Title:Cryptography Analysis and Implementation

SUBMITTED BY: ANUJ GUPTA
20BKT0073
VIT VELLORE (CSE With SPECIALIZATION IN BLOCKCHAIN TECHNOLOGY)

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

Analysis:

1- Symmetric algorithm

BLOWFISH ALGORITHM

The Blowfish algorithm is a symmetric key block cipher that was designed by Bruce Schneier in 1993. It is known for its simplicity and flexibility.

Algorithm Operation:

Key Generation:

Blowfish allows keys with a configurable length that can be between 32 and 448 bits. The state of the Blowfish algorithm, including its S-boxes (substitution boxes) and P-array (permutation array), is initialised using the key.

Subkey Generation:

A key expansion procedure modifies the initial state. Each of the 32-bit blocks of the key is XORed with a different P-array element. The key is divided into a number of subkeys by this process, which are then utilised for encryption and decryption.

Encryption and Decryption:

Blowfish encrypts and decrypts data in blocks of 64 bits. The input block is split into left and right halves during encryption. The data is subjected to a number of rounds, which typically range from 16 to 20 rounds. Utilising the S-boxes and P-array, each round consists of a number of operations, such as substitution and permutation. Next, each round's outcome is XORed with the other half. For each round, the procedure is repeated. The rounds are applied in reverse order for decryption.

Key Strengths and Advantages of Blowfish:

Security: When employed with the right key size, Blowfish is thought to be resistant to the majority of known attacks. To differential and linear cryptanalysis, it stands up well.

Speed: When compared to certain other encryption techniques, Blowfish is fairly quick. Due to its effectiveness, it can be used in applications that need for rapid encryption and decryption procedures.

Simpleness: The Blowfish algorithm is accessible to developers since it is not too difficult to comprehend and use. Its simplicity lowers the possibility of implementation mistakes.

Known Weaknesses or Vulnerabilities:

Blowfish allows for key sizes ranging from 32 to 448 bits, but smaller key sizes may be more vulnerable to brute-force assaults. Therefore, for greater security, longer key sizes are typically advised.

Limited Usage: Due to its age and lack of extensive inspection, Blowfish has mainly been supplanted by more recent encryption algorithms, such as AES. More thorough investigation has been done on newer algorithms, which are thought to be more secure.

Real-World Applications:

Virtual Private Networks (VPNs):
Blowfish has been employed in several VPN implementations to encrypt communication between remote clients and the VPN server. However, contemporary VPNs frequently use more powerful algorithms, such AES.

Disk Encryption:
Disc encryption software uses the Blowfish algorithm to safeguard the secrecy of data kept on hard drives or other storage devices. But today, more advanced and secure algorithms like AES are frequently employed for this reason.

Secure File Transfer:
Blowfish has been used in a number of file transfer protocols and apps to guarantee the confidentiality of sent files. However, more recent methods are now frequently used for secure file transfer, such as AES or the SSH protocol (which allows several encryption techniques).

2- Asymmetric algorithm

ELGAMAL ALGORITHM:

ElGamal is a public-key cryptosystem that bears Taher Elgamal's name, who created it. It is frequently utilised for data encryption and secure communication.

Algorithm Operation:

The ElGamal algorithm is based on the mathematics of modular exponentiation and the discrete logarithm problem. The algorithm involves the following steps:

Key Generation:

Select a large prime number, p, and a primitive root modulo p, g.
Choose a private key, a, which is a random integer between 1 and p-2.
Calculate the corresponding public key, A, as A = g^a mod p.

Encryption:

Convert the plaintext message into a numerical representation, m, such that $0 \leq m < p$.
Choose a random number, k, between 1 and p-2.
Compute the ciphertext as follows:
The first part of the ciphertext is c1 = g^k mod p.
The second part of the ciphertext is c2 = (A^k * m) mod p.

Decryption:

To decrypt the ciphertext (c1, c2), the recipient uses the private key, a, as follows:
Calculate the shared secret as s = (c1^a) mod p.
Compute the plaintext message as m = (c2 * s^(-1)) mod p, where s^(-1) is the modular inverse of s modulo p.

Principal Benefits and Strengths:

Security:

ElGamal offers semantic security, which prevents an attacker from deciphering the ciphertext without access to the private key.

Public-key encryption:

This encryption method employs a pair of keys, one of which is used for encryption and the other for decryption. This makes it possible for two people who have never before exchanged a secret key to communicate securely.

ElGamal allows for safe key exchange between two parties via an insecure channel and is based on the Diffie-Hellman protocol.

Asymmetric encryption makes it computationally challenging for an attacker to deduce the private key from the public key since the encryption and decryption procedures use distinct keys.

Weaknesses and Vulnerabilities:

Performance:
ElGamal encryption is less effective for large-scale data encryption since it is slower than symmetric encryption techniques.

Key management:
Because the method needs high key sizes to provide adequate security, it can result in more computational work and storage space being needed.

Chosen ciphertext attacks are possible if an attacker has the skills to manipulate ciphertexts and view the accompanying plaintexts. If this is the case, they can take advantage of flaws in the algorithm.

Real-world Illustrations

Secure communication: To guarantee the confidentiality and authentication of transmitted data, ElGamal is frequently used in email encryption, secure messaging programmes, and virtual private networks (VPNs).

ElGamal can be used to create digital signatures, offering a safe mechanism to confirm the integrity and authenticity of digital documents.

ElGamal contributes to cryptographic protocols that let several parties calculate a joint function while maintaining the privacy of their inputs. This is known as secure multi-party computation.

3- hash function

SHA 256 ALGORITHM

A cryptographic hash function that is a member of the SHA-2 (Secure Hash Algorithm 2) family is called SHA-256 (Secure Hash Algorithm 256-bit). It is frequently used in cybersecurity and cryptography for a variety of tasks like password hashing and data integrity checking.

An explanation of the algorithm's operation:

SHA-256 processes messages in blocks of 512 bits for message padding. If the message's length is not a multiple of 512 bits, it is padded with a one-bit, then zeros, and finally the message's original length is added.

Initialization:
The initialization vector (IV), also known as the initial hash value, and a set of specified constants are used to start the process.

Calculating the message digest:
512-bit blocks are used to divide up the padded message. The hash value is updated for each block using a combination of logical operations, bitwise operations, and modular addition operations.

Finalisation:

The message digest is the value of the resulting hash after all the blocks have been processed. A 256-bit (32-byte) value indicating the integrity of the original message is produced as the output.

Key benefits and strengths of SHA-256 include:

Collision Resistance:
Finding two different inputs that create the same hash value (collision) is computationally impossible. The integrity of the data is ensured by this feature.

Deterministic:
The SHA-256 algorithm consistently generates the same output from the same input. Data integrity may be easily verified thanks to this characteristic.
Widely Supported: Many cryptographic libraries and systems have adopted and support SHA-256.

Efficient:
SHA-256 is a computationally costly method, however in comparison to some other cryptographic hash functions, it is reasonably efficient.

Known flaws or vulnerabilities

Length Extension Attacks:
SHA-256 is vulnerable to these attacks, which allow an attacker to add more data to a message that has been hashed without knowing the original message. Utilising HMAC (Hash-based Message Authentication Code) structures can reduce this issue.

Future developments in quantum computing may be able to undermine the mathematical foundations of the SHA-256 algorithm and other comparable encryption methods. Post-quantum cryptography techniques are being created to overcome this issue.

Following are some instances of real-world applications of SHA-256:

Blockchain Technology:
To guarantee the security and integrity of transactions and blocks, blockchain systems like Bitcoin and Ethereum heavily rely on the SHA-256 algorithm.

Digital Signatures:
The message is hashed using SHA-256 in digital signature systems like RSA and ECDSA before being signed with the private key. Data integrity is provided by the hash function, which makes sure that any changes to the message will produce a different hash.

Password Storage:
Password hashing is used to safely store user passwords in databases. SHA-256 is typically used in conjunction with salts. During authentication, the hashed password is hashed once more and compared to the hash that was previously stored.

PART 2:

ALGORITHM IMPLEMENTED AND EXPLAINED IS ELGAMAL

ELGAMAL

Scenario:

Two parties communicate securely

Problem:

To transmit important information, Alice and Bob need to interact safely across an unsecure route. They aim to guarantee transmission integrity and secrecy.

ElGamal cryptography solution

Generating a Key:

Alice creates a corresponding public key (YA) and private key (XA).
Bob creates his appropriate public key (YB) and private key (XB).
Public key exchange between Alice and Bob is done safely.
Encryption:

Alice wants to send Bob a plaintext message that is private.
Alice determines C1 = (YBk)% q, where q is a prime number and YB is Bob's public key, using a one-time key (k) she chooses at random.
The shared secret key (shared_secret) is calculated by Alice as shared_secret = (YBXA)% q.
Alice changes every character in the plaintext to its matching ASCII value (con).
Alice uses the formula C2 = (shared_secret * con)% q to calculate the ciphertext (C2) for each character.
Bob receives the pair (C1, C2) from Alice.
Decryption:

The ciphertext (C1, C2) is given to Bob by Alice.
The shared secret key (shared_secret) is calculated by Bob as shared_secret = (C1XB)% q.
Bob gets the original plaintext character for each pair (C1, C2) using the formula plaintext = (C2 * inverse_mod(shared_secret, q))% q, where inverse_mod is the modular inverse operation.
To recover the original message, Bob reconstructs the plaintext characters.
ElGamal cryptography allows Alice and Bob to communicate securely in this case. Since only Bob, who is in possession of the private key, is able to decrypt the message, the encryption procedure maintains confidentiality. An additional layer of protection is provided by the usage of the one-time key (k). The ElGamal technique also offers integrity because any alterations made to the message while it is being transmitted would produce a different decrypted message.

Code:

```
import random
print("\t\tcode for elagamal cryptography")

print("Enter the prime number(q)")
q=int(input())
print("Enter the primitive root for q choosen")
alpha=int(input())
print("Selecting a random integer XA as the private key")
XA= random.randint(1,q-1)
def cal_YA(alpha,XA,q):
    YA=(alpha**XA)%q
    return YA
def key_display():
    print("The private key is:\t(",XA,")")
    YA=cal_YA(alpha,XA,q)
    print("The public key is:\t(",q,",",alpha,",",YA,")")
    return 0
def encrypt(con,q,YA,alpha,k):
    one_time=(YA**k)%q
    C2=one_time*con %q
    return C2
def decrypt_key(C1,C2,XA,q):
    key= (C1**XA) %q
    for i in range(q):
        if((i*key) % q == 1):
            correct = i
            break
        else:
            continue
    message= (C2*correct) %q
    return message
choice = -1
while(choice != 4):
    print("\n\npress 1 for generating keys")
    print("press 2 for encryption")
    print("press 3 if u want the decryption of enrypted")
    print("enter 4 to exit\n\n")
    choice = int(input())
    if(choice == 1):
        key_display()
    if(choice == 2):
        print("enter the value of plaintext")
        plain = input()
        l1 = list(plain)
        l2 = []
```

```
        print("Enter a random integer k to geneterate one time key (1<k<q-1) ")
        k=int(input())
        C1= (alpha**k)%q
        for i in l1:
            con = ord(i)
            YA=cal_YA(alpha,XA,q)
            enc2= encrypt(con,q,YA,alpha,k)
            l2.append(enc2)
        print("the respective c2 cordinates are \n\n", l2)
        print("\n")
    if(choice == 3):
        l3 = []
        C1= (alpha**k)%q
        for i in l2:
            result=decrypt_key(C1,i,XA,q)
            final = chr(result)
            l3.append(final)
        print("the list after decryption contains::\n", l3)
        print("\nThe decrypted text is", "".join(l3))
    elif(choice == 4):
        print("program ended")
```

CODE EXPLANAATION:

Step-by-step instructions for implementing the ElGamal algorithm:

User Input:

- The user is prompted to enter a prime number (q).
- The user is asked to provide a primitive root (alpha) for the chosen prime number.
- A random private key (XA) is generated.

Public Key Generation (key_display function):

- The public key (YA) is computed using the formula: YA = (alpha^XA) % q.
- The private key (XA) and public key (YA) are displayed.

Encryption (encrypt function):

- The user is prompted to enter plaintext.
- For each character in the plaintext, the ASCII value is computed (con).
- The one-time key (C1) is calculated as C1 = (alpha^k) % q, where k is a randomly chosen integer.
- The public key (YA) is recalculated.
- The ciphertext (C2) for each character is computed as C2 = (one_time * con) % q.
- The encrypted ciphertext values (l2) are stored.

Decryption (decrypt_key function):

- The one-time key (C1) is recalculated.
- For each ciphertext value in l2, the corresponding plaintext is obtained by:
    - Computing the key as key = (C1^XA) % q.

•Finding the correct value (i) where (i * key) % q = 1.
•Calculating the decrypted message as message = (C2 * correct) % q.
•Converting the decrypted message back to its character representation (final).
•The decrypted characters are stored in l3.

Menu-driven Interface:

The user is provided with a menu to choose different options.
Option 1 generates and displays the keys (public and private).
Option 2 encrypts the entered plaintext and displays the ciphertext.
Option 3 decrypts the ciphertext and displays the decrypted text.
Option 4 exits the program.

SNIPETS AND RESULTS DEMONSTRATION:

CODE:

crypto > elgamal.py > ...

```python
import random
print("\t\tcode for elagamal cryptography")

print("Enter the prime number(q)")
q=int(input())

print("Enter the primitive root for q choosen")
alpha=int(input())

print("Selecting a random integer XA as the private key")
XA= random.randint(1,q-1)

def cal_YA(alpha,XA,q):
    YA=(alpha**XA)%q
    return YA

def key_display():
    print("The private key is:\t(",XA,")")
    YA=cal_YA(alpha,XA,q)
    print("The public key is:\t(",q,",",alpha,",",YA,")")
    return 0

def encrypt(con,q,YA,alpha,k):
    one_time=(YA**k)%q
    C2=one_time*con %q
    return C2
def decrypt_key(C1,C2,XA,q):
    key= (C1**XA) %q
    for i in range(q):
        if((i*key) % q == 1):
            correct = i
            break
        else:
            continue
    message= (C2*correct) %q
    return message
```

```
choice = -1
while(choice != 4):
    print("\n\npress 1 for generating keys")
    print("press 2 for encryption")
    print("press 3 if u want the decryption of enrypted")
    print("enter 4 to exit\n\n")
    choice = int(input())
    if(choice == 1):
        key_display()
    if(choice == 2):
        print("enter the value of plaintext")
        plain = input()
        l1 = list(plain)
        l2 = []
        print("Enter a random integer k to geneterate one time key (1<k<q-1) ")
        k=int(input())
        C1= (alpha**k)%q
        for i in l1:
            con = ord(i)
            YA=cal_YA(alpha,XA,q)
            enc2= encrypt(con,q,YA,alpha,k)
            l2.append(enc2)
        print("the respective c2 cordinates are \n\n", l2)
        print("\n")
    if(choice == 3):
        l3 = []
        C1= (alpha**k)%q
        for i in l2:
            result=decrypt_key(C1,i,XA,q)
            final = chr(result)
            l3.append(final)
        print("the list after decryption contains::\n", l3)
        print("\nThe decrypted text is", "".join(l3))
    elif(choice == 4):
        print("program ended")
```

OUTPUTS AND WORKING AND DISCUSSION:

```
                code for elagamal cryptography
Enter the prime number(q)
131
Enter the primitive root for q choosen
3
Selecting a random integer XA as the private key


press 1 for generating keys
press 2 for encryption
press 3 if u want the decryption of enrypted
enter 4 to exit
```

A prime number was selected and its primitive root was provided
A window for next process popped to show from menu
1 was selected to get keys as:

```
1
The private key is:     ( 107 )
The public key is:      ( 131 , 3 , 75 )


press 1 for generating keys
press 2 for encryption
press 3 if u want the decryption of enrypted
enter 4 to exit
```

2 was selected to get the encryption of text:
Necessary inputs were given to get final encrypted text as:

```
2
enter the value of plaintext
ANUJ
Enter a random integer k to geneterate one time key (1<k<q-1)
7
the respective c2 cordinates are

 [74, 115, 127, 52]
```

3 was selected to show decrypted text and finally program was ended:

```
press 1 for generating keys
press 2 for encryption
press 3 if u want the decryption of enrypted
enter 4 to exit


3
the list after decryption contains::
 ['A', 'N', 'U', 'J']

The decrypted text is ANUJ


press 1 for generating keys
press 2 for encryption
press 3 if u want the decryption of enrypted
enter 4 to exit


4
program ended
```

RESULTS:

A secure channel was implemented as described in problem statement over a insecure network using Elgamal the code snippets demonstrated the complete working and also the intermediate outputs as shown above.

The program was tested and verified over 17 different test cases and verified.

Security Analysis of the Implementation:

Potential Threats or Vulnerabilities:

a. Brute Force Attacks: The randomly chosen one-time key (k) should be sufficiently large to withstand brute force attacks.

b. Key Exchange: The code assumes secure key exchange of public keys between Alice and Bob. If the key exchange is compromised, an attacker could impersonate one of the parties or perform man-in-the-middle attacks.

c. Length Extension Attacks: The code does not include measures to prevent length extension attacks on the ElGamal ciphertext.

d. Randomness Generation: The code uses Python's random.randint() function, which might not provide cryptographically secure random numbers. Secure random number generation should be used for key and one-time key generation.

e. Lack of Authentication: The implementation does not include any authentication mechanism, making it vulnerable to impersonation or unauthorized message decryption.

Countermeasures and Best Practices:

a. Brute Force Attacks: Use a sufficiently large one-time key (k) to ensure resistance against brute force attacks. The key should have enough entropy to make exhaustive search infeasible.

b. Key Exchange: Implement a secure key exchange protocol, such as Diffie-Hellman key exchange, to ensure the confidentiality and integrity of the public keys exchanged between Alice and Bob.

c. Length Extension Attacks: Apply a secure padding scheme, such as OAEP (Optimal Asymmetric Encryption Padding), to prevent length extension attacks on the ciphertext.

d. Randomness Generation: Utilize a cryptographically secure random number generator to generate the private keys, one-time keys, and any other random values used in the cryptographic operations.

e. Authentication: Implement mechanisms for message authentication, such as digital signatures or MAC (Message Authentication Code), to ensure the authenticity and integrity of the transmitted messages.

Limitations and Trade-offs:

a. Lack of Error Handling: The code does not include error handling or input validation, which can lead to unexpected behavior or vulnerabilities. Implement proper error handling to handle potential exceptions and ensure the code's robustness.

b. Performance Considerations: The code performs modular exponentiation and computations for each character in the plaintext, which can be computationally expensive for long messages. Consider optimizing the code for better performance, such as utilizing exponentiation algorithms with reduced complexity.

c. Lack of Real-world Security Considerations: The provided implementation is a simplified example for educational purposes. In real-world scenarios, additional security considerations, such as protecting against side-channel attacks, secure storage of keys, and secure transmission channels, should be taken into account.

FINAL CONCLUSION :

Conclusion:

For instructional purposes, the accompanying code provides a simple implementation of the ElGamal cryptographic algorithm. The code can be used as a starting point for understanding ElGamal's core ideas, but it has a number of security flaws and fails to take crucial real-world factors into account.

We discovered possible risks and weaknesses in the implementation through the security study, including brute force attacks, key exchange flaws, length extension attacks, randomness generating problems, and a lack of authentication. Countermeasures and best practises were suggested to increase the security of the implementation, including the use of strong and secure keys, the implementation of secure key exchange protocols, the use of secure padding schemes, the use of cryptographically secure random number generators, and the incorporation of authentication mechanisms.

The results underline how crucial it is for cryptographic systems to adhere to and apply security best practises correctly. When it comes to cybersecurity and ethical hacking, cryptography is essential. It offers methods for guaranteeing the privacy, accuracy, and reliability of data and communications. It serves as the basis for secure authentication, digital signatures, secure storage, and other processes. To protect sensitive information, find vulnerabilities, and secure systems against malicious assaults, professionals working in cybersecurity and ethical hacking must have a solid understanding of cryptographic principles.

It is crucial to remember that cryptography by itself cannot ensure general security. In addition to other security measures like secure key management, secure coding standards, secure network protocols, and frequent security audits, it must be utilised in concert with them. Furthermore, it is advised to rely on well-known and well-reviewed cryptographic libraries and protocols rather than implementing proprietary solutions because they have been subject to extensive evaluation and testing by the security community.

Overall, cryptography is a crucial part of cybersecurity and ethical hacking, and it must be properly understood and applied if solid, secure systems are to be built.

SUBMITTED BY:

ANUJ GUPTA
20BKT0073