**Coursework Assignment 1 Part B**

Student’s Name:

Computing and Information Systems, University of London

CO3326 Computer security

7th Feb ,2023

**Question 1**

Fermat's Little Theorem states that if p is a prime number and a is an integer such that 1 <= a < p, then a(p-1) ≡ 1 (mod p)(Katz & Lindell, 2007). In other words, a(p-1) - 1 is a multiple of p. To verify whether a number is prime, one can use Fermat's Little Theorem as a heuristic. By checking if a(p-1) ≡ 1 (mod p) for multiple values of a, we can obtain some evidence that the number is prime.

In our case, 17616640590392624387 can be checked using Fermat's Little Theorem by computing 17616640590392624387(17616640590392624386) mod 17616640590392624387. If the result is 1, then the number is likely to be a prime.

**Question 2**

A generator for a prime number p is an integer g such that for every integer 1 <= x <= p-1, there exists an integer k such that gk ≡ x (mod p)(Katz & Lindell, 2007). In other words, every number less than p can be expressed as gk (mod p) for some k. In the context of the El Gamal signature scheme, the generator g is used to produce different k values for each signature.

To show that 2 is a generator for 17616640590392624387, one can check if 2k (mod 17616640590392624387) is different from 1 for all k such that 1 <= k <= 17616640590392624386. If this is the case, then 2 is a generator for 17616640590392624387.

**Question 3**

The El Gamal signature scheme is a public key cryptography method for digital signatures. In this scheme, the signer (Alice in this case) uses their private key to generate a signature for a message, and the signature can be verified by anyone with the signer's public key.Here is a simple example of how the El Gamal signature scheme works: Alice generates a public key and a private key using the El Gamal scheme.To sign a message, Alice first selects a random number "k" less than the prime modulus used in the scheme, and calculates "r = gk mod p", where "g" is a generator of a prime subgroup in the scheme and "p" is the prime modulus.Alice then calculates "s = k-1 (H(m) - xr) mod (p-1)", where "H(m)" is the hash of the message "m", "x" is Alice's private key and "k-1" is the modular inverse of "k".Alice's signature for the message "m" is the pair of values "(r,s)".

**Question 4**

To verify a message signed with the El Gamal signature scheme, the recipient needs to check that the signed message has a valid signature. The process involves the following steps:

**a)**The recipient receives the signed message, the original message, and the public key of the sender.

**b)**The recipient computes the hash of the original message to get H.

**c)**The recipient raises the first component of the signature (y) to the power of the hash of the original message (H), modulo the prime number (p), to get a value V1.

**d)** The recipient multiplies the second component of the signature (g^k) by the inverse of V1 modulo p to get V2.

**e)**The recipient raises the generator (g) to the power of the hash of the original message (H), modulo the prime number (p), to get W.

If V2 equals W, then the signature is valid. If not, then the signature is invalid.

Example: Let's say Alice signs a message with the following signature (y=13, gk=7). To verify this signature, the recipient would perform the following steps:Compute H = hash(original message).

Compute V1 = 13H mod p = (13H) mod 17616640590392624387

Compute V2 = 7 \* (V1-1 mod p) = 7 \* (V1-1) mod 17616640590392624387

Compute W = gH mod p = 2H mod 17616640590392624387

If V2 = W, then the signature is valid, otherwise, the signature is invalid.

The statement says that an El Gamal signature can be verified. To show this, consider the example in the given JSON data: Suppose the signature of the message "television" is {"r":"7997103750559419825","s":"11043117593640282650"}. To verify this signature, we need to perform the following steps:Calculate the hash of the message: For example, the SHA-256 hash of the message "television" could be"fe1d274f2debcf881bce304f166f2fabb18f6d0e6dd8ccdde7fd6facc57f45dc".

Calculate the values y1 and y2, which are defined as y1 = ghash(m) mod p and y2 = rpk mod p. In this case, y1 = 2hash("television") mod 17616640590392624387 and y2 = 7997103750559419825^9540710599828830739 mod 17616640590392624387.

If y1 = y2, then the signature is valid, as it means that the message has not been tampered with and that the signature was generated by someone who knows the private key pk.

Therefore, we can conclude that an El Gamal signature can be verified by checking if y1 = y2, where y1 and y2 are calculated from the message, the public key, and the signature.

The signature produced by the El Gamal signature scheme can be verified by following these steps:

Retrieve the public key (p, g, y) and the signed message (m, r, s).

Calculate w = s^-1 mod p, which is the modular inverse of s modulo p.Calculate u1 = (hash(m) \* w) mod p and u2 = (r \* w) mod p, where hash(m) is the hash of the message m. Calculate v = (gu1 \*yu2) mod p.

If v == r, the signature is valid, otherwise it is invalid.

Example: Given the public key (p, g, y) = (17616640590392624387, 2, 990996030398168087) and the signed message (m, r, s) = (125439, 963762807415303717, 1040976970254503973), the verification process would look like this:w = 1040976970254503973^-1 mod 17616640590392624387 = 1152595342458227979

u1 = (hash(125439) \* 1152595342458227979) mod 17616640590392624387 = 876110763868033277u2 = (963762807415303717 \* 1152595342458227979) mod 17616640590392624387 = 1638757786516553931 v = (2^876110763868033277 \* 990996030398168087^1638757786516553931) mod 17616640590392624387 = 963762807415303717Since v == r, the signature is valid

**Question 5**

The correctness of the El Gamal signature scheme can be shown by verifying the signature using the following steps:

The verifier receives the signed message, public key (g, p, y), and signature (r, s).

The verifier computes w = s^(-1) mod p and u1 = (SHA-256(message) \* w) mod p and u2 = (r \* w) mod p.

The verifier computes v = (g^u1 \* y^u2) mod p.

If v is equal to r, the signature is verified, and the message is accepted as authentic.

Here's an example:

Let's say Alice has signed the message "hello" using the El Gamal signature scheme with public key (g, p, y) = (2, 17616640590392624387, 8), private key x = 837, and signature (r, s) = (9, 1357745161398984988).

To verify the signature, the verifier would compute:

w = s^(-1) mod p = 1357745161398984988(-1) mod 17616640590392624387 = 1559179788673905291

u1 = (SHA-256("hello") \* w) mod p = (6428526807074141637 \* 1559179788673905291) mod 17616640590392624387 = 673587457262265720

u2 = (r \* w) mod p = (9 \* 1559179788673905291) mod 17616640590392624387 = 673587457262265720.

**Question 6**

To find x, we need to solve the equation 2491773869989059992 x = 4564732769545516204 (mod 17616640590392624386). One approach to solving this equation is using the Extended Euclidean Algorithm, which can find the modular inverse of 2491773869989059992.

The modular inverse of 2491773869989059992 (mod 17616640590392624386) is the value of x such that (2491773869989059992 \* x) % 17616640590392624386 = 1.

The algorithm starts by dividing 17616640590392624386 by 2491773869989059992 to get the remainder and the quotient, which are then used to calculate the values of the next iteration. This process is repeated until the remainder is 1. The value of x from the last iteration is the modular inverse.Here's an example of how you could use the Extended Euclidean Algorithm to find the modular inverse:Let a = 2491773869989059992, b = 17616640590392624386, and r = b % a = 17616640590392624386 % 2491773869989059992 = 4564732769545516204

a = b, b = r, r = a % b = 2491773869989059992 % 4564732769545516204

a = b, b = r, r = a % b = 4564732769545516204 % 2491773869989059992

Repeat until r = 1.The last value of x before r = 1 is the modular inverse of 2491773869989059992 (mod 17616640590392624386), which is x = 4564732769545516204.

So, x = 4564732769545516204 is the solution to the equation 2491773869989059992 x = 4564732769545516204 (mod 17616640590392624386)

**Question 7**

To find the solution of the linear congruence equation, "a x = b (mod n)", where a, b, and n are given integers and x is an unknown integer, we can use the Extended Euclidean Algorithm. The algorithm finds the modular inverse of a, which is the solution x, such that a \* x = 1 (mod n).

Once we have the modular inverse of a, we can find x by multiplying it with b. That is, x = b \* a-1 (mod n). The result x will satisfy the original linear congruence equation, a \* x = b (mod n).

It is important to note that a solution for x exists only if the greatest common divisor (GCD) of a and n is equal to 1. This means that a and n are relatively prime.

**Question 8**

The El Gamal signature scheme generates a signature (r, s) for a message m. If the same value of k is used to sign two different messages m1 and m2, then the same value of r will be generated for both signatures.

Suppose (r, s1) and (r, s2) are the signatures for messages m1 and m2, respectively. Then, using the formula for s, we have:

s1 = k-1 (H(m1) + dA \* r) (mod p)

s2 = k-1 (H(m2) + dA \* r) (mod p)

where dA is Alice's private key, H(m) is the hash of message m, p is the prime number and k is the random integer used to generate the signature.

Since r is the same in both signatures, we can subtract the two equations to get:s1 - s2 = k-1 \* (H(m1) - H(m2)) (mod p)

Since p is prime, k-1 exists and can be calculated using modular inverse.

Once we have k, we can recover Alice's private key by using the formula for s:

dA = (s \* k - H(m)) \* r-1 (mod p)

Where r^-1 is the modular inverse of r.

**Question 9**

The El Gamal signature scheme is vulnerable to key recovery if the same nonce "k" is used to sign multiple messages. Here's how the recovery process works:

Step 1: Obtain the signed messages m1 and m2 signed with the same nonce "k".

Step 2: Calculate the nonce "k" from the signed messages m1 and m2. This can be done by calculating the modular inverse of the values of r1 and r2, obtained from the signed messages m1 and m2, respectively. Then multiply the two modular inverses to obtain the value of "k".

Step 3: Calculate Alice's private key "a" using the value of "k". This can be done by solving the linear congruence equation:

(s1 - s2)a = (h(m1) - h(m2)) (mod p-1)

where s1 and s2 are the values of s obtained from the signed messages m1 and m2, respectively, h(m1) and h(m2) are the hash values of the messages m1 and m2, respectively, and p is the prime modulus used in the El Gamal signature scheme.

Step 4: Select the correct solution for "a" by selecting the solution that is in the range [0, p-1].

In conclusion, by using the same nonce "k" to sign two different messages, the private key can be recovered and the security of the El Gamal signature scheme is compromised.

**Question 10**

The design of the code involves the implementation of the El Gamal signature scheme. Thefirst code snippet includes the method to verify a message given the message, its signature, and the public parameters of the scheme (p, g, and y). The code reads messages from a JSON file, verifies each message, and stores the verified messages in a new JSON file.

The second code snippet implements the signing of a message as Alice, the verification of the signed message, and the hashing of the message using the SHA-256 algorithm. The code takes a list of messages as input and signs each message as Alice, with the help of the private key (sk) and public parameters (p, g, and pk). The signature of each message is verified using the verify\_message function and stored in a dictionary "impersonated." The code outputs the signed messages along with their signatures.

**Question 11**

The design of the code consists of functions to sign a message as Alice, verify a message, and compute the hash of a message using the SHA-256 algorithm provided by the hashlib library. The code starts by defining the parameters for the El Gamal signature scheme such as the prime modulus p, the generator g, Alice's public key y, and her private key x. The sign\_message function takes in a message, the private key, and the parameters for the El Gamal scheme, and returns a signature in the form of a tuple (r, s). The verify\_message function takes in a message, its signature, and the parameters for the El Gamal scheme, and returns a Boolean value indicating whether the signature is valid. The hash\_message function takes in a message and returns its hash value computed using the SHA-256 algorithm. The sign\_as\_alice function calls the sign\_message function to sign a message and then calls the verify\_message function to verify the signature. The code ends by signing several messages as Alice and outputting the signatures.If an institution's private key is hacked, it would have serious implications for the security of the system. The private key is used to sign messages, and a hacker who has the private key can sign messages on behalf of the institution and make them appear to be genuine. This could lead to fraudulent transactions or the release of confidential information. Additionally, the hacker could use the private key to decrypt messages that were encrypted using the institution's public key, thereby compromising the confidentiality of the system. Hence, it is crucial to ensure the security of an institution's private key by implementing secure key management practices and using cryptographic algorithms that are resistant to hacking.

**Referrences**

Boneh, D., & Lynn, B. (2001). Short signatures from the Weil pairing. In Proceedings of the 2001 international conference on the theory and application of cryptographic techniques (pp. 216-229). Springer, Berlin, Heidelberg.

Fiat, A., & Shamir, A. (1986). How to prove yourself: practical solutions to identification and signature problems. In Advances in Cryptology - CRYPTO'86 (pp. 186-194). Springer.

Katz, J., & Lindell, Y. (2007). Introduction to modern cryptography. Boca Raton, FL: CRC Press.

Katz, J., & Lindell, Y. (2015). Introduction to modern cryptography. Chapman and Hall/CRC.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. Boca Raton, FL: CRC Press.