

Problem 1

- Compress then encrypt. (A sensible approach, can compress data first and increase efficiency.)
- Encrypt then compress. (Used in specific scenerios such as steganography.)
- The order does not matter - either one is fine. (As explained, the order does not matter, and either one is used in data encryption.)
- The order does not matter - neither one will compress the data. (Trivially, either one is fine.)

Problem 2

- $G'(k) = G(k) \parallel G(k)$ (The output is twice as long as the output of G , can be easily detected.)
- $G'(k) = G(k \oplus 1^s)$ (The XOR operation modifies the seed k before it is used by G . Since G is secure, G' is also secure.)
- $G'(k) = G(0)$ (The output of G' does not depend on the seed k , making it easy to distinguish.)
- $G'(k) = G(1)$ (Same as the previous case.)
- $G'(k) = G(k) \parallel 0$ (It won't output a random 0.)
- $G'(k_1, k_2) = G(k_1) \parallel G(k_2)$ (Combines two independent outputs G , which remain secure.)
- $G'(k) = \text{reverse}(G(k))$ (Reverse the bits of PRG output does not affect the randomness.)
- $G'(k) = \text{rotation}_n(G(k))$ (Rotate n bits of PRG output does not affect the randomness.)

Problem 3

- $p_1 = (k_1, k_2), p_2 = (k_1, k_2), p_3 = (k'_2) ([p_1, p_2] : \times; [p_1, p_3] : (k_2, k'_2); [p_2, p_3] : (k_2, k'_2))$
- $p_1 = (k_1, k_2), p_2 = (k'_1, k'_2), p_3 = (k'_2) ([p_1, p_2] : (k_1, k'_1) \vee (k_2, k'_2); [p_1, p_3] : (k_2, k'_2); [p_2, p_3] : \times)$
- $p_1 = (k_1, k_2), p_2 = (k'_1, k_2), p_3 = (k'_2) ([p_1, p_2] : (k_1, k'_1); [p_1, p_3] : (k_2, k'_2); [p_2, p_3] : (k_2, k'_2))$
- $p_1 = (k_1, k_2), p_2 = (k_2, k'_2), p_3 = (k'_2)$ (p_2 can decrypt by only a single piece.)
- $p_1 = (k_1, k_2), p_2 = (k'_1), p_3 = (k'_2) ([p_1, p_2] : (k_1, k'_1); [p_1, p_3] : (k_2, k'_2); [p_2, p_3] : \times)$

Problem 4

- No, there is a simple attack on this cipher. (The cipher has perfect secrecy.)
- Yes. (Using different k, m , can get an evenly distributed c .)
- No, only the One Time Pad has perfect secrecy. (The cipher has perfect secrecy.)

Problem 5

- ☐ $E'(k, m) = E(0^n, m)$ (It uses constant key 0^n for every encryption.)
- ☒ $E'((k, k'), m) = E(k, m) \parallel E(k', m)$ (It encrypts the message twice with two different keys and concatenates the results.)
- ☐ $E'(k, m) = E(k, m) \parallel MSB(m)$ (Appending the MSB of the message leaks part of the plaintext.)
- ☒ $E'(k, m) = 0 \parallel E(k, m)$ (Prepending a 0 does not leak any information about the plaintext.)
- ☐ $E'(k, m) = E(k, m) \parallel k$ (It reveals the key with the ciphertext.)
- ☒ $E'(k, m) = reverse(E(k, m))$ (Reversing the order of the bits does not affect its security properties.)
- ☒ $E'(k, m) = rotation_n(E(k, m))$ (Rotating the order of n bits does not affect its security properties.)

Problem 6

key: $(61747461636b206174206461776e)_{16} \oplus (6c73d5240a948c86981bc294814d)_{16}$

$= (0d07a14569fface7ec3ba6f5f623)_{16}$

defend at noon: $(646566656e64206174206e6f6f6e)_{16} \oplus (0d07a14569fface7ec3ba6f5f623)_{16}$

$= (6962c720079b8c86981bc89a994d)_{16}$

```

1 #include <bits/stdc++.h>
2 #define For(z, x, y) for(int z = x; z <= y; z++)
3 #define sz(x) ((int) x.size())
4
5 using namespace std;
6
7 int32_t main(){
8     string a, b;
9     cin >> a >> b;
10
11     For(i, 0, sz(a)-1){
12         int ta = a[i] - '0', tb = b[i] - '0';
13         cout << (ta ^ tb);
14     }
15
16     return 0;
17 }
```

Problem 7

- ☐ 21 (Not needed, covered by node 1.)
- ☐ 17 (Not needed, covered by node 1.)
- ☐ 5 (Not needed.)
- ☒ 26 (Needed to cover 26.)
- ☒ 6 (Needed to cover 27 to 30.)
- ☒ 1 (Needed to cover 15 to 22.)
- ☒ 11 (Needed to cover 23 to 24.)
- ☐ 24 (Not needed, covered by node 11.)

Extra Credit

1. SHA-256:
 - (a) Design: Operates on 32-bit words using 64 rounds of hashing.
 - (b) Output size: Produces a 256-bit hash value.
 - (c) Security: Designed to resist all known forms of cryptographic attacks, although its resistance to quantum computing attacks could be lower due to its smaller bit size compared to SHA-512.
2. SHA-512/256
 - (a) Design: Operates on 64-bit words and runs 80 rounds of hashing, which is more than SHA-256.
 - (b) Output Size: The output is truncated to 256 bits.
 - (c) Security: Theoretically, it provides security equivalent to SHA-256 against brute force attacks but is believed to be more resistant to certain types of cryptographic attacks.
3. Summary: While SHA-256 and SHA-512/256 have excellent security for current standards, SHA-512/256 is slightly better in theoretical security due to the larger internal state and word size.