CS 411 - Homework 4

Ege Demirci - 28287

Sabanci University, Fall 2023-2024, Cryptography

1. Question - 1

In the given question, our aim is to decipher the original plaintext m from a given ciphertext C, using the public key components e and N, without directly querying the oracle with C.

- Let $choP$ represent a randomly chosen plaintext value that I select. This value must be coprime to N and not equal to the plaintext corresponding to the provided ciphertext C. In our case, I select $choP = 7$.
- I then compute $choC$, which is the ciphertext obtained by encrypting $choP$ with the public key exponent e, as $choC \equiv choP^e \mod N$.
- The oracle is queried with a new ciphertext $sendingC$, which is the product of the given ciphertext C and $choC$, modulo N: $sendingC \equiv choC \cdot C \mod N$.
- Upon querying the oracle with $sendingC$, I receive $sendingM$, which is the plaintext corresponding to $sendingC$, decrypted by the oracle's private key d.

By the RSA encryption and decryption mechanism, the relationship between a plaintext x and its ciphertext y under a public key exponent e and a modulus N is given by $y \equiv x^e \mod N$. The decryption process under the private key exponent d is $x \equiv y^d \mod N$.

Utilizing the property that $ed \mod \phi(N) \equiv 1$, where $\phi(N)$ is the Euler's totient function of N, and ed is the product of the public and private exponents, we can derive that for any integer k:

$$(x^e)^d \mod N \equiv x$$

We will apply this to our chosen plaintext $choP$, but firstly we encrypt it to $choC$ and construct $sendingC$ such that:

$$sendingC \equiv choC \cdot C \mod N \equiv choP^e \cdot C \mod N$$

When the oracle decrypts $sendingC$, it essentially calculates:

$$sendingM \equiv sendingC^d \mod N \equiv (choP^e \cdot C)^d \mod N \equiv choP^{ed} \cdot C^d \mod N \equiv choP \cdot m \mod N$$

The last equality holds because $choP^{ed} \mod N \equiv choP$, and $C^d \mod N \equiv m$, where m is the original plaintext.

To isolate m, we use the modular inverse of $choP$ with respect to N:

$$m \equiv sendingM \cdot choP^{-1} \mod N$$

This equation allows us to calculate the value of m as:

m = 15605313849468876426166396992325579145606080775874599962766790634109889995826446565318677628625440227891034220 0

The numerical value of m is then converted into bytes and subsequently decoded into a human-readable Unicode string. The integrity and correctness of the obtained message are confirmed by the RSA Oracle Checker:

Message is: Bravo! You found it. Your secret code is 71848

2. Question - 2

Given an RSA encrypted ciphertext c, public key exponent e, and modulus N, the task is to decrypt and find the original four-digit PIN that has been encrypted using RSA OAEP (Optimal Asymmetric Encryption Padding) with an 8-bit random number R.

- The ciphertext is given as:

  $c = 15563317436145196345966012870951355467518223110264667537181074973436065350566$

- The public key exponent is $e = 65537$.
- The modulus is:

  $N = 73420032891236901695050447655500861343824713605141822866885089621205131680183$

- The random number R used in OAEP is an 8-bit unsigned integer, implying that R ranges from 0 to 255.
- The PIN is a four-decimal digit number, meaning it ranges from 1000 to 9999.

To find the correct PIN, I implemented a brute-force attack, trying every possible four-digit PIN and every possible 8-bit random number R until the correct combination was found that would produce the given ciphertext c when encrypted with the provided public key e and modulus N. We were able to use brute force here because the 1000-9999 range which includes 9000 values, and there are 256 R values. So the key space is 256 * 9000. And this key space is not big in cryptographic terms, all values can be tried easily.

1. The function `RSA_OAEP_Enc` performs the encryption process given a message m, the public key e, the modulus N, and the random number R.
2. The range of possible PINs is iterated over using `possiblePins = range(1000,10000)`.
3. For each possible PIN, the range of possible 8-bit random numbers is also iterated over.
4. The encryption function `RSA_OAEP_Enc` is called with each combination of PIN and R until the resulting ciphertext matches the given c.
5. When a match is found, the loop breaks, and the correct PIN is printed.

The brute-force method successfully identified the original PIN as 1308, which when encrypted with the corresponding random number R (which is 206 in this case) and the given public key parameters, results in the provided ciphertext c.

3. Question - 3

In this question, we will a message encrypted using the ElGamal encryption algorithm with a known flaw. The parameters used for the encryption, including the primes q, p, the base g, the public key h, and the ciphertext components r and t, are given.

- The large prime q is used as the order of the subgroup and is a prime divisor of $p - 1$.
- The prime p is the modulus.
- The base g is the generator of the subgroup of order q in the multiplicative group Z_p^*.
- The public key h is calculated as $g^s \mod p$, where s is the private key.
- The ciphertext is composed of two parts, r and t, where $r \equiv g^k \mod p$ and $t \equiv h^k \cdot m \mod p$ with k as the random nonce and m as the message.

The flaw in the given ElGamal implementation lies in the size of the random nonce k, which is an 8-bit unsigned integer. This limits k to a range of values from 1 to $2^{16} - 1$ and makes it feasible to perform an exhaustive search for the correct k.

To decrypt the message, the following steps were taken mathematically:

1. An exhaustive search was performed over the possible range of k, using the given g and p to compute $g^k \mod p$.
2. For each k, the computed value was compared against the given r.
3. When a match was found, indicating the correct k, the decryption process proceeded.
4. The message m was obtained using the formula $m \equiv t \cdot h^{-k} \mod p$, which can be rewritten using the modular inverse as $m \equiv t \cdot (h^k)^{-1} \mod p$.
5. The modular inverse of h^k modulo p was computed, and then multiplied by t to find the plaintext m.

The decryption function `Dec()` was used to account for the modular inverse of h raised to the power of $-k$. The correct k was determined through an exhaustive search in the range of potential nonce values.

The brute-force search successfully found the correct k (which is 31659 in this case), and the message was decrypted accordingly, yielding the numerical value and the decoded English text:

```
Numeric value of message:
2379326593814966777427883837048844070177827803620011347711969580807839402328800
3107350836517401044329057774
Decoded message: Be yourself, everyone else is already taken.
```

4. Question - 4

In this question, we should recover the second message m_2 encrypted using the ElGamal encryption algorithm with the same random nonce k. The ciphertext components for the first message (r_1, t_1) and the second message (r_2, t_2) are given, alongside the encryption parameters q, p, and g.

- Prime q which is a prime divisor of $p - 1$.
- Prime p serves as the modulus for the group.
- Generator g of the subgroup of order q.
- Ciphertext components for the first message m_1: r_1 and t_1.
- Ciphertext components for the second message m_2: r_2 and t_2.
- Both messages are encrypted with the same random nonce k, leading to $r_1 = r_2$ (which is **flaw** in this case).

Since the same nonce k was used for both messages, we can use the following properties of ElGamal encryption:

$$
\begin{aligned}
r &\equiv g^k \mod p \\
t_1 &\equiv h^k \cdot m_1 \mod p \\
t_2 &\equiv h^k \cdot m_2 \mod p
\end{aligned}
$$

Given that $r_1 = r_2$, we deduce that $t_1/m_1 \equiv t_2/m_2$ modulo p, and thus $m_2 \equiv (t_2 \cdot m_1 \cdot t_1^{-1}) \mod p$, where m_1 is the m_1, and t_1^{-1} is the modular inverse of t_1 modulo p.

The decryption process using the provided ciphertext components. The modular inverse function `modinv()` is used to compute the necessary inverses modulo p. With the known m_1, we calculate $\beta_k \equiv (t_1 \cdot m_1^{-1}) \mod p$, which represents h^k modulo p. Finally, m_2 is recovered by computing $(t_2 \cdot \beta_k^{-1}) \mod p$.

The recovery process successfully yielded the second message m_2 in both its numeric and decoded byte object form, revealing the meaningful English text:

```
m2 is:
14649973832333132475064077137516748006344032866803958320445974163973125733490688627724929099176287195232595242637865095446532200276746858473691162084509026590614830
Message2 is: A person can change, at the moment when the person wishes to change.
```

5. Question - 5

The question was to recover the secret key a from two given message signatures using the DSA scheme. The public parameters (q, p, g) and the public key (β) are known. We have two sets of message-signature pairs and the information that $k_2 \equiv 3k_1 \mod q$.

- Prime q and p are the public parameters of the DSA scheme.
- Generator g is used for generating the public key.
- Public key β is known.
- Two messages m_1 and m_2 with their respective signatures (r_1, s_1) and (r_2, s_2) are provided.

The DSA signature for a message m is a pair (r, s) where:

$$r \equiv (g^k \mod p) \mod q$$
$$s \equiv (k^{-1}(H(m) + a \cdot r)) \mod q$$

$H(m)$ is the hash of the message (which is calculated with SHAKE128 in this case), a is the secret key, and k is a nonce.

Given that $k_2 \equiv 3k_1 \mod q$, we can derive a relationship between the two signatures. The s components can be expanded as:

$$s_1 \equiv (k_1^{-1}(H(m_1) + a \cdot r_1)) \mod q$$
$$s_2 \equiv (k_2^{-1}(H(m_2) + a \cdot r_2)) \mod q$$

Using the nonce relationship, we can express k_2 in terms of k_1.

To recover the secret key a, we exploit the relationship between the nonces used for the two signatures, where $k_2 = 3k_1 \mod q$. The signatures are given by:

From these equations, we derive the values of k_1 and k_2:

$$k_1 \equiv s_1^{-1}(H(m_1) + a \cdot r_1) \mod q,$$
$$k_2 \equiv s_2^{-1}(H(m_2) + a \cdot r_2) \mod q.$$

Substituting k_2 with $3k_1 \mod q$ and rearranging the terms, we get:

$$3s_1^{-1} \cdot (H(m_1) + a \cdot r_1) \equiv s_2^{-1} \cdot (H(m_2) + a \cdot r_2) \mod q.$$

Multiplying both sides by s_1 to eliminate s_1^{-1} and by s_2 to isolate terms involving a, we obtain:

$$3s_2 \cdot (H(m_1) + a \cdot r_1) \equiv s_1 \cdot (H(m_2) + a \cdot r_2) \mod q.$$

Now, we factor out a from both sides:

$$a \cdot (3s_2 \cdot r_1 - s_1 \cdot r_2) \equiv s_1 \cdot H(m_2) - 3s_2 \cdot H(m_1) \mod q.$$

We then isolate a:

$$a \equiv (s_1 \cdot H(m_2) - 3s_2 \cdot H(m_1)) \cdot (3s_2 \cdot r_1 - s_1 \cdot r_2)^{-1} \mod q.$$

Finally, we can solve for a by computing the modular inverse of $(3s_2 \cdot r_1 - s_1 \cdot r_2) \mod q$:

$$a \equiv (s_1 \cdot H(m_2) - 3s_2 \cdot H(m_1)) \cdot \mathrm{modinv}(3s_2 \cdot r_1 - s_1 \cdot r_2, q) \mod q.$$

This equation allows us to calculate the secret key a given the signatures and the messages. By using this operations in Python, the secret key a is calculated to be:

$$a = 224768882479056124130979539634536705233906181169471385891036522645$$

In this way using the equation given, we reached the secret key.