Problem 1 Midterm_111550076_楊捷棠

(a) $f(x)=x^2+1,\ g(x)=x^3+x^2+1$

① $f(x)+g(x) \Rightarrow \{0101\} \oplus \{1101\} = x^3$ #

② $f(x)-g(x) \Rightarrow \{0101\} \oplus \{1101\} = x^3$ #

③ $f(x)\times g(x) = (x^2+1)\cdot(x^3+x^2+1) = x^5+x^4+x^3+x^2+x+1$

$\{111111\} \oplus \{100110\} = \{011001\}$, $\{01001\} \oplus \{10011\} = \{11010\}$

$\Rightarrow \{0101010\} \Rightarrow x^3+x$ #

(b) ① $f(x)+g(x) \Rightarrow \{x^2+1\} \oplus \{x+1\} = x^2+x$ #

② $f(x)-g(x) \Rightarrow \{x^2+1\} \oplus \{x+1\} = x^2+x$ #

③ $f(x)\times g(x) = \{x^2+1\}\times\{x+1\} = x^3+x^2+x+1$

$x^3+x^2+x+1 \mod x^4+x+1 = \underline{x^3+x^2+x+1}$ #

Problem 2.

(a) ① $f(x)+g(x) = \{x^7+x^5+x^4+x+1\} \oplus \{x^3+x+1\}$

$= \underline{x^7+x^5+x^4+x^3}$ #

② $f(x)-g(x) = \{x^7+x^5+x^4+x+1\} \oplus \{x^3+x+1\} = \underline{x^7+x^5+x^4+x^3}$ #

$f(x)\times g(x) = (x^7+x^5+x^4+x+1)\cdot(x^3+x+1) = x^{10}+x^8+x^7+x^2+1$

$(x^{10}+x^8+x^7+x^2+1) \mod (x^8+x^4+x^3+x+1) = \underline{x^5+1}$ #

(b) $f(X) = X^4+1 = (X^2+1)(X^2+1)$ in $GF(2^3)$

$\Rightarrow X^4+1$ is reducible over $GF(2^3)$ #

Problem 3

(a) $f(X) f^{-1}(X) \equiv 1 \pmod{P(X)}$

$X f^{-1}(X) \equiv 1 \pmod{(X^4+X+1)}$

$X \cdot X = X^2 \bmod X^4+X+1 = X^2 \neq 1$

$X \cdot X^3 = X^4 \bmod X^4+X+1 = X+1 \neq 1$

$X(X^3+1) = X^4+X \bmod X^4+X+1 = 1$

$\Rightarrow f^{-1}(X) = \underline{X^3+1}$ #

(b) $g(X) g^{-1}(X) \equiv 1 \pmod{P(X)}$

$(X^2+X) g^{-1}(X) \equiv 1 \pmod{(X^4+X+1)}$

$(X^2+X) X^3 = (X^5+X^4) \bmod (X^4+X+1) = X^2+1 \neq 1$

$(X^2+X)(X^2+X+1) = (X^4+X) \bmod (X^3+X+1) = 1$

$\Rightarrow g^{-1}(X) = \underline{X^2+X+1}$ #

Problem 4

(a) $x^8 + x^4 + 1 = (x^3 + x^2)(x^5 + x^4 + x^3 + x^2) + 1$

$(x^3 + x^2)^{-1} = x^5 + x^4 + x^3 + x^2$

$(x^3 + x^2 + x)(x^5 + x^4 + x^3 + x^2) = x^8 + x^6 + x^5 + x^3$

$x^8 + x^6 + x^5 + x^3 \mod x^8 + x^4 + 1$

$= x^6 + x^5 + x^4 + x^3 + 1$ #

(b) $(x^6 + x^3 + 1)(x^6 + x^3 + 1) = x^{12} + x^9 + x^8 + x^7 + x^6 + x^4 + x^3 + 1$

$= x^{10} + x^9 + x^7 + x^4 + 1$

$x^{10} + x^9 + x^7 + x^4 + 1 \mod x^8 + x^4 + 1 = x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$ #

Problem5

(a)

Lookup Table Implementation

Precompute Multiplication in $GF(2^8)$:

- Precompute and store the results of all possible multiplications for all byte values in $GF(2^8)$. This will yield the following three tables:
  - **Multiply by 1**: The byte itself.
  - **Multiply by 2**: Perform a bitwise left shift and reduce modulo$x^8 + x^4 + x^3 + x + 1$.
  - **Multiply by 3**: Perform a bitwise left shift followed by an XOR with the original byte, and reduce modulo $x^8 + x^4 + x^3 + x + 1$.

XOR Operation:

- After looking up the precomputed table, combine the results using XOR, which is the addition operation in *GF*(2^8).

(b)

Byte substitution, shift rows, and mix columns can be combined and implemented using lookup tables and XOR operations. The steps are as follows:

First, it is essential to understand that the entire AES round function can be optimized using lookup tables and XOR operations:

1. **Byte Substitution**: Each byte in the state matrix is replaced with the corresponding S-box value from a lookup table.
2. **Shift Rows**: The rows of the state matrix are cyclically left-shifted. This is a simple permutation operation that can be implemented as part of the lookup table.
3. **Mix Columns**: Use precomputed lookup tables for multiplication in GF(2^8) by constants such as 2 and 3, and combine the results using XOR operations.

Thus, the implementation steps are as follows:

1. Create the S-box lookup table and the lookup tables for multiplication by constants like 2 and 3 in GF(2^8).
2. In the AES round function, each byte undergoes S-box substitution and shift rows operation, followed by the mix columns transformation.
3. Combine the results from the lookup tables using XOR to ensure efficient and fast operations.

Through these lookup tables and XOR operations, the AES round function can be efficiently implemented, reducing computational overhead.

Problem6

Hint 1:

Because InvSubBytes only changes byte values without altering their positions, and InvShiftRows only changes positions without altering values, the order of these two operations does not matter and can be interchanged without affecting the result.

Hint 2:

Since InvMixColumns is a linear operation, the round key K can be adjusted to K' such that:

$\mathrm{InvMixColumns}(S \oplus K) = InvMixColumns(S) \oplus K'$
This allows the order of
AddRoundKey and **InvMixColumns** to be interchanged.

- **AddRoundKey:** This operation XORs the state with a round key.
- **InvMixColumns:** This operation is the inverse of MixColumns, undoing the column mixing.

By skipping the MixColumns operation in the final round, the code and hardware design can be simplified:

1. **Software Implementation**:
    - (a) Omitting the MixColumns operation in the final round allows the use of the same functions for other steps in both encryption and decryption

processes.

- (b) Combine the repetitive parts of the encryption and decryption processes into a single function to reduce redundant code and improve code maintainability.

2. **Hardware Implementation**:
   - (a) Omitting the MixColumns operation in the final round can save some hardware resources because there is no need to implement the MixColumns module for the final round, thus reducing chip area.
   - (b) Since the same steps are omitted in both encryption and decryption processes, the control logic can be simplified, making the hardware design simpler and more efficient.

Problem7

Under What Circumstances Could You Choose 3DES Over AES

1. **Legacy Systems**:
   - If you are dealing with legacy systems or older hardware that only supports 3DES, it might be necessary to use 3DES to ensure compatibility.
2. **Regulatory Requirements**:
   - Certain industries or regions may have regulatory requirements that specify the use of 3DES.

Advantages of Choosing AES Over 3DES

1. **Security**:
   - AES provides a higher level of security compared to 3DES. AES has larger key sizes (128, 192, and 256 bits) and stronger cryptographic properties, making it more resistant to brute-force attacks and other cryptographic attacks.
2. **Performance**:
   - AES is significantly faster than 3DES, especially on modern hardware where AES is often hardware-accelerated. This makes AES more suitable for high-

performance applications.

3. **Efficiency**:
    - AES is more efficient in terms of both speed and energy consumption. It requires fewer computational resources, making it ideal for resource-constrained environments like mobile devices and IoT.

Susceptibility of 3DES to Meet-in-the-Middle Attacks

- 3DES is susceptible to meet-in-the-middle (MitM) attacks, although the attack is more complex and requires more computational resources than it does for 2DES.
- **2DES Susceptibility**:
    - 2DES, which involves two DES encryptions with two different keys, is highly susceptible to meet-in-the-middle attacks. The attack effectively reduces the security of 2DES from 112 bits (2 keys x 56 bits) to around 57 bits by finding a collision in the middle encryption step.
- **3DES Susceptibility**:
    - 3DES (Triple DES) involves three DES operations with either two or three different keys (EDE mode: Encrypt-Decrypt-Encrypt). While meet-in-the-middle attacks on 3DES are more challenging due to the additional encryption step, they are still theoretically possible. The effective security of 3DES is reduced from 168 bits (3 keys x 56 bits) to around 112 bits due to the MitM attack complexity.

Problem8

1. Assess the Situation

- **Determine the scope of key usage**: Identify all systems, data, and applications that are currently using the old key for encryption and decryption.
- **Assess risk**: Evaluate the potential risks and impacts of the compromised key, including unauthorized data access.

2. Key Revocation

- **Revoke the old key**: Mark the old key as compromised or revoked in all key management systems. Ensure it is no longer used for any cryptographic operations.
- **Notify relevant parties**: Inform all relevant stakeholders, including IT staff, security teams, and affected business units, about the revocation.

3. Generate a New Key

- **Generate a new encryption key**: Use a secure method to generate a new cryptographic key. Ensure it meets current security standards and guidelines.
- **Store the new key securely**: Store the new key in a secure key management system that controls access and usage.

4. Key Deployment

- **Update key references**: Modify configurations in all systems and applications to use the new key for future encryption operations.
- **Encrypt new data with the new key**: Ensure that any new data being encrypted uses the new key.

5. Re-encryption of Existing Data

- **Decrypt and re-encrypt data**: For data that was encrypted with the old key, follow a secure process to decrypt it using the old key and re-encrypt it using the new key. This can be done in phases to minimize operational impact.
    - **Backup data**: Before starting, ensure you have secure backups of all data.
    - **Automate the process**: If possible, use scripts or tools to automate the re-encryption process to reduce human error and expedite the process.

6. Key Management and Rotation Policies

- **Implement key rotation policies**: Establish policies for regular key rotation and management to prevent future risks associated with long-term key usage.
- **Train staff**: Ensure all relevant personnel are trained on key management practices and the importance of adhering to policies.

7. Audit and Verification

- **Audit the process**: Conduct an audit to verify that the old key is no longer in use and that all systems are correctly using the new key.
- **Validate re-encryption**: Ensure that all data previously encrypted with the old key has been successfully re-encrypted with the new key.

8. Document the Process

- **Record actions taken**: Document all steps taken during the key revocation and deployment process, including key generation, re-encryption steps, and any issues encountered.
- **Update security policies**: Reflect any lessons learned in the company's security policies and procedures to improve future key management processes.

Problem9

a)

Potential Problems:

1. **Reusing *n***:
   - n is crucial for the security of RSA. Reusing *n* while only changing *e* makes her vulnerable to specific types of attacks.
2. **Increased Factorization Risk**:
   - Continuously using the same *n* increases the chance that *n* could be factorized, revealing the prime factors *p* and *q*. Once these factors are known, any public key using this *n* is compromised.
3. **Coprime Requirement**:
   - In RSA, *e* should be chosen such that *e* and the totient function $\phi(n)$ (where $\phi(n)=(p-1)(q-1)$ for $n=pq$) are coprime. Changing *e* while keeping *n* constant could lead to a situation where *e* is not coprime with $\phi(n)$, compromising security.
4. **Common n Attack**:
   - If an attacker obtains multiple public keys with the same *n* but different *e* values, they might be able to perform a common *n* attack using some

mathematical relations.

b)

Potential Problems:

1. **Predictability of Primes**:
    - If an attacker discovers the pattern or one of the primes used in any month, they could potentially predict the future and past primes used by Alice. This compromises the security of her keys. For example, if pp and qq for a particular month are known, calculating the next primes $p1p1$ and $q1q1$ (or the previous ones) becomes straightforward, thereby compromising the subsequent public keys.
2. **Mathematical Relationships**:
    - Successive primes share certain mathematical properties. These properties could be exploited by an attacker to find relationships between the primes, aiding in breaking the encryption. For example, knowing *p* and *q* allows an attacker to identify the pattern and mathematical relationship that Alice is using to generate her primes, thereby predicting future keys and potentially compromising encrypted data.
3. **Small Gaps Between Consecutive Primes**:
    - The gap between consecutive prime numbers is relatively small compared to the size of the numbers themselves. This makes discovering the next or previous prime more feasible for an attacker. For instance, if the prime numbers *p* and *q* are known, finding *p*1 and *q*1 (the next primes) is a computationally manageable task due to the relatively small gaps between successive primes.
4. **Prime Generation Algorithms**:
    - Prime numbers can be predicted or calculated relatively quickly using prime generation algorithms. These algorithms are efficient at finding the next prime number after a given prime. This makes it easier for an attacker to determine the primes Alice will use next once a pattern is identified. An attacker who learns one set of primes could use these algorithms to predict future primes.

Problem10

a)

1. **brief explanation of the arithmetic**

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = \begin{bmatrix} (0E \cdot v_1) \oplus (0B \cdot v_2) \oplus (0D \cdot v_3) \oplus (09 \cdot v_4) \\ (09 \cdot v_1) \oplus (0E \cdot v_2) \oplus (0B \cdot v_3) \oplus (0D \cdot v_4) \\ (0D \cdot v_1) \oplus (09 \cdot v_2) \oplus (0E \cdot v_3) \oplus (0B \cdot v_4) \\ (0B \cdot v_1) \oplus (0D \cdot v_2) \oplus (09 \cdot v_3) \oplus (0E \cdot v_4) \end{bmatrix}$$

- Each multiplication is carried out in $GF(2^8)$ and can be retrieved from a precomputed lookup table.
- The results are then combined using XOR operations according to the arithmetic rules of $GF(2^8)$.

2. **Differences from Standard Matrix Multiplication**

- Addition is performed using XOR operations.
- Multiplication involves polynomial multiplication followed by reduction modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

(b)

1. Multiply by 9.

$X \cdot 9 = X(2^3+1) = X2^3 \oplus X$

mathematically:

$X \cdot 9 = (((X \cdot 2) \cdot 2) \cdot 2) \oplus X$ #

2. Multiply by 11:

$X \cdot 11 = X(2^3+2+1) = (X \cdot 2^3) \oplus (X \cdot 2) \oplus X$

$\Rightarrow X \cdot 11 = (((X \cdot 2) \cdot 2) \cdot 2) \oplus (X \cdot 2) \oplus X$ #

3. Multiply by 13:

$X \cdot 13 = X(2^3+2^2+1) = (X \cdot 2^3) \oplus (X \cdot 2^2) \oplus X$

$\Rightarrow X \cdot 13 = (((X \cdot 2) \cdot 2) \cdot 2) \oplus ((X \cdot 2) \cdot 2) \oplus X$ #

4. Multiply by 14:

$X \cdot 14 = X(2^3+2^2+2) = (X \cdot 2^3) \oplus (X \cdot 2^2) \oplus X$

$\Rightarrow X \cdot 14 = (((X \cdot 2) \cdot 2) \cdot 2) \oplus (((X \cdot 2) \cdot 2) \oplus (X \cdot 2)$ #

c)

1. **Precomputation in $GF(2^8)$)**:
    - Since multiplication in $GF(2^8)$ can be computed through a finite number of calculations, precomputing these results can save time during execution.
2. **Using Lookup Tables (LUT)**:
    - The results can be directly retrieved from a lookup table, simplifying the process.

Pros of Using LUT

1. **Increased Efficiency**:
    - LUTs can significantly increase efficiency by reducing time complexity.
2. **Constant Time Execution**:
    - LUTs provide $O(1)$time execution for retrieving results, which helps in avoiding timing attacks by making execution time less dependent on the input data.
3. **Reduced Code Complexity**:
    - They simplify the implementation of polynomial multiplication and modulo operations, reducing code complexity.

Cons of Using LUT

1. **Initialization Overhead**:
    - Initializing these tables can consume a considerable amount of time.
2. **Memory Consumption**:
    - LUTs for Inverse MixColumns require significant memory. Each coefficient needs a table with 256 entries, and each entry typically consumes one byte.
3. **Lack of Flexibility**:
    - LUTs are less flexible to changes, making updates more cumbersome.
4. **Vulnerability to Attacks**:

- The reliance on memory lookups introduces vulnerability to cache timing attacks.

Problem11

Scenario Setup

- The attacker knows the Initialization Vector IV,C_0 ,C_1 , and C_2 .
- The attacker also knows $m_1 = m_2 = x$.

Encryption Process

The encryption process is given by:

$C_i = E_k(m_i) \oplus C_{i-1}$
where
E_k represents the encryption function under the key k.

Vulnerability Analysis

Given that m_1 and m_2 are known and equal to x, and the values of C_0 ,C_1 , and C_2 are known, the attacker can exploit this information as follows:

1. **Encryption Equation**:
   - The encryption process is described by:

     $C_i = E_k(m_i) \oplus C_{i-1}$
   - Rearranging this equation to solve for m_i gives:

     $m_i = D_k(C_i \oplus C_{i-1})$
     where
     D_k represents the decryption function under the key k .
2. **Given** $m_1 = m_2 = x$:
   - Since m_1 and m_2 are both equal to x, we can write:

     $x = m_1 = m_2 = D_k(C_1 \oplus C_0) = D_k(C_2 \oplus C_1)$

3. **Exploiting the Known Values**:
    - From the encryption equation, we have:

      $$m_1 = C_2 \oplus E_k(m_2) = C_2 \oplus E_k(m_1)$$

    - This implies:

      $$E_k(m_1) = m_1 \oplus C_2$$

4. **Computing** m_0 :
    - Using the equation for m_0 :

      $$m_0 = C_1 \oplus E_k(m_1)$$

    - Substituting $E_k(m_1)$ from the previous step:

      $$m_0 = C_1 \oplus (m_1 \oplus C_2) = x \oplus C_1 \oplus C_2$$

Conclusion

The Predecessor Block Chaining (PBC) encryption scheme is insecure because knowing that two plaintexts are the same allows an attacker to compute the previous plaintext. This is a significant security breach in any cryptographic system, as it enables the attacker to deduce sensitive information about the encrypted messages.

Problem 12

Hint 1 Steps

1. **Calculate Initial Blocks**:
    - Since $m_1 = m_2 = x$, calculate:

      $m_1 = D_k(C_1 \oplus C_0)$
      $m_2 = D_k(C_2 \oplus C_1)$

2. **Append and Modify**:

- Construct messages *M*1 and *M*2 and append *M*2's MAC to *M*1, but alter the last block of *M*1 as M_3. make M_2 process same as using it own IV_2 .

3. **Recalculate the Encryption Block**:
    - Calculate M_3 by calculating the difference between T_1 and IV_2. It is similar to $M_1||M_1$ with one block altered, and the CBC process of M_3 has initial vector = IV_1 and final encryption block = T_2.

Hint 2 Steps

1. **Assume Known Message Blocks**:
    - Assume the message *M* consists of blocks $m_0, m_1, ..., m_n$.
2. **Construct New Message**:
    - Construct a new message *M′* with blocks $m_0, m_1, ..., m_n$.
    - Set the initial vector IV to $m0 \oplus IV$.
3. **Generate Valid MAC**:
    - The output tag of *M′* will be the same as the tag of *M*, resulting in a valid MAC.