SOCIETY FOR SCIENCE AND EDUCATION UNITED KINGDOM

TNC Transactions on Networks and Communications

VOLUME 3, ISSUE 2
ISSN: 2054 - 7420

An Enriched Ciphering Method to Evaluate Performance of EEA2-algorithm for LTE Security

Gautam Siwach[1], Amir Esmailpour[2], and Ahmad Sharifinejad[3]
[1,2]*Department of Electrical and Computer Engineering,*
University of New Haven, West Haven, CT, USA
[3]*IEEE member, Oslo, Norway*
gsiwach@gmail.com, aesmailpour@newhaven.edu, a_sharifinejad@yahoo.no)

ABSTRACT

In order to address vulnerabilities in LTE security and the objective to strengthen the implementation of encryption and decryption algorithms in LTE, we proposed a method for integration of an algorithm with a dynamic matrix generation scheme of 16*16 elements during AES implementation. Implementing such algorithm eradicates the vulnerability that causes a hacker to access plain text. The vulnerability arises within the implementation of AES.

This research explores Enriched Ciphering algorithm for number of scenarios based on function calls and time metrics that increases 3.9 times when there is a change in size of the plain text. We evaluated the proposed algorithm referred to as Enriched Ciphering Algorithm based on various use cases, including compression and decompression of cipher text that provides an extra layer of complexity to strengthen the security, and provides performance improvements of 13.9 percent when executing the Enriched Ciphering algorithm with big data size.

Keywords- AES, LTE, EEA1, EEA2, PDU.

1 Introduction

In this study we introduce a new algorithm to address some security vulnerabilities in the Long-Term Evolution (LTE) technology caused by compromised encryption/decryption process. Our new proposed algorithm so called “Enriched Ciphering” is optimized for performance evaluation. This approach offers a solution to potential vulnerability in original EEA1 and EEA2 algorithms. The vulnerability of these algorithms is due to having plain text in the remaining part of cipher text within one PDU, which is exposed to intruders.

Advanced Encryption Standard (AES) can be performed on blocks of 128 bits of data according to the National Institute of Standards and Technology (NIST) standards. Once the data is encrypted by AES, and a cipher key generated then XOR operator is applied on the plain text part of PDU in order to construct a Ciphered Text (CT). In order to apply the described procedure to our payload data, the Plain Text (PT) is partitioned into blocks of 128-bit data as shown in equation (1). The final block need not be 128 bits.

$$PT = PT\,[1] + PT\,[2] + \dots + PT\,[n] \qquad (1)$$

Encryption of payload happens by using AES Counter (AES-CTR), in order to strengthen the security. Each PT block is XOR’ed with a block of key stream to generate the (CT). The AES counter encryption of each Plain text block results in 128-bits of key stream because it is partitioned into blocks of 128-

DOI: 10.14738/tnc.32.1092
Publication Date: 25th April, 2015
URL: http://dx.doi.org/10.14738/tnc.32.1092

bit data. The most significant 64 bits of the counter block PT are initialized as seen before, followed by 64 bits that are all 0. The AES function performs AES encryption under the control of the confidentiality key. (Confidentiality Key refers to the private key). In order to make sure that even final block is partitioned to the appropriate length of 128-bit data, we use the TRUNC function; it truncates the last output of the AES encryption operation to the same length as the final PT block, returning the most significant bits.

The decryption operation is similar to the encryption, and the AES-CTR uses the same AES encryption operation since AES is a standard algorithm and it does not change. So of course the same AES applies both ways for both encryption and decryption, we are not making any modification to AES algorithm we are applying AES to LTE stream to make it more secure.

We execute the new edition EEA2 algorithm that includes a matrix as an enhancement to the ciphering process called enriched ciphering. The purpose of this study is to present a secure implementation for an existing vulnerability in the LTE encryption technique and to conduct a performance evaluation in order to provide a benchmark for further improvements.

2 Background

AES algorithm implementation has a potential vulnerability. In [10] “LTE Security Potential Vulnerability and Algorithm Enhancements”, we presented the vulnerability and provided the design as a solution for it through the enriched ciphering algorithm. The breakdown for explaining the vulnerability is let us assume that we have only acquired the entire cipher text by using brute force attack, and we don't have any information on either the plain text or the key. Evidently we need more information on the original plain text or the key to be able to go any further in deciphering the content. Following this let us assume that we now have captured the plain text and cipher text within a PDU, so we are missing the key to be able to recover the entire plain text. Finally assume that if we have retrieved 128 bits of plain text then we will have the corresponding cipher text of 128 bits, and as well as the remaining cipher text.

We have described the complete vulnerability and proposed the design of a solution to address it, we implemented the proposed solution including the main encryption mechanism key generation and distribution in Matlab simulation software. The main component of the solution are presented in the Enriched-ciphering algorithm, the AES algorithm could rebuild the key when we know plain text and corresponding cipher text. The input for AES is fixed within one PDU, so that the outputs from AES are the same length as input, which will be used to XOR with the plain text. The vulnerability was that once we get the output as the cipher text per PDU and complete cipher text, then we could retrieve the plain text by simply using XOR operation between the complete cipher text and the output. As per our assumption, we get 128 bits of plain text and the corresponding cipher text and the remaining of the cipher text. We XOR the plain text and cipher text; we get the output from AES. Upon using this output to XOR with the cipher text the remaining plain text could be regenerated.

3 Literature review

The need for enhancement in authentication services is motivated due to a fact that even Password Authentication Protocol (PAP) and Lightweight Extensible Authentication Protocol (LEAP) are vulnerable as investigated in [1] by Lianfen Huang, et.al. They experimented on authentication protocols in LTE Environments and summarize authentication service as the most important services in LTE networks in “Performance of Authentication Protocols in LTE Environments.”

In [2] Bin Liu et.al investigates the implementation issues in "Parallel AES encryption engines for many core processor arrays, by exploring different granularities of data-level and task level parallelism, based on the core systems largest with 107 to 137 cores and depicting several AES implementations on a fine-grained many-core system. They concluded that design on a fine-grained many core system software platforms achieves energy efficiencies approximately 2.9-18.1 times higher as compared to other software platforms. By this theory we expect that the designed solution of adding a matrix to the AES encryption can be evaluated for performance improvement.

There are several researches for LTE, AES, and also the integration of several other platforms like VLSI, Graphical user interface and many core cross platforms those served as a motivation for this research In [3] Naga M. Kosaraju et al illustrates the AES Algorithm architecture during their research for "A high- Performance VLSI Architecture for AES Algorithm." They presented architecture by expanding the secret key used to generate subkeys in VLSI implementation of AES algorithm. They implemented a prototype chip using 0.35 µ CMOS technology resulting in a throughput of 232Mbps for iterative architecture and 1.83Gbps for pipelining architecture. They explore the architecture for key scheduling Unit and are yet another instance to prove the integration and the combination logic.

In addition, AES takes a lot of time for encrypting which has been a major concern of all the researchers like [4] Fei Shao et al during "AES Encryption Algorithm based on High Performance Computing of GPU." They have proposed different approaches in order to expedite the encryption process. Simultaneously [5] Keisuke Iwai et al in "AES encryption implementation on CUDA GPU and its analysis" define granularity and memory allocation as major contributors to effective processing in AES encryption on GPU, thereby supporting the former content. Keeping this as context, we provide a real run time graphical illustration in the results section of this paper. We use different figures for illustrating the analysis of the performance and for future work.

There had been several investigations to ensure the privacy in LTE. One of them is by [6] Khodor Hamandi et al in "Privacy Enhanced and Computationally Efficient HSK-AKA LTE Scheme," in which they explain the HSK-AKA procedure and the key management system from HSS to UE by using a periodic temporary identifier. They gave an illustration of how the process flow occurs between HSS and UE and presented a modified LTE Authentication and Key Agreement (AKA) scheme,HSK-AKA.

As iterated by [7] Mehran Mozzaffari-Kermani et al in "Concurrent structure-Independent Fault Detection Schemes for the Advanced Encryption Standard" AES has been lately accepted as the symmetric cryptography standard for confidential data transmission. However, we still strive for reliable AES architecture. Their investigation concludes with the implementation of a structure independent scheme in order to reach the error coverage of approximate 100%.

The need for analyzing compression techniques along with encryption is an important consideration and is also stated in "Securing multimedia content using Joint compression and encryption" authored by [8] A. Pandae at al. They state that the algorithmic parameterization and hardware architectures are useful to ensure secure transmission of multimedia data in resource-constrained environments such as wireless video surveillance networks. The "Enriched Ciphering algorithm" is executed along with compression and is analyzed in the context of time complexity as well.

As explored by [9] Krzysztof Jankowski et.al in "Packed AES-GCM Algorithm suitable for AES/PCLMULQDQ instructions" about the performance evaluation of authenticated Encryption Algorithms that AES block cipher working in Counter Mode (CTR), characterized with key sizes 128, 192, and 256 is used to XOR the Encrypted Counter with the plain text. They craft a gateway to further investigate the approach for parallel encryption and increment. Moreover, we support their investigation and plan to advance this research for parallel encryption.

4 The enriched ciphering method design and implementation

The design of the Enriched Ciphering algorithm is to ensure security during the process. In [15] we also presented a security structure of a ciphering algorithm that will build a matrix on the concept of generating random numbers based on permutation; the matrix will be incorporated in the encryption process by including it in the final stage of encryption algorithm in order to get the enriched cipher text. A matrix block of size 16*16 is used. Within each block of 256 elements, there are 8 binary bits, which is equal to 2 hex digits. The 256 elements are different and make the enriched cipher text. This process will enhance encryption by providing a more secure cipher text, which is not as vulnerable as the original encryption process, hence labeled: “Enriched cipher text”.

For this Enriched Ciphering algorithm, the block used depends on the cipher text stage1, and the cipher text stage1 is the plain text XOR’ed with the key stream that has dynamic elements generated by matrix. It requires a significant amount of data to break this Enriched Ciphering algorithm. In addition, the key stream will be changed per PDU. Therefore, this process will not disclose enough data for the intruder to be able to crack the Enriched Ciphering algorithm easily. However, for the same instance, considering 128 bits of plaintext and corresponding cipher text, the derivation of the key stream will be far easier. This indicates a classical trade-off scenario between complexity and space used, the higher the capacity the larger space is occupied by the data stream as data is divided per PDU and the key is generated for every 128 bit of plain text.

4.1 Key generation mechanism

In this section we describe the process of generating the key. This is to generate a random number of keys for 256 elements of the matrix block. We generate the reverse matrix based on the original matrix and a new empty matrix for each block of the original matrix and obtain the value. The first digit is the column number for reverse matrix, and the second digit is the row number for the reverse matrix. Once we get a block in reverse matrix, we fill it by a combination of the original column number and row number, by repeating this process for each block, the reverse matrix will completely be rebuilt.

$$\text{Key}=\text{key (randperm (numel(key)))} \tag{2}$$

4.2 Distribution of the key

The key is 128 times larger than the normal key that has the pain text of 128*128 bits size, so it is not easy to share the key with confidentiality. Therefore we just share it before using EEA2 that happens before NAS and AS signaling. UE generates the matrix and just encrypts it by AES and sends it to the MME before NAS and to the eNodeB before AS signaling. Since both MME and eNodeB can calculate the reverse matrix, the UE can use EEA2 between MME and eNodeB. We can choose the KASME for AES to MME, and for AES to eNodeB. We can choose the KeNB because these two keys were already shared before so they are used for mutual authentication.

4.3 Implementation

The implementation is based on the design and the key generation mechanism and the distribution of key concepts, combined in the Enriched Ciphering algorithm. The code uses the permutation concept to generate random numbers for elements of matrix block, it includes conditional statements and logic gates operations as well. Moreover, the matrix is able to provide more secure operations in terms of encryption complexity.

Matrix is dynamically generated and processes 128 bits round of plain text. In an example the plain text is passed as arguments through the matrix and is divided into 128 bits of text as one round that is transformed as per hexadecimal notion by Enriched Ciphering algorithm as step I.

The Enriched Ciphering algorithm provides a solution for possible vulnerability within the encryption process, which involves adding a matrix block to the encryption process of the cipher text results obtained upon executing the Enriched Ciphering algorithm in different scenarios. The encryption process uses a dynamic set of numbers in the form of a matrix, which is used with the cipher text to obtain an enriched block of ciphered data.

In step II the plain text is processed to cipher text. This cipher text is encrypted in step III by a matrix. 'OutCT' or Output cipher text of 128 bits shows the text after encryption by the matrix. backCT shows the decrypted text and the loop follows till the end and whole plain text is retrieved upon reducing it to its normal form. We Evaluate and analyze the procedure, design and algorithm. The performance evaluation is based on different performance evaluation metrics such as time, call, self-time and function call. 'Call' refers to calling the predefined function at a particular instance in the program. The self-time and time are the sub parts of the total time taken to execute the Enriched Ciphering algorithm. One of the sample test run of the Enriched Ciphering algorithm is depicted in this paper for performance evaluation. It is an instance run and is utilized to evaluate the performance of the Enriched Ciphering algorithm.

Enhancements occur upon adding compression and decompression functions to the existing Ciphering algorithm. These functions are included as a part of Enriched Ciphering algorithm in an effort to improve the performance even when the text is larger in size so as to make space for faster, efficient ways to build strength. The increase in the execution time of the Enriched Ciphering algorithm is associated with the number of calls for functions with increased size of text, including total time and self-time of processes.

Since the key elements of the matrix keep on changing and are dynamic in order to provide a unique security level, so the run time can't be generalized for all Enriched Ciphering algorithm executions. We provide the precise time of a particular run based on different scenarios in the cases below. Based on the single instance execution and associated outcomes, we comment on the performance improvements and analyze the complexities of the Enriched Ciphering algorithm.

The proposed modifications in the algorithm is presented below as a Pseudo code:

```
1: Define a cipher key
2: Assign number of elements of the matrix
3: Apply the permutation logic to generate the key elements
4: Input plain text
5: Compress the plain text
6: Disintegrate the input text into 128 bits round of text until the end
7: Input data type
8: BITXOR the data
9: Reshape the matrix
```

10: Retrieve the Cipher text

11: Encrypt the data

12: Use key to decrypt the encrypted data

13: Convert the cipher text into plain text

14: Use rotation and BITXOR

15: Decompress the data

16: Retrieve the plain text

5 Results

In this section we consider several test-cases to evaluate performance of the enriched ciphering method. The different cases considered in process of evaluating the performance starts with executing a round of plain text that has length of 128-bit. Self-time is the time spent in a function excluding the time spent in its child functions. Self-time also includes overhead resulting from the process of profiling. Total time is the self-time and time taken to run child functions as well. Call refers to the number of times function is called. dec2Hex, hex2dec, iscellstr are the internal functions in Enriched Ciphering algorithm the significance of these functions is that they are required to change key values when making matrix dynamic.

Case I: The instance described in Table 1 shows the numbers of function name and call. The call decides for the time. The functions are used while executing the Enriched Ciphering algorithm. They have a direct effect on run time of the Enriched Ciphering algorithm. In Figure 1, we show the self-time and real time on the axis and with a scale of .002 seconds. The Enriched Ciphering algorithm executes completely in .017s, sufficient time for the execution by the proposed solution.

Table I: Time slot details upon running the Enriched Ciphering algorithm

Function Name	Call	Total Time (Seconds)	Self-Time (Seconds)
Enriched Ciphering Algorithm	1	.017	.011
dec2hex	1	.004	.004
hex2dec	1	0.003	.003
iscellstr	1	0	.000

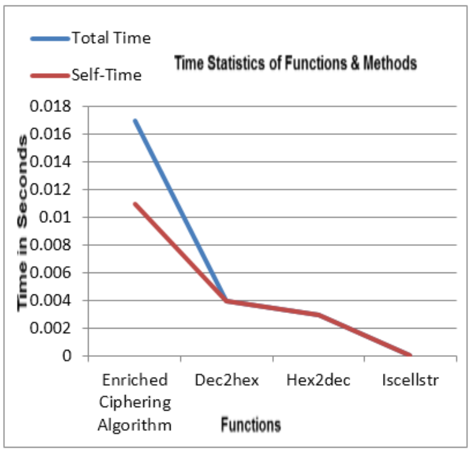


Figure 1: Illustration of total time and self-time in unit of seconds for a round of plain text, and different functions in the Enriched Ciphering algorithm

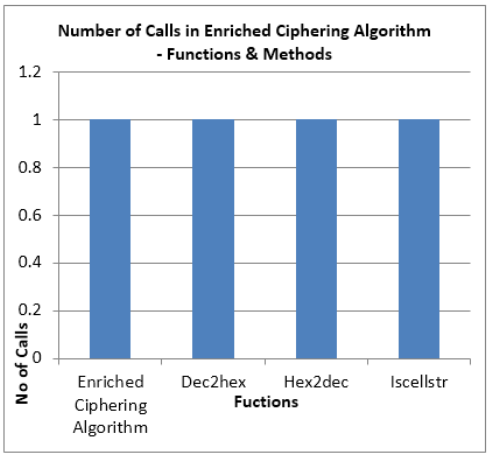


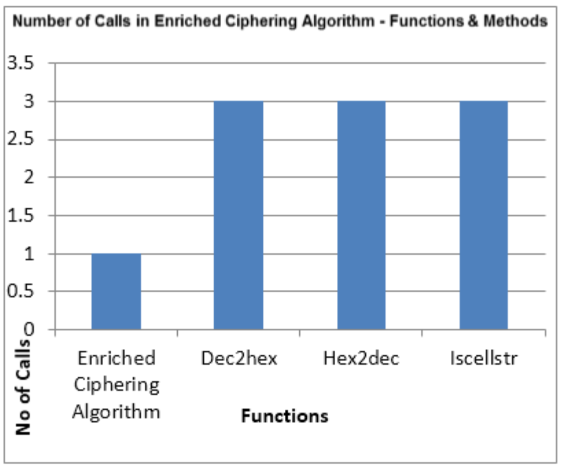
Figure 2: Illustration of number of calls for the functions associated to execution of Enriched Ciphering Algorithm

Case 2: In another case, the size of the string is increased and it is noticed that the changes occur in the execution of the Enriched Ciphering algorithm; Figure 3 shows us an increase in the number of calls upon increasing the length of the text size, when we notice the changes in terms of time complexity. Hence, we observe that there are more associated function and method calls for large size of data.

Figure 4 shows the increase in total execution time as per the increase in length of the string; although the changes are not constant; however the execution time is proportional to the length of the string which refers to the case when we execute the Enriched Ciphering Algorithm keeping the size of the string larger than 128 bits rounds of text as compared to case 1.

After execution of the Enriched Ciphering Algorithm the results show us an increase in the execution time upon increasing the length of the text; hence the execution time of Enriched Ciphering algorithm is proportional to the number of rounds per bits of text. Although the number of functions and methods called during the execution of Enriched Ciphering Algorithm are almost the same in contrast to the number of functions called by case 1.

The number of calls of the associated functions in Case2 is more than that in case 1 because the size of the string is increased from one round of text which has length of 128 bits then to the two round of text which is of length 256 bits approximately. The total time of execution of Enriched Ciphering Algorithm is more in case 2 as compared to the time of execution of Enriched Algorithm in Case 1. It is .084 seconds in this case and .017 seconds in the former. Through these cases we determine and evaluate the performance of the Enriched Ciphering Algorithm with different cases and present the potential areas for performance improvement.



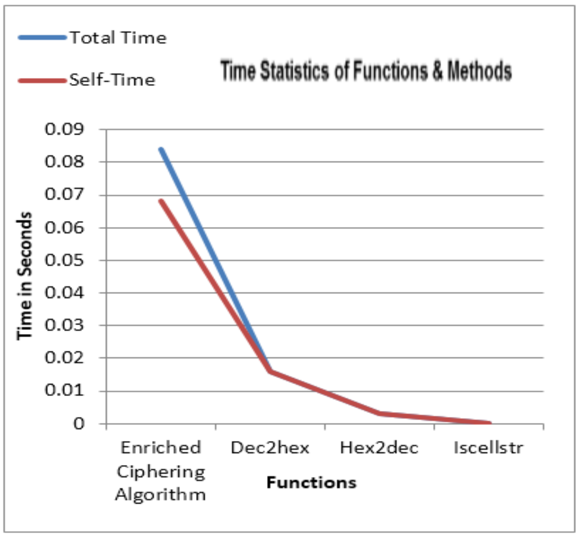


Figure: 3 Illustration of number of calls for the functions associated to execution of Enriched Ciphering algorithm with a larger string size.

Figure 4: illustration of total time and self-time in unit of seconds for different functions in the Enriched Ciphering algorithm for larger size of text

Table 2: Time slot details upon running the Enriched Ciphering algorithm and increasing the length of the plain text; Size Increase in the length of Plain

Function Name	Call	Total Time (Seconds)	Self-Time (Seconds)
Enriched Ciphering Algorithm	1	.084	.068
Dec2hex	3	.016	.016
Hex2dec	3	0.003	.003
Iscellstr	3	0	.000

Case 3: A further change of the enhancement in the proposed solution relates to large text files that can be processed in a secure and fast manner. In order to address memory issues we suggest the use of compression technique in the Enriched Ciphering algorithm and analyze the time domain at the runtime execution.

Figure 5 illustrates the number of calls associated to the functions in Enriched Ciphering algorithm per round of text when zip function is used. Create Archive and get archive are the methods and functions associated to Enriched Ciphering algorithm use to store and call values while execution of Enriched Ciphering algorithm.

Table 3: Function details upon integration of compression technique with the Enriched Ciphering algorithm, and analysis of statistics with path of associated functions a method.

Function Name	Call	Total Time (Seconds)	Self-Time(Seconds)
Enriched Ciphering Algorithm	1	.147	.081
Zip	3	.066	.000
Create	3	.066	.000
Get	3	.049	.000
Unique	6	.017	.000
Fileparts	9	.017	.017
Add	3	.017	.000
(Java Method)	15	.017	.017
Legacy	6	.017	.017
Fullfile	3	.016	.016

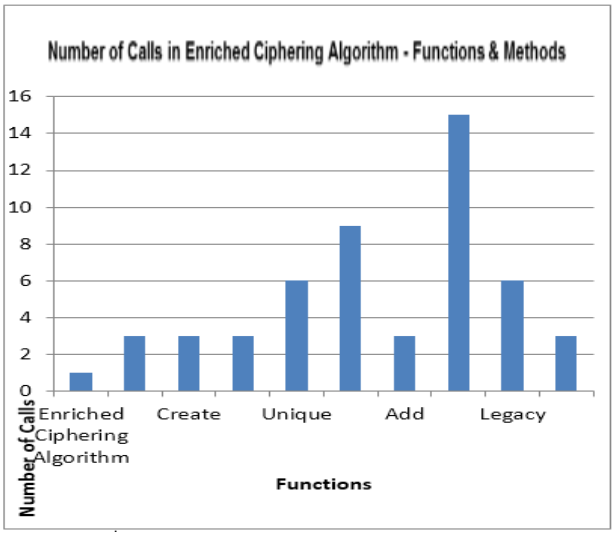


Figure 5: Illustration of number of calls when Enriched Ciphering algorithm execution involves zip function per round of text.

This case is based on an approach to obtain the overhead involved in executing the zip function at the initial level of a round of text. With this we investigate the use of encryption technique in an Enriched Ciphering algorithm when the text is small. We notice the increase in execution time and number of calls for functions.

Case 4: Compression technique stands for the memory concerns for the users. Now, finally there is an increase in the runtime of an Enriched Ciphering algorithm when using compression technique in an effort to address storage and memory issues. Here, in Case 4, our exploration of the Enriched Ciphering algorithm begins with increase in length of the string. At Table 4, the functions, and the calls appear to be more also illustrated in Figure 7. But there is a decrease in execution time as per Figure 8. Hence, we determine that compression technique yields efficient results after runtime execution when a larger size of text is involved in the process and it addresses memory concern as well.

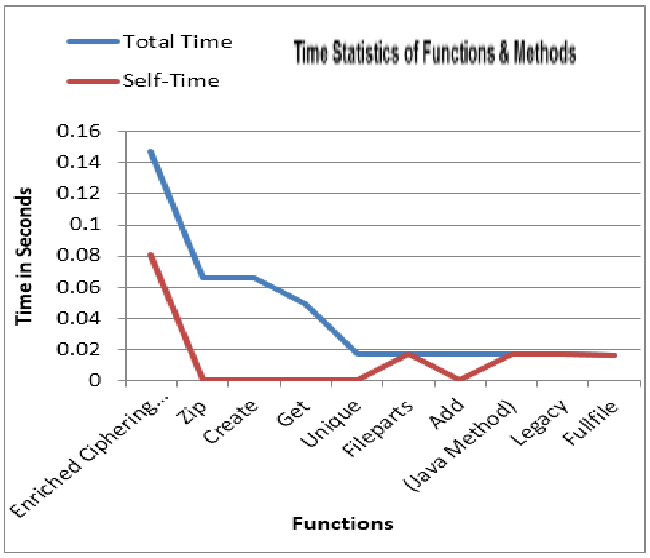


Figure 6: Illustration of total time and self-time in unit of seconds for different functions in the Enriched Ciphering algorithm when compression is involved for an initial round of text.

Table 4: Compressing and increasing the length of the plain text, and evaluating the time statistics with path of associated functions and methods

Function Name	Call	Total Time (Seconds)	Self-Time (Seconds)
Enriched Ciphering Algorithm	1	0.127	0.063
Zip	1	0.048	0
Create	1	0.048	0
Get	1	0.032	0.015
Unique	2	0.016	0.016
Dec2hex	1	0.016	0.016
Add	1	0.016	0
Get	3	0.016	0.016

Case 5: - In this scenario, we include the decompression step in order to calculate the runtime execution of the Enriched Ciphering algorithm. We integrate the unzip function to the scenario executed in Case 4 of this section, in the Enriched Ciphering algorithm and execute it.

Upon execution, we get the runtime details as mentioned in Table 5. This is the case of compression and decompression technique on a round of text where new functions also take part upon in the processing of Data upon execution of Enriched Ciphering Algorithm. Since there are few new processes involved in the Enriched Ciphering algorithm because of the changes done in the length of string and also due to the new functions and methods involved in order to evaluate the performance of the ECA.

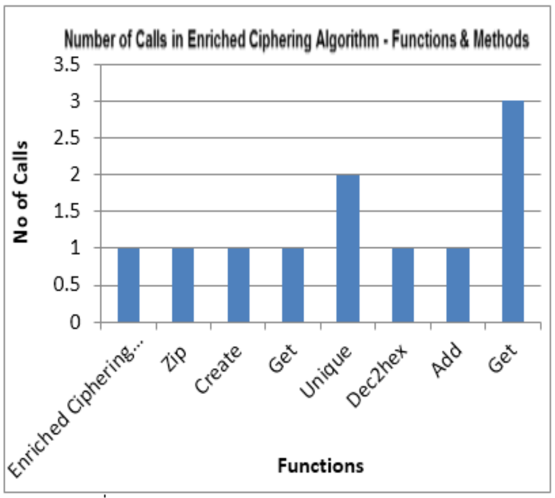


Figure 7: Illustration of number of calls when Enriched Ciphering algorithm execution involves zip function for a larger size of text

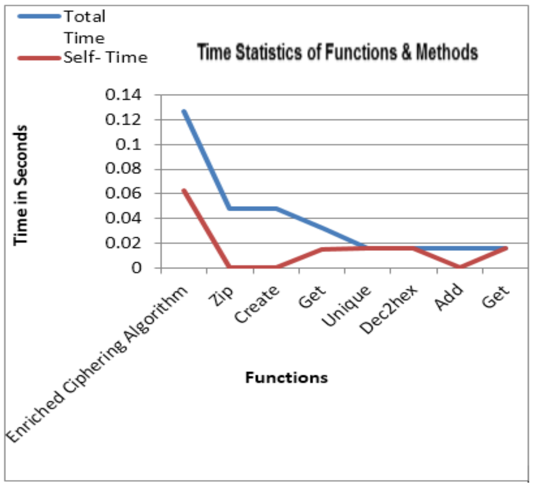


Figure 8: Illustration of total time and self-time in unit of seconds for different functions in the Enriched Ciphering algorithm when compression is involved for a larger size of text

Figure 9 shows the number of calls for the functions and methods involved in the compression and decompression technique upon integration within the Enriched Ciphering Algorithm, and when the Enriched Ciphering algorithm is executed then the indicated calls shows the execution statistics of the associated child functions as well. These associated functions appear because of their association with the decompression stage in the Enriched Ciphering algorithm.

Create, Get and add are some of the functions and methods for the Archive entries of the files having data, whereas parse, Dec2hex, Num2Str indicates the parsing, processing, and conversion of different data types. The input string processing of the file takes place in splits of data carried by File parts. Zip and Unzip functions are the direct operation on the specific set of Data in order to save memory allocation and fine tune the performance on large set of Data.

Table 5: Decompressing a round of text and evaluating the time statistics with path of associated functions and methods

Function Name	Call	Total Time (Seconds)	Self-Time(Seconds)
Enriched Ciphering Algorithm	1	0.155	0.078
Zip	1	0.047	0.015
Create	1	0.032	0
Dec2hex	1	0.016	0.016
Get	1	0.016	0
Add	1	0.016	0
Convert	4	0.016	0.016
Num2str	1	0.016	0
Int2str	1	0.016	0.016
Mode	1	0.016	0
Get	3	0.016	0
Parse	1	0.015	0
Fileparts	4	0.015	0.015
Unzip	1	0.015	0

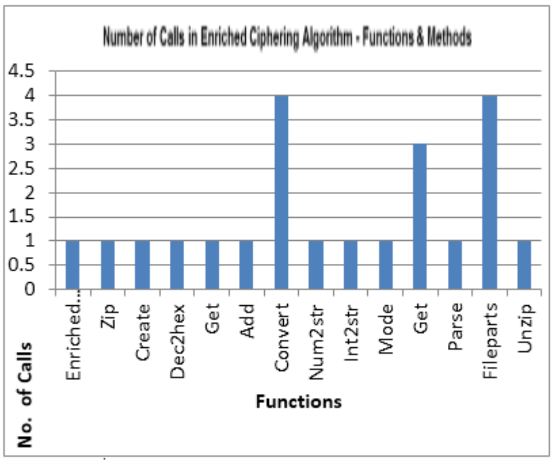


Figure 9: Illustration of number of calls with zip and unzip functions for a round of text in ECA.

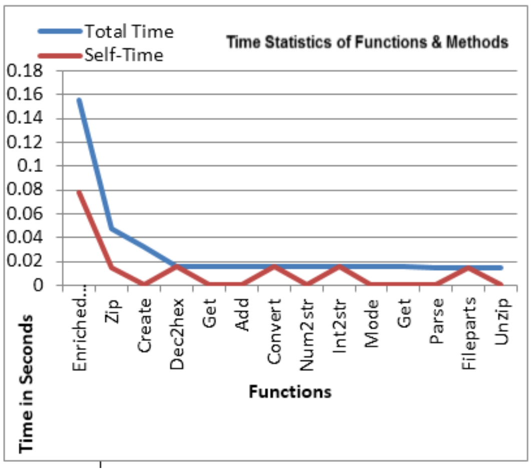


Figure 10: Illustration of total time and self-time in unit of seconds for different functions in the Enriched Ciphering algorithm when compression and decompression techniques are involved for an initial round of text.

Compression and decompression are mostly intended to save memory usage in a digital world. The time statistics described in Figure 10 and related to self-time and total time help us to analyze a slight increase in the curve of time upon including the decompression technique within the Enriched Ciphering algorithm.

Case 6: With larger text, the compression and decompression functions need to be emphasized more. We notice that there has been an increase in functions involved. In each function, the numbers of calls precede the function calls for smaller size of text. Table 6 gives the information of the number of calls, and Figure 12 provides a graphical view of the number of calls associated with the functions involved, and those called during runtime execution of the Enriched Ciphering algorithm. It is 0.32 seconds of time the Enriched Ciphering algorithm takes for complete execution (see Figure 11).

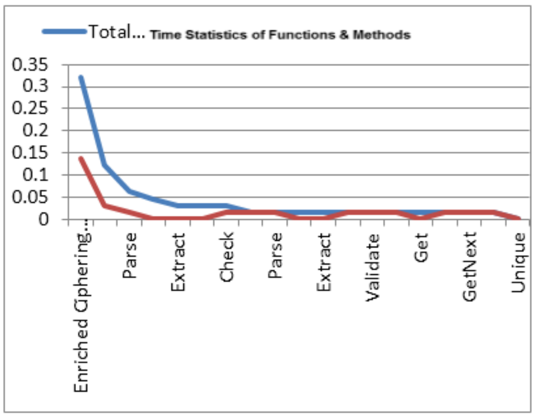


Figure 11: Illustration of total time and self-time in unit of seconds for different functions in the Enriched Ciphering algorithm when compression and decompression techniques are involved for larger set of Data and an initial round of text.

Table 6: Details of runtime execution of Enriched Ciphering algorithm upon integrating decompression with the larger text in order to evaluate the time statistics with path of associated functions and methods.

Function Name	Call	Total Time (Seconds)	Self-Time (Seconds)
Enriched Ciphering Algorithm	1	0.321	0.136
Unzip	3	0.123	0.03
Parse	3	0.062	0.016
Zip	3	0.046	0
Extract	3	0.031	0
Create	3	0.03	0
Check	3	0.03	0.015
Hex2dec	3	0.016	0.016
Parse	3	0.016	0.016
Add	3	0.016	0
Extract	3	0.016	0
(Java method)	9	0.016	0.016
Validate	3	0.016	0.016
Converte	3	0.016	0.016
Get	9	0.016	0
Isdir	3	0.016	0.016
GetNext	6	0.016	0.016
GetArc	3	0.015	0.015
Unique	6	0	0

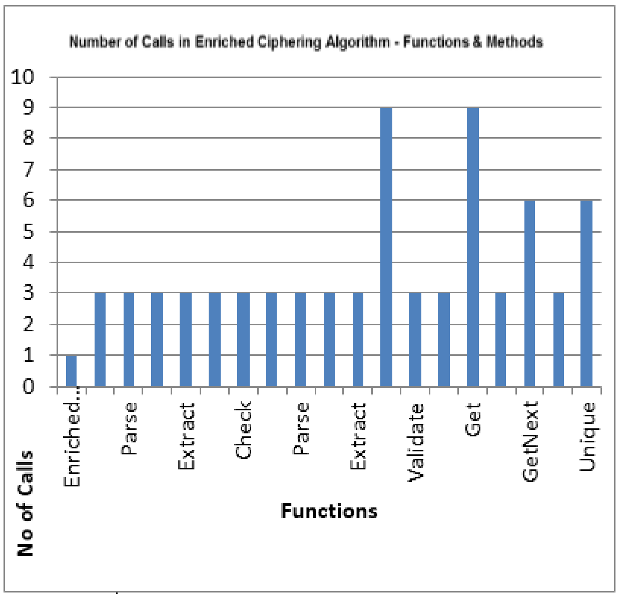


Figure 12: Illustration of number of calls when Enriched Ciphering algorithm execution involves zip and unzip functions for a larger set and per round of text.

6 Conclusion

We investigated and evaluated the Ciphering algorithm to add matrix block in order to strengthen LTE security. In this paper we present the results for performance evaluation based on different performance vectors. The results presented are real runtime execution and accurate. The simulation provides an estimated time increase of .067s being 3.9 times original of .017s execution time of Enriched Ciphering Algorithm when text size increases from 128 bits to 256 bits.

Moreover, we provide a real time listing of function calls during execution in the quest of modularizing the complete process to embed security at each phase. We also determine the increase in calls, and those are involved and appear into the system. Besides representing several other cases, we also provide the decrease of 13.60% upon including compression technique on large size of data sets to that of smaller size. We propose including compression and decompression for performance improvement to be a part of the process especially when we are dealing with large chunk of data sets so as to meet our memory allocation needs. We represent our research on integrating new techniques in the implementation and the need of improvement and strengthening LTE Security.

REFERENCES

[1]. "Lianfen Huang*, Ying Huang, Zhibin GAO, Jianan Lin, Xueyuan Jiang", Performance of Authentication Protocols in LTE Environments, 2009 International Conference on Computational Intelligence and Security, 293-297,978-0-7695-3931-7/09 © 2009 IEEE DOI 10.1109/CIS.2009.50.

[2]. "Bin Liu, IEEE, and Bevan M. Baas.", Parallel AES Encryption Engines for Many-Core Processor Arrays,"IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, and NO. 3 MARCH 2013", 536-547, www.computer.org/publications/dlib.

[3]. "Naga M. Kosaraju, Murali Varanasi, Saraju P. Mohanty", A High-Performance VLSI Architecture for Advanced Encryption Standard (AES) Algorithm, Proceedings of the 19th International Conference on VLSI Design (VLSID'06) ,1-4,1063-9667/06 © 2006 IEEE.

[4]. "Fei Shao, Zinan Chang, Yi Zhang", AES Encryption Algorithm Based on the High Performance Computing of GPU, 2010 Second International Conference on Communication Software and Networks,588-590,978-0-7695-3961-4/10 © 2010 IEEE DOI 10.1109/ICCSN.2010.124.

[5]. "Keisuke Iwai and Takakazu Kurokawa, Naoki Nisikawa", AES encryption implementation on CUDA GPU and its analysis, 2010 First International Conference on Networking and Computing, 209-214,978-0-7695-4277-5/10 © 2010 IEEE DOI 10.1109/IC-NC.2010.49.

[6]. "Scheme,Khodor Hamandi Imad Sarji Ali Chehab Imad H. Elhajj Ayman Kayssi", Privacy Enhanced and Computationally Efficient HSK-AKA LTE, 2013 27th International Conference on Advanced Information Networking and Applications Workshops,929-934,978-0-7695-4952-1/13 © 2013 IEEE DOI 10.1109/WAINA.2013.133.

[7]. "Mehran Mozaffari-Kermani, and Arash Reyhani-Masoleh", Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard, "IEEE TRANSACTIONS ON COMPUTERS, VOL. 59, NO. 5, MAY 2010",608-622,0018-9340/10/ ©2010 IEEE.

[8]. "Amit Pande and Prasant Mohapatra", Securing Multimedia Content Using Joint Compression and Encryption ,2013 Oct- Dec Published by the IEEE Computer Society,50-61,1070-986X/13/ c 2013 IEEE.

[9]. "Krzysztof Jankowski and Pierre Laurent", Packed AES-GCM Algorithm Suitable for AES/PCLMULQDQ Instructions, "IEEE TRANSACTIONS ON COMPUTERS, VOL. 60, and NO. 1, JANUARY 2011", 135-138, 0018-9340/11/ 2011 IEEE Published by the IEEE Computer Society.

[10]. "Gautam Siwach, Amir Esmailpour ", LTE Security Potential Vulnerability and Algorithm Enhancements, 2014 IEEE Canadian Conference on Electrical and Computer Engineering, 1-7, IEEE CCECE May 2014.