Elliptic Curves as a crypto system

Arnaud LECOMTE

January 28, 2025

1 Introduction

The purpose of this document is to construct a cryptosystem using elliptic curves. To do so, we need to find a trap-door function, i.e. a function that is easy to compute in one direction but extremely difficult to invert unless we have specific secret information.
In this context, our goal is to build a cyclic group using elliptic curves. The security of the cryptosystem will be based on the hardness of the discrete logarithm problem (DLP), which serves as the trap-door function.

2 Finite Set

As implied by its definition, the group G constructed from elliptic curves must operate on discrete values. To ensure this, we define a finite set, which also guarantees that G will be cyclic.
Specifically, we define our set as $\mathbb{F}_n = (\mathbb{Z}, +, \cdot)$, where $n \in \mathbb{N}$ is a prime number. Let us define our set as $\mathbb{F}_\ltimes = (\mathbb{Z}, +)$ with $n \in \mathbb{N}$ a prime number.

Let's build the operator $+$:

$$\forall x, y \in \mathbb{Z}, x +_f y = (x + y) \mod n$$

Let's now define other operators:
Multiplication:

$$\forall x, y \in \mathbb{Z}, x \cdot_f y = \underbrace{(x + x + ... + x + x)}_{\text{y}} \mod n$$

Exponantiation:

$$\forall x \in \mathbb{Z}, k \in \mathbb{N}, x^k = \underbrace{(x * x * ... * x * x)}_{\text{k}} \mod n$$

Division:

$$\forall x, y \in \mathbb{Z}\backslash\{0\} = x/_f y = x \cdot_f y^{-1} \tag{1}$$

As n is a prime number and y is not divisible by n as p ¡ n by definition, from Fermat Little Theorem have that: $y^{n-1} = 1$. So 1 becomes:

$$x \cdot_f y^{-1} = x \cdot_f y^{-1} * 1 = x \cdot_f y^{-1} * y^{n-1} = x \cdot_f y^{n-2}$$

We can easily verify that both operators $(+, \cdot)$ have the following properties to ensure $\mathbb{F}_n$ is a finite set:

- $+$ and $\cdot$ are internal operators, that is, $\mathbb{Z}$ is stable.
- $\exists e \in \mathbb{Z}$ such as $\forall a \in \mathbb{Z}, a +_f e = a$. e is the neutral element of the set.
- $\mathbb{1}$ exists and we have multiplicative identity.
- $\forall x \in \mathbb{Z}, \exists x'$ such as $xx' = x'x$.
- If a is int the set and not 0, a^{-1} is in the set.

3 Elliptic Curves

Let's define a group over elliptic curves $G = (E(x), +), \forall x \in \mathbb{R}$ with:

$$E(x)^2 = x^3 + ax + b$$

with $a, b \in \mathbb{N}$

3.1 Pre-group construction

Let us define the + operator for points on the elliptic curve. For any two points A and B on the curve, the sum $A + B$ is defined as follows:

- Draw the slope s passing through P and Q.
- Let R be the third point of intersection of this line with the elliptic curve.
- The result $P + Q$ is the point symmetric to C with respect to the x-axis (i.e., the reflection of R across the x-axis).

We can compute the coordinates of this point:

3.1.1 Slope between P and Q (affine function)

$$y = sx + C \tag{2}$$

with $s = \frac{y_q - y_p}{x_q - xp}$. We finally have that 2 is:

$$y = s(x - x_p) + y_p \tag{3}$$

3.1.2 Compute R

First case: $P \neq Q$

R is the intersection between the curve and the P - Q slope. Hence, it is the solution of:

$$\begin{aligned} x^3 + ax + b &= s^2(x - x_p)^2 + y_p^2 + 2y_p + 2s(x - x_p) \\ \iff 0 &= x^3 + ax^2 + b - s^2(x - x_p)^2 - y_p^2 + 2y_p + 2s(x - x_p) \\ &= x^3 + ax + b - s^2(x^2 - 2xx_p + x_p^2) - y_p^2 + 2y_p + 2s(x - x_p) \\ &= x^3 + ax - s^2x^2 + 2s^2xx_p - s^2x_p^2 - y_p^2 + b + 2y_p + 2s(x - x_p) \\ &= \underbrace{1}_{\text{u}} x^3 + \underbrace{-s^2}_{\text{v}} x^2 + x \underbrace{(a + 2s^2x_p - 2sy_p)}_{\text{w}} + \underbrace{b - (sx_p - y_p)^2}_{\text{z}} \end{aligned} \tag{4}$$

As we are looking for intersections between the curve and the P-Q slope, we known that x_p, x_q and x_r are solutions.
Hence, using Vieta formulas for a cubic polynom, we have that:

$$x_p + x_q + x_r = \frac{-v}{u} = s^2 \tag{5}$$

$$x_p x_q x_r = \frac{-z}{u} = -z = -b + (sx_p - y_p)^2 \tag{6}$$

$$x_p x_q + x_p x_r = \frac{w}{u} = (a + 2s^2x_p - 2sy_p) \tag{7}$$

Therefore, we have that:

$$x_r = s^2 - x_p - x_q$$

So 3 becomes:

$$\begin{aligned} y_r &= s(x_r - x_p) + y_p \\ &= y_p + x(s^2 - x_q - 2x_p) \end{aligned} \tag{8}$$

Second case: $P = Q$

In this case, we need to compute the tangent to P. To do so, we know that E is a linear combinations of differentiable functions:

$$\begin{aligned} 2ydy &= 3x^2dx + a \\ \iff \frac{dy}{dx} &= \frac{3x^2 + a}{2y} \end{aligned} \tag{9}$$

Finally, we have that:

$$s = \frac{3x_p^2 + a}{2y_p} \tag{10}$$

The result for y_r and x_r is the same as the first case with a different value and $x_q = x_p$.

Third case: $x_p = x_q$ **but** $y_p \neq y_q$

We just return the point at infinity **O**.

3.1.3 Group justification

Now that we have defined the + operator, we can easily verify graphically that the operator satisfies the following properties to ensure G is a group:

- + is an internal operator.
- + is associative.
- G holds a neutral element
- $\forall x \in E(X)$, a symmetric element exists.

3.2 Cyclic group construction

As introduced before, we want to link the group G to the DL problem. To do so, E has to operate on the finite field we defined later: $\mathbb{F}_n$.
Therefore, we now define G as $G = (E(\mathbb{F}_n), +)$.
The modular operator leads to the preservation of + properties. Hence, G is still a group.

In order to prove that G is cylic, we need to find $g \in E(\mathbb{F}_n)$ such as $G =< g >$. We will admit that such a point exists (it can be proved through the fundamental theorem of finite abelian groups and the Weil pairing).
Hence, we have that $\forall P \in E(\mathbb{F}_n), \exists e \in \mathbb{N}$ such as:

$$P = eg$$

If we don't know e, to compute P, we need to solve: $log_G(P) = e$, log_g being the elliptic curve discrete logarithm.
Hence, we have drawn the link with the DL problem.

4 Cryptosystem

4.0.1 Construction

To define the cryptosystem on elliptic curves, we have to declare the following constants:

- Parameters of the elliptic curves: a and b
- The generator point g
- The prime number n for the finite set
- The public key: P
- The private key: e

Without knowing e but only P, g, it is computationally hard to find e.

4.0.2 Robustness

The robustness of this cryptosystem will depend on the order of the group G. Accordingly to Hasse's theorem, we have the relation:

$$\#E(\mathbb{F}_n) \leq q + 1 + 2 * \sqrt{q}$$

with q the number of elements in $\mathbb{F}_n$

5 ECDSA

ECDSA is a signature algorithm with the capability to do signature generation and signature verification.
Through it, someone can verify that the signature was produced by someone who owns the private key for the associated public key, and only that person could have generated the signature.
The first part to construct such an algorithm is to find a way as an owner of the private key to prove that we own it, i.e. proving to have the knowledge of the discrete log relationship between a public key P and the generator point G.

5.0.1 First non-interactive naive attempt

Let us say we have: $P = eG$ and Q such as $Q = kP, k \in \mathbb{N}, Q = jG, j \in \mathbb{N}$.
Therefore, we have that

$$kP = jG \iff keG = jG \iff k = j/e$$

. To compute Q, the prover has to use the private key e. After being computed, the prover can send the tuple (Q, k) so that the verifier can try to compute Q from P with k.
However, this solution has an issue: The prover can pick a random k and compute Q from this k without having to use the private key.

5.0.2 Second attempt with an interactive solution

As highlighted in the first attempt, we need a way from the verifier side to compute by himself the coefficient k such as Q = k(hG + P).
The prover will then have to compute k such as the previous relationship is true.
To do so, here are the different steps of the algorithm:

- Prover creates a random j such as $Q = jG$
- Verifier sends a random h
- Prover find w such as $Q = k(hG + P) = jG \iff (hG + eG)k = jG \iff k = j/(h + e)$
- Prover sends k and Q to the verifier which can then try to compute Q by himself.

The main issue with this algorithm is that it is an interactive solution.

5.0.3 Third attempt with a non-interactive solution

We need a way to generate a random value h without having to interact with the verifier. To do so, we can say that $h = hash(Q)$. The verifier can also in this case compute h and the robustness of the previous solution is preserved. However, this solution is still subject to forgeries if the prover picks a random k.
To fix this issue, we will have: h = hash(message) and r = hash(Q). The algorithm then becomes:

- Prover receives a message to message
- Prover creates a random j such as $Q = jG$
- Prover computes h such as h = hash(message)
- Prover compute r such as r = hash(Q)

- Prover find w such as $Q = k(hG + rP) = jG \iff (hG + erG)k = jG \iff k = j/(h + er)$
- Prover sends k and Q to the verifier which can then try to compute Q by himself.