COM-401 2024 HW1 Solutions

December 2024

1 Exercise 1

Q1a

This consists of implementing the given pseudocode directly. A common mistake is not inverting $4t$ modulo N and doing integer division.

Q1b

This consists of implementing a square root finder modulo N given the factors of N. One can easily implement the extract function afterwards. Example implementation:

```
def sqrt_mod_N(a, p, q):
"""
Compute square root of a modulo N = p*q
Returns one of the square roots (there are 4 if they exist)
"""
N = p*q

# First check if a is a quadratic residue modulo p and q
if not is_square(Mod(a,p)) or not is_square(Mod(a,q)):
    return None  # No square root exists

# Compute square roots modulo p and q
rp = Mod(a,p).sqrt()
rq = Mod(a,q).sqrt()

# Use CRT to get a root modulo N
root = CRT(Integer(rp), Integer(rq), p, q)
# return one of the roots
return max([root, -root % N, N-root, root - N])
```

Q1c

Since the key generation picks primes that are very close to each other. It is possible to take the square root of the modulus and search for the prime factors. Also known as Fermat Factorization.

Q1d

Let's compute $c^3 + a$:

$$
\begin{aligned}
c^3 + a &= \left(\frac{t(t^3 - 8a)}{4(t^3 + a)}\right)^3 + a && (1)\\
&= \frac{t^{12} - 24at^9 + 192a^2t^6 - 512a^3t^3}{64(t^3 + a)^3} + a && (2)\\
&= \frac{t^{12} - 24at^9 + 192a^2t^6 - 512a^3t^3 + 64a(t^3 + a)^3}{64(t^3 + a)^3} && (3)\\
&= \frac{t^{12} - 24at^9 + 192a^2t^6 - 512a^3t^3 + 64at^9 + 192a^2t^6 + 192a^3t^3 + 64a^4}{64(t^3 + a)^3} && (4)\\
&= \frac{t^{12} + 40at^9 + 384a^2t^6 - 320a^3t^3 + 64a^4}{64(t^3 + a)^3} && (5)\\
&= \frac{(t^6 + 20at^3 - 8a^2)^2}{64(t^3 + a)^3} && (6)\\
& && (7)
\end{aligned}
$$

We compute the jacobi symbol of the resulting value $(\frac{c^3+a}{N})$, and both the nominator and the 64 disappear because they are quadratic residues. The resulting expression is equal to m.

2 Exercise 2

Q2a

The crux of this exercise consists of efficiently computing $(M, \phi_{H_1,H_2})^n$. To do so, we simply have to adapt the square and multiply algorithm, using the fact that $\phi^{(n)}_{H_1,H_2} = \phi_{H_1^n,H_2^n}$. See Algorithm 1 below.

Q2b

It suffices to observe that

$$H_1AH_2 - A = \sum_{i=0}^{a-1} \left(H_1^{i+1}MH_2^{i+1} - H_1^iMH_2^i\right) = H_1^aMH_2^a - M,$$

Algorithm 1 Square and Multiply on the Holomorph

Require: $(M, \phi_{H_1,H_2}) \in \mathcal{H}(G)$ and $n \in \mathbb{N}$ the exponent
Ensure: $y = (M, \phi_{H_1,H_2})^n$

$y \leftarrow (0, id)$ ▷ We note $y = (y_0, \phi_{H_1^y,H_2^y})$
$z \leftarrow (M, \phi_{H_1,H_2})$ ▷ We note $z = (z_0, \phi_{H_1^z,H_2^z})$
while $e > 0$ **do**
  if $e \bmod 2 = 1$ **then**
    $y \leftarrow \left(y_0 + \phi_{H_1^y,H_2^y}(z_0), \phi_{H_1^{y+1},H_2^{y+1}}\right)$
  end if
  $z \leftarrow \left(z_0 + \phi_{H_1^z,H_2^z}(z_0), \phi_{H_1^{2z},H_2^{2z}}\right)$
  $e \leftarrow \lfloor e/2 \rfloor$
end while
return y

which implies

$$\deg\left((H_1AH_2 - A - M)M^{-1}\right) = \deg(H_1^aMH_2^aM^{-1}) = \deg(H_1H_2)^a.$$

Q2c

From the previous question, we know that, from A, it is possible to retrieve $H_1^aMH_2^a$. Let us consider a vector t such that

$$\mathsf{L}(M)\mathsf{t} = \mathsf{vec}(H_1^aMH_2^a).$$

Now, we need to establish two key properties:

1. L and vec are linear function, i.e., $\mathsf{L}(X+Y) = \mathsf{L}(X) + \mathsf{L}(Y)$ and $\mathsf{vec}(X + Y) = \mathsf{vec}(X) + \mathsf{vec}(Y)$.
2. The operation $\mathsf{L}(X)v = \mathsf{vec}(Y)$ can be interpreted as

$$\mathsf{L}(X)v = \sum_{0 \le i,j \le 2} v_{i,j}(H_1^iXH_2^j) = Y.$$

   From this, it follows that

$$\mathsf{L}(X)v = \mathsf{vec}(Y) \implies \mathsf{L}(H_1XH_2)v = \mathsf{vec}(H_1YH_2).$$

Combining these results, we obtain:

$$\mathsf{L}(B)\mathsf{t} = \sum_{i=0}^{b-1} \mathsf{L}(H_1^iMH_2^i)\mathsf{t} = \sum_{i=0}^{b-1} \mathsf{vec}(H_1^{i+a}MH_2^{i+a}) = \mathsf{vec}(H_1^aBH_2^a)$$

which shows that, given t and B, we can recover $H_1^aBH_2^a$. Since $K = A + H_1^aBH_2^a$, we can retrieve the shared secret.

2.1 Exercise 3

Q3a

The idea behind the question is that we switch the multiplicative homomorphism property of El-Gamal encryption to additive homomorphism by encrypting g^x instead of x. This way, we can add and scalar multiply in the exponent modulo $p-1$ (the neuron operates modulo $p-1$ so this is enough for us).

Given a vector of n ciphertexts ($\{c_i\}_{i\in[n]}$) and n weights ($\{w_i\}_{i\in[n]}$). We first compute:

$$answer = \prod_{i=1}^{n} c_i^{w_i} \mod p \tag{8}$$

We multiply with the bias ciphertext to add the bias in the exponent.

$$answer = answer \cdot cb \mod p \tag{9}$$

Lastly, we need to divide the exponent by 1009. This was a common mistake in most solutions. to divide in the exponent, 1009 should be inverted modulo $p-1$.

$$answer = answer^{1009^{-1}} \mod p \tag{10}$$