The McEliece Cryptosystem

Suanne Au **Christina Eubanks-Turner** **Jennifer Everson**

September 17, 2003

Abstract

The McEliece cryptosystem is a public key cryptosystem whose security rests on the difficult problem of decoding an unknown error-correcting code. We give two examples of attacks to the cryptosystem, as well as a brief introduction to Goppa codes. One modification to this cryptosystem proposed by Pierre Loidreau increases the security of the system without increasing the key size or length of the code.

1 Introduction

In 1978, McEliece created a public key cryptosystem based on error-correcting codes and the difficult problem of decoding a message with random errors. RSA and other well known public key cryptosystems rely on the difficulty of factoring integers and finding the discrete log of a number, but the security of the McEliece cryptosystem does not depend on these problems. Also, the encryption and decryption is much faster than RSA. The McEliece cryptosystem is not currently used due to the relatively large public key and low data rate. However, as we will later explain, it is possible that in the future this cryptosystem will become more secure and more viable for implementation.

2 Introduction to Coding Theory

2.1 Purpose of Coding Theory and Some Basic Definitions

The purpose of coding theory is not to send secret messages but rather to ensure that information is transmitted accurately. When a message is sent through a channel, noise causes errors to occur in the message. If messages are sent in binary, for example, some of the bits may become "flipped" to the opposite value (a zero to a one or a one to a zero). Additional bits can be attached to each message so that the receiver can detect and correct the errors that occur. The purpose of coding theory is to find efficient ways to attach additional information to messages so that the receiver can correct as many errors as possible.

Definitions: Let $V = \{\begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} | c_i \in \{0,1\}\}$, where addition in V is component-wise addition (mod 2). So V is the set of all n-tuples of elements of $\mathbb{F}_2$. Then we define an $[n,k]$ *binary linear code* as a k-dimensional subspace of $V \cong \mathbb{F}_2^n$. The parameters n and k are called the *length* and *dimension*, respectively, of the $[n,k]$ binary linear code. Throughout the remainder of this paper we will use the word *code* to mean a *binary linear code*. A *codeword* is any vector in the code.

Definition: The *weight* of a vector is the number of ones it contains.

For example, the weight of $\begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0 & 1\end{bmatrix}$ is 4.

Definition: The *minimum weight* d of a code is the smallest weight of any nonzero (not all zeroes) codeword in the code.

The minimum weight is not always easy to compute, but when the minimum weight is known, it is listed as the third parameter of the code. So we can talk about $[n, k, d]$ binary linear codes.

The most important thing we want to know about a code is of course how many errors it can correct.

Important Fact: A binary linear $[n, k, d]$ code can correct $t = \lfloor \frac{d-1}{2} \rfloor$ errors.

Hence the minimum weight d tells us how good the code is. Of course, the "goodness" of a code also depends on n and k since we do not want to send too many extra bits for each message. We need to maintain efficiency of transmission as well.

Definition: We can represent an $[n, k]$ code by exactly k linearly independent vectors with entries in $\mathbb{F}_2$. We write these k vectors as the rows of a $k \times n$ matrix, which we call the *generator matrix*. The generator matrix is appropriately named because we can "generate" all vectors in the code by taking all possible linear combinations of the rows of the generator matrix.

2.2 Example of a "Good" Code–The [7,4,3] Hamming Code

We will use the Hamming [7,4,3] code to describe the general encoding and decoding processes. The generator matrix for the Hamming code is

$$\mathbf{G} = \begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0 & 1\\ 1 & 1 & 1 & 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}.$$

Since $k = 4$ our messages will have 4 digits in binary. So there are $2^4 = 16$ possible messages we can send. As an example, we will let 1100 mean "DOG" and 1110 mean "CAT." Now suppose we wish to send the message "DOG"=1100. Then we encode this message by multiplying the vector $\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix}$ by $\mathbf{G}$ on the right. We have

$$\begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix} \cdot \mathbf{G} = \begin{bmatrix}1 & 1 & 0 & 0\end{bmatrix} \cdot \begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0 & 1\\ 1 & 1 & 1 & 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix} = \begin{bmatrix}0 & 0 & 1 & 1 & 0 & 0 & 1\end{bmatrix}.$$

In general, we have

$$\begin{bmatrix}c_1 & c_2 & c_3 & c_4\end{bmatrix} \cdot \begin{bmatrix}1 & 1 & 0 & 1 & 0 & 0 & 1\\ 1 & 1 & 1 & 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 1 & 0 & 1 & 0 & 1 & 0\end{bmatrix} = \begin{bmatrix}(c_1+c_2+c_3) & (c_1+c_2+c_4) & c_2 & (c_1+c_3+c_4) & c_3 & c_4 & c_1\end{bmatrix}.$$

So when the codeword is received, assuming no errors have occurred, we can read of the original message from the 7^{th}, 3^{rd}, 5^{th}, and 6^{th} positions, in that order.

2.3 Maximum-Likelihood Decoding

Suppose Alice wants to send Bob the encoded message "DOG,"=$\begin{bmatrix}0 & 0 & 1 & 1 & 0 & 0 & 1\end{bmatrix}$, but an error occurs in transmission and Bob receives the vector $\begin{bmatrix}0 & 0 & 1 & 1 & 1 & 0 & 1\end{bmatrix}$. If Bob reads off the 7^{th}, 3^{rd}, 5^{th}, and 6^{th} positions, he gets the message 1110="CAT". Oh, no! There is still hope, however, because as we learned in the previous section, the Hamming code can correct up to $t = \lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{3-1}{2} \rfloor = 1$ error. To explain how this is done, we need to introduce a few more concepts.

Below is a table showing some of the codewords in the code (top row) and the words we obtain as a result of a single error. Notice that, for this code, the table includes all codewords because $2^4(codewords) \times 8(rows) = 128 = 2^7(words)$.

	Error Vectors								
Codewords	0000000	1101001	1110000	1001100	0101010	**0011001**	0111100	1100110	$\cdots$
	1000000	0101001	0110000	0001100	1101010	1011001	1111100	0100110	$\cdots$
	0100000	1001001	1010000	1101100	0001010	0111001	0011100	1000110	$\cdots$
	0010000	1111001	1100000	1011100	0111010	0001001	0101100	1110110	$\cdots$
	0001000	1100001	1111000	1000100	0100010	0010001	0110100	1101110	$\cdots$
	0000100	1101101	1110100	1001000	0101110	**0011101**	0111000	1100010	$\cdots$
	0000010	1101011	1110010	1001110	0101000	0011011	0111110	1100100	$\cdots$
	0000001	1101000	1110001	1001101	0101011	0011000	0111101	1100111	$\cdots$

Bob decodes in the following way. He looks up his received message, 0011101, in the table and then he decodes to the codeword at the top of the column. Alternatively, he could look at the error vector at the beginning of the row and add this error vector to his received message. Notice that this method will always result in decoding to the closest codeword in the code. By "closest," we mean the codeword which differs in the fewest number of positions from the received message. This method of decoding is called *maximum-likelihood decoding* and it is indeed the best method of decoding. However, making tables for big codes is time-consuming and inefficient. There is a faster way to decode messages, called syndrome decoding, which relies on a parity check matrix.

2.4 Syndrome Decoding

Definition: If C is an $[n, k, d]$ code, we define $C^{\perp} := \{\mathbf{v} \in V | \mathbf{v} \cdot \mathbf{c} = 0 \text{ for all } \mathbf{c} \in C\}$, where "$\cdot$" is the vector dot product (mod 2). $C^{\perp}$ is called the *dual* of the code C.

Fact: $C^{\perp}$ is $(n-k)$-dimensional.

Definition: A *parity check matrix* of a code C is an $(n-k) \times n$ matrix, where the rows are linearly independent vectors such that the dot product (mod 2) of each row with any vector in the code C is 0.

Though we will not prove it here, if the dot product (mod 2) of a vector with any row of the parity check matrix is 0, then that vector is in the code. Also, the parity check matrix for C is a generator matrix for $C^{\perp}$, and the generator matrix for C is a parity check matrix for $C^{\perp}$.

The parity check matrix $\mathbf{H}$ for the Hamming code is given below. You can easily check that the dot product (mod 2) of any row in $\mathbf{H}$ with any row of $\mathbf{G}$ is 0.

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Definition: If we take any vector $\mathbf{v} \in V$, and multiply $\mathbf{v}^t$ (the transpose of $\mathbf{v}$) on the left by the parity check matrix, we call the resulting $(n-k) \times 1$ vector the *syndrome* of $\mathbf{v}$.

For example, let's compute the syndrome of Bob's received message, $\bar{\mathbf{c}} = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}$.

$$\mathbf{H} \cdot \bar{\mathbf{c}}^t = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Let $\mathbf{c} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$ and let $\mathbf{e} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$, the error vector corresponding to $\bar{\mathbf{c}}$. Then we see that

$$\mathbf{H} \cdot \bar{\mathbf{c}}^t = \mathbf{H} \cdot (\mathbf{c} + \mathbf{e})^t = \mathbf{H} \cdot \mathbf{c}^t + \mathbf{H} \cdot \mathbf{e}^t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \mathbf{H} \cdot \mathbf{e}^t = \mathbf{H} \cdot \mathbf{e}^t,$$

since the dot product (mod 2) of a codeword with each row of the parity check matrix is 0. It is true in general that the syndrome of a received message is the same as that of its corresponding error vector. Notice further that the syndrome of an error vector with ones in positions $i_1, i_2, ...i_j$ is the sum (mod 2) of the corresponding columns of the parity check matrix (i.e., (column i_1) + $\cdots$ + (column i_j)). Now we will describe the process of syndrome decoding. This works for any binary linear $[n, k, d]$ code.

Syndrome Decoding:

1. First, compute the syndromes of the single-error vectors. For each new syndrome, store the syndrome and its corresponding single-error vector.

2. Do the same for the double-error vectors, storing only new syndromes with their corresponding double-error vectors.

3. Continue this process until all 2^{n-k} syndromes have been obtained.

4. To decode a received message, simply compute the syndrome, find the corresponding error vector in the list, and add that error vector to the received message. This results in decoding to the nearest codeword in the code.

Notice that syndrome decoding saves time over making the full table of possible words and error vectors. The latter table requires 2^n entries to be computed whereas the syndrome table has just $2 \cdot 2^{n-k} = 2^{n-k+1}$ entries to be computed. The reader might notice that 2^{n-k+1} is still incredibly huge when n is large and twice the size of k, as is generally the case. This is true and is, in fact, the problem on which the security of the McEliece cryptosystem lies. It is only in knowing the actual code that one knows the fast decoding algorithm with which the code is equipped. Have you noticed the fast decoding algorithm for the Hamming code?

Look again at the columns of the parity check matrix $\mathbf{H}$. That's right, each column is the binary representation of one of the numbers 1 through 7, in that order. So when the syndrome of a received

message is computed, say with an error in the i^{th} position, then the corresponding syndrome is the i^{th} column of $\mathbf{H}$, which is the number i in binary. So the receiver knows to correct position i, without ever having to compute any chart or table–just one syndrome! For the Hamming code this works for any received message with a single error (or no errors), which is the maximum decoding capability of the Hamming code (since $t = \lfloor \frac{3-1}{2} \rfloor = 1$). Now that is fast!

3 Goppa Codes

The McEliece cryptosystem relies on the use of a large family of codes, that is, a set of codes with the same length and dimension. More importantly, these codes must be unpredictable because the security of the system depends upon the fact that a third party does not know the fast decoding algorithm and so they are forced to use the much lengthier process of syndrome decoding.

3.1 What is a Goppa Code?

Though we cannot go into all the details of Goppa codes here, we will describe what it means to be a codeword and what objects are necessary for the construction of a Goppa code.

As described in [6] and [2], we first pick a finite field $\mathbb{F}_{2^m}$ of $n = 2^m$ elements. Next we choose a labelling $L = \{\zeta_i : 0 \leq i \leq 2^m - 1\}$ of $\mathbb{F}_{2^m}$. We select an irreducible polynomial $g(z) \in \mathbb{F}_{2^m}[z]$ of degree t. We call this polynomial the *generating polynomial* of the Goppa code. Then we say that a vector $\mathbf{c} = (c_0, c_1, ..., c_{2^m-1})$ is in the irreducible Goppa code $\Gamma(L, g)$ if and only if

$$\sum_{i=0}^{2^m-1} c_i (z - \zeta_i)^{-1} \equiv 0 \bmod g(z).$$

Note that $(z - \zeta_i)^{-1}$ exists since $\mathbb{F}_{2^m}[z]/g(z)$ is a field. It is an important fact that the degree t of the irreducible polynomial gives a lower bound for the error-correcting capability of the code.

3.2 Why Goppa Codes?

There are several reasons why Goppa codes are the primary choice for the McEliece cryptosystem. First of all, Goppa codes have a fast polynomial time decoding algorithm [6]. Another reason is that Goppa codes are "easy to generate but hard to find". Any irreducible polynomial over a finite field $\mathbb{F}_{2^m}$ can be used to create a Goppa code, but the generator matrices of Goppa codes are nearly random [2]. For any fixed length n, there are many different Goppa codes. Though the exact number of Goppa codes, given n and t, is not know, Ryan and Fitzpatrick [2] found a way to calculate upper bounds, which are exact for some of the small parameters. For example, the upper bound for the number of Goppa codes of length 128 which are able to correct at least 10 errors is $1037499670492467 \approx 1.04 \times 10^{15}$, while the upper bound for Goppa codes of the same length able to correct at least 15 errors is $2376547806952061120164378 1 \approx 2.38 \times 10^{25}$. In fact, the number of Goppa codes grows exponentially with the length of the code and the degree of the generating polynomial [2]. Goppa codes are still the primary family of codes used with the McEliece cryptosystem.

4 Overview of the McEliece Cryptosystem

The secret key space is a family of Goppa Codes. The private key is chosen as follows: pick a generator matrix for a random Goppa code from the key space, a $k \times k$ invertible binary matrix $\mathbf{S}$

and an $n \times n$ permutation matrix $\mathbf{P}$. Only the receiver knows this private key. The public key is $\mathbf{G}' = \mathbf{SGP}$, where $\mathbf{G}$ is the generating matrix for the Goppa code used.

Let $\mathbf{x}$ be the k-bit message to be encrypted. The sender sends $\mathbf{x}' = \mathbf{xG}' + \mathbf{e}$, where $\mathbf{e}$ is a random n-bit error vector with weight t, where t is the degree of the generating polynomial of the Goppa code (which is also the lower bound of the error-correcting capability of the code). After the receiver receives $\mathbf{x}'$, to decrypt the message, the receiver computes $\mathbf{x}'\mathbf{P}^{-1} = \mathbf{xSG} + \mathbf{eP}^{-1}$. By using the efficient decoding algorithm of the code, the receiver is left with $\mathbf{xS}$. Since $\mathbf{S}$ is invertible, the receiver can recover $\mathbf{x}$. Loidreau says, "The security of the system depends on the difficult problem of decoding a code up to its error-correcting capability" [6].

5 Cryptanalysis of the System

5.1 Types of Attacks

The security of the McEliece cryptosystem is based on two computationally hard problems, which are exhaustive search on the key space and maximum-likelihood decoding (syndrome decoding). Therefore, the two types of attacks on this cryptosystem, structural and decoding, are concerned with these two problems respectively.

5.1.1 Structural Attack

A structural attack consists of attempts to reconstruct a decoder for the code generated by the public key, $\mathbf{G}'$. If such an attempt is successful, then the private key $\mathbf{G}$ would be revealed and the cryptosystem would be completely broken. The family of codes can be divided into classes of equivalent codes. Two codes of length n are said to be *equivalent* if there exists a permutation of the n positions changing one code to the other. The code, C', generated by $\mathbf{G}'$, and the underlying code generated by $\mathbf{G}$ are from the same equivalence class. An attacker has to compare a representative code from each class to C' in order to determine an equivalent code. Since the equivalent classes have very small cardinality, this process is beyond the capabilities of even the most powerful computers. If the original parameters, $n = 1024$ and $t = 50$, are used, this structural attack requires the examination of about 2^{466} codes. Once an equivalent code, C'', is found, the permutation between C' and C'' can be recovered by methods such as the Support Splitting Algorithm discussed in Section 5.2.

5.1.2 Decoding Attack

A decoding attack consists of decoding the intercepted ciphertext. In the case of a successful decoding attack, the plaintext message is recovered but the cryptosystem remains intact. Since the underlying code and C' are equivalent, they have the same error-correcting capability. Thus, the cost of the attack depends only on the parameters of C'–its length, dimension and error-correcting capability. When the length is 1024, the dimension is 524 and the error-correcting capability is 50, decoding one word requires 2^{64} binary operations [6]. As the power of computers increases, the McEliece cryptosystem with the original parameters becomes less secure everyday. However, we cannot increase the size of the public key too much since it is too large for implementation as it is.

Under different circumstances, many efficient decoding attacks are possible, but structural attacks remain infeasible in general.

5.2 Support Splitting Algorithm

We first find a Goppa code which is equivalent to the code generated by the public key. Then, using the *support* of the code (the set of positions where any codeword in the code is nonzero), we seek to find the permutation between the two codes.

The Support Splitting Algorithm (S.S.A.) works by determining a set of properties of a code and one of its positions, which are invariant when the support is permuted. These properties are called the signature. There are no time estimates to find the signature since they are relative to the code. The main difficulty with implementation of the algorithm is picking a suitable invariant. An example of a reasonable invariant is the weight distribution of the code. When the size of the code increases, the computation of the signature becomes more difficult. For details on how to find the signature, see [8]. "The discriminancy of a signature is measured by how often it will be different for two distinct positions of the same code" [8].

"If the signatures of the positions of a code are all different, then the permutation between this code and any other one, permutation-equivalent to it, can be recovered by matching the sets of the signatures of both codes. If the signatures are not all different, a procedure, which is usually efficient, enables us to recover the permutation" [8]. After this procedure, the permutation is known and one is able to determine the private Goppa code. Sendrier also states that the S.S.A. is polynomial in the length of the code and exponential in the dimension of the intersection of the code with its dual.

5.3 Message-Resend and Related-Message Attack [1]

One of the main disadvantages of the original McEliece public-key cryptosystem is that encrypting the same message more than once is recognizable and the plaintext can be recovered by an attacker in αk^3 time, where α is a small constant and k is the message size of the underlying code. The system also fails to protect any messages that are known to have a linear relation to one another. Since this decoding attack does not depend on the structure of the code, changing the code or increasing the parameter sizes does not effectively blunt this attack.

McEliece believes that the security of his cryptosystem is based on solving the problem of decoding an arbitrary binary word to the nearest codeword in an arbitrary linear code under the restriction that the word has weight less than the error-correcting capability of the code. An obvious attack goes as follows. Suppose the Goppa code has a 524×1024 generator matrix and a generating polynomial of degree 50. If an attacker could guess 524 coordinates of a ciphertext word, $\mathbf{c}$, that are not garbled by the random error vector, $\mathbf{e}$, then $\overline{\mathbf{c}} = \mathbf{m}\overline{\mathbf{SGP}}$, where $\overline{\mathbf{c}}$ and $\overline{\mathbf{SGP}}$ are the restriction of $\mathbf{c}$ and the public key, $\mathbf{SGP}$, to those 524 coordinates. Note that $\overline{\mathbf{SGP}} \in \mathbb{F}_2^{524\times 524}$ is known and if it is checked to be invertible, then $\mathbf{m}$ can be recovered.

Let's consider the cost of this attack. The attacker must guess 524 ungarbled columns out of the possible $974 = 1024 - 50$, since there are 50 errors in the ciphertext. The number of possible guesses is $\binom{1024}{524}$ and if the 974 ungarbled columns are known, the number of correct guesses is $\binom{974}{524}$; therefore $\frac{\binom{1024}{524}}{\binom{974}{524}} \approx 1.37 \times 10^{16}$ is the number of guesses required to succeed. So, the cost is $w = k \times 1.37 \times 10^{16}$, where k is the cost of inverting a 524×524 matrix, which is roughly 524^3. This exhaustive method is not feasible under current technology.

5.3.1 Message-Resend Condition

Now, suppose a message is encrypted and sent twice whether by accident or as a result of the attacker's action, then we have $\mathbf{c_1} = \mathbf{mSGP} + \mathbf{e_1}$ and $\mathbf{c_2} = \mathbf{mSGP} + \mathbf{e_2}$, where $\mathbf{e_1} \neq \mathbf{e_2}$. This is

called a message-resend condition, which is easily detectable by observing the weight of the sum of the two ciphertexts, $\mathbf{c_1}$ and $\mathbf{c_2}$. When the plaintext messages are identical, the sum of $\mathbf{c_1}$ and $\mathbf{c_2}$ cannot exceed 100. For details on how to detect a message-resend condition, see Heiman [5]. In the case that the attacker recognizes a message-resend condition, $\mathbf{m}$ can be easily recovered. We will only discuss the case where the number of ciphertexts of the same message, which is called the resend depth, is two. The cost of the attack is even smaller for greater resend depth.

5.3.2 Method of Attack

From the two equations above, we have $\mathbf{c_1} + \mathbf{c_2} = \mathbf{e_1} + \mathbf{e_2} \pmod 2$. Now, define L_0 as the set of coordinates where $(\mathbf{c_1} + \mathbf{c_2})$ is zero and L_1 as the set of coordinates where $(\mathbf{c_1} + \mathbf{c_2})$ is one, i.e.,

$$L_0 = \{\ell \in \{1, 2, \cdots, 1024\} : \mathbf{c_1}(\ell) + \mathbf{c_2}(\ell) = \mathbf{e_1}(\ell) + \mathbf{e_2}(\ell) = 0\}$$

$$L_1 = \{\ell \in \{1, 2, \cdots, 1024\} : \mathbf{c_1}(\ell) + \mathbf{c_2}(\ell) = \mathbf{e_1}(\ell) + \mathbf{e_2}(\ell) = 1\}.$$

For every $\ell \in L_0$, either $\mathbf{e_1}(\ell) = 0 = \mathbf{e_2}(\ell)$ or $\mathbf{e_1}(\ell) = 1 = \mathbf{e_2}(\ell)$. The probability of $\mathbf{e_i}$ having a one at any coordinate is $\frac{50}{1024}$, and since the error vectors $\mathbf{e_1}$ and $\mathbf{e_2}$ are chosen independently, the probability of both of them having a one at the same coordinate is $\left(\frac{50}{1024}\right)^2 \approx 0.0024$. In other words, $\ell \in L_0$ implies neither $\mathbf{c_1}(\ell)$ nor $\mathbf{c_2}(\ell)$ is garbled by an error vector. For every $\ell \in L_1$ precisely one of $\mathbf{c_1}(\ell)$ and $\mathbf{c_2}(\ell)$ is garbled by an error vector. Thus, after computing L_0 and L_1 from $\mathbf{c_1}$ and $\mathbf{c_2}$, the attacker should try to guess 524 ungarbled columns from those indexed by L_0.

Let's analyze this strategy. Let p_i be the probability that precisely i coordinates are simultaneously garbled by $\mathbf{e_1}$ and $\mathbf{e_2}$. We know that $\binom{50}{i}$ is the number of ways for $\mathbf{e_1}$ to have i ones in common with $\mathbf{e_2}$. The remaining ones in $\mathbf{e_1}$ must be in the remaining 974 positions where $\mathbf{e_2}$ does not have any ones. This yields $\binom{974}{50-i}$ possibilities. Since there are $\binom{1024}{50}$ possibilities for $\mathbf{e_2}$,

$$p_i = \frac{\binom{50}{i}\binom{974}{50-i}}{\binom{1024}{50}}.$$

We know $\mathbf{e_1}$ and $\mathbf{e_2}$ differ in at most 100 places and for every i for which $\mathbf{e_1}(\ell) = 1 = \mathbf{e_2}(\ell)$, the size of L_1 is reduced by two, so the possible values $|L_1|$ can take on are $(100 - 2i), 0 \leq i \leq 50$. Therefore, the expected cardinality of L_1 is $\sum_{i=0}^{50}(100 - 2i)p_i \approx 95.1$. That implies the expected cardinality of L_0 is $1024 - 95 = 929$. For example, if $|L_1| = 94$ and $|L_0| = 930$, then the attacker only needs $\frac{\binom{930}{524}}{\binom{927}{524}} \approx 12$ guesses to succeed, and the cost is only 12β, where β is the cost of inverting a 524×524 matrix. These results are 10^{15} times better than the exhaustive attack analyzed above.

5.3.3 Related-Message Condition

The message-resend condition, $\mathbf{m_1} + \mathbf{m_2} = \mathbf{0}$, can be generalized to any linear relation between the messages, for example $\mathbf{m_1} + \mathbf{m_2}$. Suppose we have two ciphertext messages, $\mathbf{c_1} = \mathbf{m_1SGP} + \mathbf{e_1}$ and $\mathbf{c_2} = \mathbf{m_2SGP} + \mathbf{e_2}$, then $\mathbf{c_1} + \mathbf{c_2} = \mathbf{m_1SGP} + \mathbf{m_2SGP} + \mathbf{e_1} + \mathbf{e_2} = (\mathbf{m_1} + \mathbf{m_2})\mathbf{SGP} + \mathbf{e_1} + \mathbf{e_2}$, which the attacker can calculate by applying the public key to the known relation $\mathbf{m_1} + \mathbf{m_2}$. Since $\mathbf{c_1} + \mathbf{c_2} + (\mathbf{m_1} + \mathbf{m_2})\mathbf{SGP} = \mathbf{e_1} + \mathbf{e_2}$, the attack can be done with the same procedure as the message-resend attack by using $\mathbf{c_1} + \mathbf{c_2} + (\mathbf{m_1} + \mathbf{m_2})\mathbf{SGP}$ instead of $\mathbf{c_1} + \mathbf{c_2}$, and the cost is the same also.

Remark: Note that this is a per-message attack. The plaintext is revealed with very little work, but the secret key remains safe. Early countermeasures include introducing an element of local randomness into a message before it is encrypted, and the randomness must be spread

through the plaintext in some complicated fashion. However, such a scheme extracts a penalty in data rate. The original McEliece cryptosystem was subject to this message-resend and related-message attack until the Niederreiter variant was discovered, which can completely eradicate this problem of encrypting the same message twice [6].

6 Strengthening the McEliece Cryptosystem

Several modifications to the McEliece cryptosystem have been suggested using different families of codes than the Goppa codes. Generalized Reed-Solomon (GRS) codes, Gabidulin codes, and Reed-Muller codes, are among those which have been tried [4]. The modifications using Reed-Solomon codes and Reed-Muller codes have been broken, and Gabidulin's system was shown by Gibson [4] to have little advantage over the original McEliece system.

In 2000 Pierre Loidreau [6] presented a modification to strengthen the McEliece cryptosystem against decoding attacks by using the automorphism group of Goppa codes and without increasing the size of the public key. The ciphertext messages are garbled by a larger amount of errors than what the code can correct. Yet because of the nature of the random error vector, the plaintext can be recovered. The private key of the modified system is not changed, but the public key involves an additional set called a t-tower decodable set.

Definition: Let $\mathcal{E}$ be a set of words of length $n = 2^m$, let $\mathbb{F}_{2^s}$ be a subfield of $\mathbb{F}_{2^m}$ and let $\sigma : z \to z^{2^s}$ be the Frobenius automorphism of the extension field. Then $\mathcal{E}$ is t-tower decodable if both of the following conditions hold.

1. For all $\mathbf{e} \in \mathcal{E}$, there exists a linear combination

$$E = \sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{e}),\ \epsilon_i \in \mathbb{F}_2$$

   having a Hamming weight less than the error-correcting capability, t, of the code.

2. The knowledge of E enables the receiver to recover $\mathbf{e} \in \mathcal{E}$ in a unique way.

Note that the linear combination maps words of length $n = 2^m$ to vectors with weight less than the correcting capability in a one-to-one fashion. This fact is crucial to the improvement of the system's security. For details on the construction of a t-tower decodable set, see [6].

Encryption: With the public key $\mathbf{G}' = \mathbf{SGP}$ and the public knowledge of how to generate a t-tower decodable set, $\mathcal{E}$, the sender can choose a random word $\mathbf{e}$ from the generated $\mathcal{E}$ and compute $\mathbf{c} = \mathbf{mG}' + \mathbf{e}$. Then the ciphertext, $\mathbf{c}$, is transmitted to the receiver.

Decryption: The receiver first computes $\mathbf{cP}^{-1} = \mathbf{mSG} + \mathbf{eP}^{-1}$. Since $\mathcal{E}$ is t-tower decodable, by part (1) of the definition, there is a linear combination, $E = \sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{e})$, with weight less than t. We have

$$\sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{cP}^{-1}) = \sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{mSG}) + \sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{eP}^{-1}).$$

Note that $\mathbf{mSG}$ is a codeword and since σ is an automorphism of the code, $\sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{mSG})$ is

also a codeword. The matrix $\mathbf{P}^{-1}$ is a permutation, so

$$\sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{e}\mathbf{P}^{-1}) = \left(\sum_{i=0}^{m/s-1} \epsilon_i \sigma^i(\mathbf{e}) \right) \mathbf{P}^{-1},$$

which is a decodable pattern. Thus, E can be recovered by applying the decoding algorithm, and the knowledge of E provides a unique way to find $\mathbf{e}$.

The complexity of encryption is the same as in the original system, but the decryption requires extra work and the cost depends on the structure of $\mathcal{E}$. In order for this scheme to be effective, $\mathcal{E}$ must satisfy the following conditions:

1. The elements of $\mathcal{E}$ must have weight larger than the error-correcting capability of the code. This condition increases the security of the cryptosystem against decoding attacks.

2. The set $\mathcal{E}$ has to be larger enough so that an exhaustive search of the error vectors is not feasible.

3. The way to generate $\mathcal{E}$ must not reveal helpful information to a potential attacker.

7 Conclusion

The McEliece cryptosystem is secure, but it is not currently employed due to the size of the public key and the approximate 50% data rate. Some methods have been found which increase the data rate up to 80% and compress the size of the private key [9]. However, we do not know how these methods would work with Loidreau's modification.

We believe that in the future the McEliece cryptosystem will become more appealing. As technology advances, the drawbacks to the system will become less of an obstacle so that Goppa codes with even larger parameters can be used. Also, since the number of Goppa codes increases exponentially with the length of the code and the degree of the generating polynomial, the work factor for the structural attacks against the system should actually increase.

The McEliece system provides an alternative to the current public key cryptosystems, which are based solely upon a few major problems in number theory. Encryption and decryption are much faster in the McEliece system since it requires the multiplication of a vector by a matrix whereas RSA requires raising a number to a power. Therefore, the ongoing study of this system is vital to the future of cryptography.

References

[1] T. Berson. *Failure of the McEliece Public-Key Cryptosystem Under Message-Resend and Related-Message Attack*, Advances in Cryptology–Crypto '97, pages 213-220, Berlin; New York: Springer, 1997.

[2] P. Fitzpatrick, J.A. Ryan. *On the number of irreducible Goppa codes*, Workshop on Coding and Cryptography 2003, 2003.

[3] E.M. Gabidulin, A.V. Paramonov, O.V. Tretjakov. *Ideals over a Non-Commutative Ring and their Application in Cryptology*, Advances in Cryptology–Eurocrypt '91, pages 482-489, Berlin; New York: Springer-Verlag, 1991.

[4] K. Gibson. *The Security of the Gabidulin Public Key Cryptosystem*, Advances in Cryptology–Eurocrypt '96, pages 212-223, Springer-Verlag, 1996.

[5] R. Heiman. *On the security of cryptosystems based on linear error-correcting codes*, M.Sc. Thesis, Feinberg Graduate School, Weitzman Institute of Science, Rehovot, 1987.

[6] P. Loidreau. *Strengthening McEliece Cryptosystem*, Springer-Verlag Berlin Heidelberg, 2000.

[7] V. Pless, *Introduction to the Theory of Error-Correcting Codes 1998 Edition*, John Wiley & Sons, Inc., Canada, 1998.

[8] N. Sendrier. *The Support Splitting Algorithm*, INRIA, 1999.

[9] H. Sun. *Enhancing the Security of the McEliece Public-Key Cryptosystem*. Journal of Information Science and Engineering 16, pages 799-812, 2000.