2019 22nd International Conference on Computer and Information Technology (ICCIT), 18-20 December 2019

An Enriched Kasumi Encryption Algorithm by Modifying F-functions for 4G LTE

Syeda Ishrat Mahjabin
Computer Science & Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
imnnishat@gmail.com

Tanvir Ahmed
Computer Science & Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
tanvir.cse@aust.edu

Sakif Azline
Computer Science & Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
sakifazline0@gmail.com

Timittra Hridi
Computer Science & Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
timittra.cse.aust@gmail.com

Zuheb Ahmed
Computer Science & Engineering
Ahsanullah University of Science and Technology
Dhaka, Bangladesh
zuhebahmed40@gmail.com

Md. Ashraful Hoque
Computer Science & Engineering
Southeast University
Dhaka, Bangladesh
ashraful@seu.edu.bd

***Abstract*—Long Term Evolution (LTE) has become most wide-ranging Fourth Generation (4G) cellular networks for its high speed data rates and advanced features. This new generation network provides more security than previous generation mobile communication system. But this popular and secure telecommunication system is also affected by some cryptanalysis under some cases. Thus it requires strong security algorithms for users and network security purposes. Kasumi is one of the most popular block cipher security algorithms that used F-function composed of FI, FL, and FO subfunctions. The function F is proclaimed as the core of the block ciphers. It creates the relationship between cipher text and encryption key more complex by offering confusion property.**

In this paper, the components of F-function of KASUMI have been investigated to determine its cryptographic strength and proposed an enhanced Kasumi algorithm by modifying its functionalities. Finally, this paper shows the comparison in performances between original and improved Kasumi algorithms.

***Index Terms*—Kasumi, optimization, time, memory**

I. Introduction

In the past years, a variety of mobile telecommunication systems has been evolved. For instance, the second generation which is global System for Mobile Communications (2G-GSM) commercially launched in 1991[1]. In 2002 3G-UMTS has been appeared [3], and in 2010 the fourth generation of Long Term Evolution (4G-LTE) technology has been developed [5]. Nowadays, more people are connected via mobile communication systems and huge confidential information such as bank transaction, emails and voice calls being exchanged through these systems. Therefore, there is a demand for security mechanisms like symmetric cryptographic algorithms to address security problems of information confidentiality, authenticity and integrity. As there are many cryptographic algorithms for securing mobile communications but for securing LTE 4G, an encryption algorithm named Kasumi cipher which is a block cipher algorithm is most popularly used. Kasumi converts the plain text by taking one block at a time [4]. This algorithm creates barrier to secure LTE 4G communication and also fights against different security attacks. But this has some weaknesses. It is affected by some complex security attacks and discloses users personal information to the attackers [6,7,8]. Thus the attackers get chances to execute their evil motives. To minimize the security risks of attacks and provide subscribers a secure network, it is very important to improve the performance of Kasumi algorithm so that it can prevent different and complex security attacks to make the connection more secured.

The main objective of this paper is to optimize the original Kasumi algorithm by reducing its execution time and memory. The F-functions- FI, FO and FL and the Key Schedule function of Kasumi have been modified in this paper to enhance the overall performance of Kasumi. The FI function has been optimized by eliminating the variable dependencies. FO function is modified by shortening some of its operations. In FL function, the input and output module have been modified for further improvement. In case of Key Schedule function, simple loop merging technique is used. In this paper, several different 64-bit plain texts have been taken as input for testing and observed the 64-bit cipher text as output for both original and optimized Kasumi algorithm to get the visible results of reducing execution time and memory.

This paper is organized as follows. Section II describes the basic structure of Kasumi algorithm. In section III, shows proposed algorithm by modifying all three F-functions, FL, FO and FI as well as Key Schedule function. In section IV the simulation result has been shown. Section V describes the

978-1-7281-5842-6/19/$31.00 ©

performance analysis as well as a clear comparison between original and optimized Kasumi. The last section VI concludes the whole paper in short.

II. KASUMI

Kasumi is a block cipher algorithm which is the modified version of MISTY1 [2]. It produces 64-bit output from the 64-bit input under the control of a 128-bit key. Kasumi decomposes into a number of sub-functions (FL, FO, FI) which are used in conjunctions with associated sub-keys (KL, KO, KI) in a Feistel structure comprising a number of rounds [4].

A. F-functions of Kasumi

- FL subfunction takes 32-bit strings as an input and outputs 32-bit strings with 32-bit associated subkey, KL. The 32-bit input string splits into two 16-bit strings and then perform operations. FL subfunction mainly performs logical operation with its KL subkey which is divided into two parts (Figure-1) such as $KL_{i,1}$ and $KL_{i,2}$ [11].

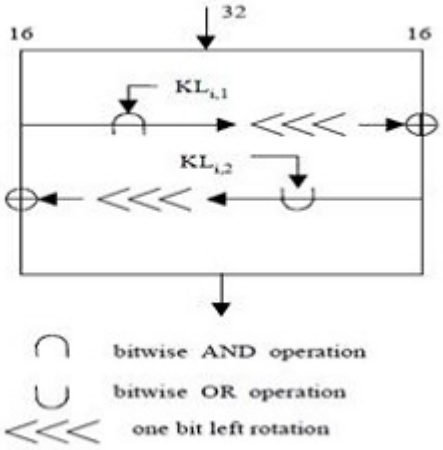

Fig. 1. FL Subfunction [4].

- FI subfunction takes 16-bit input and output 16-bit string as shown in Figure-2. This input string is divided into two unequal bits 7-bit and 9-bit. This uses two S-boxes. S7 box uses to map a 7-bit input into a 7-bit output and S9 box maps a 9-bit input into a 9-bit output [11]. FI subfunction of Kasumi uses two additional functions ZE() and TR():
  ZE(x) - It takes the 7-bit value x and converts it into a 9-bit value.
  TR(x) - This function takes the 9-bit value x as input and converts it into a 7-bit value to show as output.
- FO subfunction takes 32 bits input string and displays 32 bits string as output (Figure-3) with two 48 bits subkeys KO and KI [9]. This uses three rounds of FI function.

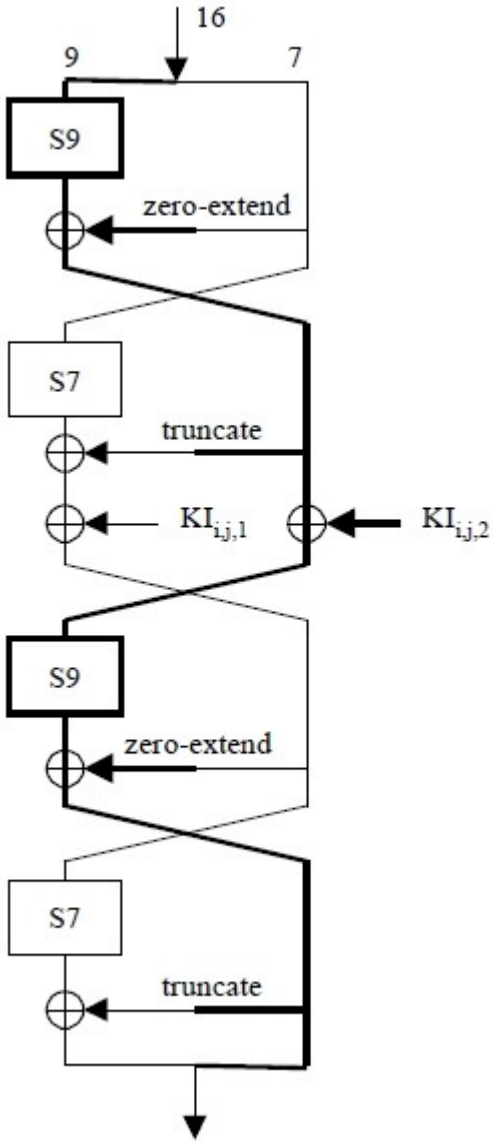

Fig. 2. FI Subfunction [4].

B. Sub-Keys of Kasumi

In Kasumi there are three subkeys are used with their associated subfunctions. KL, KI and KO are the three subkeys of Kasumi (Figure-3).

- KL is a 32 bits subkey of subfunction FL.
- KI is a 16 bits subkey of subfunction FI.
- FO function uses two subkeys. As it uses three rounds of FI so it has two associated subkeys KO and KI.

III. OPTIMIZATION OF KASUMI ALGORITHM

In this paper, the original Kasumi algorithm [4] has been modified and optimized to reduce its execution time and memory so that it can perform faster than the previous algorithm by using less time and memory for execution. In case of encryption algorithms, if the performance is improved then it would secure the specific network more as well. Here, changes

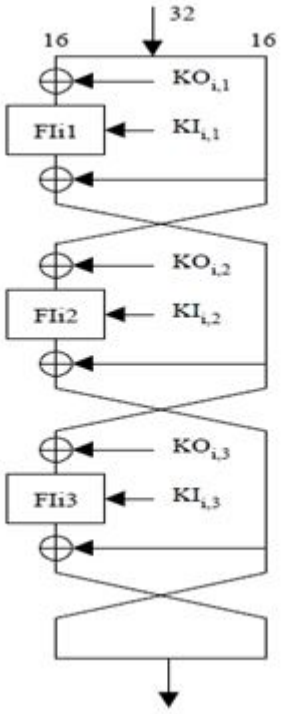

Fig. 3. FO Subfunction [4].

are made in FI, FO and FL subfunctions of Kasumi with the Key Schedule function.

A. FI Sub-Function

The Kasumi C program written in paper [4] has been rewrite and also the original algorithm has been modified accordingly to eliminate the variable dependencies by taking additional local variables inside the subfunction to complete different operations using S9 and S7 boxes for intermediate computational results. Using too many local variables can degrade the performance of the algorithm. Only necessary local variables are used to decrease the time and space complexity of modified Kasumi algorithm. The equations inserted below show the changes made in original FI function. Instead of taking two variables here four local variables are taken for the same task. Using of local variables divided the FI function into two parts internally.

The part-1 (Figure-4) works with two local variables, nine1 and seven1, where,

$$nine1 = (input >> 7) \tag{1}$$

$$seven1 = (input \ \& \ 0x7F) \tag{2}$$

Here, nine1 variable uses S9 box and seven1 variable uses S7 box of Kasumi for once during FI function execution. The boxes (S7 and S9) are designed to implement combinational logic of Kasumi algorithm easily.

In Figure-4 the part-1 shows that these two local variables also use two unequally divided subkey $KI_{i,j,1}$ and $KI_{i,j,2}$. These subkeys are 9-bit and 7-bit and splitted from 16-bit main subkey KI of FI function. Equation (3) and (4) are showing the calculation of part-1 using subkeys.

$$sres = seven1 \bigoplus (subkey >> 9) \tag{3}$$

$$nres = nine1 \bigoplus (subkey \ \& \ 1FF) \tag{4}$$

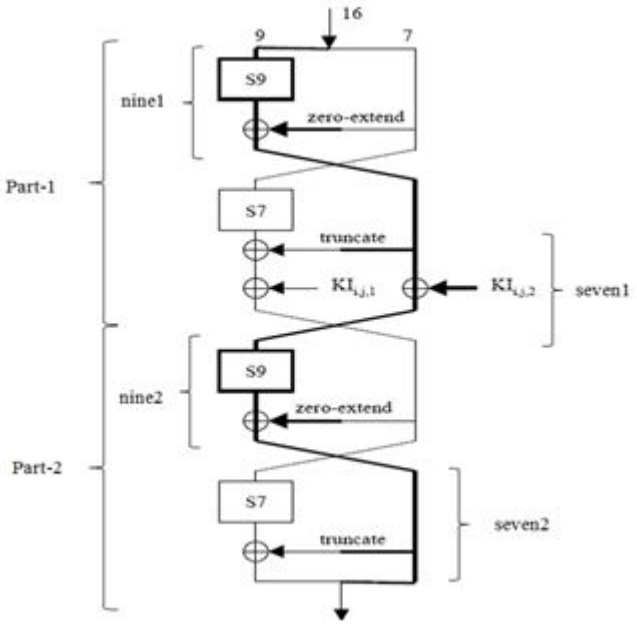

Fig. 4. Optimized FI Function.

From the part-2 of Fig. 4, it can be seen that two more local variables are used, nine2 and seven2. Both nine2 and seven2 variables work with the previous computation of nine1 and seven1. These two variables use S7 and S9 boxes of Kasumi for the second time to generate final output. But in case of part-2 of this modified FI function, no new subkeys are introduced. The equations (5) and (6) showing the working procedure of seven2 and nine2.

$$seven2 = (S7[sres] \bigoplus (nres \ \& \ 0x7F)) \tag{5}$$

$$nine2 = (S9[nres] \bigoplus sres) \tag{6}$$

Finally, another local variable has been returned by containing the final result of FI() function.

This optimization does not affect or change the original result of FI function. But eliminating variable dependencies reduced the execution time and memory to a short extent and helps to improve the performance Kasumi algorithm

B. FO Sub-Function

FO function has been simply shortened by converting some larger operation into smaller operations without affecting the result of the original algorithm. This optimized function of Kasumi differs a little than the original FO function of original Kasumi algorithm. (7) and (8) equations mentioned below shows the optimization inside FO function where Lj indicates left side operations and Rj indicates right side operations. In shortened function the left side first takes 16-bit input splitted from 32-bit input and the input gets XORed with a part of KO and KI subkey and then the result gets XORed with right side 16-bit input string. The right side operations follow the same path as left side but it takes right side 16-bit input string first and after necessary XORing with KO and KI it gets XORed with left side input string to produce the 32-bit final result of optimized FO function.

$$L_j = FI(L_{j-1} \bigoplus KO_{i,j}, KI_{i,j}) \bigoplus R_{j-1} \tag{7}$$

$$R_j = FI(R_{j-1} \bigoplus KO_{i,j}, KI_{i,j}) \bigoplus L_{j-1} \tag{8}$$

C. FL Sub-Function

In FL subfunction optimization, we have worked with one by one side at a time. According to original Kasumi, the 32 bit input string is divided into two parts, 16-bit each. The figure-5 is the structural representation of the FL function optimization. Firstly, the input of 16-bit string has been taken separately from the total 32-bit strings input to compute the tasks of left side of function. The first 16-bit of input string of one side (considered as left side input string in original Kasumi) is intersected by KLi,1, 16-bit subkey of FL function which is one half part of KL, 32-bit subkey of FL function. After this intersection, we get the primary result from left side. Then used the another 16-bit string as input which is considered as right side input in original Kasumi. Then the new input string has been XORed with the first result that is previously computed. A new result has been computed after XORing and that has been united with KLi,2, another 16-bit subkey which is the second half part of KL 32-bit subkey and lastly the whole result after Union has been XORed with the first 16-bit input string. In the modification of this paper, the last operation takes the two separately taken 16-bit input string together and displays the result which is altogether a 32-bit output string.

To say more about FL function modification, this has also been shortened by converting multiple line operations into single line operation.

The input of the new optimized function FL contains a 32-bit data input string I and a 32-bit subkey KLi like the original Kasumi.

The given equation(9) is showing that the 32-bit subkey KLi has been splitted into two more subkeys- KLi,1 and KLi,2. The equation(10) defines the L variable that takes first 16-bit string as input. Equations (11), (12) and (13) are representing the computations of FO function. Finally, (14) presents the final result of FO function.

$$KL_i = KL_{i,1} || KL_{i,2} \tag{9}$$

$$L = I >> 16 \tag{10}$$

$$A = L \cap KL_{i,1} \tag{11}$$

$$R = I >> 16 \tag{12}$$

$$L' = R \bigoplus ROL(A) \tag{13}$$

$$Res = ROL(L' \cup KL_{i,2}) \bigoplus L \tag{14}$$

The structure of optimized FL function is given below.

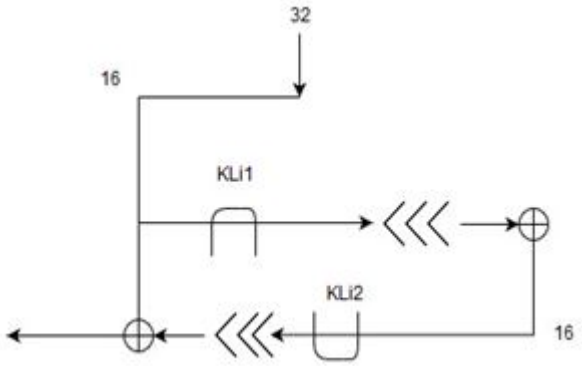

Fig. 5. Optimized FL Function.

D. Key Schedule Function

Kasumi uses a 128-bit key which can be considered as K. Each and every rounds of Kasumi uses 128-bit key those are derived from K. The Key Schedule function does the key derivation of Kasumi algorithm. In this paper, inside key schedule function we have done simple loop merging to combining different loops to one. In case of reducing time and space, this loop merging helps a lot. To combine multiple loops into one can help to reduce the size of the original code so that it will take less execution space in RAM also it inroduces instruction level parallelism that can decrease the execution time and increase running speed. The merged loop of Key Schedule function is given below:

for $num = 0$ to num++ **do**
  $key[num] \leftarrow (k[2n] < 8) + k[2n+1]$
  $kPrime[num] \leftarrow key[num] \wedge C[num]$
end for

IV. Simulation Results

The new optimized algorithm was tested using the NIST[10] statistical tests using Cygwin64 Terminal, a linux application by running NIST test suite and all 15 tests of randomness for this algorithm had been passed. The encryption time required by the optimized Kasumi algorithm shows the decrement by some seconds compared to the original Kasumi algorithm. Also the modified algortihm takes less memory to execute compared to the original one. The authenticity of this analysis can be verified from Table-1 and Table-2 also from the Figure-6 and Figure-7 with the clear graphical representation.

TABLE I
THE ORIGINAL KASUMI ALGORITHM

Sl. No. of Iterations	No. of Iterations	Execution Time taken in second	Execution Memory taken in MB
1	5 x 10^4	0.25	1.7
2	1 x 10^5	0.438	1.75
3	4 x 10^5	1.968	1.95
4	6 x 10^5	3.67	2.18
5	8 x 10^5	4.22	2.32
6	10 x 10^5	5.797	2.58
7	12 x 10^5	6.897	2.77
8	14 x 10^5	8.65	2.97
9	16 x 10^5	10.374	3.09
10	18 x 10^5	10.95	3.29
11	20 x 10^5	11.67	3.51

TABLE II
THE OPTIMIZED KASUMI ALGORITHM

Sl. No. of Iterations	No. of Iterations	Execution Time taken in second	Execution Memory taken in MB
1	5 x 10^4	0.14	1.64
2	1 x 10^5	0.343	1.7
3	4 x 10^5	1.468	1.85
4	6 x 10^5	2.65	2.05
5	8 x 10^5	3.437	2.2
6	10 x 10^5	4.469	2.33
7	12 x 10^5	5.953	2.63
8	14 x 10^5	7.58	2.88
9	16 x 10^5	9.062	2.99
10	18 x 10^5	9.577	3.18
11	20 x 10^5	10.23	3.39

There are many ways of testing execution time and space of algorithms but in this paper, the computation is done by taking a large array of numbers in a single time to both the original and modified Kasumi algorithm to find out the approximate time and memory taken. The number of iterations column in both tables indicate the number of times, the output texts are generated except any hold. In this paper, total 2x10^5 number of iterations is computed for both the execution time and space. This amount of iteartions shows the clear comparison between original and optimized Kasumi for this reason more iterations were not required. In case of less amount of iterations, the differences between two algorithms would not be understood well. The testing was software based. Time testing was done by Cygwin64 terminal, a Linux based Application and the execution memory testing was computed by geeksforgeeks online IDE. The testing processor was Intel(R) Core™ i3-4130 CPU @ 3.40 GHz. This result will vary for more advanced processor like Core i5 or Core i7. The optimized algorithm will show far better results and will be more attack resistant if the processor is faster and more advanced.

From the data of Table-1 and Table-2 the comparison of execution time and memory between original Kasumi and optimized Kasumi algorithm can be clearly stated.

V. Performance Analysis

The given figures, Figure-6 and Figure-7 are the graphical representation of our analysis so that we can understand the difference between the original and proposed(optimized) Kasumi algorithm's required execution time and space clearly.

In X-axis the SL. No. of iterations indicates a serial number for each iteration e.g. SL No. of iterations is 1 when the actual number of iterations is 5x10^4 and in Y-axis of the Figure-6 indicates the Execution time in seconds which we have computed. Similar goes for Figure-7. In the case of Figure-7, the Y-axis indicates execution memory in MB.

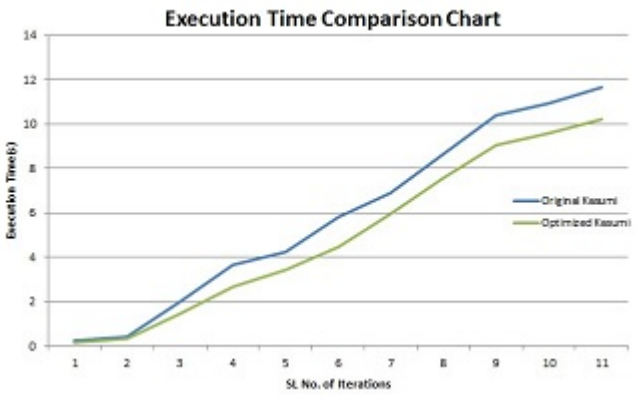

Fig. 6. Execution Time Comparison Chart (Kasumi and Optimized Kasumi).

Figure-6 shows that the execution time has decreased in Optimized Kasumi algorithm. When the number of iterations are fewer, the original and optimized Kasumi algorithm doesn't show visible differences. But from the above figure, it can be clearly understood that with the increasing of iterations number, the optimized Kasumi algorithm takes less time for execution and shows better performance than the original one and it becomes visible in graph(Figure-6) from the 4x10^5 iterations. From then the differences between time increased simultaneously.

The Figure-7 is showing the clear graphical representation of Execution Memory Comparison between original Kasumi and optimized Kasumi algorithm. From this figure, the comparison between the execution memory taken by both original and optimized Kasumi algorithm can be clearly stated. The optimized algorithm of Kasumi did not reduce the execution memory much but it certainly requires less memory to execute than original Kasumi.

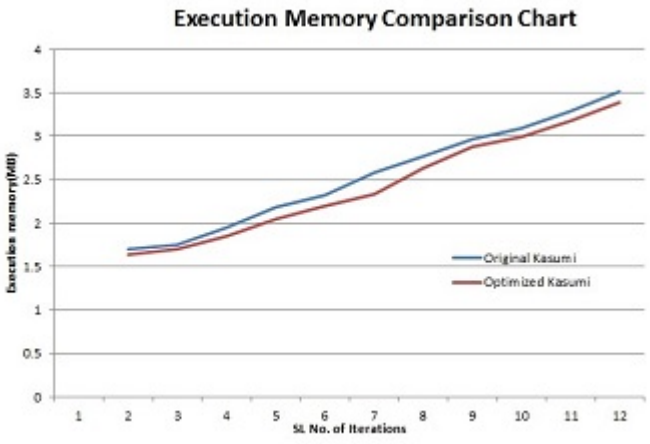

Fig. 7. Execution Memory Comparison Chart (Kasumi and Optimized Kasumi).

From both result analysis, it is clear that optimized Kasumi performs better than the original Kasumi as it reduces both its execution time and memory. From the study of different cryptanalysis against Kasumi it is found that all the attacks against original Kasumi requires a fixed amount of data in a fixed amount of time e.g. boomerang attack, rectangle attack etc.[7] But in this optimization, the execution time has been reduced and now it can process more data in fewer times. The data and time computation of the attackers get changed due to this optimization. So, the optimized one becomes more attack resistant than the original and capable of providing more security to the LTE 4G subscribers by reducing attacking probabilities.

VI. Conclusion

The main purpose of this research was to optimize Kasumi- an encryption algorithm to get better performance for securing 4G LTE more than the previous algorithm. In this paper, the components of F-functions of Kasumi such as FI, FO and FL functions and also Key Schedule Function have been optimized and this optimization reduced the execution time, execution memory of original Kasumi algorithm. The new Kasumi algorithm is capable of producing more encrypted data in less time using less memory due to these optimizations alongside it makes more strong security barriers against attacks than the original one.

Acknowledgment

We would like to acknowledge everyone who played important roles in our academic accomplishments. Special thanks to our supervisor, Tanvir Ahmed. His excellent guidance, supervision and inspiration made this research possible. Heartiest gratitude goes to our research fellows for their contributions and to those network security researchers who researched on Kasumi algorithm as their studies helped us a lot to pursue our goals in this field.

References

[1] Jay R Churi, Sudhish T Surendran, Shreyas Ajay Tigdi and Sanket Yewale. Article: Evolution of Networks (2G-5G). IJCA Proceedings on International Conference on Advances in Communication and Computing Technologies 2012 ICACACT(3):8-13, August 2012.

[2] Rjoub, Abdoul & Ghabashneh, Ehab. (2018). Low Power High Speed MISTY1 Cryptography Approaches Journal of Circuits, Systems and Computers. 27. 10.1142/S0218126618502006Biham E., Dunkelman O., Keller N. (2005) Related-Key Boomerang and Rectangle Attacks. In: Cramer R. (eds) Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science, vol 3494. Springer, Berlin, Heidelberg.

[3] Wireless Mobile Communication - A Study of 3G Technology Amit K. Mogal Department of Computer Science, CMCS College, Nashik-13 Email: amit.mogal@gmail.com, Int. J. Advanced Networking and Applications, Volume: 03 Issue: 05 Pages:01-06 (2012) Special Issue of NCETCSIT 2011 - Held on 16-17 Dec, 2011 in Bapurao Deshmukh College of Engineering, Sevagram.

[4] Specification of the 3GPP Confidentiality and Integrity Algorithms - Document 2: KASUMI Specification, 3GPP TS 35.201 version 4.1.0 Release 4, ETSI,Reference: RTS/TSGS-0335201Uv4R1.

[5] A Close Examination of Performance and Power Characteristics of 4G LTE Networks, Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, Oliver Spatscheck, University of Michigan, AT&T Labs - Research.

[6] A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony, Orr Dunkelman, Nathan Keller , and Adi Shamir, Faculty of Mathematics and Computer Science, Weizmann Institute of Science.

[7] Eli Biham, Orr Dunkelman, and Nathan Keller, New Results on Boomerang and Rectangle Attacks, proceedings of Fast Software Encryption 2002, Lecture Notes in Computer Science 2365, pp. 1–16, Springer 2002.

[8] A Single-Key Attack on 6-Round KASUMI,Teruo Saito, NEC Software Hokuriku, Ltd.1, Anyoji, Hakusan, Ishikawa 920-2141, Japantsaito@qh.jp.nec.com.

[9] Application of FSM Machine and S-Box in KASUMI Block Cipher to Improve Its Resistance Against Attack, Raja Muthalagu and Subeen Jain(Corresponding author: Raja Muthalagu), Department of Electrical and Electronic Engineering, Birla Institute of Technology and Science, Pilani, 345055 Dubai International Academic City, Dubai, United Arab Emirates, (Email: raja.m@dubai.bits-pilani.ac.in), (Received July 2, 2017; revised and accepted Oct. 22 & Nov. 5, 2017).

[10] Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A.,Dray, J., Vo, S.: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. NIST special publication 800-22, Revision 1a. National Institute of Standards and Technology (NIST), Gaithersburg,MD, USA (2001). https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

[11] Reducing the time required by KASUMI to generate output by modifying the FL and FI functions, Raja Muthalagu · Subeen Jain, Received: 1 November 2017 / Accepted: 12 March 2018, © Springer International Publishing AG, part of Springer Nature 2018.