Computer Security Midterm

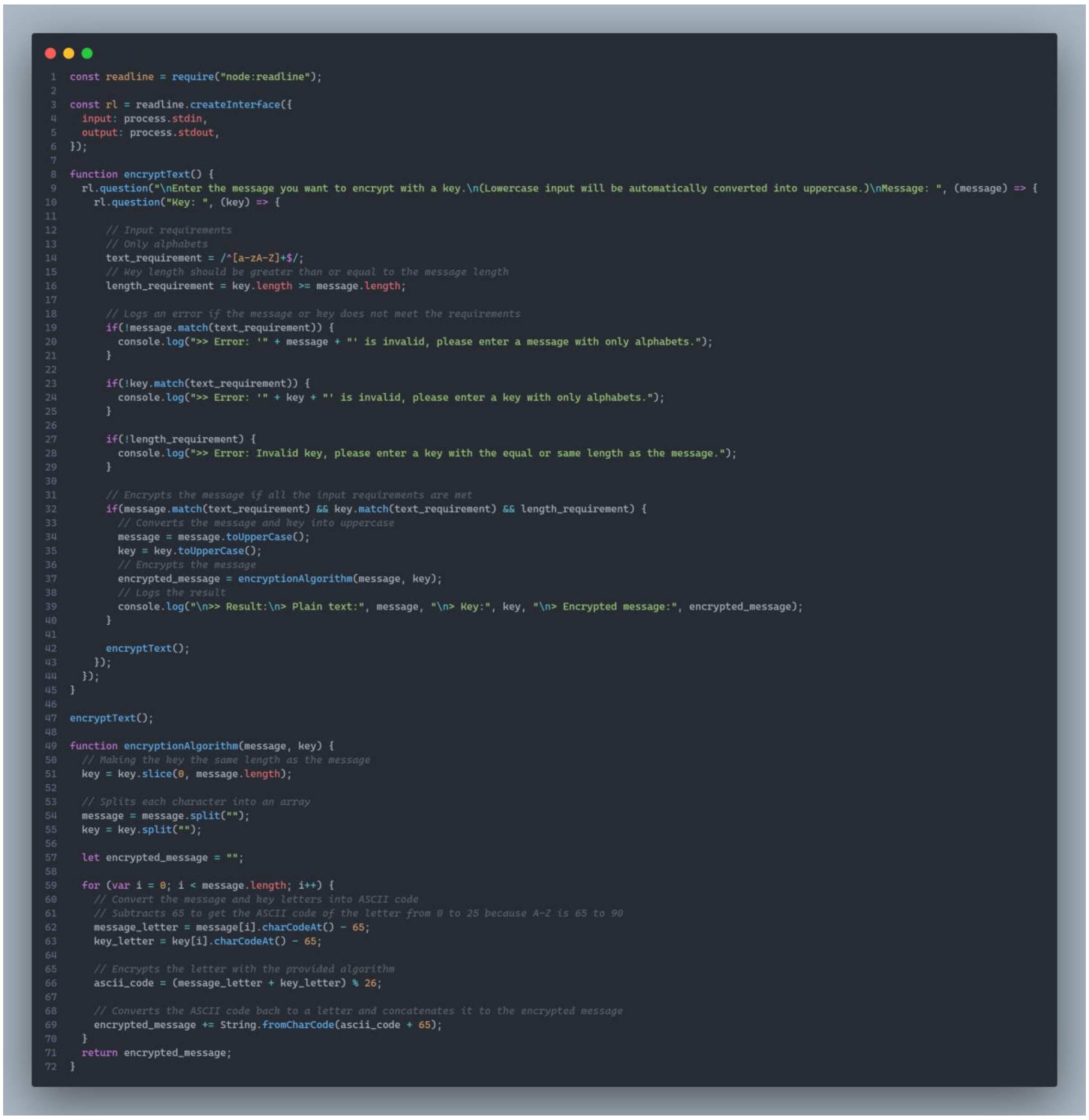
Part B: Cryptography

Contents

Part B Question 1

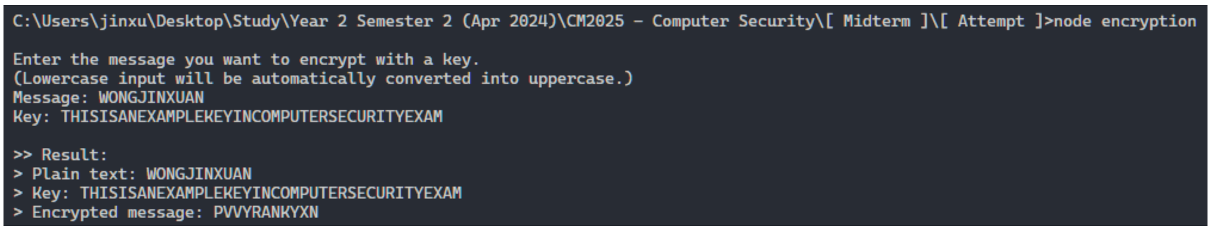
Question 1a)

Screenshot of my encryption program.



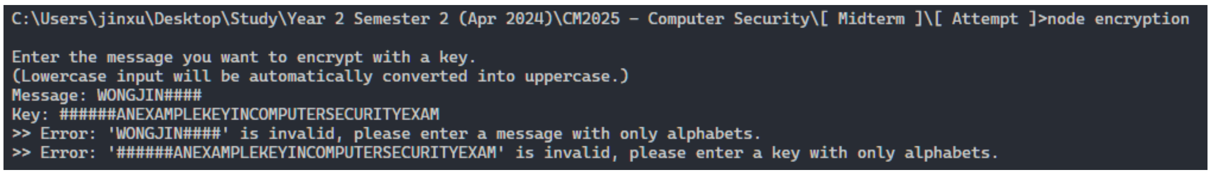
```javascript
const readline = require("node:readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

function encryptText() {
  rl.question("\nEnter the message you want to encrypt with a key.\n(Lowercase input will be automatically converted into uppercase.)\nMessage: ", (message) => {
    rl.question("Key: ", (key) => {

      // Input requirements
      // Only alphabets
      text_requirement = /^[a-zA-Z]+$/;
      // Key length should be greater than or equal to the message length
      length_requirement = key.length >= message.length;

      // Logs an error if the message or key does not meet the requirements
      if(!message.match(text_requirement)) {
        console.log(">> Error: '" + message + "' is invalid, please enter a message with only alphabets.");
      }

      if(!key.match(text_requirement)) {
        console.log(">> Error: '" + key + "' is invalid, please enter a key with only alphabets.");
      }

      if(!length_requirement) {
        console.log(">> Error: Invalid key, please enter a key with the equal or same length as the message.");
      }

      // Encrypts the message if all the input requirements are met
      if(message.match(text_requirement) && key.match(text_requirement) && length_requirement) {
        // Converts the message and key into uppercase
        message = message.toUpperCase();
        key = key.toUpperCase();
        // Encrypts the message
        encrypted_message = encryptionAlgorithm(message, key);
        // Logs the result
        console.log("\n>> Result:\n> Plain text:", message, "\n> Key:", key, "\n> Encrypted message:", encrypted_message);
      }

      encryptText();
    });
  });
}

encryptText();

function encryptionAlgorithm(message, key) {
  // Making the key the same length as the message
  key = key.slice(0, message.length);

  // Splits each character into an array
  message = message.split("");
  key = key.split("");

  let encrypted_message = "";

  for (var i = 0; i < message.length; i++) {
    // Convert the message and key letters into ASCII code
    // Subtracts 65 to get the ASCII code of the letter from 0 to 25 because A-Z is 65 to 90
    message_letter = message[i].charCodeAt() - 65;
    key_letter = key[i].charCodeAt() - 65;

    // Encrypts the letter with the provided algorithm
    ascii_code = (message_letter + key_letter) % 26;

    // Converts the ASCII code back to a letter and concatenates it to the encrypted message
    encrypted_message += String.fromCharCode(ascii_code + 65);
  }
  return encrypted_message;
}
```

Output when the plaintext is my name, and the key is THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM:



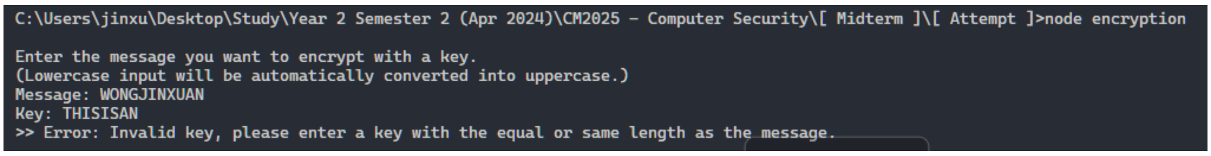
```
C:\Users\jinxu\Desktop\Study\Year 2 Semester 2 (Apr 2024)\CM2025 - Computer Security\[ Midterm ]\[ Attempt ]>node encryption

Enter the message you want to encrypt with a key.
(Lowercase input will be automatically converted into uppercase.)
Message: WONGJINXUAN
Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM

>> Result:
> Plain text: WONGJINXUAN
> Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM
> Encrypted message: PVVYRANKYXN
```

Output when the input includes characters other than alphabets



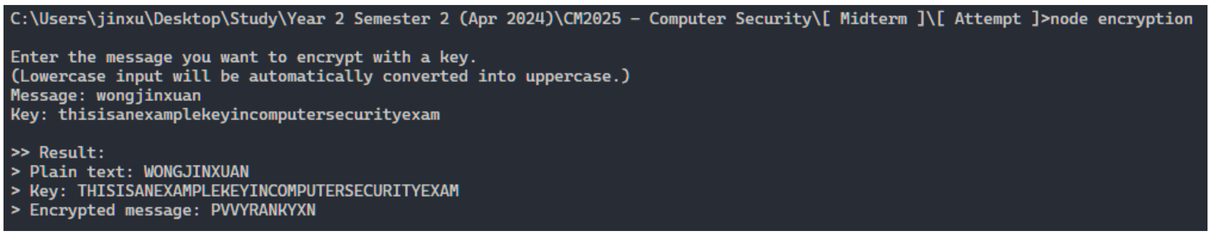
```
C:\Users\jinxu\Desktop\Study\Year 2 Semester 2 (Apr 2024)\CM2025 - Computer Security\[ Midterm ]\[ Attempt ]>node encryption

Enter the message you want to encrypt with a key.
(Lowercase input will be automatically converted into uppercase.)
Message: WONGJIN####
Key: ######ANEXAMPLEKEYINCOMPUTERSECURITYEXAM
>> Error: 'WONGJIN####' is invalid, please enter a message with only alphabets.
>> Error: '######ANEXAMPLEKEYINCOMPUTERSECURITYEXAM' is invalid, please enter a key with only alphabets.
```

Output when the key is shorter than the plaintext



```
C:\Users\jinxu\Desktop\Study\Year 2 Semester 2 (Apr 2024)\CM2025 - Computer Security\[ Midterm ]\[ Attempt ]>node encryption

Enter the message you want to encrypt with a key.
(Lowercase input will be automatically converted into uppercase.)
Message: WONGJINXUAN
Key: THISISAN
>> Error: Invalid key, please enter a key with the equal or same length as the message.
```

Output when the plaintext or key is entered in lowercase



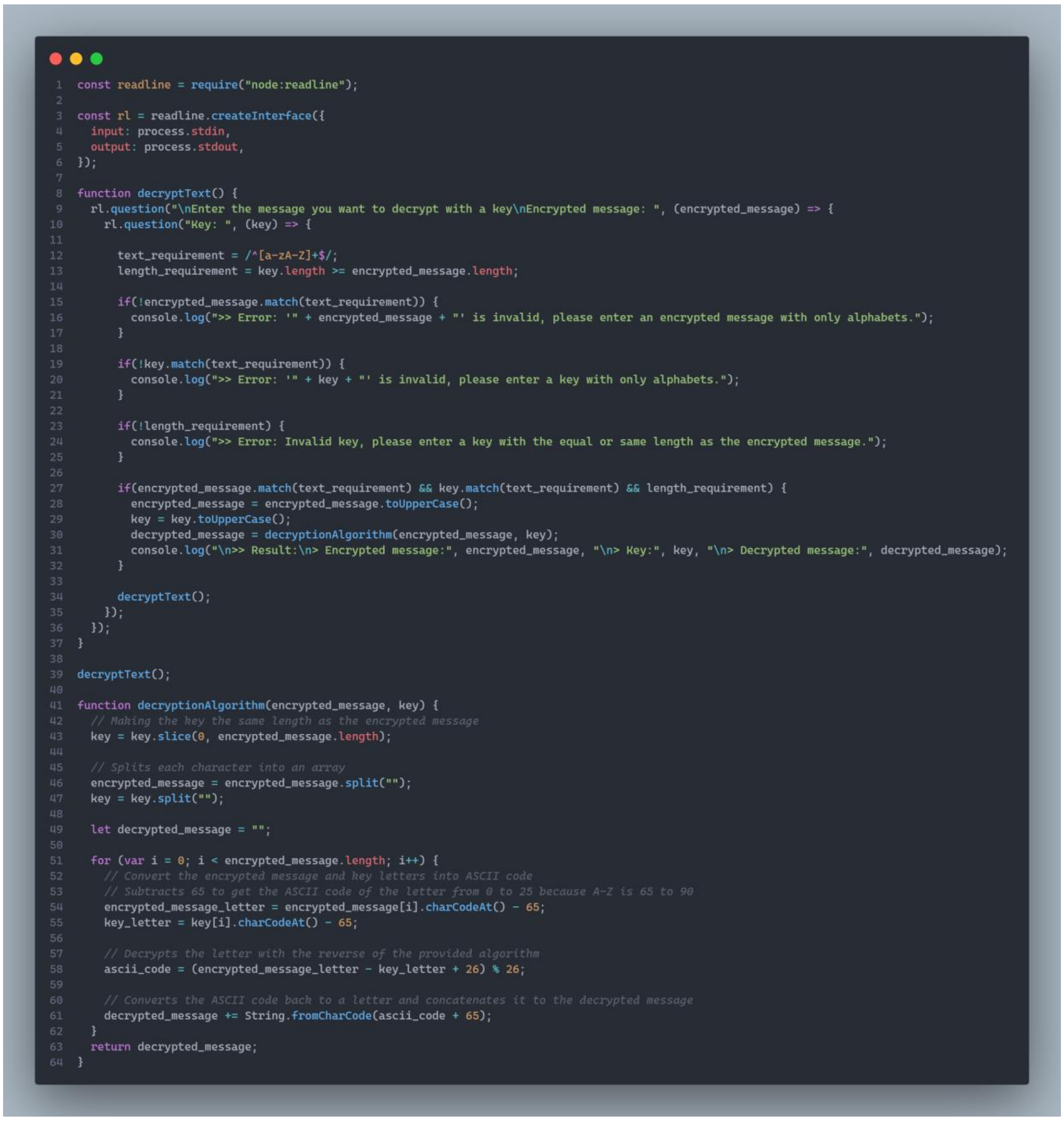
```
C:\Users\jinxu\Desktop\Study\Year 2 Semester 2 (Apr 2024)\CM2025 - Computer Security\[ Midterm ]\[ Attempt ]>node encryption

Enter the message you want to encrypt with a key.
(Lowercase input will be automatically converted into uppercase.)
Message: wongjinxuan
Key: thisisanexamplekeyincomputersecurityexam

>> Result:
> Plain text: WONGJINXUAN
> Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM
> Encrypted message: PVVYRANKYXN
```

Question 1b)

First, the same input validation used in the encryption process will be applied to ensure that the input matches the requirements, where only capital letters are accepted. Both the ciphertext and the key will then be converted to uppercase because the encryption algorithm only works on uppercase letters. The excess part of the key will then be sliced to match the length of the ciphertext, ensuring both the ciphertext and the key have the same length. After that, the input strings of both the ciphertext and key will be split into arrays. Using a for loop, each element in the array (each letter of the input) will be computed by subtracting 65 from the ASCII value of each character. Since ASCII values of A to Z are between 65 to 90, so this will map the letters into the range of 0 to 25.

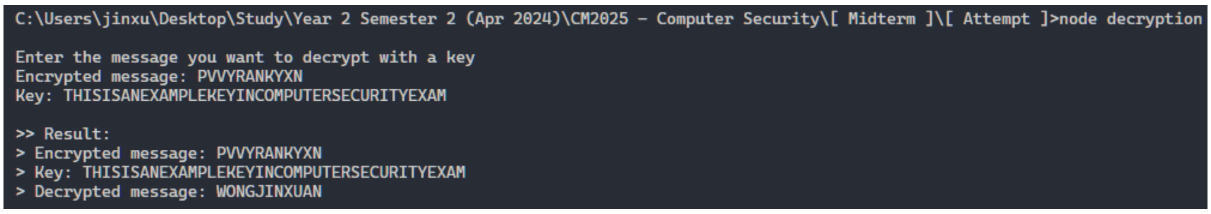
To decrypt the ciphertext, the reverse of the encryption algorithm is implemented. Instead of adding the ASCII value after subtracting 65, then modulo 26 the result, decryption subtracts the ASCII values of the key character from the ASCII values of the ciphertext characters. This may ends up with a negative number if the ciphertext character’s ASCII value is less than the key character's. For instance, if the ciphertext character is A, which maps to 0, and the key character is B, which maps to 1, the subtraction would return -1. To fix this issue, 26 is added to ensure that the result is always positive before performing modulo 26. This ensures that the decryption process will only have characters mapped between 0 to 25. At the end, 65 is added back to the mapped values to revert them back to their original ASCII values, which are then converted back into characters based on their ASCII values.

Screenshot of my decryption program:



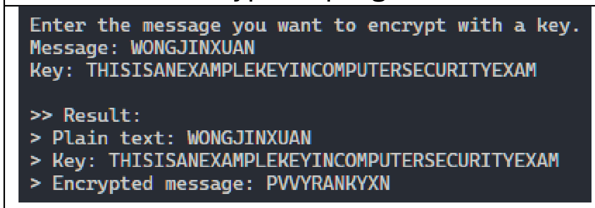
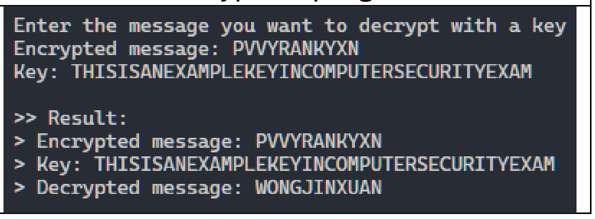
```javascript
const readline = require("node:readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

function decryptText() {
  rl.question("\nEnter the message you want to decrypt with a key\nEncrypted message: ", (encrypted_message) => {
    rl.question("Key: ", (key) => {

      text_requirement = /^[a-zA-Z]+$/;
      length_requirement = key.length >= encrypted_message.length;

      if(!encrypted_message.match(text_requirement)) {
        console.log(">> Error: '" + encrypted_message + "' is invalid, please enter an encrypted message with only alphabets.");
      }

      if(!key.match(text_requirement)) {
        console.log(">> Error: '" + key + "' is invalid, please enter a key with only alphabets.");
      }

      if(!length_requirement) {
        console.log(">> Error: Invalid key, please enter a key with the equal or same length as the encrypted message.");
      }

      if(encrypted_message.match(text_requirement) && key.match(text_requirement) && length_requirement) {
        encrypted_message = encrypted_message.toUpperCase();
        key = key.toUpperCase();
        decrypted_message = decryptionAlgorithm(encrypted_message, key);
        console.log("\n>> Result:\n> Encrypted message:", encrypted_message, "\n> Key:", key, "\n> Decrypted message:", decrypted_message);
      }

      decryptText();
    });
  });
}

decryptText();

function decryptionAlgorithm(encrypted_message, key) {
  // Making the key the same length as the encrypted message
  key = key.slice(0, encrypted_message.length);

  // Splits each character into an array
  encrypted_message = encrypted_message.split("");
  key = key.split("");

  let decrypted_message = "";

  for (var i = 0; i < encrypted_message.length; i++) {
    // Convert the encrypted message and key letters into ASCII code
    // Subtracts 65 to get the ASCII code of the letter from 0 to 25 because A-Z is 65 to 90
    encrypted_message_letter = encrypted_message[i].charCodeAt() - 65;
    key_letter = key[i].charCodeAt() - 65;

    // Decrypts the letter with the reverse of the provided algorithm
    ascii_code = (encrypted_message_letter - key_letter + 26) % 26;

    // Converts the ASCII code back to a letter and concatenates it to the decrypted message
    decrypted_message += String.fromCharCode(ascii_code + 65);
  }
  return decrypted_message;
}
```

Output when the ciphertext is my encrypted name, and the key is THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM:



```
C:\Users\jinxu\Desktop\Study\Year 2 Semester 2 (Apr 2024)\CM2025 - Computer Security\[ Midterm ]\[ Attempt ]>node decryption

Enter the message you want to decrypt with a key
Encrypted message: PVVYRANKYXN
Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM

>> Result:
> Encrypted message: PVVYRANKYXN
> Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM
> Decrypted message: WONGJINXUAN
```

The table below compares the output of the encryption and decryption program.

Encryption program	Decryption program
	

```
Enter the message you want to encrypt with a key.
Message: WONGJINXUAN
Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM

>> Result:
> Plain text: WONGJINXUAN
> Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM
> Encrypted message: PVVYRANKYXN
```

```
Enter the message you want to decrypt with a key
Encrypted message: PVVYRANKYXN
Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM

>> Result:
> Encrypted message: PVVYRANKYXN
> Key: THISISANEXAMPLEKEYINCOMPUTERSECURITYEXAM
> Decrypted message: WONGJINXUAN
```

From the table, the output from the decryption program matches the plaintext input in the encryption program, while the output from the encryption program matches with the ciphertext input in the decryption program.

Question 1c)

Condition 1: Small key size encryption algorithms

If the encryption algorithm used to encrypt the message has a small key size, an attacker can exploit that vulnerability to decipher the ciphertext without having the key. Encryption algorithm with small key sizes is especially weak against techniques like brute force attacks, where an attacker tries all possible keys until they find the correct key.

For example, Data Encryption Standard (DES) only supports key size of 56 bits, which means with enough processing power, the key could be brute forced to crack the code. As parallel processing becoming more well-known, Electronic Frontier Foundation (EFF) [1] created a 56-bit DES cracking machine that supports parallel processing to prove that DES was weak. The machine processed 90,000,000 keys per seconds, and in 1998, it eventually cracked the DES code with less than 3 days by brute forcing.

Condition 2: Nature of substitution cipher

If the encryption algorithm used substitution cipher to encrypt a message, an attacker can exploit the vulnerability of the nature of substitution cipher to decrypt the ciphertext without having the key. In substitution cipher, each plaintext character is replaced with another character based on the key, which means each plaintext character will map to the same ciphertext throughout the encrypted message. The attacker can perform frequency analysis to learn compare the frequency of each letter used in the ciphertext against the plaintext language.

For example, the Kryptos sculpture [2] at the CIA headquarters feature four encrypted messages, from K1 to K4. David Stein, a CIA agent, decrypted the first section of the encrypted messages, K1, in 1998. He collected a huge amount of ciphertext data, then performed frequency analysis of the ciphertext characters and compare them with English letter frequencies. Through a long process of frequency analysis and testing, he successfully decrypted the K1 section of the Kryptos sculpture.

Question 1d)

Factor 1: Complexity of the key

The complexity of a key is important to enhance algorithm security. A robust key should include different types of characters like numbers, case-sensitive letters, symbols, and others to enhance the algorithm security. This can increase the complexity of the key, thus making it harder to be cracked.

For instance, if a key has a key length of 1 and consists solely of lowercase letters, it results in 26 possible combinations. This small range of possible combination can be easily cracked by trying all possible combinations. However, if the key includes all human-readable and printable ASCII characters, which has 95 possible characters, the key with a length 1 would have 95 possible combinations. As the key length increases along with the complexity of the key, the difficulty to crack it increases exponentially, which shows how key complexity can enhance the algorithm security.

Factor 2: Length of the key

The length of a key is crucial for defining a robust key to enhance algorithm security. A longer key increases the total number of possible combinations a key can have. It creates a high level of security and be insusceptible to brute force attacks due to the exponentially increased combination.

For example, Data Encryption Standard (DES) developed in the 1970s, supports the key length of only 56 bits, which means the key has 2^{56} possible combinations. In contrast, Advanced Encryption Standard (AES) supports key length up to 256 bits, which means the key has 2^{256} possible combination. The increase in the key length makes AES significantly harder to crack, which shows how longer key length can enhance the algorithm security.

Factor 3: Management of the keys

Having good key management is important to enhance algorithm security. A good key management involves implementing security measures to protect the generation, distribution, and retrieval of keys. This ensures that the keys to not be stolen during these processes.

For example, generating the key with a suitable key length and complexity ensures that it is hard to crack. When distributing keys, both the key itself and the entire distribution process should be encrypted to provide a double layer of security. This means that even if the process is intercepted, the attack would need to decrypt two layers of encryption to access the key. Storing the key in a secured place, like an encrypted file can prevent unauthorized access. Additionally, changing keys regularly also ensures that each key has a limited lifespan, so even if a key is compromised, it would not be usable forever.

Part B Question 2

Question 2a)

Data Encryption Standard (DES) is an encryption algorithm that encrypts data in 64-bit blocks. If the size of the plaintext does not fit the 64-bit block size, padding is added to ensure it fits the block size requirement.

For example, the plaintext “COMPSEC” consists of 7 characters, since each character is usually 1 byte, so in this case, “COMPSEC” translate to 7 bytes (56 bits). DES requires a 64-bit block to work, so 1 extra byte (8 bits) of padding is needed to make the plaintext reach the required length.

A simple padding method [3] is to add the required number of bytes to the end of the plaintext. Since “COMPSEC” only need one more byte to reach 8 bytes (64 bits), decimal “1” can be added to the end of the plaintext, resulting in “COMPSEC1”, which turn it into a 64-bit block, ensuring it to work with the DES encryption algorithm.

Question 2b)

<u>Step 1: Replace characters of the plaintext</u>

Replace "YYY" with the first three letters of my name, “WON”, the plaintext results in “WONComputerSecurity”.

<u>Step 2: Calculate the padding</u>

“WONComputerSecurity” consists of 19 characters. Each character is usually 1 byte, so the length of the plaintext is 19 bytes. Since DES only accepts input to be multiples of 8 bytes (64 bits), so the plaintext is still required to be padded to be the multiple of 8 bytes. In this case, the next multiple of 8 after 19 bytes is 24 bytes.

24 bytes – 19 bytes = 5 bytes

So, 5 extra bytes of paddings needs to be added to the plaintext to fit the DES block size. A simple padding method [3] is to add the required number of bytes to the end of the plaintext. In this case, 5 bytes of hexadecimal value “05” will be added.

Step 3: Convert the plaintext into hexadecimal

The first row indicates the plaintext, while the second row indicates the hexadecimal value of the respective character. [4]

W	O	N	C	o	m	p	u	t	e	r	S	e	c	u	r	i	t	y
57	4F	4E	43	6F	6D	70	75	74	65	72	53	65	63	75	72	69	74	79

Step 4: Adding the padding

According to step 2, 5 bytes of hexadecimal value of “05” will be added to the hexadecimal value of the plain text. The plaintext “WONComputerSecurity” will be padded into “WONComputerSecurity55555”, which results in the following hexadecimal form.

57 4F 4E 43 6F 6D 70 75 74 65 72 53 65 63 75 72 69 74 79 05 05 05 05 05

Question 2c)

Before performing RSA formula, the messages have to be converted into number. The most common way to convert plaintext into a number is by translating each byte into their respective ASCII value, then converting the value into hexadecimal notation.

Message = Computer Security

Step 1: Getting ASCII value of each byte

First row = bytes, Second row = ASCII value

C	o	m	p	u	t	e	r	S	e	c	u	r	i	t	y
67	111	109	112	117	116	101	114	83	101	99	117	114	105	116	121

Step 2: Converting the ASCII values into Hexadecimal value

In RSA, plaintext often needs to be padded to ensure it meets the block size requirement, converting them into hexadecimal makes it easier to represent and modify the byte value

First row = bytes, Second row = ASCII value, Third row = Hexadecimal value

C	o	m	p	u	t	e	r	S	e	c	u	r	i	t	y
67	111	109	112	117	116	101	114	83	101	99	117	114	105	116	121
43	6F	6D	70	75	74	65	72	53	65	63	75	72	69	74	79

Step 3: Combining the hexadecimal value

43 6F 6D 70 75 74 65 72 53 65 63 75 72 69 74 79

Step 4: Adding padding if necessary

RSA encryption requires messages to fit within fixed-size blocks. When a plaintext message is shorter than the block size, padding is added to ensure the message length matches the block size for encryption.

Assuming the RSA requires blocks of 24 bytes (192-bits). So, the plaintext is still required to be padded to be the multiple of 24 bytes. In this case, the next multiple of 24 after 16 bytes is 24 bytes.

24 bytes – 16 bytes = 8 bytes

This means that 8 bytes of padding is required to meet the block size requirement. A simple padding method, byte padding can be used. Byte padding adds the required number of bytes to the end of the plaintext. In this case, 8 bytes is required, so 8 bytes of hexadecimal value of "08" will be added to the end of the plaintext. Resulting in the following value:

43 6F 6D 70 75 74 65 72 53 65 63 75 72 69 74 79 08 08 08 08 08 08 08 08

Step 5: Combining all the values

Combined hexadecimal value:

436F6D707574657253656375726974790808080808080808

Step 6: Converting the combined hexadecimal value into large decimal

Large decimal:

67111109112117101114831011171141051831011711411 05121

After converting the plaintext "ComputerSecurity" into a large decimal number, it is ready to be used in the RSA encryption formula.

Question 2d)

A digital signature is a cryptographic technique that verifies the authenticity of digital messages or documents. It ensures that the message is sent by the expected user to the other end without being tampered by an attacker.

It is possible to generate a digital signature using RSA as it is a type of asymmetric key algorithm that uses two linked keys, a public key and a private key. The public key is shared with everyone, while the private key is kept secret to the owner. When a message is sent, a hash function is used to hash the message into a non-human readable format. The hash is then encrypted using the sender's public or private key. When the receiver receives the message, the opposite key of the sender is then used to decrypt the encryption to retrieve the hashed message. If both hash matches, the messages are then verified to be authentic and not tampered.

On the other hand, it is impossible to generate a digital signature using DES as it is a type of symmetric key algorithm that uses the same key to both encrypt and decrypt a message. This means that anyone with the key can both encrypt and decrypt the message, which fails to verify the authenticity of a message because there are no separate keys that are linked to sender. For example, by using RSA, the sender can sign a message by encrypting it with their private key. The receiver can then decrypt the message using the sender's public key, ensuring that the message came from the sender. In contrast, DES uses the same key for both encryption and decryption. When the sender encrypts a message using DES, anyone with shared key can decrypt and tamper with the message. Furthermore, the receiver cannot verify the message's authenticity and whether they are tampered because DES relies on a single key for both operations.

References

1. Buchanan, William. Cryptography, River Publishers, 2017. (Page 62) ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/londonww/detail.action?docID=30251706 . Created from londonww on 2024-06-20 09:42:52.
2. LEMMiNO. (2024, May 4). The unbreakable Kryptos Code [Video]. YouTube. https://www.youtube.com/watch?v=jVpsLMCIB0Y
3. Padding schemes for block ciphers. (n.d.). CryptoSys PKI Pro Manual. https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html
4. IBM. (2021). ASCII and Hex Equivalents. IBM Documentation. https://www.ibm.com/docs/en/iis/11.7?topic=reference-ascii-hex-equivalents
5. Michael Cobb (2021). RSA algorithm (Rivest-Shamir-Adleman). TechTarget Security. https://www.techtarget.com/searchsecurity/definition/RSA
6. tanujajoshi24 (2023). RSA and Digital Signatures. GeeksForGeeks. https://www.geeksforgeeks.org/rsa-and-digital-signatures/
7. shubhamupadhyay (2023). Data encryption standard (DES) | Set 1. GeeksForGeeks. https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/
8. marstato (2016). How do ciphers change plaintext into numeric digits for computing? StackExchange – Cryptography. https://crypto.stackexchange.com/a/37855
9. Wikipedia contributors (2024). Padding (cryptography). Wikipedia. https://en.wikipedia.org/wiki/Padding_(cryptography)