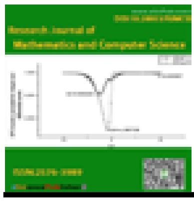
Research Article **RJMCS (2019) 3:17**



Research Journal of Mathematics and Computer Science (ISSN:2576-3989)

A New approach in One Time Pad key management

Gilda R. Bansimba, Regis F. Babindamana, Peter A. G. Kidoudou

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

ABSTRACT

Let $(\mathbb{P},\mathcal{C},\mathcal{K},\varepsilon_k,\mathcal{D}_k)$ be the One Time Pad cryptosystem. We consider $\mathbb{P}=\mathcal{C}=\mathcal{K}$.

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.

***Correspondence to Author:**
Regis F. Babindamana
Université Marien Ngouabi, Faculté des Sciences et Techniques, BP: 69, Brazzaville, Congo

How to cite this article:
Gilda R. Bansimba, Regis F. Babindamana, Peter A. G. A New approach in One Time Pad key management. Research Journal of Mathematics and Computer Science, 2019; 3:17 .

eSciPub LLC, Houston, TX USA.
Website: https://escipub.com/

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 0.2, proposition 0.3, proposition 0.4 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 appareance, 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 0.6 and remark 0.7)

• 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 0 and 1)

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

<u>Notations:</u>

$\cup$: the union

$++$: the concatenation

$\mathbb{P}$: the plaintext space

$\mathcal{C}$: the ciphertext space

$\mathcal{K}$: the key space

ε_k : the encryption method (algorithm) using the key k

$\mathcal{D}_k$: the decryption method (algorithm) using the key k

$PRNG$: Pseudo Random Number Generator

$CSPRNG$: Cryptographically Secure Pseudo Random Number Generator

$\mathbb{Z}$: ring of integers ($\{\cdots,-3,-2,-1,0,1,2,3,\cdots\}$)

$(\mathbb{F}_2)^l$: the field of binary representation base of length l

One Time Pad

Definition 0.1 Consider the keyword K with m characters and $k=(k_1,k_2,\ldots,k_m)$ its corresponding numeric vector.

Given a message $\mathcal{M}=x_1x_2x_3\ldots x_m$ of the same length as the keyword, to encrypt. Let $\mathbb{P}=$

$\mathcal{C} = \mathcal{K}$, then we define the encryption and decryption methods as follow:

$e_k(x_1, x_2, \dots, x_m) = (x_1 + k_1, x_2 + k_2, \dots, x_m + k_m)$: encryption

$d_k(y_1, y_2, \dots, y_m) = (y_1 - k_1, y_2 - k_2, \dots, y_m - k_m)$: decryption

and we easily verify the bijection that $d_k(e_k(x)) = x$, since $d_k(e_k(x)) = d_k(x_1 + k_1, x_2 + k_2, \dots, x_m + k_m) = (x_1 + k_1 - k_1, x_2 + k_2 - k_2, \dots, x_m + k_m - k_m)$

$$= (x_1, x_2, \dots, x_m) = x.$$

At the $binary$ level, we operate in the field $(\mathbb{F}_2)^l \cong (\mathbb{Z}/2\mathbb{Z})^l = \{0,1\}^l$ where l represents the length. We set $\mathbb{P} = \mathcal{C} = \mathcal{K} = (\mathbb{F}_2)^l$ and define $\oplus$ and $\ominus$ respectively the addition and substraction in $(\mathbb{F}_2)^l$.

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 $m = (m_1, m_2, \cdots, m_n)$. Let K be the key of same number of bits as m. We separate K in units of $l\ bits$ each, $K = (k_1, k_2, \cdots, k_n)$.

Hereby we operate the XOR operation in $(\mathbb{F}_2)^l$ on each unit.

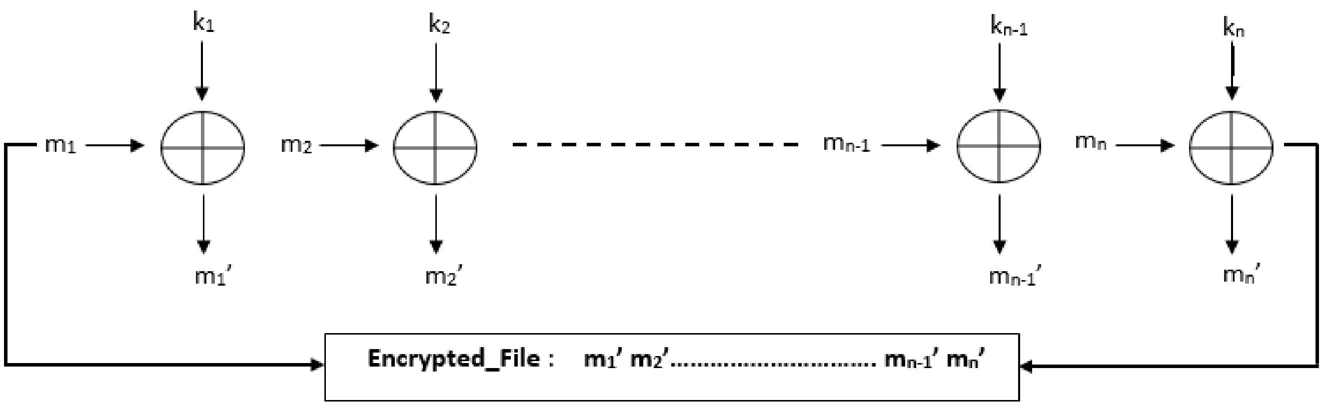
Encryption:

$One\_Time\_Pad(m,k) = (m_1 \oplus k_1, m_2 \oplus k_2, \cdots, m_n \oplus k_n) = (m_1', m_2', \cdots, m_n') = m'$

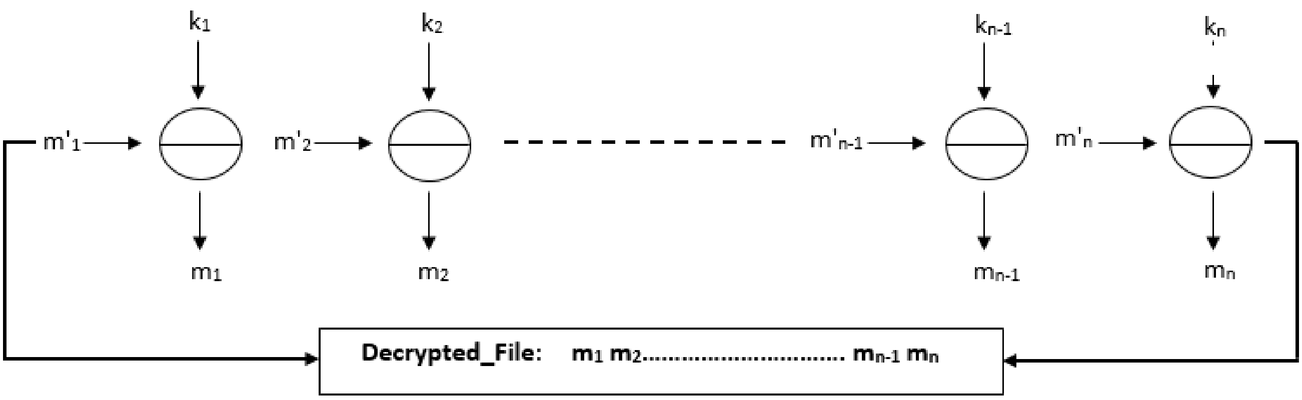
Decryption:

$One\_Time\_Pad(m',k) = (m_1' \ominus k_1, m_2' \ominus k_1, \cdots, m_n' \ominus k_1) = (m_1, m_2, \cdots, m_n) = m$

ENCRYPTION



DECRYPTION



Key FootPrint

Definition 0.2 Given a number $k = x_1 x_2 \cdots x_n$ of $n\ digits$ in decimal base, $n \geq 2$.

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

Proposition 0.3

1. $_{\mathcal{FP}}(-k) = -_{\mathcal{FP}}(k)$

Proof

(1) $_{\mathcal{FP}}(-k) = (-\sum_{i=1}^{n} x_i, -\sum_{i\ even}^{n} x_i, -\sum_{i=1}^{n} {x_i}^i)$ since $-k = -x_1x_2\cdots x_n$.

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

$_{\mathcal{FP}}(-k) = (-\alpha_1, -\alpha_2, -\alpha_3) = -(\alpha_1, \alpha_2, \alpha_3)$. Therefore $_{\mathcal{FP}}(-k) = -_{\mathcal{FP}}(k)$

Proposition 0.4 For every triplet $(\alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}^3$, if there exits a number k such that $_{\mathcal{FP}}(k) = (\alpha_1, \alpha_2, \alpha_3)$, 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 $_{\mathcal{FP}}(k_1) = _{\mathcal{FP}}(k_2) = (\alpha_1, \alpha_2, \alpha_3)$.

If $n \neq m$:

there's a contradiction from the definition 0.2, since k_1 and k_2 don't have the same number of digits, $\alpha_{1_{k_1}}$ could be equal to $\alpha_{1_{k_2}}$, as in $k_1 = 11243$, $k_2 = 218$ but not the even position numbers, ie $\alpha_{2_{k_1}} \neq \alpha_{2_{k_2}}$ then this implies $_{\mathcal{FP}}(k_1) \neq _{\mathcal{FP}}(k_2)$.

If $n = m$:

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

Since $_{\mathcal{FP}}(k_1) = _{\mathcal{FP}}(k_2) \Longrightarrow (\sum_{i=1}^{n} x_i,\ \sum_{i\ even}^{n} x_i,\ \sum_{i=1}^{n} {x_i}^i) = (\sum_{i=1}^{m} y_i,\ \sum_{i\ even}^{m} y_i,\ \sum_{i=1}^{m} {y_i}^i)$

$$\Longrightarrow \begin{cases} \sum_{i=1}^{n} x_i = \sum_{i=1}^{m} y_i \ (1) \\ \sum_{i\ even}^{n} x_i = \sum_{i\ even}^{m} y_i \ (2) \\ \sum_{i=1}^{n} {x_i}^i = \sum_{i=1}^{m} {y_i}^i \ (3) \end{cases} \Longleftrightarrow$$

$$\begin{cases} x_1 + x_2 + \cdots + x_n = y_1 + y_2 + \cdots + y_m (1) \\ x_2 + x_4 + \cdots + x_{2i+2\leq n} = y_2 + y_4 + \cdots + y_{2i+2\leq m} \ (2) \\ x_1 + x_2^2 + \cdots + x_n^n = y_1 + y_2^2 + \cdots + y_m^m \ (3) \end{cases}$$

(1) could hold like in $k_1 = 2732301$ and $k_2 = 6253110$ as $\alpha_{1_{k_1}} = \alpha_{1_{k_2}} = 18$ but not (2) nor (3). Also (2) could hold like in $k_1 = 2017569$ and $k_2 = 2413579$ as $\alpha_{2_{k_1}} = \alpha_{2_{k_2}} = 17$ but not (1) and (3). More generally, $(1), (2), (3)$ yield a Contradiction, since $k_1 \neq k_2$ ie $x_1x_2\cdots x_n \neq y_1y_2\cdots y_m$, ($\exists\ i \in [1, n]$ 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:

$$\Longrightarrow \begin{cases} x_1 + x_2 + \cdots + x_n = \alpha_1 (1) \\ x_2 + x_4 + \cdots + x_{2i+2\leq n} = \alpha_2 \ (2) \\ x_1 + x_2^2 + \cdots + x_n^n = \alpha_3 \ (3) \end{cases} \Longleftrightarrow$$

$$\begin{cases} x_1 + x_3 + \cdots + x_{2i+1} = \alpha_1 - \alpha_2 (4) \\ x_1 + x_2^2 + \cdots + x_n^n = \alpha_3 \ (3) \end{cases} (\star)$$

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

Algorithm 1: Footprint Computation

Input: $key = x_1x_2 \ldots x_n \in \mathcal{K}$

Output: $(\alpha_1, \alpha_2, \alpha_3) \in \mathbb{Z}^3$ such that $\alpha_1 := \sum_{i=1}^{n} x_i$, $\alpha_2 := \sum_{i even}^{n} x_i$, $\alpha_3 := \sum_{i=1}^{n} x_i^i$

```
1. Initialization: α1 = α2 = α3 = j = 0;
2. For x_i in key do
3.     j ← j + 1
4.     α1 ← α1 + x_i
5.     if j is even then
6.         α2 ← α2 + x_i
7.     end
8.     α3 ← α3 + x_i^j
9. end
10. if key<0 then
11.     return (−α1, −α2, −α3)
12. end
13. return (α1, α2, α3)
```

Example 0.5 *Given the* 1024 bit *key*

k=
1343953790552502795798195948217077919135014389676600354625097596284902011018597111889978
6695613868137702976510996987804764262464923407044582500862405722874006568502951860193436020
2581422178700136665773307761103400703298462037423486127757463287428154255421066900496076156
8142490486386684294139296181527585702 01,

we have:

$_{\mathcal{FP}}(k)$=(1366,706,
3569665061469090299949973072774256518670172522154118295991262466536881335656823597383176049
0220400481798912593098061509657234468592350390970364916451830588249029796197742581500826036
7555943462278039066156732684142686746350223597670138395541265022167797786255835931235764167
76798226)

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 $\_\mathrm{CSPRNG}$, containing the following functions: $\_\mathrm{CSPRNG\_sample}$ and $\_\mathrm{CSPRNG\_choice}$ 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: *_brightness_level*, *_microphone_noise_level* with their values.

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

Example 0.6 $n = 16bytes$ $(= 128bits \simeq 39digits)$, 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 $_{\mathcal{FP}}\_Database = \{\}$. $E = \{Decimal(log(95.5))\} + \{Decimal(log(19.1253))\} = \{55912624748668\} + \{95101206586408\}$

$\Longrightarrow$ $E = \{5591262474866895101206586408\}$, $(F = \{5\}, G = \{5\}), (F = \{5,5\}, G\{5,5\})$, $(F = \{5,5,9\}, G = \{5,9,5\}) \cdots (F = \{5,5,9,1,2,6,2,4,7,4,8,6,6,8,9,5,1,0,1,2,0,6,5,8,6,4,0,8\}, G = \{4,0,5,9,1,2,8,5,8,9,6,0,2,6,5,8,6,1,7,4,6,6,8,1,4,5,2,0\})$, key:= $\text{CSPRNG}(16) = 253613410644908366399663329023200789681$. $n = 39 > card(G) = 28$ then from a random choice, we get the secure $Key = 653663810194984356349663429523206785671$,

We verify $Footprint\ _{\mathcal{FP}}(Key) = (180,81,44247811540186752348141564167477658)$, since $_{\mathcal{FP}}(Key)$ not in $_{\mathcal{FP}}\_Database$, then the $Fooootprint$ is saved and the Key used for encryption.

Performances

The flollowing 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 s	
	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 0.7 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 $1 \times 60s \times 60min \times 24h \times 365d = 31536000\ keys/year$.

$n \approx \log_{10}^{2\ number\ of\ bits}$ 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.

Complete Implementation with a negative key

The implementation we give here is done considering $\mathbb{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}/n\mathbb{Z}$ with

$n = Card(table1)$ as an example for text messaging.We define:

$table1$: the table that maps any alphanumerical character to a positive integer.

$table2$: the table that maps any integer up to the $table1$'s length to its corresponding character in $table1$. 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 *EKgen* algorithm then we define the Encryption and Decryption algorithms as follows:

Algorithm 1: Encryption

Input: $message = m_0 \ldots m_l \in \mathbb{P},\ key = k_0 \ldots k_l \in \mathcal{K}$ from *EKgen*

Output: $cipher \in \mathcal{C}$

Initialization: $l := message\ \ length,\ cipher := ""$

For $i \leftarrow 0\ to\ \ l\ \boldsymbol{do}$

  if key < 0 **then**

    $number \leftarrow table1[m_i] - table1[k_{i+1}]\ \ mod\ \ (table1\ \ length)$

  end

  else

    $number \leftarrow table1[m_i] + table1[k_i]\ \ mod\ \ (table1\ \ length)$

  end

  $letter \leftarrow table2[number]$

  $cipher \leftarrow cipher + +letter$

end

return $cipher$

Note that the message length will always be the same as the key length since the key is generated by *EKgen* accordingly to the message length (bytes, ...).

Algorithm 1: Encryption

Input: $cipher = m'_0 \ldots m'_l \in \mathcal{C},\ key = k_0 \ldots k_l \in \mathcal{K}$ from *EKgen*

Output: $message \in \mathbb{P}$

Initialization: $l := cipher\ \ length,\ message := ""$;

For $i \leftarrow 0\ to\ \ l\ \boldsymbol{do}$

  if key < 0 **then**

    $number \leftarrow table1[m'_i] + table1[k_{i+1}]\ \ mod\ \ (table1\ \ length)$

  end

  else

    $number \leftarrow table1[m'_i] - table1[k_i]\ \ mod\ \ (table1\ \ length)$

  end

  $letter \leftarrow table2[number]$

  $message \leftarrow message + +letter$

end

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 $footprint$ ($_{\mathcal{FP}}(key)$) 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 prorocol for this implementation.

References

1. S.G.Srikantaswamy, Dr.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, Dr.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), Vol.3, No.6, November 2011.
3. Ms Sunita, Ms 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, Volume 4, Issue 1, January 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.
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.
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, VOL.11 No.2, February 2011.
7. Makato Matsumoto, Takuji Nishimura; *Mersenne Twister: A 623-dimentionally equidistributed uniform pseudorandom number generator*.
8. Richard J. Hughes, D.M. Alde, P. Dyer, G.G. Luther, G.L. Morgan, M. Schauer ; *Quantum Cryptography*. University of California, Physiscs 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. Dr Mads Haar; *Introduction to Randomness and Random Numbers* https://www.random.org/randomness/