Problem 1

† Data compression is often used in data storage and transmission. Suppose you want to use data compression in conjunction with encryption. Does it make more sense to:

■ Compress then encrypt.

■ Encrypt then compress.

■ The order does not matter – either one is fine.

□ The order does not matter – neither one will compress the data.

Explanation:

◎"Compress then encrypt." :

Compressing data reduces its size, making it more efficient to transmit or store. Encrypting compressed data ensures that sensitive information remains secure during transmission or storage. This approach is commonly employed in network communications, file transfers, and data backups, among other scenarios.

◎"Encrypt then compress." :

If the confidentiality is the top priority, then "encrypt then compress" will be our choice. Encrypting plaintext first provides an additional layer of security by obscuring any patterns or redundancies in it. It makes more difficult for attackers to gain insights into the data, even if they can access the compressed form.

◎"The order does not matter– either one is fine."

The two orders are both acceptable, depending on the specific purpose for which the data will be used.

◎ "The order does not matter– neither one will compress the data.":

Both of two methods will compress the data. Therefore, This option is false.

Problem 2

† Let G: {0,1}s $\rightarrow$ {0,1}n be a secure PRG. Which of the following is a secure PRG:

□ G'(k) = G(k) || G(k)

■ G'(k) = G(k$\oplus$1s)

□ G'(k) = G(0)

□ G'(k) = G(1)

□ G'(k) = G(k) || 0

■ G'(k1, k2) = G(k1) || G(k2)

■ G'(k) = reverse(G(k))

■ G'(k) = rotation$_n$(G(k))

Explanation:

◎G'(k) = G(k) || G(k):

It is not a secure PRG since it does not follow the property of unpredictability. We

can know that the former part of G'(k) is always same to the latter part. If someone knows what G(k) is, then he/she can easily predict that the latter part is still G(K). Therefore, this option is not correct.

◎G'(k) = G(k⊕1s):

k⊕1s is the input in G. Since G is a secure PRG, its output should be indistinguishable from random, regardless of the input. Therefore, G' is also a secure PRG.

◎G'(k) = G(0) and G'(k) = G(1):

Those constructions ignore the input k and always outputs G(0) or G(1). Since G(0) and G(1) are fixed strings, it is trivially distinguishable from a random string, and thus both G' are not secure PRG.

◎G'(k) = G(k) || 0:

This construction concatenates the output of G(k) with a fixed string of zeros. It is not a secure PRG since it does not follow the property of unpredictability. If someone knows what G(k) is, then he/she can easily predict that the last bit must be 0. Therefore, this option is not correct.

◎G'(k1, k2) = G(k1) || G(k2):

Since G is a secure PRG, the output G(k1) and G(k2) are indistinguishable from a random string of length n. If we concatenate G(k1) and G(k2), the result will definitely not break this property. Therefore, G' is also a secure PRG.

◎G'(k) = reverse(G(k)):

'Reverse' is a kind of fixed order permutation. Since G(k) is indistinguishable from random, applying a fixed permutation does not break this property, and therefore, G' is a secure PRG.

◎G'(k) = rotation$_n$(G(k)):

Similar to the previous case, rotation n positions is also a kind of fixed order permutation. Since G(k) is indistinguishable from random, applying a fixed rotation does not break this property, and thus G' is a secure PRG.

Problem 3

Let (E, D) be a (one-time) semantically secure cipher with key space K = $\{0,1\}^k$. A bank wishes to split a decryption key k ∈ $\{0,1\}^k$ into two pieces $p_1$ and $p_2$ so that both are needed for decryption. The piece p1 can be given to one executive and $p_2$ to another so that both must contribute their pieces for decryption to proceed.

The bank generates random $k_1$ in $\{0,1\}^k$ and sets $k_1' \leftarrow k \oplus k_1$. Note that $k_1 \oplus k_1'$ = k. The bank can give $k_1$ to one executive and $k_1'$ to another. Both must be present for decryption to proceed since, by itself, each piece contains no information about the secret key k (note that each piece is a one-time pad encryption of k).

Now, suppose the bank wants to split k into three pieces $p_1$, $p_2$, $p_3$ so that any two of

the pieces enable decryption using k. This ensures that even if one executive is out sick, decryption can still succeed. To do so the bank generates two random pairs ($k_1$, $k_1'$) and($k_2$, $k_2'$) as in the previous paragraph so that $k_1 \oplus k_1' = k_2 \oplus k_2' = k$. How should the bank assign pieces so that any two pieces enable decryption using k, but no single piece can decrypt?

☐ $p_1 =(k_1, k_2)$, $p_2 = (k_1, k_2)$, $p_3 = (k_2')$

☐ $p_1 =( k_1, k_2)$, $p_2 = (k_1', k_2')$, $p_3 = (k_2')$

■ $p_1 =(k_1, k_2)$, $p_2 = (k_1', k_2)$, $p_3 = (k_2')$

☐ $p_1 =(k_1, k_2)$, $p_2 = (k_2, k_2')$, $p_3 = (k_2')$

☐ $p_1 =(k_1, k_2)$, $p_2 = (k_1')$, $p_3 = (k_2')$

Explanation:

◎$p_1 =(k_1, k_2)$, $p_2 = (k_1, k_2)$, $p_3 = (k_2')$:

If we take $p_1$ and $p_2$, we only have $k_1$ and $k_2$. Due to the lack of $k_1'$ and $k_2'$, we cannot reconstruct k. Therefore, this option is not correct.

◎$p_1 =( k_1, k_2)$, $p_2 = (k_1', k_2')$, $p_3 = (k_2')$:

If we take $p_2$ and $p_3$, we only have $k_1'$ and $k_2'$. Due to the lack of $k_1$ and $k_2$, we cannot reconstruct k. Therefore, this option is not correct.

◎$p_1 =(k_1, k_2)$, $p_2 = (k_1', k_2)$, $p_3 = (k_2')$:

If we take $p_1$ and $p_2$, we can use $k_1$ and $k_1'$ to reconstruct k.

If we take $p_1$ and $p_3$, we can use $k_2$ and $k_2'$ to reconstruct k.

If we take $p_2$ and $p_3$, we can use $k_2$ and $k_2'$ to reconstruct k.

Therefore, this option is correct.

◎$p_1 =(k_1, k_2)$, $p_2 = (k_2, k_2')$, $p_3 = (k_2')$:

$p_2$ contains both $k_2$ and $k_2$. That implies that if someone possesses $p_2$, he/she can reconstruct k on his/her own without needing any other pieces. Therefore, this option is not correct.

◎$p_1 =(k_1, k_2)$, $p_2 = (k_1')$, $p_3 = (k_2')$:

If we take $p_2$ and $p_3$, we only have k1' and k2'. Due to the lack of k1 and k2, we cannot reconstruct k. Therefore, this option is not correct.

Problem 4

Let M = C =K={0,1, 2, ..., 255 } and consider the following cipher defined over (K, M, C):

E(k, m) = m + k (mod 256); D(k, c) = c- k (mod 256)

Does this cipher have perfect secrecy?

☐ No, there is a simple attack on this cipher.

■ Yes

☐ No, only the One Time Pad has perfect secrecy.

Explanation:
If we have two plaintexts, m0 and m1, and randomly choose one of them to encrypt using function E, then observing the resulting ciphertext does not allow us to distinguish which plaintext was chosen. Therefore, the cipher exhibits perfect security. "Yes" is the correct option, while the others are incorrect.

Problem 5
† Let (E, D) be a (one-time) semantically secure cipher where the message and ciphertext space is $\{0,1\}^n$. Which of the following encryption schemes are (one-time) semantically secure?

- ☐ E' (k, m) = $E(0^n, m)$
- ■ E' ((k, k'), m) = E(k, m) || E(k', m)
- ☐ E' (k, m) = E(k, m) || MSB(m)
- ■ E' (k, m) = 0 || E(k, m) (i.e. prepend 0 to the ciphertext)
- ☐ E' (k, m) = E(k, m) || k
- ■ E' (k, m) = reverse(E(k, m))
- ■ E' (k, m) = $\text{rotation}_n$(E(k, m))

Explanation:
◎E'(k, m) = $E(0^n, m)$:
This scheme is not semantically secure because it always uses the same key $0^n$ for encryption. An adversary can simply encrypt all possible messages with the key $0^n$ and compare the ciphertexts to break the scheme.
◎E' ((k, k'), m) = E(k, m) || E(k' , m):
The ciphertext consists of two independent encryptions of the same message using different keys, and both encryptions are semantically secure. After concatenating them, it still not reveals any information about the plaintext. Therefore, this option is correct.
◎E' (k, m) = E(k, m) || MSB(m):
It is not semantically secure because it reveals the most significant bit of the plaintext message m. An adversary can deduce information about the plaintext from the MSB.
◎E' (k, m) = 0 || E(k, m) (i.e. prepend 0 to the ciphertext):
Prepending a fixed bit (0) to the ciphertext does not leak any additional information about the plaintext, and we have already known that E is a semantically secure encryption. Therefore, E' (k, m) = 0 || E(k, m) must be semantically secure. This option is correct.
◎E' (k, m) = E(k, m) || k:
E' (k, m) is not semantically secure because it reveals the key k used for encryption. An adversary can learn the key from the ciphertext (by comparing many different

ciphertext) and decrypt the message. Therefore, this option is not true.

◎E' (k, m) = reverse(E(k, m)):

'Reverse' is a kind of fixed order permutation. Since E(k, m) is semantically secure, performing fixed permutation does not leak any information about the plaintext. Therefore, this option is correct.

◎E' (k, m) = $\text{rotation}_n$(E(k, m)):

Similar to the previous case, rotation n positions is also a kind of fixed order permutation. Since E(k, m) is semantically secure, performing fixed permutation does not leak any information about the plaintext. Therefore, this option is correct.

Problem 6

Suppose you are told that the one time pad encryption of the message "attack at dawn" is 6c73d5240a948c86981bc294814d (the plaintext letters are encoded as 8-bit ASCII and the given ciphertext is written in hex). What would be the one time pad encryption of the message "defend at noon" under the same OTP key?

Explanation:

(1) Convert the plaintext "attack at dawn" to its binary representation. ➔

01100001 01110100 01110100 01100001 01100011 01101011 00100000 01100001 01110100 00100000 01100100 01100001 01110111 01101110

(2) Convert the given ciphertext from hexadecimal to binary. ➔

01101100 01110011 11010101 00100100 00001010 10010100 10001100 10000110 10011000 00011011 11000010 10010100 10000001 01001101

(3) Find the OTP key by XORing the plaintext and ciphertext. ➔

OTP KEY = 01100001 01110100 01110100 01100001 01100011 01101011 00100000 01100001 01110100 00100000 01100100 01100100 01110111 01101110 ⊕

01101100 01110011 11010101 00100100 00001010 10010100 10001100 10000110 10011000 00011011 11000010 10010100 10000001 01001101

= 00001101 00000111 10100001 01000101 01101001 11111111 10101100 11100111 11101100 00111011 10100110 11110101 11110110 00100011

(4) Encode the new plaintext "defend at noon" in binary using 8-bit ASCII.

"defend at noon" in ASCII binary:➔

01100100 01100101 01100110 01100101 01101110 01100100 00100000 01100001 01110100 00100000 01101110 01101111 01101111 01101110

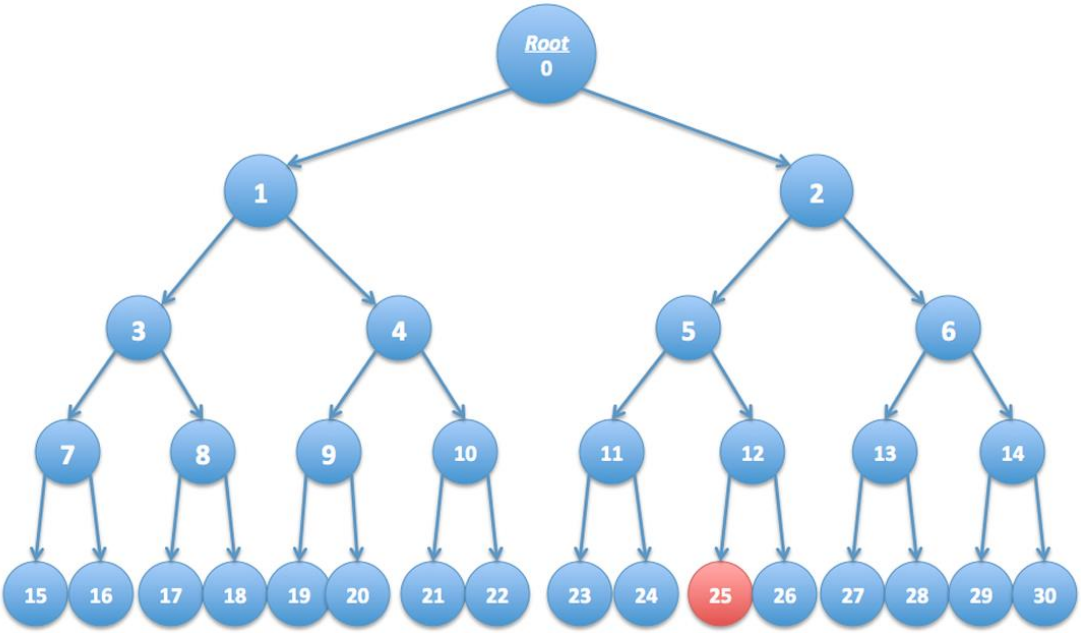
(5) Encrypt "defend at noon" using the OTP key obtained in Step 3.

Ciphertext = Plaintext ⊕ OTP key

= 01100100 01100101 01100110 01100101 01101110 01100100 00100000 01100001 01110100 00100000 01101110 01101111 01101111 01101110 ⊕

00001101 00000111 10100001 01000101 01101001 11111111 10101100 11100111 11101100 00111011 10100110 11110101 11110110 00100011
= 01101001 01100010 11000111 00100000 00000111 10011011 10001100 10000110 10011000 00011011 11001000 10011010 10011001 01001101
(6) Convert the binary ciphertext to hexadecimal.
➔ **6962C720079B8C86981BC89A994D**

Problem 7
† The movie industry wants to protect digital content distributed on DVD's. We develop a variant of a method used to protect Blu-ray disks called AACS.
Suppose there are at most a total of n DVD players in the world (e.g. n = 232). We view these n players as the leaves of a binary tree of height log2n. Each node in this binary tree contains an AES key ki. These keys are kept secret from consumers and are f ixed for all time. At manufacturing time each DVD player is assigned a serial number i ∈ [0, n-1]. Consider the set of nodes Si along the path from the root to leaf number i in the binary tree. The manufacturer of the DVD player embeds in player number i the keys associated with the nodes in the set Si. A DVD movie m is encrypted as
E(kroot, k) ‖ E(k, m)
where k is a random AES key called a content-key and kroot is the key associated with the root of the tree. Since all DVD players have the key kroot all players can decrypt the movie m. We refer to E(kroot, k) as the header and E(k, m) as the body. In what follows the DVD header may contain multiple ciphertexts where each ciphertext is the encryption of the content-key k under some key ki in the binary tree.
Suppose the keys embedded in DVD player number r are exposed by hackers and published on the Internet. In this problem we show that when the movie industry distributes a new DVD movie, they can encrypt the contents of the DVD using a slightly larger header (containing about log2n keys) so that all DVD players, except for player number r, can decrypt the movie. In effect, the movie industry disables player number r without affecting other players.
As shown below, consider a tree with n = 16 leaves. Suppose the leaf node labeled 25 corresponds to an exposed DVD player key. Check the set of keys below under which to encrypt the key k so that every player other than player 25 can decrypt the DVD. Only four keys are needed.




- [ ] 21
- [ ] 17
- [ ] 5
- [x] 26
- [x] 6
- [x] 1
- [x] 11
- [ ] 24

Explanation:

The player 15~22 can use key 1 to decrypt the DVD.

The player 27~30 can use key 6 to decrypt the DVD.

The player 23~24 can use key 11 to decrypt the DVD.

The player 26 can use key 26 to decrypt the DVD.

Player 25 is the only one for which no decryption key exists to unlock the DVD.

Therefore, the set of keys is {26, 6, 1, 11}.

Extra Credit

Did SHA-256andSHA-512-truncated-to-256-bits have the same security properties? Which one is better? Please explain in detail.

Explanation:

No, SHA-256 and SHA-512-truncated-to-256-bits do not have the same security properties. SHA-256 and the full SHA-512 have undergone thorough cryptanalysis and are deemed resistant to known attacks, the truncated version of SHA-512 has not been subjected to extensive analysis. This lack of scrutiny leaves it potentially

vulnerable to attacks. Moreover, Truncating SHA-512 to 256 bits is not a standardized or widely adopted practice, and it may raise compatibility and interoperability concerns. Therefore, SHA-256 is the better one.