Crypto Engineering Quiz 3

1.

☐ 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.
Since (B. Encrypt then compress) could reduce the bytes needed to send the message, so it is a reasonable approach. At the same time, (A. Compress then encrypt) could reduce the bytes needed to encrypt, probably could reduce the number of random numbers needed to use and thus lower the requirement of the design of pseudo-random number generator. So, I think both methods are reasonable.

2.

For a secure PRG $G(k): \{0,1\}^s \rightarrow \{0,1\}^n$ ■ $G'(k) = G(k) \parallel G(k)$ ■ $G'(k) = G(k \oplus 1^s)$ ☐ $G'(k) = G(0)$ ☐ $G'(k) = G(1)$ ■ $G'(k) = G(k) \parallel 0$ ■ $G'(k_1, k_2) = G(k_1) \parallel G(k_2)$ ■ $G'(k) = reverse(G(k))$ ■ $G'(k) = rotation_n(G(k))$
Since $G(k)$ is secure, so when the randomness of function $G'(k)$ is built on the randomness of $G(k)$, then it is also secure. Since $G(k)$ is unpredictable, so $G(k) \parallel G(k)$, $reverse(G(k)), rotation_n(G(k))$ are also unpredictable. For $G'(k) = G(k \oplus 1^s)$, consider $k' = k \oplus 1^s$, then since $G(k') = (k \oplus 1^s) = G'(k)$, so $G'(k)$ is also secure. For $G'(k) = G(k) \parallel 0$, since the first n bits of $G'(k)$ is unpredictable, so it is also secure. For $G'(k_1, k_2) = G(k_1) \parallel G(k_2)$, since $G(k_1)$ and $G(k_2)$ are both unpredictable, so $G'(k_1, k_2)$ is surely secure.
As for $G'(k) = G(0)$ and $G'(k) = G(1)$ if the number is used for one time, then the number is predictable afterwards. So, I think those two are not secure.

3.

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

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

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

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

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

For the case $p_1 = (k_1, k_2),\ p_2 = (k_1{}', k_2),\ p_3 = (k_2{}')$

p_1 and p_2 can get k by $k_1 \oplus k_1'$

p_1 and p_3 can get k by $k_2 \oplus k_2'$

p_2 and p_3 can get k by $k_2 \oplus k_2'$

And anyone can't get k by themselves.

For $p_1 = (k_1, k_2),\ p_2 = (k_1, k_2),\ p_3 = (k_2')$

p_1 and p_2 can't get k by themselves.

For $p_1 = (k_1, k_2),\ p_2 = (k_1', k_2'),\ p_3 = (k_2')$

p_2 and p_3 can't get k by themselves.

For $(k_1, k_2),\ p_2 = (k_2, k_2'),\ p_3 = (k_2')$

p_2 could get k by the key pair p_2, so the key k is leaked.

For $p_1 = (k_1, k_2),\ p2 = (k_1'),\ p_3 = (k_2')$

p_2 and p_3 can't get k by themselves.

4.

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

■ Yes

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

$$\forall m_0, m_1 \in M \left(len(m_0) = len(m_1)\right) and\ \forall c \in C$$

$$\Pr[E(k, m_0) = c] = \Pr[m_0 + k = c\ (mod\ 256)] = \Pr[k = c - m_0\ (mod\ 256)] = |K|^{-1}$$

$$\Pr[E(k, m_1) = c] = \Pr[m_1 + k = c\ (mod\ 256)] = \Pr[k = c - m_1\ (mod\ 256)] = |K|^{-1}$$

Since $\Pr[E(k, m_0) = c] = \Pr[E(k, m_1) = c]\ \forall c \in C$

So, this cipher has perfect secrecy.

5.

☐ $E'(k, m) = E(0^n, m)$

■ $E'((k, k'), m) = E(k, m) \parallel E(k', m)$

☐ $E'(k, m) = E(k, m) \parallel MSB(m)$

■ $E'(k, m) = 0 \parallel E(k, m)\ (i.e. prepend\ 0\ to\ the\ ciphertext)$

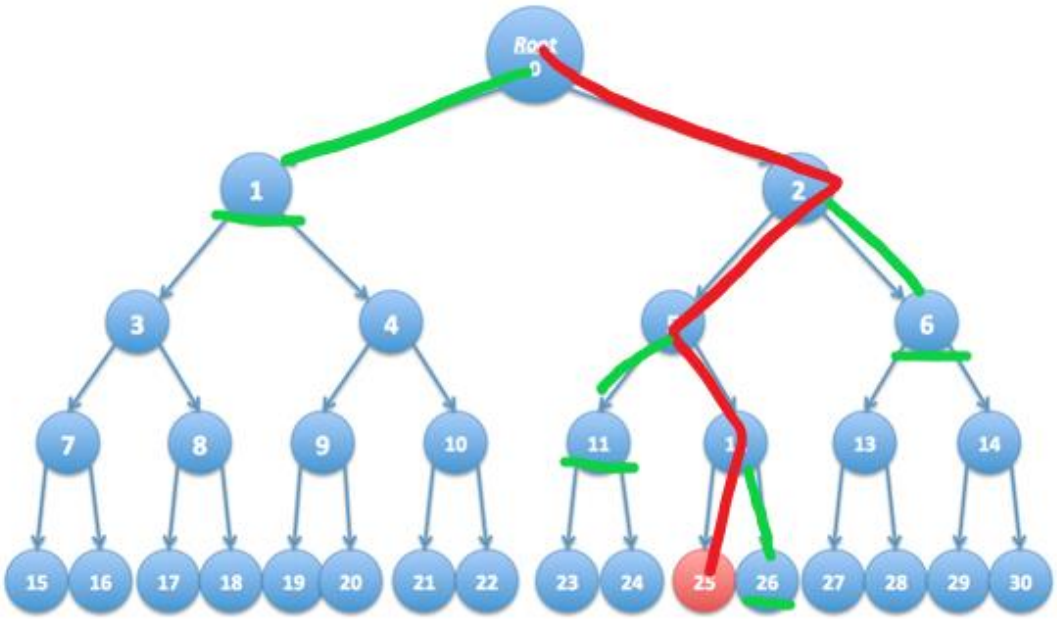
☐ $E'(k, m) = E(k, m) \parallel k$

■ $E'(k,m) = reverse(E(k,m))$ ■ $E'(k,m) = rotation_n(E(k,m))$
Since $E(k,m)$ is semantically secure, so adding any constant into the cipher, concatenating two semantically secure cipher texts, any permutation to the cipher text (reverse or rotation) both not include the information of plaintext, so those are also semantically secure.
For $E'(k,m) = E(0^n,m)$, it always uses same key to encrypt, so it is easy to crack and thus not semantically secure. For $E'(k,m) = E(k,m) \parallel MSB(m)$, it includes the information to the cipher text, so it is not semantically secure. For $E'(k,m) = E(k,m) \parallel k$, since we can extract the key from the cipher text, then decrypt and get the plain text. So, it is not semantically secure (and not secure).

6.

The hex of the message “attack at dawn” is: 61747461636b206174206461776e And the encrypted message is: 6c73d5240a948c86981bc294814d So, we xor the two number above to get the key: d07a14569fface7ec3ba6f5f623 The hex of the message “defend at noon” is: 646566656e64206174206e6f6f6e We xor the hex and the key to get the encrypted message: 6962c720079b8c86981bc89a994d

7.



- ☐ 21
- ☐ 17
- ☐ 5
- ■ 26
- ■ 6
- ■ 1
- ■ 11
- ☐ 24

Since node 25 is cracked and so the keys from node 25 to the root is useless, so we need to cover the rest of the tree with keys other with S_{25}.

We can use the keys at the opposite side every time the path from root to node 25 reaches an intersection to cover the rest of the tree by a minimum number of keys, as the image shown above.

Extra.

I don't think so.

Suppose SHA-256 and SHA-512 has same security properties, which I assume the hash values are distributed evenly and the avalanche effect occurs.

But SHA-512-truncated-to-256-bits could probably distribute not-so-evenly.

Like when most of the 512-bit hash values with same 256-bit prefix P has slightly more occurrence than other prefixes. Then the probability of P in truncated version could be lot bigger than other prefixes.

From another point, I think if the avalanche effect happen mainly in the last 256-

bit part of a hash value for some set of input, then the truncated version could has a lot less avalanche effect for the set of input.