The McEliece Code-based Public Key Cryptosystem

Palmer Adonis Lao
laopa@clarkson.edu

Sean P. Lyons
lyonssp@clarkson.edu

December 11, 2014

Abstract

As the world draws closer to the era of quantum computing, there is a fear on the minds of cryptographers and those who depend on the cryptographic technologies in place in everyday life. Shor's algorithm, once it is useful in practice, has the ability to break systems, like RSA and ElGamal, that protect the world's most sensitive data. This paper presents a system developed by Robert McEliece. This code-based, public key cryptosystem, known as the McEliece cryptosystem, is one of few existing classic cryptosystems that cannot be broken by Shor's algorithm or other extensions of the Hidden Subgroup Problem. In our paper we will present some necessary exposition on linear codes, explanation of the hardness of decoding a linear code, and the details of the cryptosystem itself.

1 Error-correcting Codes

Common data transmissions are ridden with potential for error. These errors can take shape as a result of all kinds of physical imperfections in the channel over which data is being sent. Regardless, we don't seem to have any issues sending text messages over immense distances or receiving those messages when impurities effect them. This is a result of **error-correcting codes**. A common method of providing some facility to correct transmission errors is redundancy. As an example, suppose I want to send a single bit, 0. Now suppose as a way of correcting any error, I simply append two extra copies of the string to itself to get the message 000. Any corruption that occurs when I send this bit string across a channel will reflect on the string as flipped bits. Because the original message has been tripled, it is clear that 100 is not a valid bit-string, because there is no string that you can triple-duplicate to get 100. By the same token, 110 is also an invalid string. If the receiver sees behavior like this, they know there was an error in transmission. It is clear from the example that our error-correcting scheme can only **detect** up to 2 errors. That is, if all three bits get flipped due to transmission error, the receiver would be unaware because 111 is a valid bit string.

Now let's turn to the issue of **correcting** the error. If a single bit is flipped, rendering the bit string 100 on the receiving end, can the error be corrected? The answer is yes. If we assume that there has only been one transmission error, then we know that the original string only consisted of the bits that are the majority bits in the corrupted string. However, there is an issue here on the receiving end. How does the receiver know that the original string was not 111 and the received string does not reflect 2 transmission errors? The error-correcting scheme described is said to only **correct** 1 error, because once there is a chance that a second error may have occured, there is no way to recover the original message.

A couple of the properties hidden in the example above actually generalize to some important properties of linear codes of arbitrary length. Let us state some key definitions before proceeding.

Definition 1.1. A linear code of length n and rank k is a linear subspace C with $dim(C) = k$ of $\mathbb{F}_q^n$, where $\mathbb{F}_q$ is the finite field on q elements.

Definition 1.2. The distance between two codewords is the number of elements in which they differ. The distance of a linear code is the minimum distance between two distinct codewords in the code.

The example above is an example of the code $\{000, 111\}$. The codewords in this linear code are 000 and 111. The number of bits in which these two codewords differ, the distance of the code, is 3. A code $\{000, 001, 110, 111\}$ does not have the same distance as $\{000, 111\}$ because the distance is the **minimum** distance between two elements in the code. In the case of the 4-codeword linear code above, that minimum distance is 1.

With respect to error-detecting, the amount of errors a code can detect is dependent on the distance of that code. It is more straightforward to see that a code with distance d can correct $d-1$ errors. That is because, by the definition of distance, corrupting $d-1$ bits in a valid codeword will necessarily produce an invalid codeword. Both sender and receiver know that it is an invalid codeword because they know what the bit strings contained in the code are. That is why in the code $\{000, 111\}$, with distance 3, we can detect up to $3 - 1 = 2$ errors.

With respect to error-correcting, the amount of errors a code can correct is also dependent on the distance of that code. The amount of errors that a code with distance d can correct is $\lceil \frac{d}{2} \rceil - 1$. The reason for this is that once $\frac{d}{2}$ bits are corrupted, there is an ambiguity when it comes to determining what the original, uncorrupted codeword is. Hence, the code $\{000, 111\}$, with distance 3, can correct $\lceil \frac{3}{2} \rceil - 1 = 2 - 1 = 1$ error.

Throughout the rest of the paper we will refer to a linear code of length n, dimension k, and distance d as an [n,k,d] code or exclude the distance and say (n, k)-linear code. Additionally, when we talk about linear codes, we will talk about binary codes, i.e. linear codes of length n whose codewords are of the form $\{0, 1\}^n$.

2 Decoding a Linear Code

3 The McEliece Cryptosystem

The McEliece cryptosystem is an assymetric encryption scheme which derives its security from the hardness of the decoding general linear codes, as described in section 2 of this paper. The algorithm has garnered more attention after the realization that many common classical public-key cryptosystems will become insecure with the existence of effective quantum computers; the algorithms already exist and the current practicality of implementing the algorithms is the only roadblock. The cryptosystem leans on three different algorithms: a key generation algorithm, an encryption algorithm, and a decryption algorithm. Of these three algorithms, only the decryption algorithm is deterministic. The key generation and encryption algorithms both rely on randomization, meaning there is no strict guarantee that instance of the algorithm will generate different keys, and encrypting the same plaintext with the same keys will not necessarily yield the same ciphertext. In this section we will explore these three algorithms in detail for a receiver, Alice, a sender, Bob, and a potential evesdropper, Eve.

3.1 Key Generation

1) Alice selects a binary (n, k)-linear code C capable of correcting t errors. This code must possess an efficient decoding algorithm and generates a $k \times n$ generator matrix G for the code C.

2) Alice selects a random $k \times k$ binary non-singular matrix S.

3) Alice selects a random $n \times n$ permutation matrix P.

4) Alice computes the $k \times n$ matrix $\hat{G} = SGP$

5) Alice's public key is $(\hat{G}, t)$; her private key is (S, G, P)

The details of the nature of the linear code we are dealing with should be sufficiently explained by previous sections. However, we have not touched on the idea of a generator matrix. So recall that an (n, k)-linear code C has dimension k, meaning that a basis for C consists of k vectors. The generator matrix has k rows because, by definition, the rows of a generator matrix form the basis for a linear code.

Note that there are 2^{k^2} possible matrices that S could be and 2^{n^2} possible matrices that P could be. For that reason, we can safely assume that factoring the public key $\hat{G}$ in order to crack this encryption scheme is not trivial.

3.2 Encryption

1) Bob encodes the message m as a binary string of length k.
2) Bob computes the vector $c' = m\hat{G}$.
3) Bob generates a random n-bit vector z containing exactly t ones.
4) Bob computes the ciphertext as $c = c' + z$ and sends it to Alice over an insecure channel.

We see a pattern in this encryption function similar to that of RSA and ElGamal. The message is masked with the public key under an operation that is not trivially reversed, except by the receiver who has access to the secret keys.

3.3 Decryption

1) Alice computes P^{-1}
2) Alice computes $\hat{c} = cP^{-1}$
3) Alice uses the decoding algorithm for the code C to decode $\hat{c}$ to $\hat{m}$
4) Alice computes $m = \hat{m}S^{-1}$