



CNS-Presentation

Topic : RC4

Under the guidance of : Dr Manjunath D R

Team - Valmika G , Sharath S




Overview

RC4 stands for Rivest Cipher 4. Ron Rivest invented RC4 in 1987, and it is a stream cipher. Because RC4 is a stream cipher, it encrypts data bytes by bits. Because of its speed and simplicity, RC4 is the most extensively used stream cipher of all the stream ciphers.

While RC4 is known for its ease of use and speed in software, it has been found to have several weaknesses, making it insecure. When the beginning of the output keystream isn't destroyed, or when non-random or linked keys are utilized, it's highly vulnerable. The usage of RC4, in particular, has resulted in relatively insecure protocols such as WEP.

As of 2015, several state cryptologic agencies were suspected of being able to break RC4 when it was employed in the TLS protocol. RFC 7465, published by the Internet Engineering Task Force, prohibits the use of RC4 in TLS, and Mozilla and Microsoft have issued similar recommendations.




Working of RC4

RC4 creates a pseudo-random bit stream (a keystream). These, like any other stream cipher, can be used for encryption by utilizing bit-wise exclusive or to combine it with the plaintext. The same procedure is used for decryption (since exclusive-OR is a symmetric operation).

The cipher uses a secret internal state that is divided into two sections to generate the keystream –

1)Key-Scheduling Algorithm

2)Pseudo random generation algorithm



The key-scheduling algorithm is known to initialize the permutation using a variable-length key, typically between 40 and 256 bits (KSA). A pseudo-random generating technique then generates the stream of bits.

For encryption –

The user enters the Plaintext and a secret key.

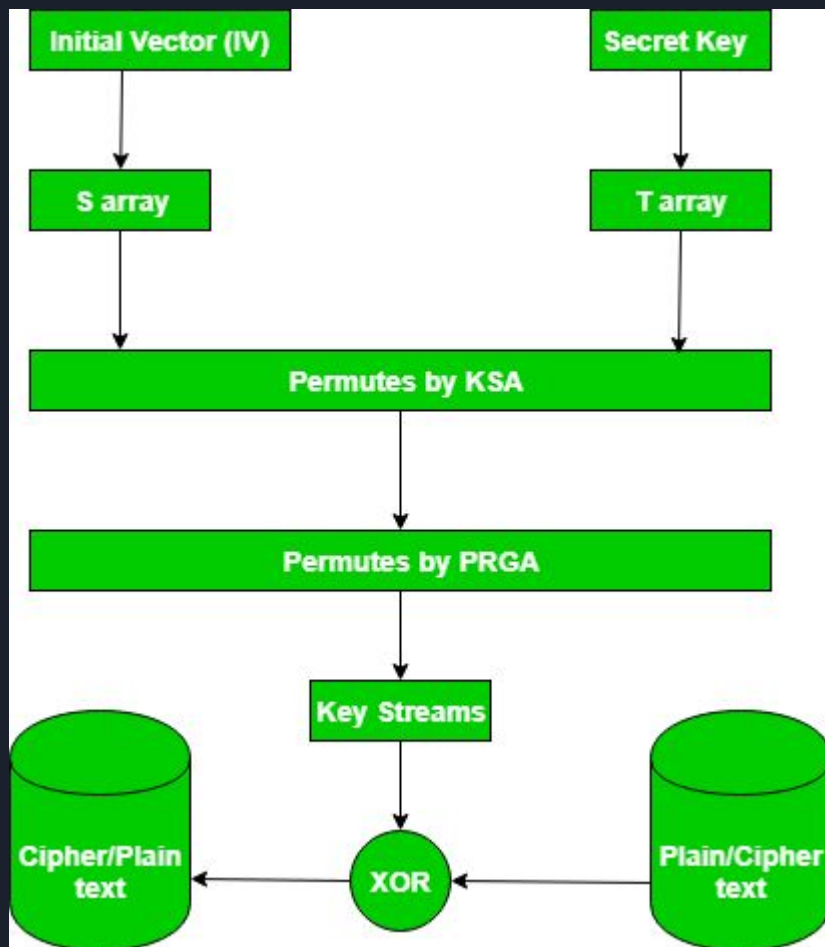
For the secret key entered, the encryption engine creates the keystream using the KSA and PRGA algorithms.

Plaintext is XORed with the generated keystream. Because RC4 is a stream cipher, byte-by-byte XORing is used to generate the encrypted text.

This encrypted text is now sent in encrypted form to the intended recipient.

For Decryption –

The same byte-wise X-OR technique is used on the ciphertext to decrypt it.





Algorithm


Key-Scheduling Algorithm:

for

i = 0 to 255 do $S[i] = i;$

$T[i] = K[i \bmod k - \text{len}];$

we use T to produce the initial permutation of S . Starting with $S[0]$ to $S[255]$, and for each $S[i]$ algorithm swap it with another byte in S according to a scheme dictated by $T[i]$, but S will still contain values from 0 to 255 :



Pseudo random generation algorithm (Stream Generation): Once the vector S is initialized, the input key will not be used. In this step, for each $S[i]$ algorithm swap it with another byte in S according to a scheme dictated by the current configuration of S . After reaching $S[255]$ the process continues, starting from $S[0]$ again

```
i, j = 0;
while (true)
    i = (i + 1) mod
256;
j = (j + S[i]) mod 256;
Swap(S[i], S[j]);
t = (S[i] + S[j]) mod
256;
k = S[t];
```

```

#include <iostream>
#include <vector>

using namespace std;

vector<int> RC4Initialization(const vector<int>& key) {
    vector<int> S(256);
    for (int i = 0; i < 256; ++i) {
        S[i] = i;
    }

    int j = 0;
    for (int i = 0; i < 256; ++i) {
        j = (j + S[i] + key[i % key.size()]) % 256;
        swap(S[i], S[j]);
    }

    return S;
}

vector<int> RC4GenerateKeystream(const vector<int>& S, int length) {
    vector<int> keystream(length);
    int i = 0, j = 0;

    for (int k = 0; k < length; ++k) {
        i = (i + 1) % 256;
        j = (j + S[i]) % 256;
        swap(S[i], S[j]);
        keystream[k] = S[(S[i] + S[j]) % 256];
    }

    return keystream;
}

```

```

vector<int> RC4Encrypt(const vector<int>& plaintext, const vector<int>& key) {
    vector<int> S = RC4Initialization(key);
    vector<int> keystream = RC4GenerateKeystream(S, plaintext.size());

    vector<int> ciphertext(plaintext.size());
    for (int i = 0; i < plaintext.size(); ++i) {
        ciphertext[i] = plaintext[i] ^ keystream[i];
    }

    return ciphertext;
}

int main() {
    vector<int> plaintext = {65, 66, 67}; // Example plaintext
    vector<int> key = {1, 2, 3};           // Example key

    vector<int> ciphertext = RC4Encrypt(plaintext, key);

    cout << "Plaintext: ";
    for (int ch : plaintext) {
        cout << ch << " ";
    }
    cout << endl;

    cout << "Ciphertext: ";
    for (int ch : ciphertext) {
        cout << ch << " ";
    }
    cout << endl;

    return 0;
}

```




Features of RC4

Symmetric key algorithm: RC4 is a symmetric key encryption algorithm, which means that the same key is used for encryption and decryption.

1. **Stream cipher algorithm:** RC4 is a stream cipher algorithm, which means that it encrypts and decrypts data one byte at a time. It generates a key stream of pseudorandom bits that are XORed with the plaintext to produce the ciphertext.
2. **Variable key size:** RC4 supports variable key sizes, from 40 bits to 2048 bits, making it flexible for different security requirements.
3. **Fast and efficient:** RC4 is a fast and efficient encryption algorithm that is suitable for low-power devices and applications that require high-speed data transmission.
4. **Widely used:** RC4 has been widely used in various applications, including wireless networks, secure sockets layer (SSL), virtual private networks (VPN), and file encryption.
5. **Vulnerabilities:** RC4 has several vulnerabilities, including a bias in the first few bytes of the keystream, which can be exploited to recover the key. As a result, RC4 is no longer recommended for use in new applications.



Advantages

1. **Fast and efficient:** RC4 is a very fast and efficient encryption algorithm, which makes it suitable for use in applications where speed and efficiency are critical.
2. **Simple to implement:** RC4 is a relatively simple algorithm to implement, which means that it can be easily implemented in software or hardware.
3. **Variable key size:** RC4 supports variable key sizes, which makes it flexible and adaptable for different security requirements.
4. **Widely used:** RC4 has been widely used in various applications, including wireless networks, secure sockets layer (SSL), virtual private networks (VPN), and file encryption.



DisAdvantages

RC4 is considered weak and no longer recommended for use in secure systems due to several vulnerabilities and weaknesses:

1. **Key Bias:** The initial bytes of the RC4 keystream exhibit statistical biases, making certain key bytes more likely to appear than others. These biases can be exploited in attacks, such as the Fluhrer-Mantin-Shamir (FMS) attack and the PTW attack, to recover parts of the secret key.
2. **Lack of Confusion and Diffusion:** RC4 lacks strong confusion and diffusion properties, which are essential for modern cryptographic algorithms. This makes RC4 susceptible to algebraic and mathematical attacks.
3. **Weak Key Schedule:** The key scheduling algorithm (KSA) used in RC4 has limitations. It is not sufficiently randomizing and can lead to biases in the internal state, affecting the security of the cipher.