The McEliece Elliptic Curves System

Zach Conway and Lena Pang
Codes and Ciphers with Dr. Anurag Agarwal
Final Project

April 29, 2024

Introduction

With the rise of quantum computing, many cryptosystems are under siege. But one system—the McEliece system, based on Goppa codes from curves—has resisted the threat of being efficiently broken (at least thus far). The complexity of the encryption and decryption scheme leaves adversaries to time-consuming, inefficient hacking attempts.

Elliptic curves, on the other hand, are a class of algebraic curve whose points can be united under a particular geometric operation to form a group structure with highly fascinating symmetries (that even hold when carried into the discrete realm of finite fields). Since the discrete logarithm problem within this group is computationally intractable, and the group's behavior under repeated point multiplication can be highly chaotic, its use can find value in several different cryptographic protocols from key exchange to random number generation.

In this paper, we will discuss how McEliece cryptosystem is constructed from linear Goppa codes and error correction, as well as implementations of elliptic curves in cryptographic protocols in modern times.

History

One major application of elliptic curves is the study of number theory. Their use in this context can be found as early as the early 300s AD, where they were studied with regard to Diophantine Equations, or polynomial equations whose solutions have integer or rational values. In particular, Andrew Wiles used elliptic curves and their relationship with modular forms to write a proof of Fermat's Last Theorem, the final version of which was published in 1995. While the proof is heavily beyond our scope, elliptic curves provided a medium for efforts in otherwise vastly disparate fields of mathematics to be synergized in a concrete proof of a previously unsolved problem.

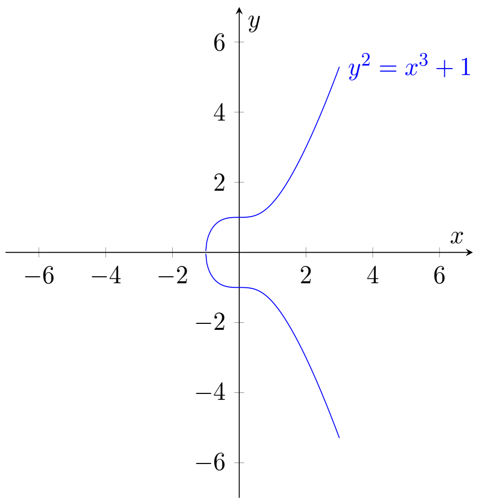
In modern times, as computers became more prevalent in cryptography, the need for efficient and secure algorithms and mathematical structures grew as well. The group structure behind elliptic curves turned out to be deeply applicable in this area for several different protocols, but we'll elaborate more on those once the mathematics behind elliptic curves is established.

Background & Theory

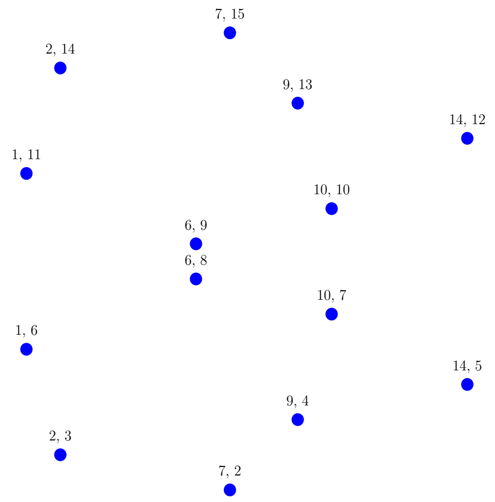
Elliptic Curves

Elliptic curves are of the form $y^2 = a_3x^3 + a_2x^2 + a_1x + a_0$. They are symmetrical across the x-axis. For the sake of this paper, we will focus on elliptic curves over finite fields, which creates a more discrete plot like the graph on the right.

Graph of $y^2 = x^3 + 1$ over $\mathbb{F}_{\mathbb{R}}$



Graph of $y^2 = x^3 + 1$ over $\mathbb{F}_{17}$



We can treat an elliptic curve as a **group**. The group elements are all points (x_i, y_i) of a given elliptical curve (of the form $y^2 = a_3x^3 + a_2x^2 + a_1x + a_0$), along with the identity element, which we consider a point at "infinity" that lies on every vertical line. The group operation is elliptic curve addition. To add any two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, you draw a line connecting the two, and reflect the intersection across the x-axis to produce the new point. (If you add a point to itself, you use the tangent line at that point to get your new intersect. If you add two points that are on the same vertical line, you get the "infinity" identity element.) Elliptic curve groups over finite fields implement the same idea but with modulo, reducing the total number of elements to a much smaller set. They can be cyclic, but are not always.

Goppa Codes

An $[n, k]$ linear code is, in essence, a linear subspace containing all possible code vectors (or code words). k is the dimension of the linear code, and n is the length of any given code vector. **Goppa codes**, in particular, can be constructed by using irreducible functions over finite fields.

We start by first defining our **function** $g(x)$ over $\mathbb{F}_q^n$, where q is the mth power of some prime p. With binary Goppa codes, we use $\mathbb{F}_{2^m}^n$. The function must be square-free (or ideally irreducible, meaning it cannot be divided by a lesser-degree polynomial), and we denote the degree of freedom of $g(x)$ as t.

We then define the **support**, a finite subset of our chosen field $\mathbb{F}_{2^m}^n$: $L = \{\alpha_1, \alpha_2, ...\alpha_n \mid g(\alpha_i) \neq 0\}$. This contains a finite number of elements for which, when plugged into our function $g(x)$, the outcome is nonzero.

We also define the **syndrome** of a code vector c:

$$S_c(x) = -\sum_{i=0}^{n-1} \frac{c_i}{g(\alpha)} \frac{g(x) - g(\alpha_i)}{x - \alpha_i} \mod g(x)$$

Finally, we can then create our Goppa code, $\Gamma(g, L)$ as a collection of code vectors c in $\mathbb{F}_{2^m}^n$ for which the syndrome is 0, in other words $S_c(x) = 0$:

$$\Gamma(g, L) = \{c \in \mathbb{F}_{2^m}^n \mid \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \equiv 0 \mod g(x)\}$$

This equivalency qualifier implies a few things. Firstly, for all α_i in L, $x - \alpha_i$ must have an inverse modulo $g(x)$, because $\frac{1}{x-\alpha_i} \equiv (x - \alpha_i)^{-1} \pmod{g(x)}$. Secondly, every Goppa code must include the zero vector (or zero "word"), $\vec{0}$, which is the same length as all the other code vectors but comprised entirely of zeroes as every component.

From any Goppa code, we can create two matrices: a generating matrix that produces code vectors, and a parity check matrix, that can identify and locate non-code vectors.

The **generating matrix** $G_{k\times n}$ has rows that form the basis vectors of the Goppa code subspace. When in reduced row echelon form, it is of the form $\left[I \mid P\right]$ where I is an $n \times n$ identity matrix, and P simply consists of the rest of the components of each Goppa code basis vector.

The **parity check matrix** $H_{n-k\times n}$ is a matrix representation of the syndrome, so it will produce $\vec{x}H^T = \vec{0}$ for any x that is a code vector. For any code vector with error, however, it will produce a vector that is equal to one of the rows of H^T; whichever row, say the ith row, is the location of the error in x (the ith component of x). It is of the form $\left[-P^T \mid I\right]$.

An example generating matrix (already in reduced row echelon with row swaps to fit the $\left[I \mid P\right]$ form) and parity check matrix for a $[5, 3]$ are listed below.

$$G_{k\times n} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}, H_{n-k\times n} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Error Correction

When there is only 1 error in a given vector, the parity check matrix can "locate" the error. Let a vector $m = c \oplus e$ be composed of a code vector c and an error vector e (of degree 1, so there is only one error). Performing the multiplication $H\vec{m}^T$ will produce a non-zero vector identical to the ith column of H, and the error is subsequently identified as the ith component of m. (And when working in binary, it is easy to fix the error after it is found: simply flip the value, and you now have the corrected code vector.)

However, Goppa code implementation uses many errors (up to degree t, as previously established). For example, when McEliece first published his cryptosystem, he used a $[1024, 524]$ Goppa code, which can correct as many as 50 errors. As such, the error correction process becomes a bit more complicated.

For Goppa codes, specifically, we know that any vector $m = c \oplus e$ will be a code vector with up to t errors. Taking the syndrome of the equation, we get $S_m(c) \equiv S_c(x) + S_e(x) \mod g(x)$. By the definition of our Goppa code vectors, $S_c(x) \equiv 0 \mod g(x)$, so we can reduce the equivalency to $S_m(x) \equiv S_e(x) \mod g(x)$.

From the syndrome, we can create an **error locator polynomial** for errors of up to degree t—in other words, this polynomial can isolate our c from $m = c \oplus e$. It is defined as follows:

$$\sigma_c(x) = \prod_{j \in \tau_c} (x - \alpha_j), \text{ where } \frac{\sigma_c'(x)}{\sigma_c(x)} \equiv S_e(x) \mod g(x)$$

While the proof is beyond the scope of this paper, we can separate this into $\sigma(X) = A^2(x) + xB^2(x)$—squares and non-squares—for any polynomial, where the degree of the A is $\leq \frac{t}{2}$ and the degree of B is $\leq \frac{t-1}{2}$. Then we apply the Extended Euclidean Algorithm repeatedly to reduce the degrees of A and B until they meet the requirements, and then apply the polynomial to $m = c \oplus e$ identify the erroneous bits. We can then flip those bits, and are left with the error-free code vector c.

Encryption & Decryption

The McEliece cryptosystem is based on matrix encryption and intentional error that we can fix (and subsequently decrypt) using Goppa codes. The process requires:

- Private key: (G, S, P)
    - Goppa polynomial $g(z)$ of degree t with a corresponding Goppa code $[n, k]$, that can be used to produce generator matrix $G_{k \times n}$
    - random binary invertible/non-singular matrix $S_{k \times k}$
    - random permutation matrix $P_{n \times n}$
- Public key: (t, G')
    - degree t of Goppa polynomial

    - matrix $G'_{k \times n} = SGP$

<u>The encryption process</u>:

1. If necessary, convert message to binary strings m.
2. **Permutation and Linear Transformation**: Encrypt m plaintexts into Goppa codewords by computing mG' (from the public key).
3. **Error Creation**: Choose a random error vector e of weight t or less (has t or fewer non-zero components; also known from the public key) and add it to a string.
4. Send over $y = mG' + e$.

<u>The decryption process</u>:

1. **Invert Permutation**: Calculate P^{-1} and use it to compute $y' = yP^{-1}$.
2. **Correct Errors**: As described in *Background & Theory: Error Correction*, construct, iterate, and apply the error polynomial from the Goppa code, and flip the identified error bits.

   Without knowing G or the Goppa code, this step is incredibly difficult (and due to the nature of matrix multiplication, it is very difficult to isolate G from $G' = SGP$ without knowing S and P).
3. **Invert Linear Transformation**: Calculate S^{-1} and use it to compute the plaintext binary $m = m'S^{-1}$.
4. If necessary, convert m from binary back into readable plaintext.

In terms of efficiency, the encryption process is very fast (simple matrix multiplication and adding in some randomness), and the decryption process—with knowledge of the private key and error locating algorithm—is also relatively direct. Some keys are more "lightweight" than others, but that depth is out of the scope of our project.

The Goppa code-based structure makes the McEliece cryptosystem one of a few systems to withstand the upcoming threat of quantum computing, lending it credibility and security in these rapidly changing times.

Applications & Examples

Elliptic Curve Diffie-Hellman Key Exchange

An elliptic curve's group structure, along with a generator for a cyclic group on that curve, can be used exploited for any algorithm or protocol that relies on the difficulty of the Discrete Logarithm Problem within a group. One such application is a variation on Diffie-Hellman Key Exchange using repeated addition of points on the curve as opposed to modular exponentiation.

We take an elliptic curve E over a finite field of prime order, $\mathbb{F}_p$. In traditional Diffie-Hellman, we publicize a primitive root g as well as the prime p for the modulus we're working in; we do something similar here, where instead we pick a point P in E that generates a cyclic group of points in E and publicize P and E.

Suppose Alice would like to agree on a shared secret key with Bob. Alice can take a secret natural number a and calculate aP. Likewise, Bob can calculate bP, and the two exchange these points over an insecure channel. Due to the difficulty of the Discrete Logarithm Problem on the group formed by elliptic curves, it is computationally intractable for a third party to deduce a or b from the available public information (P, E, aP, bP). Now, both Alice and Bob can multiply what they received by their respective secret numbers. Since this scalar multiplication/group exponentiation commutes, Alice's $a(bP)$ is equal to Bob's $b(aP)$, and they've agreed upon a shared key.

Elliptic Curve Cryptosystem (ElGamal)

Similarly, the notion of exponentiation of a generator and a variation on the Discrete Logarithm Problem allows elliptic curve groups to define a variation of the ElGamal cryptosystem.

We start similar to the Diffie-Hellman setup: Alice publicizes a generating point P on an elliptic curve E over a finite field $\mathbb{F}$. She also selects a secret number a and publicizes aP.

Now, if Bob would like to send Alice a message m (encoded as a point on E), he first selects a random number k and calculates $c_1 = kP$ and $c_2 = m + k(aP)$. These two points collectively forms his ciphertext, and he sends this over an insecure channel to Alice.

Alice can use $c_1 = kP$ and her secret number a to compute akP, then she can compute $c_2 - akP = m + akP - akP = m$, recovering the original message.

Someone watching the public channel has access to E, P, aP, kP, and $m + akP$. However, due to not being able to retrieve a or k (Discrete Logarithm Problem), they have no feasible means of recovering the original message.

Elliptic Curves Cryptography vs. RSA?

We demonstrated different cryptographic protocols that utilize the Discrete Logarithm Problem of elliptic curves, but how do these protocols compare to those that leverage the discrete logarithm problem on regular modular exponentiation, like the RSA cryptosystem?

RSA has a similar use-case to elliptic curve cryptography as a public-key cryptosystem, as well as the original Diffie-Hellman compared to the elliptic curve variation. The value behind elliptic curves is their *efficiency* in comparison to RSA, specifically with regard to key size as well as the time taken to generate keys, encrypt, decrypt, and perform digital signatures. Researchers Soram Ranbir Singh, Ajoy Kumar Khan, and Soram Rakesh Singh used Java 7 to implement and evaluate RSA against elliptic curve cryptography for varying levels of security.

The resulting data showed the public key generation and decryption times to be several orders of magnitude faster for elliptic curves than for RSA, while encryption times were still a fraction of the encryption times for RSA. This differential becomes more pronounced as the key size increases, especially for decryption. It's also worth noting that the overall key size necessary to achieve a certain level of security is much smaller for elliptic curves than for RSA. For instance, the security achieved by a key of 7680 bits in RSA can be achieved with a key only 384 bits long in elliptic curve cryptography.

Additional research also showed that elliptic curve cryptography can overtake RSA on computational architectures with particularly small word sizes (the number of bits packets of information are broken into when accessing memory). Thus, elliptic curves also present themselves as an extremely powerful option when used in smaller devices with constraints on power or budget, such as sensors or other data collecting hardware that have to communicate over networks.

Elliptic Curves for Pseudorandom Number Generation (and a backdoor)

A scheme using elliptic curves to perform pseudorandom number generation was proposed at an NIST workshop in 2004, called Dual EC due to its use of two points P and Q on an elliptic curve (standard is the NIST P-256 curve). The random number generator, starting with some random 256-bit integer s_0 as an initial state (achieved from some external source of randomness), utilizes multiplication on both P and Q to produce random numbers. Each time a random number needs to be generated, the current state s_k is multiplied with P to produce a new point, whose x-coordinate is taken to be s_{k+1}. The random number is then produced from this internal state by taking the x-coordinate of $s_{k+1}Q$.

Supposedly, by the difficulty of the elliptic curve Discrete Logarithm Problem, it should be unfeasible to infer the internal state of the random number generator from the numbers being outputted. However, a backdoor exists that can be exploited in this mechanism.

Suppose there exists some number d such that $P = dQ$. An attacker can take a given output, r_1, and use the elliptic curve to infer a y-coordinate and consider a point R on the curve, which should be s_1Q for whatever state s_1 the random number generator is in. If the attacker calculates the $dR = d(s_1Q) = s_1(dQ) = s_1P$.

While the attacker can't know s_1 from this, the x-coordinate of this new point is defined as the next state s_2, meaning the attacker can now start predicting and generating the "random" numbers ahead of time. While the most significant 16 bits are often discarded from the output, this leaves 2^{16} possible values that r can be, which is feasible for the attacker to brute-force until they find the correct state.

While this protocol received criticism from academics Kristian Gjøsteen, who proposed a vulnerability in the system, as well as a deepening of this wound in the protocol's security by Berry Schoenmakers and Andrey Sidorenko. Despite this, the NIST (National Institute of Standards and Technology) published a standard in June 2006 including the unrepaired Dual EC protocol.

This was particularly damaging considering the use of Dual EC in the implementation of TLS by several libraries, such as OpenSSL, BSafe, and SChannel. TSL, or Transport Layer Security, is a standard protocol for HTTPS web security, meaning a significant deal of power was held by anyone who knew this backdoor number. Significant suspicion in this regard is held towards the NSA, who supposedly originated the protocol and also paid \$10,000,000 to RSA in a deal for them to use Dual EC as their random number generator in BSafe. While the vulnerabilities and backdoor had come to light in 2007, and wariness around them remained since, information regarding the NSA's influence in pushing Dual EC had not been brought into public knowledge until around 2013-2014.

Conclusion

In this paper, we discussed the history and several applications of elliptic curves throughout time, including number theory and Fermat's Last Theorem earlier on and cryptographic protocols like key exchange, public-key cryptography, and random number generation in modern times (as well as an exploitation of elliptic curve structure in this area). We learned about how the geometry of elliptic curves yields group structure, producing a Discrete Logarithm Problem that can be leveraged for these cryptographic purposes. The central cryptosystem we studied, the McEliece cryptosystem, utilized Goppa linear codes constructed from curves, which correct errors as part of the decryption process.

We saw how elliptic curves could provide protocols for cryptography with similar security yet vastly improved efficiency and reduced key size compared to the original methods.

Elliptic curves and the McEliece cryptosystem turned out to be extremely fascinating topics of study, with the symmetries of elliptic curves giving way to an unexpected system of mathematics and McEliece taking advantage of intricate systems and deliberate use of error correction. While we learned a lot, there was content beyond our reach within the time we had; digesting the mathematics behind the McEliece cryptosystem and connecting the generation of linear codes from elliptic curves; the actual generation of Goppa codes demanded too much prerequisite knowledge for us to give us the attention it needed while staying within the scope of the project, though it would be an interesting direction of research later on. Additionally, the underlying math regarding curves, genus, and projective geometry can get extremely involved, but we may have the opportunity to learn about them later on in our career.

As a whole, however, we gleaned a lot of knowledge not only concerning this cryptosystem but with regard to codes, error correction, and the creativity with with cryptographic systems can be formulated and executed.

References

Avanzi, R., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., & Vercauteren, F. (2005).*Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Chapman & Hall/CRC.

Bernstein, D.J., Lange, T., Niederhagen, R. (2016). Dual EC: A Standardized Back Door. In: Ryan, P., Naccache, D., Quisquater, JJ. (eds) The New Codebreakers. *Lecture Notes in Computer Science()*, vol 9100. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-49301-4_17

Engelbert, D., Overbeck, R., Schmidt, A. (2007). A Summary of McEliece-Type Cryptosystems and their Security. de Gruyter.
https://www.degruyter.com/document/doi/10.1515/JMC.2007.009/pdf

Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.C. (2004). Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In: Joye, M., Quisquater, JJ. (eds) Cryptographic Hardware and Embedded Systems - CHES 2004. CHES 2004. Lecture Notes in Computer Science, vol 3156. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-28632-5_9

Marcus, M. (2019). White Paper on McEliece with Binary Goppa Codes.
https://www.hyperelliptic.org/tanja/students/m_marcus/whitepaper.pdf

McEliece, R. J. (2004). The Theory of Information and Coding (2nd ed). *Encyclopedia of Mathematics and its Applications.* Cambridge University Press.

Naveed Ahmed Azam, Ikram Ullah, Umar Hayat (2021). A fast and secure public-key image encryption scheme based on Mordell elliptic curves. *Optics and Lasers in Engineering*, Volume 137
https://doi.org/10.1016/j.optlaseng.2020.106371.

S. R. Singh, A. K. Khan and S. R. Singh (2016). Performance evaluation of RSA and Elliptic Curve Cryptography, 2016 2nd International Conference on Contemporary

Computing and Informatics (IC3I), Greater Noida, India, 2016, pp. 302-306, doi: 10.1109/IC3I.2016.7917979.

Valentijn, A. (2015). Goppa Codes and Their Use in the McEliece Cryptosystems. Syracuse University Honors Program Capstone Projects. 845. https://surface.syr.edu/honors_capstone/845