know Python, but of course the full potential of such materials can only be realized by running them and, even better, by experimenting with them.

The paper is based on the material presented at the session "Computer Algebra in Coding Theory and Cryptography" (CACTC) of the 24th Conference on Applications of Computar Algebra (ACA 2018, Santiago de Compostela, June 18–22). Its main purpose is to present an overview of those developments that are more closely related to the McEliece public-key crypto-systems (McECS) and their symbolic implementation. It is structured as follows: Sect. 1, a detailed description of the mathematical ingredients concurring in a McECS; Sect. 2, a brief report on the structure and functionality of PyECC; Sect. 3, a quick survey of Goppa codes, seen as a special class of alternant codes, with emphasis on their decoding (see [23,62]) and their bearing on the effective construction and implementation of a McECS; Sect. 4, an outline of the historical evolution and current significance of McECS; Sect. 5, an account of the security levels of McECS and comments on the so-called post-quantum scenario (see [12,34]). The last section collects some conclusions, future outlook, and acknowledgments.

1 Ingredients of a McEliece System

In this section we describe the (mathematical) ingredients that concur in a McEliece criptosystem with a view to chart what will be needed to work with it from a symbolic computation perspective.

General ingredients

- $F = F_q$, a finite field of cardinal q (*base field*). The most important case will be $F = \mathbb{Z}_2$, but we cannot avoid higher cardinals because high degree extensions of the base field are required.
- k a positive integer. The vectors of F^k are called *information vectors*, or *messages*.
- $n > k$ an integer. The vectors of F^n are called *transmission vectors*. If $\boldsymbol{x} \in F^n$, we let $|\boldsymbol{x}|$ denote the number of non-zero components of $\boldsymbol{x}$ and we say that it is the *weight* of $\boldsymbol{x}$.

These ingredients are public and we assume that any user of the protocol can generate information and transmission vectors and has facilities to send transmission vectors to any other user.

Key generation

A receiving user needs the following data, where $F(r, s)$ denotes the space of matrices of shape $r \times s$ with entries in F and $F(r) = F(r, r)$:

- $G \in F(k, n)$ such that $\operatorname{rank}(G) = k$;
- $S \in F(k)$ invertible chosen at random;[2]
- $P \in F(n)$ permutation matrix chosen at random;
- t, a positive integer; and
- $g : X \to F^k$, $X \subseteq F^n$, such that for any $\boldsymbol{u} \in F^k$ and all $\boldsymbol{e} \in F^n$ with $|\boldsymbol{e}| \leq t$,

$$\boldsymbol{x} = \boldsymbol{u}G + \boldsymbol{e} \in X \text{ and } g(\boldsymbol{x}) = \boldsymbol{u}. \tag{1.1}$$

The map g is called a t-error-correcting G-*decoder*, or simply *decoder*, and the vectors of X are said to be g-*decodable*.

These data are used to define a *private key* and a *public key*, as follows:

- Private key: $\{G, S, P\}$
- Public key: $\{G', t\}$, where $G' = SGP$.

[1] http://jupyter.org/.

[2] All random choices that appear in this paper are drawn according to the uniform distribution.

Remark Although each user may prefer to work with a private decoder, a complete system should provide a *default decoder* that can be readily used by anyone (see Sect. 3.2 for information on what are the best decoders available). Moreover, it is reasonable to assume that all users can be emitters and receivers. The constraints on the system imposed by security demands will be analyzed in Sect. 5.

Encryption protocol

The protocol that a user has to follow to encrypt and send a message $\boldsymbol{u}$ to the user whose public key is $\{G', t\}$ consists of two steps:

- Random generation of a transmission vector $\boldsymbol{e}$ of weight t;
- Sending the vector $\boldsymbol{x} = \boldsymbol{u}G' + \boldsymbol{e} = \boldsymbol{u}SGP + \boldsymbol{e}$ to that user.

Decrypting protocol

Consists of four steps that only use private data of the receiver and the vector $\boldsymbol{x}$ sent by the emitter:

- Set $\boldsymbol{y} = \boldsymbol{x}P^{-1}$, so that $\boldsymbol{y} = (\boldsymbol{u}S)G + \boldsymbol{e}P^{-1}$.
- Set $\boldsymbol{x}' = g(\boldsymbol{y})$. Since P is a permutation matrix,

  $$|\boldsymbol{e}P^{-1}| = |\boldsymbol{e}| = t,$$

  and hence $\boldsymbol{x}'$ is well defined, as g corrects t errors. The result is $\boldsymbol{x}' = (\boldsymbol{u}S)G$, which says that $\boldsymbol{x}'$ is the linear combination of the rows of G with coefficients $\boldsymbol{u}' = \boldsymbol{u}S$.
- Since G has rank k, $\boldsymbol{u}'$ is uniquely determined by $\boldsymbol{x}'$ and can be obtained by solving the system of linear equations $\boldsymbol{x}' = \boldsymbol{u}'G$, where $\boldsymbol{u}'$ is the unknown vector.
- Let $\boldsymbol{u} = \boldsymbol{u}'S^{-1}$, which agrees with the message sent by the emitter.

The computacional aspects of these ingredients are considered in next section: 2.1 (Modular arithmetic), 2.2 (Univariate polynomials), 2.3 (Construction of finite rings and fields), 2.4 (Vectors and matrices), and 2.5 (Utilities for MECS).

2 The PyECC CAS

In the remaining of this section, we illustrate how PyECC works with a few general considerations and examples (cf. Fig. 1).

2.1 Modular Arithmetic

The function `ZZ()` creates the ring $\mathbb{Z}$ of integers, and `Zn(n)` creates the ring $\mathbb{Z}_n = \mathbb{Z}/(n)$ (integers modulo n). In particular we get, when n is prime, the prime finite fields. If A and B are (already constructed) rings, and there exists a "canonical" homomorphism from A to B, the image of $a \in A$ in B by that homomorphism is denoted `a » B`. For example, if m is an integer and $B = \mathbb{Z}_n$, the expression `m » B` is the class of m modulo n, that is, the image of m under the canonical homomorphism $\mathbb{Z} \to \mathbb{Z}_n$. ⋄ Modular arithmetic.[3]

2.2 Univariate Polynomials

To create the polynomial ring P with indeterminate X over an (already constructed) ring A, we have the function

```
[P,X] = polynomial_ring(A),
```

[3] These kind of entities provide links that work with the pdf version. To find their destinations when reading the paper version, go directly to the page https://mat-web.upc.edu/people/sebastia.xambo/Papers/PyACA.html.

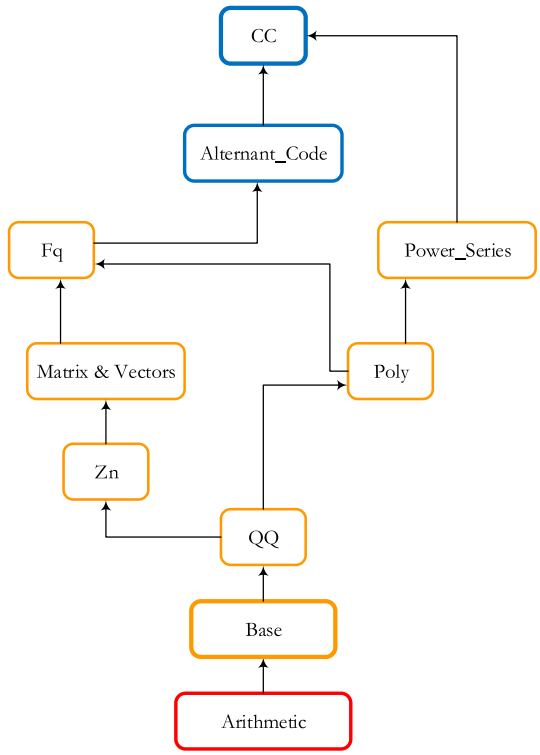

Fig. 1 Schematic architecture of PyECC, a hierarchy of Python classes driven by several function definition files. The function files are grouped in two components: Low level *arithmetic utilities* (in red) and high level interfaces (in blue). The classes lie in between and are depicted in orange. The arrows correspond to dependencies. For details about this architecture, as well as on its installation and usage, see [48]

which returns the polynomial ring $A[X]$ and binds it to P.

On the other hand, the function

```
p = get_irreducible_polynomial(P,t)
```

returns a monic irreducible polynomial $p \in P$ of degree t. Since this function iterates the random selection of a monic polynomial $p \in P$ of degree t while p is not irreducible, it is important to know the probability that a choice of p is irreducible. The result is that this probability is $\sim 1/t$, as a consequence of *Gauss formula* giving the number $I_q(t)$ of monic irreducible polynomials of degree t over F_q:

$$I_q(t) = \frac{1}{t}\sum_{d|t} \mu(t/d)q^d = \frac{q^t}{t} + \cdots,$$

where μ is the Möbius function. In fact, the probability in question is $I_q(t)/q^t = \frac{1}{t} + \cdots$, as q^t is the number of monic irreducible polynomial of degree t.

Jupyter Notebooks

We illustrate this resource with the computation of $I_q(t)$ (see Fig. 2). With the command `from PyECC import *`, the PyECC package is loaded. In the definition of the function `irr(q,m)`, `divisors` and `mu_moebius` are PyECC functions that yield the sorted list of divisors of m and the value $\mu(m//d)$ of the Möbius function for the integer quotient of m by d, which in Python is expressed as $m//d$ (remark that the value of the expression m/d is a decimal number, even when d divides m). ⋄ Computation of $I_q(m)$.

From the definition of `M` (notation from the listing), we see that `show(M)` yields the values of irr(q,m) for $1 \le m \le 7$ and $q \le 9$.

Irr_q(m): number of monic irreducible polynomils of degree m over F_q

```
from PyECC import *

def irr(q,m):
    N = 0
    D = divisors(m)
    for d in D:
        N += mu_moebius(d)*q**(m//d)
    return N//m

M=[[irr(q,m) for m in range(1,8)] for q in [2,3,4,5,7,8,9]]

show(matrix(M))
```

```
[[2     1       2       3       6       9       18]
 [3     3       8       18      48      116     312]
 [4     6       20      60      204     670     2340]
 [5     10      40      150     624     2580    11160]
 [7     21      112     588     3360    19544   117648]
 [8     28      168     1008    6552    43596   299592]
 [9     36      240     1620    11808   88440   683280]] :: Matrix[Z]
```

Fig. 2 Image of the first cell of a Jupyter notebook displaying the code for the PyECC function irr(q,m) that computes the number of monic irreducible polynomials of degree m over F_q

2.3 Construction of Finite Rings and Fields

If A is a ring, and $C = [1, a_1, \ldots, a_r]$ is a list of elements of A with first element 1, PyECC creates the ring

$$B = A[X]/(f = X^r + a_1 X^{r-1} + \cdots + a_{r-1}X + a_r),$$

and assigns the value $X \mod f$ to x, with the expression

`[B,x] = extension(A,C,'x','B').`

As said above, the ring of univariate polynomials $P = A[X]$ can be created with the expression

`[P,X] = polynomial_ring(A,'X')`

Now if $f = X^r + a_1 X^{r-1} + \cdots + a_{r-1}X + a_r \in P$, the expression

`[B,x] = extension(A,f,'x','B').`

has the same value as the previous expression `[B,x] = extension(A,C,'x','B')`. If A is a field, then B is a field if and only if f is irreducible in $A[X]$. $\diamond$ Construction of rings and fields.

2.4 Vectors and Matrices

Let A be a ring and n a positive integer. The expression `v=vector(A,n)` creates the zero vector of length n with entries from A, and assigns it to `v`. If $a \in A$ and $0 \le j < n$, the expression `v[j] = a` sets the value of the j-th component of `v` equal to a. A vector can be created from a list `a` of elements of A with the expression `vector(a)`.

Similarly, to create a matrix M of type $k \times n$ with coefficients from the ring A, the basic expression is

`M = matrix(A,k,n)`

Initially, M is identically zero, but an element $a \in A$ can be assigned to the (i, j) entry with the expression $M[i, j] = a$. More conveniently, if $a_1, \ldots, a_k$ are lists of elements of A of the same length n, the expression

`matrix` $(a_1, \ldots, a_k)$ creates the $k \times n$ matrix whose i-th row is a_i. We have seen an instance of this at work in Fig. 2, where the lists of lists M is used to get `matrix(M)` which on printing it yields a more informative arrangement than just `show(M)`.

$\diamond$ Vectors and matrices.

2.5 Construction of P and S

The description of the ingredients of a McEliece system given in Sect. 1 shows that for their implementation we require utilities that range from a rather elementary character, such as the construction of the matrices P and S needed to setup a public key, till others that are more involved, like the contruction of a matrix G with the properties sought after and the implementation of efficient encryption and decryption procedures. The latter depends on the theory of Goppa codes and will be explained in next section, 3, while in this section we focus on the construction of the matrices P and S.

For the construction of P we have the function `rd_permutation_matrix(n)`, which creates a random permutation matrix of order n. $\diamond$ Random permutation matrices.

The function `rd_GL(k,F)`, k a positive integer and F a finite field, yields a random invertible $k \times k$ matrix with entries in F. The beauty of this function is that it guarantees that the random choice is done according to the the uniform distribution. For $k = 1$, it amounts to choosing a non-zero element of F at random, which is the task of the function `rd_nonzero(F)`. No mystery in this, as the choice is done by choosing an integer j at random in the range $0..(q-1)$, $q = |F|$, and returning `element(j,F)`.

The core of `rd_GL(k,F)` involves the function `rd_extend(A)`, which creates, given an invertible matrix $A \in F(k)$, an invertible matrix $B \in F(k+1)$ *which is distributed uniformly if A has the same property*. Thus we get a random invertible matrix of order k by iterating $k-1$ times the expression `A = rd_extend(A)` launched with a random A of order 1.

The definition of `rd_extend(A)` is as an *iterative* procedure inspired in the *recursive* algorithm of D. Randall published in [44]. First, select a non-zero vector v of length $k+1$ with entries in F and let r be the index of its first non-zero component. The expression `rd_insert(A,r)` creates a matrix in $F(k, k+1)$ by inserting a random column of elements of F between the columns of indexes $r-1$ and r of A, and then an invertible matrix in $F(k+1)$ by adding a first row of 0's except for a 1 in the position of index r, and finally this matrix is multiplied on the right by the matrix obtained from the identity matrix I_{k+1} by replacing its r-th row by the vector v.

Instead of providing here a more detailed account, which you can find at [48] (Function summary/rd_GL), let us illustrate here the construction in the case of invertible matrices in $F(2)$. Initially we have a random non-zero element $a \in F$ and v can have one of the following two forms: $v = [b, y]$ or $v = [0, b]$ ($b, y \in F$, $b \neq 0$). In the first case, the procedure just outlined boils down to (setting $x \in F$) to the expression

$$\begin{pmatrix} 1 & 0 \\ x & a \end{pmatrix} \begin{pmatrix} b & y \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} b & y \\ xb & xy + a \end{pmatrix}.$$

In the second case, the expression is (again setting $x \in F$)

$$\begin{pmatrix} 0 & 1 \\ a & x \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & b \end{pmatrix} = \begin{pmatrix} 0 & b \\ a & xb \end{pmatrix}.$$

The first form yields all the invertible matrices in $F(2)$ whose first row has a non-zero element in the first position. The number of matrices obtained in this way is $m_0 = q^2(q-1)^2$. The second expression yields all the invertible matrices in $F(2)$ whose first row begins with 0. The number of such matrices is $m_1 = q(q-1)^2$. Since $m_0 + m_1 = (q^2-1)(q^2-q)$, which is the cardinal of the linear group $\mathrm{GL}(2, F)$, we see that the procedure yields the elements of this group uniformly when x is picked uniformly at random in F and a, b are chosen uniformly at random in $F^\times$ (the group of non-zero elements of F). $\diamond$ `rd_GL(k,F)`.

If it is plain, by all that has been said so far, that it would be easy to create a rank k matrix of type $k \times n$ with coefficients in F, but this would hardly advance our purposes, as G has to satisfy property (1.1). To insure this, we

need to resort to classical Goppa codes. What is in any case immediate is that the public key matrix G' is given, assuming that we already have G, as the value of the expression `S*G*P`.

3 Goppa Codes as a Class of Alternant Codes

A (*linear*) *code* of type $[n, k]_q$ over the finite field $F = F_q$ is an F-vector subspace C of dimension k of F_q^n. We say that n and k are the *length* and the *dimension* of C, respectively, and we write $C \sim [n, k]_q$. The *rate* of C is the quotient k/n. Instead of $[n, k]_2$, we simply write $[n, k]$ and we say that C is a binary code.

A matrix G of size $k \times n$ is a *generating matrix* of C if its rows form a basis of C. In the other direction, given a matrix G of size $k \times n$ with entries in F, we write $\langle G \rangle$ to denote the F-vector subspace of F^n spanned by the rows of G. Is is a code of type $[k', n]_q$, where k' is the rank of G. In the case that G is a generating matrix of C, it is clear that $\langle G \rangle = C$.

If G is a generating matrix of the code $C \sim [n, k]_q$, the linear map

$$f = f_G : F^k \to F^n, \quad \boldsymbol{u} \mapsto \boldsymbol{u}G,$$

is one-to-one and its image is C. We say that f is an *encoder* for C. The encoding of an *information vector* $\boldsymbol{u} \in F^k$ is the code vector $\boldsymbol{x} = f(\boldsymbol{u})$.

Example (The binary Hamming code [7,4]): $\boldsymbol{x} = \boldsymbol{u}G$, where

$$G = \begin{pmatrix} 1\,0\,0\,0\,1\,1\,1 \\ 0\,1\,0\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1\,0\,1 \\ 0\,0\,0\,1\,0\,1\,1 \end{pmatrix}.$$

Linear codes can also be produced by what we may call *dual construction*, which works as follows. Let H be a matrix of size $r \times n$ with entries in F_q such that $\operatorname{rank}(H) = r$. Then

$$C_H = \{\boldsymbol{x} \in \mathbb{F}^n : \boldsymbol{x}H^T = 0\} = \ker H^T$$

is a linear code of type $[n, n - r]_q$. The matrix H is said to be a *control matrix* of C_H. Note that we can form a generating matrix of C_H by choosing a basis of $\ker H^T$.

Example The matrix

$$H = \begin{pmatrix} 1\,1\,1\,0\,1\,0\,0 \\ 1\,1\,0\,1\,0\,1\,0 \\ 1\,0\,1\,1\,0\,0\,1 \end{pmatrix}$$

is a control matrix of the binary Hamming code [7, 4]. Indeed, if we let R denote the submatrix formed by the three last columns of G, then $G = I_4|R$, $H = R^T|I_3$, and

$$GH^T = (I_4|R)\begin{pmatrix} R \\ I_3 \end{pmatrix} = R + R = 0.$$

Therefore, $C = \langle G \rangle \subseteq \ker H^T$, an inclusion that in fact is an equality, because both spaces in the inclusion relation have dimension 4. For two matrices A and B with the same number of rows, `splice(A,B)` returns the splicing operation $A|B$. Similarly, if A and B have the same number of columns, `stack(A,B)` yields the stacking $\binom{A}{B}$ of A over B.

$\diamond$ The Hamming code [7, 4, 3].

Remark A generating matrix of the form $G = I_k|R$ is said to be *systematic* with respect to the components $1, \ldots, k$. Note that $\boldsymbol{x} = \boldsymbol{u}G = \boldsymbol{u}|\boldsymbol{u}R$, which shows that we can recover $\boldsymbol{u}$ by extracting the first k components de $\boldsymbol{x}$. In this

case, an argument similar to the one explained in the previous example allow us to conclude that $H = (-R^T)|I_{n-k}$, which is $R^T|I_{n-k}$ in the binary case, is a control matrix of $\langle G\rangle$.

The dual construction admits a generalization that will be needed for defining (classical) Goppa codes and which can be described as follows. Let $K = F_q$ and $F = F_{q^m}$, where m is a positive integer. Let H be an $r \times n$ matrix with entries in F with $\operatorname{rank}(H) = r$. Then we can define a code $C'_H \subseteq K^n$ by the formula

$$C'_H = C_H \cap K^n = \{\boldsymbol{x} \in K^n : \boldsymbol{x}H^T = 0\}.$$

Unlike the previous case, there is no closed formula for $k' = \dim C'_H$, and in general we only have the bounds $n - r \geq k' \geq n - rm$. Indeed, we can obtain a control matrix H' of C'_H by replacing each entry in H by the column vector formed with the m components of that entry with respect to a basis of F over K, which clearly satisfies $C'_H = C_{H'}$. Since the type of H' is $rm \times n$, we have

$$\dim C'_H = \dim C_{H'} = n - \operatorname{rank}(H') \geq n - rm.$$

The inequality $k' \leq n - r$ follows from the observation that vectors of K^n that are linearly independent over K remain linearly independent over F.

It is to be noted, nevertheless, that $k' = n - \operatorname{rank}(H')$, which means that we can compute k' in any given particular case. The Example 3.1 provides information, and an illustration, on how to solve this problem from a computational point of view.

A *decoder* of C is a surjective map $g : F^n \to C \sqcup \mathcal{E}$, where $\mathcal{E}$ is a set of "error messages", that satisfies $g(\boldsymbol{x}) = \boldsymbol{x}$ for any $\boldsymbol{x} \in C$.

The set $g^{-1}(C) \subseteq K^n$ is the set of *decodable vectors*, while $K^n - g^{-1}(C) = g^{-1}\mathcal{E}$ is the set of *error vectors*. A decoder is *complete* if $g^{-1}(C) = K^n$ (in this case $\mathcal{E} = \emptyset$).

We say that the decoder g has *correcting capacity* t if $g(\boldsymbol{y}) = \boldsymbol{x}$ for any vector $\boldsymbol{y} \in K^n$ such that $|\boldsymbol{y} - \boldsymbol{x}| \leq t$.

If we set $B(\boldsymbol{x}, t) = \{\boldsymbol{y} \in \mathbb{F}^n : |\boldsymbol{y} - \boldsymbol{x}| \leq t\}$ (this set is called the *Hamming ball* of center $\boldsymbol{x}$ and radius t), then we have $\cup_{\boldsymbol{x}\in C} B(\boldsymbol{x}, t) \subseteq D$ and $g(B(\boldsymbol{x}, t)) = \{\boldsymbol{x}\}$.

$\diamond$ Example: Conding and decoding the Hamming code [7, 4, 3].

3.1 Computing a Control Matrix

The function `blow(H,F)` computes H', so that $k' = n - \operatorname{rank}(H')$. To get a control matrix over F with linearly independent rows we can use the function `prune(`H'`)`, which supresses the rows that are in the span of the preceding ones, so that at the end we get a control matrix H'' over F with linearly independent rows. In particular, $k' = n - r''$, where $r'' = \operatorname{rank}(H'')$ is the number of rows of H''. $\diamond$ Example of `blow` and `prune`.

3.2 Decoders

PyECC has implementations of three fast decoders: PGZ, PGZm, and BMS. They decode alternant codes and in particular the classical Goppa codes. The first two are based on the Peterson-Gorenstein-Zierler algorithm [27,38,62], following [23], and BMS is an implementation of the so-called Berlekamp-Massey-Sugiyama algorithm (cf. [62]). The simplest is PGZm, as it is largely based on linear algebra over a finite field. The fastest, asymptotically, is the legendary BMS, based on the extended Euclidean algorithm for univariate polynomials over finite fields to solve the 'key equation' (which allows finding the error positions) and on a 'Forney's formula' (that delivers the error-values). The function PGZ is an hybrid of PGZm and BMS, as it uses, after determining the 'error-locator' as in PGZm, a Forney's procedure to find the errors (instead of relying on solving a linear system of equations). For most codes used in practice, however, all three qualify as efficient decoders. $\diamond$ Example of a Goppa code and its decoding.

3.3 A Toy Example

Our system does not yet have the speed that is required to process a McEliece system with the parameters that are needed to guarantee a high security level (cf. Sect. 5). But the speed can be increased by using features that Python supports, an issue that we touch briefly in Sect. 6 on pointing out possible future developments. Meanwhile, the following simple example is meant to see in detail the working of a McEliece system with the tools considered up to here.

◇ A toy illustration of the McEliece system, step by step.

4 Evolution of the McEliece Systems

The context in which McEliece proposed his system was created by the works of Diffie and Hellman [20], in which they introduced the notion of public key cryptographic system; of Merkle and Hellman [31], which proposed the system based on the difficulty of the knapsack problem;[4] and particularly of Rivest, Shamir and Adleman, [46], whose celebrated RSA system relies on the difficulty of factoring large integers (see Sect. 1).

The McEliece system was the first public key system based on the theory of algebraic codes introduced by Goppa in 1970, [26], and that we here call *classical Goppa codes* (within the class of general Goppa codes, they correspond to the codes of genus 0).[5] Let us also mention that the parameters chosen by McEliece to illustrate various aspects of his system were $[n, k] = [1024, 524]$ and $t = 50$, which guaranteed a security of more than 60 bits, considerable 40 years ago, but not, as we will see in the next section, in today's circumstances.

A modification of the McEliece system was proposed by Niederreiter in 1986, [35]. The private key of this system is the triple $\{H, S, P\}$ of binary matrices of shapes $n \times (n-k)$, $(n-k) \times (n-k)$, and $n \times n$, respectively. These matrices have have to satisfy the following conditions: H is a control matrix of a binary classical Goppa code $\Gamma \sim [n, k]$ (thus $y \in \Gamma$ if and only if $yH^T = 0$) that can correct t errors; S is a non-singular matrix chosen uniformly at random; and P is a permutation matrix, also chosen uniformly at random. The corresponding public key is the pair $\{H', t\}$, where $H' = PHS$. The messages u are binary vectors of length n and weight t. The encrypted message is $x = uH' = uPHS$. The receiver recovers u as follows: (1) Get $\boldsymbol{s} = xS^{-1} = uPH$; (2) Knowing H, can produce $u' = uP = D(\boldsymbol{s}, x)$ with a suitable decoder D (Patterson's algorithm, [38], which can be considered as a variant or the standar decoders BMS or PGZ); and (3) Since also knows P, can find $u = u'P^{-1}$, which is the sent message. The parameters chosen for the initial illustration were $[n, k] = [1024, 644]$, $t = 38$. For the efficient generation of messages of length n and weight t, see [50].

In the study of the McEliece and Niederreiter systems presented in Sendrier's 2002 habilitation memoir, [51], we read: "After more than 20 years of efforts [and cites over a dozen works], no cryptanalysis has has been able to break these systems". As we will see, this judgment continues to be valid today.

The problem of modifying the McEliece and Niederreiter systems in order that they are able to work with substantially smaller public keys, is considered by Gaborit in [24]. Serve as an illustration of the result, the following facts: for lengths 2047 and 4095, public keys of 12 Kb and 20 Kb, respectively, are sufficient. For more details of how matters stood one decade ago, see [22]. According to the autors, "code based cryptography is an interesting alternative to number theoretic cryptography", as "many basic cryptographic functions [...] can be realized using code thoretic concepts".

Let us also mention [12], which in particular proposes new parameters for the McEliece and Niederreiter systems that achieve good security levels against all known attacks and with considerably smaller key lengths. The authors also state that "Quantum computers do not seem to give any significant improvements in attacking code-based systems, beyond the generic improvements possible with Grover's algorithm, and so the McEliece encryption scheme is one of the interesting candidates for post-quantum cryptography" (for the Grover algorithm, and refereces to it, see [47], Sect. 5).

[4] Note that this system was broken in 1982 by A. Shamir, see [53].

[5] McEliece does not quote Goppa's article and instead refers to pages 179–180 and 193–194 of [29].

Although the notion of post-quantum cryptography was introduced, as we have already said, in 2003, in our reckoning the most visible start occurs when to the publication of the magnificent conventional cryptography manual [28] followed, 1 year later, the volume [10]. In this volume, the first overview of this new perspective on cryptography, the outstanding introductory chapter by D. J. Bernstein [7] outlines, in particular, an account of the four paradigms that are considered as more promising. In addition to the code-based cryptography (studied in the chapter by Overbeck and Sendrier, [37]), we find systems based on multivariate polynomials, on lattices (see [57] for a recent article on this question), an on hash functions. In the Bernstein's paper cited above we find, just before the presentation of the four paradigms, the *leitmotiv* of post-quantum cryptography: "there is no justification for the leap from "quantum computers destroy RSA and DSA and ECDSA" to "quantum computers destroy cryptography." There are many important classes of cryptographic systems beyond RSA and DSA and ECDSA". In fact, in the chapter by Overbeck and Sendrier quoted above, we read: "Three decades later, some parameter adjustment[s] have been required, but no attack is known to represent a serious threat on the [McEliece] system, even on a quantum computer".

These views have been repeated, and even reinforced, in recent years. We mention a sample of works that go deeper in several directions: [13,15] (this article proposes new parameters to achieve a good level of security against all known attacks), [3,32,34] (doctoral thesis) [39] (another doctoral thesis, in which the generalized Srivastava codes are revitalized, and with effective implementations), [19,45,58].

Throughout this post-quantum decade, a pioneer to take into account is D. J. Bernstein, who has already been cited and of whom we recommend his website [6]. It is also worth visiting the page that he publishes with T. Lange, [43], and follow the PQC 2018 conference, focused on achieving a standardization of post-quantum cryptographic protocols and about which you can find information in [42].

5 Security Analysis and the Post-quatum Scenario

McEliece himself considered the issue of the security that his system could offer [30]. In a general approach, we assume that a eavesdropper, let's say E, has seized the encrypted vector x sent by a user and knows the public key of the recipient, namely $G' = SGP$ and t. What possibilities does E have of obtaining the initial message u?

Trying to decode $x = uG' + e$ using the known generating matrix G' can hardly be considered promising, if k is large, because the problem of decoding linear codes $[n, k]$ is NP-complete (cf. [5]).

The other attack that McEliece considered to his system is more surprising. Suppose the eavesdropper selects k coordinates from x at random. If E is lucky, none of these coordinates will be in error and hence will be able to determine u by solving the system of linear equations $\bar{x} = u\bar{G}'$, where $\bar{x}$ is the vector formed with the selected entries of x and $\bar{G}'$ is the submatrix of G' formed with the columns whose indexes correspond to those of $\bar{x}$. Otherwise, at least one of the selected entries will be erroneous and the linear system will have no solution. This attack is more serious than what may appear to be in a first glance. Indeed, to decide whether the linear equations $\bar{x} = u\bar{G}'$ have a solution, and to find a solution if it exists, amounts to a number of operations of the order k^3, while the probability of choosing k error-free entries is

$$\binom{n-t}{k}/\binom{n}{k} = (1 - \tfrac{t}{n})(1 - \tfrac{t}{n-1}) \dots (1 - \tfrac{t}{n-(k-1)}) < (1 - t/n)^k,$$

so that the expected number of operations that E will have to carry out is higher than $k^3/(1 - t/n)^k$. With the original parameters of McEliece, this amounts to an effort of not less than 2^{64} operations, a fact that is usually expressed by saying that the system achieves a level of 64 security bits (relative to the attack in question).

Figure 3 summarizes the security level of the McEliece system at present. We have added to the original graphic the gray lines to facilitate the comparison with the security level of the original system. We see that taking into account all known attacks it still provides 63 security bits, and that with a slightly higher rate (0.625 instead of $524/1024 = 0.512$) we get, for the same length, 64 security bits, in remarkable agreement with the initial analysis. Sendrier also remarks that the McEliece and Niederreiter systems are significantly easier to implement, for a comparable security level, than most of the other systems, but that they require a relatively longer keys. The discovery of a digital signature scheme based on McEliece by the same author is also mentioned [18].

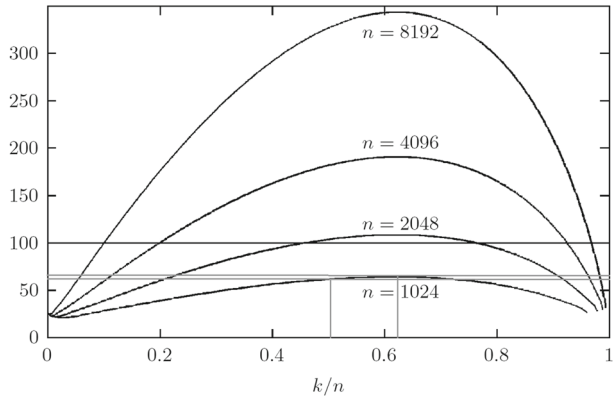

Fig. 3 Security level in bits of the McEliece system as a function of of the transmission rate of the code, k/n, and by the code lengths $n = 2^l$ ($l = 10, \ldots, 13$). The value is quoted as the $\log_2$ of the *work factor* (number of computational units) of the strongest known attack. Graphic adapted from Figure 6.2 in [51]

The references that follow reveal that Sendrier's results on the degree of security remain essentially valid, often with variations in the codes used: [1,2,11,21,25,32,52,56].

The article [2] consigns the initial recommendations of the PQCrypto project, already mentioned, in terms of confidence in the security of the systems, rather than in their efficiency. For example, for McEliece systems based on classical binary Goppa codes, the parameters proposed to achieve 128 bits of security are $n = 6960$, $k = 5413$ and $t = 119$. The 128-bit security is less than the 172 bits given by the McEliece count that we discussed earlier, which is explained by the fact that the estimates take into account all the currently known attacks. For more details about the McEliece system proposed by Bernstein et al., and also about the Niederreiter variant, see [36].

We finish the section with some references on advances in quantum computing that can help to understand why this resource, even with all the power imaginable in the future, may not be as powerful as it has been estimated in past times and that it will probably never be enough to break some of the systems that, like McEliece, have the potential to be really post-quantum: [8,9,14,17].

It should be mentioned, however, that the pioneering work of Shor [54] contributed to promote the field of *quantum cryptography*, whose purpose is to use the quantum properties of systems to design cryptographic systems. For an elementary description of these systems, see [16], and the bibliography therein. For a general overview, which includes the relationship with post-quantum cryptography, see [36,59] and the already quoted [8,10,11].

6 Conclusions and Next Steps

We believe that having access to the source code of a computer algebra system like PyECC, and to its use as illustrated in this paper, may be appealing to people that like to advance in the understanding of the conceptual structures by means of carefully experimenting with them by means of computational environments that properly reflect those structures. This may be enough for students, teachers, and researchers whose main concern is not computation by itself, but rather as a means to better come to grips with theoretical materials. In fact, the code may be helpful at different levels of involvement: as listings that indicate how the theoretical algorithms may be translated into programs, as a tool for carrying on meaningful computational activities, or, to the most undertaking, as a platform for tinkering with the system with a view to improving and extending it.

One important task ahead is increasing the speed of PyECC by all possible means, an endeavor that we are beginning by systematically coding the low level powerhouse routines in cyton. A benchmark for this will certainly be the coding of McEliece systems with the parameters required for real post-quantum cryptography.

Acknowledgements We are grateful to Irene Márquez Corbella and Emilio Suárez Canedo, organizers of the special session on "Computer Algebra in Coding Theory and Cryptography" of the 24th Conference on Applications of Computar Algebra (Santiago de Compostela, June 18–22, 2018), for accepting our proposed talk, and to Ignacio Luengo Velasco for many enlightening discussions about post-quantum cryptography.

References

1. Aragon, N., Gaborit, P., Hauteville, A., Tillich, J.-P.: Improvement of generic attacks on the rank syndrome decoding problem. https://hal.archives-ouvertes.fr/hal-01618464 (2017)
2. Augot, D., Batina, L., Bernstein, D.J., Bos, J., Buchmann, J., Castryck, W., Dunkelman, O., Güneysu, T., Gueron, S., Hülsing, A., Lange, T., Mohamed, M.S.E., Rechberger, C., Schwabe, P., Sendrier, N., Vercauteren, F., Yang, B.Y.: Initial recommendations of long-term secure post-quantum systems. PQCRYPTO / ICT-645622 / Horizon 2020 (2015). https://pqcrypto.eu.org/docs/initial-recommendations.pdf
3. Barbier, M., Barreto, P.S.L.M.: Key Reduction of McEliece's cryptosystem using list decoding. In: Kuleshov, A., Blinovsky, V.M., Ephremides, A., (eds), International Symposium of Information Theory (ISIT), Saint-Peterburg, Russia. IEEE, pp 2657–2661 (2011). https://hal.inria.fr/inria-00565343/file/preprint.pdf
4. Berlekamp, E.: Algebraic Coding Theory. McGraw-Hill, 1968. Revised edition by Aegean Park Press in 1984, revised edition by World Scientific Publishing Co. in 2015, with a new Preface
5. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.: On the inherent intractability of certain coding problems. IEEE Trans. Inf. Theory **24**(3), 384–386 (1978)
6. Bernstein, D.J.: Index of formal scientific papers (2018). https://cr.yp.to/papers.html. Consulted on 1st February
7. Bernstein, D.J.: Introduction to post-quantum cryptography. In: Post-Quantum Cryptography. Springer, pp. 1–14. Introductory chapter of [10] (2009). http://www.pqcrypto.org/www.springer.com/cda/content/document/cda_downloaddocument/9783540887010-c1.pdf
8. Bernstein, D. J.: Is the security of quantum cryptography guaranteed by the laws of physics? (2017). https://sidechannels.cr.yp.to/qkd/holographic-20160326.pdf
9. Bernstein, D.J., Biasse, J.F., Mosca, M.: A low-resource quantum factoring algorithm. In: Lange, T., Takagi, T., (eds.) Post-Quantum Cryptography-8th International Workshop, PQCrypto 2017, Lecture Notes in Computer Science 10346. Springer, pp. 330–346 (2017). Proceedings of the PQCrypto 2017 workshop held at Utrecht, the Netherlands, June 26-28. https://cr.yp.to/papers/grovernfs-20170419.pdf
10. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-Quantum Cryptography. Springer, Berlin (2009)
11. Bernstein, D.J., Lange, T.: Post-quantum cryptography -dealing with the fallout of physics success. Nature **549**, 188–194 (2017)
12. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. In: Buchmann, J., Ding, J., (eds) Post-Quantum Cryptography, Number 5299 in Lecture Notes in Computer Science, Proceedings of the Second international Workshop PQCrypto 2008, October 17–19, Cincinnati, OH, USA. Springer pp. 31–45 (2008)
13. Bernstein, D.J., Lange, T., Peters, C.: Wild McEliece incognito. In: Yang, B.Y., (eds) Post-quantum Cryptography–4th International Workshop, PQCrypto 2011, Lecture Notes in Computer Science 7071, Proceedings of the PQCrypto 2011 Workshop Held at Taipei, Taiwan, November 29–December 2, 2011. Springer, pp 244–254 (2011) . https://cr.yp.to/codes/wild2-20110915.pdf
14. Bernstein, D.J., Yang, B.Y.: Asymptotically faster quantum algorithms to solve multivariate quadratic equations (2017). https://cr.yp.to/papers/groverxl-20171215.pdf
15. Biswas, B.: Implementational aspects of code-based cryptography. PhD thesis, L'École Polytechnique / INRIA (2010). https://hal.archives-ouvertes.fr/pastel-00523007/
16. Cameron, P.J.: Notes on Cryptography (2013). http://www.maths.qmul.ac.uk/~pjc/notes/crypt.pdf
17. Campbell, E.T., Terhal, B.M., Vuillot, C.: The steep road towards robust and universal quantum computation (2016). https://arxiv.org/pdf/1612.07330.pdf
18. Courtois, N.T., Finiasz, M., Sendrier, N.: How to achieve a McEliece-based digital signature scheme. In: International Conference on the Theory and Application of Cryptology and Information Security. Springer, pp. 157–174 (2001)
19. Deneuville, J.-C., Gaborit, P., Zémor, G.: Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In: Lange, T., Takagi, T., (eds), Post-Quantum Cryptography, Number 10346 in Lecture Notes in Computer Science. Proceedings of the International Workshop PQCrypto 2017, pp 18–34 (2017). https://rd.springer.com/chapter/10.1007%2F978-3-319-59879-6_2
20. Diffie, W., Hellmann, M.: New directions in cryptography. IEEE Trans. Inf. Theory **22**, 644–654 (1976)
21. Dinh, H., Moore, C., Russell, A.: McEliece and Niederreiter cryptosystems that resist quantum Fourier sampling attacks. In: Rogaway, P. (ed) CRYPTO 2011. Springer, pp. 761–779 (2011). https://www.iacr.org/archive/crypto2011/68410758/68410758.pdf
22. Engelbert, D., Overbeck, R., Schmidt, A.: A summary of McEliece-type cryptosystems and their security. J. Math. Cryptol. **1**(2), 151–199 (2007)
23. Farré, R., Sayols, N., Xambó-Descamps, S.: On the PGZ decoding algorithm for alternant codes. Comput. Appl. Math. (in press) (2018). arXiv:1704.05259
24. Gaborit, P.: Shorter keys for code based cryptography. In: Proceedings of Workshop on Codes and Cryptography, pp. 81–90 (2005)
25. Gaborit, P., Ruatta, O., Schrek, J.: On the complexity of the rank syndrome decoding problem. IEEE Trans. Inf. Theory **62**(2), 1006–1019 (2016). https://arxiv.org/pdf/1301.1026.pdf
26. Goppa, V.D.: A new class of linear correcting codes. Probl. Pederachi Inf. **6**(3), 24–34 (1970). (in Russian)
27. Gorenstein, D., Zierler, N.: A class of error-correcting codes in p^m symbols. J. Soc. Ind. Appl. Math. **9**(2), 2007–214 (1961)
28. Katz, J., Lindell, Y.: Modern Cryptography. Cryptography and Network Security. Chapmann & Hall/CRC, London (2008)
29. McEliece, R.J.: The Theory of Information and Coding Volume 3 of The Encyclopedia of Mathematics and its Applications. Addison-Wesley, Boston (1977)

30. McEliece, R. J.: A public-key cryptosystem based on algebraic coding theory. Jet Propulsion Laboratory, DSN Progress Report 42-44 (1978). http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.pdf
31. Merkle, R.C., Hellman, M.E.: Hiding information and signatures in trapdoor knapsacks. IEEE Int. Symp. Inf. Theory **24**(5), 525–530 (1978)
32. Misoczki, R., Tillich, J.-P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes (2012). https://eprint.iacr.org/2012/409.pdf
33. Molina, S., Sayols, N., Xambó-Descamps, S.: A bootstrap for the number of $\mathbb{F}_{q^m}$-rational points on a curve over $\mathbb{F}_q$ (2017)
34. Niebuhr, R.: Attacking and Defending Code-based Cryptosystems. Ph.D. thesis (2012)
35. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Probl. Control Inf. Theory **15**, 159–166 (1986)
36. NISTIR-2018. Report on post-quantum cryptography (2018). https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf
37. Overbeck, R., Sendrier, N.: Code-based cryptography. In: Post-Quantum Cryptography. Springer. See [10], pp. 95–146 (2009)
38. Patterson, N.J.: The algebraic decoding of Goppa codes. IEEE Trans. Inf. Theory **21**(2), 203–207 (1975)
39. Persichetti, E.: Improving the Efficiency of Code-Based Cryptography. Ph.D. thesis, Department of Mathematics, University of Auckland, New Zeeland (2012). https://www.math.auckland.ac.nz/~sgal018/EdoardoPhD.pdf
40. Peterson, W.W.: Encoding and error-correction procedures for the Bose–Chaudhuri codes. IRE Trans. Inf. Theory **6**, 459–470 (1960)
41. Peterson, W.W., Weldon, E.J.: Error-Correcting Codes (2nd edition). MIT Press, Boston (1972)
42. Post-Quantum Cryptography 2018. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. First PQC Standardization Conference organized by the NIST Computer Security Resource Center
43. Post-quantum cryptography (2018). https://pqcrypto.org/
44. Randall, D.: Efficient generation of random nonsingular matrices, 1991. Report No. UCB/CSD-91-658 (November 1991). https://www2.eecs.berkeley.edu/Pubs/TechRpts/1991/CSD-91-658.pdf
45. Repka, M., Zadaj, P.: Overview of the McEliece cryptosystem and its security. Tatra Mt. Math. Publ. **60**, 57–83 (2014)
46. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosistems. Commun. ACM **21**, 120–126 (1978)
47. Rué, J., Xambó-Descamps, S.: Introducció matemàtica a la computació quàntica. Butlletí de la Societat Catalana de Matemàtiques, 28(2), 183–231 (2013). English version available at https://mat-web.upc.edu/people/sebastia.xambo/QC/qc.pdf
48. Sayols, N., Xambó-Descamps, S.: A Python package for the construction, coding and decoding of error-correcting codes (2017). https://mat-web.upc.edu/people/sebastia.xambo/PyECC.html
49. Sayols, N., Xambó-Descamps, S.: Computer algebra tales on Goppa codes and McEliece cryptography: Python sources and companion materials (2018). https://mat-web.upc.edu/people/sebastia.xambo/Papers/PyACA.html
50. Sendrier, N.: Efficient generation of binary words of given weight. In: Proceedings of the 5th IMA Conference on Cryptography and Coding. Springer, pp. 184–187 (1995). https://link.springer.com/chapter/10.1007/3-540-60693-9_20
51. Sendrier, N.: Cryptosystmes à clé publique basés sur les codes correcteurs d'erreurs. Université Pierre et Marie Curie, Institut National de Recherche en Informatique et Automatique, INRIA Rocquencourt. Mémoire d'habilitation à diriger des recherches (2002)
52. Sendrier, N.: On the use of structured codes in code based cryptography. In: Nikova, S., Prenell, B., Storme, L., (eds), Coding Theory and Cryptography III. Contactforum. Koninklijke Vlaamse Academie van België voor Wetenschaeppen en Kunsten. pp. 59–68 (2009). https://www.rocq.inria.fr/secret/Nicolas.Sendrier/PDFs/Sen10c.pdf
53. Shamir, A.: A polynomial-time algorithm for breaking the basic Merkle–Hellman cryptosystem. IEEE Trans. Inf. Theory **30**(5), 699–704 (1984)
54. Shor, P.: Algorithms for quantum computation: Discrete logarithms and factorization. In: Symposium of Foundations on Computer Science (Santa Fe, New Mexico, 1994). For a revised and expanded version of this paper, see [53] (1994)
55. Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**, 1484–1509 (1997)
56. Tillich, J.-P.: The decoding failure probability of MDPC codes (2018). https://arxiv.org/pdf/1801.04668.pdf
57. Wang, J.: Quantum resistant random linear code based public key encryption scheme RLCE. In: Proceedings IEEE ISIT 2016, pp. 2519–2523 (2016). https://eprint.iacr.org/2015/298.pdf
58. Wang, J.: Decoding generalized Reed-Solomon codes and its application to RLCE encryption schemes (2017). https://arxiv.org/pdf/1702.07737.pdf
59. Wikipedia. Quantum cryptography (2018). https://en.wikipedia.org/wiki/Quantum_cryptography
60. Xambó-Descamps, S., Miret, J.M., Sayols, N.: Intersection Theory and Enumerative Geometry–A Computational Primer. Springer, Berlin (2019)
61. Xambó-Descamps, S., Sayols, N.: Error-Correcting Codes. A Computational Primer. Universitext. Springer, 2019. 2nd edition of [60], with Three New Chapters, Corrections, and an Extended Introductory Chapter. Companion materials, including source code, available at https://mat-web.upc.edu/people/sebastia.xambo/PyECC.html
62. Xambó-Descamps, S.: Block Error-correcting Codes: A Computational Primer. Universitext, Springer (2003)