Exercício 3

Implementação do Esquema ElGamal em Curvas Elípticas

In [88]:
```
from sage.all import * # type: ignore
import hashlib
from ex2 import ECDSA  # Reutilizamos a classe ECDSA para obter a curva
```

1. Implementação do Esquema ElGamal IND-CPA Seguro

- Implementação do esquema PKE ElGamal em curvas elípticas
- Utiliza o grupo abeliano aditivo da curva elíptica escolhida (ed25519)

In [89]:
```
class PKE_ElGamal_EC_CPA:
    def __init__(self, curve):
        self.curve = curve  # Usamos a curva elíptica definida (ed25519)
        self.G = curve.G  # Ponto gerador da curva
        self.n = curve.n  # Ordem do subgrupo

    def keygen(self):
        """Gera um par de chaves pública e privada."""
        private_key = ZZ.random_element(1, self.n)  # Chave privada
        public_key = self.G.mult(private_key)  # Chave pública = sk * G
        return private_key, public_key

    def encrypt(self, public_key, plaintext_point):
        """Cifra um ponto da curva utilizando ElGamal-EC IND-CPA."""
        r = ZZ.random_element(1, self.n)  # nonce aleatório
        C1 = self.G.mult(r)  # C1 = r * G
        C2 = plaintext_point.copy()
        C2.soma(public_key.mult(r))  # C2 = P + r * pk
        return (C1, C2)

    def decrypt(self, private_key, ciphertext):
        """Decifra um ponto da curva utilizando ElGamal-EC IND-CPA."""
        C1, C2 = ciphertext
        shared_secret = C1.mult(private_key)  # sk * C1
        shared_secret_inv = shared_secret.sim()  # Inverso do ponto
        decrypted_point = C2.copy()
        decrypted_point.soma(shared_secret_inv)  # Recuperar P = C2 - sk * C1
        return decrypted_point
```

1.1. Teste do IND-CPA com ataques

In [90]:
```
def test_elgamal_ec_cpa():
    print("\n=== Teste do PKE ElGamal EC IND-CPA ===")
    curve = ECDSA("ed25519")
    elgamal = PKE_ElGamal_EC_CPA(curve)
    sk, pk = elgamal.keygen()
    plaintext = curve.G.mult(5)
    print("\nMensagem original:")
    print(f"({plaintext.x}, {plaintext.y})")

    print("\nA cifrar...")
    ciphertext = elgamal.encrypt(pk, plaintext)
```

```
    print(f"C1: ({ciphertext[0].x}, {ciphertext[0].y})")
    print(f"C2: ({ciphertext[1].x}, {ciphertext[1].y})")

    print("\nA decifrar...")
    decrypted = elgamal.decrypt(sk, ciphertext)
    print(f"Mensagem decifrada: ({decrypted.x}, {decrypted.y})")
    assert decrypted.eq(plaintext), "Erro: Mensagem não foi recuperada corretament
    print("✓ Teste bem-sucedido: ElGamal-EC-CPA está a funcionar corretamente!\n")

    # Teste de integridade - Modificação em C1
    print("\nTeste: Modificação de C1")
    try:
        c1_mod = ciphertext[0].copy()
        c1_mod.soma(curve.G)  # Modificar C1
        ciphertext_mod = (c1_mod, ciphertext[1])
        elgamal.decrypt(sk, ciphertext_mod)
        print("✗ Erro: Decifrou com C1 adulterado! (esperado)")
    except Exception as e:
        print("✓ Ataque detetado: falha na integridade de C1.")

    # Teste de integridade - Modificação em C2
    print("\nTeste: Modificação de C2")
    try:
        c2_mod = ciphertext[1].copy()
        c2_mod.soma(curve.G)  # Modificar C2
        ciphertext_mod = (ciphertext[0], c2_mod)
        elgamal.decrypt(sk, ciphertext_mod)
        print("✗ Erro: Decifrou com C2 adulterado! (esperado)")
    except Exception as e:
        print("✓ Ataque detetado: falha na integridade de C2.")
```

2. Implementação do Esquema ElGamal IND-CCA Seguro

In [91]:

```
class PKE_ElGamal_EC_CCA(PKE_ElGamal_EC_CPA):
    def __init__(self, curve):
        super().__init__(curve)

    def hash_point(self, point):
        """Transforma um ponto da curva num hash SHA-256."""
        point_bytes = (str(point.x) + str(point.y)).encode()
        return hashlib.sha256(point_bytes).digest()

    def encrypt_cca(self, public_key, plaintext_point):
        """Cifra um ponto da curva com ElGamal-EC IND-CCA."""
        r = ZZ.random_element(1, self.n)
        C1 = self.G.mult(r)
        C2 = plaintext_point.copy()
        C2.soma(public_key.mult(r))

        hashed_C1 = self.hash_point(C1)
        hashed_C2 = self.hash_point(C2)

        return (C1, C2, hashed_C1, hashed_C2)

    def decrypt_cca(self, private_key, ciphertext):
        """Decifra um ponto da curva com ElGamal-EC IND-CCA."""
        C1, C2, hash_C1, hash_C2 = ciphertext

        # Verificar a integridade de C1 e C2
        if self.hash_point(C1) != hash_C1 or self.hash_point(C2) != hash_C2:
            raise ValueError("Falha na verificação da integridade: cifra alterada!

        shared_secret = C1.mult(private_key)
```

```
        shared_secret_inv = shared_secret.sim()
        decrypted_point = C2.copy()
        decrypted_point.soma(shared_secret_inv)

        return decrypted_point
```

2.2. Teste do IND-CCA com ataques

In [92]:

```
def test_elgamal_ec_cca():
    print("\n=== Teste do PKE ElGamal EC IND-CCA ===")
    curve = ECDSA("ed25519")
    elgamal = PKE_ElGamal_EC_CCA(curve)
    sk, pk = elgamal.keygen()
    plaintext = curve.G.mult(5)
    print("\nMensagem original:")
    print(f"({plaintext.x}, {plaintext.y})")

    print("\nA cifrar...")
    ciphertext = elgamal.encrypt_cca(pk, plaintext)
    print(f"C1: ({ciphertext[0].x}, {ciphertext[0].y})")
    print(f"C2: ({ciphertext[1].x}, {ciphertext[1].y})")

    print("\nA decifrar...")
    decrypted = elgamal.decrypt_cca(sk, ciphertext)
    print(f"Mensagem decifrada: ({decrypted.x}, {decrypted.y})")
    assert decrypted.eq(plaintext), "Erro: Mensagem não foi recuperada corretament
    print("✓ Teste bem-sucedido: ElGamal-EC-CCA a funcionar corretamente!\n")

    # Teste de integridade - Modificação em C1
    print("\nTeste: Modificação de C1")
    try:
        c1_mod = ciphertext[0].copy()
        c1_mod.soma(curve.G)  # Modificar C1
        ciphertext_mod = (c1_mod, ciphertext[1], ciphertext[2], ciphertext[3])
        elgamal.decrypt_cca(sk, ciphertext_mod)
        print("✗ Erro: Decifrou com C1 adulterado! (não esperado)")
    except Exception as e:
        print("✓ Ataque detetado: falha na integridade de C1.")

    # Teste de integridade - Modificação em C2
    print("\nTeste: Modificação de C2")
    try:
        c2_mod = ciphertext[1].copy()
        c2_mod.soma(curve.G)  # Modificar C2
        ciphertext_mod = (ciphertext[0], c2_mod, ciphertext[2], ciphertext[3])
        elgamal.decrypt_cca(sk, ciphertext_mod)
        print("✗ Erro: Decifrou com C2 adulterado! (não esperado)")
    except Exception as e:
        print("✓ Ataque detetado: falha na integridade de C2.")

    # Teste de integridade - Modificação no hash
    print("\nTeste: Modificação do hash de integridade")
    try:
        hash_mod = bytearray(ciphertext[3])
        hash_mod[0] ^= 0xFF  # Alterar um bit do hash de C2
        ciphertext_mod = (ciphertext[0], ciphertext[1], ciphertext[2], bytes(hash_
        elgamal.decrypt_cca(sk, ciphertext_mod)
        print("✗ Erro: Decifrou com hash adulterado! (não esperado)")
    except Exception as e:
        print("✓ Ataque detetado: falha na verificação do hash.")
```

3. Implementação do protocolo "Oblivious Transfer" k-out-of-n

In [93]:
```
## to do
```

4. Main

In [94]:
```
if __name__ == "__main__":
    # teste para o IND-CPA seguro
    test_elgamal_ec_cpa()
    # teste para o IND-CCA seguro
    test_elgamal_ec_cca()
```

```
=== Teste do PKE ElGamal EC IND-CPA ===

Mensagem original:
(33467004535436536005251147249499675200073690106659565782908757308821616914995, 43
097193783671926753355113395909008640284023746042808659097434958891230611693)

A cifrar...
C1: (15240041954961234022580125845879390008640072474979817058862180335794896038375, 20859954557781896330836201713757706475957773115521892689817097975591187747851)
C2: (54910518094461008196066884751897850363667253334426018082446206706803919536795, 50162109888492580364772290350060389779189386628301895007812951116349592376246)

A decifrar...
Mensagem decifrada: (33467004535436536005251147249499675200073690106659565782908757308821616914995, 43097193783671926753355113395909008640284023746042808659097434958891230611693)
✓ Teste bem-sucedido: ElGamal-EC-CPA está a funcionar corretamente!


Teste: Modificação de C1
✗ Erro: Decifrou com C1 adulterado! (esperado)

Teste: Modificação de C2
✗ Erro: Decifrou com C2 adulterado! (esperado)

=== Teste do PKE ElGamal EC IND-CCA ===

Mensagem original:
(33467004535436536005251147249499675200073690106659565782908757308821616914995, 43097193783671926753355113395909008640284023746042808659097434958891230611693)

A cifrar...
C1: (46159746961648427778564494129429002304713872201803344741660094844737760370716, 33449764459168793293830830488166169893824167885533855103972040313457991686928)
C2: (55307092973967267335793312204527086703404066525662567884000664861915098236871, 35724842133974601959356032595093328932790896165010628400894808341966798996361)

A decifrar...
Mensagem decifrada: (33467004535436536005251147249499675200073690106659565782908757308821616914995, 43097193783671926753355113395909008640284023746042808659097434958891230611693)
✓ Teste bem-sucedido: ElGamal-EC-CCA a funcionar corretamente!


Teste: Modificação de C1
✓ Ataque detetado: falha na integridade de C1.

Teste: Modificação de C2
✓ Ataque detetado: falha na integridade de C2.

Teste: Modificação do hash de integridade
✓ Ataque detetado: falha na verificação do hash.
```