A New Fast Stream Cipher : MAJE4

Sheena Mathew, K. Paulose Jacob

***Abstract*- A new fast stream cipher, MAJE4 is designed and developed with a variable key size of 128-bit or 256-bit. The randomness property of the stream cipher is analysed by using the statistical tests. The performance evaluation of the stream cipher is done in comparison with another fast stream cipher called JEROBOAM. The focus is to generate a long unpredictable key stream with better performance, which can be used for cryptographic applications.**

***Keywords*- cryptography, stream cipher, pseudo random number generator (PRNG)**

I. Introduction

Stream ciphers are an important class of symmetric encryption algorithms. Their basic design feature is the same as that for a One-Time-Pad cipher, which encrypts by XOR'ing the plain text with a random key. But for a One-Time-Pad Cipher it is required to have a key of the same size as the plain text, which makes it impractical for most applications. While the stream ciphers require only a short random key. This key is expanded into a pseudo-random key stream, which is then XORed with the plain text to generate the cipher text. Again the same key stream is used to decrypt by XORing with the cipher text to form the plain text as shown in Fig. 1. The security of the stream cipher rests in the key. So the random number generators occupy a central place in cryptographic designs owing to their property of picking numbers unpredictably and in using these numbers to choose cryptographic keys [6-13].

Here the design goal of stream cipher is to efficiently generate pseudo-random bits, which are identical to truly random bits.

PRNGs (Pseudo Random Number Generators) used for cryptographic purposes are required to be

1. of maximum period to accommodate the long length of the transmitted message.
2. fast to speed up the process.
3. difficult to analyse, since analysis could penetrate the cryptographic system.
4. capable of producing a good distribution of values.

Department of Computer Science
Cochin University of Science and Technology
Kochi, Kerala, India - 682 022
Sheenamathew@cusat.ac.in, kpj@cusat.ac.in

0-7803-9503-4/05/$20.00 ©2005 IEEE

PRNGs find extensive applications in computer simulation, numerical analysis, genetic algorithms, and automatic password generation.

The following PRNGs were already considered for study and implementation and extensive analysis were done to evaluate their performance and to select an appropriate PRNG.

1. Shift Register Based Generators
   a.Linear Shift Register[1]
   Linear Feedback Shift Register (LFSR)[1]
   b.Nonlinear Shift Register[2]
   Geffe Generator[2]
2. Arithmetic and Algebraic Operations Based Generators
   a. Linear Congruential Generators (LCGs)[3]
   b. X^2 mod N –Single precision and Multi precision[4]
3. A Fast Stream Cipher 'JEROBOAM'[5]

Among the above PRNGs, JEROBOAM was found to be a reliable generator compared to the other generators, because it passed all the five randomness tests considered for study for all the random streams produced and has not undergone attacks. Hence while designing the new stream cipher MAJE4, the important features of JEROBOAM were also considered. The cipher MAJE4 is designed to work efficiently on 32-bit processors.

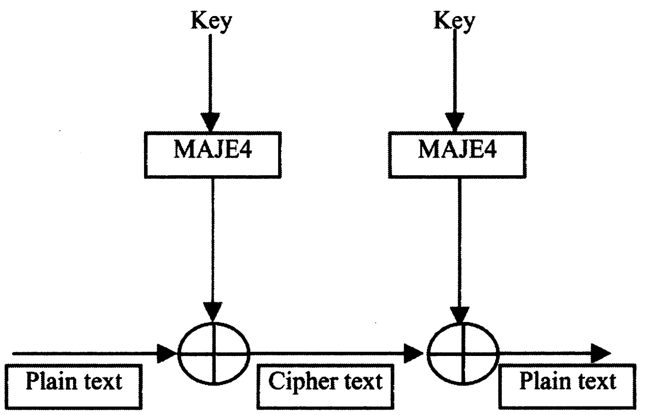

Fig. 1. Stream Cipher

The cipher MAJE4 is designed for using a key of size 128 or 256 bit. MAJE4 produces a pseudorandom stream, which can be used to XOR the plain text of any length. The aim of the work is to design a secure stream cipher, which is highly secure in software.

II. Design Considerations For MAJE4

1. The encryption sequence should have a large period. A pseudo random number generator uses a function that produces a deterministic stream of bits that eventually repeats itself. The longer the key, the longer it takes for a brute force attack and more difficult to do the cryptanalysis.
2. The key stream should approximate the properties of a true random stream as possible.
3. Suitable for hardware or software. Uses only primitive computational operations commonly found on microprocessors.
4. Simple and Fast. Use simple algorithm, which is easy to implement and eases the task of determining the strength of the algorithm.
5. Low memory requirement. To make it suitable for devices with restricted memory.
6. Mixed operators. The use of more than one arithmetic and / or Boolean operator complicates cryptanalysis. Use primitive operators like + and ^ since these operators do not commute and hence cryptanalysis becomes more difficult.
7. Variable number of rounds. An increase in the number of rounds increases cryptanalytic strength. Also it increases the encryption / decryption time. A variable number of rounds permit the user to make a compromise between security and execution speed.

III. Description Of MAJE4

The mathematical operators used are

1. Addition: Addition of words, denoted by +

2.Bitwise exclusive OR: This operation is denoted by ^.

3. Right shift operation: The right shift of word x right by y bits is denoted by x>> y.

All the above-mentioned design considerations were taken care while designing MAJE4 stream cipher. MAJE4 has a key length of 128 or 256-bit. Here the randomness property has been tested with the primary statistical tests like frequency test, serial test, runs test, poker test and auto correlation test. Since it uses only primitive computational operators like +, ^, >> etc, it is suitable for hardware and software implementations. MAJE4 uses an algorithm, which is easy to implement and fast also. The memory requirement for MAJE4 is less and hence it is suitable for devices having restricted memory. Here the nonlinearity is obtained by alternative usage of + and ^ operators, which complicates

A. Key setup

One can choose between a 128-bit key or 256-bit key.

256-bit key: The key is stored in eight 32 bit words $key_{[0]}$, $key_{[1]}$, $key_{[2]}$, $key_{[3]}$, $key_{[4]}$, $key_{[5]}$, $key_{[6]}$ and $key_{[7]}$.

128-bit key: The first four 32 bit words, ie. $key_{[0]}$, $key_{[1]}$, $key_{[2]}$ and $key_{[3]}$ are considered for storing the key.

B. Algorithm

Steps:

```
1. Assign the key length kl either as 128-bit or 256-bit.
2. if kl = 128 then
       kln=2, div=4
   else
       kln=3, div=8
3. if kl = 128 then consider two lsb's of $key_{[0]}$ and find the
   decimal equivalent of these two lsb's and store in the variable 'in'.
   else
   if kl = 256 then consider three lsb's of $Key_{[0]}$ and find
   the decimal equivalent of these three lsb's and store it in a variable 'in'.
4. ran =$key_{[0]}$ ^ $key_{[in]}$
5. if kl = 128 then consider two lsb's of ran and find the
   decimal equivalent of that and store in the variable 'in1'.
6. if kl = 256 then consider three lsb's of ran and find the
   decimal equivalent of that and store in the variable 'in1'.
7. check the 16th bit in ran,
   if it is 1 then
       newran=($key_{[in1]}$ + $key_{[in1+1 mod div]}$) ^ ($key_{[in1+2 mod div]}$ + $key_{[in1+3 mod div]}$)
   else
       newran=($key_{[in1]}$ ^ $key_{[in1+1 mod div]}$) + ($key_{[in1+2 mod div]}$ ^ $key_{[in1+3 mod div]}$)
8. The output 32-bit word is newran, which can be used to
   XOR with the corresponding word in the plain text.
9. Advance all the keys as
   $key_{[i]}$ = $key_{[i]}$ * $key_{[i]}$ + $key_{[i]}$ >>20
10. go to step3
```

IV. Randomness Tests

The analysis of MAJE4 is done mainly using the tests listed below [3]. These tests are commonly used for determining whether the binary sequences possess some specific characteristics that a truly random sequence is likely to exhibit.

1. Frequency Test.
2. Serial Test
3. Poker Test
4. Runs Test
5. Autocorrelation Test

Frequency Test (monobit test): To test whether the number of 0's and 1's in the sequences are approximately the same, as would be expected for a random sequence.

Serial test (2 bit test): To determine whether the number of 00,01,10 and 11 as subsequences of s are approximately the same, as would be expected for a random sequence.

Poker test: Let m be a +ve integer. Divide the sequence into n ÷ m non-overlapping parts of length m. To test whether the number of each sequence of length m are approximately the same, as expected for a random sequence.

Runs test: To determine whether the number of runs of various lengths in the sequence is as expected in the random sequence.

Auto-correlation test: To check whether correlation between the sequence and the shifted version of it is approximately 0 when the number of shifts is not divisible by the period as expected for a random sequence.

The frequency test is for uniformity and the other tests are for independence.

V. Results

Here Frequency, Serial, Poker and Runs tests are analysed using the Chi-square table and Autocorrelation test is analysed using Normal table, as specified for each randomness tests. The fast stream cipher MAJE4 successfully passes all the five statistical tests for every run. Tables 1,2 and 3 show the results of the specified tests.

Table I: Statistical Analysis Using Autocorrelation Test

Number of random numbers generated	Total number of bits produced	Statistical Analysis	
		128-bit key	256-bit key
		Autocorrelation test	Autocorrelation test
300	9600	2.0855	1.5007
500	16000	2.2158	1.8581
800	25600	2.0762	2.3439
1000	32000	2.4944	1.6045
2000	64000	1.6368	1.4902

Table II: Statistical Analysis Using Frequency, Serial, Poker And Runs Tests With 128 Bit Key

Number of random numbers generated	Total no. of bits produced	Statistical Analysis			
		128-bit key			
		Fre-quency test	Serial test	Poker test	Runs test
500	16000	1.3690	3.6252	294.27	24.00
1000	32000	1.5961	3.5682	566.25	28.43
1500	48000	1.4083	3.4354	538.08	19.97
4000	128000	0.0632	1.2458	2099.7	29.93
6000	192000	0.2475	1.8224	2022.3	30.12
8000	256000	0.4100	2.9614	4101.1	32.33
10000	320000	0.0903	3.7286	4159.0	30.25

Table III: Statistical Analysis Using Frequency, Serial, Poker And Runs Tests With 256 Bit Key

Number of random numbers generated	Total no. of bits produced	Statistical Analysis			
		256-bit key			
		Frequ-ency test	Serial test	Poker test	Runs test
500	16000	0.3610	4.2347	269.69	12.79
1000	32000	0.8820	5.5466	501.05	15.86
1500	48000	0.3100	5.9851	481.64	18.87
4000	128000	0.0031	1.6390	2044.4	27.17
6000	192000	0.0316	2.3485	2011.3	27.32
8000	256000	0.0082	2.3659	4109.3	21.40
10000	320000	0.0630	1.6504	4116.4	16.16

VI. Performance Evaluation

The summary of performance evaluation is presented here by comparing with JEROBOAM stream cipher and shown in Table 4.

A. Timing Analysis

From the timing analysis it can be noted that when we compare JEROBOAM 128-bit and MAJE4 128-bit, MAJE4 128-bit is almost 9 times faster as shown in Fig.2. Also the memory requirement is less in MAJE4. The evaluation is done using Pentium IV Processor, Linux Operating System and C compiler.

Table IV: Timing Analysis

PRNGs used	Key length	No. of random numbers generated	No. of random bits per each random number	Total no. of bits produced (speed Mbps)
JERO-BOAM	128-bit	26,80,000	16	40.89
MAJE4	128-bit	1,15,39,399	32	352.15
MAJE4	Variable 128-bit	58,34,000	32	178.03
MAJE4	Variable 256-bit	43,99,999	32	134.27

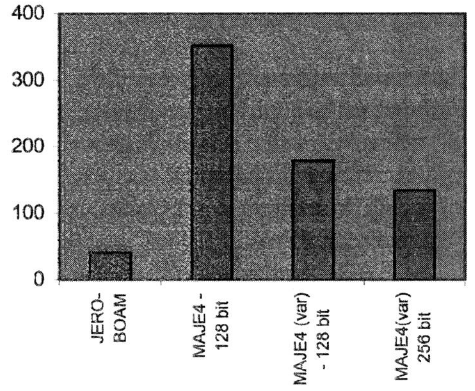

Fig. 2. Comparison of Number of Random Bits Produced per Second (in Mbps)

B. Memory Requirement

On comparing the memory required for executable files of JEROBOAM 128-bit and MAJE4 128-bit, MAJE4 was found consuming lesser space compared to JEROBOAM. The memory size required for optimised code for JEROBOAM is 6341 bytes, for MAJE4 128-bit is 5435 bytes and for MAJE4 128-bit or 256-bit is 5678 bytes.

VII. CONCLUSIONS

From the results of analysis and performance evaluation it can be concluded that MAJE4 is a reliable generator, which is much faster than JEROBOAM. All the five statistical tests are passed by this generator for all the random streams produced. Since the memory requirement is less, it can be used for devices with restricted memory for security purposes. Since it is faster and produce independent bits, this stream cipher can be used for applications that require encryption / decryption of a stream of data sent through the Internet.

REFERENCES

[1] Bruce Schneier, Applied Cryptography, 2nd Edition, John Wiley and Sons, 1996.

[2] Kencheng Zeng, Chung-Huang Yang, Dah-Yea Wei and T.R.N. Rao, "Pseudorandom Bit Generators in Stream-Cipher Cryptography", IEEE Computer, February 1991, pp 8-17

[3] D. E. Knuth, The Art of Computer Programming - Vol..2, Seminumerical Algorithms, Addison -Wesley, 1969.

[4] L. Blum, M. Blum and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator", SIAM Journal on Computing, Volume 15, Issue 2, May 1986, pp 364 - 383.

[5] Herve Chabanne and Emmanuel Michon, "Jeroboam", 5th International Workshop on Fast Software Encryption, Springer-Verlag, London, 1998, pp 49-59.

[6] Ritter T., "The Efficient Generation of Cryptographic Confusion Sequences", Cryptologia, Vol 15, No 2, 1991, pp 81-139.

[7] U.V. Vazirani and V.V. Vazirani, "Efficient and Secure Pseudo- Random Number Generation, Advances in Cryptology", CRYPTO 84, LNCS Vol.196, Springer – Verlag, 1985, pp 193-202.

[8] Mustak E. Yalcin, Johan A. K. Suykens and Joos Vandewalle, "True Random Bit Generation From a Double-Scroll Attractor", IEEE Transactions on Circuits and Systems, Vol. 51, No.7, July 2004, pp 1395-1404.

[9] J. Boyar, "Inferring Sequences Produced by Pseudo-Random Number Generators", Journal of ACM, Vol.36(1), Jan 1989, pp 129-141.

[10] T. Seigenthaler, "Decrypting a class of Stream Ciphers Using Cipher text Only" , IEEE Transactions on computer, Vol C-34, No.1, Jan 1985, pp 81-85.

[11] Park Stephen K. and Keith W. Miller, "Random Number Generators : Good ones are hard to find", Communications of the ACM, October 1988, pp.1192-1201.

[12] William Aiello, Sivaramakrishnan Rajagopalan and Ramarathnam Venkatesan, "Design of Practical and Provably Good Random Number Generators", SODA'95, ACM Jan 1995.pp 1-9.

[13] William Stallings, Cryptography and Network Security: Principles and Practices, Third Edition, Prentice Hall, 2003.