A New Approach in One Time Pad Key Management

Gilda R. Bansimba, Regis F. Babindamana and Peter Kidoudou

Université Marien Ngouabi
Faculté des Sciences et Techniques
BP: 69, Brazzaville, Congo

This article is distributed under the Creative Commons by-nc-nd Attribution License. Copyright © 2020 Hikari Ltd.

Abstract

Let \((p, c, k, \varepsilon_k, d_k)\) be the One Time Pad cryptosystem. We consider \(p = c = k\).

In this paper, we improve the key management with introduction of the concept of mathematical key footprint to ensure the uniqueness of every generated key without storing it. We also combine the default operating system’s randomness Application Programming Interface (API) CSPRNG, with some further local system entropy parameters mainly the micro level of noise and environment brightness to enhance key generation randomness using any personal device. We introduce the use of negative keys to enlarge the key space \(K\) and give related algorithms.

Keywords: one time pad, random number generator, key management, key-footprint, security, cryptography, unbreakable cipher

1 Introduction

The One time pad (OTP) encryption scheme is a particular case of the historical Vigenere encryption method, a polyalphabetical subsitution encryption where the key length is the same as the message length. It’s proven [9] that if a key is "truly" random and used only once (avoiding pattern analysis attacks) while performing one time pad, then one can expect
the communication to be unbreakable. Furthermore, this raises an important issue of key management and distribution [1] that makes it’s implementation very difficult in practical. This is particularly due to the fact that "truly" randomness in a cryptographic view can only be achieved in physical phenomenon either with chaotic behaviour or moreover in quantum phenomenon [10]. However it’s not all, an efficient implementation of OTP also requires the storage of every used key to avoid collisions to forthcoming generated keys. Following this, We improve the key management with introduction of the concept of key footprint. We enhance the key generation randomness on a personal device environment by combining the operating system’s randomness Application Programming Interface (API), the so-called Cryptographically Secure Pseudorandom Number Generator (CSPRNG) module with some further local system entropy parameters to generate better randomness.

The main idea of this paper is to propose an efficient implementation of one time pad at the key management level on a personal device environment. Here, we present the main contributions of the paper:

- We introduce the notion of "key-footprint" that is a function that keeps a trace or footprint of a key without storing it. This both reduces risks in case of a malicious database access and provide a secure and efficient verification way to ensure uniqueness of a generated key. (see definition 3.1, proposition 3.2, proposition 3.3 and algorithm 1)

- We enhance the key generation algorithm by combining the CSPRNG operating system’s randomness Application Programming Interface (API) with two cross-platform system entropy parameters mainly environment brightness and microphone noise level, but this can be extended to camera number of colours, letters frequency of appearance, keyboard typing speed, mouse position, battery level, usb ports state, sensor, light,... dependently of the operating environment. We give an algorithm (see algorithm 2, example 4.1 and remark 4.2)

- We introduce the use of negative keys more generally in $\mathbb{Z}$, either positive or negative to enlarge the key space and give effective algorithms. (see algorithms 3 and 4)

Next is the glossary of different notations and abbreviations used in this paper.

**Notations:**

$\bigcup$: the union

$\oplus$: the concatenation

$\mathbb{P}$: the plaintext space
C: the ciphertext space
K: the key space
ε\_k: the encryption method (algorithm) using the key k
D\_k: the decryption method (algorithm) using the key k
PRNG: Pseudo Random Number Generator
CSPRNG: Cryptographically Secure Pseudo Random Number Generator
Z: ring of integers (\{\cdots, -3, -2, -1, 0, 1, 2, 3, \cdots\})
(F_2)^l: the field of binary representation base of length l

2 One Time Pad

Definition 2.1 Consider the keyword K with m characters and k = (k_1, k_2, ..., k_m) its corresponding numeric vector.
Given a message M = x_1x_2x_3...x_m of the same length as the keyword, to encrypt. Let P = C = K, then we define the encryption and decryption methods as follow:
e_k(x_1, x_2, ..., x_m) = (x_1 + k_1, x_2 + k_2, ..., x_m + k_m) : encryption
d_k(y_1, y_2, ..., y_m) = (y_1 - k_1, y_2 - k_2, ..., y_m - k_m) : decryption
and we easily verify the bijection that d\_k(e\_k(x)) = x, since
d\_k(e\_k(x)) = d\_k(x_1 + k_1, x_2 + k_2, ..., x_m + k_m)
= (x_1 + k_1 - k_1, x_2 + k_2 - k_2, ..., x_m + k_m - k_m)
= (x_1, x_2, ..., x_m)
= x.

At the binary level, we operate in the field (F_2)^l ∼ (Z/2Z)^l = \{0, 1\}^l where l represents the length. We set P = C = K = (F_2)^l and define ⊕ and ⊖ respectively the addition and substraction in (F_2)^l.
Given a file m (audio, text, video, ...) to be encrypted, we encode m into binary form and separate in units of l bits each, then we have m = (m_1, m_2, ..., m_n).
Let K be the key of same number of bits as m. We separate K in units of l bits each, K = (k_1, k_2, ..., k_n).
Hereby we operate the XOR operation in (F_2)^l on each unit.
Encryption:
One-Time_Pad(m, k) = (m_1⊕k_1, m_2⊕k_2, ..., m_n⊕k_n) = (m'_1, m'_2, ..., m'_n) = m'

Decryption:
One-Time_Pad(m', k) = (m'_1⊕k_1, m'_2⊕k_2, ..., m'_n⊕k_n) = (m_1, m_2, ..., m_n) = m
3 Key FootPrint

Definition 3.1 Given a number \( k = x_1x_2\cdots x_n \) of \( n \) digits in decimal base, \( n \geq 2 \).

We define the footprint of the key \( k \) that we denote \( FP(k) \), the 3-dimensional vector \( (\alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}^3 \) such that
\[
\alpha_1 = \sum_{i=1}^{n} x_i, \quad \alpha_2 = \sum_{i \text{ even}}^{n} x_i, \quad \alpha_3 = \sum_{i=1}^{n} x_i^i
\]

Proposition 3.2
1. \( FP(-k) = -FP(k) \)

Proof
\[
FP(-k) = (-\sum_{i=1}^{n} x_i, -\sum_{i \text{ even}}^{n} x_i, -\sum_{i=1}^{n} x_i^i)
\]
since \( -k = -x_1x_2\cdots x_n \).

As \( \sum_{i=1}^{n} x_i = \alpha_1 \),
\( \sum_{i \text{ even}}^{n} x_i = \alpha_2 \) and \( \sum_{i=1}^{n} x_i^i = \alpha_3 \) then \( FP(-k) = (-\alpha_1, -\alpha_2, -\alpha_3) = -(\alpha_1, \alpha_2, \alpha_3) \).

Therefore \( FP(-k) = -FP(k) \)

Proposition 3.3 For every triplet \( (\alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}^3 \), if there exits a number \( k \) such that \( FP(k) = (\alpha_1, \alpha_2, \alpha_3) \), then \( k \) is unique.

Proof Assume there are two numbers \( k_1 = x_1x_2\cdots x_n \) and \( k_2 = y_1y_2\cdots y_m \) of respectively \( n \) and \( m \) digits such that \( k_1 \neq k_2 \) with \( FP(k_1) = FP(k_2) = \).
there’s a contradiction from the definition 3.1, since $k_1$ and $k_2$ don’t have the same number of digits, $\alpha_{1k_1}$ could be equal to $\alpha_{1k_2}$, as in $k_1 = 11243$, $k_2 = 218$ but not the even position numbers, ie $\alpha_{2k_1} \neq \alpha_{2k_2}$ then this implies $\mathcal{FP}(k_1) \neq \mathcal{FP}(k_2)$.

If $n = m$:

Then $\mathcal{FP}(k_1) = (\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} y_i, \sum_{i=1}^{n} x_i^i)$ and $\mathcal{FP}(k_2) = (\sum_{i=1}^{m} y_i, \sum_{i=1}^{m} x_i, \sum_{i=1}^{m} y_i^i)$ Since $\mathcal{FP}(k_1) = \mathcal{FP}(k_2) \implies (\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i, \sum_{i=1}^{n} x_i^i) = (\sum_{i=1}^{m} y_i, \sum_{i=1}^{m} y_i, \sum_{i=1}^{m} y_i^i)$

$(1)$ could hold like in $k_1 = 2732301$ and $k_2 = 6253110$ as $\alpha_{1k_1} = \alpha_{1k_2} = 18$ but not $(2)$ nor $(3)$.

Also $(2)$ could hold like in $k_1 = 2017569$ and $k_2 = 2413579$ as $\alpha_{2k_1} = \alpha_{2k_2} = 17$ but not $(1)$ and $(3)$.

More generally, $(1), (2), (3)$ yield a Contradiction, since $k_1 \neq k_2$ ie $x_1 x_2 \cdots x_n \neq y_1 y_2 \cdots y_m$, $\exists i \in [1, n]$ such that $x_i \neq y_i$.

hence $k_1 = k_2$, ie $x_1 = y_1, x_2 = y_2, \cdots, x_n = y_m$.

Now we show that recovering the key $k$ from the knowledge of $(\alpha_1, \alpha_2, \alpha_3)$ such that $(\alpha_1, \alpha_2, \alpha_3) = \mathcal{FP}(k)$ is equivalent to solving the system:

$$
\begin{align*}
\sum_{i=1}^{n} x_i &= \sum_{i=1}^{m} y_i \quad (1) \\
\sum_{i=1}^{n} x_i^i &= \sum_{i=1}^{m} y_i^i \quad (2) \\
\sum_{i=1}^{n} x_i^2 + \cdots + x_n^2 &= \sum_{i=1}^{m} y_i^2 + \cdots + y_m^2 \quad (3)
\end{align*}
$$

the obtained system $(\ast)$ is given by (3) and (4) and is a two-equations nonlinear system of degree $n$ with $n$ unknowns.

**Algorithm 1: Footprint computation.**

**Input:** key $= x_1 x_2 \cdots x_n \in \mathcal{K}$

**Output:** $(\alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}^3$ such that

$\alpha_1 := \sum_{i=1}^{n} x_i, \alpha_2 := \sum_{i=even}^{n} x_i, \alpha_3 := \sum_{i=even}^{n} x_i^i$

1. **Initialization:** $\alpha_1 = \alpha_2 = \alpha_3 = j = 0$
2. for $x_i$ in key do
   3. $j \leftarrow j + 1$
   4. $\alpha_1 \leftarrow \alpha_1 + x_i$
   5. if $j$ is even then
      6. $\alpha_2 \leftarrow \alpha_2 + x_i$
   7. end
   8. $\alpha_3 \leftarrow \alpha_3 + x_i^j$
9. end
10. if $key < \emptyset$ then
11. $\quad$ return $(-\alpha_1, -\alpha_2, -\alpha_3)$
12. end
13. return $(\alpha_1, \alpha_2, \alpha_3)$
Example 3.4 Given the 1024 bit key

Given the 1024 bit key

\[ k = 1343953790552502795798195948217077919135014389676600354625097596284902011018597111889978 \\
669561386813770297651099698780476426246492340704458250086240572874006568502951860193436020 \\
2581422178700136665773307761103400732984620374234862127757463287428154255421066900496076156 \\
814249048636868429413929618152758570201, \]
we have:

\[ \mathcal{F}(k) = (1366, 706, \]

and

\[ 356906506146909029994997307277425651867017252215411829599126246653681335656823597383176049 \\
0220400018179891259308980615096572344685923503907036491645183058824902979619742581500826036 \\
7555943462276039066156732684142686746350223597670138395541265022167797786255835931235764167 \\
76798226) \]

4 Enhanced key generation algorithm (EKgen)

In this section, we consider the following:

- Assume we have an operating system’s randomness Application Programming Interface (API) CSPRNG we call \( \_\text{CSPRNG} \), containing the following functions:
  \( \_\text{CSPRNG}\_\text{sample} \) and \( \_\text{CSPRNG}\_\text{choice} \) respectively the function that returns a random sample of size \( l \) from a set or list given as argument, and the function that returns a random choice of an element from a set or list given as argument of the function.

- Assume we have functions: \( \_\text{brightness\_level} \), \( \_\text{microphone\_noise\_level} \) with their values.

We consider the function \( \text{Decimal} \) that returns the decimal part of a float number and \( \log \) as the decimal logarithm.
Algorithm 2: EKgen Key Generation.

**Input:** $n := $ size or number of bytes to encrypt

**Output:** $Key := $ Cryptographically secure unique key of size $n$

1 **Initialization:**
   - $e_1 := _brightness_level$, $e_2 := _microphone_noise_level$,
   - $E := \{\text{Decimal}(\log(e_1))\} + \{\text{Decimal}(\log(e_2))\}$, $F := G := \text{Empty\_list}$,
   - $FP\_Database := \{\} ;$
2 **foreach** digit $\in E$ **do**
   3 **add** digit in $F$
   4 $G \leftarrow _{CSPRNG\_sample}(F, \text{card}(F))$
5 **end**
6 $key \leftarrow _{CSPRNG}(n)$
7 **if** $n > \text{card}(G)$ **then**
   8 **foreach** digit $\in G$ **do**
   9 **replace** _CSPRNG\_choice(key) in key by
   10 _CSPRNG\_choice(G)
11 **end**
12 $Key \leftarrow _{CSPRNG\_choice}(\{-key, key\})$
13 **end**
14 **else**
15 **foreach** digit $\in key$ **do**
16 **replace** _CSPRNG\_choice(G) in G by _CSPRNG\_choice(key)
17 **end**
18 $Key \leftarrow _{CSPRNG\_choice}(\{-_{CSPRNG\_sample}(G, n), _{CSPRNG\_sample}(G, n)\})$
19 **end**
20 **if** $FP(\text{Key}) \in FP\_Database$ **then**
21 **go to** 1
22 **end**
23 **else**
24 $FP\_Database \leftarrow FP\_Database \cup \{FP(\text{Key})\}$
25 **return** $Key$
26 **end**

**Example 4.1** $n = 16\text{bytes} (= 128\text{bits} \approx 39\text{digits})$, in an environment (mobile phone or computer) with brightness percentage $_brightness\_level = 95.5$,
microphone noise level $_microphone\_noise\_level = 19.1253$ and Footprint database $FP\_Database = \{\}$.
$E = \{\text{Decimal}(\log(95.5))\} + \{\text{Decimal}(\log(19.1253))\} = \{55912624748668\} + \{95101206586408\}$
$\implies E = \{5591262474866895101206586408\}$, $(F = \{5\}, G = \{5\}),(F = \{5,5\}, G = \{5,5\}),(F = \{5,5,9\}, G = \{5,9,5\}) \cdots$
\[ F = \{5, 5, 9, 1, 2, 6, 2, 4, 7, 4, 8, 6, 6, 8, 9, 5, 1, 0, 1, 2, 0, 6, 5, 8, 6, 4, 0, 8\}, \]
\[ G = \{4, 0, 5, 9, 1, 2, 8, 5, 8, 9, 6, 0, 2, 6, 5, 8, 6, 1, 7, 4, 6, 6, 8, 1, 4, 5, 2, 0\}\] 
key := \_CSPRNG\(16\) = 25361341064490836399663329023200789681. \( n = 39 > \text{card}(G) = 28 \) then from the random choice, we get the secure \( Key = 653663810194984356349663429523206785671 \) We verify \( Footprint \_FP(Key) = (180, 81, 442478115401867523414564167477658) \), since \( FP(Key) \) not in \( _FP\text{-Database} \), then the \( Footprint \) is saved and the \( Key \) used for encryption.

**Performances**

The following results have been collected while running in a computer environment with *Intel(R) Core(TM)* i3 – 6006U CPU @ 2.00GHz 2.00GHz technical details.

| Key size (Bytes) | Key Generation + verification time | Encryption |
|-----------------|-----------------------------------|------------|
| 8               | 0.0632 s                          | 0.0        |
| 64              | 0.0862 s                          | 0.00200 s  |
| 128             | 0.2446 s                          | 0.00454 s  |
| 256             | 0.6858 s                          | 0.0331 s   |
| 512             | 2.4876 s                          | 0.2335 s   |
| 1024            | 11.70158 s                        | 1.8048 s   |
| 2048            | 65.0299 s                         | 14.1251    |
| 4096            | 369.3401 s                        |            |
| 8192            | 2108.0893 s                       |            |

From this table we see that the encryption key of a text message of 19.728 characters (65536 bits) is generated and verified in 2108.0893 seconds, whereas the one of 155 characters (512 bits) is generated in 0.0862 seconds.

For optimization purpose, as solution we propose to encrypt data of size longer than 256 bytes (2048bits) with multiple keys of size at most 256 bytes after partitioning data to be encrypted in segments of at most 256 bytes. This significantly reduces key generation and verification time by at most \( k \times 0.0862 \) where \( k \) is the number of 2048 bits segments in the same environment as above.

**Remark 4.2** Considering on average a text message of size \( n \) characters, and assume one sends a message every one second. since each sent message represents a new generated key, then we have \( 1 \times 60s \times 60min \times 24h \times 365d = 31536000 \) keys/year. 
\[ n \approx \log_{10} \frac{\text{number of bits}}{10^{20-1}} \] then the probability of a collision is reached after a minimum of \( \frac{10^{n-1}}{31536000} \) years.

For example, while sending text messages of 20 characters at the rate of one every one second then we shall expect collisions of keys after \( \frac{10^{20-1}}{31536000} \approx \)
317097919837 years, We clearly see that this could happen many billions of milleniums after, therefore keys availability is guaranteed.

5 Complete Implementation with a negative key

The implementation we give here is done considering $\mathbb{P} = \mathbb{C} = \mathbb{K} = \mathbb{Z}/n\mathbb{Z}$ with $n = \text{Card}(\text{table}1)$ as an example for text messaging. We define:

- $\text{table}1$: the table that maps any alphanumerical character to a positive integer.
- $\text{table}2$: the table that maps any integer up to the $\text{table}1$’s length to its corresponding character in $\text{table}1$. Note that we could also use the ASCII code correspondances instead.

Now assume we have generated a Key $k \in \mathbb{Z}/n\mathbb{Z}$ (here we do not need this structure to be a field, nor a ring since we are just dealing with the additive law, therefore a group) from our $\text{EK}gen$ algorithm then we define the Encryption and Decryption algorithms as follows:

**Algorithm 3: Encryption.**

**Input:** message $= m_0...m_l \in \mathbb{P}$, key $= k_0...k_l \in \mathbb{K}$ from $\text{EK}gen$

**Output:** cipher $\in \mathbb{C}$

1. **Initialization:** $l := \text{message length}$, cipher := ””;
2. for $i \leftarrow 0$ to $l$ do
3.    if $key < 0$ then
4.        number $\leftarrow \text{table}1[m_i] - \text{table}1[k_{i+1}] \mod (\text{table}1 \text{ length})$
5.    end
6.    else
7.        number $\leftarrow \text{table}1[m_i] + \text{table}1[k_i] \mod (\text{table}1 \text{ length})$
8.    end
9.    letter $\leftarrow \text{table}2[number]$
10.   cipher $\leftarrow$ cipher + letter
11. end
12. return cipher

Note that the message length will always be the same as the key length since the key is generated by $\text{EK}gen$ accordingly to the message length (bytes, ...).
Algorithm 4: Decryption.

Input: $cipher = m'_0...m'_l \in \mathcal{C}$, $key = k_0...k_l \in \mathcal{K}$ from $EKgen$
Output: $message \in \mathbb{P}$

1. **Initialization**: $l := cipher$ length, $message := ""$;
2. for $i \leftarrow 0$ to $l$ do
3.     if $key < 0$ then
4.         $number \leftarrow table1[m'_i] + table1[k_{i+1}] \mod (table1$ length)
5.     end
6.     else
7.         $number \leftarrow table1[m'_i] - table1[k_i] \mod (table1$ length)
8.     end
9.     $letter \leftarrow table2[number]$  
10.    $message \leftarrow message + letter$
11. end
12. return $message$

Conclusion

In this paper, we have contributed at the key management level in a local environment with the introduction of the concept of key footprint ($FP(key)$) that keeps trace of a generated key without storing it, reducing risks and thus providing a secure and efficient verification way to ensure uniqueness of a generated key, and also at the key generation level by adding some further local system entropy to the CSPRNG system Application Programming Interface (API) on a personal device environment. We have given encryption and decryption algorithms taking into account positive and negative keys as an example for text messaging.

As future work, we shall study the complexity of recovering the key from the knowledge of its Footprint, add some entropy at the key generation level and we could exploit the idea of sending every encrypted data or message with the encrypted key as header encrypted with the suitable key-exchange protocol for this implementation.

References

[1] S.G. Srikantaswamy, H.D. Phaneendra, An Enhanced Practical Difficulty of One-Time Pad Algorithm Resolving the Key Management and Distribution Problem, *Proceedings of the International MultiConference*
of Engineers and Computer Scientists, 2018 Vol I, IMECS 2018, March 14-16, 2018, Hong Kong.

[2] S.G. Srikantaswamy, H.D. Phaneendra, Enhanced OneTime Pad Cipher with MoreArithmetic and Logical Operations with Flexible Key Generation Algorithm, International Journal of Network Security & Its Applications (IJNSA), 3 (6) (2011), 243-248. https://doi.org/10.5121/ijnsa.2011.3618

[3] Sunita, Ritu Malik, Hyper Encryption as an Advancement of one Time Pad an Unbreakable Cryptosystem, International Journal of Advanced Research in Computer Science and Software Engineering, 4 (1) (2014).

[4] Omotunde Ayokunle A., Faith Adekogbe, Onuiri Ernest, Precious Uchendu, An Implementation of a One-Time Pad Encryption Algorithm for Data Security in Cloud Computing Environment, Research Journal of Mathematics and Computer Science, (2017). https://doi.org/10.28933/rjmcs-2017-10-1402

[5] Devipriya M., Sasikala G., A New Technique for One Time Pad Security Scheme with Complement Method, International Journal of Advanced Research in Computer Science and Software Engineering, to appear.

[6] Raman Kumar, Roma Jindal, Abhinav Gupta, Sagar Bhalla, Harshit Arora, A Secure Authentication System - Using Enhanced One Time Pad Technique, IJCSNS International Journal of Computer Science and Network Security, 11 (2) (2011).

[7] Makato Matsumoto, Takuji Nishimura, Mersenne Twister: A 623-dimentionally equidistributed uniform pseudorandom number generator, ACM Transactions on Modeling and Computer Simulation, 8 (1) (1998) 3-30. https://doi.org/10.1145/272991.272995

[8] Richard J. Hughes, D.M. Alde, P. Dyer, G.G. Luther, G.L. Morgan, M. Schauer, Quantum Cryptography, University of California, Physics Division Los Alamos National Laboratory. https://arxiv.org/pdf/quant-ph/9504002.pdf

[9] Mort Yao, One-Time Pad, https://wiki.soimort.org/crypto/one-time-pad/

[10] Mads Haar, Introduction to Randomness and Random Numbers https://www.random.org/randomness/

Received: August 29, 2019; Published: March 22, 2020