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 used in everyday life. Shor's algorithm, once it is useful in practice, has the ability to break systems like RSA and El-Gamal, 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. A generator matrix G of a $[n, k, d]$ linear code is a $k \times n$ matrix with row space equivalent to C.

Definition 1.3. 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, or equivalently, the minimum weight of any nonzero codewords.

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.

2 Linear codes and cryptosystems

In theory, most public-key cryptosystems are as hard to break as the one-way functions they are typically based on. For example, the security of RSA is based on the fact that factoring large composite numbers is (thought to be) exponentially harder than generating and multiplying large primes. Another commonly seen example of one such system is elliptic curve ElGamal, which is based on the discrete log problem over finite Abelian groups. However, researchers have discovered quantum algorithms that solve both of these problems, and more generally, the Hidden Subgroup Problem restricted to finitely generated Abelian groups in polylogarithmic time.

A natural question to ask is whether or not there are public-key cryptosystems that can be both implemented on classical computers but still be safe from quantum attacks. One such difficult problem associated with linear codes is the Nearest Codeword Problem (NCP). Given a generator matrix G for a code C, and a (possibly noisy) observation of a codeword y, the nearest codeword problem is to find a vector x so that $d(xG, y)$ is minimized (where d represents the Hamming distance). It can be shown that this problem is NP-Hard through a series of reductions through Max-Cut, Not-All-Equal 3-SAT, and finally, 3-SAT.

This very naturally suggests a cryptosystem based on the NCP. With a linear code, one can simply encode a message and flip some correctable number of bits to encrypt the message. But then to recover the message, someone needs to solve the NCP. However, the hardness of NCP is reliant on the fact that the code could be any linear code. It's often the case that certain specializations of NP-Hard or NP-Complete problems can be solved quickly. For example, a special case of the Subset Sum problem is solved in polynomial time by cashiers everyday because the set of American coin values is superincreasing. Similarly, there exist linear codes where it is fast to solve the NCP. We discuss examples of such codes later in these notes.

The only remaining obstacle to a public-key cryptosystem based on the hardness of the NCP is to hide or obscure the properties of your chosen code that make it efficient to decode from eavesdroppers. Since we're representing the linear code as a generator matrix, we can apply regular rules of linear algebra to "hide" the original generator matrix. Multiplying and adding rows and columns, as well as swapping rows in the generator matrix preserves the row space and thus the code specified by the generator matrix.