JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

695.744 - Reverse Engineering and Vulnerability Analysis

Professor - Tom McGuire

Research Paper

Due date: 08/20/23

INDEX

Abstract

This research paper aims to offer insights into the Elliptic Curve Cryptography (ECC) concept and its practical applications. It explores the mathematical foundations for implementing ECC in projective space through Finite Fields and Abelian Groups. The paper concludes by analyzing the attributes, encoding methodologies, encryption, and decryption procedures of ECC, shedding light on the algebraic structures that underpin secure cryptographic operations. The paper also identifies security implications and potential vulnerabilities, thereby providing a holistic grasp of ECC's significance in modern cryptographic security.

Foundations

Abelian Groups

Group: Let S be the non-empty set equipped with a binary operation denoted by '•', then $S \times S \to S, (a, b) \to a \bullet b$ and satisfies the four axioms:

- Closure: $\forall a,\ b \in S$, the result of the operation $a{\cdot}b$ is also in S.
- Associativity: $\forall a, b, c \in S$, then $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- Identity: $\exists e \in S$, called the identity element such that $\forall a \in S, a \cdot e = e \cdot a = a$.
- Inverse: $\forall a \in S$, there exists a^{-1}, such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

Abelian Group:

- Commutativity: If the binary operation is commutative for all a and b in the ring $a \cdot b = b \cdot a$, it is called an Abelian Group.

Ring: If the set S has binary addition and multiplication operations defined:

- Addition Forms an Abelian Group: The set S with the addition operation forms an Abelian group, where the identity element is denoted as 0.
- Associativity: For all elements a, b, and c in S, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- Distributive Laws: For all elements a, b, and c in S, $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$.

Commutative Ring: The multiplication operation is commutative.

- Commutativity: For all elements a and b in S, $a \cdot b = b \cdot a$.

Equivalence Class

Equivalence relations are a concept that helps us establish a way to perceive certain elements within a set as being "equivalent" in a particular context. When dealing with mathematical structures like groups, rings, or fields, identifying elements with common characteristics is often valuable.

Let's consider a non-empty set S and an equivalence relation denoted by $\mathbf{R}$ This relation satisfies three critical properties:

- Reflexivity: $\forall a \in S$, it's related to itself, meaning $a\mathbf{R}a$ holds.
- Symmetry: $\forall a, b \in S$, if a is related to b the reverse is also true, $a\mathbf{R}b$ implies $b\mathbf{R}a$.
- Transitivity: $\forall a, b, c \in S$ are related such that $a\mathbf{R}b$ and $b\mathbf{R}c$ then this implies $a\mathbf{R}c$.

In the context of an equivalence relation $\mathbf{R}$ acting on a set S, the **equivalence class** of an element x, denoted as $[x]\mathbf{R}$ encompasses all elements of the set that are related to x under the relation. In other words, $[x]\mathbf{R}$ consists of all the elements y from set S where the relation $x\mathbf{R}y$ holds.

For instance, let's consider an example involving congruence modulo $n \in \mathbb{Z}$. When two elements x and y are related by this congruence relation, it means that either n divides the difference $x - y$ or y can be obtained by adding a multiple of n to x, indicated as $y = x + kn$ where $k \in \mathbb{Z}$.

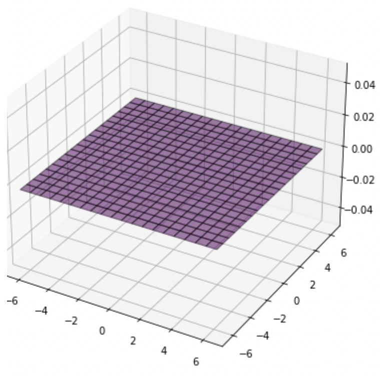
Breaking down the properties of congruence modulo n:

- Reflexivity: Any element a is congruent to itself modulo n as $a - a = 0$ which is a multiple of n. Hence, $a \equiv a(\mod n)$
- Symmetry: If a is congruent to b modulo n, then their difference $a - b$ is a multiple of n. Similarly, $b - a$ is also a multiple of n, which means $b \equiv a(\mod n)$.
- Transitivity: If a is congruent to b modulo n and b is congruent to c modulo n, then their differences $a - b$ and $b - c$ are both multiples of n. Adding these differences together results in $a - c$, which is also a multiple of n. Therefore, $a \equiv c(\mod n)$.

Projective Space

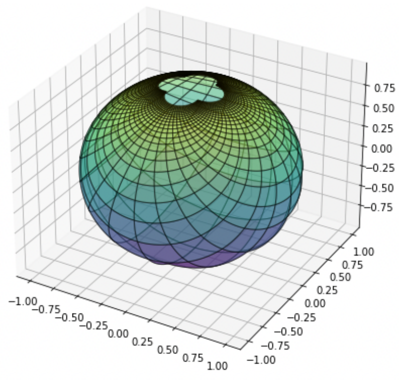
To create an environment conducive to Elliptic Curve Cryptography is possible, we need to define a set of rules that will govern the behavior of numbers. Some of those rules are derived from affine and projective space properties.

“**Affine space** is a set of points with a notion of parallel lines and a vector space structure without a fixed origin point.”[1] $\mathbf{A}^n(\mathbb{R}) = \mathbb{R}^n = \{(x_1, \ldots, x_n) \mid x_i \in \mathbb{R}\}$



$\mathbf{A}^n(\mathbb{R})$ represents n-dimensional Euclidean space over the real numbers. It is the set of all n-tuples $(x_1, x_2, \ldots, x_n)$ where each component x_i belongs to the set of real numbers, denoted as $\mathbb{R}$.[2] We recover the line, plane, and three-dimensional space by setting n = 1, 2, or 3, respectively.

Stereographic projection is a geometric technique that maps points from one dimensional space onto another. Cartesian coordinates[3] define and work with points on elliptic curves in the affine plane. They correspond to the usual (x, y) coordinates in a two-dimensional plane. Projective coordinates extend this representation to include points at infinity.



Projective space is a mathematical construct that generalizes the concept of perspective in geometry. It includes all lines passing through the origin in a vector space. However, unlike Euclidean space, projective space does not distinguish between points on the same line through the origin. This means that parallel lines in Euclidean space intersect at a single point in projective space, known as a "point at infinity.”[4] Projective space can perform transformations that Euclidean space cannot.

Figure: All points meet at the north pole.[5]

$\mathbf{P}^n$ represents an n-dimensional projective space.

$$\mathbf{P}^1(\mathbb{R}) = \{[x, y] \mid x, y \in \mathbb{R}, (x, y) \neq (0, 0)\}$$
$$= \{[r, 1] \mid r \in \mathbb{A}^1(\mathbb{R})\} \cup \{[1, 0]\}$$

Understanding projective space is complex; hence we will illustrate it using a projection from a 2-D circle into a 1-D plane or line.

[1] https://en.wikipedia.org/wiki/Euclidean_space

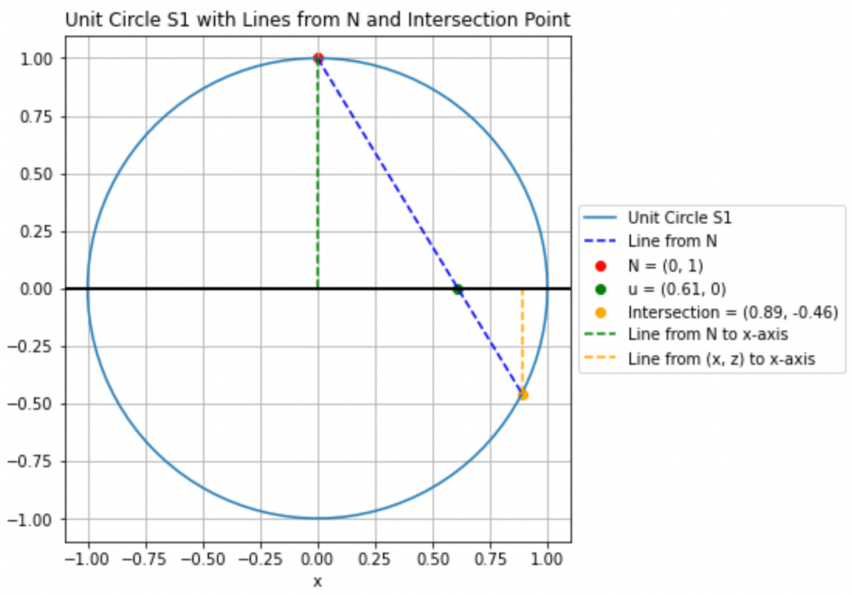
[2] http://www.cds.caltech.edu/~marsden/wiki/uploads/math1c-10/textbooks/matrix_algebra.pdf

[3] https://en.wikipedia.org/wiki/Homogeneous_coordinates

[4] https://en.wikipedia.org/wiki/Stereographic_projection

[5] Algorithm attached

Let the unit circle[6] S^1 defined by $x^2 + z^2 = 1$ in the (x, z)-plane with a point $N = (0,\ 1)$. This circle can also be represented in projective space $\mathbf{A}^n(\mathbb{R})$ as a subset of $\mathbf{P}^1(\mathbb{R})$, where $\mathbf{P}^1(\mathbb{R})$ is the set of points $[x, y]$ with $x, y \in \mathbb{R}$ and $[x, y] \neq [0,0]$. Notably, the point $N = [0,1]$ is the north pole, and its antipode is the south pole.



All non-tangent lines that go through N will intersect the circle at one point $(x,\ z) \neq N$. In the context of projective space, we view this intersection as a point $[r,\ 1]$ on $\mathbf{P}^1(\mathbb{R})$ where $r \in \mathbf{A}^n(\mathbb{R})$. Additionally, the x-axis intersects the circle at a single point u. Thus, we may regard the circle S^1 in projective space as an extended line with a point "at infinity."

Figure: Unit circle[7]

The mapping $\pi_N : S^1\{N\} \rightarrow \mathbf{P}^1(\mathbb{R})$ defined by $\pi_N(x,\ y) = [r,\ 1]$ is termed stereographic projection away from the north pole N. In the context of projective space, this mapping captures the projection of points on S^1 to points on the projective line $\mathbf{P}^1(\mathbb{R})$. The inverse map $\pi^{-1} : \mathbf{P}^1(\mathbb{R}) \rightarrow S^1\{N\}$ is the rational parameterization of the circle in projective space.

For all points $(x,\ y)$ with $y < 1$ on the circle S^1 in the $(x,\ y)$-plane and for all real numbers u, the following relationships hold within the context of projective space:

$$[r,1] = \pi_N(x,\ y) = [2u,\ 1] \text{ where } r \in \mathbf{A}^n(\mathbb{R})$$

$$[x,\ y] = \pi^{-1}([r,\ 1]) = \frac{[2u,\ u^2 - 1]}{[u^2 + 1]}$$

A projective space comprises sets of equivalence classes. Each non-zero vector within $\mathbb{R}^{n+1}$ uniquely defines a line originating from the origin. Two such vectors x and y, indicate the same line if their relationship is denoted as $x\mathbf{R}y$. This equivalence can be

[6] College of the Holy Cross, Spring 2013 Math 392 Stereographic Projection

[7] Algorithm attached

represented as $[x, y, z]$, where $[x, y, z]$ is an equivalence class of the tuple (x, y, z). For a given field $\mathbb{F}$:

$$\begin{aligned}\mathbf{P}^2(\mathbb{R}) &= \{[x, y, z] \mid x, y, z \in \mathbb{R}, (x, y, z) \neq (0, 0, 0)\} \\ &= \{[x, y, 1] \mid (x, y) \in \mathbf{A}^2(\mathbb{R})\} \cup \{(x, y, 0) \in \mathbf{P}^2(\mathbb{R})\}\end{aligned}$$

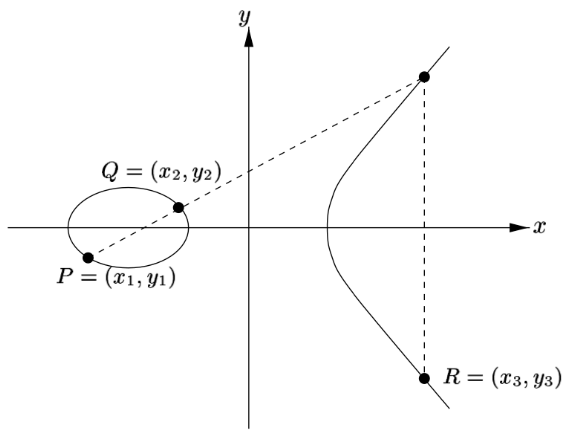
Transitioning from projective to affine spaces, an affine curve embodies solutions of type $f(x, y) = 0$. Conversely, a projective curve arises from a homogeneous equation $f(x, y, z) = 0$, which can be confined to the affine curve using the equation $f(x, y, 1) = 0$. Notably, the projective curve $x^2 + y^2 - z^2$ corresponds to the unit circle in the affine curve, given by $x^2 + y^2 - 1 = 0$. Although the projective curve encompasses an affine curve, the latter includes points at infinity that the former does not account for.

Elliptic Curves

Elliptic curves are an algebraic equation of the form $y^2 = x^3 + ax + b$, where a, b are constants, and x, y are variables representing points on the curve.

A point in $\mathbf{P}^2(\mathbb{R})$ is given by the homogeneous coordinates* $[x, y, z] \in \mathbb{R}$ and $(x, y, z) \neq (0,0,0)$ ensures we exclude the point at infinity from this description, as it does not have well-defined Cartesian coordinates. In the second representation, a point in $\mathbf{P}^2(\mathbb{R})$ is given by either $[x, y, 1]$ where (x, y) is a point in the affine plane $\mathbf{A}^2(\mathbb{R})$, or $[x, y, 0]$, where (x, y) is a point in the projective line $\mathbf{P}^1(\mathbb{R})$.

$$y^2z = x^3 + axz^2 + bz^3$$



To add two points on the curve, let's say P and Q, we take these points and construct a line equation that passes through both P and Q. This line will intersect the elliptic curve at another point, which we'll call T. By taking the x-coordinate of point T, we can create a vertical line that crosses the elliptic curve once more, resulting in another point, R. It's important to note that due to the symmetry of the elliptic curve, we can negate the y-coordinate to get the mirrored point.

Figure: Point addition on the Elliptic Curve[8]

[8] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)." Certicom Research, Canada. Dept. of Combinatorics & Optimization, University of Waterloo, Canada. Certicom Corporation 2001 cps wp 001-1.

Since we're dealing with a projective curve, the vertical line we draw passes through the point at infinity. With this process, we find that the sum of points P and Q is equal to point R.

Properties of addition on an elliptic curve S with a point at infinity $e = [0, 1, 0]$:

- Associativity: $\forall a, b, c \in S, (a + b) + c = a + (b + c)$.
- Commutative: $\forall a, b \in S, a + b = b + a$
- Identity: $\forall a \in S, a + e = e + a = a$
- Inverse: $\forall a \in S, a + a^{-1} = e$

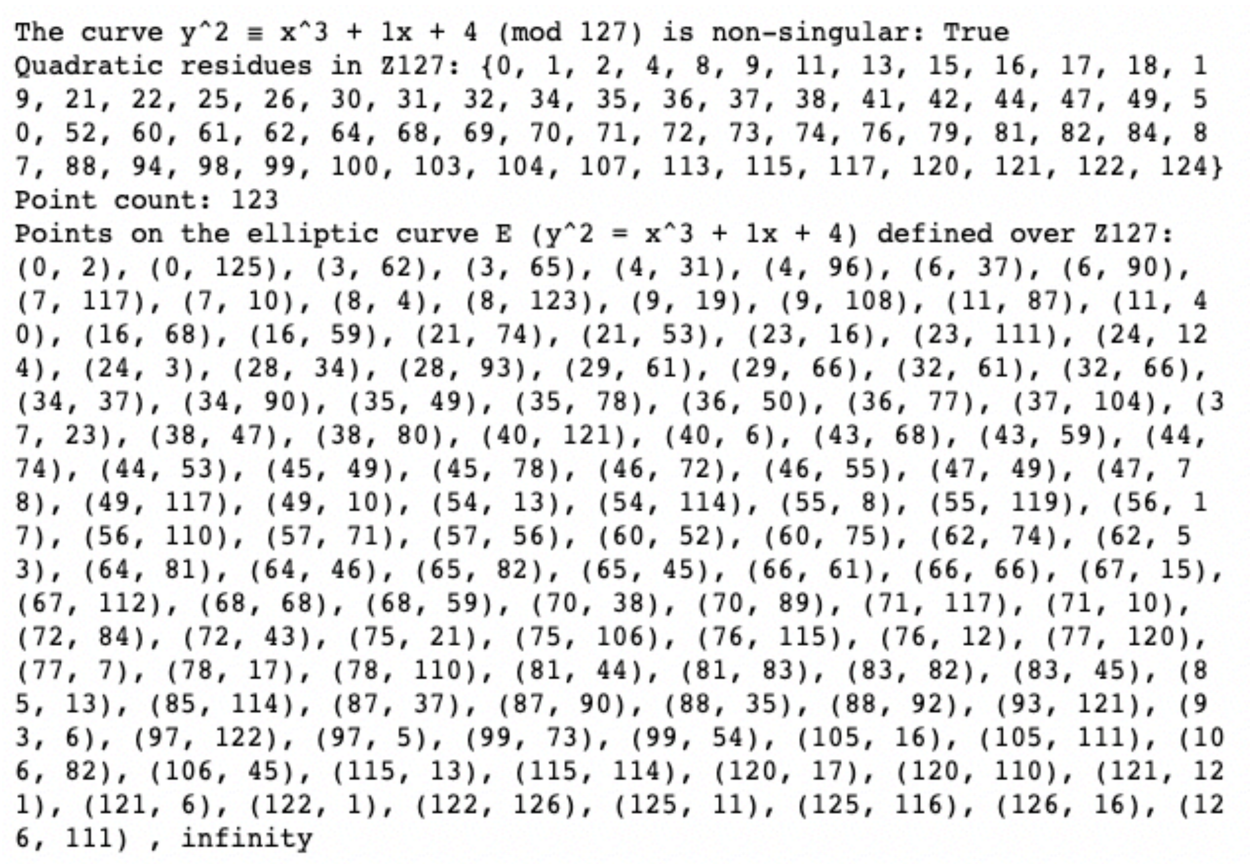
```
import math

# Given points P and Q
x_P = 3
x_Q = 5
a = 1
b = 4

def solve_y(x, a, b):
    return math.sqrt(abs(x**3 - a*x + b))

def slope(x1, y1, x2, y2):
    return (y2 - y1) / (x2 - x1)

def third_point(m, x1, x2):
    return m**2 - x1 - x2

y_P = solve_y(x_P, a, b)
y_Q = solve_y(x_Q, a, b)

print("y_P:", y_P)
print("y_Q:", y_Q)

m = slope(x_P, y_P, x_Q, y_Q)
print("Slope (m):", m)

x_R = third_point(m, x_P, x_Q)
print("x_R:", x_R)

y_R = solve_y(x_R, a, b)
print("y_R:", y_R)
```

```
y_P: 5.291502622129181
y_Q: 11.135528725660043
Slope (m): 2.922013051765431
x_R: 0.5381602746875256
y_R: 1.9020251865884827
```

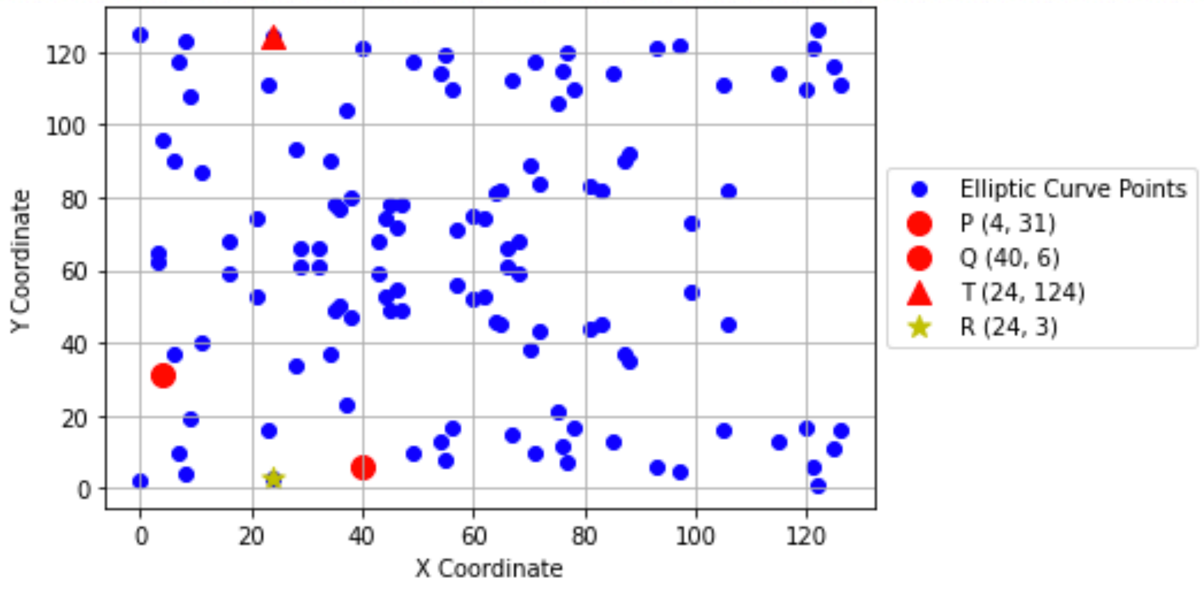
Since projective space comprises sets of equivalence classes, we can generalize the above over a finite field $\mathbb{F}$.

$$\mathbf{P}^n(\mathbb{R}) = [x_1, \ldots, x_n, 1] \,|\, (x_1, \ldots, x_n) \in \mathbf{A}^n(\mathbb{F}) \cup [x_1, \ldots, x_n, 0] \in \mathbf{P}^n(\mathbb{F})$$

Given a prime p, we choose $a, b \in F_p$ so that E $y^2 = x^3 + ax + b$ is an elliptic curve.



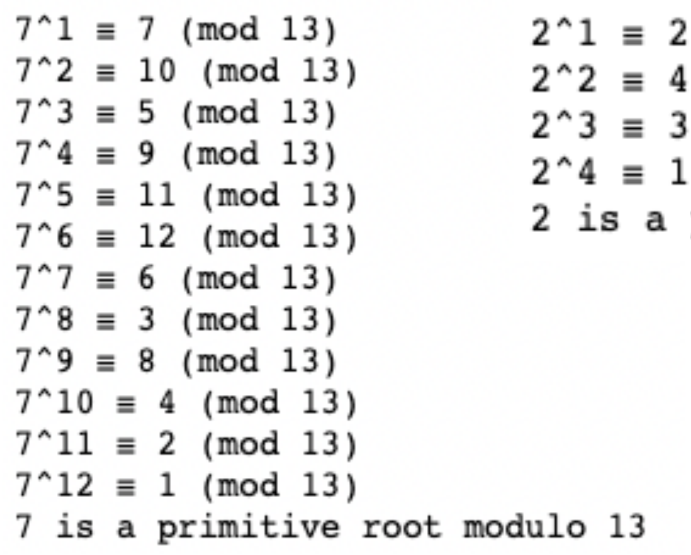
```
The curve y^2 ≡ x^3 + 1x + 4 (mod 127) is non-singular: True
Quadratic residues in Z127: {0, 1, 2, 4, 8, 9, 11, 13, 15, 16, 17, 18, 1
9, 21, 22, 25, 26, 30, 31, 32, 34, 35, 36, 37, 38, 41, 42, 44, 47, 49, 5
0, 52, 60, 61, 62, 64, 68, 69, 70, 71, 72, 73, 74, 76, 79, 81, 82, 84, 8
7, 88, 94, 98, 99, 100, 103, 104, 107, 113, 115, 117, 120, 121, 122, 124}
Point count: 123
Points on the elliptic curve E (y^2 = x^3 + 1x + 4) defined over Z127:
(0, 2), (0, 125), (3, 62), (3, 65), (4, 31), (4, 96), (6, 37), (6, 90),
(7, 117), (7, 10), (8, 4), (8, 123), (9, 19), (9, 108), (11, 87), (11, 4
0), (16, 68), (16, 59), (21, 74), (21, 53), (23, 16), (23, 111), (24, 12
4), (24, 3), (28, 34), (28, 93), (29, 61), (29, 66), (32, 61), (32, 66),
(34, 37), (34, 90), (35, 49), (35, 78), (36, 50), (36, 77), (37, 104), (3
7, 23), (38, 47), (38, 80), (40, 121), (40, 6), (43, 68), (43, 59), (44,
74), (44, 53), (45, 49), (45, 78), (46, 72), (46, 55), (47, 49), (47, 7
8), (49, 117), (49, 10), (54, 13), (54, 114), (55, 8), (55, 119), (56, 1
7), (56, 110), (57, 71), (57, 56), (60, 52), (60, 75), (62, 74), (62, 5
3), (64, 81), (64, 46), (65, 82), (65, 45), (66, 61), (66, 66), (67, 15),
(67, 112), (68, 68), (68, 59), (70, 38), (70, 89), (71, 117), (71, 10),
(72, 84), (72, 43), (75, 21), (75, 106), (76, 115), (76, 12), (77, 120),
(77, 7), (78, 17), (78, 110), (81, 44), (81, 83), (83, 82), (83, 45), (8
5, 13), (85, 114), (87, 37), (87, 90), (88, 35), (88, 92), (93, 121), (9
3, 6), (97, 122), (97, 5), (99, 73), (99, 54), (105, 16), (105, 111), (10
6, 82), (106, 45), (115, 13), (115, 114), (120, 17), (120, 110), (121, 12
1), (121, 6), (122, 1), (122, 126), (125, 11), (125, 116), (126, 16), (12
6, 111) , infinity
```



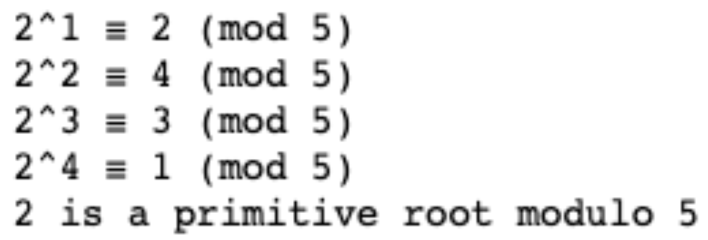
ElGamal
Properties

ElGamal is based on the mathematical properties of modular exponentiation, while ECC is based on the mathematical properties of elliptic curves. However, they share a similar encryption strategy.

To understand how an ElGammal key is constructed, we first need to know how to find a primitive root. "A primitive root mod n is an element $g \in Z_n^*$ mod n, whose powers generate all of Z_n^*. That is, every element $b \in Z_n^*$ can be written as g^x mod n, for some integer x."[9]



```
7^1 ≡ 7 (mod 13)
7^2 ≡ 10 (mod 13)
7^3 ≡ 5 (mod 13)
7^4 ≡ 9 (mod 13)
7^5 ≡ 11 (mod 13)
7^6 ≡ 12 (mod 13)
7^7 ≡ 6 (mod 13)
7^8 ≡ 3 (mod 13)
7^9 ≡ 8 (mod 13)
7^10 ≡ 4 (mod 13)
7^11 ≡ 2 (mod 13)
7^12 ≡ 1 (mod 13)
7 is a primitive root modulo 13
```



```
2^1 ≡ 2 (mod 5)
2^2 ≡ 4 (mod 5)
2^3 ≡ 3 (mod 5)
2^4 ≡ 1 (mod 5)
2 is a primitive root modulo 5
```

Note that when reduced, the root powers comprise all possible remainder modulo g, except 0.

To construct an ElGammal[10] encryption system, we need the following:

- a very large prime number p
- identify α, a primitive root mod p

First entity:

- select a random private key a
- calculate $\beta = \alpha^a \mod p$

The first public key is (p, α, β)

Second entity:

- select a random private key a'
- calculate $\beta' = \alpha^{a'} \mod p$

The second public key is (p, α, β')

[9] https://brilliant.org/wiki/primitive-roots/#:~:text=Let%20n%20be%20a%20positive,pmod%7Bn%7D%5Cbig).

[10] https://math.asu.edu/sites/default/files/elgamal.pdf

We will assume that the data has been encoded and our message is a large number x.

We can encrypt using $E(x) = x\beta^{a'} \mod p$. Given a ciphertext, $y = (\beta', t)$ we can decrypt it using $E^{-1}(y) = t\beta'^{-a} \mod p$.[11]

Example

In our example, we will encrypt and decrypt the word 'HELLO' by simulating Bob's sending a message to Alice using an ElGammal cryptography scheme.

We can use a simple encoding function that takes a plaintext message as input. It uses a mapping that assigns each uppercase letter (A-Z) to a numerical value (0-25), following the pattern A=0, B=1, ..., Z=25.

Our plaintext is equivalent to the array: [7, 4, 11, 11, 14]. This will be our message x.

Now we need to find a large prime number $p = 101$.

```
7^97 ≡ 48 (mod 101)
7^98 ≡ 33 (mod 101)
7^99 ≡ 29 (mod 101)
7^100 ≡ 1 (mod 101)
7 is a primitive root modulo 101
```

Therefore $\alpha = 7$

The next step is for Bob to encrypt the message by calculating $E(x) = x\beta^{a'} \mod p$ for all x. Alice receives Bob's message and deciphers it using her private key and Bob's public key $E^{-1}(y) = t\beta'^{-a} \mod p$. Once he has the values, Alice can decode them with the predefined convention and retrieve the message 'HELLO'.

```
Alice's header (beta): 93
Bob's header (beta): 75
Encrypted messages: [(75, 76), (75, 29), (75, 4), (75, 4), (75, 51)]
Decrypted messages: [7, 4, 11, 11, 14]
Decoded characters: ['H', 'E', 'L', 'L', 'O']
```

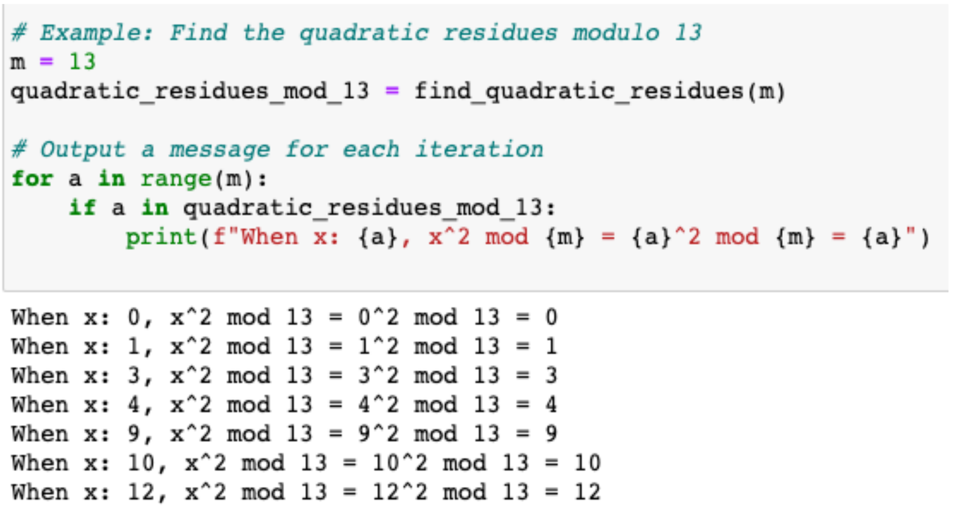
The main difference between ElGamal and ECC is that the first uses a primitive root. In contrast, the latter uses the elliptic curve to establish the mathematical foundation for its cryptographic operations.

[11] https://homepages.math.uic.edu/~leon/mcs425-s08/handouts/el-gamal.pdf

Elliptic Curve Cryptography

Properties

Quadratic residues are used to define operations on elliptic curves that make solving the discrete logarithm problem computationally difficult. This property makes it hard to derive the private key from the public key, even for powerful adversaries.



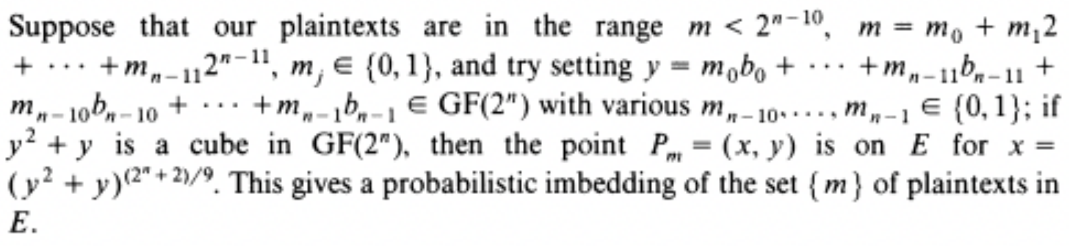
```python
# Example: Find the quadratic residues modulo 13
m = 13
quadratic_residues_mod_13 = find_quadratic_residues(m)

# Output a message for each iteration
for a in range(m):
    if a in quadratic_residues_mod_13:
        print(f"When x: {a}, x^2 mod {m} = {a}^2 mod {m} = {a}")
```

```
When x: 0, x^2 mod 13 = 0^2 mod 13 = 0
When x: 1, x^2 mod 13 = 1^2 mod 13 = 1
When x: 3, x^2 mod 13 = 3^2 mod 13 = 3
When x: 4, x^2 mod 13 = 4^2 mod 13 = 4
When x: 9, x^2 mod 13 = 9^2 mod 13 = 9
When x: 10, x^2 mod 13 = 10^2 mod 13 = 10
When x: 12, x^2 mod 13 = 12^2 mod 13 = 12
```

Encoding

Let's say our message is a series of characters since our encryption algorithm takes a point as input. Then, we need to have a defined strategy for preparing that message for encryption and rebuilding it after decryption. The encoding scheme may vary. One commonly used approach is to generate a point $(x, \sqrt{x^3 + ax + b})$

"To encode one tests in turn putative x co-ordinates x=1000m,1000m+1,…,1000m+999 =1000,1000+1, …,1000+999 until we find one where 3 x +ax+b. 3 ++ is a square modulo p and then compute the corresponding y."[12]



Suppose that our plaintexts are in the range $m < 2^{n-10}$, $m = m_0 + m_1 2 + \cdots + m_{n-11} 2^{n-11}$, $m_j \in \{0, 1\}$, and try setting $y = m_0 b_0 + \cdots + m_{n-11} b_{n-11} + m_{n-10} b_{n-10} + \cdots + m_{n-1} b_{n-1} \in \mathrm{GF}(2^n)$ with various $m_{n-10}, \ldots, m_{n-1} \in \{0, 1\}$; if $y^2 + y$ is a cube in $\mathrm{GF}(2^n)$, then the point $P_m = (x, y)$ is on E for $x = (y^2 + y)^{(2^n + 2)/9}$. This gives a probabilistic imbedding of the set $\{m\}$ of plaintexts in E.

Figure: excerpt.[13]

[12] https://crypto.stackexchange.com/questions/103132/encoding-a-message-as-points-on-elliptic-curve

[13] https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/S0025-5718-1987-0866109-5.pdf

Encryption

To generate a public key, we need the following:

- p some prime number
- G basepoint on curve
- c some random number

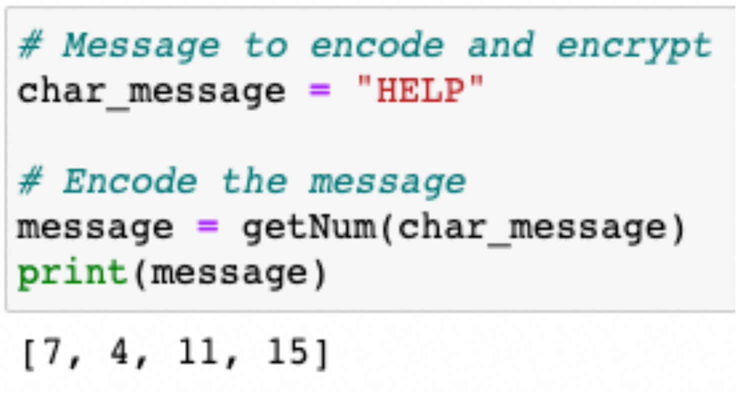
Given plaintext m encoded as the point p_m and a random integer k , the encrypted message is the point $(kG, p_m + k(cG))$.

Decryption

We use the public key variable c to find $p_m = [p_m + k(cG)] + -(ckG)$

Example

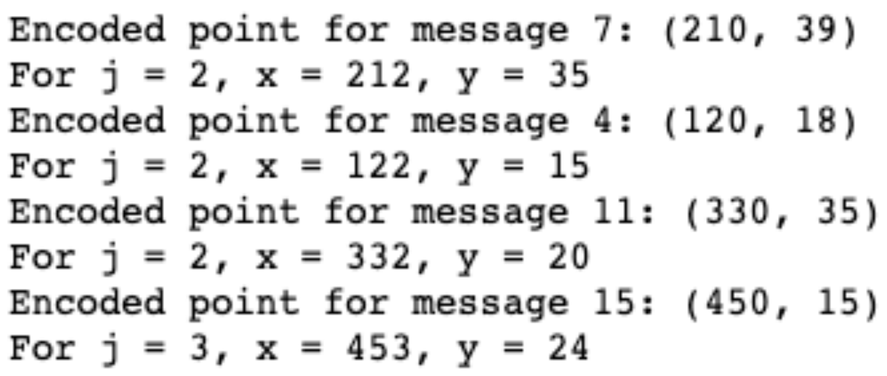
Encoding



```
# Message to encode and encrypt
char_message = "HELP"

# Encode the message
message = getNum(char_message)
print(message)
```

```
[7, 4, 11, 15]
```

k = 30



```
Encoded point for message 7: (210, 39)
For j = 2, x = 212, y = 35
Encoded point for message 4: (120, 18)
For j = 2, x = 122, y = 15
Encoded point for message 11: (330, 35)
For j = 2, x = 332, y = 20
Encoded point for message 15: (450, 15)
For j = 3, x = 453, y = 24
```

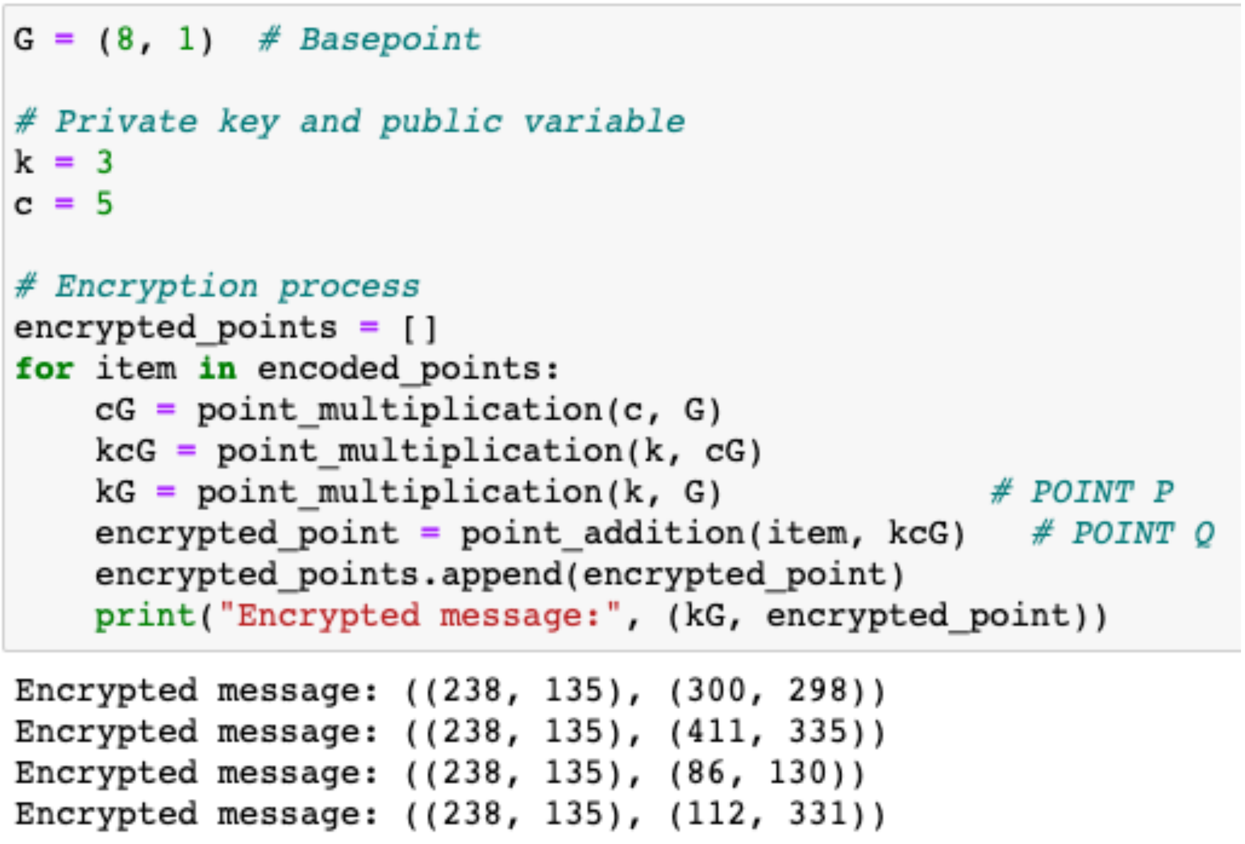
Encryption

Let E be the elliptic curve given by $y^2 = x^3 - 1x + 4$ over p = 457.

```
# Given elliptic curve parameters
a = -1
b = 4
p = 457
```

```
elliptic_curve_points = points_on_elliptic_curve(p, b)
print("Point count:", len(elliptic_curve_points) + 1)

print(f"Points on the elliptic curve E (y^2 = x^3 + {a}x + {b}) defined ove
print(", ".join(map(str, elliptic_curve_points)), ', infinity')
```

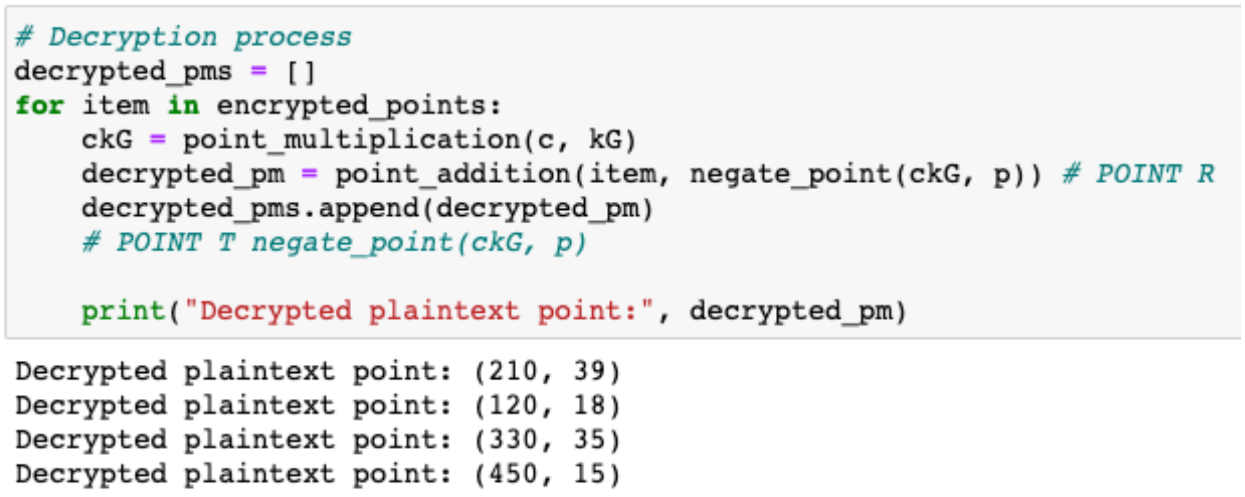
```
Point count: 477
Points on the elliptic curve E (y^2 = x^3 + -1x + 4) defined over Z457:
(0, 1), (0, 456), (1, 1), (1, 456), (2, 456), (2, 1), (3, 456), (3, 1),
(4, 456), (4, 1), (5, 1), (5, 456), (8, 456), (8, 1), (10, 456), (10, 1),
(11, 456), (11, 1), (12, 456), (12, 1), (15, 1), (15, 456), (16, 456), (1
6, 1), (18, 456), (18, 1), (19, 456), (19, 1), (21, 456), (21, 1), (22,
1), (22, 456), (25, 1), (25, 456), (27, 1), (27, 456), (28, 456), (28,
1), (29, 1), (29, 456), (31, 456), (31, 1), (32, 456), (32, 1), (39, 45
6), (39, 1), (40, 1), (40, 456), (42, 456), (42, 1), (43, 1), (43, 456),
(44, 1), (44, 456), (48, 1), (48, 456), (49, 456), (49, 1), (53, 456), (5
```



```
G = (8, 1)  # Basepoint

# Private key and public variable
k = 3
c = 5

# Encryption process
encrypted_points = []
for item in encoded_points:
    cG = point_multiplication(c, G)
    kcG = point_multiplication(k, cG)
    kG = point_multiplication(k, G)                    # POINT P
    encrypted_point = point_addition(item, kcG)    # POINT Q
    encrypted_points.append(encrypted_point)
    print("Encrypted message:", (kG, encrypted_point))
```

```
Encrypted message: ((238, 135), (300, 298))
Encrypted message: ((238, 135), (411, 335))
Encrypted message: ((238, 135), (86, 130))
Encrypted message: ((238, 135), (112, 331))
```

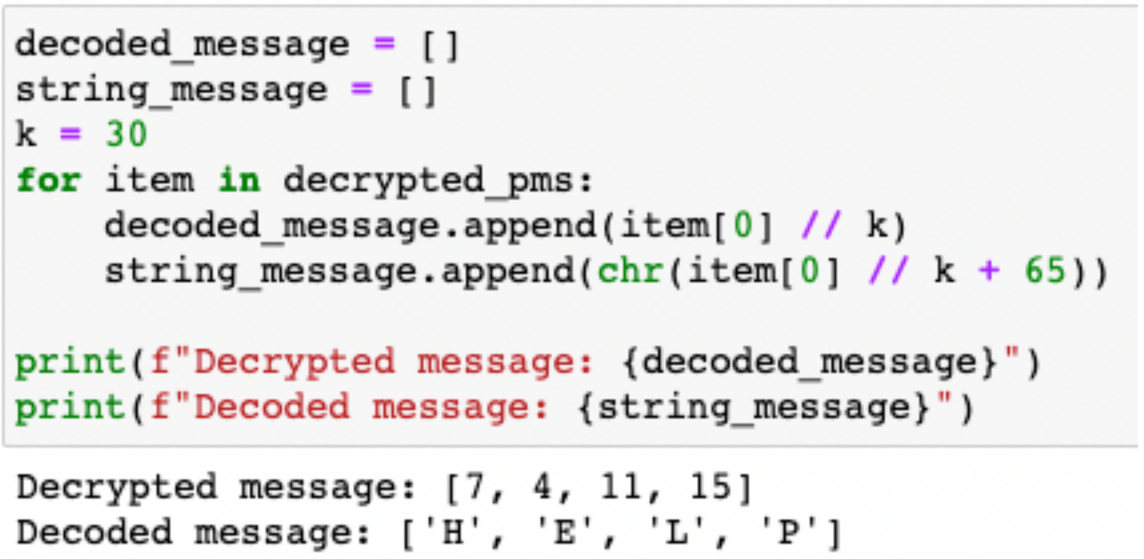
Decryption



```
# Decryption process
decrypted_pms = []
for item in encrypted_points:
    ckG = point_multiplication(c, kG)
    decrypted_pm = point_addition(item, negate_point(ckG, p)) # POINT R
    decrypted_pms.append(decrypted_pm)
    # POINT T negate_point(ckG, p)

    print("Decrypted plaintext point:", decrypted_pm)
```

```
Decrypted plaintext point: (210, 39)
Decrypted plaintext point: (120, 18)
Decrypted plaintext point: (330, 35)
Decrypted plaintext point: (450, 15)
```

Decoding



```
decoded_message = []
string_message = []
k = 30
for item in decrypted_pms:
    decoded_message.append(item[0] // k)
    string_message.append(chr(item[0] // k + 65))

print(f"Decrypted message: {decoded_message}")
print(f"Decoded message: {string_message}")
```

```
Decrypted message: [7, 4, 11, 15]
Decoded message: ['H', 'E', 'L', 'P']
```

Analysis

Projective coordinates (X, Y, Z) extend the representation of points on an elliptic curve to include points at infinity. Points at infinity are represented using homogeneous coordinates where Z = 0. These points are necessary for completing the group structure on an elliptic curve, allowing the definition of point addition operations. They eliminate the need for special cases and divisions by zero that can arise in affine coordinates, making the arithmetic operations more elegant and efficient.

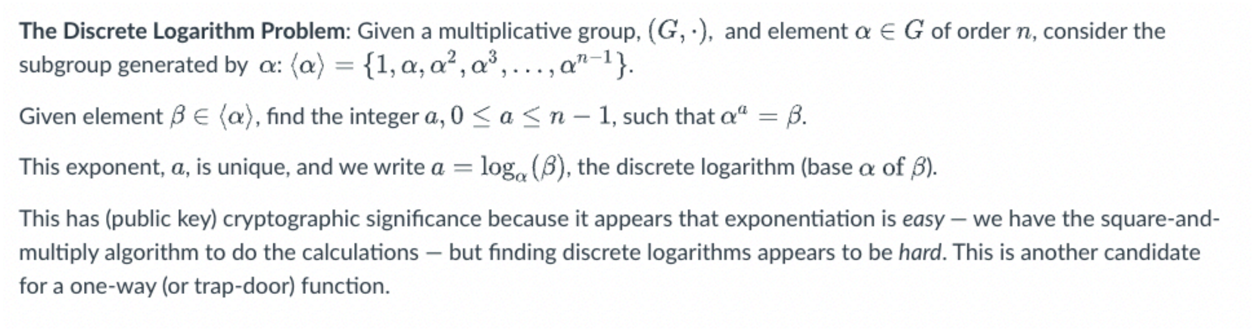
The operation of point addition defines the abelian group structure of the points on an elliptic curve. Given two points, P and Q, on the curve, their sum P + Q is another point on the curve. The addition operation satisfies the properties of closure, associativity, commutativity, identity element (point at infinity), and inverse element for every point. These properties are the defining characteristics of an abelian group.

The projective nature of elliptic curves is leveraged to define an "addition" operation on points, which makes the set of points on the elliptic curve (including the point at infinity) form an abelian group.

The points on an elliptic curve in P^2 form an abelian group under an operation called "point addition." This operation considers the points at infinity and allows for secure cryptographic operations, such as public-key cryptography, using the Elliptic Curve Cryptography (ECC) algorithm.

Without the finite field and the modulo operations, the points on the elliptic curve would not form a well-defined abelian group, and the security guarantees of ECC would not hold.

ECC security stems from the discrete logarithm problem: the challenge of finding the exponent ("scalar") "k" when given a point "P" and its scalar multiple "Q = k * P." This problem is believed to be computationally infeasible for large enough curves and is the foundation of ECC's security.



The Discrete Logarithm Problem: Given a multiplicative group, $(G, \cdot)$, and element $\alpha \in G$ of order n, consider the subgroup generated by α: $\langle \alpha \rangle = \{1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{n-1}\}$.

Given element $\beta \in \langle \alpha \rangle$, find the integer $a, 0 \leq a \leq n-1$, such that $\alpha^a = \beta$.

This exponent, a, is unique, and we write $a = \log_\alpha(\beta)$, the discrete logarithm (base α of β).

This has (public key) cryptographic significance because it appears that exponentiation is *easy* — we have the square-and-multiply algorithm to do the calculations — but finding discrete logarithms appears to be *hard*. This is another candidate for a one-way (or trap-door) function.

Analogs of the older discrete logarithm (DL) cryptosystems in which the subgroup is replaced by the group of points on an elliptic curve over a finite field.

ECDSA

Properties

ECDSA is primarily used for generating and verifying digital signatures. A digital signature is a cryptographic technique that provides authenticity, integrity, and non-repudiation for digital messages.

In ECDSA, a private key holder can sign a message, and anyone with the corresponding public key can verify the signature. This ensures that the message was signed by the holder of the private key and that the message hasn't been altered since the signature was generated.

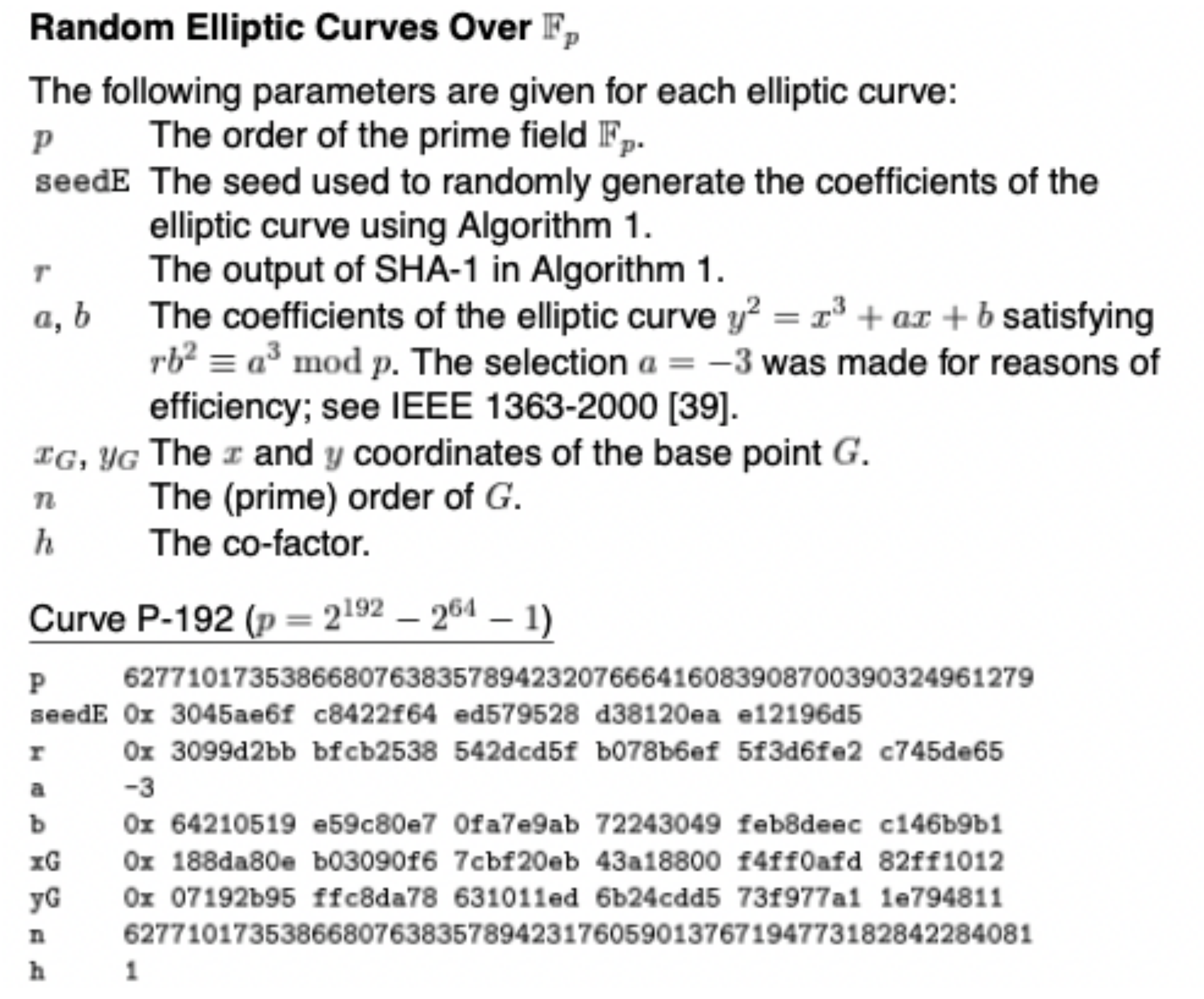
Signature generation and verification.

Randomness and Nonce:

Generating a secure signature requires using a random value called a "nonce" ('k' in ECDSA). This nonce ensures that the same message doesn't produce the same signature each time it's signed. However, improper nonce handling can lead to vulnerabilities like repeated nonce attacks. Therefore, proper random number generation is crucial to maintaining the security of ECDSA.

To construct an ECDSA encryption system, we need the domain parameters $D = (q, FR, a, b, G, n, h)$ and a key pair (d, Q)

p	order of the prime field $\mathbb{F}_p$
seedE	used to generate coefficients of the curve
r	the output of SHA-1
a, b	coefficients of the curve



Random Elliptic Curves Over $\mathbb{F}_p$

The following parameters are given for each elliptic curve:

p	The order of the prime field $\mathbb{F}_p$.
seedE	The seed used to randomly generate the coefficients of the elliptic curve using Algorithm 1.
r	The output of SHA-1 in Algorithm 1.
a, b	The coefficients of the elliptic curve $y^2 = x^3 + ax + b$ satisfying $rb^2 \equiv a^3 \bmod p$. The selection $a = -3$ was made for reasons of efficiency; see IEEE 1363-2000 [39].
x_G, y_G	The x and y coordinates of the base point G.
n	The (prime) order of G.
h	The co-factor.

Curve P-192 ($p = 2^{192} - 2^{64} - 1$)

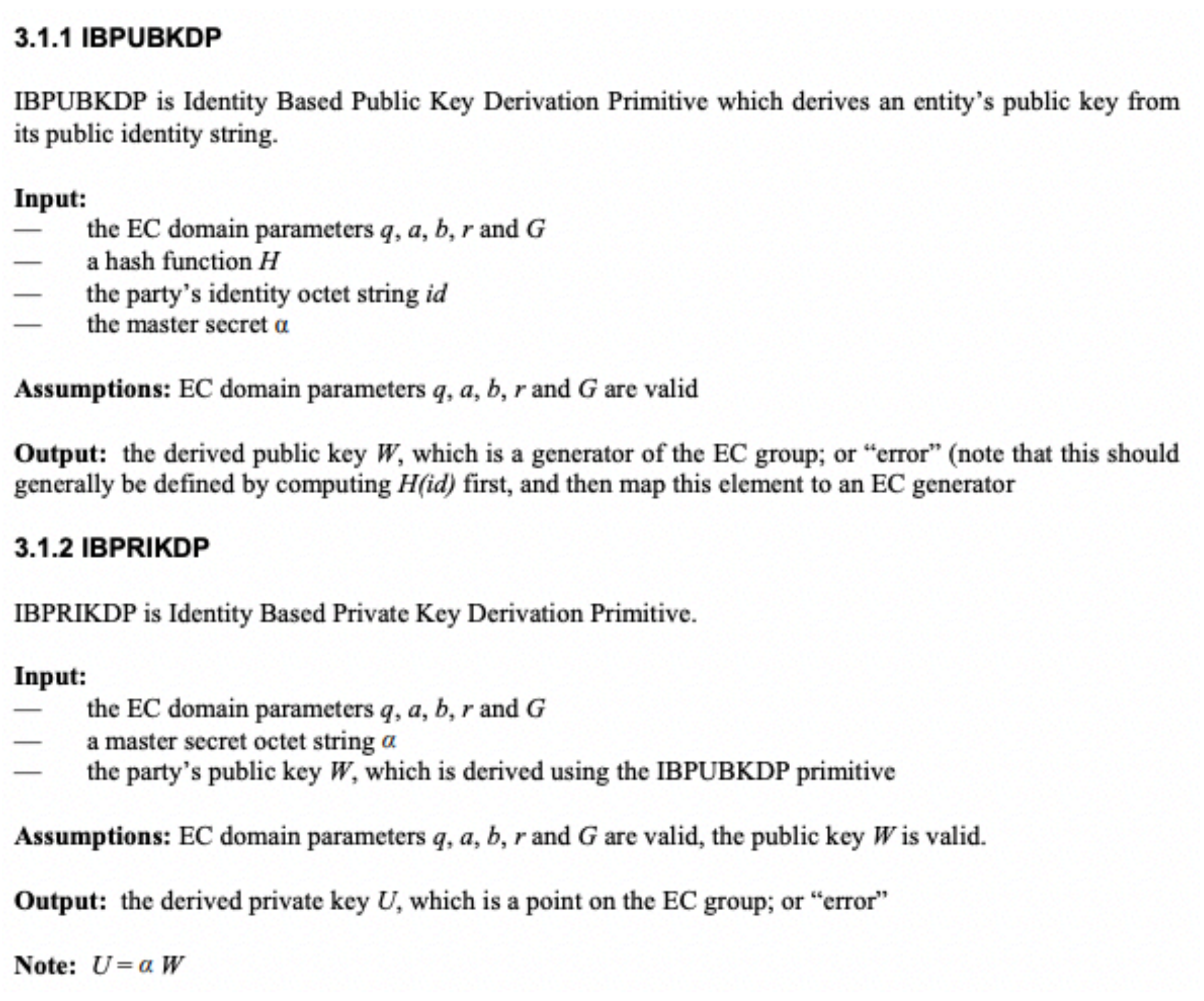
```
p     6277101735386680763835789423207666416083908700390324961279
seedE 0x 3045ae6f c8422f64 ed579528 d38120ea e12196d5
r     0x 3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65
a     -3
b     0x 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1
xG    0x 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
yG    0x 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811
n     6277101735386680763835789423176059013767194773182842284081
h     1
```

First entity:

- select a random integer k, $1 \leq k \leq n-1$
- compute $kG = (x_1, y_1)$ and convert x_1

Second entity:

https://webpages.charlotte.edu/yonwang/papers/P1363IBKAS.pdf



3.1.1 IBPUBKDP

IBPUBKDP is Identity Based Public Key Derivation Primitive which derives an entity's public key from its public identity string.

Input:

— the EC domain parameters q, a, b, r and G
— a hash function H
— the party's identity octet string *id*
— the master secret α

Assumptions: EC domain parameters q, a, b, r and G are valid

Output: the derived public key W, which is a generator of the EC group; or "error" (note that this should generally be defined by computing *H(id)* first, and then map this element to an EC generator

3.1.2 IBPRIKDP

IBPRIKDP is Identity Based Private Key Derivation Primitive.

Input:

— the EC domain parameters q, a, b, r and G
— a master secret octet string α
— the party's public key W, which is derived using the IBPUBKDP primitive

Assumptions: EC domain parameters q, a, b, r and G are valid, the public key W is valid.

Output: the derived private key U, which is a point on the EC group; or "error"

Note: $U = \alpha W$

Example

Figure: Curve P-192[14]

[14] Don Johnson, Alfred Menezes, and Scott Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)." Certicom Research, Canada. Dept. of Combinatorics & Optimization, University of Waterloo, Canada. Certicom Corporation 2001 cps wp 001-1.

Analysis

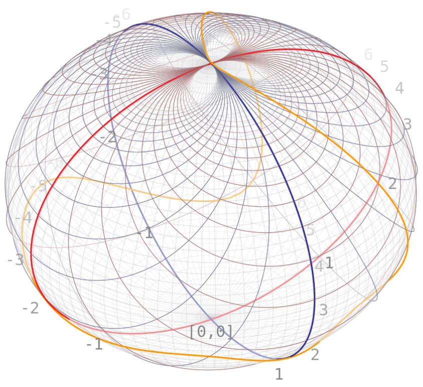
How to attack it? How it’s secure? How do all the pieces come together? Quantum resistance? Improvements?

Conclusion

The research paper provides a comprehensive exploration of the principles, implementations, and security aspects of Elliptic Curve Cryptography (ECC) and its applications. It delves into the foundational mathematical concepts, including Abelian Groups, Equivalence Classes, and Projective Space, which are essential for understanding the underlying algebraic structures. The paper then proceeds to elaborate on Elliptic Curves themselves, elucidating their properties, addition operations, and the connection between affine and projective representations.

The paper effectively parallels ECC and ElGamal encryption, highlighting their shared encryption strategies rooted in mathematical properties. It elaborates on ElGamal's primitive root concept and ECC's use of quadratic residues, offering insights into the cryptographic mechanisms of both approaches. Furthermore, the research presents an illustrative example of encryption and decryption using ElGamal, showcasing how messages are transformed, transmitted, and restored through cryptographic processes.

The heart of the paper lies in its exploration of ECC, where quadratic residues play a pivotal role in establishing the computational difficulty of solving the discrete logarithm problem. The report elucidates ECC's practical implementation and inherent security strengths by explaining the encoding, encryption, and decryption steps. The analysis section delves into projective coordinates and the abelian group structure of points on an elliptic curve, offering a deeper understanding of ECC's security foundations.



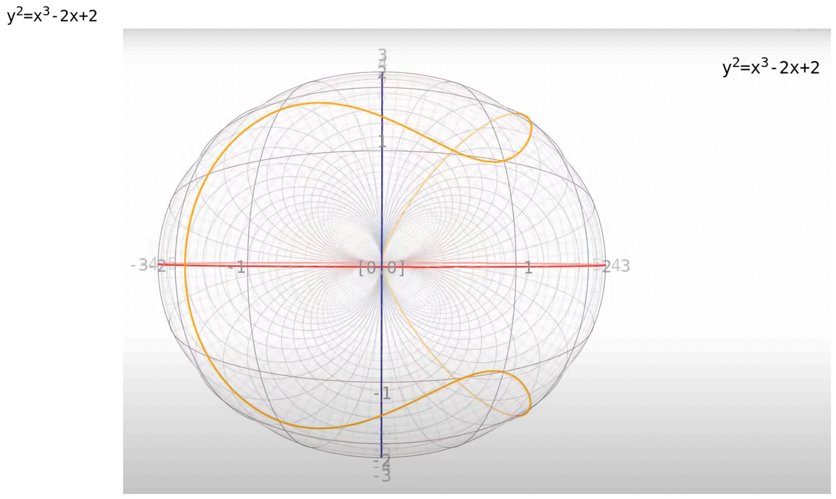


Figure 2: Elliptic Curve 3D[15]

Figure: elliptic from all sides[16]

[15] https://trustica.cz/en/2018/04/05/elliptic-curves-point-at-infinity-revisited/

[16] https://youtu.be/Hk0Fr-k7wmQ