**WQD7010**

**Network & Security (Sem 2, Session 2023/2024)**

**Tutorial 2**

**Asymmetric Encryption and Message Authentication**

1. Suppose we have a hashing algorithm that generates a 32-bit digest by fusing together two distinct 16-bit sub-functions: XOR and RXOR. These sub-functions are described in lecture slides as "two simple hash functions."
2. Imagine a scenario where an attacker tampers with the input data, flipping an odd number of bits. Evaluate the effectiveness of this hashing algorithm in detecting such tampering attempts. Provide a rigorous justification for your conclusion, considering the properties of XOR and RXOR.
3. Now, let's explore the algorithm's behavior when an even number of bits are maliciously altered. Determine whether the hash function will consistently identify all such modifications. If there are specific tampering patterns that can evade detection, meticulously describe and characterize those scenarios.
4. From a security standpoint, assess the robustness of this 32-bit hashing algorithm when employed for authentication purposes. Analyze its resilience against potential attacks, considering factors such as collision resistance and preimage resistance. Provide a thoughtful commentary on whether this algorithm offers sufficient protection for verifying the integrity and authenticity of data. Support your analysis with well-reasoned arguments.

**Sol:**

a**. Yes. The XOR function is simply a vertical parity check. If there is an odd number of errors, then there must be at least one column that contains an odd number of errors, and the parity bit for that column will detect the error. Note that the RXOR function also catches all errors caused by an odd number of error bits. Each RXOR bit is a function of a unique "spiral" of bits in the block of data. If there is an odd number of errors, then there must be at least one spiral that contains an odd number of errors, and the parity bit for that spiral will detect the error.**

**b. No. The checksum will fail to detect an even number of errors when both the XOR and RXOR functions fail. In order for both to fail, the pattern of error bits must be at intersection points between parity spirals and parity columns such that there is an even number of error bits in each parity column and an even number of error bits in each spiral.**

**c. It is too simple to be used as a secure hash function; finding multiple messages with the same hash function would be too easy.**

1. Analyze the hash functions below that operate on input data represented as a series of base-10 integers, D = (m1, m2, ..., mt). For each hash function, determine which of the requirements for a cryptographic hash function listed in slides 54 (requirement of hash function) are satisfied. Explain your answers.
2. h = (Σ(i=1 to t) mi) mod n, for some predefined value n.
3. h = (Σ(i=1 to t) (mi)^2) mod n
4. Calculate the hash value from part (b) for the message D = (237, 632, 913, 423, 349) and n = 757.

**Sol:**

1. **The hash function h = (Σ(i=1 to t) mi) mod n satisfies properties 1-3 of a cryptographic hash function but not properties 4-6.**

**Properties satisfied:**

1. **The hash can be applied to a message of any size - Adding up the message elements handles arbitrary length**
2. **It produces a fixed-length output of size n**
3. **It is relatively easy to compute h(x) for any given message x**

**Properties not satisfied: 4. It is not one-way or preimage resistant. Given a hash h, a message x where h = H(x) can easily be found by setting x = h. 5. It is not second preimage resistant. For any message D, adding multiples of n to any element produces a modified message D' that collides with D. 6. It is not collision resistant. Any two messages (x1, x2, ...) and (x1+k*n, x2-k*n, ...) will collide for any integer k.**

1. **This hash function h = (Σ(i=1 to t) (mi)^2) mod n satisfies properties 1-4 if n is a large composite number, but not properties 5-6.**

**Properties satisfied:**

1. **Handles arbitrary message length**
2. **Produces fixed-length output of size n**
3. **Relatively easy to compute**
4. **One-way/preimage resistant if n is a large composite, as taking modular square roots is hard**

**Properties not satisfied: 5. Not second preimage resistant. Negating any message element produces a collision, since (-x)^2 = x^2. 6. Not collision resistant for same reason as above. x and -x collide.**

1. **For message D = (237, 632, 913, 423, 349) and n = 757: h = (237^2 + 632^2 + 913^2 + 423^2 + 349^2) mod 757 = (56169 + 399424 + 833569 + 178929 + 121801) mod 757 = 1589892 mod 757 = 192**

**So the hash value is 192.**

**In summary, the first hash is too simplistic and fails one-wayness and collision resistance. The second hash, if used with a large composite modulus n, achieves one-wayness but still fails collision resistance due to the square operation's symmetry. Neither is well-suited as a cryptographic hash. The key lessons are that cryptographic hash functions need to be carefully designed to achieve the necessary security properties and resist analytical attacks.**

1. In an RSA cryptosystem, you are given the following parameters:

p = 17, q = 19, and a specific integer used for encryption

1. Determine the integer utilized for decryption.
2. Enumerate the components of the public and private key pairs.

You need to encrypt the plaintext message, M = RSA.

1. Transform this message into its corresponding numerical representation based on the ASCII table.

To encrypt this message, you have the option to either amalgamate the numerical values or partition them into blocks of 2 digits (Hint: ensure that the condition M < n is satisfied).

1. Apply the encryption and decryption processes to the message RSA.
2. If the ciphertext's decimal representation (not its character equivalent) is 2190236, perform the decryption process on this value and transform the result into its corresponding message format according to the ASCII table.

**Sol:**

**Given RSA parameters:**

**p = 17, q = 19**

**a) Determine the integer utilized for decryption.**

**n = pq = 323, φ(n) = (p – 1)(q – 1) = 288**

**Find d and e so that de mod 288 = 1, find this number so that gcd of this number is d\*e.**

**Let's choose e = 5 (it satisfies the condition gcd(e, φ(n)) = 1)**

**de mod 288 = 1, so d = 173.**

**b) Enumerate the components of the public and private key pairs.**

**PU = {5, 323}, PR = {173, 323}**

**You need to encrypt the plaintext message, M = RSA.**

**c) Transform this message into its corresponding numerical representation based on the ASCII table.**

**To encrypt this message, you have the option to either amalgamate the numerical values or partition them into blocks of 2 digits (Hint: ensure that the condition M < n is satisfied).**

**M1 = R = 82, M2 = S = 83, M3 = A = 65**

**M = 828365**

**d) Apply the encryption and decryption processes to the message RSA.**

**To encrypt this message, we will partition it into blocks of 2 digits to satisfy the condition M < n, n = 323.**

**M1 = 82, 82 < 323**

**C1 = M1^e mod 323**

**82^5 mod 323**

**[82^4 mod 323 \* 82 mod 323] mod 323**

**[82^2 mod 323 \* 82^2 mod 323 \* 82 mod 323] mod 323**

**[256 \* 256 \* 82] mod 323**

**C1 = 309**

**-------------------------------------------------------------------**

**M2 = 83, 83 < 323**

**C2 = M2^5 mod 323**

**83^5 mod 323**

**[83^4 mod 323 \* 83 mod 323] mod 323**

**[83^2 mod 323 \* 83^2 mod 323 \* 83 mod 323] mod 323**

**[121 \* 121 \* 83] mod 323**

**C2 = 24**

**-------------------------------------------------------------------**

**M3 = 65, 65 < 323**

**C3 = M3^5 mod 323**

**65^5 mod 323**

**[65^4 mod 323 \* 65 mod 323] mod 323**

**[65^2 mod 323 \* 65^2 mod 323 \* 65 mod 323] mod 323**

**[196 \* 196 \* 65] mod 323**

**C3 = 84**

**To decrypt:**

**M1 = C1^d mod 323**

**= 309^173 mod 323 <<< similar step as above (breakdown into smaller set)**

**= 82**

**M1 = 82, which is ASCII for 'R'**

**--------------------------------------------------**

**M2 = C2^173 mod 323**

**= 24^173 mod 323 <<< similar step as above (breakdown into smaller set)**

**= 83**

**M2 = 83, which is ASCII for 'S'**

**--------------------------------------------------**

**M3 = C3^173 mod 323**

**= 84^173 mod 323 <<< similar step as above (breakdown into smaller set)**

**= 65**

**M3 = 65, which is ASCII for 'A'**

**e) If the ciphertext's decimal representation (not its character equivalent) is 2190236, perform the decryption process on this value and transform the result into its corresponding message format according to the ASCII table.**

**C1=219, C2=02, C3=36**

**M1 = C1^d mod 323**

**= 219^173 mod 323**

**= 68**

**M2 = C2^d mod 323**

**= 2^173 mod 323**

**= 79**

**M3 = C3^d mod 323**

**= 36^173 mod 323**

**= 71**

**M = 686779 = DOG**

1. Consider a Diffie–Hellman scheme with a common prime q = 353 and a primitive root a = 3.
2. If user A has public key YA = 40, what is A’s private key XA?
3. If user B has public key YB = 248, what is the shared secret key K?.
4. Eve intercepts the communication between User A and User B and chooses her own private key XE = 201. What is Eve's public key YE, and what shared secret key K\_BE will User B compute using Eve's public key YE, assuming User B's private key XB = 156?
5. What shared secret key K\_EB will Eve compute using User B's public key YB, and what shared secret key K\_AE will User A compute using Eve's public key YE?
6. After the man-in-the-middle attack, what are the shared secret keys between User A and Eve (K\_AE) and between User B and Eve (K\_BE)? Explain how Eve can intercept, decrypt, and read the messages sent between User A and User B without their knowledge.
7. If User A sends a message encrypted with K\_AE, describe the steps Eve can take to modify the message and send it to User B, making User B believe it came securely from User A. How does the lack of authentication in the Diffie-Hellman key exchange protocol make it vulnerable to man-in-the-middle attacks, and what additional security measures can be implemented to mitigate this risk?

**Sol:**

**Solution a and b:**

* **User A's private key XA can be found by solving the equation: YA = a^XA mod q 40 = 3^XA mod 353 Using the discrete logarithm, we find that XA = 97.**
* **The shared secret key K between User A and User B is calculated using: K = YB^XA mod q K = 248^97 mod 353 K = 160**

**Solution c:**

* **Eve's public key YE is calculated using: YE = a^XE mod q YE = 3^201 mod 353 YE = 13**
* **User B computes the shared secret key K\_BE using Eve's public key YE: K\_BE = YE^XB mod q K\_BE = 13^156 mod 353 K\_BE = 40**

**Solution d:**

* **Eve computes the shared secret key K\_EB using User B's public key YB: K\_EB = YB^XE mod q K\_EB = 248^201 mod 353 K\_EB = 40**
* **User A computes the shared secret key K\_AE using Eve's public key YE: K\_AE = YE^XA mod q K\_AE = 13^97 mod 353 K\_AE = 40**

**Solution e:**

* **After the man-in-the-middle attack, the shared secret key between User A and Eve (K\_AE) is 40, and the shared secret key between User B and Eve (K\_BE) is 40.**
* **Eve can intercept, decrypt, and read the messages sent between User A and User B by performing the following steps:**
  1. **Eve intercepts the public keys YA and YB sent between User A and User B.**
  2. **Eve generates her own private key XE and computes her public key YE.**
  3. **Eve sends her public key YE to User B, pretending to be User A, and to User A, pretending to be User B.**
  4. **Eve computes the shared secret keys K\_EB (with User B) and K\_AE (with User A) using the intercepted public keys and her own private key.**
  5. **When User A sends a message encrypted with K\_AE, Eve can decrypt it using K\_AE, read the message, encrypt it with K\_BE, and send it to User B.**
  6. **When User B sends a message encrypted with K\_BE, Eve can decrypt it using K\_BE, read the message, encrypt it with K\_AE, and send it to User A.**
  7. **User A and User B believe they are communicating securely with each other, but in reality, Eve can read and modify their messages.**

**Solution f:**

* **If User A sends a message encrypted with K\_AE, Eve can perform the following steps to modify the message and send it to User B:**
  1. **Eve intercepts the encrypted message from User A.**
  2. **Eve decrypts the message using the shared secret key K\_AE.**
  3. **Eve modifies the decrypted message as desired.**
  4. **Eve encrypts the modified message using the shared secret key K\_BE.**
  5. **Eve sends the encrypted modified message to User B.**
  6. **User B decrypts the message using K\_BE, believing it came securely from User A.**
* **The lack of authentication in the Diffie-Hellman key exchange protocol makes it vulnerable to man-in-the-middle attacks because:**
  1. **The protocol does not provide a way to verify the identities of the communicating parties.**
  2. **An attacker can intercept the public keys exchanged between the parties and replace them with their own public keys.**
  3. **The parties compute the shared secret keys based on the intercepted public keys, unknowingly establishing separate shared keys with the attacker.**
* **To mitigate the risk of man-in-the-middle attacks, additional security measures can be implemented, such as:**
  1. **Using authentication mechanisms like digital signatures or public key certificates to verify the identities of the communicating parties.**
  2. **Employing secure channels (e.g., SSL/TLS) to protect the exchange of public keys and prevent interception.**
  3. **Implementing protocols like Station-to-Station (STS) or Secure Remote Password (SRP) that combine Diffie-Hellman key exchange with authentication.**

1. Consider a variant of CMAC that XORs the second key K1 after applying the final block cipher encryption, rather than before. This variant can be expressed as: VMAC(K, M) = CBC(K, M) ⊕ K1 where the message M is an integer multiple of the block size.

Suppose an adversary can obtain the MACs for three chosen messages:

Message 0 = 0n (n is the cipher block size)

Message 1 = 1n

Message (1 || 0) which is message 1 concatenated with message 0

Let the corresponding MACs be:

T0 = CBC(K, 0) ⊕ K1

T1 = CBC(K, 1) ⊕ K1

T2 = CBC(K, [CBC(K, 1)]) ⊕ K1

Show that the adversary can now compute the correct MAC for a new message 0 || (T0 ⊕ T1) without making any additional queries to the MAC oracle.What modification you have to made from the original CMAC (Figure 1) diagram?

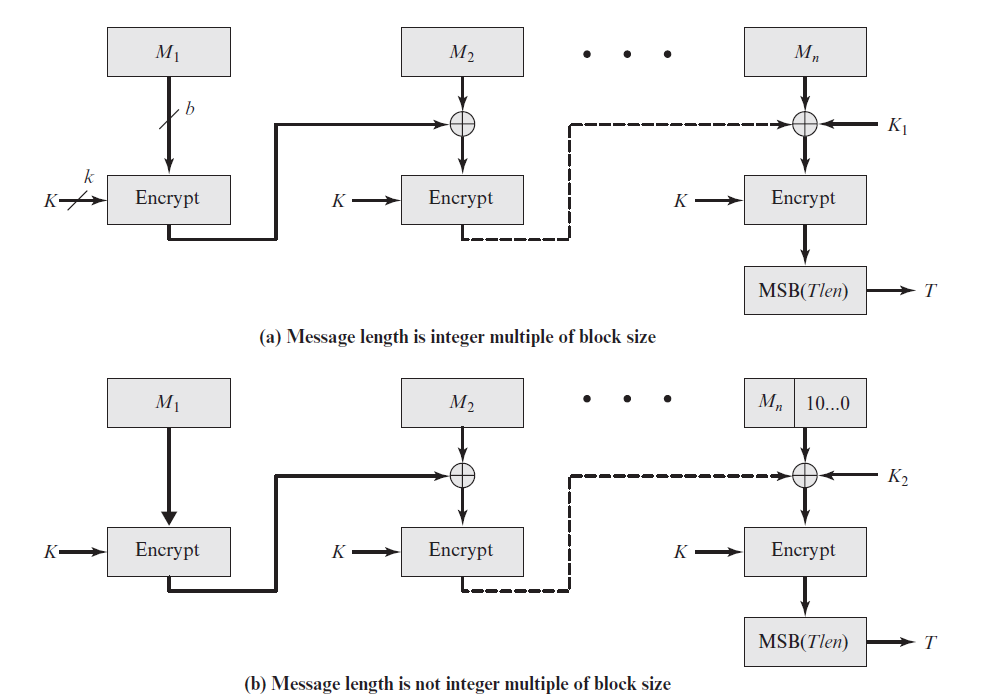


Figure 1: CMAC

**Sol:**

**The vulnerability arises because XORing the key after the final encryption allows an adversary to manipulate MACs in unintended ways.**

**From the given information:**

**CBC(K, 0) = T0 ⊕ K1**

**CBC(K, 1) = T1 ⊕ K1**

**CBC(K, [CBC(K, 1)]) = T2 ⊕ K1**

**For the target message 0 || (T0 ⊕ T1):**

**The first block encrypts to CBC(K, 0) = T0 ⊕ K1**

**This becomes the IV for the second block.**

**The second message block is T0 ⊕ T1**

**XORing this with the IV (T0 ⊕ K1) gives:**

**(T0 ⊕ K1) ⊕ (T0 ⊕ T1) = T1 ⊕ K1 = CBC(K, 1)**

**Encrypting this block yields:**

**CBC(K, [CBC(K, 1)]) = T2 ⊕ K1**

**Finally XORing K1 gives the MAC:**

**(T2 ⊕ K1) ⊕ K1 = T2**

**Therefore, the adversary can compute a valid MAC T2 for the message 0 || (T0 ⊕ T1) without knowledge of the key K. This chosen-message attack breaks the security of this CMAC variant.**

**The root issue is that XORing K1 after encryption enables exploiting the CBC structure to manipulate intermediate values and construct new valid MACs. The standard CMAC avoids this by XORing K1 before the final encryption.**

**The problem is based on the CMAC diagram, with the modification that the final XOR with K1 is moved after the encryption. This small change introduces a significant vulnerability that allows MAC forgeries as demonstrated in the solution.**