CS 411 - Homework 3

Ege Demirci - 28287

Sabanci University, Fall 2023-2024, Cryptography

1. Question - 1

In this RSA decryption task, we are given the parameters N, C, and e, with the condition that the plaintext M is much smaller than N (denoted as $M \ll N$) and is a 128-bit number.

Normally RSA typically involves modular exponentiation, but in this case, due to the small size of M relative to N and the small exponent $e = 2^4 + 1 = 17$, a direct extraction approach can be used. Since M is significantly smaller than N, the result of C^e would still be smaller than N, allowing us to avoid the modular reduction part of RSA decryption.

To solve this, I used a method to find the e-th root of C. I used a binary search algorithm for finding the e-th root, as it's efficient and precise for this purpose. This method involves iteratively narrowing down the range in which the e-th root of C lies, by comparing midpoints raised to the power of e with C.

Once I found the e-th root of C, I treated this value as my plaintext M. To confirm that my solution was correct, I checked that the bit length of M was 128 bits, aligning with the problem's specifications.

Here is the m:

340282366920938463463374607431768211455

And bit length : 128

2. Question - 2

In this problem, we are tasked with decrypting a message encrypted using RSA, The provided parameters were n, e, cp, and cq.

a. Explanation of Insecurity: The insecurity arises because cp and cq are directly related to the factors of n. Anyone obtaining cp or cq can essentially recover p and q, which are supposed to be private. Since RSA's security relies on the difficulty of factoring the product of two large primes (in this case, $n = pq$), knowing p and q directly compromises the security of the system. If an attacker obtains either cp or cq, they can factor n. This is because cp and cq are effectively the RSA encryptions of kp and kq, respectively. In other words, since RSA is vulnerable when the plaintext (in this case, kp or kq) is close to the modulus n, an attacker can use cp or cq to easily factor n by computing the greatest common divisor (GCD) of n and cp (or cq). This vulnerability arises from the fact that the blinding factor k does not sufficiently obscure the relationship between the modulus n and its factors.

To factor n, I use the Extended Euclidean Algorithm (EEA). The EEA finds the greatest common divisor (GCD) of two numbers and also the coefficients of Bézout's identity, which are the integers x and y such that $ax + by = \gcd(a, b)$.

Here, I use EEA to find the GCD of n and cp, and similarly, n and cq. Since cp and cq are derived from p and q respectively, the GCDs found in this way are actually p and q.

After obtaining p and q, I calculated Euler's totient function $\phi(n)$ as $(p - 1) \times (q - 1)$. The next step was to find the modular multiplicative inverse of e modulo $\phi(n)$ to get the decryption exponent d. This was done using the 'modinv' function, which also uses the Extended Euclidean Algorithm.

With the decryption exponent d in hand, I decrypted the ciphertext cm by computing $cm^d \bmod n$. To ensure the plaintext was correctly formatted, I converted the decrypted number into bytes and then decoded it from UTF-8. At this part, the output I get was corrupted, and I noticed that it is reversed; so I reversed the string to get the message.

The secret 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.

3. Question - 3

In my analysis of the combining function $F(x_1, x_2, x_3, x_4)$, I evaluated it based on nonlinearity degree, balance, and correlation.

1. Nonlinearity Degree: The nonlinearity degree of a Boolean function is determined by the highest degree of any term in its Algebraic Normal Form. In F, the term with the highest degree is $x_1x_2x_3x_4$, a product of four variables. Therefore, the nonlinearity degree of F is 4.

2. Balance: To assess the balance of F, I constructed the following table of its output for all possible input combinations:

x_1	x_2	x_3	x_4	$F(x_1, x_2, x_3, x_4)$
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

From the table, it's evident that F is not balanced, as it produces a significantly unequal number of 0s and 1s. There are five 1s, and eleven 0s.

3. Correlation: For the correlation analysis part, I compared F with each possible linear function of its inputs.

x_1: 5/16

x_2: 9/16

x_3: 7/16

x_4: 13/16

From the numbers we get, we can easily say that the output is correlated to X4 and X2. X2 is bigger than 1/2 so it is correlated with the output, X4 is much bigger so it's highly correlated to output as well.

4. Result:

Based on the analysis we made, the combining function $F(x_1, x_2, x_3, x_4)$ is not a good combining function for cryptographic purposes, mainly due to its lack of balance and high correlation.

The nonlinearity degree of 4 is a positive aspect, as it suggests resistance to linear attacks. However, the balance of a Boolean function is crucial in cryptographic applications.In the case of F, the output is heavily skewed towards 0s, which can make it vulnerable to certain types of cryptanalytic attacks due to its predictability.

Furthermore, the high correlation with x_4 (12 out of 16 times) and x_2 (9 out of 16 times) is a significant drawback. A high correlation with one of the inputs, as seen with x_4 and x_2, makes the system more predictable and susceptible to such attacks.

In summary, the lack of balance and high correlation with an input variable mean that F is not an ideal combining function for secure cryptographic systems. A good combining function should exhibit high nonlinearity, be balanced, and have low correlation with all linear functions of its inputs.

4. Question - 4

The RSA encryption is defined as $m^e \mod N$ and decryption as $C^d \mod N$, where m is the plaintext, e and d are the public and private exponents, respectively, and N is the modulus.

To decrypt C without the private key, we first needed to factorize the modulus N into its prime factors, but RSA's security relies on the difficulty of factoring large numbers. I wrote a function factorize(n) to achieve this. This function iteratively divides N by 2 (to remove all factors of 2) and then checks for other factors starting from 3, incrementing by 2 each time (to check only odd numbers, as even numbers other than 2 cannot be prime).

Once N is factorized into its prime factors p and q, I calculated the totient of N, denoted as $\phi(N) = (p-1)(q-1)$. The totient is used to find the private exponent d, which is the modular multiplicative inverse of e modulo $\phi(N)$. I used the pow() function with a third argument to calculate this modular inverse.

With d in hand, I then decrypted the ciphertext C using the formula $M = C^d \mod N$, where M is the plaintext message. I converted this number back to a string to retrieve the original message.

Message is: Aloha!

5. Question - 5

In the given task, we were required to conduct arithmetic operations within the Galois Field GF(2^8), utilizing the specific irreducible polynomial 111000011. The server provided me two binary polynomials: a(x) as '10001101' and b(x) as '11000110'.

I employed the BitVector module in Python for these operations. For the first part of the task, I multiplied a(x) and b(x) within GF(2^8), using the irreducible polynomial. The result of this multiplication was '10001100'.

In the second part, I focused on finding the multiplicative inverse of a(x) in GF(2^8), again using the same irreducible polynomial as the modulus. The result of this computation was '11000111'. Both results gave me a "Congrats" message, confirming the accuracy of my solution.

6. Question - 6

In this question, our goal was to perform modular multiplications for three sets of values and then reconstruct the results from a single multiplication modulo a large number Q.Firstly, we define our variables a1, b1, q1, a2, b2, q2, a3, b3, q3, and calculate the individual modular multiplications r1, r2, and r3 for each set of values. The next step involves calculating the combined modulus Q, which is the product of all individual moduli (q1, q2, q3). This combined modulus is used in the CRT. To apply the CRT, we calculate n1, n2, n3, which are Q divided by each individual modulus. We then find the modular inverses m1, m2, and m3 of these n-values with respect to their respective moduli.

The overall result R is computed as the sum of the products of each individual result (r1, r2, r3), their respective modular inverses (m1, m2, m3), and the n-values (n1, n2, n3), all modulo Q. Finally, to reconstruct the individual results (r1, r2, r3) from R, we simply take R modulo each individual modulus (q1, q2, q3). My output confirms that the reconstructed results match the original modular multiplications, validating our approach and the correct application of the Chinese Remainder Theorem in this context.

Here is the output for r1, r2, r3, R, (reconstructedr1, reconstructedr2, reconstructedr3) as you see r - reconstructed duos are same.

(1643182479, 363289399, 2376063578, 17531516279242048504396112056, (1643182479, 363289399, 2376063578))