Trabalho Prático 1 - Problema 2 - Grupo 18

Introdução ao problema

Neste problema foi-nos proposto que construíssemos uma classe Python que implementasse um KEM-RSA. De seguida, devemos ser capazes de construir um PKE que seja IND-CCA seguro, usando o KEM construído anteriormente e a transformação de Fujisaki-Okamoto.

Resolução do problema

In [ ]:
```
import os, hashlib

class KEM_RSA_receiver:

    def __init__(self, param_size):
        self.p = random_prime(2**param_size - 1, False, 2**(param_size-1)
        self.q = random_prime(2**(param_size+1) - 1, False, 2**param_size
        self.n = self.p * self.q
        self.phi = (self.p-1) * (self.q-1)

        self.pu = ZZ.random_element(self.phi)
        while gcd(self.pu, self.phi) != 1:
            self.pu = ZZ.random_element(self.phi)

        self.__pr = Integer( mod( xgcd(self.pu, self.phi)[1], self.phi )

    def h(self):

        return ZZ.random_element(self.n)

    def f(self, r, salt):
        e = power_mod(int(r.decode()), self.pu, self.n)

        k = hashlib.pbkdf2_hmac('sha256', r, salt, 500000)

        return(str(e).encode(), k)

    def reveal(self, salt, c):

        r = power_mod(int(c.decode()), self.__pr, self.n)

        k = hashlib.pbkdf2_hmac('sha256', str(r).encode(), salt, 500000)

        return k
```

In [ ]:
```
class KEM_RSA_sender:

    def __init__(self, pu, n):

        self.pu = pu
        self.n = n

    def h(self):

        return ZZ.random_element(self.n)

    def f(self, r, salt):
        e = power_mod(int(r.decode()), self.pu, self.n)

        k = hashlib.pbkdf2_hmac('sha256', r, salt, 500000)

        return(str(e).encode(), k)

    def encapsulate(self):

        r = self.h()

        salt = os.urandom(16)
        ek = self.f(str(r).encode(), salt)

        return (ek, salt)
```

In [ ]:
```
kem_rsa_receiver = KEM_RSA_receiver(1024)
kem_rsa_sender = KEM_RSA_sender(kem_rsa_receiver.pu, kem_rsa_receiver.n)

(ek, salt) = kem_rsa_sender.encapsulate()
print('Encapsulated key:', ek[1])

decapsulated_key = kem_rsa_receiver.reveal(salt, ek[0])
print('Decapsulated key:', decapsulated_key)
```

```
Encapsulated key: b"\xdbH\x98Fbo\x8a\xc5$Ayg&N\x9f']\xcf\x86\x01\xe8\xa9\
xab\xf6\xa6\x1c\x10\xe3a\x11\xdb\xaf"
Decapsulated key: b"\xdbH\x98Fbo\x8a\xc5$Ayg&N\x9f']\xcf\x86\x01\xe8\xa9\
xab\xf6\xa6\x1c\x10\xe3a\x11\xdb\xaf"
```

In [ ]:
```
def xor(xs, ys):

    masked = b''
    i = 0
    while i < len(xs):
        for j in range(len(ys)):
            if i < len(xs):
                masked += (xs[i] ^^ ys[j]).to_bytes(1, byteorder='big')
                i += 1
            else:
                break

    return masked
```

In [ ]:
```
class PKE_sender:

    def __init__(self, kem_rsa):
        self.kem_rsa = kem_rsa

    def encrypt(self, x):

        # ϑr ← h
        r = self.kem_rsa.h()

        # ϑy ← x ⊕ g(r)
        gr = hashlib.sha256(str(r).encode()).digest()
        y = xor(x, gr)

        # (e,k) ← f(y‖r)
        yi = Integer('0x' + hashlib.sha256(y).hexdigest())
        salt = os.urandom(16)
        (e, k) = self.kem_rsa.f(str(yi + r).encode(), salt)

        # ϑc ← k ⊕ r
        c = xor(str(r).encode(), k)

        # E´(x) ≡ (y,e,c)
        return (y, (e, salt), c)
```

In [ ]:
```
class PKE_receiver:

    def __init__(self, kem_rsa):
        self.kem_rsa = kem_rsa

    def decrypt(self, y, e, salt, c):

        # ϑk ← KREv(e)
        k = self.kem_rsa.reveal(salt, e)

        # ϑr ← c ⊕ k
        r = xor(c, k)

        # if (e,k) ≠ f(y‖r) then ⊥ else y ⊕ g(r)
        yi = Integer('0x' + hashlib.sha256(y).hexdigest())
        ek = self.kem_rsa.f(str(yi + int(r)).encode(), salt)

        if (e, k) != (ek[0], ek[1]): # if (e,k) ≠ f(y‖r) then ⊥
            raise IOError
        else: # else y ⊕ g(r)
            gr = hashlib.sha256(r).digest()
            pt = xor(y, gr)

        # D´(y,e,c)
        return pt
```

In [ ]:
```
msg = 'Hello World!'
pke_sender = PKE_sender(kem_rsa_sender)
(y, (e, salt), c) = pke_sender.encrypt(msg.encode())

pke_receiver = PKE_receiver(kem_rsa_receiver)
pt = pke_receiver.decrypt(y, e, salt, c)

if (msg == pt.decode()):
    print('A mensagem foi recebida corretamente.')
else:
    print('A mensagem foi recebida incorretamente.')
```

```
A mensagem foi recebida corretamente.
```