CS411-HW3

Ahmet Furkan Ün

November 23, 2023

Question 1

Since both message M and e very small compared to N, we can say that ciphertext is M^e. In other words, $M^e = M^e$ mod N. Therefore M can be calculated as $M = C^{1/e}$. Therefore I could easily obtained the message with this basic calculation. However, the cipertext was too large for standard python library to calculate. So I used "decimal" library to overcome "OverflowError: int too large to convert to float" error. After that I could easily calculate $C^{1/e}$ and find the message "340282366920938463463374607431768211455" which has 128-bit length.

Question 2

(a)

When an attacker obtains both c_p and c_q, they essentially have the following equations:

$$c_p \equiv (k \cdot p)^e \pmod{n}$$
$$c_q \equiv (k \cdot q)^e \pmod{n}$$

Then the GCD of c_p and n:

$$\gcd(c_p, n) = \gcd((k \cdot p)^e, n)$$

Since k is coprime to n (n is $p \cdot q$ and is relatively prime to k if $k \neq p$ and $k \neq q$ which is not logical), we can rewrite this as:

$$\gcd(c_p, n) = \gcd(p^e, n)$$

Now, because c_p is congruent to $p^e \pmod{n}$, the GCD can be written as:

$$\gcd(c_p, n) = \gcd(c_p - p^e, n)$$

This implies that p divides $(c_p - p^e)$. In other words, p is a factor of the difference $c_p - p^e$. This is because $c_p \equiv p^e \pmod{n}$ implies that $c_p - p^e$ is a multiple of n, and since p is a prime factor of n, it must also be a factor of $c_p - p^e$.

Similar reasoning can be applied to c_q. Therefore, when the attacker computes the GCD of c_p and n, the result will be p, and when they compute the GCD of c_q and n, the result will be q.

The security of the RSA depends on the security of the factors p and q. This approach reveals the p and q.

(b)

As explained in the Part (a), to obtain prime numbers p and q where $p \cdot q = n$, I calculated the $\gcd(c_p, n)$ and $\gcd(c_q, n)$ to get factors p and q respectively. After that calculating $\phi(n)$ is easy since $\phi(n) = (p-1) \cdot (q-1)$. Then I calculated the d as e^{-1} mod $\phi(n)$. After getting d it is very easy to get the plaintext by calculating C^d mod n. When I converted the plaintext bits I obtained to string I get the message:

> "I am free. Every single thing that I've done I decided to do. My actions are governed by nothing but my own free will. Do you wanna know what I hate more than everything else in this world? Anyone who isn't free."

Question 3

(a) Nonlinearity Degree

We want our combining function to be highly nonlinear (Nonlinearity of a function is given as the maximum of the order of the terms in function's algebraic normal form). In our case the term $x_1 \cdot x_2 \cdot x_3 \cdot x_4$ has the highest order so our combining function has nonlinearity degree of 4 and 4 is the maximum degree we can achieve since it combines the outputs of 4 LFSRs. Our function has high nonlinearity degree which is desirable.

(b) Balance

x_1	x_2	x_3	x_4	$z = F(x_1, x_2, x_3, x_4)$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

If we look at table constructed to assess all the possible combinations of x_1, x_2, x_3 and x_4, we can see that the output of the combining function is mostly 0. We have 11 0's and only 5 1's. Since the number of possible 0 and 1 values of the combining function is not close to each other, we can say that it is unbalanced.

(c) Correlation

To check the correlations, we need to compare each output of the LFSR's with the output of the combining function. When we count the identical values we will get the following correlations.

LFSR	Correlation
x_1	$\frac{5}{16} = 31.25\%$
x_2	$\frac{9}{16} = 56.25\%$
x_3	$\frac{7}{16} = 43.75\%$
x_4	$\frac{13}{16} = 81.25\%$

The output of the combining function has high correlation with x_4 and moderate correlation with x_2. This high correlations are undesired for a combining function.

Overall, this combining function is not good enough due to its unbalance and high correlations. This high correlations make it open for Correlation Attacks which drastically reduces the effort to break the encryption.

Question 4

Since we have a small N here, we can try to factorize N to obtain p and q. To do this, I wrote a simple loop that starts with 11 and goes to square root of N with a step size of 2. This will ensure that I skip all the numbers that are divisible by 2. For other small primes (3, 5, 7), I also checked the divisibility since this calculations are much easier. In other word I checked every number until the square root of N if the number is not divisible by 2, 3, 5, 7. For every number that satisfies this conditions, I checked if it divides N. When I found this number, I immediately stop the loop since I know that $N = p \cdot q$. So after finding the first root p, the second root q is basically N/p.

After that calculating $\phi(n)$ is easy since $\phi(N) = (p-1) \cdot (q-1)$. Then I calculated the d as e^{-1} mod $\phi(N)$. After getting d it is very easy to get the plaintext by calculating C^d mod N. When I converted the plaintext bits I obtained to string I get the message: "Aloha!"

Question 5

In the given assignment, the objective was to perform arithmetic operations within the Galois Field GF(2^8) using the specified irreducible polynomial '111000011'. The server provided two binary polynomials: '10010100' for a(x) and '11100000' for b(x).

To achieve this, I used the BitVector module in Python. In the first part of the task, I performed the multiplication of a(x) and b(x) within GF(2^8), sticking to the irreducible polynomial. The resulting polynomial c(x) was determined as '11011110'.

Moving on to the second part, the focus was on computing the multiplicative inverse of a(x) within GF(2^8), again using the same irreducible polynomial as the modulus. The outcome of this calculation was '00011011'. Both computed results were validated with a "Congrats" message, confirming the accuracy of the solution.

Question 6

So, let's break down what we did with those modular multiplications using the Chinese Remainder Theorem (CRT).

First, we got these three values $a_1, b_1, a_2, b_2, a_3, b_3$. Also, we have these three modulus $q_1, q_2, q3$. These are the steps we need to perform:

1. Compute the combined modulus:
   $N = q_1 \times q_2 \times q_3$

2. Compute partial modulus:
   $N_1 = N/q_1 = q_2 \times q_3$
   $N_2 = N/q_2 = q_1 \times q_3$
   $N_3 = N/q_3 = q_1 \times q_2$

3. Compute modular inverses:
   $M_1 = N_1^{-1} \mod q_1$
   $M_2 = N_2^{-1} \mod q_2$
   $M_3 = N_3^{-1} \mod q_3$

4. Combine results using CRT:
   $x = (a_1 \cdot b_1 \cdot N_1 \cdot M_1 + a_2 \cdot b_2 \cdot N_2 \cdot M_2 + a_3 \cdot b_3 \cdot N_3 \cdot M_3) \mod N$

After performing these steps I obtain R as: 1753151627924204850439611 2056

1. Original Results r1, r2, r3:
   $a_1 \times b_1 \mod q1 = 1643182479$
   $a_2 \times b_2 \mod q2 = 363289399$
   $a_3 \times b_3 \mod q3 = 2376063578$

2. Reconstruct r1, r2, r3 from R:
   $R \mod q_1 = 1643182479$
   $R \mod q_2 = 363289399$
   $R \mod q_3 = 2376063578$