RC5 is a block cipher and typically uses these modes of operation.

ECB (Electronic codebook)
CBC (cipher block chaining)
CFB (cipher feedback)
OFB (output feedback)
CTR (counter)

Language - Java
GUI : JavaFX
Mode: ECB
PDF : SEE

64-bit plaintext book
↓ ↓
A ts[0]
B ts[1]
[Rounds: R=12]
↓
1 → For each round i:
A = t((A XOR B) <<< B) ts[2*i]
B = ((B XOR A) <<< A) ts[2*i+1]
64bit ciphetext book.

1. Permutation choice-1 (PC-1) and final Permutation (inverse of PC-2):

PC-1: Permutes a 64 bit key to remove parity bits, leaving 56 bits.

PC-2: (Final permutation choice): Reduces 56-bit combined key halves to 48 bit round keys

2. Compression BOX: 64 → 48 bits

We'll simulate a design that selects 48-bits from a 64 bit input.

3. Expansion BOX: 48 → 64 bits

we simulate a method to expand 48 bits to 64 by repeating some bits

```
from typing import List

def permute(input_bits: str, table: List[int]) -> str:
    """ permute input_bits according to the table."""
    return ''.join(input_bits[i-1] for i in table)
```

#1. permutation choice-1 (PC-1): from 64-bit key to 56 bit key

PC1_TABLE = [

57, 49, 41, 33, 25, 17, 9
1, 58, 50, 42, 34, 26, 18
10, 2, 59, 51, 43, 35, 27
19, 11, 3, 60, 52, 44, 36
63, 55, 47, 39, 31, 23, 15
7, 62, 54, 46, 38, 30, 22
14, 6, 61, 53, 45, 37, 29
21, 13, 5, 28, 20, 12, 4

]

#2. <u>compression Box</u>: compress 64 to 48 bits (custom example)

compression_BOX = [

1, 3, 5, 7, 9, 11, 13, 15,
17, 19, 21, 23, 25, 27, 29, 31,
33, 35, 37, 39, 41, 43, 45, 47,
49, 51, 53, 55, 57, 59, 61, 63,
2, 4, 6, 8, 10, 12, 14, 16,
18, 20, 22, 24, 26, 28, 30, 32

]

Scanned with CamScanner

#3. Expansion Box: Expand 48 bits to 64

(custom example)

Expansion_Box = [

1, 2, 3, 4, 5, 6, 7, 8,

5, 6, 7, 8, 9, 10, 11, 12,

9, 10, 11, 12, 13, 14, 15, 16,

13, 14, 15, 16, 17, 18, 19, 20,

17, 18, 19, 20, 21, 22, 23, 24,

21, 22, 23, 24, 25, 26, 27, 28,

25, 26, 27, 28, 29, 30, 31, 32,

1, 2, 3, 4, 5, 6, 7, 8,

#Example 64 bits input: (ASCII binary string)

input_64bit = "0110110101111010001010101010001

1000110011011101010011111001

1010"

#1. Apply PC-1

key_56bit = permute (input_64bit, PC1_

TABLE)

print(f"56-bit key (pc-1): {key_56bits}")

#2. compress 64→ 48

compressed_48bits = permute(input_64
bit, COMPRESSION_BOX)
print(f"compressed 48-bit output:
{compressed_48bit}")

#3. Expand 48→64

Expanded_64bit = permute(compressed
_48bit, EXPANSION_BOX)
print(f"Expanded 64bit output:
{Expanded_64bit}").

1. Permutation choice-1 (PC-1) and Final Permutation (inverse of PC-2):

PC-1: Permutes a 64bit key to remove parity bits, leaving 56 bits.

PC-2: (Final permutation/choice): Reduces 56-bit combined key halves to 48 bit round keys

2. Compression Box: 64 → 48 bits

we'll simulate a design that selects 48-bits from a 64 bit input.

3. Expansion Box: 48 → 64 bits

we simulate a method to expand 48 bits to 64 by repeating some bits.

```
from typing import List

def permute(input_bits: str, table: List[int]) -> str:
    """ permute input_bits according to the table. """
    return ''.join(input_bits[i-1] for i in table)
```

#1. permutation choice-1 (pc-1): from 64-bit key to 56 bit key

PC1_TABLE = [

57, 49, 41, 33, 25, 17, 9

1, 58, 50, 42, 34, 26, 18

10, 2, 59, 51, 43, 35, 27

19, 11, 3, 60, 52, 44, 36

63, 55, 47, 39, 31, 23, 15

7, 62, 54, 46, 38, 30, 22

14, 6, 61, 53, 45, 37, 29

21, 13, 5, 28, 20, 12, 4

]

#2. compression Box: compress 64 to 48 bits (custom example)

compression_BOX = [

1, 3, 5, 7, 9, 11, 13, 15,

17, 19, 21, 23, 25, 27, 29, 31

33, 35, 37, 39, 41, 43, 45, 47,

49, 51, 53, 55, 57, 59, 61, 63,

2, 4, 6, 8, 10, 12, 14, 16,

18, 20, 22, 24, 26, 28, 30, 32

]

Scanned with CamScanner

#3. Expansion Box: Expand 48bits to 64

(custom example)

Expansion_BOX = [

1,2,3,4,5,6,7,8,

5,6,7,8,9,10,11,12,

9,10,11,12,13,14,15,16

13,14,15,16,17,18,19,20,

17,18,19,20,21,22,23,24,

21,22,23,24,25,26,27,28,

25,26,27,28,29,30,31,32

1,2,3,4,5,6,7,8

]

#Example 64bits input (any binary string)

input_64bit = "0110110101111010001616161700011

1000110011611161610100101111001

1010"

#1. Apply PC-1

key_56bit = permute (input_64bit, PC1_

TABLE)

Q.1: Prove Fermat's little theorem:

If p is a prime number and a is an integer such that $a \not\equiv 0 \pmod{p}$, then:

$$a^{p-1} \equiv 1 \pmod{p}$$

Proof using modular arithmetic:

$S = \{1, 2, 3, \ldots, p-1\}$

multiply each element in S by a, where $a \not\equiv 0 \pmod{p}$.

$S' = \{a.1, a.2, a.3, \ldots, a.(p-1)\} \pmod{p}$

Thus,

$$a.1 \cdot a.2 \cdots a.(p-1) \equiv 1.2 \cdots (p-1) \pmod{p}$$

$$a^{p-1}.(p-1)! \equiv (p-1)! \pmod{p}$$

since $(p-1)! \not\equiv 0 \pmod{p}$

$$a^{p-1} \equiv 1 \pmod{p}$$

∴ Fermat's little theorem is proved.

Given,

$a = 7$

$p = 13$

13 is prime and $7 \not\equiv 0 \pmod{13}$

$7^{12} \equiv 1 \pmod{13}$

So, $7^{12} \bmod 13 = 1$

1. Efficient modular Exponentiation.

$c \equiv m^e \bmod n$ and $m \equiv c^d \bmod n$

2. key insights for Euler's theorem.

$a^{\phi(n)} \equiv 1 \pmod{n}$

$\phi(n) = (p-1)(q-1)$

3. Primality testing:

$a^{p-1} \equiv 1 \pmod{p}$

Q.2: Euler's theorem statement:

compute $\phi(n)$ for $n = 35, 45, 100$.

If n is factored into distinct primes as

$n = p_1^{k_1} \cdot p_2^{k_2} \cdots p_n^{k_n}$

$$\phi(n) = n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_n}\right)$$

$\phi(35)$

$35 = 5 \cdot 7$

$$\phi(35) = 35\left(1-\frac{1}{5}\right)\left(1-\frac{1}{7}\right) = 35 \cdot \frac{4}{5} \cdot \frac{6}{7} = 24$$

$\phi(45)$

$45 = 3^2 \cdot 5$

$$\phi(45) = 45\left(1-\frac{1}{3}\right)\left(1-\frac{1}{5}\right) = 45 \cdot \frac{2}{3} \cdot \frac{4}{5} = 24$$

$\phi(100)$

$100 = 2^2 \cdot 5^2$

$$\phi(100) = 100\left(1-\frac{1}{2}\right)\left(1-\frac{1}{5}\right) = 100 \cdot \frac{1}{2} \cdot \frac{4}{5} = 40$$

Ex: $a^{\phi(n)} \equiv 1 \bmod n$

This is known as Euler's theorem.

Proof:

- $a \in \mathbb{Z}$
- $\gcd(a, n) = 1$
- $R = \{r_1, r_2, \ldots, r, \phi(n)\}$

1. permutes the elements of R.

$$r_1 r_2 \cdots r_{\phi(n)} \equiv (ar_1)(ar_2)\cdots(ar_{\phi(n)}) \bmod n$$

$$r_1 r_2 \cdots r_{\phi(n)} \equiv a^{\phi(n)} \cdot r_1 r_2 \cdots r_{\phi(n)} \bmod n$$

$\therefore 1 \equiv a^{\phi(n)} \mod n$

Q.E.D

Q.2: Solve the system using the Chinese Remainder theorem.

$$\left.\begin{aligned} x &\equiv 2 \mod 3 \\ x &\equiv 3 \mod 4 \\ x &\equiv 1 \mod 5 \end{aligned}\right\}$$

$x \equiv 11 \mod 60 \quad \{N = 3 \cdot 4 \cdot 5 = 60\}$

Step-1:

- $n_1 = 3$
- $n_2 = 4$
- $n_3 = 5$
- $N = n_1 n_2 n_3 = 60$
- $N_i = \frac{N}{n_i}$
- $N_1 = \frac{60}{3} = 20$
- $N_2 = \frac{60}{4} = 15$
- $N_3 = \frac{60}{5} = 12$

$N_i M_i \equiv 1 \mod n_i$

Step-2

$N_1 = 20 \mod n_1$

$M_1 = 2$:

$20.2 = 40 \equiv 1 \mod 3$

$M_1 = 2$

$N_2 = 15 \mod 4$:

$15\, M_2 \equiv 1 \mod 4$

$M_2 = 3$:

$15.3 = 45 \equiv 1 \mod 4$

$M_2 = 3$

$N_3 = 12 \mod 5$:

$12 M_3 \equiv 1 \mod 5$

$M_3 = 3$:

$12.3 = 36 \equiv 1 \mod 5$

$M_3 = 3$

Step-3:

$x \equiv a_1 N_1 M_1 + a_2 N_2 M_2 + a_3 N_3 M_3 \mod N$