Exercises and Solutions on ChaCha20 Stream Cipher

Exercise 1.
ChaCha20 is a modern stream cipher that uses an "ARX" design: **A**ddition, **R**otation, and **X**OR. Unlike AES, it does not use S-boxes. We consider a "Toy" version using 4-bit words.

The fundamental building block is the **Quarter Round (QR)**. Given four 4-bit values (a, b, c, d), the QR operations are:

1. $a = (a + b) \pmod{16}$
2. $d = (d \oplus a) \lll 1$ (Rotate bits left by 1)
3. $c = (c + d) \pmod{16}$
4. $b = (b \oplus c) \lll 2$ (Rotate bits left by 2)

Initial State:

Let the 4-bit variables be initialized as:

$$a = 0011, \quad b = 1010, \quad c = 0101, \quad d = 1100$$

(a) Compute the state (a, b, c, d) after one Quarter Round.

(b) Using the new value of a as the first 4 bits of the keystream, encrypt the plaintext $P = 1111$ by computing $C = P \oplus a$.

Solution 1.
We follow the ARX steps sequentially.

Step 1: Addition ($a = a + b$ mod 16):

$$a = 0011 + 1010 = 3 + 10 = 13 \rightarrow \mathbf{1101}$$

Step 2: XOR and Rotate ($d = (d \oplus a) \lll 1$):

$$d \oplus a = 1100 \oplus 1101 = 0001$$

Rotating 0001 left by 1 bit gives **0010**.

Step 3: Addition ($c = c + d$ mod 16):

$$c = 0101 + 0010 = 5 + 2 = 7 \rightarrow \mathbf{0111}$$

Step 4: XOR and Rotate ($b = (b \oplus c) \lll 2$):

$$b \oplus c = 1010 \oplus 0111 = 1101$$

Rotating 1101 left by 2 bits ($1101 \rightarrow 1011 \rightarrow 0111$) gives **0111**.

Resulting State:

$$\boxed{a = 1101,\ b = 0111,\ c = 0111,\ d = 0010}$$

Encryption:

Using $a = 1101$ and $P = 1111$:

$$C = P \oplus a = 1111 \oplus 1101 = \mathbf{0010}$$

Exercise 2.
Consider the same Quarter Round operations as in Exercise 1. Now, let the initial state be:

$$a = 1111, \quad b = 0001, \quad c = 1000, \quad d = 0110$$

(a) Perform two consecutive Quarter Rounds on this state.

(b) If the final value of d is used as keystream, decrypt the ciphertext $C = 1010$.

Solution 2.
First Quarter Round:

Step 1: $a = 1111 + 0001 = 15 + 1 = 16 \equiv 0 \pmod{16} \rightarrow$ **0000**

Step 2: $d \oplus a = 0110 \oplus 0000 = 0110$, rotate left by 1: **1100**

Step 3: $c = 1000 + 1100 = 8 + 12 = 20 \equiv 4 \pmod{16} \rightarrow$ **0100**

Step 4: $b \oplus c = 0001 \oplus 0100 = 0101$, rotate left by 2: $0101 \rightarrow 1010 \rightarrow$ **0101**

State after first round: $a = 0000, b = 0101, c = 0100, d = 1100$

Second Quarter Round:

Step 1: $a = 0000 + 0101 = 5 \rightarrow$ **0101**

Step 2: $d \oplus a = 1100 \oplus 0101 = 1001$, rotate left by 1: **0011**

Step 3: $c = 0100 + 0011 = 4 + 3 = 7 \rightarrow$ **0111**

Step 4: $b \oplus c = 0101 \oplus 0111 = 0010$, rotate left by 2: $0010 \rightarrow 0100 \rightarrow$ **1000**

Final state: $\boxed{a = 0101,\ b = 1000,\ c = 0111,\ d = 0011}$

Decryption: $P = C \oplus d = 1010 \oplus 0011 =$ **1001**

Exercise 3.
In ChaCha20, the keystream must be unpredictable. Suppose an attacker knows that the initial state is $a = 0000, b = 0000, c = 0000, d = 0000$.

(a) Compute the state after one Quarter Round.

(b) Explain why using all-zero initialization is cryptographically weak.

Solution 3.
Quarter Round with all zeros:

Step 1: $a = 0000 + 0000 = 0 \rightarrow$ **0000**

Step 2: $d \oplus a = 0000 \oplus 0000 = 0000$, rotate left by 1: **0000**

Step 3: $c = 0000 + 0000 = 0 \rightarrow$ **0000**

Step 4: $b \oplus c = 0000 \oplus 0000 = 0000$, rotate left by 2: **0000**

Result: $\boxed{a = 0000,\ b = 0000,\ c = 0000,\ d = 0000}$

Cryptographic weakness: The all-zero state is a fixed point under the Quarter Round operation. This means no matter how many rounds are applied, the state remains all zeros, producing a keystream of all

zeros. This completely breaks the security of the cipher, as an attacker can trivially predict the keystream and decrypt any ciphertext by simply XORing with zeros (which leaves the plaintext unchanged). A secure stream cipher requires high-entropy initialization from a secret key and nonce to ensure unpredictable keystream generation.

Exercise 4.
ChaCha20 uses multiple Quarter Rounds applied to different combinations of state variables. Consider a simplified "column round" that applies Quarter Rounds to two different 4-tuples from an 8-variable state.

Initial state: $s_0 = 0010, s_1 = 0100, s_2 = 0110, s_3 = 1000, s_4 = 1010, s_5 = 1100, s_6 = 1110, s_7 = 0001$

(a) Apply one Quarter Round to (s_0, s_1, s_2, s_3).

(b) Apply one Quarter Round to (s_4, s_5, s_6, s_7).

(c) Compare the final values of s_0 and s_4.

Solution 4.
Quarter Round on (s_0, s_1, s_2, s_3)**:**

Step 1: $s_0 = 0010 + 0100 = 2 + 4 = 6 \rightarrow$ **0110**

Step 2: $s_3 \oplus s_0 = 1000 \oplus 0110 = 1110$, rotate left by 1: **1101**

Step 3: $s_2 = 0110 + 1101 = 6 + 13 = 19 \equiv 3 \pmod{16} \rightarrow$ **0011**

Step 4: $s_1 \oplus s_2 = 0100 \oplus 0011 = 0111$, rotate left by 2: **1101**

Result: $s_0 = 0110, s_1 = 1101, s_2 = 0011, s_3 = 1101$

Quarter Round on (s_4, s_5, s_6, s_7)**:**

Step 1: $s_4 = 1010 + 1100 = 10 + 12 = 22 \equiv 6 \pmod{16} \rightarrow$ **0110**

Step 2: $s_7 \oplus s_4 = 0001 \oplus 0110 = 0111$, rotate left by 1: **1110**

Step 3: $s_6 = 1110 + 1110 = 14 + 14 = 28 \equiv 12 \pmod{16} \rightarrow$ **1100**

Step 4: $s_5 \oplus s_6 = 1100 \oplus 1100 = 0000$, rotate left by 2: **0000**

Result: $s_4 = 0110, s_5 = 0000, s_6 = 1100, s_7 = 1110$

Comparison: $\boxed{s_0 = 0110 = s_4}$

Both values are equal after their respective Quarter Rounds, despite starting from different initial values. This demonstrates that the Quarter Round function can map different inputs to the same output value in certain positions, though the overall states remain different.

Exercise 5.
In a stream cipher attack scenario, an attacker intercepts two ciphertexts that were encrypted using the same keystream (a critical security violation). The ciphertexts are $C_1 = 1011$ and $C_2 = 0110$.

(a) If the attacker knows that the first plaintext is $P_1 = 0100$, recover the keystream.

(b) Using the recovered keystream, decrypt the second ciphertext C_2.

(c) Explain why keystream reuse is catastrophic for stream cipher security.

Solution 5.
Recovering the keystream:

Since $C_1 = P_1 \oplus K$ where K is the keystream, we have:

$$K = C_1 \oplus P_1 = 1011 \oplus 0100 = \mathbf{1111}$$

Decrypting C_2:

Using the recovered keystream:

$$P_2 = C_2 \oplus K = 0110 \oplus 1111 = \mathbf{1001}$$

Why keystream reuse is catastrophic:

Keystream reuse completely breaks stream cipher security because:

1. If an attacker knows one plaintext-ciphertext pair, they can immediately recover the keystream and decrypt all other messages encrypted with the same keystream.
2. Even without knowing any plaintext, the attacker can compute $C_1 \oplus C_2 = (P_1 \oplus K) \oplus (P_2 \oplus K) = P_1 \oplus P_2$, which reveals the XOR of the two plaintexts. This eliminates the key entirely from the equation and can reveal patterns or allow known-plaintext attacks.
3. This is why stream ciphers like ChaCha20 use a nonce (number used once) combined with the key to ensure each message is encrypted with a unique keystream, even when using the same secret key.