

### Assessment – 3

#### Cryptography Analysis and Implementation

**Objective:** The objective of this assignment is to analyze cryptographic algorithms and implement them in a practical scenario.

##### **DES**

A symmetric encryption technique that uses 64-bit data blocks is called the Data Encryption Standard (DES). Here is a quick explanation of the DES algorithm's operation:

**Key Generation:** DES employs a 56-bit encryption key. However, the effective key size is decreased to 56 bits due to security concerns by using 8 of the 64 bits for parity tests. An algorithm that incorporates human input and a random component creates the key.

**Key Expansion:** For each encryption cycle, a separate 48-bit subkey is created from the 56-bit key. This expansion comprises a series of mathematical procedures that permute and transform the key.

**Initial Permutation (IP):** The bits in the 64-bit plaintext block are rearranged in accordance with a predetermined pattern.

**Rounds of Encryption:** DES uses 16 rounds of encryption. The following steps are involved in each round:

- a. **Expansion:** Using a specified expansion function, the 32-bit right half of the data from the preceding round is enlarged to 48 bits.
- b. **Key Mixing:** The relevant round subkey is XORed with the extended 48-bit data.
- c. **Substitution:** Eight 6-bit blocks make up the final product. A predetermined S-box (substitution box), which converts each block's six bits into four bits using a lookup table, is used to substitute each block.
- d. **Permutation:** Using a predetermined permutation pattern, the substituted data is then rearranged.
- e. **XOR and Swap:** The data that has been permuted is XORed with the 32-bit left half of the data from the previous round. The next round's right half is the 32-bit block that was produced, and the left half is the preceding right half.

After the 16 cycles, the data is subjected to a final permutation (FP), which is the opposite of the initial permutation.

The same procedure is used to decrypt the ciphertext, but the subkeys are used in reverse.

Strengths:

1. **Simplicity:** DES has a relatively simple structure, making it easy to implement and understand.
2. **Performance:** DES was designed to be efficient in hardware implementations, which made it suitable for many applications at the time.

Weaknesses:

1. **Key Length:** The key length of 56 bits is considered short by today's standards, and it can be brute-forced within a reasonable amount of time using modern computing resources.
2. **Vulnerability to Attacks:** DES has been subjected to various cryptanalytic attacks over the years. The most significant attack is the exhaustive search, where all possible keys are tested until the correct one is found.
3. **Security Margin:** Due to advances in computing power, the security margin of DES has significantly decreased over the years. It is now recommended to use more robust encryption algorithms.

Common Use Cases:

1. **Legacy Systems:** DES may still be encountered in legacy systems that have not been updated or where compatibility with older systems is required.
2. **Educational Purposes:** DES is often used in academic settings to teach the fundamentals of cryptography and encryption algorithms.

## **Elliptic Curve Cryptography ECC**

Key generation, key exchange, encryption, and decryption are all components of the elliptic curve cryptography (ECC) algorithm. The ECC algorithm is described in the following general terms:

### **Generating a Key:**

Pick a base point on an elliptic curve that is specified over a finite field.

Pick a private key, which is a chance number that falls within a certain range.

By employing elliptic curve scalar multiplication, multiply the base point by the private key to create the public key.

### **Exchanging keys**

Alice and Bob, two parties, concur on an elliptic curve and its base point.

Both Alice's public and private keys are generated.

Bob uses both his private key and public key in the same way.

Public keys are exchanged between Alice and Bob.

To obtain a shared secret point on the curve, each party multiplies their respective private key and the public key they got.

### **Encryption:**

Bob is the recipient of a communication from Alice.

Alice creates a session key at random.

By dividing the base point by the session key, she calculates the coordinates of an ephemeral public key.

The shared secret point is calculated by Alice by dividing Bob's public key by her session key.

Alice uses the shared secret point to derive a symmetric encryption key.

She uses the symmetric encryption key to encrypt the communication.

### **Decryption:**

Alice gives Bob the temporary public key and the encrypted message.

Bob multiplies his private key with Alice's transitory public key to determine the shared secret point.

Bob and Alice arrive at the same symmetric encryption key.

He uses the symmetric encryption key to decrypt the communication.

Strengths:

1. **Security:** ECC provides a high level of security with smaller key sizes compared to other public-key algorithms such as RSA. This makes ECC more efficient in terms of computational resources and bandwidth.
2. **Key Size Efficiency:** ECC provides an equivalent level of security with significantly smaller key sizes compared to other asymmetric algorithms, reducing the computational and storage requirements.

Weaknesses:

1. **Implementation Complexity:** ECC requires careful implementation and parameter selection to ensure security. Improper implementation can introduce vulnerabilities.
2. **Patented Algorithms:** Some ECC algorithms may be subject to patents, which can limit their use and adoption.

Common Use Cases:

1. **Secure Communication:** ECC is widely used in protocols such as Transport Layer Security (TLS) to secure communication channels over the internet.
2. **Internet of Things (IoT):** ECC's efficiency and security make it suitable for resource-constrained devices in IoT environments.
3. **Digital Signatures:** ECC is used for generating and verifying digital signatures, ensuring the integrity and authenticity of digital documents.

## MD5 (Message Digest Algorithm 5)

is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. It takes an input message of any length and produces a fixed-size output, which is commonly represented as a 32-digit hexadecimal number.

The algorithm follows the following steps:

1. **Padding:** The input message is padded to make its length a multiple of 512 bits (64 bytes). The padding is done in such a way that the resulting message length is congruent to 448 modulo 512. The padding starts with a single bit "1" followed by a sequence of "0" bits until the length requirement is met. After that, the original message length is appended as a 64-bit representation.
2. **Initialization:** MD5 uses four 32-bit state variables (A, B, C, D) and a 64-element table (T[1..64]) containing precomputed values. The initial values of A, B, C, and D are fixed and serve as the initial state.
3. **Message Processing:** The padded message is divided into blocks of 512 bits. Each block is further divided into 16 words of 32 bits each (M[0..15]). The message is processed in a loop for each block.
4. **Round Function:** MD5 uses four basic functions (F, G, H, I) that operate on the state variables A, B, C, and D, as well as the current block's words. Each function takes three inputs (X, Y, Z) and produces a 32-bit output. The functions and their operations are as follows:
  - $F(X, Y, Z) = (X \text{ AND } Y) \text{ OR } ((\text{NOT } X) \text{ AND } Z)$
  - $G(X, Y, Z) = (X \text{ AND } Z) \text{ OR } (Y \text{ AND } (\text{NOT } Z))$
  - $H(X, Y, Z) = X \text{ XOR } Y \text{ XOR } Z$
  - $I(X, Y, Z) = Y \text{ XOR } (X \text{ OR } (\text{NOT } Z))$
5. **Round Operations:** Each block goes through four rounds of processing. In each round, a different function is used, and the results are combined with the state variables using modular addition. The block's 16 words are used as inputs to the functions in a predetermined order.
6. **Output:** After processing all the blocks, the resulting values of A, B, C, and D are concatenated, usually in little-endian format. The resulting 128-bit hash value represents the message's fingerprint.

Strengths:

1. **Speed:** MD5 is fast and efficient in terms of computation, making it suitable for applications where performance is critical.
2. **Compatibility:** MD5 is supported by many software systems and programming languages, making it easy to implement and use.

#### Weaknesses:

1. Collision Vulnerability: MD5 is considered weak against collision attacks, where two different inputs produce the same hash value. This weakness makes it unsuitable for security-critical applications.
2. Preimage Vulnerability: MD5 is also susceptible to preimage attacks, where an attacker can find an input that produces a specific hash value, compromising the integrity of the system.

#### Common Use Cases:

1. Checksums: MD5 checksums are used to verify the integrity of files. By comparing the MD5 hash of a downloaded file with the provided hash

## IMPLEMENTING DES

### CODE

```
#include <iostream>
#include <cstring>
#include <openssl/des.h>

// Function to encrypt using DES algorithm
void encryptDES(const unsigned char* plaintext, const unsigned char* key,
unsigned char* ciphertext) {
    DES_cblock keyEncrypt;
    DES_key_schedule schedule;

    // Prepare the key for encryption
    memcpy(keyEncrypt, key, 8);
    DES_set_odd_parity(&keyEncrypt);
    DES_set_key_checked(&keyEncrypt, &schedule);

    // Encrypt the plaintext
    DES_ecb_encrypt((DES_cblock*)plaintext, (DES_cblock*)ciphertext,
&schedule, DES_ENCRYPT);
}

// Function to decrypt using DES algorithm
void decryptDES(const unsigned char* ciphertext, const unsigned char* key,
unsigned char* plaintext) {
    DES_cblock keyEncrypt;
    DES_key_schedule schedule;

    // Prepare the key for decryption
    memcpy(keyEncrypt, key, 8);
    DES_set_odd_parity(&keyEncrypt);
```

```
DES_set_key_checked(&keyEncrypt, &schedule);

// Decrypt the ciphertext
DES_ecb_encrypt((DES_cblock*)ciphertext, (DES_cblock*)plaintext,
&schedule, DES_DECRYPT);
}

int main() {
    // Input plaintext and key
    const unsigned char plaintext[] = "Hello, DES!";
    const unsigned char key[] = "SecretKey";

    // Buffer to store the encrypted and decrypted data
    unsigned char ciphertext[32];
    unsigned char decryptedtext[32];

    // Encrypt the plaintext
    encryptDES(plaintext, key, ciphertext);

    // Decrypt the ciphertext
    decryptDES(ciphertext, key, decryptedtext);

    // Print the encrypted and decrypted messages
    std::cout << "Plaintext: " << plaintext << std::endl;
    std::cout << "Ciphertext: " << ciphertext << std::endl;
    std::cout << "Decrypted text: " << decryptedtext << std::endl;

    return 0;
}
```

## Potential Threats for DES algo

When implementing the DES algorithm, there are several potential threats and vulnerabilities that could be exploited. Here are some of them:

1. **Key Security:** The security of the algorithm heavily relies on the secrecy and strength of the encryption key. If the key is weak or compromised, the entire system becomes vulnerable. Therefore, it is crucial to use a strong and random key and protect it using secure key management practices.

Countermeasure: Implement a robust key management system that includes strong key generation techniques, secure storage, and appropriate key rotation policies. Consider using longer key lengths and more advanced encryption algorithms, such as AES, for stronger security.

2. **Brute Force Attacks:** DES has a relatively small key length (56 bits), which makes it susceptible to brute force attacks. An attacker could systematically try all possible keys until the correct one is found, especially with the increasing computational power available today.

Countermeasure: Use stronger encryption algorithms, such as AES, which provide longer key lengths and better resistance against brute force attacks. AES is the recommended replacement for DES.

3. Cryptanalysis Attacks: Over time, new attacks and vulnerabilities against cryptographic algorithms are discovered. DES, being an older algorithm, has known weaknesses and vulnerabilities. Advanced cryptanalysis techniques, such as differential and linear cryptanalysis, can exploit these weaknesses.

Countermeasure: Upgrade to more modern and secure encryption algorithms, such as AES. AES is widely adopted and extensively analyzed, providing better resistance against known cryptanalysis attacks.

4. Implementation Flaws: Errors or vulnerabilities in the implementation of the algorithm can compromise the security. These flaws could be related to key handling, input validation, or insecure cryptographic operations.

Countermeasure: Follow secure coding practices and rely on well-tested cryptographic libraries or frameworks. Regularly update and patch the implementation to address any discovered vulnerabilities or weaknesses.

5. Side-Channel Attacks: Side-channel attacks target information leaked during the execution of the algorithm, such as timing, power consumption, or electromagnetic emissions. These attacks can potentially reveal sensitive information, including the encryption key.

Countermeasure: Implement countermeasures against side-channel attacks, such as using constant-time algorithms, ensuring consistent execution time regardless of input, and applying physical protections like tamper-resistant hardware.

6. Regulatory and Compliance Issues: Depending on the context and the data being encrypted, there may be legal or compliance requirements to consider. Using an outdated encryption algorithm like DES may not meet the necessary security standards and could lead to compliance violations.

Countermeasure: Understand the relevant legal and compliance requirements for your specific use case. Ensure that your encryption algorithms align with the recommended standards and industry best practices. Upgrade to modern and approved encryption algorithms to meet compliance obligations.



## **Trade Offs**

During the implementation process of the DES algorithm, several limitations and trade-offs were encountered. Here are some of them:

1. **Key Length:** One significant limitation of DES is its relatively short key length of 56 bits. This limited key length provides weaker security compared to modern encryption algorithms. Brute force attacks can become feasible as computational power increases. As a result, the key length of DES is a trade-off between security and efficiency.
2. **Speed:** DES was designed for efficient hardware implementation in the 1970s when computational power was limited. While DES is relatively fast compared to other encryption algorithms of that time, it is slower than modern encryption algorithms like AES. When performance is a critical factor, the speed of DES can be a limitation.
3. **Security:** Over time, DES has been subjected to various cryptanalysis attacks, and several vulnerabilities have been discovered. Although DES is still considered secure for certain legacy applications, it is no longer considered secure for protecting sensitive data against advanced attacks. Therefore, the security provided by DES is a significant limitation compared to modern algorithms like AES.
4. **Compatibility:** DES is an older encryption algorithm that may not be widely supported by modern systems and frameworks. Some programming languages or platforms may lack built-in support for DES, requiring additional effort to integrate or implement the algorithm.
5. **Regulatory Compliance:** Depending on the industry or jurisdiction, there may be specific regulations or compliance requirements that mandate the use of stronger encryption algorithms like AES. Implementing DES may not meet these requirements, necessitating an upgrade to a more secure algorithm.
6. **Key Management:** As DES uses a fixed key length, key management can be challenging. With a limited key space, the risk of key collisions and key reuse increases. This limitation can impact the overall security of the system if proper key management practices are not implemented.

**THANK YOU**