MANIPAL ACADEMY
OF HIGHER EDUCATION

(Deemed-to-be-University under Section 3 of the UGC Act, 1956)

Information Security Lab

V Semester: B. Tech CCE (ICT3126)

Year: 2024

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGYMANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

CONTENT

Course Objectives

· Assess potential system vulnerabilities.
· Gain practical insight into various algorithms that can provide system, device, and network security.
· Use techniques and tools to fix security vulnerabilities

Course Outcomes

1. Examine security threats and vulnerabilities.
2. Demonstrate the various methods which can combat different security threats.
3. Model a security infrastructure appropriately to provide maximum security against any breach.

Evaluation Plan

· Split up of 60 marks for Regular Lab Evaluation

– Mid sem: 1 / 2 questions (Scenario based):20 Marks

– Record: 5 Marks: 3 Submissions

– Quiz: 15 Marks

– Project Midsem Evaluation : 10 Marks

· End Semester Lab evaluation: 40 marks (Duration 2 hrs)

– Program: 20 Marks

– Project: 20 Marks

Pre-Lab Session Instructions

1. Students should carry the Lab Manual Book and the required stationery to every labsession
2. Be on time and follow the institution's dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In-Lab Session Instructions

1. Follow the instructions on the allotted exercises
2. Show the program and results to the instructors on completion of experiments
3. Prescribed textbooks and class notes can be kept ready for reference if required
4. Avoid using LLMs

General Instructions for the Exercises in Lab

· Implement the given exercise individually and not in a group.

– Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.

– Comments should be used to state the problem.

– The statements within the program should be properly indented

Instructions to Students

- Plagiarism (copying from others) is strictly prohibited and would invite severe penalties in evaluation.

- In case a student misses a lab, he/ she must ensure that the experiment is completed before the next evaluation with the permission of the faculty concerned.

- Students missing out on the lab for genuine reasons like conferences, sports or activities assigned by the Department or Institute will have to take prior permission from the HOD to attend additional lab (with another batch) and complete it before the studentgoes on leave. The student could be awarded marks for the write-up for that day provided he submits it during the immediate next lab.

- Students who feel sick should get the HOD's permission to evaluate the lab records. However, attendance will not be given for that lab.

- Students will be evaluated only by the faculty with whom they are registered even though they carry out additional experiments in another batch.

- The presence of the student during the lab end semester exams is mandatory even ifthe student assumes he has scored enough to pass the examination

- Minimum attendance of 75

- If the student loses his book, he/she will have to rewrite all the lab details in the lab record.

- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

  The students should NOT Bring mobile phones or any other electronic gadgets to the lab.

- The students should NOT Go out of the lab without permission.

Lab No. 1: Basic Symmetric Key Ciphers

Objectives

- To familiarize with Substitution Ciphers.
- To understand the working of transposition ciphers.

Symmetric key ciphers use the same key to encrypt and decrypt data. They are often used in combination with other algorithms into a symmetric encryption schemes.

Symmetric key cryptography schemes are categorized as stream ciphers and block ciphers. Stream ciphers work on a single bit (byte or computer word) at a time and execute some form of feedback structure so that the key is repeatedly changing.

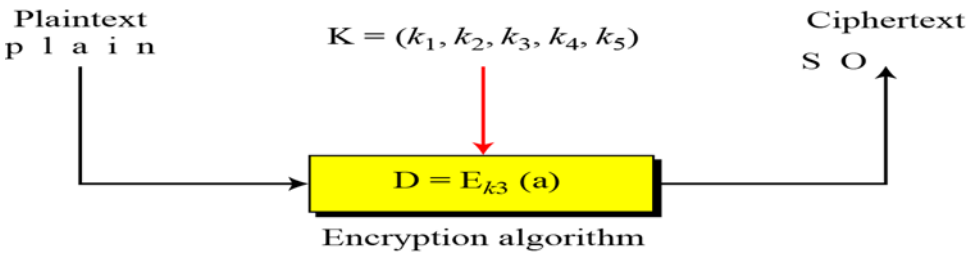


Figure 1: Stream Cipher

Block Cipher

In a block cipher, a group of plaintext symbols of size m ($m > 1$) are encrypted together creating a group of ciphertext of the same size. A single key is used to encrypt the whole block even if the key is made of multiple values as shown in Figure 2.

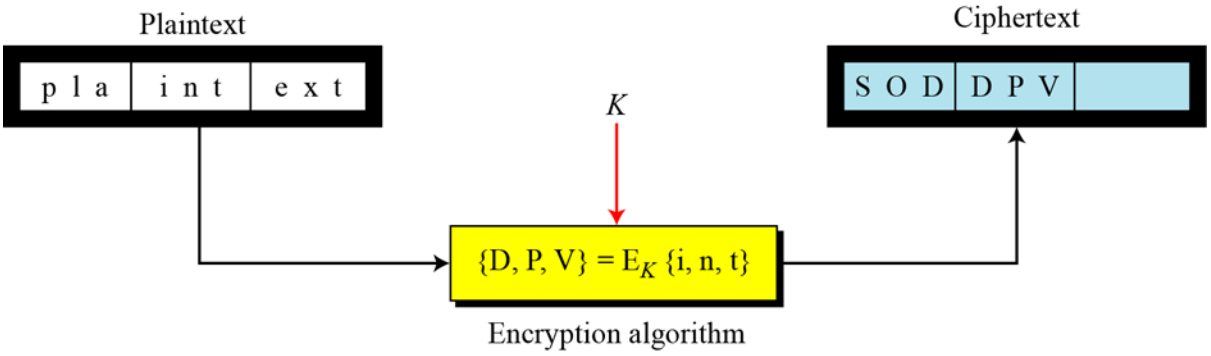


Figure 2: Block Cipher

A block cipher is so-called because the scheme encrypts one block of information at a time utilizing the same key on each block. In general, the same plaintext block will continually encrypt to the same ciphertext when using the similar key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

Block ciphers can operate in several modes such as Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB) mode.

Substitution Ciphers

A substitution cipher replaces one symbol with another. Substitution ciphers can be categorized as either monoalphabetic ciphers or polyalphabetic ciphers

In monoalphabetic substitution, the relationship between a symbol in the plaintext to a symbol in the ciphertext is always one-to-one. The simplest monoalphabetic cipher is the additive cipher, shown in Figure 3. This cipher is sometimes called a shift cipher and sometimes a Caesar cipher.

When the cipher is additive, the plaintext, ciphertext, and key are integers in Z_{26}.

In a multiplicative cipher, the plaintext and ciphertext are integers in Z_{26}; the key is an integer in Z_{26}*.

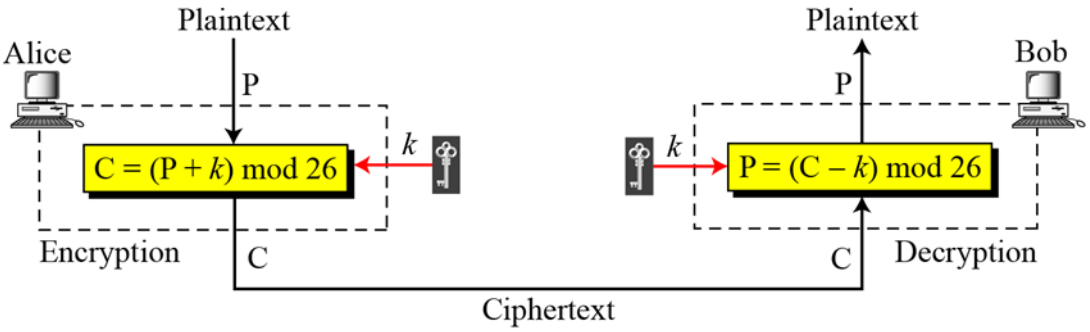


Figure 3. Additive Cipher

In polyalphabetic substitution, each occurrence of a character may have a different substitute. The relationship between a character in the plaintext to a character in the ciphertext is one-to-many.

Auto Key Cipher

$P = P_1P_2P_3 \ldots$ $C = C_1C_2C_3 \ldots$ $k = (k_1, P_1, P_2, \ldots)$

Encryption: $C_i = (P_i + k_i) \bmod 26$ Decryption: $P_i = (C_i - k_i) \bmod 26$

Transposition Cipher

A transposition cipher does not substitute one symbol for another, instead it changes the location of the symbols.

- Keyless Transposition Ciphers
- Keyed Transposition Ciphers
- Combining Two Approaches

Lab Exercises

1) Encrypt the message "I am learning information security" using each of the following ciphers. Ignore the space between words. Decrypt the message to get the original plaintext:
   a) Additive cipher with key = 20
   b) Multiplicative cipher with key = 15
   c) Affine cipher with key = (15, 20)

2) Encrypt the message "the house is being sold tonight" using each of the following ciphers. Ignore the space between words. Decrypt the message to get the original plaintext:
   a) Vigenere cipher with key: "dollars"
   b) Autokey cipher with key = 7

3) Use the Playfair cipher to encipher the message "The key is hidden under the door pad". The secret key can be made by filling the first and part of the second row with the word "GUIDANCE" and filling the rest of the matrix with the rest of the alphabet.

4) Use a Hill cipher to encipher the message "We live in an insecure world". Use the following key:
   K = [03 03
   02 07]

5) John is reading a mystery book involving cryptography. In one part of the book, the author gives a ciphertext "CIW" and two paragraphs later the author tells the reader that this is a shift cipher and the plaintext is "yes". In the next chapter, the hero found a tablet in a cave with "XVIEWYWI" engraved on it. John immediately found the actual meaning of the ciphertext. Identify the type of attack and plaintext.

6) Use a brute-force attack to decipher the following message. Assume that you know it is an affine cipher and that the plaintext "ab" is enciphered to "GL":
   XPALASXYFGFUKPXUSOGEUTKCDGEXANMGNVS

Additional Exercises

1. Use a brute-force attack to decipher the following message enciphered by Alice using an additive cipher. Suppose that Alice always uses a key that is close to her birthday, which is on the 13th of the month:
NCJAEZRCLAS/LYODEPRLYZRCLASJLCPEHZDTOPDZOLN&BY

2. Eve secretly gets access to Alice's computer and using her cipher types "abcdefghi". The screen shows "CABDEHFGL". If Eve knows that Alice is using a keyed transposition cipher, answer the following questions:

a) What type of attack is Eve launching?
b) What is the size of the permutation key?
c) Use the Vigenere cipher with keyword "HEALTH" to encipher the message "Life is full of surprises".

Lab No. 2: Advanced Symmetric Key Ciphers

Objectives

- To understand the working of DES and AES algorithms.
- To implement AES and DES algorithm.
- Analyze the performance of AES and DES algorithms

DES:

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST)

The encryption process is made of two permutations (P-boxes, initial and final permutations, and sixteen Feistel rounds as shown in Figure 4

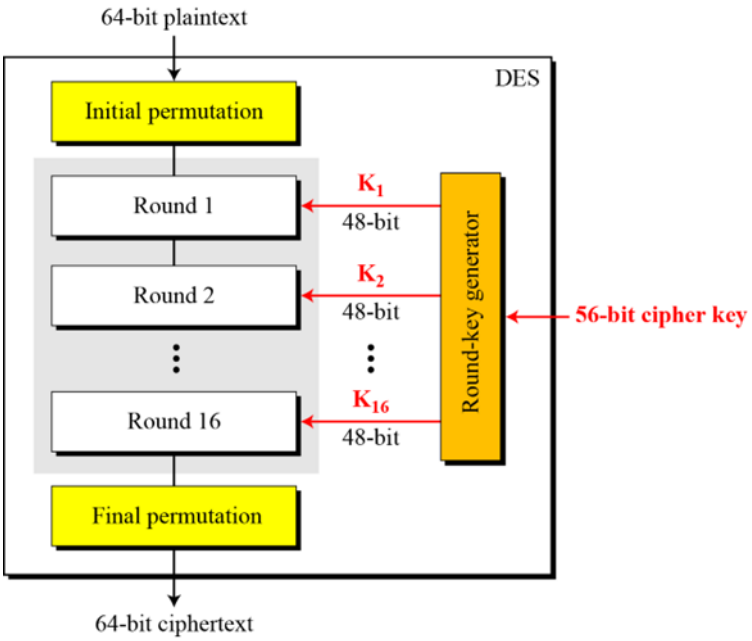


Figure 4: General structure of DES

Initial Permutation (IP): The 64-bit plaintext block is permuted according to a fixed table, shuffling the bits to create a new order.

Key Schedule Generation: The 56-bit key is divided into two 28-bit halves. Each half is then rotated and permuted according to a predefined schedule to produce sixteen 48-bit round keys, one for each round of encryption.

16 Rounds of Encryption: The 64-bit block is split into two 32-bit halves, called Left (L) and Right (R).

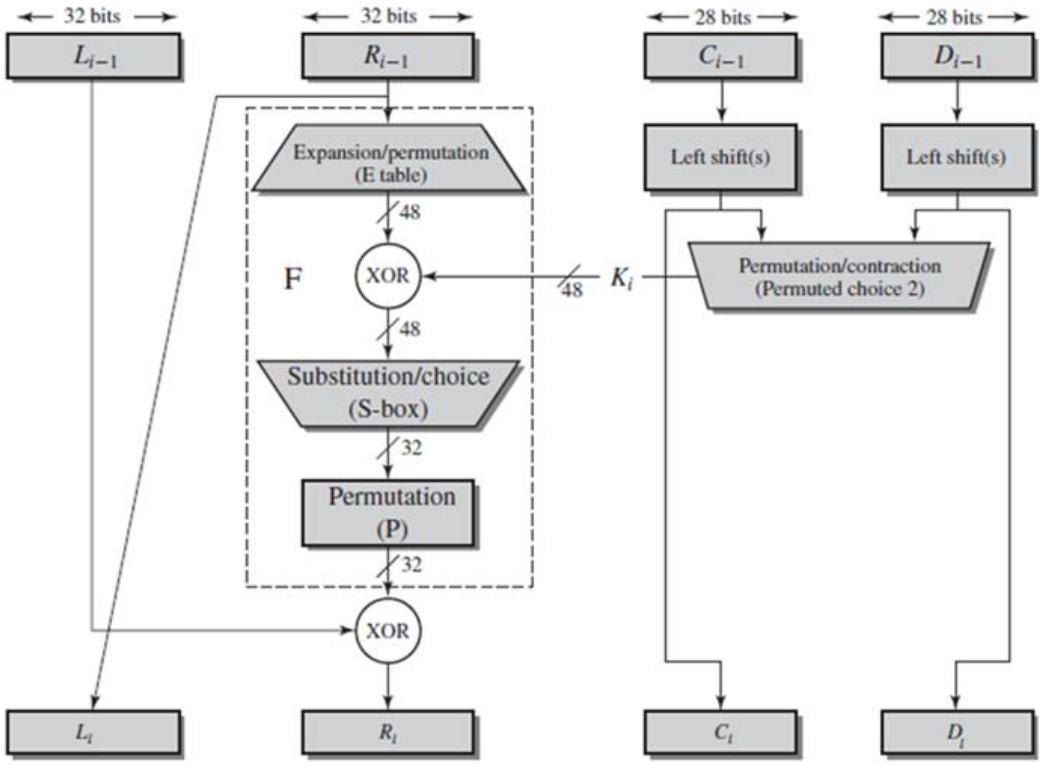


Figure 5: Single Round of DES Algorithm

Single round of DES is shown in Figure 5. For each of the 16 rounds, a new right half is generated by expanding the previous right half to 48 bits using the Expansion (E) function.

The expanded right half is XORed with the round key. That result is passed through a series of substitution boxes (S-boxes), which reduce the 48-bit output back to 32 bits. The S-box output is then permuted using the Permutation (P) function. The new right half is XORed with the previous left half. The previous right half becomes the new left half.

- After 16 rounds, the left and right halves are recombined and permuted using the final permutation (FP).

Final Permutation (FP): The combined left and right halves are permuted according to a fixed table to produce the final 64-bit ciphertext block.

The Advanced Encryption Standard (AES):

AES is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST) in December 2001. AES is a non-Feistel cipher that encrypts and decrypts a data block of 128 bits. It uses 10, 12, or 14 rounds. The key size, which can be 128, 192, or 256 bits, depends on the number of rounds.

Main components of AES are as follows as shown in Figure 6.

Initial Round: **AddRoundKey**: Each byte of the state is combined with a round key using the XOR operation.

Main Rounds (Repeated for 9, 11, or 13 times depending on key size):

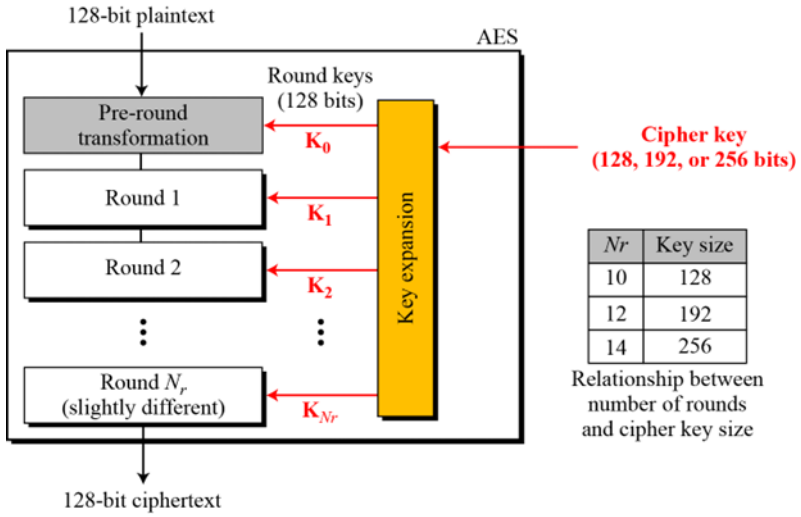
SubBytes: A non-linear substitution step where each byte is replaced with another byte using an S-box (substitution box).

ShiftRows: A transposition step where each row of the state is shifted cyclically by a certain number of bytes.

MixColumns: A mixing operation which operates on the columns of the state, combining the four bytes in each column.

AddRoundKey: Each byte of the state is combined with a round key using the XOR operation.

Final Round: It has SubBytes, ShiftRows and AddRoundKey steps



Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds and cipher key size

Figure 6: General design of AES encryption cipher

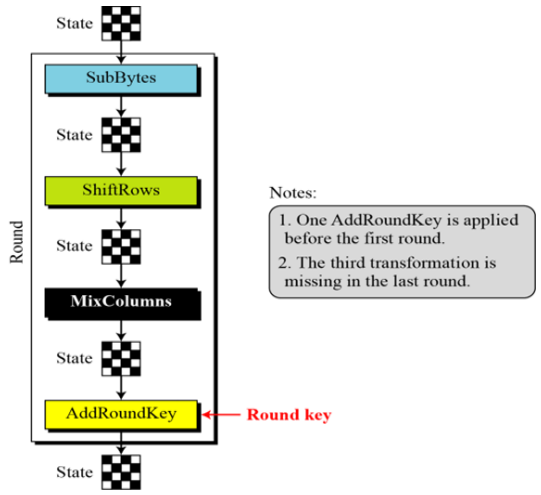


Figure 7: Structure of each round at the encryption site

To provide security, AES uses four types of transformations: substitution, permutation, mixing, and key-adding.

Detailed description about each type of transformation is as follows:

Substitution

The first transformation, SubBytes, is used at the encryption site. To substitute a byte, we interpret the byte as two hexadecimal digits.

Transformation Using the GF(2^8) Field

AES also defines the transformation algebraically using the GF(28) field with the irreducible polynomials ($x^8 + x^4 + x^3 + x + 1$)

The SubBytes and InvSubBytes transformations are inverses of each other.

Another transformation found in a round is shifting, which permutes the bytes as shown in Figure 8. In the encryption, the transformation is called ShiftRows.

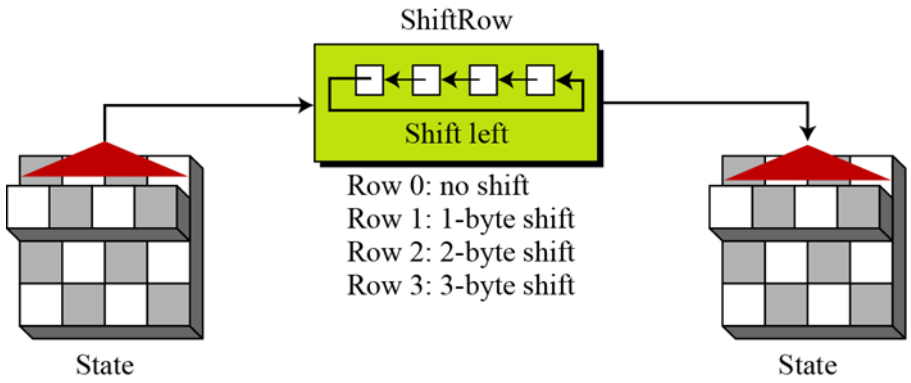


Figure 8: ShiftRows transformation

InvShiftRows

In the decryption, the transformation is called InvShiftRows and the shifting is to the right.

MixColumns

The MixColumns transformation operates at the column level; it transforms each column of the state to a new column. The MixColumns and InvMixColumns transformations are inverses of each other.

Key Adding

AddRoundKey proceeds one column at a time. AddRoundKey adds a round key word with each state column matrix; the operation in AddRoundKey is matrix addition.

Exercises:

1. Encrypt the message "Confidential Data" using DES with the following key: "A1B2C3D4". Then decrypt the ciphertext to verify the original message.

2. Encrypt the message "Sensitive Information" using AES-128 with the following key: "0123456789ABCDEF0123456789ABCDEF". Then decrypt the ciphertext to verify the original message.

3. Compare the encryption and decryption times for DES and AES-256 for the message "Performance Testing of Encryption Algorithms". Use a standard implementation and report your findings.

4. Encrypt the message "Classified Text" using Triple DES with the key "1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF". Then decrypt the ciphertext to verify the original message.

5. Encrypt the message "Top Secret Data" using AES-192 with the key "FEDCBA9876543210FEDCBA9876543210". Show all the steps involved in the encryption process (key expansion, initial round, main rounds, final round).

Additional Exercises:

1. Using DES and AES(128, 192, and 256 bits key).encrypt the five different messages using same key.

   a. Consider different modes of operation
   b. Plot the graph which shows execution time taken by each technique.
   c. Compare time taken by different modes of operation

2. Encrypt the following block of data using DES with the key "A1B2C3D4E5F60708". The data to be encrypted is: Mathematica

Block1:
5468697320697320612063 6f6e666964656e7469616c206d657373616765

Block2:
416e64207468697320697320746865207365636f6e6420626c6f636b

   a. Provide the ciphertext for each block.
   b. Decrypt the ciphertext to retrieve the original plaintext blocks.

3. Using AES-256, encrypt the message "Encryption Strength" with the key "0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF". Then decrypt the ciphertext to verify the original message.
Encrypt the message "Secure Communication" using DES in Cipher Block Chaining (CBC) mode with the key "A1B2C3D4" and an initialization vector (IV) of "12345678". Provide the ciphertext and then decrypt it to retrieve the original message.

4. Encrypt the message "Cryptography Lab Exercise" using AES in Counter (CTR) mode with the key "0123456789ABCDEF0123456789ABCDEF" and a nonce of "0000000000000000". Provide the ciphertext and then decrypt it to retrieve the original message.

Lab No. 3: Asymmetric Key Ciphers

Objectives

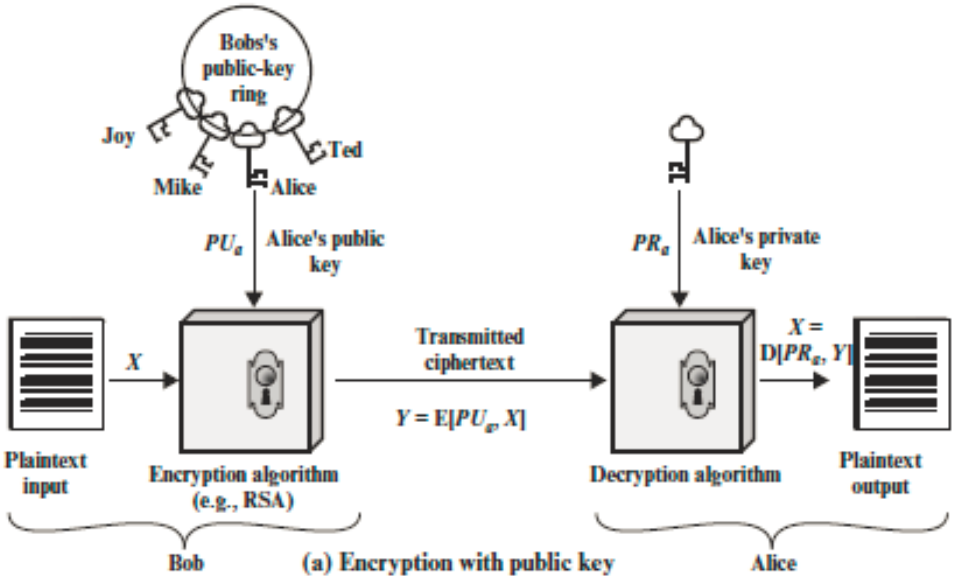
- To demonstrate the ability to generate, use, and understand the public and private keys in various asymmetric encryption schemes.

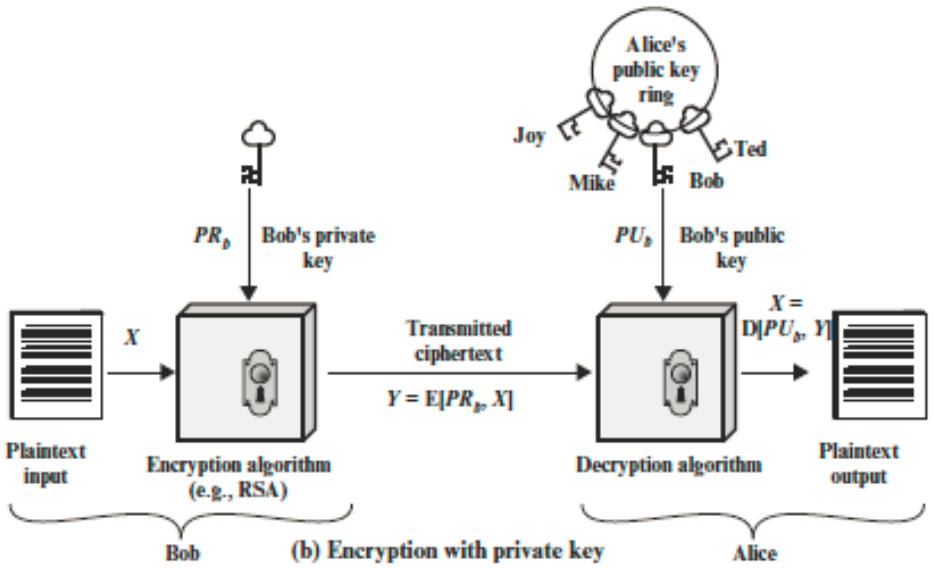
- To understand the performance implications of different asymmetric encryption algorithms

- Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristics:

- It is computationally infeasible to determine the decryption key given only knowledge of cryptographic algorithms and encryption keys.

- Either of the two related keys can be used for encryption, with the other used for decryption





THE RSA ALGORITHM

The RSA scheme is a cipher in which the plaintext and ciphertext are integers between 0 and n - 1 for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 21024. We examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytical implications of

RSA. RSA uses the mathematical properties of prime numbers and modular arithmetic.

- Public keys are used for encryption, and private keys are used for decryption.

- The security of RSA relies on the difficulty of factoring large composite numbers into their prime factors.

RSA is widely used for secure data transmission, digital signatures, and key exchange mechanisms due to its robustness and security features

Key Generation

- Generate Two Large Prime Numbers: Choose two distinct large prime numbers, p and q. These primes should be large enough to ensure the security of

the RSA algorithm.

- Compute the Modulus:

Calculate $n=p\times q$; n is used as the modulus for both the public and private keys.

- Compute Euler's Totient Function: Calculate $\phi(n)=(p-1)\times(q-1)$.

$\phi(n)$ represents the number of integers less than n that are relatively prime to n.

- Choose the Public Exponent: Select an integer ee such that $1<e<\phi(n)$ and $\gcd(e,\phi(n))=1$.

The public exponent e is typically chosen as a small prime number like 3 or 65537 for efficiency.

- Compute the Private Exponent: Calculate the private exponent d such that $d\times e\equiv 1 (\mod \phi(n))$.

d is the modular multiplicative inverse of ee modulo $\phi(n)$.

- Public and Private Keys: The public key consists of (n,e). The private key consists of (n,d).

Encryption

- Convert the Message: Convert the plaintext message M into an integer m such that $0\leq m<n$. This can be done using a suitable padding scheme to ensure the message is within the valid range.

- Encrypt the Message: Compute the ciphertext cc using the public key: $c=m^e \mod n$

Decryption

- Decrypt the Ciphertext: Compute the plaintext message m using the private key: $m=c^d \bmod n$

- Convert the Integer Back to Message: Convert the integer m back to the original plaintext message M.

ELGAMAL Encryption Algorithm

The ElGamal encryption algorithm is an asymmetric key encryption algorithm based on the Diffie-Hellman key exchange. It was designed by Taher ElGamal in 1985 and provides both encryption and digital signature functionalities.

Key Components

Public Parameters:

- A large prime number *p*.

- A generator *g* of the multiplicative group of integers modulo *p*.

Private Key:

- A random integer x such that $1 \leq x \leq p-2$.

Public Key:

- Compute $y=g^x \bmod p$.

- The public key is the tuple (p,g,y).

Key Generation

- Choose a large prime number *p*.

- Choose a generator *g* for the multiplicative group of integers modulo *p*.

- Select a private key x where $1 \leq x \leq p-2$
- Compute the public key component y as $y = g^x \bmod p$.

Public key: (p,g,y)

Private key: x

Encryption

- Convert the plaintext message M into an integer m such that $0 \leq m < p$.
- Choose a random integer k such that $1 \leq k \leq p-2$.
- Compute the ciphertext components:
  - $c_1 = g^k \bmod p$
  - $c_2 = m \cdot y^k \bmod p$

Ciphertext: (c_1, c_2)

Decryption

Use the private key *xx*x to compute:

- $s = c_1^x \bmod p$

Compute the plaintext message m as:

- $m = c_2 \cdot s^{-1} \bmod p$

- Here, s^{-1} is the modular inverse of *s* modulo p.

Plaintext: M

Elliptic Curve Cryptography (ECC)

Elliptic Curve Cryptography (ECC) is an asymmetric encryption technique based on the algebraic structure of elliptic curves over finite fields. It offers significant advantages over traditional RSA encryption, such as smaller key sizes for equivalent security levels and faster computational speeds.

Key Components

Elliptic Curve Definition:

An elliptic curve over a finite field Fp is defined by an equation of the form $y^2=x^3+ax+b \bmod p$, where a and b are constants defining the curve's shape, and p is a prime number.

Base Point G (Generator):

- A specific point on the curve used as the base for generating keys and performing operations.

Private Key d:

- A random integer d chosen as the private key.

Public Key Q:

- The public key is computed as $Q=d \cdot G$, where .(dot) denotes elliptic curve point multiplication.

Key Generation

Curve Parameters:

- Choose a suitable elliptic curve (e.g., secp256k1) defined over a finite field Fp .

- Select curve parameters a, b, and a prime modulus p.

Base Point G Selection:

- Choose a base point G on the elliptic curve.

Private Key Generation:

- Generate a random integer *d* such that $1 \leq d <$ order of G.

Public Key Computation:

- Compute the public key Q as $Q = d \cdot G$.

Public key: Q

Private key: *d*

Encryption and Decryption

ECC is primarily used for key exchange rather than direct encryption of messages. The Diffie-Hellman key exchange and elliptic curve DSA (ECDSA) are commonly used protocols based on ECC for secure communication and digital signatures

Lab Exercises:

1. Using RSA, encrypt the message "Asymmetric Encryption" with the public key (n, e). Then decrypt the ciphertext with the private key (n, d) to verify the original message.

2. Using ECC (Elliptic Curve Cryptography), encrypt the message "Secure Transactions" with the public key. Then decrypt the ciphertext with the private key to verify the original message.

3. Given an ElGamal encryption scheme with a public key (p, g, h) and a private key x, encrypt the message "Confidential Data". Then decrypt the ciphertext to retrieve the original message.

4. Design and implement a secure file transfer system using RSA (2048-bit) and ECC (secp256r1 curve) public key algorithms. Generate and exchange keys, then encrypt and decrypt files of varying sizes (e.g., 1 MB, 10 MB) using both algorithms. Measure and compare the performance in terms of key generation time, encryption/decryption speed, and computational overhead. Evaluate the security and efficiency of each algorithm in the context of file transfer, considering factors such as key size, storage requirements, and resistance to known attacks. Document your findings, including performance metrics and a summary of the strengths and weaknesses of RSA and ECC for secure file transfer.

5. As part of a project to enhance the security of communication in a peer-to-peer file sharing system, you are tasked with implementing a secure key exchange mechanism using the Diffie-Hellman algorithm. Each peer must establish a shared secret key with another peer over an insecure channel. Implement the Diffie-Hellman key exchange protocol, enabling peers to generate their public and private keys and securely compute the shared secret key. Measure the time taken for key generation and key exchange processes.

Additional Exercises:

1. With the ElGamal public key ($p = 7919$, $g = 2$, $h = 6465$) and the private key $x = 2999$, encrypt the message "Asymmetric Algorithms". Decrypt the resulting ciphertext to verify the original message.

2. Using ECC (Elliptic Curve Cryptography), encrypt the message "Secure Transactions" with the public key. Then decrypt the ciphertext with the private key to verify the original message.

3. Encrypt the message "Cryptographic Protocols" using the RSA public key (n, e) where n = 323 and e = 5. Decrypt the ciphertext with the private key (n, d) where d = 173 to confirm the original message

4. You are tasked with implementing a secure communication system for a healthcare organization to exchange sensitive patient information securely between doctors and hospitals. Implement the ElGamal encryption scheme to encrypt patient records and medical data, ensuring confidentiality during transmission. Generate public and private keys using the secp256r1 curve and use ElGamal encryption to encrypt patient data with the recipient's public key and decrypt it with the recipient's private key. Measure the performance of encryption and decryption processes for data of varying sizes.

5. You are conducting a study to evaluate the performance and security of RSA and ElGamal encryption algorithms in securing communication for a government agency. Implement both RSA (using 2048-bit keys) and ElGamal (using the secp256r1 curve) encryption schemes to encrypt and decrypt sensitive messages exchanged between agencies. Measure the time taken for key generation, encryption, and decryption processes for messages of various sizes (e.g., 1 KB, 10 KB). Compare the computational efficiency and overhead of RSA and ElGamal algorithms. Perform the same for ECC with RSA and ElGamal.

Lab No. 4: Advanced Asymmetric Key Ciphers

Objectives :

- Implement and compare the performance of multiple asymmetric encryption algorithms (e.g., RSA, ElGamal, Rabin) in a controlled environment, measuring factors such as encryption/decryption speed and key generation time.

- Design and develop a modular key management system capable of handling various cryptographic protocols, with emphasis on scalability, security, and ease of integration.

- Create a flexible framework for testing different access control mechanisms in cryptographic systems, allowing for easy implementation and evaluation of various policies and revocation strategies

Asymmetric Encryption Algorithms: Asymmetric encryption, also known as public-key cryptography, uses a pair of mathematically related keys: a public key for encryption and a private key for decryption. This approach allows secure communication without the need to share secret keys. Common asymmetric algorithms include RSA, ElGamal, and Rabin. Each has its own mathematical foundations and security properties, making comparative analysis valuable for understanding their strengths and weaknesses in different scenarios.

Key Management Systems: Key management is a critical aspect of cryptographic systems, encompassing the generation, exchange, storage, use, and replacement of cryptographic keys. A robust key management system ensures the security and integrity of encrypted communications. It must handle tasks such as key generation, distribution, storage, rotation, and revocation. The challenges in key management increase with the scale and complexity of the system, especially in distributed environments.

Access Control in Cryptographic Systems: Access control in cryptography involves mechanisms to restrict access to encrypted data or cryptographic operations. This includes managing who can encrypt or decrypt data, as well as controlling access to keys. Implementing flexible access control policies is crucial for maintaining security in various scenarios, from simple user authentication to complex, attribute-based access control systems. The ability to revoke access and update policies dynamically is also an important consideration in modern

cryptographic systems.

Mathematical Foundations

RSA Algorithm:

1. Key Generation:

- Choose two large prime numbers p and q

- Compute n = p * q

- Compute φ(n) = (p-1) * (q-1)

- Choose e such that 1 < e < φ(n) and gcd(e, φ(n)) = 1

- Compute d such that d * e ≡ 1 (mod φ(n))

- Public key: (n, e), Private key: (n, d)

2.Encryption:

For plaintext m, ciphertext c = m^e mod n

3. Decryption:

m = c^d mod n

ElGamal Algorithm:

1. Key Generation:

- Choose a large prime p and a generator g of the multiplicative group of integers modulo p

- Choose a random integer x, $1 < x < p-1$

- Compute $y = g^x \bmod p$

- Public key: (p, g, y), Private key: x

2. Encryption:

- Choose a random k, $1 < k < p-1$

- Compute $c1 = g^k \bmod p$

- Compute $s = y^k \bmod p$

- For plaintext m, compute $c2 = m * s \bmod p$

- Ciphertext: (c1, c2)

3. Decryption:

- Compute $s = c1^x \bmod p$

- Compute $m = c2 * s^{(-1)} \bmod p$

Rabin Algorithm:

1. Key Generation:

- Choose two large primes p and q, where $p \equiv q \equiv 3 \pmod 4$

- Compute $n = p * q$

- Public key: n, Private key: (p, q)

2. Encryption:

For plaintext m, ciphertext c = m^2 mod n

3.Decryption:

- Compute mp = c^((p+1)/4) mod p

- Compute mq = c^((q+1)/4) mod q

- Use Chinese Remainder Theorem to find four square roots:

r1 = (yp * p * mq + yq * q * mp) mod n

r2 = n - r1

r3 = (yp * p * mq - yq * q * mp) mod n

r4 = n - r3

where yp * p + yq * q = 1

- One of these roots is the original message m

Each of these algorithms has its own unique mathematical properties that contribute to its security and performance characteristics. RSA relies on the difficulty of factoring large numbers, ElGamal is based on the discrete logarithm problem, and Rabin's security is tied to the difficulty of finding square roots modulo a composite number.

Key Management Systems:

1. Key Entropy:

Entropy H of a key K with n possible values, each with probability p(i):

$H(K) = -\sum_{i=1}^{n} p(i) * \log_2(p(i))$

2. Key Derivation Function (KDF):

DK = KDF(Key, Salt, Iterations)

Where DK is the derived key, Key is the original key or password, Salt is a random value,

and Iterations is the number of times the function is applied.

3. Key Rotation:

For a system with N keys and a rotation period of T:

Rotation Rate = N / T

4. Key Expiry:

If a key K is created at time t0 with lifetime L:

Expiry Time = t0 + L

5. Diffie-Hellman Key Exchange:

Public parameters: prime p, generator g

Alice computes: A = g^a mod p

Bob computes: B = g^b mod p

Shared secret: K = A^b mod p = B^a mod p = g^(ab) mod p

Access Control in Cryptographic Systems:

1. Role-Based Access Control (RBAC):

Access(User, Object) = ∃Role : HasRole(User, Role) ∧ CanAccess(Role, Object)

2. Attribute-Based Access Control (ABAC):

Access(User, Object, Environment) = f(UserAttributes, ObjectAttributes, EnvironmentAttributes)

Where f is a policy function evaluating to true or false.

3. Bell-LaPadula Model:

- Simple Security Property: S(Subject) ≥ C(Object) for read access

- *-Property: S(Subject) ≤ C(Object) for write access

Where S is the security level of the subject and C is the classification of the object.

4. Mandatory Access Control (MAC):

AccessGranted = (SubjectClearance ≥ ObjectClassification) ∧ PolicyRulesatisfied

5. Discretionary Access Control (DAC):

AccessMatrix[Subject, Object] = {Rights}

Where Rights could be Read, Write, Execute, etc.

6. Time-based Access Control:

Access (User, Object, Time) = (Time ≥ StartTime) ∧ (Time ≤ EndTime) ∧ OtherConditions

7. Probabilistic Access Control:

P(Access Granted | Conditions) = f(UserTrustLevel, ObjectSensitivity, EnvironmentRisk)

Where P is probability and f is a function mapping conditions to a probability.

These equations and mathematical concepts form the basis for implementing and analyzing key management and access control systems. They allow for quantitative assessment of security properties, guide the design of secure systems, and provide a framework for evaluating the effectiveness of different approaches in cryptographic access control.

Lab exercises

Question 1

SecureCorp is a large enterprise with multiple subsidiaries and business units located across different geographical regions. As part of their digital transformation initiative, the IT team at SecureCorp has been tasked with building a secure and scalable communication system to enable seamless collaboration and information sharing between their various subsystems.

The enterprise system consists of the following key subsystems:

1. Finance System (System A): Responsible for all financial record-keeping, accounting, and reporting.

2. HR System (System B): Manages employee data, payroll, and personnel-related processes.

3. Supply Chain Management (System C): Coordinates the flow of goods, services, and information across the organization's supply chain.

These subsystems need to communicate securely and exchange critical documents, such as financial reports, employee contracts, and procurement orders, to ensure the enterprise's overall efficiency.

The IT team at SecureCorp has identified the following requirements for the secure communication and document signing solution:

1. Secure Communication: The subsystems must be able to establish secure communication channels using a combination of RSA encryption and Diffie-Hellman key exchange.

2. Key Management: SecureCorp requires a robust key management system to generate, distribute, and revoke keys as needed to maintain the security of the enterprise system.

3. Scalability: The solution must be designed to accommodate the addition of new subsystems in the future as SecureCorp continues to grow and expand its operations.

Implement a Python program which incorporates the requirements.

Question 2:

HealthCare Inc., a leading healthcare provider, has implemented a secure patient data management system using the Rabin cryptosystem. The system allows authorized healthcare professionals to securely access and manage patient records across multiple hospitals and clinics within the organization. Implement a Python-based centralized key management service that can:

- Key Generation: Generate public and private key pairs for each hospital and clinic using the Rabin cryptosystem. The key size should be configurable (e.g., 1024 bits).

- Key Distribution: Provide a secure API for hospitals and clinics to request and receive their public and private key pairs.

- Key Revocation: Implement a process to revoke and update the keys of a hospital or clinic when necessary (e.g., when a facility is closed or

compromised).

- Key Renewal: Automatically renew the keys of all hospitals and clinics at regular intervals (e.g., every 12 months) to maintain the security of the patient data management system.

- Secure Storage: Securely store the private keys of all hospitals and clinics, ensuring that they are not accessible to unauthorized parties.

- Auditing and Logging: Maintain detailed logs of all key management operations, such as key generation, distribution, revocation, and renewal, to enable auditing and compliance reporting.

- Regulatory Compliance: Ensure that the key management service and its operations are compliant with relevant data privacy regulations (e.g., HIPAA).

- Perform a trade-off analysis to compare the workings of Rabin and RSA.

Additional Questions

Question 1

DigiRights Inc. is a leading provider of digital content, including e-books, movies, and music. The company has implemented a secure digital rights management (DRM) system using the ElGamal cryptosystem to protect its valuable digital assets. Implement a Python-based centralized key management and access control service that can:

- Key Generation: Generate a master public-private key pair using the ElGamal cryptosystem. The key size should be configurable (e.g., 2048 bits).

- Content Encryption: Provide an API for content creators to upload their digital content and have it encrypted using the master public key.

- Key Distribution: Manage the distribution of the master private key to authorized customers, allowing them to decrypt the content.

- Access Control: Implement flexible access control mechanisms, such as:
    - Granting limited-time access to customers for specific content
    - Revoking access to customers for specific content
    - Allowing content creators to manage access to their own content
- Key Revocation: Implement a process to revoke the master private key in case of a security breach or other emergency.
- Key Renewal: Automatically renew the master public-private key pair at regular intervals (e.g., every 24 months) to maintain the security of the DRM system.
- Secure Storage: Securely store the master private key, ensuring that it is not accessible to unauthorized parties.
- Auditing and Logging: Maintain detailed logs of all key management and access control operations to enable auditing and troubleshooting.

Question 2

Suppose that XYZ Logistics has decided to use the RSA cryptosystem to secure their sensitive communications. However, the security team at XYZ Logistics has discovered that one of their employees, Eve, has obtained a partial copy of the RSA private key and is attempting to recover the full private key to decrypt the company's communications.

Eve's attack involves exploiting a vulnerability in the RSA key generation process, where the prime factors (p and q) used to generate the modulus (n) are not sufficiently large or random.

Develop a Python script that can demonstrate the attack on the vulnerable RSA cryptosystem and discuss the steps to mitigate the attack.

Lab No.5: Hashing

Objectives

1. To implement user defined hashing function.
2. To demonstrate the application of hash function.

Introduction

A hash function H takes an input data block M of variable length and generates a fixed-size hash value h=H(M). An effective hash function ensures that applying it to a large set of inputs yields outputs that are evenly distributed and seemingly random. The primary goal of a hash function is to maintain data integrity, such that any change in the input M, even by a single bit, will likely result in a different hash value.

For security purposes, a special type of hash function, known as a cryptographic hash function, is used. This algorithm has two key properties:

- The one-way property, which makes it computationally infeasible to find a data object that matches a pre-specified hash result.
- The collision-free property, which makes it computationally infeasible to find two distinct data objects that produce the same hash result.

Due to these properties, cryptographic hash functions are commonly employed to verify whether data has been altered.

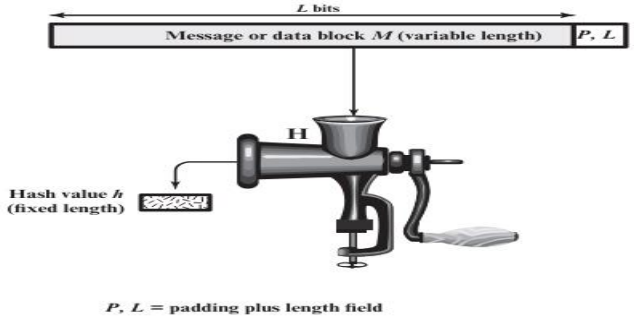


Figure 5.1 Cryptographic Hash Function

Figure 5.1 illustrates how a cryptographic hash function operates. Generally, the input data is padded to make its length an integer multiple of a specific fixed size (e.g., 1024 bits). This padding includes a field that represents the length of the original message in bits. The length field acts as a security measure, making it more challenging for an attacker to create a different message with the same hash value. [Ref: Cryptography and Network Security: Principles and Practice 7th Global Edition]

Lab Exercises

1. Implement the hash function in Python. Your function should start with an initial hash value of 5381 and for each character in the input string, multiply the current hash value by 33, add the ASCII value of the character, and use bitwise operations to ensure thorough mixing of the bits. Finally, ensure the hash value is kept within a 32-bit range by applying an appropriate mask.
2. Using socket programming in Python, demonstrate the application of hash functions for ensuring data integrity during transmission over a network. Write server and client scripts where the server computes the hash of received data and sends it back to the client, which then verifies the integrity of the data by comparing the received hash with the locally computed hash. Show how the hash verification detects data corruption or tampering during transmission.
3. Design a Python-based experiment to analyze the performance of MD5, SHA-1, and SHA-256 hashing techniques in terms of computation time and collision resistance. Generate a dataset of random strings ranging from 50 to 100 strings, compute the hash values using each hashing technique, and measure the time taken for hash computation. Implement collision detection algorithms to identify any collisions within the hashed dataset.

Additional Exercise

1. Write server and client scripts where the client sends a message in multiple parts to the server, the server reassembles the message, computes the hash of the reassembled message, and sends this hash back to the client. The client then verifies the integrity of the message by comparing the received hash with the locally computed hash of the original message.

Lab No. 6: Digital Signature

Objectives

- Demonstrate the use of digital signatures
- Demonstrate the verification of a digital signature

Background

A digital signature is a mathematical technique used to validate the authenticity and integrity of a digital message. A digital signature is the equivalent of a handwritten signature. Digital signatures can actually be far more secure. The purpose of a digital signature is to prevent the tampering and impersonation in digital communications.

Using Digital Signatures

In this part, you will use a website to verify a document signature between Alice and Bob. Alice and Bob share a pair of private and public RSA keys. Each of them uses their private key to sign a legal document. They then send the documents to each other. Both Alice and Bob can verify each other's signature with the public key. They must also agree on a shared public exponent for calculation

Table 1 – RSA Public and Private Keys

Public RSA Key	d94d889e88853dd89769a18015a0a2e6bf82bf356fe14f251fb4f5e2df0d9f9a94a68a30c428b39e3362fb3779a497eceaea37100f264d7fb9fb1a97fbf621133de55fdcb9b1ad0d7a31b379216d79252f5c527b9bc63d83d4ecf4d1d45cbf843e8474babc655e9bb6799cba77a47eafa838296474afc24beb9c825b73ebf549
Private RSA Key	47b9cfde843176b88741d68cf096952e950813151058ce46f2b048791a26e507a1095793c12bae1e09d82213ad9326928cf7c2350acb19c98f19d32d577d666cd7bb8b2b5ba629d25ccf72a5ceb8a8da038906c84dcdb1fe677dffb2c029fd8926318eede1b58272af22bda5c5232be066839398e42f5352df58848adad11a1
Public Exponent	10001

Step 1: Sign the Document.

Alice signs a legal document and send it to Bob using the RSA public and private keys shown in the table above. Now Bob will have to verify Alice's digital signature in order to trust the authenticity of the electronic document.

Step 2: Verify Digital Signature.

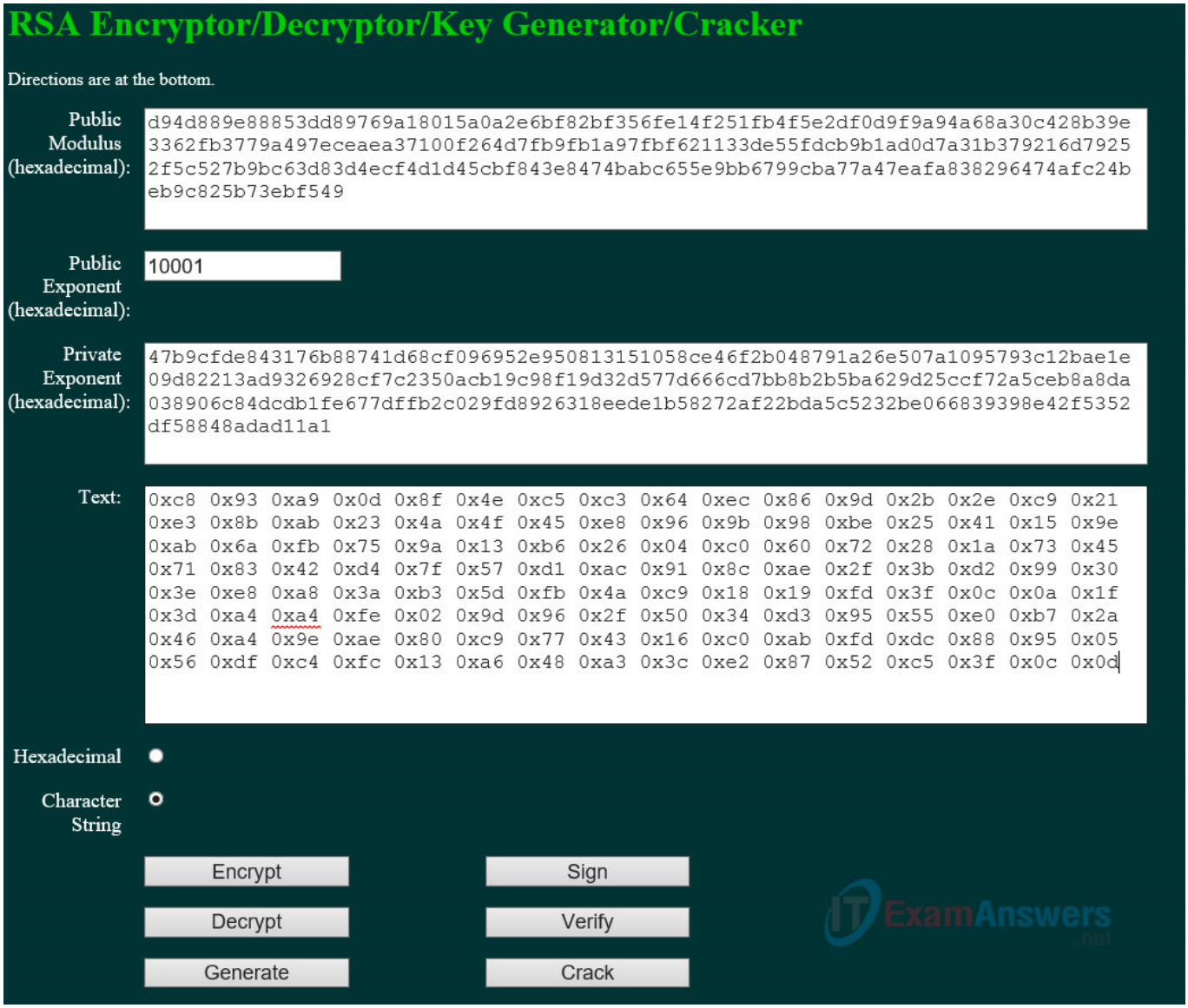
Bob receives the document with a digital signature shown in the table below.

Table 2 – Alice's Digital Signature

Alice's Digital Signature
0xc8 0x93 0xa9 0x0d 0x8f 0x4e 0xc5 0xc3 0x64 0xec 0x86 0x9d 0x2b 0x2e 0xc9 0x21 0xe3 0x8b 0xab 0x23 0x4a 0x4f 0x45 0xe8 0x96 0x9b 0x98 0xbe 0x25 0x41 0x15 0x9e 0xab 0x6a 0xfb 0x75 0x9a 0x13 0xb6 0x26 0x04 0xc0 0x60 0x72 0x28 0x1a 0x73 0x45 0x71 0x83 0x42 0xd4 0x7f 0x57 0xd1 0xac 0x91 0x8c 0xae 0x2f 0x3b 0xd2 0x99 0x30 0x3e 0xe8 0xa8 0x3a 0xb3 0x5d 0xfb 0x4a 0xc9 0x18 0x19 0xfd 0x3f 0x0c 0x0a 0x1f 0x3d 0xa4 0xa4 0xfe 0x02 0x9d 0x96 0x2f 0x50 0x34 0xd3 0x95 0x55 0xe0 0xb7 0x2a 0x46 0xa4 0x9e 0xae 0x80 0xc9 0x77 0x43 0x16 0xc0 0xab 0xfd 0xdc 0x88 0x95 0x05 0x56 0xdf 0xc4 0xfc 0x13 0xa6 0x48 0xa3 0x3c 0xe2 0x87 0x52 0xc5 0x3f 0x0c 0x0d

Click here to use the online RSA tool to verify the authenticity of Alice's digital signature.

Table 3 – Online Digital Signature Tool



a. Copy and paste the **public** and **private** keys from Table 1 above into the **Public Modulus** and **Private Exponent** boxes on the website as shown in the picture above.

b. Make sure the Public Exponent is 10001.

c. Paste Alice's digital signature from Table 2 in the box labeled text on the website as shown above.

d. Now BOB can verify the digital signature by clicking the **Verify** button near the bottom center of the website. Whose signature is identified?

Alice's name should be displayed.

Step 3: Generate a Response Signature.

Bob receives and verifies Alice's electronic document and digital signature. Now Bob creates an electronic document and generates his own digital signature using the private RSA Key in Table 1 (Note: Bob's name is in all capital letters).

Table 4 – BOB Digital Signature

BOB's Digital Signature
0x6c 0x99 0xd6 0xa8 0x42 0x53 0xee 0xb5 0x2d 0x7f 0x0b 0x27 0x17 0xf1 0x1b 0x62 0x92 0x7f 0x92 0x6d 0x42 0xbd 0xc6 0xd5 0x3e 0x5c 0xe9 0xb5 0xd2 0x96 0xad 0x22 0x5d 0x18 0x64 0xf3 0x89 0x52 0x08 0x62 0xe2 0xa2 0x91 0x47 0x94 0xe8 0x75 0xce 0x02 0xf8 0xe9 0xf8 0x49 0x72 0x20 0x12 0xe2 0xac 0x99 0x25 0x9a 0x27 0xe0 0x99 0x38 0x54 0x54 0x93 0x06 0x97 0x71 0x69 0xb1 0xb6 0x24 0xed 0x1c 0x89 0x62 0x3d 0xd2 0xdf 0xda 0x7a 0x0b 0xd3 0x36 0x37 0xa3 0xcb 0x32 0xbb 0x1d 0x5e 0x13 0xbc 0xca 0x78 0x3e 0xe6 0xfc 0x5a 0x81 0x66 0x4e 0xa0 0x66 0xce 0xb3 0x1b 0x93 0x32 0x2c 0x91 0x4c 0x58 0xbf 0xff 0xd8 0x97 0x2f 0xa8 0x57 0xd7 0x49 0x93 0xb1 0x62

Bob sends the electronic document and digital signature to Alice.

Step 4: Verify Digital Signature.

a. Copy and paste the **public** and **private** keys from Table 1 above into the **Public Modulus** and **Private Exponent** boxes on the website as shown in the picture above.

b. Make sure the Public Exponent is 10001.

c. Paste Bob's digital signature from Table 4 in the box labeled text on the website as shown above.

d. Now Alice can verify the digital signature by clicking the **Verify** button near the bottom center of the website. Whose signature is identified?

Bob's name should be displayed.

Part 2: Create Your Own Digital Signature

Now that you see how digital signatures work, you can create your own digital signature.

Step 1: Generate a New Pair of RSA Keys.

Go to the website tool and generate a new set of RSA public and private keys.

a. Delete the contents of the boxes labeled **Public Modulus**, **Private Modulus** and **Text**. Just use your mouse to highlight the text and press the delete key on your keyboard.

b. Make sure the "Public Exponent" box has **10001**.

c. Generate a new set of RSA keys by clicking the **Generate** button near the bottom right of the website.

d. Copy the new keys in Table 5.

Table 5 – New RSA Keys

Public Key	A string of 256 hexadecimal characters will be displayed for both the public and private keys. The keys will be different from one another.

Private key	A string of 256 hexadecimal characters will be displayed for both the public and private keys. The keys will be different from one another.

e. Now type in your full name into the box labeled **Text** and click **Sign**.

Table 6 – Personal Digital Signature

Personal Digital Signature	A string of characters with this format will be displayed. 0x23 0x90 …….

Part 3: Exchange and Verify Digital Signatures

Now you can use this digital signature.

Step 1: Exchange your new public and private keys in Table-5 with your lab partner.

a. Record your lab partner's public and private RSA keys from their Table-5.

b. Record both keys in the table below.

Table 7- Lab Partners RSA Keys

Public key	A string of 256 hexadecimal characters will be displayed for both the public and private keys. The keys will be different from one another.
Private key	A string of 256 hexadecimal characters will be displayed for both the public and private keys. The keys will be different from one another.

c. Now exchange their digital signature from their Table-6. Record the digital signature in the table below.

Lab Partner's Digital Signature	A string of characters with this format will be displayed. 0x23 0x90 …….

Step 2: Verify Lab Partners Digital Signature

a. To verify your lab partner's digital signature, paste his or her public and private keys in the appropriate boxes labeled **Public and Private modulus** on the website.

b. Now paste the digital signature in the box labeled **Text**.

c. Now verify his or her digital signature by clicking the button labeled verify.

d. What shows up in the Text box?

Answers will vary.

Lab Exercises

1. Try using the Elgammal, Schnor asymmetric encryption standard and verify the above steps.
2. Try using the Diffie-Hellman asymmetric encryption standard and verify the above steps.
3. Try the same in a client server-based scenario and record your observation and analysis.

Additional Exercise

1. Explore the link https://www.nmichaels.org/rsa.py for better understanding.

   Demonstrate CIA traid using RSA encryption and digital signature along with SHA hashing

Lab No. 7: Partial Homomorphic Encryption

Objectives

- Implement and Understand Additive Homomorphic Encryption
- Implement and Understand Multiplicative Homomorphic Encryption
- Evaluate and Apply Partial Homomorphic Encryption

Introduction to Homomorphic Encryption

Homomorphic encryption allows computations on encrypted data without decryption. This enables secure analysis of sensitive information while preserving privacy. There are two main types of HE:

- **Partially Homomorphic Encryption (PHE):** Supports either addition or multiplication homomorphically.
- **Fully Homomorphic Encryption (FHE):** Supports both addition and multiplication homomorphically.

Partially Homomorphic Encryption (PHE)

PHE allows performing either addition or multiplication on encrypted data. Some evident PHE schemes include Paillier and ElGamal.

Homomorphic Addition (Paillier Cryptosystem)

This exercise demonstrates homomorphic addition using the Paillier Cryptosystem.

Problem: Add two encrypted integers (a and b).

This program demonstrates how to perform homomorphic addition on encrypted integers using the Paillier Cryptosystem.

Key functionalities and Explanations:

- generate_keypair: Generates a public/private key pair for encryption and

decryption.

- encrypt: Encrypts a message (integer) using the public key and random blinding for security.
- decrypt: Decrypts a ciphertext using the private key (optional, for demonstration only).
- homomorphic_add: Performs homomorphic addition on two encrypted messages. This works because multiplying encrypted messages under Paillier corresponds to adding the original messages.

Python Code:

Python

```
from Crypto.PublicKey import RSA
from Crypto.Random import get_random_bytes

def generate_keypair(nlength=1024):
  """Generates a public/private key pair"""
  key = RSA.generate(nlength)
  pub_key = key.publickey()
  return pub_key, key

def encrypt(pub_key, message):
  """Encrypts a message using the public key"""
  random_bytes = get_random_bytes(16)
  p, q = pub_key.n // 2, pub_key.n // 2 + 1
  while math.gcd(p, q) != 1:
    p, q = pub_key.n // 2, pub_key.n // 2 + 1
  m_dot = pow(message, 2, pub_key.n)
  r_dot = pow(int.from_bytes(random_bytes, byteorder='big'), 2, pub_key.n)
  ciphertext = m_dot * r_dot % pub_key.n
  return ciphertext

def decrypt(priv_key, ciphertext):
  """Decrypts a ciphertext using the private key"""
  p = priv_key.n // 2
```

```
  l = (ciphertext - 1) // pub_key.n
  message = math.floor(l * pow(p, -1, priv_key.n))
  return message

def homomorphic_add(ciphertext1, ciphertext2, pub_key):
  """Performs homomorphic addition on ciphertexts"""
  return ciphertext1 * ciphertext2 % pub_key.n

# Generate key pair
pub_key, priv_key = generate_keypair()

# Encrypt integers
a = 5
b = 10
ciphertext_a = encrypt(pub_key, a)
ciphertext_b = encrypt(pub_key, b)

# Homomorphic addition
ciphertext_sum = homomorphic_add(ciphertext_a, ciphertext_b, pub_key)

# Decrypt the sum (optional)
# decrypted_sum = decrypt(priv_key, ciphertext_sum)
# print(f"Decrypted sum: {decrypted_sum}")

print(f"Ciphertext of a: {ciphertext_a}")
print(f"Ciphertext of b: {ciphertext_b}")
print(f"Ciphertext of a + b: {ciphertext_sum}")
```

Secure Medical Diagnosis (ElGamal Cryptosystem)

Explanation:

ElGamal uses a public key for encryption and a private key for decryption.

encrypt creates a random blinding factor and encrypts the message.

homomorphic_comparison multiplies ciphertexts with additional manipulation to achieve encrypted comparison (>).

Decrypting the comparison result (optional) reveals if the first message was greater than the second.

The diagnosis is made based on the decrypted comparison result (demonstration only, not recommended in practice).

Problem: Perform a secure diagnosis on encrypted patient data (blood pressure).

- A doctor wants to diagnose patients with high blood pressure (> 130) without decrypting their actual blood pressure readings.
- Use ElGamal encryption, which supports homomorphic comparison.

Python Code:

Python

```
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

def generate_keypair(p=1024):
  """Generates a public/private key pair"""
  while True:
    x = int.from_bytes(get_random_bytes(p // 8), byteorder='big')
    if pow(2, p - 1, p) == 1 and 1 < x < p-1:
      break
  g = 2
  y = pow(g, x, p)
  return ((p, g, y), x)  # Public key, private key

def encrypt(pub_key, message):
  """Encrypts a message using the public key"""
  p, g, y = pub_key
```

```
  r = int.from_bytes(get_random_bytes(p // 8), byteorder='big')
  while math.gcd(r, p) != 1:
    r = int.from_bytes(get_random_bytes(p // 8), byteorder='big')
  a = pow(g, r, p)
  b = (message * pow(y, r, p)) % p
  return (a, b)

def decrypt(priv_key, ciphertext):
  """Decrypts a ciphertext using the private key"""
  p, g, _ = priv_key
  a, b = ciphertext
  x = priv_key
  message = (b * pow(a, -x, p)) % p
  return message

def homomorphic_comparison(ciphertext1, ciphertext2, pub_key):
  """Performs homomorphic comparison on ciphertexts (greater than)"""
  p, g, y = pub_key
  a1, b1 = ciphertext1
  a2, b2 = ciphertext2
  return (a1 * a2) % p, (b1 * b2 * pow(y, 1, p)) % p  # Encrypted result (m1 > m2)

# Generate key pair
pub_key, priv_key = generate_keypair()

# Blood pressure readings (already encrypted)
blood_pressure1 = encrypt(pub_key, 120)
blood_pressure2 = encrypt(pub_key, 140)

# Homomorphic comparison (encrypted result)
ciphertext_comparison = homomorphic_comparison(blood_pressure1, blood_pressure2,
pub_key)

# Decrypt the comparison result (optional - for demonstration only)
```

```
# decrypted_comparison = decrypt(priv_key, ciphertext_comparison)
# print(f"Decrypted comparison: {decrypted_comparison} (True if blood pressure 1 > blood pressure 2)")

# Diagnosis based on the encrypted comparison result
diagnosis = ciphertext_comparison[0] * pow(ciphertext_comparison[1], -1, pub_key[0]) % pub_key[0]
if diagnosis > 1:
  print("Diagnosis: High Blood Pressure detected.")
else:
  print("Diagnosis: Normal Blood Pressure.")
```

Key functionalities:

- generate_keypair: Generates a public/private key pair for ElGamal encryption.
- encrypt: Encrypts a message (blood pressure reading) using the public key and a random factor.
- homomorphic_comparison: Performs homomorphic comparison on two encrypted messages. This leverages ElGamal's properties to create an encrypted result indicating if one message is greater than the other.
- decrypt (not recommended): While included for demonstration, decrypting the comparison result in a real-world scenario would reveal sensitive information. Diagnosis should rely on the encrypted comparison itself.
- Explore additional libraries like PALISADE (https://palisade-crypto.org/) for more advanced PHE functionalities.

Software Requirements:

- Python 3.x (https://www.python.org/downloads/)
- NumPy library (https://numpy.org/)

Lab Exercises

1. Implement the Paillier encryption scheme in Python. Encrypt two integers (e.g., 15 and 25) using your implementation of the Paillier encryption scheme. Print the ciphertexts. Perform an addition operation on the encrypted integers without decrypting them. Print the result of the addition in encrypted form. Decrypt the result of the addition and verify that it matches the sum of the original integers.

2. Utilize the multiplicative homomorphic property of RSA encryption. Implement a basic RSA encryption scheme in Python. Encrypt two integers (e.g., 7 and 3) using your implementation of the RSA encryption scheme. Print the ciphertexts. Perform a multiplication operation on the encrypted integers without decrypting them. Print the result of the multiplication in encrypted form. Decrypt the result of the multiplication and verify that it matches the product of the original integers.

Additional Questions

Implement similar exercise for other PHE operations (like homomorphic multiplication using ElGamal) or explore different functionalities within Paillier.

1a: Homomorphic Multiplication (ElGamal Cryptosystem): Implement ElGamal encryption and demonstrate homomorphic multiplication on encrypted messages. (ElGamal supports multiplication but not homomorphic addition.)
1b: Secure Data Sharing (Paillier): Simulate a scenario where two parties share encrypted data and perform calculations on the combined data without decryption.
1c: Secure Thresholding (PHE): Explore how PHE can be used for secure multi-party computation, where a certain number of parties need to collaborate on a computation without revealing their individual data.
1d: Performance Analysis (Benchmarking): Compare the performance of different PHE schemes (Paillier and ElGamal) for various operations.

Lab No. 8: Searchable Encryption

Objectives

Understand the Fundamentals of Searchable Encryption (SE)

Implement and Perform Encrypted Data Searches

Analyze the Security and Efficiency Trade-offs

Searchable encryption (SE)

It is a cryptographic technique that enables efficient search operations on encrypted data without compromising its confidentiality. [1] This paradigm is essential in modern data management, where data privacy and utility are often conflicting objectives. SE offers a solution by allowing authorized users to search for specific information within encrypted datasets without revealing any underlying data.

Core Concepts used in the SE

- **Encryption:** The foundational layer, where data is transformed into an unreadable format using cryptographic algorithms. Standard symmetric or asymmetric encryption techniques can be employed.
- **Indexing:** A crucial component for enabling search. Data is indexed using cryptographic primitives to generate searchable representations, often termed "encrypted indexes." These indexes must allow for efficient search operations without exposing the underlying data.
- **Query Processing:** When a search query is issued, it is encrypted and compared against the encrypted indexes. Matching results are identified, and their encrypted locations or identifiers are returned. The actual data remains encrypted until accessed by an authorized user with the decryption key.

Categories of SE

Symmetric Searchable Encryption (SSE) is a cryptographic primitive designed to reconcile the conflicting objectives of data confidentiality and searchability. In SSE, a shared secret key is employed to encrypt data and construct searchable indexes. This approach enables authorized parties to perform search operations over encrypted data without compromising the underlying information. While SSE offers advantages in terms of efficiency compared to its public-key counterpart, it necessitates careful design to prevent information leakage through the search process and address challenges related to key management and distribution.
Major Points

- SSE employs a shared secret key for both encryption and decryption.
- SSE offers relatively efficient search performance but suffers from crucial management challenges.
- Some of the SSE schemes include the Boneh-Goh-Nissim (BGN) and the Cuccioletta-Damiani-Di-Crescenzo (CDD) schemes.

Public-key searchable Encryption (PKSE) is a cryptographic technique that enables search operations on encrypted data while maintaining data confidentiality using asymmetric cryptography. Unlike Symmetric Searchable Encryption (SSE), PKSE employs a public-key/private-key pair for encryption and decryption. PKSE allows for flexible access control, as multiple users can encrypt data using the public key, while only the holder of the corresponding private key can decrypt and search the data. PKSE offers greater flexibility in key management than SSE but often incurs higher computational overhead due to public-key operations.
Major Points

- PKSE uses public-key cryptography, where data is encrypted using a public key and decrypted with the corresponding private key.
- PKSE provides greater flexibility in key management and access control.
- Typically, less efficient than SSE due to the computational overhead of public-key operations.

SSE sample code

```
import random

from Crypto.Cipher import AES

from Crypto.Random import get_random_bytes
```

```
from Crypto.Util.Padding import pad, unpad
import hashlib

def encrypt_data(key, data):
  cipher = AES.new(key, AES.MODE_CBC)
  iv = cipher.iv
  ciphertext = cipher.encrypt(pad(data.encode(), AES.block_size))
  return iv, ciphertext

def decrypt_data(key, iv, ciphertext):
  cipher = AES.new(key, AES.MODE_CBC, iv)
  plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)
  return plaintext.decode()

def create_index(documents, key):
  index = {}
  for doc_id, doc in documents.items():
    for word in doc.split():
      word_hash = hashlib.sha256(word.encode()).digest()
      if word_hash not in index:
        index[word_hash] = []
      index[word_hash].append(doc_id)
  # Encrypt the index
  encrypted_index = {}
  for word_hash, doc_ids in index.items():
    encrypted_index[encrypt_data(key, word_hash)[1]] = [encrypt_data(key, str(doc_id))[1]
for doc_id in doc_ids]
```

```
  return encrypted_index

def search(encrypted_index, query, key):
  query_hash = hashlib.sha256(query.encode()).digest()
  encrypted_query_hash = encrypt_data(key, query_hash)[1]
  if encrypted_query_hash in encrypted_index:
    return [decrypt_data(key, *encrypt_data(key, doc_id))[0] for doc_id in
encrypted_index[encrypted_query_hash]]
  else:
    return []

# Example usage
documents = {
  "doc1": "this is a document with some words",
  "doc2": "another document with different words",
  "doc3": "yet another document with some common words"
}

key = get_random_bytes(16)

encrypted_index = create_index(documents, key)

query = "document"
results = search(encrypted_index, query, key)
print(results)
```

PKSE Sample Code:

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Hash import SHA256
import random
def generate_keys():
keyPair = RSA.generate(2048)
pubKey = keyPair.publickey()
privKey = keyPair
return pubKey, privKey
def encrypt_data(pubKey, data):
cipher = PKCS1_OAEP.new(pubKey)
ciphertext = cipher.encrypt(data.encode())
return ciphertext
def decrypt_data(privKey, ciphertext):
cipher = PKCS1_OAEP.new(privKey)
plaintext = cipher.decrypt(ciphertext)
return plaintext.decode()
def create_index(documents, pubKey):
index = {}
for doc_id, doc in documents.items():
for word in doc.split():
word_hash = SHA256.new(word.encode()).digest()
if word_hash not in index:
index[word_hash] = []
index[word_hash].append(doc_id)
```

```
# Encrypt the index
encrypted_index = {}
for word_hash, doc_ids in index.items():
encrypted_index[encrypt_data(pubKey, word_hash)] = [encrypt_data(pubKey, str(doc_id)) for doc_id in doc_ids]
return encrypted_index
def search(encrypted_index, query, pubKey, privKey):
query_hash = SHA256.new(query.encode()).digest()
encrypted_query_hash = encrypt_data(pubKey, query_hash)
if encrypted_query_hash in encrypted_index:
encrypted_doc_ids = encrypted_index[encrypted_query_hash]
doc_ids = [decrypt_data(privKey, doc_id) for doc_id in encrypted_doc_ids]
return doc_ids
else:
return []
# Example usage
documents = {
"doc1": "this is a document with some words",
"doc2": "another document with different words",
"doc3": "yet another document with some common words"
}
pubKey, privKey = generate_keys()
encrypted_index = create_index(documents, pubKey)
query = "document"
results = search(encrypted_index, query, pubKey, privKey)
print(results)
```

_____

Lab Exercise 1: Execute the following for SSE:

1a. Create a dataset: Generate a text corpus of at least ten documents. Each document should contain multiple words.

1b. Implement encryption and decryption functions: Use the AES encryption and decryption functions.

1c. Create an inverted index: Build an inverted index mapping word to the list of document IDs containing those words.

- Encrypt the index using the provided encryption function.

1d. Implement the search function:

- Take a search query as input.
- Encrypt the query.
- Search the encrypted index for matching terms.
- Decrypt the returned document IDs and display the corresponding documents

Lab Exercise 2: Execute the following for PKSE:

2a. Create a dataset:

- Generate a text corpus of at least ten documents. Each document should contain multiple words.

2b. Implement encryption and decryption functions:

- Use the Paillier cryptosystem for encryption and decryption.

2c. Create an encrypted index:

- Build an inverted index mapping word to the list of document IDs containing those words.

- Encrypt the index using the Paillier cryptosystem.

2d. Implement the search function:

- Take a search query as input.
- Encrypt the query using the public key.
- Search the encrypted index for matching terms.
- Decrypt the returned document IDs using the private key.

Additional Questions:

1. Demonstrate how to securely store and transmit data using GnuPG. Additionally, show how to create a digital signature for the data and verify the signature after transmission.
2. Configure and use Snort as a Network Intrusion Detection System (NIDS) to monitor real-time network traffic. Capture network traffic, apply Snort rules, and analyze the logs to identify any potential intrusions.