McEliece's Public Key Cryptosystem based on Non-Binary Algebraic Codes

Hugo Rodriguez*, Jose Alejandro Perez†, Ismael Soto†, Cesar Azurdia†, Carolina Lagos‡ and Ivan Derpich*
*Department of Industrial Engineering, University of Santiago of Chile, Chile
hugo.rodriguez@usach.cl
†Department of Electrical Engineering, University of Santiago of Chile, Chile
{jose.perezb,ismael.soto,cazurdi}@usach.cl
‡Faculty of Management and Economics, University of Santiago of Chile, Chile
carolina.lagos@usach.cl

***Abstract*—In this paper we carried out the construction of a public key cryptosystem based on algebraic geometry codes. Basically, we have developed the scheme proposed by McEliece. However, instead of using binary linear codes, we have worked with non-binary linear algebraic geometry codes on Hermitian curves. In the decryption process, we have used Sakata's algorithm, which plays an important role. An application is proposed for this system in order to be applied in a storage and retrieval systems for magnetic recording over nanoparticles.**

***Index Terms*—McEliece criptosystem, public key, algebraic geometry, coding, non-binary, Sakata.**

I. Introduction

In data communication as well as for safeguarding information, a wide range of modern cryptosystems are based on NP-hard mathematical problems. Considering public key cryptography, these problems incorporate some kind of one-way function having quite simple forward computation. However, determining its inverse is a extremely difficult issue to solve by using current computers, and therefore this constitutes the strength of these cryptosystems. However the latter is changing because of the distributed computing, and the development of increasingly faster processors. This new scenery is forcing the modern cryptographic schemes to improve their parameters in order to keep the same security standards.

Furthermore, with the advent of quantum computing, many of the current cryptographic schemes could become obsolete over time. In [4], the authors warn that quantum computers might break cryptosystems like RSA and ElGamal. Quantum computers will eventually be able to overcome the current difficulty of performing factorization of big integers, and extracting the discrete logarithms. It is also stated that McEliece cryptosystem would resists attacks of quantum computing. Moreover, it can be seen in [5] that the McEliece cryptosystem is presented as one of the possible candidates for post-quantum cryptography. The McEliece cryptosystem was introduced in [1] as the first public key encryption scheme based on coding theory. It uses the syndrome decoding problem which is NP-complete [2]. This paper deals with cryptosystems based on error correcting codes. These systems, such as McEliece, currently have implementation problems related to performance. However, in this paper it is assumed that in the future these problems will be overcome.
The remainder of this paper is organized as follows. In Section II an introduction to the McEliece public key encryption scheme is given. In Section III the algebraic tools for coding and decoding the algebraic codes are provided. In Section IV the algorithmic complexity of the cryptography scheme is computed, considering only the most complex calculations. In Section V two simple examples are given. In Section VI the result analysis is presented. Finally, in Section VII conclusions are presented.

II. System Description

Fig. 1 shows a block diagram of a public key cryptosystem (PKC). The system works with two keys generated by the receiver, one public and another private. The sender encrypts the message with the public key of the receiver while the receiver decrypt it by using his private key. The foundation of the system lies on the so called *one-way functions*.

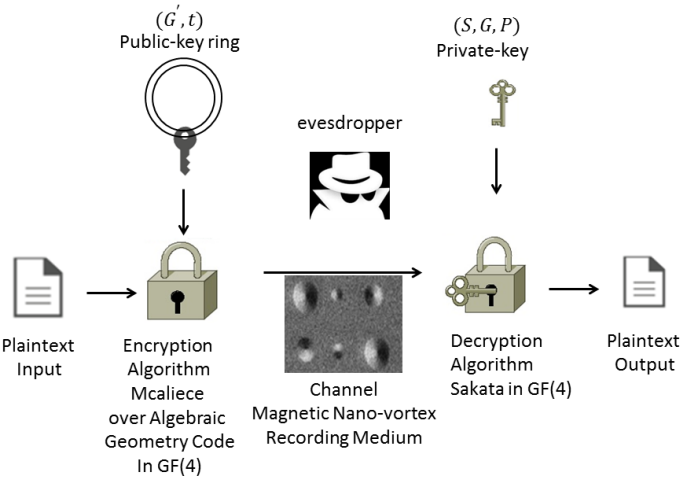

Fig. 1: Block diagram of a public key cryptosystem

To perform the encryption function, a well known and efficient algorithm, is used. However, the algorithm to compute the decryption function is NP-Hard, unless the extra needed information, i.e., the private key, is available.

Although an eavesdropper could intercept the encrypted message, without the private key it could not be decrypted. Among the one-way functions used in cryptography, the discrete logarithm problem, the big integer factorization problem and the knapsack problem are the most frequent. All of these are considered NP-hard mathematical problems. Something different was proposed by McEliece in 1978 when he developed an encryption scheme based in algebraic codes [1]. The original scheme used a binary Goppa code. The basic idea is to take a message and disguise it, in such a way an eavesdropper is faced to the syndrome decoding problem. On the other hand, the key's owner by using a decoding algorithm can easily remove the disguise and recover the message. Let $C(n, k, d)$ be an linear code having a decoding algorithm capable to correct a maximum of t errors, where n, k, d are codeword length, message length and code distance, respectively. We describe this code by its $k \times n$ generator matrix G. In order to create the disguise, two further matrices are defined. The $k \times k$ non-singular matrix S, called "the scrambler", and the $n \times n$ permutation matrix P which have a single 1 by row and by column, and 0's elsewhere. From these, a new generator matrix G' is built, hiding the original structure of G:

$$G' = SGP \tag{1}$$

The resulting code generated by G', is a linear code having the same properties of rate and distance as the original code generated by G [1]. The matrix G' is shared as public key while S, G and P are kept secret. If someone is interested in sending an encrypted message, must split the message into blocks (i.e., vectors of length k) and encrypt each, separately, before to be sent.

In the encryption process, a message block x is multiplied by G' and a random error vector e, of weight t and length k, is added. The resulting vector y, is the encrypted block to be sent:

$$y = xG' \oplus e. \tag{2}$$

An eavesdropper who intercepts y, would have to find the nearest codeword of the code generated by G'. This would involve calculating the syndrome of y and comparing it to the syndromes of all the error vectors of weight t. Since there are $\binom{n}{t}$ of these error vectors, the difficulty of proper choosing the value of n and t turns this computation in infeasible. On the other hand, the legitimate receiver performs the calculation

$$yP^{-1} = (xG' \oplus e)P^{-1} = xSG \oplus eP^{-1} = xSG \oplus e' \tag{3}$$

where e' is a vector of weight t, and P^{-1} is the inverse matrix of P. After this, the decoding algorithm is applied to remove the error vector e' in order to recover the scrambled codeword $(xS)G$. Finally, the original message x is obtained by multiplying xS on the right by S^{-1}.

The McEliece cryptosystem based on algebraic geometry curves [14] can be applied to a typical transmission system as well as storing and retrieving information in memory channels. In the Fig. 1 an example is given where the channel is made up of a nano magnetic media composed of magnetic nanodisck [13] capable of store quaternary information. The cryptosystem includes channel coding and security functions that bring up resistance to burst errors and data protection at simultaneously.

A. Non-Binary Extension

McEliece's PKC was originally defined on binary Goppa codes. However, we can extend it to non-binary general algebraic geometry (AG) codes. Even more, any linear code with a good decoding algorithm can be used. To construct a non-binary AG code, we can use the Justesen's simplified method [6], which is clearly illustrated in [8] and does not require a thorough understanding of algebraic geometry. Basically, we need an irreducible affine smooth curve over a particular finite field $\mathbb{GF}(q)$, and an ordered monomial basis $\{1, x, y, x^2, xy, \ldots\}$. Hermitian curves, defined for quadratic finite fields in the form $q = w^2$ are the most commonly used since they generate a highest number of affine points and, therefore, longer codes can be achieved for a given genus. Once we know all affine points, starting from these we can construct the generator matrix, involved in the key generation process.

A Hermitian curve over $\mathbb{GF}(q)$, where $q = w^2$, in its bivariate form is given by

$$h(x, y) = x^{w+1} + y^w + y \tag{4}$$

where $m = w + 1$ is the degree of $h(x, y)$ and its genus g is

$$g = \frac{w(w-1)}{2} \tag{5}$$

The degree and genus of the curve determine the parameters of the AG code, namely, its length, message length and minimum distance. The code length is determined by the number of affine points $n = w^3$ that satisfy $h(x, y) = 0$. The message length k and the designed distance d^* of the code are determined by the genus g of the curve.

By using Justesen's simplified construction, a nonnegative integer j must be chosen such as

$$m - 2 \leq j \leq \left\lfloor \frac{n-1}{m} \right\rfloor \tag{6}$$

Once j has been chosen, the dimension k of the code is calculated as

$$k = n - mj + g - 1 \tag{7}$$

and the designed distance d^* of the code is

$$d_{min} \geq d^* = mj - 2g + 2 \tag{8}$$

The maximum number of errors the code can correct is given by

$$t = \left\lfloor \frac{(d^* - 1)}{2} \right\rfloor \tag{9}$$

The generator matrix G is constructed from an ordered monomial basis F in the form

$$F = \{x^i y^j \mid (i,j)\ ordered\ by\ <_T; 0 \leq j < w\}. \tag{10}$$

The rows of the generator matrix G are constructed by evaluating the monomials in (10) at each affine point $P_i, i = 0, \ldots, n$ that satisfy (4). The resulting matrix is non-systematic although it can be transformed to the systematic form by using the well-known Gauss-Jordan method or similar. The systematic form facilitates the recovery of the original message at the end of the decoding process. The order of the matrix G is k rows by n columns. The matrix G forms the private key system along with the matrices S and P. We note that, the order of G determine the order of S and P. The scrambler matrix S is a square matrix of order k whose elements belong to the finite field $\mathbb{GF}(w^2)$ as well as the elements in G. On the other hand, the permutation matrix P is a square matrix of order n whose elements belong to the set $\{0, 1\}$. Then the matrix $G' = SGP$ represent the public key to hide the structure of G.

B. Decoding an AG Code

In the decryption process we have a hard decoding problem. For non-binary AG codes this process involves two important phases, namely, finding the error locations and determining the error magnitudes at these locations. In order to perfom these tasks, we use the algorithms 3 and 4 to determinate error positions and the inverse discrete Fourier transforms(IDFT) to find the error magnitudes, respectively. These algorithms are based in the Sakata's algorithm [11]. This returns a minimum set of error-locating polynomials from a syndrome set, arranged as bivariate sequence in a graded order given into an array S.

A syndrome $S_{a,b}$ corresponding to the coordinate (a, b) of the array is defined by

$$S_{a,b} = \sum_{i=1}^{n} r_i x_i^a y_i^b \tag{11}$$

where r_i is the i-*th* non-binary symbol in the received word and n is the length of the code. For an Hermitian AG code we must consider three types of syndromes: known, implied and missing. From this array we can generate a set F of bivariate monic polynomials

$$f^{(i)}(x,y) = \sum_{(k,l) \leq_T \left(t_1^{(i)}, t_2^{(i)}\right)} f_{k,l}^{(i)} x^k y^l \tag{12}$$

where $f_{k,l}^{(i)}$ are the coefficients of the terms $x^k y^l$ in the ith polynomial in F, and $t_1^{(i)}, t_2^{(i)}$ are the powers of x and y respectively, of the leading term of $f^{(i)}(x, y)$.

The polynomial $f^{(i)}(x, y)$ is a connection polynomial which forms recursive relations among the syndromes $S_{a,b}$ if its coefficients satisfying

$$\sum_{(k,l) \leq_T \left(t_1^{(i)}, t_2^{(i)}\right)} f_{k,l}^{(i)} S_{a-t_1^{(i)}+k, b-t_2^{(i)}+l} = 0 \tag{13}$$

For Hermitian codes with $m = w + 1$ the recursive relation becomes

$$S_{a,b} = S_{a-m,b+1} + S_{a-m,b+m-1} \tag{14}$$

The equation (11) is used to compute the known syndromes $S_{a,b}$ by filling the array S for all $(a, b) \mid a + b \leq j$. Likewise the equation (14) is used to compute the implied syndromes $S_{a,b} \mid (a, b)$ is ranging from $(j + 1, 0)$ to $(m, j + 1 - m)$ in total order.

The unknown syndromes $S_{a,b}, a < m$, are determined using the majority voting scheme described in [7] [9]. This majority voting is used by Sakata's algorithm when performing hard-decision decoding of AG codes.

In order to find a new unknown syndrome, a candidate syndrome value, v_i related to $f^{(i)}(x, y)$ can be calculated as

$$\sum_{(k,l) \leq_T \left(t_1^{(i)}, t_2^{(i)}\right)} f_{k,l}^{(i)} S_{a+k-t_1^{(i)}, b+l-t_2^{(i)}} = -v_i \tag{15}$$

When is not possible to use (16), alternatively the candidate syndrome can be calculated as

$$\sum_{(k,l) \leq_T \left(t_1^{(i)}, t_2^{(i)}\right)} f_{k,l}^{(i)} S_{a+k+m-t_1^{(i)}, b-m+1-t_2^{(i)}} - S_{a,b-m+2} = w_i \tag{16}$$

when both formulas are feasible to use, one of them must be elected. On contrary, if neither of them is satisfied then the polynomial $f^{(i)}(x, y)$ is not used to determine a candidate syndrome.

The discrepancy of each polynomial $f^{(i)}(x, y)$ for the syndrome $S_{a,b}$ is

$$\delta_{S_{a,b}}^{f^{(i)}(x,y)} = \sum_{(k,l) \leq_T \left(t_1^{(i)}, t_2^{(i)}\right)} f_{(k,l)}^{(i)} S_{(a+k-t_1^{(i)}, b+l-t_2^{(i)})} \tag{17}$$

If all $f^{(i)}(x, y)$ for a particular syndrome $S_{a,b}$ have no discrepancy then the process continues incrementing (a, b) with respect to the total order $\leq_T$. Otherwise new values for $f^{(i)}(x, y)$ must be determined. The full algorithm can be read from [7], [9]. The most relevant concepts we need to know about the updating process are the following

III. Non-Binary Cryptography Scheme

Now we introduce the non-binary public-key cryptography scheme used in this work.

A. Public-Key Generation Process

In [1] McEliece defined a public-key cryptosystem scheme on $\mathbb{GF}(2^m)$ where m is the length of the binary code having correction capacity t. For non-binary finite fields $\mathbb{GF}(p^m)$, where an appropriate value of p in the keys generation process may increase the strength of the system. In the case

of algebraic codes based on Hermitian curves defined by $h(x,y)_w = x^{w+1}+y^w+y$, the finite field is of the form $\mathbb{GF}(w^2)$.

Algorithm 1 outlines the steps necessary to construct the public-key. Once the finite field $\mathbb{GF}(q)$ has been generated, the genus g of the curve and the general parameters n, k, d^* of the code are calculated. Then, the kxn generator matrix G is constructed having entrances in $\mathbb{GF}(w^2)$. Two further matrices need to be constructed, namely, the scrambling S and the permutation P matrices. The elements of S belong to $\mathbb{GF}(w^2)$. The permutation matrix contains only one 1 for each row and each column. Finally, the public-key G' is calculated as $G' = SGP$.

Algorithm 1 Key Generation

Require: $\exists\ h(x,y) = x^{w+1} + y^w + y$
$g \leftarrow \frac{w(w-1)}{2}$, $m \leftarrow w+1$;
$q \leftarrow w^2$;
// Set up the AG code parameters
Select $j \in \mathbb{Z}^+ \cup \{0\} \mid m-2 \leq j \leq \lfloor \frac{n-1}{m} \rfloor$
$n \leftarrow w^3$
$k \leftarrow n + mj + g - 1$
$d^* \leftarrow n - k - g + 1$
// Construct $\mathbb{GF}(q)$ as an extension of $\mathbb{GF}(2)$
$\mathbb{GF}(q) \leftarrow \{\alpha_0, \alpha_1, \ldots, \alpha_{q-1}\}$.
// Construct the generator matrix G
$F \leftarrow \{x^i y^j \mid (i,j)\ ordered\ by\ <_T\}$; $\#(F) = k$.
$P \leftarrow \{(x,y) \in \mathbb{GF}(w^2) \mid x^{w+1} + y^w + y = 0\}$
for $i = 1$ to k **do**
  for $j = 1$ to n **do**
    $G \leftarrow G \cup f_i(p_j)$; $f_i \in F$, $p_j \in P$
// Construct a random scrambler matrix S.
$S \leftarrow rand(k,k)$; $s_{ij} \in \mathbb{GF}(q)$
// Construct a random $n \times n$ permutation matrix P.
// Compute $G' = SGP$.
return G' and SGP.

B. Encryption Process

The encryption process is straightforward. Let x be a message block to be encrypted of length k. The encrypted code is $c = x * G' + e$. The error e is a randomly generated vector of length n and weight t. Hence, the sender must know both G' and t the encryption parameters.

C. Decryption Process

The decryption process becomes quite complex when non-binary AG codes are used. However, in general terms the process is not different from the binary decryption process. The additional complexity is given on the non-binary finite field operations and primarily on the decoding algorithm of AG code used. Algorithm 2 outlines the general steps involved into the decryption process. From the knowledge of the curve and its parameters as well as the private key system $(P_n, G_{k,n}, S_k)$. The first step is to recover the original positions of each component of c. This process is carried out by multiplying c by the inverse matrix P^{-1}

The second step consists in remove the random vector e controllably introduced during the encryption process. This is the most complex step and is outlined in Section II-B. The details of this process are displayed in the Algorithms 3 and 4.

The third and last step in Algorithm 2 consist in the recovering of the original vector x. This is carried out simply by multiplying the code c' by the inverse matrix S_k^{-1}

Algorithm 2 Decryption Process

Require: a encrypted code c of arbitrary length n; the curve $h(x,y)_w$ and private key system $(P_n, G_{k,n}, S_k)$.
1. Permute c
  $c \leftarrow cP_n^{-1}$
2. Remove the error vector e from inside c
  2.1. Use Algorithm 4 to detect the errors in c
  2.2. Use Algorithm 5 to compute error magnitudes in c detected
  2.3. $c' \leftarrow c - e$
3. Compute x
  $x \leftarrow c'S_k^{-1}$
return x.

Algorithm 3 corresponds to the error detection process. Inside this algorithm, two algorithms (later explained), are contained. The received codeword c, possibly noisy, and the n affine points of the Hermitian curve are used as input. The polynomials F used for the recursion, G used as auxiliary, and $\Delta(F) = \phi$ are initialized. After running the algorithm the ultimately recursive polynomial F is returned, and is used to compute the error positions into the vector c. The algorithm contains three main loops inside. The first loop calculates the known syndromes. The second loop completes the syndrome array. Finally The third loop calculates the error positions.

Algorithm 4 is based on the Sakata's algorithm and it is the core of decoding and decryption process. Its primary function is to update the set F through iterations.

IV. Algorithmic Complexity

Considering all the algorithms presented in Section III-C, the most complex is Algorithm 4. We note that in order to facilitate the operations all polynomials were represented by matrices. In fact, the syndromes array is a polynomial that grows according to Algorithm 3. Beside this, some lines in the algorithm could represent many lines of computing code, and the measurements were made on this code. The parameter which determines the problem size is the length of syndrome sequence n. The longer is the length of the sequence, the larger is the problem size. The syndrome sequence is increased with each new position read (a, b).

Table I shows the number of basic computational operations needed for each line of the Algorithm 4 and their reduced computational order. The *best case* for the algorithm occurs

when a polynomial in F does not reach the point (a, b) in the partial order (i.e., when the condition in line 4 is not satisfied). In this case the executed lines are the ranges $[1, .., 4]$ and $[32, .., 34]$. The function which describes the recurrence relation, deduced from Table I, is

$$T(n) = 4n^3 + 66n^2 + 81n - 5 \tag{18}$$

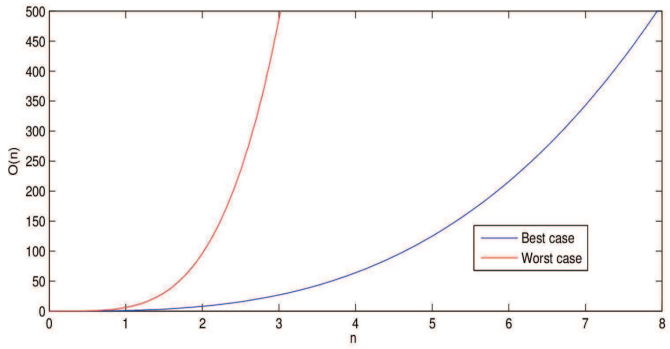

Fig. 2: Algorithmic Complexity of Algorithm 4

On the other hand, the *worst case* occurs when a polynomial reaching the point (a, b) does not generate the syndrome $S_{a,b} \in S$. The lines executed in this case are in the ranges $[1, .., 18]$ and $[32, .., 34]$. In this case de recurrence relation, deduced from Table I, becomes

$$T(n) = 6n^4 + 414n^3 - 398n^2 - 38n - 19 \tag{19}$$

From (18) and (19) the asymptotic upper bounds are $O(n) = n^3$ and $O(n) = n^4$, respectively. Their graphs are represented in Fig. 2, where both functions are polynomials.

TABLE I: Elementary operations on each code line of Algorithm 4

Line #	T(n)	O(n)	Line #	T(n)	O(n)
1:	$2n^2 + 21n - 9$	n^2	20:	1	1
2:	$2n + 2$	n	21:	-	-
3:	$(9n - 5)$	n	22:	$2n - 3$	2
4:	3	1	23:	1	1
5:	$(13n - 6)$	n	24:	-	-
6:	1	1	25:	-	-
7:	4	1	26:	-	-
8:	$5n^2 - 3n + 2$	n^2	27:	$24n - 3$	n
9:	$2n^2 + 21n - 9$	n^2	28:	$12n - 11$	n
10:	$15n^2 - 19n + 16$	n^2	29:	-	-
11:	$3n - 1$	n	30:	1	1
12:	$4n + 1$	n	31:	-	-
13:	$n^2 - n + 15$	n^2	32:	1	1
14:	$9n - 5$	n	33:	-	-
15:	$13n - 6$	n	34:	$2n^2 + 21n - 9$	n^2
16:	1	1	35:	-	-
17:	$24n - 3$	n	36:	1	1
18:	$10n - 8$	n	37:	1	1
19:	-	-	38:	$10n^3 - 14n^2 + 21n - 19$	n^3
20:	1	1	39:	3	1

From another point of view, it has been shown that increasing the diversity in the coding process inside an encryption system, the entire system becomes to be more robust in terms of attacks work factor [15].

V. Study Cases

In this section an illustrative and simple example is used to depict the key generation, encryption, and decryption processes.

A. Depicting the Cryptographic Process in $\mathbb{GF}(4)$

1) Generating keys: Let $h(x, y) = x^3 + y^2 + y$ be a Hermitian curve (4) over a finite field $\mathbb{GF}(4)$ having $f(x) = x^2 + x + 1$ as primitive polynomial. All four elements in $\mathbb{GF}(4)$ are shown in Table II. According to (5) the genus of the curve is $g = 2(2-1)/2 = 1$ and its degree $m = 2 + 1 = 3$, while the field size $q = w^2 = 2^2 = 4$. The number of affine points is $n = w^3 = 2^3 = 8$. The code parameters k and d^* depend on the value selected for j, which is bounded by the relation (6). Hence,

$$1 \leq j \leq 2 \tag{20}$$

If we choose $j = 1$ then, from (7) and (8) we obtain

$$k = 5$$

$$d^* = 3$$

As a result a Hermitian code $(8, 5, 3)$ is achieved, and is defined by the curve $h(x, y) = x^3 + y^2 + y$ over $\mathbb{GF}(2^2)$. Therefore, from (9) the number of errors that the code is able to correct is 1.

TABLE II: Elements of field $\mathbb{GF}(4)$

0	1	α	$\alpha^2 = \alpha + 1$

Now let us build the generator matrix G of the code. Firstly, we define the monomial basis F according to (10). The base is composed by the firsts $k = 5$ monomials, arranged in total graded lexicographic order of its powers. Each monomial in (21) is evaluated in each affine point represented in Table III in order to obtain each row of the generator matrix G.

$$F = \{1, x, y, x^2, xy\} \tag{21}$$

Table III shows the $n = 8$ affine points that satisfy $h(x, y) = 0$.

TABLE III: Affine points of $h(x, y)$ over $\mathbb{GF}(2^2)$

$P_1 = (0, 0)$	$P_2 = (0, 1)$	$P_3 = (1, \alpha)$	$P_4 = (1, \alpha^2)$
$P_5 = (\alpha, \alpha)$	$P_6 = (\alpha, \alpha^2)$	$P_7 = (\alpha^2, \alpha)$	$P_8 = (\alpha^2, \alpha^2)$

Hence,

$$G_{5,8} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \\ 0 & 0 & \alpha & \alpha^2 & \alpha^2 & 1 & 1 & \alpha \end{pmatrix}$$

From this, the systematic matrix results

$$G_{5,8} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ \alpha & 1 & \alpha & 0 & 1 & 0 & 0 & 0 \\ \alpha^2 & 0 & \alpha & 0 & 0 & 1 & 0 & 0 \\ 1 & \alpha^2 & \alpha^2 & 0 & 0 & 0 & 1 & 0 \\ 0 & \alpha & \alpha^2 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

The process continues with the construction of the 5×5 scrambler matrix S and the 8×8 permutation matrix P which are generated randomly. Thus we can consider:

$$S_5 = \begin{pmatrix} \alpha^2 & 1 & 0 & 0 & 0 \\ 0 & \alpha^2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & \alpha \end{pmatrix}$$

and

$$P_8 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

From (1) the encryption matrix $G'_{5,8}$ results

$$G'_{5,8} = \begin{pmatrix} 0 & 1 & \alpha & 0 & 1 & 1 & 0 & \alpha^2 \\ 0 & \alpha^2 & \alpha^2 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & \alpha^2 & 1 & 1 & \alpha & 0 & 0 \\ 0 & 0 & \alpha^2 & 1 & \alpha^2 & 1 & 0 & 0 \\ 0 & 0 & \alpha^2 & 0 & 1 & 0 & \alpha & 0 \end{pmatrix}$$

The other component of the public key is the number of controlled errors t to be include in the encryption process, calculated by using (9).

2) Encryption: Let $x = (\alpha, 0, \alpha^2, 1, \alpha)$ the message to be encrypted and $e = (0, 0, 0, 0, 0, \alpha, 0, 0)$ the induced controlled error of weight $t = 1$. Then the encrypted message results

$$c = x * G' + e = (\alpha^2, \alpha, \alpha^2, \alpha, 0, 0, \alpha^2, 1) \tag{22}$$

3) Decryption: The decryption process becomes quite complex when we use non-binary AG codes. However, in this example we only depict the general process, and details of the inner process of algebraic decoding are omitted. In order to decrypt c we proceed as follows:

Firstly, $c' = cP^{-1}$ is calculated

$$c' = c * P^{-1} = (0, \alpha^2, 0, 1, \alpha, \alpha^2, \alpha, \alpha^2) \tag{23}$$

The resultant vector c' is then subjected to the algebraic decoding process that includes the Algorithm 4 for finding the error location, and the Algorithm 3 that determines the magnitudes of the errors. After correcting the error, results

$$c'' = (\alpha, \alpha^2, 0, 1, \alpha, \alpha^2, \alpha, \alpha^2) \tag{24}$$

From c'' we extract the scrambled message symbols

$$x' = (1, \alpha, \alpha^2, \alpha, \alpha^2) \tag{25}$$

Finally, the original message is obtained by a de-scrambling process by performing $x = x' * S^{-1}$.

$$\begin{aligned} x &= (1, \alpha, \alpha^2, \alpha, \alpha^2) \begin{pmatrix} \alpha & \alpha^2 & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & \alpha \\ 0 & 0 & 0 & 0 & \alpha^2 \end{pmatrix} \\ &= (\alpha, 0, \alpha^2, 1, \alpha) \end{aligned} \tag{26}$$

B. Depicting the Cryptographic Process in Binary

Let $H(7, 4, 3)$ a Hamming code with generator matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

In this case, since the distance of code is 3, and the code can correct only 1 error. Therefore, the maximum error number we can introduce in the encryption process is 1.

1) Generating keys: The scrambling and permutation matrix are:

$$S = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

and

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Hence, the resulting public key $G' = SGP$ is

$$G' = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and the private key is formed by the system (G, S, P).

2) Encryption: Let $x = (1, 1, 0, 1)$ the message to be encrypted and $e = (0, 0, 1, 0, 0, 0, 0)$ the induced controlled error of weight $t = 1$. Then the encrypted message is

$$c = x * G' + e = (1, 1, 0, 0, 1, 1, 1) \tag{27}$$

3) Decryption: In order to retrieve the original message, the decryption process begins de-permuting the received encrypted word, i.e., $c' = cP^{-1}$.

$$c' = c * P^{-1} = (1, 0, 1, 1, 1, 1, 0) \tag{28}$$

where

$$P^{-1} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

After this, the Hamming decoding process must be performed. After correcting error, it yields

$$c'' = (1, 1, 1, 1, 1, 1, 0) \tag{29}$$

From c'' we extract the scrambled message symbols.

$$x' = (1, 1, 1, 1) \tag{30}$$

Finally, the original message is calculated as $x = x' * S^{-1}$.

$$\begin{aligned} x &= (1, 1, 1, 1) \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix} \\ &= (1, 1, 0, 1) \end{aligned} \tag{31}$$

As we can see this vector corresponds to the original message, so the encryption-dencryption process succeed.

VI. Analysis of Results

We have developed simple examples to depict both binary and non-binary cryptographic scheme by performing the key generation, encryption and decryption process of messages according to Section III. In the non-binary case, a generator matrix $G_{5,8}$ has been constructed from the Hermitian code $C(8, 5, 3)$ defined by the curve $h(x, y) = x^3+y^2+y$ over $\mathbb{GF}(4)$. Afterwards, we have randomly generated both scrambler matrix S_5 and permutation matrix P_8. These three matrices constitute the private key system (G, S, P) while the resulting matrix product $G'_{5,8} = S_2 G_{5,8} P_8$ defines the public key.

The code correction capability contributes to the security level of our cryptosystem in terms of establishing the maximum number of random components in the encrypted code. In particular for this simple example, the used code $C(8, 5, 3)$ can correct only one error. This means that only one controlled error can be added to the codeword during the encryption process. By another hand, in the non-binary system both the S and P matrix improve diversity since the combinatorial complexity increases dramatically, specially due to the non-binary scrambling matrix. As a result, for an eavesdropper who is trying to guess the private key the task becomes a extremely very hard issue.

The encryption process presented requires splitting the original message into blocks of fixed length k. During the encryption, an error random vector vector having Hamming weight t is added to each block . However, this intentional error can be solved by the algebraic code. The correcting capacity of the algebraic code determines the randomness inherent in the encrypted code, and greater correction capability means greater randomness in the encrypted code.

In general terms, the encryption and decryption process is basic and simple. However, the computational effort for the decryption process is significative. Although in this example we have only worked with small matrices, the complexity of decoding operations is remarkable.

The decryption process is marked by the complexity of the algebraic decoding process and the pretensions that we have regarding to security levels we want to achieve. To achieve a higher security level, better and longer error correcting codes must be used, increasing the complexity of operations.

VII. Conclusions

In this paper we have shown a method for building a public-key encryption system using the McEliece scheme over Hermitian curves. To the best of our knowledge, all reported work to date has performed with binary codes which used binary decoding. The attack Work factor in the non-binary case is greater than the binary one. Since both the generator and scrambling matrices are non-binary, the attacker's effort to guess their entrances is much harder than in the binary case. The strength of McEliece cryptosystem over Hermitian curves lies on the complexity of the non-binary decoding.

The presented McEliece security system may also be applied to storage and retrieving information using the magnetic vortex in nanoparticles. By using the magnetic vortex in magnetic nanodisks, a single memory point can store quaternary information, that is, in the Galois field $GF(4)$. Therefore, it seems feasible to implement the proposed system in this kind of memories. The vortex memories storage is being currently under research and practical systems haven't been presented yet.

Acknowledgments

The authors acknowledge the financial support of the "Center for Multidisciplinary Research on Signal Processing" (CONICYT/ACT1120 Project), USACH/DICYT 061413SG Project and DICYT-Usach-Project 06-1317-DC, Project CORFO 14IDL2-2991914, PhD Chile CONICYT Scholarship 2012 Res. Ex. 1108, and PhD Chile CONICYT Scholarship 2008.

References

[1] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," *The Deep Space Network Progress Report*, pp. 114-116, 1978.

[2] E. Berlekamp, R. McEliece and H. van Tilborg, *On the inherent intractability of certain coding problems, IEEE Transaction Information Theory*, vol.24, no. 3, pp. 384-386, 1978.

[3] S. Sakata, "Extension of the Berlekamp-Massey Algorithm to N Dimensions," *Academic Press, Inc. NY*, vol.84, no. 2, pp. 207-239, 1990.

[4] H. Dinh,C. Moore and A. Russel, "The McEliece Cryptosystem Resists Quantum Fourier Sampling Attacks," (Submitted on 13 Aug 2010 (v1), last revised 15 Oct 2010 (this version, v2)). , 2010.

[5] D. J. Bernstein, T. Lange, and C. Peters, "Attacking and Defending the McEliece Cryptosystem," *Springer Berlin Heidelberg*, pp. 31-46, 2008,

[6] J. Justesen, K.J. Larsen, H.E. Jensen, and T. Hoholdt, "Fast Decoding of Codes from Algebraic Plane Curves," *IEEE Transaction Information Theory*, vol.38, no. 1, pp. 111-119, 1992.

[7] G. L. Feng and T. R. N. Rao, "Decoding algebraic-geometric codes up to the designed minimum distance," *IEEE Transactions on Information Theory*, vol.39, no. 1, pp. 37-45, 1993.

[8] R.A. Carrasco and M. Johnston, "Non-binary Error Control Coding for Wireless Communication and Data Storage," 1^{st} edition, John Wiley & Sons, United Kingdom, 2008.

[9] M. Johnston and R.A. Carrasco, "Construction and performance of algebraic-geometric codes over AWGN and fading channels," *IEE Proceedings - Communications*, vol.152, no. 5, pp. 713-722, 2005.

[10] M. Johnston, R. A. Carrasco and B. L. Burrows, "Design of algebraic-geometric codes over fading channels," *Electronics Letters*, vol.40, no. 21, pp. 1355-1356, 2004.

[11] S. Sakata, J. Justesen, Y. Madelung, H. E. Jensen, and T. Hoholdt, "Fast decoding of algebraic-geometric codes up to the designed minimum distance," *IEEE Transactions on Information Theory*, vol.41, no. 6, pp. 1672-1677, 1995.

[12] C.W. Liu, "Determination of Error Values for Decoding Hermitian Codes with the Inverse Affine Fourier Transform," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 82, no. 10, pp. 2302-2305, 1999.

[13] M. Asmat-Uceda, X. Cheng, X. Wang, D. J. Clarke, O. Tchernyshyov, y K. S. Buchanan, "A comparison of numerical simulations and analytical theory of the dynamics of interacting magnetic vortices," *AIP Publishing*, vol.117, no. 12, 2015.

[14] L. Ma y C. Xing, "Constructions of block codes from algebraic curves over finite fields 1," *Surv. Comb.*, vol. 424, pp. 295-324 2015.

[15] I. Soto, I. Jiron, A. Valencia, R. Carrasco, "Secure DNA data compression using algebraic curves," *Electronics Letters* vol. 51, Issue: 18 pp. 1466-1468 2015.

Algorithm 3 Updating Connection Polynomials

Require: $(a,b), S, F, G, \Delta(F)$;

$\Psi^{(e)}_{\Delta(F)} \leftarrow \{(x,y) \in \Delta'(F) \mid (x,y)\ is\ a\ minimal\ point\}$
for each $f^{(i)}(x,y) \in F$ **do**
  $(s_1,s_2) \leftarrow bideg(f^{(i)})$
  if $(s_1,s_2) \leq (a,b)$ **then**
    $\delta_{S_{a,b}}^{f^{(i)}(x,y)} = \sum_{(i,j)=(0,0)}^{(s_1,s_2)} f^{(i)}_{(i,j)} S_{(a+i-s_1,b+j-s_2)} | f^{(i)}_{(i,j)}$ coeff. in $f^{(i)}(x,y)$
    if $\delta_{S_{a,b}}^{f^{(i)}(x,y)} \neq 0$ **then**
      if $(a-s_1, b-s_2) \notin \Delta(F)$ **then**
        $\Delta(F^*) \leftarrow \Delta(F) \bigcup \{(x,y) \in N^2 \mid (x,y) \leq (a-s_1, b-s_2)\}$
        $\Psi^{(e)}_{\Delta(F^*)} \leftarrow \{(x,y) \in \Delta'(F^*) \mid (x,y)\ is\ a\ minimal\ point\}$
        $\Gamma_\Psi \leftarrow \Psi_{\Delta(F^*)} \setminus \Psi_{\Delta(F)}$
        for each $(e_1,e_2) \in \Psi_{\Delta(F^*)}$ **do**
          if $(e_1,e_2) \in \Gamma_\Psi$ **then**
            $f' \leftarrow x^{|e_1-s_1|} y^{|e_2-s_2|} f^{(i)}$
            $(s'_1, s'_2) \leftarrow bideg(f')$
            $\delta^{f'}_{S_{a,b}} \leftarrow \sum_{(k_1,k_2)=(0,0)}^{(s'_1,s'_2)} f'_{(k_1,k_2)} S_{(k_1-s'_1,k_2-s'_2)}$
            if $\delta^{f'}_{S_{a,b}} \neq 0$ **then**
              find $g^{(n)} \in G$ so that $\delta^{f'}_{S_{a,b}} \neq 0$
              $f'' \leftarrow \delta^{f^{(i)}}_{S_{a,b}} g^{(n)}$; **break loop**
            else
              $f'' \leftarrow 0$; **break loop**
            $F^* \leftarrow f' + f''$
          else $F^* \leftarrow f^{(i)}$
      else
        find $g^{(n)} \in G$ so that $\delta^{g^{(n)}}_{S_{a,b}} \neq 0$
        $F^* \leftarrow f^{(i)} + \delta^{f^{(i)}}_{S_{a,b}} g^{(n)}$
    else $F^* \leftarrow f^{(i)}$
  else $F^* \leftarrow f^{(i)}$
  $\Psi^{(i)}_{\Delta(F^*)} \leftarrow \{(x,y) \in \Delta(F^*) \mid (x,y)\ is\ a\ maximal\ point\}$
$\Delta(F) \leftarrow \Delta(F^*)$
$F = F*$
$G \leftarrow updateInteriorPolynomials\left(G, \Psi^{(i)}_{\Delta(F^*)}\right)$
return F, G and $\Delta(F)$

Algorithm 4 Error Detection Process

Require: $c = \{c_1, c_2, \ldots, c_n\}$; $P = \{(x_i, y_i) \in (\mathbb{GF}(q^2))^2 \mid h(x, y) = 0; i = 1..n\}$
Ensure: $F = 1$, $G = 1$, $\Delta(F) = \phi$
for all $(a, b) \mid (0, 0) \preceq (a, b) \preceq (0, j)$ **do** ▷ 1^{st} Loop
  $S_{a,b} = \sum_{i=1}^{n} c_i x_i^a y_i^b$ ▷ known
  $S \leftarrow S_{a,b}$
for all $(a, b) \mid (0, 0) \preceq (a, b) \preceq (n, n)$ **do** ▷ 2^{nd} Loop
  if $(a, b) \succeq (0, j)$ **then**
    if $(a, b) \preceq (m, j - 1 + m)$ **then** ▷ implied
      $S_{(a,b)} = S_{(a-m,b+m-1)} + S_{(a-m,b+1)}$
    else ▷ unknown
      Majority Voting Scheme to find $\hat{S}_{(a,b)}$
      $S_{(a,b)} \leftarrow \hat{S}_{(a,b)}$
    $S \leftarrow S \cup S_{(a,b)}$
  Use Algorithm 3 to update F, G **and** $\Delta(F)$
$P \leftarrow \{(x, y) \in (\mathbb{GF}(w^2))^2 \mid x^{w+1} + y^w + y = 0\}$
for all $(x, y) \in P$ **do** ▷ 3^{rd} Loop
  $test = TRUE$
  for each $f^{(i)} \in F$ **do**
    if $f^{(i)}(x, y) \neq 0$ **then**
      $test = FALSE$
  if $test = TRUE$ **then**
    $P' \leftarrow (x, y)$
return P'

Algorithm 5 Error Magnitudes Process

Require: S, F, c.
  1. Compute error locations
  for each $(x, y) \in h(x, y) = 0$ **do**
    if $\forall f \in F \mid f(x, y) = 0$ **then**
      $E \leftarrow (x, y)$
  2. Compute error Magnitudes
  for each $(e_1, e_2) \in E$ **do**
    if $e_1 \neq 0$ **then**
      if $e_2 \neq 0$ **then**
        Compute IDFT
      else
        $TransformadaSimple$
      $Transformadasimple$
    else
      $compuesto$
  return x.