Homework 4: Criptografia

Fabian Andres Prada Nino

The arithmetic procedures in this homework were done using 64 bit integers (**long long int** in C++).

In order to decode the message the following steps were implemented:

1) **Calculating** $\phi(N)$.

Given $N = 2147483621 = (14741)(145681)$,where $p = 14741$ and $q = 145681$ are primes numbers, we easily get $\phi(N) = (p-1)(q-1) = (14740)(145680) = 2147323200$.

2) **Finding** d :=Inverse of e modulo $\phi(N)$.

Here I implemented the extended version of Euclid's algorithm of MCD (*Maximium Common Divisor*). Given a pair of positive integer (a, b) (inputs), this algorithm find a set of integers (x, y, MCD) (outputs) that satisfies $ax + by = MCD$. The algorithm proceeds recursively as follows:

$extMCD(a, b)$:

- **Set** $q = \lfloor \frac{a}{b} \rfloor$.
- **Set** $r = a - bq$.
- **If**$(r == 0)$: **return** $(x = 0, y = 1, MCD = b)$.
- **Else:**
  Set $(x', y', MCD') = extMCD(b, r)$
  return$(x = y', y = x' - y'q, MCD = MCD')$.

Using this algorithm I got $(\phi(N))(6795) + (e)(-222638527) = 1$. Therefore we can set $d := 1924684673 = -222638527 + \phi(N)$, as the inverse of e modulo $\phi(N)$.

3) **Calculating** $x^d(modN)$ for each encoded block.

In order to calculate the dth power of a number an reduce it modulo N, I implemented a method that works recusively as follows:

$expMod(numero, exponente, modulo)$:

- **if** $(exponente == 0)$ **return** 1.
- **else**
  - **if**$(exponente(mod\ 2) == 0)$
  - $resultado' = expMod(numero, \frac{exponente}{2}, modulo)$
  - $cociente = \lfloor \frac{(resultado')^2}{modulo} \rfloor$
  - **return** $resultado = (resultado)'^2 - (cociente) * (modulo)$
  - **else**
  - $resultado' = expMod(numero, \frac{exponente-1}{2}, modulo)$
  - $cocienteParcial = \lfloor \frac{(resultado')^2}{modulo} \rfloor$
  - $resultadoParcial = (resultado')^2 - (cocienteParcial) * (modulo)$
  - $cocienteFinal = \lfloor \frac{(resultadoParcial)*(numero)}{modulo} \rfloor$
  - **return** $resultado = (resultadoParcial) * (numero) - (cocienteFinal) * (modulo)$

Observations

- The previous algorithm performs $O(\log_2(exponente))$ products, divisions and substractions to compute $x^d(modN)$.
- Since $N < 2^{32}$, the product of any two numbers already reduced modulo N (i.e, two numbers $a, b \in \{0, 1, ..., N-1\}$) is less than 2^{64}. Then products (as well as divisions and substractions) are exactly performed using 64 bit arithmetics.

Applying the previous algorithm to the encoded numbers I get:

- $x_1^d \equiv 5395265 \bmod N$.
- $x_2^d \equiv 3809362 \bmod N$.
- $x_3^d \equiv 6911589 \bmod N$.
- $x_4^d \equiv 7566368 \bmod N$.
- $x_5^d \equiv 2826323 \bmod N$.
- $x_6^d \equiv 6840685 \bmod N$.

- $x_7^d \equiv 6910496 \bmod N$.
- $x_8^d \equiv 2826305 \bmod N$.
- $x_9^d \equiv 6581349 \bmod N$.
- $x_{10}^d \equiv 7168366 \bmod N$.

3) **Descomposing** $x^d(modN)$ in base 256.

I applied the following method:

$descomposicionBase(numero, base)$

- $coef2 = \lfloor \frac{numero}{(base)^2} \rfloor$
- $coef1 = \lfloor \frac{numero - coef2 * (base)^2}{base} \rfloor$
- $coef0 = numero - coef2 * (base)^2 - coef1 * base$
- **return** $(coef2, coef1, coef0)$

Converting these coeficients to characters using the ASCII table I obtained the message:

RSA: Rivest + Shamir + Adleman