Introduction to Cryptography Lab: Elliptic Curves

Student: ChingWei Hsieh
Matriculation Number: 2350179
Professor: Heike Neumann

Contents

Abstract

In this paper we focus on researching two elliptic curves with key lengths 256 and 512 bit. The two respective curves names we are implementing from brainpool[1] are: ‘brainpoolP512r1’ and ‘brainpoolP256r1’. Brainpool provides several elliptic curve domain parameters over finite prime fields for use in cryptographic applications. The domain parameters are consistent with the relevant international standards, and can be used in X.509[2] certificates and certificate revocation lists (CRLs), for Internet Key Exchange (IKE), Transport Layer Security (TLS), XML signatures, and all applications or protocols based on the cryptographic message syntax (CMS)[3][4].

The goal is to implement algorithms in Python regarding elliptic curve point multiplication and other elliptic curve operations such as point addition and also taking into account the infinity point. Additionally, we make performance measures for the implemented algorithms especially on point multiplication. Note that the performance measurements may have some variance, so we decided on how many measurements we made and applied basic statistics like mean values, variances etc. The test strategy we acknowledged is recognized by using the test vectors[5] provided by Brainpool in order to make sure that the algorithm we implemented is a correct ECC point implementation.

[1] https://datatracker.ietf.org/doc/html/rfc5639
[2] https://www.itu.int/en/ITU-T/C-I/conformity/Pages/Cschemes.aspx
[3] https://datatracker.ietf.org/doc/html/rfc5652
[4] https://www.ietf.org/standards/
[5] https://datatracker.ietf.org/doc/html/rfc7027

Introduction to Elliptic Curves

Elliptic-curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The primary benefit of ECC allows smaller keys compared to non-EC cryptography to provide equivalent security furthermore reducing storage and transmission requirements.(i.e. that an elliptic curve group could provide the same level of security afforded by an RSA-based system with a large modulus and correspondingly larger key: for example, a 256-bit elliptic curve public key should provide comparable security to a 3072-bit RSA public key.)

The use of elliptic curves in cryptography was suggested independently by Neal Koblitz[6] and Victor S. Miller[7] in 1985. Elliptic curve cryptography algorithms entered wide use in 2004 to 2005. At the RSA Conference 2005, the National Security Agency (NSA) announced Suite B which exclusively uses ECC for digital signature generation and key exchange. The suite is intended to protect both classified and unclassified national security systems and information.

Elliptic curves are applicable for encryption, digital signatures, pseudo-random generators and other tasks. They are also used in several integer factorization algorithms that have applications in cryptography, such as Lenstra elliptic-curve factorization. Elliptic curve cryptography is also widely used by the cryptocurrency Bitcoin and Ethereum version 2.0 in recent years. Ethereum version 2.0 makes extensive use of elliptic curve pairs using BLS signature[8] for cryptographically assuring that a specific Eth2 validator has actually verified a particular transaction.[9]

---

[6] https://en.wikipedia.org/wiki/Neal_Koblitz#Biography
[7] https://en.wikipedia.org/wiki/Victor_S._Miller
[8] https://en.wikipedia.org/wiki/BLS_digital_signature
[9] https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

Methods

For current cryptographic purposes, an elliptic curve is a plane curve over a finite field which consists of the points satisfying the equation:

$$y^2 = x^3 + ax + b$$

along with a distinguished point at infinity, which can be obtained when a point is multiplied by 0. Therefore we have to respect mathematical definitions and take into account operations that are done with the infinity point. Further details are explained in the point multiplication/point addition section. The point of Infinity would be represented as O in our report.

The elliptic curve cryptography (ECC) uses elliptic curves over the finite field $\mathbb{F}$p (where p is prime and p > 3). This means that the field is a square matrix of size p x p and the points on the curve are limited to integer coordinates within the field only. All algebraic operations within the field (like point addition and multiplication) result in another point within the field. The elliptic curve equation over the finite field $\mathbb{F}$p takes the following modular form:

$$y^2 = x^3 + ax + b \,(mod\, p)$$

Therefore, we could use this idea to acquire a secret number which could be used as a private key and perform point multiplication for this private key with a Generator point and acquire a point that is also on the curve which could be used as our public key.

It is very fast to calculate P = k * G, using the well-known ECC multiplication algorithms in time log_2(k_), e.g. the "double-and-add algorithm". For 256-bit curves, it will take just a few hundreds simple EC operations. It is extremely slow (considered infeasible for large k) to calculate k = P / G.

This asymmetry (fast multiplication and infeasible slow opposite operation) is the basis of the security strength behind the ECC cryptography, also known as the ECDLP problem. In the ECC cryptography, many algorithms rely on the computational difficulty of the ECDLP problem over carefully chosen field $\mathbb{F}$p and elliptic curve, for which no efficient algorithm exists.

Important Parameters for an Elliptic Curve

These are parameters that are necessary to define our Elliptic Curve and could also be found when looking through Brainpool or NIST. Cryptographers carefully selected the elliptic curve domain parameters (curve equation, generator point, cofactor, etc.) to ensure that the key space is large enough for certain cryptographic strength. Therefore it is recommended to take a predefined curve from the documents instead of defining one's own curve, as there may be many potential problems as the official curves have all been verified and tested.

p is the order of the curve (total number of all EC points on the curve, which also includes the point at infinity.) and could also correspond to our field for our elliptic curve equation. It is important that p is a prime as it may cause unwanted results if not carefully selected.

h is the curve cofactor (the number of non-overlapping subgroups of points, which together hold all curve points) For curves with cofactor = 1 there is only one subgroup and the order n of the curve is equal to the total number of different points over the curve, including the infinity.

x, y is our Generator point G (or also called base point)
When the generator point G and p are carefully selected, and the cofactor = 1, all possible EC points on the curve (including the special point infinity) can be generated from the generator G by multiplying it by integer in the range [1...p].

A, B responds to our elliptic curve parameter defined and could be used in the equation

Point Multiplication/Point Addition/Double and Add

Two points over an elliptic curve (EC points) can be added and the result is another point. This operation is known as EC point addition. If we add a point P to itself, the result is P + P = 2 * P. If we add P again to the result, we will obtain 3 * P and so on, this is how EC point multiplication is defined. In result, when we perform point multiplication we are still using point addition as our ground level operation to obtain desired results. In order to increase efficiency and resources a common developed method, double and add is now widely used to assist this operation.

Double and add:
There are four cases in Point multiplication we have to take into consideration before performing double and add algorithms. (here we already assume that the scalar Q (our private key) we multiply with P (our generator point) is already within or equal to our field). It is necessary to ensure that all the operations we are planning to perform later on are inbound and in correct signs.

Point multiplication cases:

1. If Q = O (point of infinity), then we get O as a result.
2. If Q%n == 0, we get O as result because the point is on the border of our field
3. If Q < 0, we take the negation of our y coordinate of P (generator point) as our new generator point due to the curve characteristics being symmetric.[10]
4. When neither our private key falls into any category above, then we can proceed to point addition operation without further adjustments.

[10] https://andrea.corbellini.name/ecc/interactive/reals-mul.html?a=-3&b=1&px=0&py=1

Double and add:
We transform the value into binary representation and Iterate over all bits starting by the LSB perform double and add algorithm. If the bit is a 1, we “add”. If the cofactor of the bit is a 0, “double”. Its principle of operation can be better explained with an example.

Take n=151, Its binary representation is 10010111, this binary representation can also be turned into a sum of powers of two:

$151 = 1\cdot2^7 + 0\cdot2^6 + 0\cdot2^5 + 1\cdot2^4 + 0\cdot2^3 + 1\cdot2^2 + 1\cdot2^1 + 1\cdot2^0$

$= 2^7 + 2^4 + 2^2 + 2^1 + 2^0$

$151\cdot P = 2^7\cdot P + 2^4\cdot P + 2^2\cdot P + 2^1\cdot P + 2^0\cdot P$

In the end, we can compute 151·P performing just seven doublings and four additions.

Point addition cases:
We come to this operation whenever we perform both “double” and “add”, it is important to redefine/override the addition arithmetic method so we could define cases and expect the program to perform accordingly.

1. If Q = O (point of infinity), then P+Q=P.
2. If x0 from P = x1 from Q, then P+Q=O.
3. If x0 from P ≠ x1 from Q, we then take P + Q to be −R, the point opposite R. Normal point addition could be defined as:

$$P + Q = R$$
$$(x_p, y_p) + (x_q, y_q) = (x_r, y_r)$$

Assuming the elliptic curve, E, is given by y2 = x3 + ax + b, this can be calculated as:

$$\lambda = \frac{y_q - y_p}{x_q - x_p}$$
$$x_r = \lambda^2 - x_p - x_q$$
$$y_r = \lambda(x_p - x_r) - y_p$$ [11]

Performance measurements

In this section, we wish to make simple performance measurements for solely the point multiplication method. One way to do so would be monitoring the elapsed time for the function that we wish to measure. There are various libraries that we could apply and take into account for our code, but after doing some research[12] it is discovered that the “timeit” library may be the most optimal in this case. Nevertheless, we still collected mean values, standard deviations and variances that were measured by using the “timer” library for comparison.

---

[11] https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication#Point_at_infinity

[12] https://stackoverflow.com/questions/7370801/how-to-measure-elapsed-time-in-python?page=1&tab=votes#tab-top

Count = 10	BrainpoolP256r1: timer()	BrainpoolP256r1: timeit()	BrainpoolP512r1 timer()	BrainpoolP512r1: timeit()
Mean	0.049112821	0.05046932	0.2335783	0.23845077
Standard Deviation	0.0050900397	0.0027207231	0.024205423	0.015508393
Variance	2.5908504E-5	7.4023344E-6	5.8590249E-4	2.4051025E-4

Verification with test factors

In order to verify that the algorithm implemented is correct, it is necessary that we do this verification with test vectors released on the official papers, in this case we use both “rfc6932”[13] and “rfc7027”[14]. The methodology on how this works is, we are provided some “tested” private keys and expected public keys when we make operations on the curve. The test vector was based on performing a simple Diffie-Hellman[15] example but since our report is mainly focused on Elliptic Curves, we merely test whether the curve correctly calculates the expected results. A minor complication while running these test vectors would be comparing different data types, our code gives us integer type as result for our public key coordinate meanwhile in the appendix the test vectors are in hexadecimal format, therefore it is necessary to convert them into same types which would be integer type in this case, and compare whether the x-coordinate calculated and y-coordinate calculated is respectively equal as expected in our appendix.

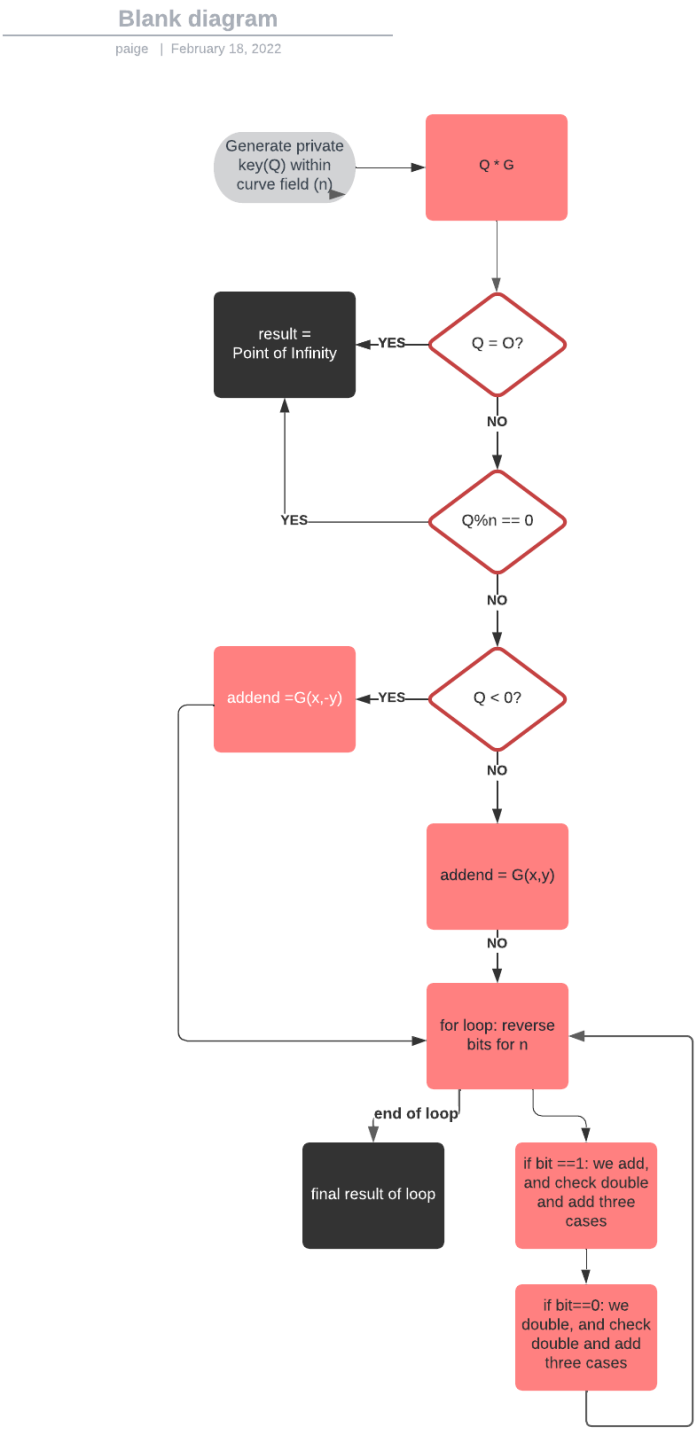
As a result, we have successfully obtained the expected outcomes from the test vectors.

[13] https://datatracker.ietf.org/doc/html/rfc6932#appendix-A.2

[14] https://datatracker.ietf.org/doc/html/rfc7027#appendix-A.3

[15] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#:~:text=The%20Diffie%E2%80%93Hellman%20key%20exchange%20method%20allows%20two%20parties%20that,using%20a%20symmetric%2Dkey%20cipher

Flow Diagram



References

1. https://datatracker.ietf.org/doc/html/rfc7027#page-7
2. https://en.wikipedia.org/wiki/Elliptic_curve
3. https://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/
4. https://lucid.app/lucidchart/b82ef584-67a1-45f5-803b-190ebcf1497b/edit?beaconFlowId=917838B6B4AA4C2C&invitationId=inv_c7f2c2c7-eaae-4e77-ba88-13d77ac7a7c0&page=8NqHfv-d0e2e#
5. https://datatracker.ietf.org/doc/html/rfc7027#page-7
6. https://www.mobilefish.com/services/cryptocurrency/cryptocurrency.html#refPrivateKeyHex
7. https://www.youtube.com/watch?v=vQ1-bQ4Jt5U