Quiz 3

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.

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.

1. **Compress then encrypt**: This makes more sense and is generally the recommended approach. Data compression works by eliminating redundancies and patterns in data, making it smaller. Encryption, on the other hand, transforms data into a format that can only be read by someone who has the correct decryption key. If you compress data after encryption, the encryption process would have already removed data patterns, making the compression far less effective, if at all. Therefore, compressing before encrypting is the usual practice as it maintains the efficiency of the compression and then secures the compressed data through encryption.
2. **Encrypt then compress**: There is a latest way to encrypt before compressing. So this may be make sense.
3. **The order does not matter – either one is fine**: Due to different situations, the order does not matter.
4. **The order does not matter – neither one will compress the data**: This is incorrect. Data compression can indeed reduce the size of data, but its effectiveness is greatly reduced if performed after encryption.

So, the correct choices are compress then encrypt, encrypt the compress and the order does not matter — either one is fine.

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 $\oplus$ 1^s)

☐ G'(k) = G(0)

☐ G'(k) = G(1)

☐ G'(k) = G(k) || 0

☐ G'(k_1, k_2) = G(k_1) || G(k_2)

☐ G'(k) = reverse(G(k))

☐ G'(k) = $rotation_n$(G(k))

Hint:
"|| " denotes concatenation.
"reverse(x)" reverses the string x so that the first bit of x is the last bit of reverse(x), the second bit of x is the second to last bit of reverse(x), and so on.
"$rotation_n(x)$" rotates the string x by n positions. If n>0, it rotates right; if n<0, it rotates left, and characters shifted off one end reappear at the other.

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 $\oplus$ 1^s)

☐ G'(k) = G(0)

☐ G'(k) = G(1)

☐ G'(k) = G(k) || 0

☐ G'(k_1, k_2) = G(k_1) || G(k_2)

☐ G'(k) = reverse(G(k))

☐ G'(k) = $rotation_n$(G(k))

Hint:
"|| " denotes concatenation.
"reverse(x)" reverses the string x so that the first bit of x is the last bit of reverse(x), the second bit of x is the second to last bit of reverse(x), and so on.
"$rotation_n(x)$" rotates the string x by n positions. If n>0, it rotates right; if n<0, it rotates left, and characters shifted off one end reappear at the other.

1. $G\prime(k) = G(k)\ ||\ G(k)$**:** This option simply concatenates the output of G(*k*) with itself. This does not maintain security because the pattern of the

output becomes predictable; the second half of the output is always the same as the first half, which can be exploited.

2. $G\prime(k) = G(k \oplus 1^s)$: This option represents applying a bitwise XOR between the key *k* and a string of 1s of length *s*. This could still be considered secure if *G* is secure, because the XOR operation with a fixed string doesn't structurally alter the unpredictability of *G* applied to different inputs.
3. $G\prime(k) = G(0)$: This is not secure because it always produces the same output for any input key *k*, removing all unpredictability.
4. $G\prime(k) = G(1)$: Similar to the previous, this always produces the same output for any input *k*, which is not secure.
5. $G\prime(k) = G(k) \,||\, 0$: Adding a fixed value (in this case a series of 0s) at the end of the output does not inherently compromise the security, but it doesn't add to the security either; however, it's not the typical form of enhancing a PRG's security.
6. $G\prime(k_1, k_2) = G(k_1) \,||\, G(k_2)$: If *G* is a secure PRG, then using two different keys k_1 and k_2 to generate two independent outputs and concatenating them should remain secure, as it preserves the unpredictability and distribution properties required for a secure PRG.
7. $G\prime(k) = reverse(G(k))$: Reversing the output of a secure PRG should not affect its security, as the unpredictability and uniform distribution characteristics of the output are preserved, just in reverse order.
8. $G\prime(k) = rotation_n(G(k))$: Rotating the output of a secure PRG by *n* positions, either left or right, should also not affect its security, as all elements remain in the output but in different positions, maintaining its unpredictability and distribution properties.

From the options provided, the transformations that could still be considered secure PRGs, assuming *G* is secure, would be:

- $G\prime(k) = G(k \oplus 1^s)$
- $G\prime(k_1, k_2) = G(k_1) \,||\, G(k_2)$
- $G\prime(k) = reverse(G(k))$
- $G\prime(k) = rotation_n(G(k))$

These transformations do not inherently compromise the unpredictability or distribution properties that a secure PRG should have.

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 $\in \{0,1\}^k$ into two pieces p_1 and p_2 so that both are needed for decryption. The piece p_1 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?

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

1 p_1, p_2 cannot generate k

2 p_2, p_3 cannot generate k

4 p_2 cannot generate k

5 p_2, p_3 cannot generate k

The answer should be no. 3 $p1 = (k1, k2), p2 = (k1^{'}, k2), p3 = (k2^{'})$

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 has perfect secrecy?

- ☐ No, there is a simple attack on this cipher.
- ☐ Yes
- ☐ No, only the One Time Pad has perfect secrecy.

if $E(k,m) = c, m + k(mod256) = c$

then $k = c - m(mod256)$

so the Pr [E (k, m) = c] = 1/256

it is hard to crack c, the answer is yes.

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) = $rotation_n$(E(k, m))

1. $E'(k,m) = E(0_n, m)$: This scheme uses a fixed key (all zeros) regardless of *k*. This is not semantically secure because using a fixed key for different messages compromises the security by making it easier to analyze patterns in the ciphertexts.
2. $E'((k,k'),m) = E(k,m) \,||\, E(k',m)$: This scheme involves using two keys *k* and *k'* to encrypt the message *m* twice and concatenating the results. This remains semantically secure as long as the original cipher *E* is secure, and both *k* and *k'* are used only once. The use of two different keys maintains the unpredictability of the output.

3. $E'(k, m) = E(k, m) \,||\, MSB(m)$: This appends the most significant bit (MSB) of the message to the ciphertext. This compromises semantic security because part of the plaintext (the MSB) is directly revealed in the ciphertext, providing information about the plaintext without needing the key.
4. $E'(k, m) = 0 \,||\, E(k, m)$: This simply prepends a 0 to the ciphertext generated by the secure encryption function *E*. This does not compromise the security of the underlying cipher, so it remains semantically secure. The prepended bit does not provide additional information about the plaintext.
5. $E'(k, m) = E(k, m) \,||\, k$: This appends the key *k* to the ciphertext. This scheme is not semantically secure because it reveals the encryption key *k* alongside the ciphertext, compromising all security.
6. $E'(k, m) = reverse(E(k, m))$: This involves reversing the ciphertext generated by *E*. Since the security of *E* depends on its input (the plaintext and the key) and not on the order of the ciphertext's bits, this operation does not compromise semantic security. The reversed ciphertext is as secure as the original provided *E* is secure.
7. $E'(k, m) = rotationn(E(k, m))$: This rotates the ciphertext by *n* positions. Like reversing, rotation does not affect the semantic security provided by *E*, assuming the original cipher *E* is secure. The content and encryption of the message remain unchanged in their security implications; only the order of the ciphertext bits changes.

Based on these analyses, the encryption schemes that maintain (one-time) semantic security, assuming the original encryption scheme *E* is secure, are:

- $E'((k, k'), m) = E(k, m) \,||\, E(k', m)$
- $E'(k, m) = 0 \,||\, E(k, m)$
- $E'(k, m) = reverse(E(k, m))$
- $E'(k, m) = rotationn(E(k, m))$

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?

Given:

- Plaintext: "attack at dawn"
- Ciphertext (hexadecimal): "6c73d5240a948c86981bc294814d"

We will proceed as follows:

1. Convert the plaintext "attack at dawn" into hexadecimal.
2. XOR the plaintext hexadecimal with the ciphertext hexadecimal to find the key.
3. Convert the new message "defend at noon" into hexadecimal.
4. XOR the new message hexadecimal with the key to get the new ciphertext.

Let's perform these operations.

The one-time pad (OTP) key derived from the original plaintext "attack at dawn" and the given ciphertext is `0d07a14569fface7ec3ba6f5f623` in hexadecimal.

Using this OTP key to encrypt the new message "defend at noon" results in the ciphertext `6962c720079b8c86981bc89a994d` in hexadecimal.

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. $n = 2^{32}$). We view these n players as the leaves of a binary tree of height $log_2 n$. Each node in this binary tree contains an AES key k^i. These keys are kept secret from consumers and are fixed for all time. At manufacturing time each DVD player is assigned a serial number i $\in$ [0, n-1]. Consider the set of nodes S_i 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 S_i. A DVD movie m is encrypted as

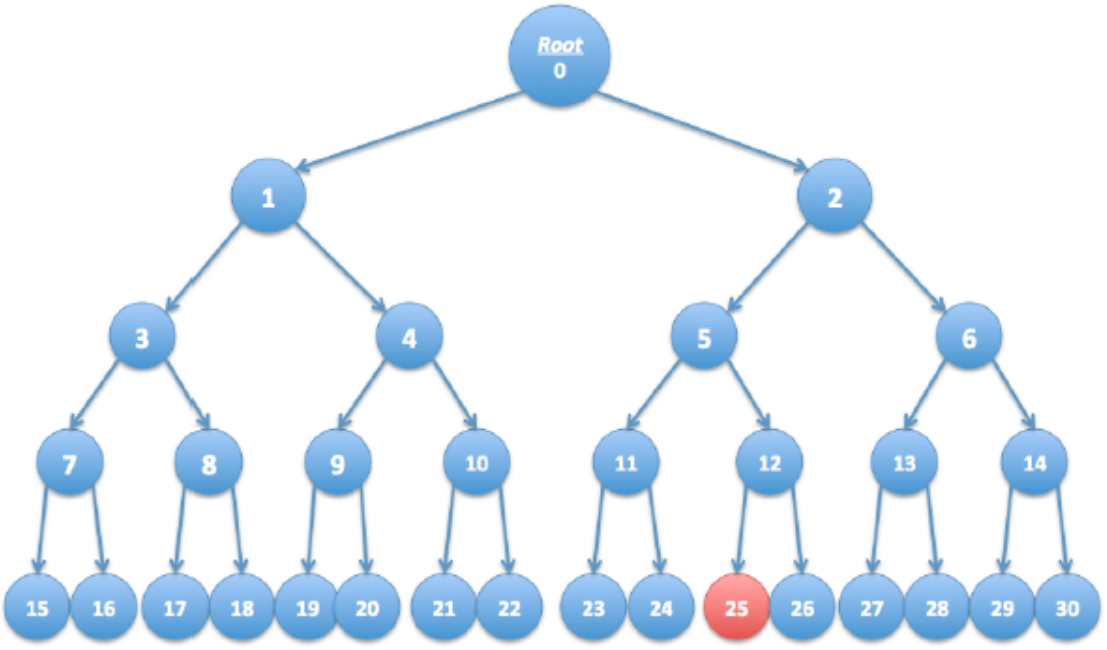
$E(k_{root}, k) \| E(k, m)$

where k is a random AES key called a content-key and k_{root} is the key associated with the root of the tree. Since all DVD players have the key k_{root} all players can decrypt the movie m. We refer to $E(k_{root}, k)$ 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 k_i 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 $log_2 n$ 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
- ☐ 26
- ☐ 6
- ☐ 1
- ☐ 11
- ☐ 24

The set of keys under which to encrypt the key k so that every player other than player 25 can decrypt the DVD would include the keys associated with nodes:

- 1 (covers all players to the left half of the tree)
- 11 (covers the left side of the subtree under node 2)
- 6 (covers the right side of the subtree under node 2, but not including the compromised player)
- 26 (covers the siblings of the compromised player, ensuring all players except for player 25 are included).

Extra Credit

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

	SHA-512-truncated-to-256-bits	SHA-256
performance	O	X
security	O	X
adoption	X	O

- **Performance**: SHA-512 is generally slower than SHA-256 on 32-bit platforms but can be faster on 64-bit platforms due to the way it handles larger words. Therefore, SHA-512-truncated-to-256-bits might perform better than SHA-256 on 64-bit hardware. If performance is a key factor and the application runs on 64-bit hardware, SHA-512-truncated-to-256-bits might be preferred.
- **Security**: Theoretically, SHA-512-truncated-to-256-bits could offer better security due to its larger internal state and block size, providing a higher security margin. However, in practical terms, SHA-256 already offers a high level of security, and no significant weaknesses have been found in either algorithm that would compromise their intended security levels.
- **ComAdoption**: SHA-256 is more widely adopted and has been extensively analyzed and integrated into numerous protocols and systems. If compatibility and widespread support are important, SHA-256 might be the preferable choice.