E²CT: Energy Efficient Cipher Technique

Harshit Bhatia [1], Rahul Johari [2], Kalpana Gupta [3]

[1] REVAL India Private Limited, Gurugram, India
[2]SWINGER (Security, Wireless IoT Network Group of Engineering and Research) Lab, USICT, GGSIP University, Sector-16C, Dwarka, Delhi, India
[3]C-DAC, NOIDA, India
droid.harshit@gmail.com
rahuljohari@hotmail.com
kalpana7gupta@gmail.com

Abstract. The conventional techniques for symmetric and asymmetric cryptography are not optimized for usage on handheld devices in their raw form. They do not focus on optimized usage of battery over mobile devices and hence drain significant battery when deployed over the wireless handheld devices. Furthermore, they make use of a limited domain of keys and a limited number of mathematical operations. The major portion of the existing traditional symmetric cipher techniques is covered by those that rely on a single key-function for generation of keys that are used to garble the plain-text to unintelligible text before sending it over an unsecure network. The increase in the number of encoding operations and keys add significantly to the strength of a cryptographic technique. This paper presents a power optimized symmetric key technique that aims to reduce battery footprint without compromising on security by using multiple keys coupled with multiple encryption operations.

Keywords: Green, Symmetric, Cryptography, Encryption, Decryption, Energy Efficient.

1 Introduction

There exist plenty of cryptographic techniques that provide the security of the sensitive data. [1, 2]. However, such traditional techniques were not aimed at catering to the handheld devices with limited battery and resources. The techniques are not power optimized and consume enormous amount of battery thus making them an unsuitable choice for deployment in the mobile devices over the wireless network. The demand for new and improved cryptographic techniques is high, especially for the intricate hand-held devices that transmit sensitive information over network. The newer techniques aimed for handheld devices need to be cheaper (in terms of battery consumption) and faster without making any compromises with the security to ensure data transmission at an overall lower energy cost. The technique proposed in this text is a power optimized cryptographic approach aimed for mobile hand-held devices to secure the data with minimal consumption of the energy and hardware resources. [1]

2 Proposed System

2.1 The cryptosystem

The proposed system introduces a lightweight cipher technique that aims at reduction of overall battery consumption over a mobile handled device without making heavy compromises on data security. The power optimized version of the predecessor, PentaPlicative Cipher Technique, also uses a set of five predefined keys as an input to a symmetric key cipher technique. However, unlike the PentaPlicative technique, the power optimized version makes use of much cheaper mathematical operations – XOR. In order to disguise the true length of plaintext from the sniffer, the power optimized technique also makes use of the bit-dispersion technique. The technique has the strong grounds because of the multiple number of keys with multiple mathematical operations, which makes it difficult to decipher the plain text and hence effectively decreasing the overall probability of the cipher text to get decrypted by anyone other than the intended recipient.

2.2 Related Work

In [3] authors(s) present a new technique, the Cross-Language Cipher (CLCT) Technique, which is aimed at securing the plaintext data by character mapping. In [4] author(s) presented a new tool built in Java that demonstrates how the Dictionary attack and Brute Force attacks are used to break the authentication and highlights the Injection via SQL. In [5] authors presented a rudimentary cipher technique that is aimed at providing the security by employing three set of pre-defined keys in the process of encryption and decryption. In [6] author(s) presented the Pentaplicative Cipher Technique which makes use of five keys to encrypt the plaintext input by the user. In [7] author(s) had designed, implemented and evaluated a new Algorithm for Scheduling that makes use of a neural network predictor model to power off the unused servers in a Cloud computing environment and essentially reducing the consumption of power. In [8] author(s) propose various queuing algorithms to utilize the resources and task assignment is done in an efficient manner to ensure that the overall cost of operations is reduced while also making it a cleaner approach by decreasing the ill-effects of the data center on environment resulting in Green Eco System.

3 Methodology

The Energy Efficient extension of PentaPlicative cipher technique aims at being readily used in mobile handheld devices. This expects the technique to be stricter when it comes to resource consumption and in turn battery consumption. Considering this, Energy Efficient Cipher technique makes use of caching principle to store the keys to cut down the dependency of gathering keys from device's GPS and physical address for every operation performed. Furthermore, the technique is lightweight without making significant compromises with the security.

The GPS along with the mobile network data consumes a considerable amount of battery from device, hence limiting the pings made to them for grabbing the location helps in reduction of battery usage by a noticeable amount. Biggest culprits in faster battery discharge is Screen brightness and CPU usage [9]. The PentaPlicative Cipher technique caters to this problem of on-screen time by reduction of screen brightness during the process. The CPU and memory usage have been notably reduced by employing garbage collection to free up memory and killing any lingering daemon threads upon completion of encryption/decryption process, thereby making an observable reduction in the overall battery consumption by the handheld device.

3.1 Key Arrangement

The Energy Efficient Cipher technique makes use of a set of five private keys (kept secret) which are derived from the physical information of the handheld device. The set of keys is unique to every device; hence authentication of sender can easily be made. The IMEI number along with the location coordinates form the input key vector. The five keys (K_1, K_2, K_3, K_4 and K_5) are derived from the input domain as the following text describes. The unique International Mobile Equipment Identity (IMEI) number forms the first three keys (K_1, K_2, K_3) with each key of length five integer numbers and the other two keys (K_4, K_5) are given by first five integers (without decimal) derived from latitude and longitude coordinates respectively. The length of keys has been fixed to five to make computations faster and cheaper in terms of resource consumption. This allows for an optimized lightweight technique that would limit the actual battery discharge.

Additionally, using the physical information available on devices decreases the human intervention to bare minimum in the complete process, thus enabling the technique to be easily plugged in with other data storage or transmission applications that require data encryption.

The caching mechanism is also embedded in the key generation process. This is necessary because the key generation relies heavily on the physical information of the device and this would require the technique to make connection to the network for grabbing the location coordinates and reading of the hardware chipset for the IMEI number. This entire process wastes many precious CPU cycles as well as adds an additional burden by using the heavy battery sucking resources of the device. The technique makes sure to close the network connections if there are cached keys available to it and hence saving an appreciable number of unnecessary pings that would have been wasted otherwise.

3.2 Encryption operation

The paper introduces the technique with an example that draws by taking a sample plaintext and sample keys portraying the encryption and decryption on input domain. The technique, like its predecessor, also makes use of the ASCII character set. The input characters of the plaintext are first converted into the decimal numbers by mak-

ing use of the ASCII character to decimal conversion and then further mathematical operations are performed on them with the set of five input keys.

In order to mask the true length of the Plaintext (P_T), the technique makes use of an operation called the "Bit-Dispersion". This operation first converts the ASCII characters to their corresponding decimal values and then converts the decimal values to a Base2 Binary number. Each decimal number is thus converted to an 8-digit binary number. These 8-digit binary numbers are grouped together to form a long stream of binaries. The function further makes a group of 6 bits from this stream and then converts this -bit binary number to the corresponding ASCII character. If there are any remainder bits which are left after the grouping, are appending with padding of zeroes to make it a 6-bit binary number and this is converted to the corresponding ASCII character too. This new set of ASCII characters will be the final Cipher Text (C_T) which would be of a different length as the original Plaintext.

There are five other pre-defined mathematical operations other than the Bit-Dispersion that the technique uses. These mathematical operations when performed in a sequential manner would result in the Ciphertext (C_T) which can then be sent out by the sender to the receiver. These operations, denoted by E_1, E_2, E_3, E_4, E_5, are as follows:

$$E_1 = (P_T \text{ XOR } K_1) \tag{1}$$

$$E_2 = (E_1 + K_2) \bmod 256 \tag{2}$$

$$E_3 = (E_2 * K_3) \bmod 256 \tag{3}$$

$$E_4 = (E_3 - K_4) \bmod 256 \tag{4}$$

$$E_5 = (E_4 \text{ XOR } K_5) \tag{5}$$

$$C_T = \text{bit dispersion } (E_5) \tag{6}$$

The table 1and 2 depicts the process of encryption with the help of an example. The example clearly defines the input domain of plaintext and secret set of keys, followed by the set of encoding operations. Please note that for this example the keys that are selected are very small and simplistic numbers to ease the demonstration of mathematical operations, but in the practical world much larger keys would be used.

Plaintext (P_T) - CIPHER

Let the private keys be:

K1 = 17

K2 = 19

K3 = 17

K4 = 13

K5 = 15

Table 1. Encryption Table

P_T	E_1= (P.T. XOR K_1)	E_2= (E_1 + K_2) mod 256	E_3= (E_2 * K_3) mod 256	E_4= (E_3 – K_4) mod 256	E_5 = (E_4 XOR K_5)
C(67)	(67 XOR 17) = 82 (R)	(82 + 29) mod 256 = 111 (o)	(111 * 13) mod 256 = 163 (ú)	(163 - 57) mod 26 = 106 (j)	(106 XOR 19) = 121 (y)
I(73)	(73 XOR 17) = 88 (X)	(88 + 29) mod 256 = 117 (u)	(117 * 13) mod 256 = 241 (±)	(241 - 57) mod 256 = 184 (©)	(184 XOR 19) = 171 (½)
P(80)	(80 XOR 17) = 65 (A)	(65 + 29) mod 256 = 94 (^)	(94 * 13) mod 256 = 198 (ã)	(198 - 57) mod 256 = 141 (ì)	(141 XOR 19) = 158 (×)
H(72)	(72 XOR 17) = 89 (Y)	(89 + 29) mod 256 = 118 (v)	(118 * 13) mod 256 = 254 (■)	(254 - 57) mod 256 = 197 (┼)	(197 XOR 19) = 214 (Í)
E(69)	(69 XOR 17) = 84 (T)	(84 + 29) mod 256 = 113 (q)	(113 * 13) mod 256 = 189 (¢)	(189 - 57) mod 256 = 132 (ä)	(132 XOR 19) = 151 (ù)
R(82)	(82 XOR 17) = 67 (C)	(67 + 29) mod 256 = 96 (`)	(96 * 13) mod 256 = 224 (Ó)	(224 - 57) mod 256 = 167 (º)	(167 XOR 19) = 180 (┤)

Table 2. Bit dispersion Operation

Obtained E_5	121	171	158	214	151	180		
E_5 binary	01111 001	10101 011	10011 110	11010 110	10010 111	10110 100		
Cipher	01111 0	01101 0	10111 0	01111 0	11010 1	10100 1	01 0001	11 0100
Cipher Text	36	32	56	36	65	51	36	64

*Final transmitted Cipher text for '**CIPHER**' plaintext is* $ 8$A3$@

3.3 Decryption operation

The Ciphertext received at the receiver's end needs to be converted to the actual plain text message and this process is called the decryption. In order to decrypt the garbled cipher text message, the receiver also uses the same set of keys that sender used to encrypt the message. The first step is to change the length of the cipher text to match the original length of the plain text message. The Bit-dispersion operation that sender performed needs to be neutralized by the reverse bit-dispersion mechanism. This process now re-groups the 6-bit binary characters to the 8-bit binary characters and then converts the 8-bit binary to the corresponding ASCII Character. The extra padding bits in the form of zeroes, that were added during the Bit-Dispersion process are also

removed and the original length of plaintext is restored on received cipher text. This is followed by the set of pre-defined mathematical operations using the set of private keys to count the effects of encryption to finally obtain the original desired plaintext message. The mathematical steps are denoted as D_1, D_2, D_3, D_4 and D_c denoted the reverse bit-dispersion operation. The mathematical operations to compute the plaintext (P_T) on the receiver's end are depicted as follows:

$$D_c = \text{reverse bit dispersion } (C_T) \tag{7}$$

$$D_1 = (D_c \text{ XOR } K_5) \tag{8}$$

$$D_2 = (D_1 + K_4) \bmod 256 \tag{9}$$

$$D_3 = (D_1 * K_3^{-1}) \bmod 256 \tag{10}$$

$$D_4 = (D_3 - K_2) \bmod 256 \tag{11}$$

$$P_T = (D_4 \text{ XOR } K_1) \tag{12}$$

The tables 3 and 4 depict the usage of the mathematical operations in the decryption process by making use of an example. The example depicted here is an extension of the same example depicted in Tables 1 and 2. The decryption operation takes in the input the same set of five keys as private keys and also uses the output of encryption operation as the input of decryption operation as a cipher text.

The Cipher text (C_T) is: $ 8$A3$@

The modulo inverse of the Key K_3 is denoted as K_3^{-1} and is computed to be: 197 (satisfies $K_3 \ K_3^{-1} \equiv 1 \bmod 256$)

Table 3. Reverse Bit Dispersion Operation

Obtained C	36	32	56	36	65	51	36	64
C in binary	01111 0	01101 0	10111 0	01111 0	11010 1	10100 1	01 0001	11 0100
Re-Dispersed	01111 001	10101 011	10011 110	11010 110	10010 111	10110 100		
Dispersed ASCII	121	171	158	214	151	180		

Dispersed text to be used to obtain plaintext is y½×Íù┼

Table 4. Decryption Table

D_c	$D_1 = (C \text{ XOR } K_5)$	$D_2 = (D_1 + K_4) \bmod 256$	$D_3 = (D_2 * K_3^{-1}) \bmod 256$	$D_4 = (D_3 - K_2) \bmod 256$	$D_5 = (D_4 \text{ XOR } K_1)$
y (121)	(121 XOR 19) = 106 (j)	(106 + 57) mod 256 = 163 (ú)	(163 * 197) mod 256 = 111 (o)	(111 - 29) mod 256 = 82 (R)	(82 XOR 17) = 67 (C)
½ (171)	(171 XOR 19) = 184 (©)	(184 + 57) mod 256 = 241 (±)	(241 * 197) mod 256 = 117 (u)	(117 - 29) mod 256 = 88 (X)	(88 XOR 17) = 73 (I)
× (158)	(158 XOR 19) = 241 (ì)	(241 + 57) mod 256 = 198 (ã)	(198 * 197) mod 256 = 94 (^)	(94 - 29) mod 256 = 65 (A)	(65 XOR 17) = 80 (P)
Í (214)	(214 XOR 19) = 197 (┼)	(197 + 57) mod 256 = 254 (■)	(254 * 197) mod 256 = 118 (v)	(118 - 29) mod 256 = 89 (Y)	(89XOR 17) = 72 (H)
ù (151)	(151 XOR 19) = 132 (ä)	(132 + 57) mod 256 = 189 (¢)	(189 * 197) mod 256 = 113 (q)	(113 - 29) mod 256 = 84 (T)	(84 XOR 17) = 69 (E)
┤ (180)	(180 XOR 19) = 167 (º)	(167 + 57) mod 256 = 224 (Ó)	(224 * 197) mod 256 = 96 (`)	(197 - 29) mod 256 = 67 (C)	(67 XOR 17) = 82 (R)

*Final intended plain text message is '**CIPHER**'*

4 Mathematical Modelling

The mathematical operations cannot be applied directly to the plain text which is a string of characters. Before the encryption operations may be applied, the plain text character needs to be converted from the text format to the corresponding ASCII decimal number value. Upon this ASCII decimal encoding, the encryption operation can be summarized as a set of mathematical equations which when applied in the correct order result in the final cipher text which can then be transmitted to the receiver. Each individual encryption equation from the set of mathematical equations can be denoted by $E_n(x)$; and each equation when applied on the plain text, denoted by P(x), outputs the final Cipher text which is denoted by C(x) as follows:

$$C(x) = f_{dispersion}(E_5(x)) \tag{13}$$

$$\text{where, } E_5(x) = (E_4(x) \text{ XOR } K_5(x)), \tag{14}$$

$$\text{and, } E_4(x) = (E_3(x) - K_4(x)) \bmod 256, \tag{15}$$

$$\text{and, } E_3(x) = (E_2(x) * K_3(x)) \bmod 256, \tag{16}$$

$$\text{and, } E_2(x) = (E_1(x) + K_2(x)) \bmod 256, \tag{17}$$

$$\text{and, } E_2(x) = (P(x) \text{ XOR } K_1(x)) \tag{18}$$

The Bit-dispersion function in the above equations, is given by the function fdispersion(En(x)) and the private keys are given by the function Kn(x), where n ϵ [1, 5]. The length of the plain text is given as 'n' and that of cipher text is denoted by 'm'

where $n < m$, since the bit dispersion function eliminates the one-to-one character mapping and thus changes the length of final cipher text.

1. The conversion of the string of characters into their corresponding ASCII decimal numbers is the first operation which is performed. This encoding of plan text characters can be represented as a function P(x), where P(x) comprises of individual decimal values and each of these decimal value for 'n' number of characters of plain text can be represented as $P_1(x)\ P_2(x)\ P_3(x) \ldots P_n(x)$ and each character $P_i(x)$ represented in Base10 decimal value belongs to the range $0 \leq P_i(x) \leq 255$.
2. Each encryption operation is a linear mathematical operation which involves the five private keys and each encryption mathematical operation can be represented as the function E(x) comprised of a mathematical operation (represented as ϕ) and a key function K(x) and is represented as, $E_i(x) = E_{i-1}(x)\ \phi\ K_i(x)$
3. The Base10 decimal number that is obtained from the encryption functions $E_4(x)$ needs to be converted to a Base2 binary number. This Decimal to Binary conversion is carried out for each individual decimal number from the set of 'n' numbers and for a decimal number represented as x_i the binary number can be obtained as a set of following procedure – "Keep dividing the quotient by 2 until the quotient is 0 and the all the remainder represented in a reverse order is the binary number". This can be illustrated as a set of equations:

$$Q_0 = x_i / 2 \text{ remainder } R_0 \tag{19}$$

$$Q_1 = Q_0 / 2 \text{ remainder } R_1 \tag{20}$$

$$Q_j = Q_{j-1} /2 \text{ remainder } R_j,\ Q_j \in [1, 0] \tag{21}$$

$$Q_{j+1} = Q_j / 2 \text{ remainder } R_{j+1},\ R_{j+1} \in [1, 0] \tag{22}$$

The Base2 binary number representation of decimal integer x_i is $R_{j+1}R_j \ldots R_2 R_1$

4. The bit-dispersion function $f_{dispersion}$ groups all the 8-bit binary numbers together and then combine them into a 6-bit binary number which is then converted back to the Base10 decimal number. The binary numbers need to be converted back to the decimal numbers and this Base2 to Base10 conversion involves, "multiplying the sum total by 2 and adding the remainder bit to it" and is shown as follows for a binary number $R_{j+1}R_j \ldots R_2R_1$ the final Decimal number is D_{j+1}:

$$D_1 = 2 \text{ x } 0 + R_1 \tag{23}$$

$$D_2 = 2 \text{ x } D1 + R_2 \tag{24}$$

$$D_j = 2 \text{ x } Dj\text{-}1 + R_j \tag{25}$$

$$D_{j+1} = 2 \text{ x } D_j + R_{j+1} \tag{26}$$

5. Conclusively, these transformed decimal integers obtained as the result of encryption operation $E_5(x)$ is mapped to an ASCII character each and this reverse ASCII

mapping gives the final Cipher text C(x) which is then returned to the receiver as the message. This transmitted cipher text is of length 'm' which is greater than the length of plaintext 'n', i.e. $m > n$.

6. The average execution time is given by equation, $T = (\Delta T_0 + \Delta T_1 + \Delta T_2 + \Delta T_3 + \Delta T_4 + \Delta T_5 + \Delta T_6 + \Delta T_7 + \Delta T_8 + \Delta T_9 + \Delta T_{10}) / 11$.
7. The Time complexity can be computed and depicted in Big-Oh notation as 'O(n)' where 'n' is the length of the plaintext. The calculation of the Time taken for various process is specified in table 5.

Table 5. Time Calculation.

S. No.	Operations	Time taken
1.	$E_1(y)$	ΔT_0
2.	$E_2(y)$	ΔT_1
3.	$E_3(y)$	ΔT_2
4.	$E_4(y)$	ΔT_3
5.	$E_5(y)$	ΔT_4
6.	C(y)	ΔT_5
7.	ASCII convert	ΔT_6
8.	Base10 to Base2	ΔT_7
9.	Bit Dispersion	ΔT_8
10.	Base2 to Base10	ΔT_9
11.	Reverse ASCII	ΔT_{10}

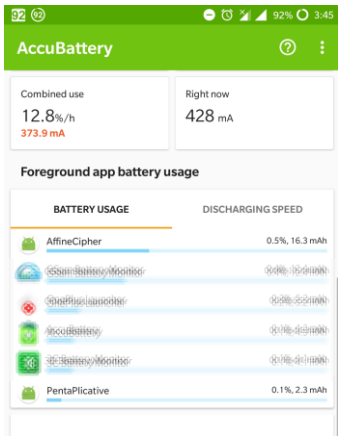

Fig. 1. Battery Usage of PentaPlicative vs Affine cipher

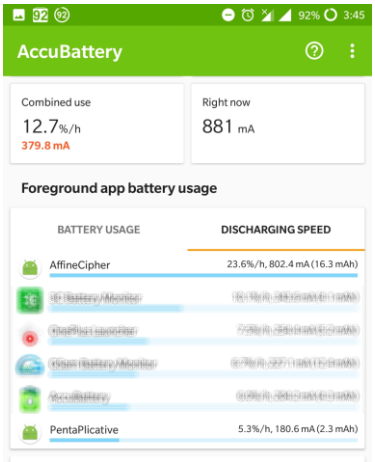

Fig. 1. Discharge Speed - PentaPlicative vs Affine cipher

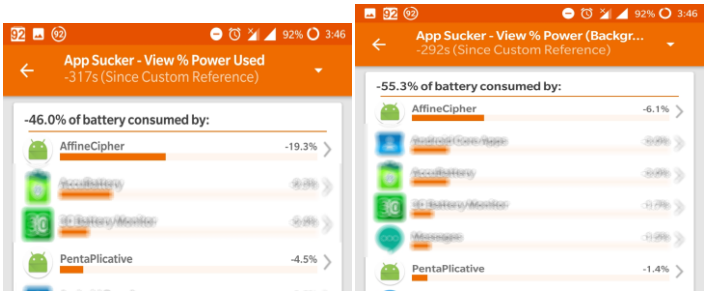

Fig. 2. Battery Usage Foreground and Background of PentaPlicative vs Affine Cipher

Fig. 3. CPU Usage of PentaPlicative vs Affine Cipher

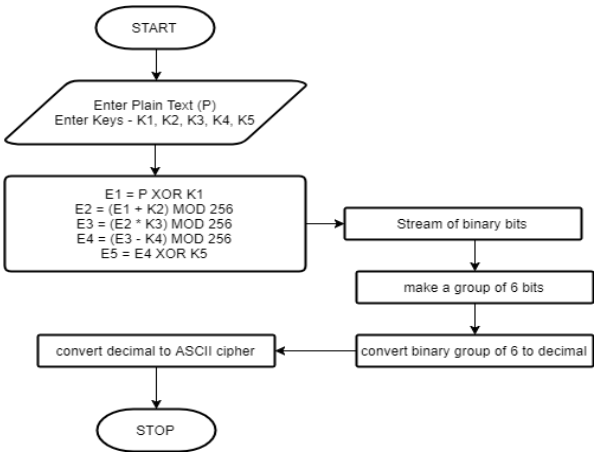

Fig. 5. Flowchart of Encryption in PentaPlicative Cipher

5 Results: PentaPlicative vs Affine Cipher Technique

A comparison is drawn between two symmetric key techniques, PentaPlicative Cipher technique and Affine Cipher technique. The PentaPlicative Cipher technique uses five keys as opposed to the two-key cryptosystem of affine cipher. The use of limited keys makes the affine cipher highly vulnerable to attacks that make use of a system of linear equations to decipher the text. However, the use of increased number of keys and operations coupled with the bit dispersion operation used in PentaPlicative cipher technique makes it impenetrable to such attacks. It would be expected that since affine cipher uses lesser keys and lesser operations, hence it would also result in a lower consumption of battery than PentaPlicative. However, this would have been true if no

battery discharge optimizations were not included for PentaPlicative technique. It makes use of cached keys thus reducing the hefty battery consumption as opposed to the standard implementation of the affine cipher with the same input vector of keys of IMEI number and location coordinates. The PentaPlicative also reduces screen brightness and prompts garbage collection upon exit, thus making a significant improvement in effectively curbing memory as well as battery utilization. Battery usage and discharge speed were measured for both PentaPlicative and Affine Cipher using AccBattery App [10]. It reveals that battery usage for Affine Cipher is relatively higher than PentaPlicative (Fig 1) and the former has a greater battery discharge speed over the period (Fig 2). The CPU Usage along with foreground and background battery usage has also been included for the comparative analysis of the two apps provided by GSamBattery [11] and TrepnProfiler [12]. It indicates a greater foreground task and higher usage of CPU in the background and as result a larger resource utilization for Affine Cipher (Fig 3 and 4). The obtained results have been briefed in the mentioned table. Please note that the simulation of the Energy Efficient Cipher Technique was carried out in a controlled environment (Table. 6). Both techniques were coded as android applications and the performance results were measured under the same conditions. The running time of Pentaplicative Cipher Technique is 1.9 milliseconds. [6]

Table 6. Simulation environment

Simulation Environment	
O.S. used	Android 8.0.0
Mobile Model	OnePlus 3T (A3003)
RAM	6 GB
Development IDE	Android Studio 2.3.1
Compile SDK	Version 25
Development Lang	Java, XML
Java Version	1.8.0 build 221-b11

6 Conlusion

The Energy Efficient Cipher technique is a Green Cryptographic technique that is robust and lightweight to cater to the handheld devices. It has been optimized to make minimal use of the hardware resources and to use light-weight operations to secure the data. This, in turn, reduces the battery consumption by the Energy Efficient Cipher Technique. The additional mechanisms implemented in the crypto system in form of caching greatly reduce the number of CPU cycles and thereby significantly reducing the battery consumption and making it a perfect fit for the hand-held devic-

es. Moreover, the reduction of human interaction in the process of the selection of private keys by automatically selecting the five private keys as the physical information of the device, contributes to the security and make it much more difficult for the keys to be deciphered. Furthermore, the obtained results clearly support the Energy Efficient Cipher Technique in terms of the battery drainage and CPU usage over a less secure traditional affine cipher technique. Conclusively, the result is a strong green cipher technique.

References

1. Behrouz A. Forouzan, “Cryptography and Network Security”, Tata McGraw-Hill Special Indian Edition 2007.
2. William Stallings. “Cryptography and Network Security-Principles and Practices” Pearson Education, fourth Edition 2007.
3. L. Singh, R. Johari. "CLCT: Cross Language Cipher Technique." In *International Symposium on Security in Computing and Communication*, pp. 217-227. Springer International Publishing, 2015.
4. I. Jain, R. Johari, R.L Ujjwal “CAVEAT: Credit Card Vulnerability Exhibition and Authentication Tool”. In: Second International Symposium on Security in Computing and Communications (SSCC‟14), pp 391- 399.Springer 2014.
5. R. Johari, H. Bhatia, S. Singh and M. Chauhan. “Triplicative Cipher Technique” *Procedia Computer Science* 78: 217-223, 2016.
6. Garg N., Bhatia H., Johari R. (2019) Pentaplicative Cipher Technique. In: Bhattacharyya S., Hassanien A., Gupta D., Khanna A., Pan I. (eds) International Conference on Innovative Computing and Communications. Lecture Notes in Networks and Systems, vol 55. Springer, Singapore.
7. Duy, Truong Vinh Truong, Yukinori Sato, and Yasushi Inoguchi. "Performance evaluation of a green scheduling algorithm for energy savings in cloud computing." In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1-8. IEEE, 2010.
8. Sanjeevi, P., and P. Viswanathan. "A green energy optimized scheduling algorithm for cloud data centers." In *Computing and Network Communications (CoCoNet), 2015 International Conference on*, pp. 941-945. IEEE, 2015.
9. Shye, Alex, Benjamin Scholbrock, and Gokhan Memik. "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures." In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pp. 168-178. IEEE, 2009.
10. AccuBattery Application downloaded from https://play.google.com/store/apps/details?id=com.digibites.accubattery&hl=en
11. GSamBattery Application downloaded from https://play.google.com/store/apps/details?id=com.gsamlabs.bbm&hl=en
12. TrepnProfiler Application downloaded from https://play.google.com/store/apps/details?id=com.quicinc.trepn&hl=en