THE METHOD OF STRENGTHENING OF STREAM CIPHERS FOR USING THE SAME KEY MULTIPLE TIMES

Trunov D., master, d.n.trunov@gmail.com
Self-employed person, Pokrovsk, Ukraine

There are two main types of ciphers used in modern encryption: block ciphers that encrypt data in blocks of a given size as well as stream ciphers, in which the encryption of individual bits or bytes occurs independently of each other [1]. Stream ciphers can be quite simple, convenient and reliable, but have a significant drawback: repeated use of the same encryption key significantly reduces the strength of the cipher.

In general, a stream cipher encryption scheme might look like this [2]: $c_i = p_i \; XOR \; k_i$, where c_i is the current single element of ciphertext, p_i is the element of plaintext, k_i is the key stream element, and *XOR* is the "exclusive or" operation. For ciphers like a one-time pad [3], the key stream *k* must be truly random, which is rather impractical. For other ciphers, such as Salsa20 [4], the key stream is pseudo-random and is generated according to some algorithm and a given main key.

The danger in such schemes is, for example, that if an attacker has at least one pair of the plaintext *p* and the ciphertext *c*, it is very easy to obtain the key: $k = p \; XOR \; c$. And then he can decrypt all messages encrypted with the same key: $p = c \; XOR \; k$. To avoid this, it is proposed to consider a method for strengthening stream ciphers in the form of an additional algorithm that makes implicit the dependence between the elements of streams *p*, *c* and *k*.

Let's start with an example. Suppose there are two stream ciphers with key streams *k* and *l*. Then the encryption of a single element would look like this: $c_i = p_i \; XOR \; k_i \; XOR \; l_i$. Having c_i and p_i one can only get the result $k_i \; XOR \; l_i$, but not k_i and l_i separately. As long as the *XOR* operation is applied, it does not make much sense, but if we apply some more complex function like $F(p_i, k_i, l_i)$, then extracting k_i and l_i from it can be quite problematic.

To implement such a function in the proposed method we need: some conditional stream cipher *SCipher*, byte arrays *M* for source/encrypted data and *K* for key byte stream, byte arrays *T* for the substitution table and *R* for the reverse substitution table (256 bytes each), and an auxiliary hash bye *H*. We also need auxiliary bytes *B, S, E,* and the cycle counter *i*.

Consider initializing the algorithm on a pseudocode and assume that the *SCipher(K)* function generates a stream of keys of the required length and saves it to the array *K*. Initialization begins with the preparation of *K* and *H*:

```
SCipher(K); // 256 bytes
H = 0;
```

This is followed by a loop of initial filling of the substitution table:

T[i] = i; // for i = 0..255

Next is a loop of table *T* mixing and hash calculating:

```
B = T[i];
T[i] = T[K[i]];
T[K[i]] = B;
H = (H XOR K[i]) >>> 1; // here >>> is cyclic shift to the right
```

At the end there is a loop to form the reverse table of substitutions R:

R[T[i]] = i;

Having a mixed substitution table *T*, its corresponding reverse table *R* and some hash value *H* we can start the encryption and decryption. To do this we need to get a new array of keys *K*:

SCipher(K);

The encryption of the array *M* in this scheme involves not only the *XOR* operation with the keys *K[i]*, but also *XOR* with the hash *H*, encryption using the substitution table *T*, as well as a new change of the hash *H* and exchange of two elements of the table *T* by dynamically determined addresses. Encryption loop on the pseudocode:

```
S = M[i] XOR K[i];
E = (T[S] XOR H) <<< 1; // here <<< is cyclic shift to the left
M[i] = E;
H = (T[E] XOR T[H]) <<< 3; // new hash value
B = T[H];              // exchange of two elements of the table T
T[H] = T[S];
T[S] = B;
```

In the decryption loop, we only need to change the order of operations over the elements of the array *M* and use the inverse table *R* instead of *T*:

```
E = M[i];
S = (E >>> 1) XOR H;
S = R[S];
M[i] = S XOR K[i];
```

Also before the exchange in the table *T* we need to exchange in the table *R*:

B = R[T[H]];
R[T[H]] = R[T[S]];
R[T[S]] = B;

The idea is that even having the original and encrypted values of any element *M[i]*, it is impossible to reliably determine the exact values of the current hash *H*, the key *K[i]* and the entire table *T* or at least a part of it. Even more so, it is impossible to reliably establish these values, having several ciphertexts encrypted with the same key, but not having at least one plaintext.

Is it possible to somehow save all possible combinations of *H*, *K[i]* and substitutions *E = T[S]* and, having enough pairs of ciphertexts and plaintexts, gradually weed out the unfit variants and soon come to the only correct one? Theoretically, it is possible. However, it may require so many computational resources that it would be easier to go through all the possible variants of the main key of the SCipher algorithm (brute force attack on SCipher).

Thus, if we initially take a strong stream cipher as *SCipher*, then strengthening it with the above method (or a similar one) will allow not only to safely use the same key multiple times, but also to keep the convenience and relative simplicity of the byte-by-byte stream cipher.

References

1. Gary C. Kessler. An Overveiw of Cryptography. – URL: https://www.garykessler.net/library/crypto.html

2. Stream cipher: Wikipedia, the free encyclopedia. – URL: https://en.wikipedia.org/wiki/Stream_cipher

3. One-time pad: Wikipedia, the free encyclopedia. – URL: https://en.wikipedia.org/wiki/One-time_pad

4. Daniel J. Bernstein. The Salsa20 family of stream ciphers: Department of Mathematics, Statistics, and Compute science. The University of Illinois at Chicago. – URL: https://cr.yp.to/snuffle/salsafamily-20071225.pdf

Анотація

Розглянуто метод посилення потокових шифрів у вигляді додаткового алгоритму, що ускладнює зламування шифру при багаторазовому застосуванні одного й того ж ключа шифрування. Показано, що можливий злом такого алгоритму буде складнішим за метод грубої сили відносно лише потокового шифру.

Ключові слова: потоковий шифр, посилення, ключ шифрування, злом.

Аннотация

Рассмотрен метод усиления потоковых шифров в виде дополнительного алгоритма, усложняющего взлом шифра при многократном применении одного и того же ключа шифрования. Показано, что возможный взлом такого алгоритма будет сложнее метода грубой силы по отношению только к потоковому шифру.

Ключевые слова: потоковый шифр, усиление, ключ шифрования, взлом.

Abstract

A method of strengthening stream ciphers in the form of an additional algorithm is considered. The method complicates cipher breaking when the same cipher key is used multiple times. It is shown that the possible breaking of such an algorithm will be more complicated than the brute force method in relation to the stream cipher only.

Keywords: stream cipher, strengthening, encryption key, breaking.