

Full-Time-Pad Symmetric Stream Cipher

Improved One-Time-Pad Encryption Scheme

Taha Canturk
kibnakanoto@protonmail.com

2024-05-20

Version 1.0

License to copy this document is granted provided it is identified as "Full-Time-Pad", in all material mentioning or referencing it.

Abstract

One-Time-Pad Encryption Scheme is a secure algorithm but there are 2 main security risks. One, a key cannot be reused. Two, plaintext length equals key length which is very inefficient when dealing with long plaintexts. These 2 security risks only exist due to a lack of confusion and diffusion per ciphertext. As denoted by Claude Shannon in the report he published in 1945, A Mathematical Theory of Cryptography, A secure cryptographic algorithm requires confusion and diffusion. The **Full-Time-Pad** symmetric stream cipher is developed based on the **One-Time-Pad** with solutions to the security risks while maintaining high speed computation. To achieve diffusion, the key is permuted in it's byte array form using a constant permutation matrix. To achieve the confusion, the key is manipulated in it's 32-bit integer representation using Modular **A**ddition in F_p , Bitwise **R**otations, and **X**or (**ARX**). The permutation guarantees that every time there is a manipulation, each 32-bit number is made up of a different byte order.

Contents

1	Introduction	1
1.1	Pre-requisite Terminology	1
1.2	Applications	1
1.3	Key Generation	1
1.4	Prerequisite Mathematics	2
1.5	Vector Permutation	2
2	Security Vulnerabilities	2
2.1	Brute-Force	2
2.1.1	Birthday Problem	3
2.1.2	Denial of Service (DoS)	6
2.2	Reverse Engineering the Transformation	6
2.3	Collision-Resistance	6
2.3.1	Different Permutation Matrices	6
2.3.2	Number of Rounds	6
2.3.3	Constant - F_p - Prime Galois Field Size	6
2.3.4	Constant - r - Dynamic Rotation Constant	6
3	Hashing	6
3.1	Diffusion - Permutation	6
3.1.1	Vector Permutation	6
3.1.2	Dynamic vs. Static	6
3.2	Dynamic Matrix Permutation	6
3.2.1	Deravation	6
3.2.2	Dynamic Permutation Matrix Values	7
3.2.3	Other Options	7
3.3	Confusion - ARX	7
3.3.1	A - Modular Addition	7
3.3.2	R - Bitwise Rotation	7
3.3.3	X - XOR	7
3.4	Key Transformation	7
4	Cipher	7
4.1	Transformation	7
4.2	Avalanche Effect - Plaintext	7
4.2.1	Encryption Index	7
4.3	Long Plaintexts	7

1 Introduction

1.1 Pre-requisite Terminology

Key	32-byte random array that's transformed, then hashed before XORed with the plaintext to encrypt
Symmetric	Same key is used for encryption and decryption
Stream	Plaintext is encrypted without separating it into blocks
Plaintext	Plain data before encryption
Ciphertext	Encrypted plaintext
Cipher	Encryption algorithm. Plaintext is transformed into a ciphertext that can only be reversed with a key
Diffusion	plaintext/key is spread out in the ciphertext
Confusion	The ciphertext has no possible statistical analysis, or cryptanalysis to determine the plaintext
Bit	0 or 1. Smallest discrete unit for computation
Byte	8-bit number
Galois Field	Finite Field where there are only limited number of numbers. Only prime galois fields (F_p) are used where size of the field is denoted by prime number p
Avalanche Effect	An aspect of diffusion. If smallest unit (1 bit) of data is changed, the ciphertext changes in an unrecognizable way.

1.2 Applications

1.3 Key Generation

The 32-byte key should be generated using a cryptographically secure method, including but not limited to cryptographic random number generators and Elliptic Cryptography Diffie Hellman (ECDH) protocol with Hash-based Key Derivation Function (HKDF)

1.4 Prerequisite Mathematics

1.5 Vector Permutation

2 Security Vulnerabilities

In One-Time-Pad, key isn't reusable. Here is the proof:

```
let  $m_1, m_2$  be 2 plaintexts
let  $k$  be the key
let  $c_1 = m_1 \oplus k$ 
let  $c_2 = m_2 \oplus k$ 
 $c_1 \oplus c_2 = (m_1 \oplus k) \oplus (m_2 \oplus k)$ 
 $c_1 \oplus c_2 = m_1 \oplus m_2$ 
```

Since the key is reused, the 2 ciphertext's XORed factor out the key since $k \oplus k = 0$. Using cryptanalysis, the 2 plaintexts can be found.

For $c_1 \oplus c_2 = m_1 \oplus m_2$ to not hold true, for each encryption, the key needs to be different. If k is transformed each time so that it has an avalanche effect. Even with no confusion, it would still be secure since $k' \oplus k \neq 0$ where k' is transformed key.

But there is another concern,

What if the plaintext and ciphertext are known, then it is possible to find k so don't use k without transformation, since $\text{plaintext} \oplus \text{ciphertext} = \text{key}$. So for each plaintext, key needs to be transformed irreversibly and it also requires confusion since if k' is found, k is still unknown but if k is found, then all instances of k'_n are known, which means that:

```
 $k'_1 = \text{hash}(k + 1)$  where  $\text{hash}()$  is an irreversible transformation
 $k'_2 = \text{hash}(k + 2)$ 
 $c_1 \oplus c_2 = (m_1 \oplus k'_1) \oplus (m_2 \oplus k'_2)$ 
 $c_1 \oplus c_2 \neq m_1 \oplus m_2$ 
 $m_1 \oplus c_1 = k'_1$ 
 $m_2 \oplus c_2 = k'_2$ 
 $k'_1, k'_2$  are calculated using an irreversible hashing algorithm
```

\therefore the Full-Time-Pad Cipher requires both diffusion and confusion

2.1 Brute-Force

Due to the use of a galois field. The total number of combinations per 256-bit key isn't $a = 2^{256}$, but rather $b = 4294967291^8$ where $p = 4294967291$ for arithmetic in F_p and there are 8 32-bit numbers in a 256-bit key.

```
 $a = 115792089237316195423570985008687907853269984665640564039457584007913129639936_{10}$ 
 $b = 115792088158918333131516597762172392628570465465856793992332884130307292657121_{10}$ 
let  $\Delta = a - b$ 
 $\Delta = 1078397862292054387246515515224699519199783770047124699877605836982815_{10}$ 
```

So the difference Δ is a somewhat large integer. The number of combinations with a galois field is lower than without a galois field ($b < a$). This isn't a big concern as their difference measured exponentially is only around $2^{\log_2 \Delta} \approx 2^{229}$ which means that their difference is around 2^{229} , this is a negligible difference as the difference between 2^{230} and 2^{229} is also huge.

∴ Using a galois field doesn't negatively impact number of combinations in terms of brute force as the total number of combinations when using a galois field vs not is a negligible amount

2.1.1 Birthday Problem

The birthday problem is a paradox. It goes as follows: how many people are required so that there is more than 50% chance that at least 2 people have the same birthday.

The answer is an unexpected 23 people.

In the context of this encryption algorithm, it might be a concern, as number of key reused (with transformation) increase, the chances of finding the key increase:

let V_c be the number of combinations per key without order and repetitions
 let k be the number of keys needed for hash(key) to have a 50% chance to equal another hash(key)
 let V_t be the number of combinations per key with order and repetitions

$$V_c = \frac{b!}{(b-k)!} = \frac{4294967291^8!}{(4294967291^8 - k)!}$$

$$V_t = b^k = 4294967291^{8^k}$$

$$P(A) = \frac{V_c}{V_t}$$

$$P(A) = \frac{\frac{b!}{(b-k)!}}{b^k}$$

$$P(B) = 1 - P(A) = 50\%$$

$$P(A) = 1 - 50\%$$

$$1 - 50\% = \frac{\frac{b!}{(b-k)!}}{b^k}$$

$$\frac{1}{2}b^k = \frac{b!}{(b-k)!}$$

$$\text{since } 50\% = \frac{1}{2}$$

$$\log_b \frac{1}{2}b^k = \log_b \frac{b!}{(b-k)!}$$

$$\log_b \frac{1}{2} + \log_b b^k = \log_b b! - \log_b (b-k)!$$

$$0 = \log_b b! - \log_b (b-k)! - \log_b \frac{1}{2} - k$$

$$\text{since } \log_b b^k = k$$

According to Ramanujan's Approximation:

$$\log_b b! \approx \frac{b \ln b - b + \frac{\ln \left[\frac{1}{\pi^3} + b(1+4b(1+2b)) \right]}{6}}{\ln b} + \frac{\ln \pi}{2}$$

And

$$\log_b(b-k)! \approx \frac{(b-k) \ln(b-k) - (b-k) + \frac{\ln \left[\frac{1}{\pi^3} + (b-k)(1+4(b-k)(1+2(b-k))) \right]}{6}}{\ln b} + \frac{\ln \pi}{2}$$

Recall:

$$0 = \log_b b! - \log_b(b-k)! - \log_b \frac{1}{2} - k \quad \text{isolate } \log_b(b-k)!$$

$$\log_b(b-k)! = \log_b b! - \log_b \frac{1}{2} - k$$

Combine both equations for $\log_b(b-k)!$:

$$\log_b b! - \log_b \frac{1}{2} - k \approx \frac{(b-k) \ln(b-k) - (b-k) + \frac{\ln \left[\frac{1}{\pi^3} + (b-k)(1+4(b-k)(1+2(b-k))) \right]}{6}}{\ln b} + \frac{\ln \pi}{2}$$

$$\begin{aligned} \frac{b \ln b - b + \frac{\ln \left[\frac{1}{\pi^3} + b(1+4b(1+2b)) \right]}{6}}{\ln b} + \frac{\ln \pi}{2} - \log_b \frac{1}{2} - k &\approx \frac{(b-k) \ln(b-k) - (b-k)}{\ln b} + \\ &+ \frac{\frac{\ln \left[\frac{1}{\pi^3} + (b-k)(1+4(b-k)(1+2(b-k))) \right]}{6}}{\ln b} + \frac{\ln \pi}{2} \end{aligned}$$

$$\begin{aligned}
& \frac{b \log b + \frac{\ln \left[\frac{1}{\pi^3} + b(1+4b(1+2b)) \right]}{6} + \frac{\ln \pi}{2} - \ln b \log_b \frac{1}{2} - \ln bk}{\ln b} \approx \frac{(b-k) \ln(b-k) + k}{\ln b} + \\
& \quad + \frac{\ln \left[\frac{1}{\pi^3} + (b-k)(1+4(b-k)(1+2(b-k))) \right]}{6} + \frac{\ln \pi}{2} \\
\text{let } C &= b \ln b + \frac{\ln \left[\frac{1}{\pi^3} + b(1+4b(1+2b)) \right]}{6} - \ln b \log_b \frac{1}{2} \approx (b-k) \ln(b-k) + k + \ln bk \\
& \quad + \frac{\ln \left[\frac{1}{\pi^3} + (b-k)(1+4(b-k)(1+2(b-k))) \right]}{6} \\
\text{let } f(k) &= (b-k) \ln(b-k) + k + \ln bk + \frac{\ln \left[\frac{1}{\pi^3} + (b-k)(1+4(b-k)(1+2(b-k))) \right]}{6} - C = 0
\end{aligned}$$

$\therefore f(k)$ can be used to evaluate how many keys it would take so that 2 hashes have a 50% chance of being equal. $f(k)$ can be evaluated using the secant algorithm

After running `test/secant.py`, given the parameters:

Based on Wikipedia Article: Birthday Attack, we can approximate x_0 and x_1

$$x_0 = \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \times \ln(2) \times b} \text{ (due to Approximation of number of people)}$$

$$x_1 = \sqrt{b} \text{ (due to square approximation)}$$

error tolerance: $e = 1 \times 10^{-200}$

for $b = 4294967291^8$, we get $k_1 = 400651867432320527534628274526034254879$ for the root.

And for $b = 2^{256}$, we get $k_2 = 400651869298001176472314306405665023048$ for the root

So then $\Delta k = k_2 - k_1 = 1865680648937686031879630768169 \approx 2^{101}$ Since the difference between k_1 and k_2 is negligible (2^{101} isn't big considering the magnitude of the numbers used). We can conclude that using a galois field doesn't increase risk of birthday attacks which justifies the use of Galois fields to increasing avalanche effect.

2.1.2 Denial of Service (DoS)

2.2 Reverse Engineering the Transformation

2.3 Collision-Resistance

2.3.1 Different Permutation Matrices

2.3.2 Number of Rounds

2.3.3 Constant - F_p - Prime Galois Field Size

2.3.4 Constant - r - Dynamic Rotation Constant

3 Hashing

3.1 Diffusion - Permutation

3.1.1 Vector Permutation

3.1.2 Dynamic vs. Static

3.2 Dynamic Matrix Permutation

3.2.1 Deravation

Algorithm 1 Dynamic Permutation Matrix Deravation Pseudo-code

```
1: Input: an array of incrementing numbers (0-31)  $A$ 
2: Output: Most Efficient Permutation Matrix  $V$  ( $16 \times 32$ )
3: Begin
4:  $P \leftarrow \text{copy of } A$ 
5: for  $k = 0$  to 4 do
6:   for  $i = 0$  to 8 do
7:      $P_i \leftarrow A_{i \times 4}$ 
8:      $P_{i+8} \leftarrow A_{i \times 4 + 1}$ 
9:      $P_{i+16} \leftarrow A_{i \times 4 + 2}$ 
10:     $P_{i+24} \leftarrow A_{i \times 4 + 3}$ 
11:   end for
12:    $A \leftarrow \text{copy of } P$ 
13:    $V.append(P)$ 
14:    $C \leftarrow \text{copy of } P$ 
15:   for  $m = 0$  to 3 do
16:     for  $i = 0$  to 8 do
17:       for  $n = 0$  to 4 do
18:          $P_{i \times 4 + n} \leftarrow C_{(1+n+m) \bmod 4 + i \times 4}$ 
19:       end for
20:     end for
21:      $V.append(P)$ 
22:   end for
23:    $A \leftarrow \text{copy of } P$ 
24: end for
25: Return  $V$ 
```

Python code is in the test/perm.py

3.2.2 Dynamic Permutation Matrix Values

$$\left(\begin{array}{cc} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 & 1 & 5 & 9 & 13 & 17 & 21 & 25 & 29 & 2 & 6 & 10 & 14 & 18 & 22 & 26 & 30 & 3 & 7 & 11 & 15 & 19 & 23 & 27 & 31 \\ 4 & 8 & 12 & 0 & 20 & 24 & 28 & 16 & 5 & 9 & 13 & 1 & 21 & 25 & 29 & 17 & 6 & 10 & 14 & 2 & 22 & 26 & 30 & 18 & 7 & 11 & 15 & 3 & 23 & 27 & 31 & 19 \\ 8 & 12 & 0 & 4 & 24 & 28 & 16 & 20 & 9 & 13 & 1 & 5 & 25 & 29 & 17 & 21 & 10 & 14 & 2 & 6 & 26 & 30 & 18 & 22 & 11 & 15 & 3 & 7 & 27 & 31 & 19 & 23 \\ 12 & 0 & 4 & 8 & 28 & 16 & 20 & 24 & 13 & 1 & 5 & 9 & 29 & 17 & 21 & 25 & 14 & 2 & 6 & 10 & 30 & 18 & 22 & 26 & 15 & 3 & 7 & 11 & 31 & 19 & 23 & 27 \\ 12 & 28 & 13 & 29 & 14 & 30 & 15 & 31 & 0 & 16 & 1 & 17 & 2 & 18 & 3 & 19 & 4 & 20 & 5 & 21 & 6 & 22 & 7 & 23 & 8 & 24 & 9 & 25 & 10 & 26 & 11 & 27 \\ 28 & 13 & 29 & 12 & 30 & 15 & 31 & 14 & 16 & 1 & 17 & 0 & 18 & 3 & 19 & 2 & 20 & 5 & 21 & 4 & 22 & 7 & 23 & 6 & 24 & 9 & 25 & 8 & 26 & 11 & 27 & 10 \\ 13 & 29 & 12 & 28 & 15 & 31 & 14 & 30 & 1 & 17 & 0 & 16 & 3 & 19 & 2 & 18 & 5 & 21 & 4 & 20 & 7 & 23 & 6 & 22 & 9 & 25 & 8 & 24 & 11 & 27 & 10 & 26 \\ 29 & 12 & 28 & 13 & 31 & 14 & 30 & 15 & 17 & 0 & 16 & 1 & 19 & 2 & 18 & 3 & 21 & 4 & 20 & 5 & 23 & 6 & 22 & 7 & 25 & 8 & 24 & 9 & 27 & 10 & 26 & 11 \\ 29 & 31 & 17 & 19 & 21 & 23 & 25 & 27 & 12 & 14 & 0 & 2 & 4 & 6 & 8 & 10 & 28 & 30 & 16 & 18 & 20 & 22 & 24 & 26 & 13 & 15 & 1 & 3 & 5 & 7 & 9 & 11 \\ 31 & 17 & 19 & 29 & 23 & 25 & 27 & 21 & 14 & 0 & 2 & 12 & 6 & 8 & 10 & 4 & 30 & 16 & 18 & 28 & 22 & 24 & 26 & 20 & 15 & 1 & 3 & 13 & 7 & 9 & 11 & 5 \\ 17 & 19 & 29 & 31 & 25 & 27 & 21 & 23 & 0 & 2 & 12 & 14 & 8 & 10 & 4 & 6 & 16 & 18 & 28 & 30 & 24 & 26 & 20 & 22 & 1 & 3 & 13 & 15 & 9 & 11 & 5 & 7 \\ 19 & 29 & 31 & 17 & 27 & 21 & 23 & 25 & 2 & 12 & 14 & 0 & 10 & 4 & 6 & 8 & 18 & 28 & 30 & 16 & 26 & 20 & 22 & 24 & 3 & 13 & 15 & 1 & 11 & 5 & 7 & 9 \\ 19 & 27 & 2 & 10 & 18 & 26 & 3 & 11 & 29 & 21 & 12 & 4 & 28 & 20 & 13 & 5 & 31 & 23 & 14 & 6 & 30 & 22 & 15 & 7 & 17 & 25 & 0 & 8 & 16 & 24 & 1 & 9 \\ 27 & 2 & 10 & 19 & 26 & 3 & 11 & 18 & 21 & 12 & 4 & 29 & 20 & 13 & 5 & 28 & 23 & 14 & 6 & 31 & 22 & 15 & 7 & 30 & 25 & 0 & 8 & 17 & 24 & 1 & 9 & 16 \\ 2 & 10 & 19 & 27 & 3 & 11 & 18 & 26 & 12 & 4 & 29 & 21 & 13 & 5 & 28 & 20 & 14 & 6 & 31 & 23 & 15 & 7 & 30 & 22 & 0 & 8 & 17 & 25 & 1 & 9 & 16 & 24 \\ 10 & 19 & 27 & 2 & 11 & 18 & 26 & 3 & 4 & 29 & 21 & 12 & 5 & 28 & 20 & 13 & 6 & 31 & 23 & 14 & 7 & 30 & 22 & 15 & 8 & 17 & 25 & 0 & 9 & 16 & 24 & 1 \end{array} \right)$$

3.2.3 Other Options

3.3 Confusion - ARX

3.3.1 A - Modular Addition

3.3.2 R - Bitwise Rotation

3.3.3 X - XOR

3.4 Key Transformation

4 Cipher

4.1 Transformation

4.2 Avalanche Effect - Plaintext

4.2.1 Encryption Index

4.3 Long Plaintexts