CM2025 Computer Security: Midterm Coursework June 2024

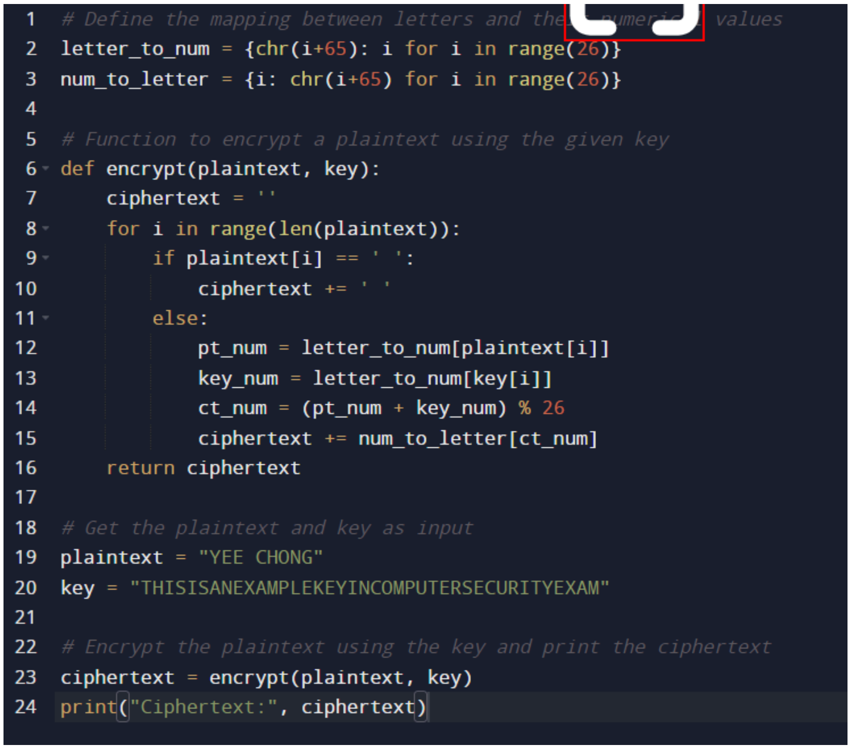
student #: 230668566

PART B: Cryptography

1

(a) Write a program in Python, C/C++ or JavaScript to take both the plaintext and the key as its input, then print out the cipher. Assume the plaintext is your name, for the key=”THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM” what is the output?

In this case, I have used my name, less the surname, as the plaintext to be encrypted using this algorithm that is written in Python.

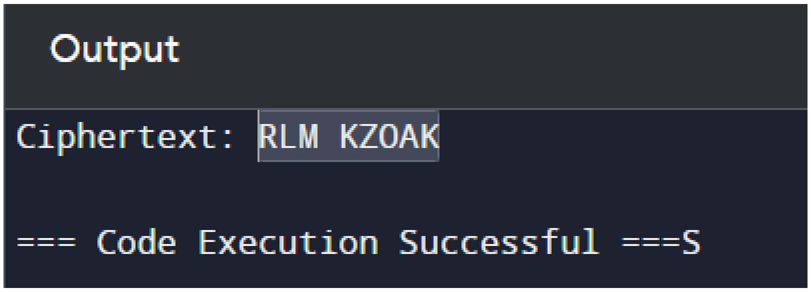


```python
# Define the mapping between letters and their numerical values
letter_to_num = {chr(i+65): i for i in range(26)}
num_to_letter = {i: chr(i+65) for i in range(26)}

# Function to encrypt a plaintext using the given key
def encrypt(plaintext, key):
    ciphertext = ''
    for i in range(len(plaintext)):
        if plaintext[i] == ' ':
            ciphertext += ' '
        else:
            pt_num = letter_to_num[plaintext[i]]
            key_num = letter_to_num[key[i]]
            ct_num = (pt_num + key_num) % 26
            ciphertext += num_to_letter[ct_num]
    return ciphertext

# Get the plaintext and key as input
plaintext = "YEE CHONG"
key = "THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM"

# Encrypt the plaintext using the key and print the ciphertext
ciphertext = encrypt(plaintext, key)
print("Ciphertext:", ciphertext)
```



```
Ciphertext: RLM KZOAK

=== Code Execution Successful ===S
```

The output of this algorithm is RLM KZOAK as shown in the screenshot above.

(b) Explain how one can decrypt the encrypted message using the encryption algorithm in part (a)? Write a program, in JavaScript, C/C++ or Python to take both the ciphertext and the key as its input, then print out the plaintext. Test it using the results of the previous program.

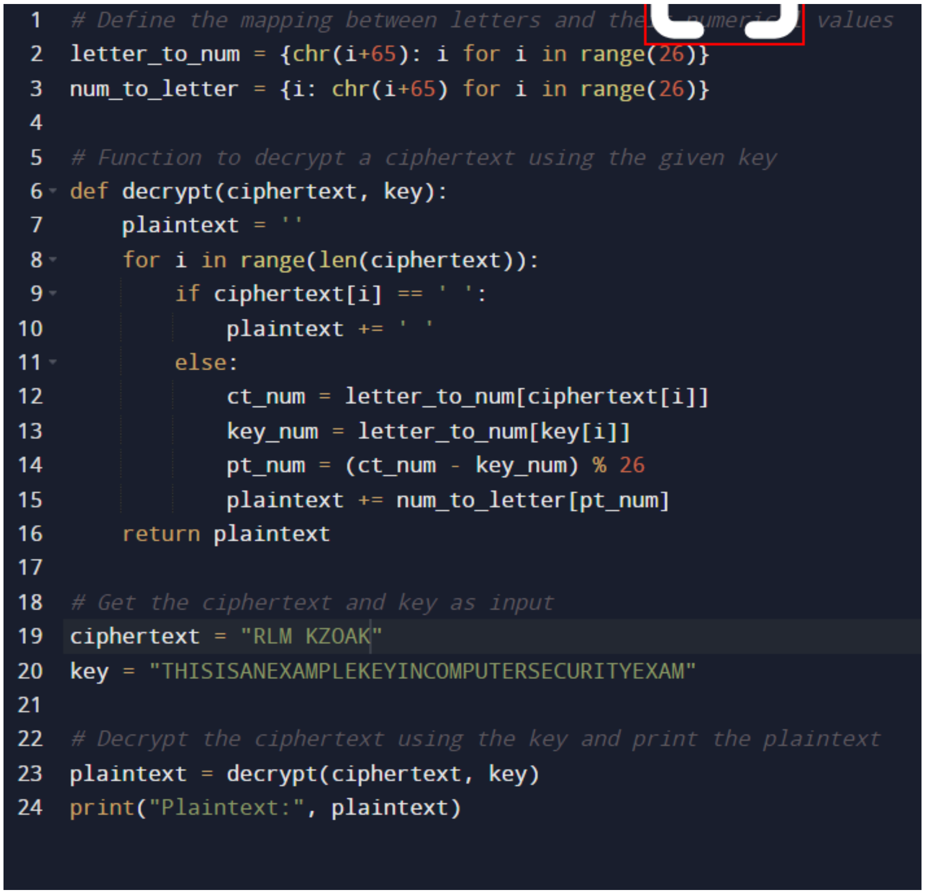
The encryption algorithm used in part (a) is a symmetric key encryption algorithm, where the same key is used for both encryption and decryption.

Each letter in the plaintext and the key are assigned a numerical value, where 'A' is 0, 'B' is 1, ..., 'Z' is 25.

The numerical values of the corresponding plaintext and key letters are added together, modulo 26, to get the ciphertext.

The decryption process is the reverse of the encryption process. Hence, the decryption can be performed by subtracting the corresponding key value from the ciphertext value, modulo 26.

Below shows the screenshot of the program that I wrote to decrypt messages that are encrypted using the algorithm in part (a) using the same ciphertext that i derived from part (a), “RLM KZOAK” and key.

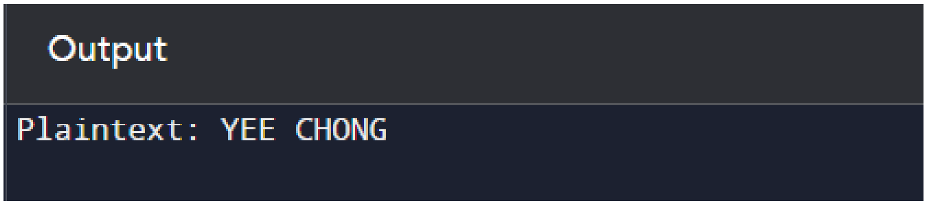


```python
# Define the mapping between letters and their numeric values
letter_to_num = {chr(i+65): i for i in range(26)}
num_to_letter = {i: chr(i+65) for i in range(26)}

# Function to decrypt a ciphertext using the given key
def decrypt(ciphertext, key):
    plaintext = ''
    for i in range(len(ciphertext)):
        if ciphertext[i] == ' ':
            plaintext += ' '
        else:
            ct_num = letter_to_num[ciphertext[i]]
            key_num = letter_to_num[key[i]]
            pt_num = (ct_num - key_num) % 26
            plaintext += num_to_letter[pt_num]
    return plaintext

# Get the ciphertext and key as input
ciphertext = "RLM KZOAK"
key = "THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM"

# Decrypt the ciphertext using the key and print the plaintext
plaintext = decrypt(ciphertext, key)
print("Plaintext:", plaintext)
```



```
Plaintext: YEE CHONG
```

(c) Under what conditions (specify two) can a cipher be broken, meaning that an attacker can decipher the encrypted message without having the key? Support your reasoning with example(s).

The two reasons are:

- **Weak or Predictable Key**

  As mentioned in Bruce Schneier's book "Applied Cryptography", 1996,

  “The strength of a cryptographic system is determined by the **strength of the key, not the strength of the algorithm**." (emphasis is mine)

  Therefore, if the key used for encryption is weak, predictable, or easily guessable, it can be broken by an attacker. This is known as a brute-force attack, where the attacker tries all possible key combinations until the correct one is found.

For example, The WEP (Wired Equivalent Privacy) protocol used in early Wi-Fi networks had a known weakness in its key generation process. The keys were too short (40-bit or 104-bit) and the algorithm used to generate them was predictable. This allowed attackers to easily crack the encryption and gain access to the network.

- **Insufficient Encryption Strength**

  If the encryption algorithm itself is weak or has vulnerabilities, it can be broken by an attacker. This is known as a cryptanalysis attack, where the attacker analyzes the ciphertext and the algorithm to find a way to decrypt the message without the key.

  For example, the DES (Data Encryption Standard) algorithm, once widely used, was eventually broken due to its relatively short 56-bit key length. With the increasing computational power of computers, brute-force attacks became feasible, and DES was eventually replaced by more secure algorithms like AES.

(d) Identify THREE key factors for defining a robust key to enhance algorithm security? Explain your reasons.

The three reasons are:

- **Key Length**

  Longer keys provide a larger key space, making it exponentially more difficult for attackers to brute-force the key and so often when the waiting time outweighs the benefits, it would become not visible for the attacker to continue the said attack.

  As seen in NIST Special Publication 800-57 Part 1 (Revision 5) - "Recommendation for Key Management: Part 1, it recommends using encryption keys with a length of at least 112 bits to provide adequate security, with 128-bit or longer keys being preferred for most applications.

- **Key Randomness**

  Unpredictable and random keys are much harder for attackers to guess or derive.

  This is important as it is also mentioned in many renowned articles such as the ETF RFC 4086 "Randomness Requirements for Security", it says:

  “**Randomness is crucial** for the security of any cryptographic system." (emphasis is mine)

- **Key Management**

  Keys generated for Cryptographic usage should be kept safe and secure and readily accessible as seen in NIST SP 800-133 "Recommendation for Cryptographic Key Generation”, "**Cryptographic keys shall be protected** from unauthorized disclosure and modification."

2

(a) What is the purpose of padding in cryptography algorithms such as DES? Explain it using a simple example.

The primary purpose of padding in cryptography algorithms, such as DES (Data Encryption Standard), is to ensure that the input data is of the appropriate length for the encryption process.

DES operates on 64-bit blocks of data. If the input data is not a multiple of 64 bits, the algorithm would not be able to process the data directly.

For example, let's say we have a 20-byte message.Since the input data is not a multiple of 8 bytes, padding is required. A common padding scheme is to add bytes with a specific value (e.g., 0x00) to the end of the input data until it reaches the next multiple of 8 bytes to be successfully encrypted by the algorithm DES.

(b) We know that the size of the key in DES is 64 bits, assume for padding we just add 0 at the end of the plaintext (the original message), what is the result of padding for the following text? At first replace the “YYY” with the first 3 letters of your name or family then calculate the padding (show the result in hexadecimal format): “YYYComputerSecurity” Show your answer step by step.

Step 1 Replace "YYY" with the first 3 letters "YEE":

"YEEComputerSecurity"

Step 2 Calculate the length of the plaintext:

"YEEComputerSecurity" is 18 bytes long.

Since the DES block size is 8 bytes, we need to pad the plaintext to the next multiple of 8 bytes.

The plaintext is 18 bytes, so we need to add 2 more bytes to reach the next multiple of 8 bytes.

Step 3 Padding the plaintext with 0x00 bytes:

"YEEComputerSecurity00"

Step 4 Convert the padded plaintext to hexadecimal format:

"59 45 45 43 6f 6d 70 75 74 65 72 53 65 63 75 72 69 74 79 00"

Therefore, the result of padding the plaintext "YEEComputerSecurity" is:

59 45 45 43 6f 6d 70 75 74 65 72 53 65 63 75 72 69 74 79 00

(c) We know that RSA works with numbers. Assume we want to encrypt the plaintext M="ComputerSecurity". We know we should use the following formula: C=Me mod N in which (e,N) is the public key. But before using this formula, at first, we should convert M into a number. To do so, there are different solutions. Search the Internet and explain one of them. Show different steps of the selected solution on the given plaintext M="ComputerSecurity" as an example. Consider padding as well.

The method that I would be discussing is by using **Numeric Mapping**.

In this method, each character in the plaintext is assigned a unique numeric value based on a predefined mapping.

One common approach is to use a simple numeric mapping where the letters A-Z are assigned the values 1-26, and the letters a-z are assigned the values 27-52.

For example, the mapping would be:

A = 1, B = 2, ..., Z = 26

a = 27, b = 28, ..., z = 52

Additionally, numbers, punctuation, and special characters, by assigning them numeric values outside the range of the letters (e.g., 53-62 for numbers, 63-72 for punctuation, and so on).

Using this numeric mapping, the plaintext "ComputerSecurity" would be converted as follows:

C = 3, o = 44, m = 42, p = 45, u = 46, t = 47, e = 40, r = 46, S = 19, e = 40, c = 3, u = 46, r = 46, i = 42, t = 47, y = 48

The final numerical value would be: **3444254647404619404647424748**.

(d) What is a digital signature? Is it possible to generate a digital signature using RSA? What about DES? Explain your reasoning.

A digital signature is a mathematical technique used to verify the authenticity and integrity of digital messages, documents, or software. It is a cryptographic method that allows the recipient of a message to verify the identity of the sender and ensure that the message has not been altered during transmission (Stallings, 2017).

It is possible to generate a digital signature using RSA, but not with DES. RSA is a public-key cryptographic algorithm that can be used for both encryption and digital signatures. The process of generating an RSA digital signature involves the following steps:

The signer generates a public-private key pair using the RSA algorithm. The signer uses their private key to sign a message or document, producing a digital signature. The verifier can then use the signer's public key to verify the authenticity and integrity of the signed message or document. The security of RSA digital signatures relies on the difficulty of factoring large prime numbers, which is the basis of the RSA algorithm.

On the other hand, DES (Data Encryption Standard) is a symmetric-key cryptographic algorithm, which means it uses the same key for both encryption and decryption. DES is not suitable for generating digital signatures because it is a secret-key algorithm, and the verifier would need to have access to the same secret key as the signer to verify the signature.

References Cited in Part B

Cryptography and Network Security: Principles and Practice, 8th edition (Accessed 14 June, 2024)

https://www.pearson.com/en-us/subject-catalog/p/cryptography-and-network-security-principles-and-practice/P200000003477/9780135764213

RFC 4086 "Randomness Requirements for Security" (Accessed 14 June, 2024)

https://datatracker.ietf.org/doc/html/rfc4086

NIST Special Publication 800-57 Part 1 (Revision 5) - "Recommendation for Key Management: Part 1 (Accessed 14 June, 2024)

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

Cryptography and Network Security: Principles and Practice (2017) by William Stallings

https://books.google.com/books?id=Ipw8swEACAAJ