

## Exercise 1

To show that  $(m_1, m_2) = (m'_1, m'_2)$ , we have to show that  $m_1 \equiv x_T^{-1}c_1 \pmod{p}$  and  $m_2 \equiv y_T^{-1}c_2 \pmod{p}$ .

$$\begin{array}{l} m_1 \equiv x_T^{-1}c_1 \pmod{p} \wedge m_2 \equiv y_T^{-1}c_2 \pmod{p} \\ m_1 \equiv x_T^{-1}x_S m_1 \pmod{p} \wedge m_2 \equiv y_T^{-1}y_S m_2 \pmod{p} \end{array} \quad \left. \begin{array}{l} \\ \end{array} \right\} \begin{array}{l} c_1 \equiv x_S m_1 \pmod{p} \text{ and} \\ c_2 \equiv y_S m_2 \pmod{p} \end{array}$$

For  $(m_1, m_2) = (m'_1, m'_2)$  to hold, we just have to show that  $T^{-1}S \equiv 1 \pmod{p}$ .

$$\begin{array}{l} T^{-1}S \equiv (n_A R)^{-1}kQ_A \pmod{p} \\ \equiv n_A^{-1}(kP)^{-1}kn_AP \pmod{p} \\ \equiv 1 \pmod{p} \end{array} \quad \left. \begin{array}{l} \\ \\ \end{array} \right\} \begin{array}{l} T = n_A R \text{ and } S = kQ_A \\ R = kP \text{ and } Q_A = n_AP \\ n_A n_A^{-1} = 1, k k^{-1} = 1 \\ \text{and } PP^{-1} = 1 \end{array}$$

Thus,  $(m_1, m_2) = (m'_1, m'_2)$ .

## Exercise 2

```
from sage.all import *

def gen_pub_key(A, B, p, x_p, y_p):
    Fp = FiniteField(p)
    E = EllipticCurve(Fp, [A, B])
    assert(E.is_on_curve(x_p, y_p))

    P = E([x_p, y_p])
    n_a = ZZ(Fp.random_element())
    Q_A = n_a * P

    return (Q_A, n_a)

def encrypt(A, B, p, x_p, y_p, Q_A, m_1, m_2):
    Fp = FiniteField(p)
    E = EllipticCurve(Fp, [A, B])
    assert(E.is_on_curve(x_p, y_p))

    P = E([x_p, y_p])
    k = ZZ(Fp.random_element())
    R = k * P

    S = k * Q_A
    c_1 = (S[0] * m_1) % p
    c_2 = (S[1] * m_2) % p

    return (R, c_1, c_2)
```

```

def decrypt(A, B, p, x_p, y_p, R, n_a, c_1, c_2):
    Fp = FiniteField(p)
    E = EllipticCurve(Fp, [A, B])
    assert(E.is_on_curve(x_p, y_p))

    T = n_a * R
    m_1 = (T[0] ** (-1) * c_1) % p
    m_2 = (T[1] ** (-1) * c_2) % p

    return (m_1, m_2)

```

Running this functions with the Curve P-384, we can see the functions are defined correctly (Fig. 1).

```

rui@rui-VivoBook-ASUSLaptop-X513IA-M513IA:~/Desktop/FCUP/Criptografia Aplicada/TPC/8 - Criptografia de Curva Eliptica
→ 8 - Criptografia de Curva Eliptica python -i ellipticcurve.py
>>> p = 2**384 - 2**128 - 2**96 + 2**32 - 1
>>> A = -3
>>> B = 0xb3312fa7e23ee7e4988e056be3f82d19181d9c6efe8141120314088f5013875ac656398d8a2ed19d2a85c8edd3ec2aef
>>> Fp = FiniteField(p)
>>> E = EllipticCurve(Fp, [A, B])
>>> P = E.random_element()
>>> x_p, y_p = P[0], P[1]
>>>
>>> Q_A, n_a = gen_pub_key(A, B, p, x_p, y_p)
>>>
>>> m_1 = 15
>>> m_2 = 42
>>>
>>> R_, c_1, c_2 = encrypt(A, B, p, x_p, y_p, Q_A, m_1, m_2)
>>>
>>> d_1, d_2 = decrypt(A, B, p, x_p, y_p, R_, n_a, c_1, c_2)
>>>
>>> d_1
15
>>> d_2
42
>>>
→ 8 - Criptografia de Curva Eliptica

```

Figure 1: Menezes–Vanstone variant for ECC ElGamal