International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

New Secure and Advanced Algorithm for Stream Ciphers Extended RC4 and FPGA Implementation

Nikhil Sankar[1], **K. Kalpana**[2]

[1] PG Scholar, Hindusthan Institute of Technology, Coimbatore 32, India

[2]Assistant Professor, Department of ECE, Hindusthan Institute of Technology, Coimbatore-32 India

Abstract: ***RC4 is the most widely used stream ciphers uses in cryptography. A lot of modifications of RC4 cipher can be seen in open literature. This paper mainly analysis and describe the design issue of stream ciphers in Network security and proposes a variation in the algorithm which increases the efficiency of the algorithm. The performance evaluation of the stream cipher is done in comparison with present stream ciphers. The focus is to generate a long unpredictable key stream with better performance, which can be used for cryptographic applications.***

Keywords: cryptography, stream ciphers, RC4 modifications, PRGA, KSA, FPGA implementation

1. Introduction

To secure information communications over the network, different encryption algorithms have been used. The encryption algorithms are further categorized into two broad categories: Symmetric and Asymmetric. In symmetric algorithms, same key is used for both encryption and decryption. Asymmetric algorithms use different keys for encryption and decryption. RC4, AES, DES are examples for symmetric encryption algorithm and RSA is an example for asymmetric encryption algorithm 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 XOR-ing with the cipher text to form the plain text. 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.

RC4 is a stream cipher algorithm and operates on individual bits to secure the algorithm. The speed of the encryption and decryption is a very important aspect of security algorithms in working with applications. A slow cryptographic algorithm can slow the speed of an application and reduce its effectiveness.

RC4 are used in protocols like SSL, WEP, WPA, and applications like Skype, Remote Desktop and Microsoft Point-to-Point. There are many other applications which use RC4 as the encryption algorithm. It is used in hardware based encryption mechanisms as well. Due to its light weight it has become popular despite of various attacks on RC4 [8]. In open literature, there are a lot of articles, which describe various attacks on RC4 and a lot of them are theoretical [8, 10]. There are a lot of publications on hardware implementations of RC4 to enhance performances [9]. Many known attacks on RC4 that unveil some part of the secret internal state, are based on fixing some elements of the S-box with the values of *i* and *j* that give information about the outputs at certain rounds with probability one or very close to one. This leads in distinguishing attacks on the cipher and helps to obtain the secret internal state with probability that is notably larger than expected. So, the correlations between the internal state and the external state violate the ―randomness feature of a cipher, which leads to an unsecure communication.

The RC4 stream cipher is implemented in hardware by P. Kitsos, G. Kostopoulos, N. Sklavos ,and O. Koufopavlou VLSI Design Laboratory, Electrical and Computer Engineering Department, University of Patras, Patras, Greece . The same hardware implementation is fast and reliable as compared to software implementations and block ciphers algorithms.RC4 is used for encryption in the wired equivalent privacy (WEP) protocol (part of the IEEE 802.11b wireless LAN security standard), IEEE 802.11 i Lotus Notes, Apple computer's AOCE and Oracle secure SQL. The IEEE 802.11 i uses the Temporal Key Integrity Protocol (TKIP) and the Advanced Encryption Page Layout

2. Existing System: RC4

The RC4 algorithm which was initially proposed in 1987 uses a variable length key and its operations are byte oriented. It uses a deterministic algorithm to produce a random permutation. The RC4 algorithm can be divided into two phases: Key Scheduling Algorithm (KSA) and Pseudo Random Generation Algorithm (PRGA). KSA a makes use of the variable length key to initialize a 256 Bytes array S. This operation is known as the initialization of the S-block. The new key is then used to produce a random permutation of the initialized array S. This marks the end of the KSA phase. Once the array S has been initialized, the key is no longer used. PRGA phase now begins. It produces a random sequence of words from the permutation in S known as the key stream. During the decryption process, the key stream is then XORed with the plaintext to produce the ciphertext. During decryption, the ciphertext is XORed with the keystream to produce the plaintext

The logic for RC4 algorithm (KSA and PRGA) is shown below.

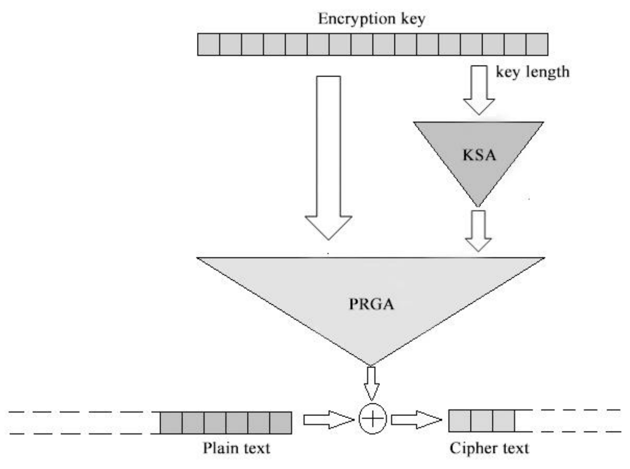
```
KSA
begin
for i=0 to 255
$S_i$=i;
$K_i$=K[i mod n];
end for
k=0;
for i=0 to 255
j=(j+$S_i$+$K_i$) mod 256
swap($S_i$,$S_j$)
end for
end
```

```
PRGA
begin
i=0;
j=0;
while(true)
i=(i+1) mod 256
j=(j+$S_i$) mod 256;
swap($S_i$,$S_j$);
t=($S_i$+$S_j$) mod 256
K=$S_t$;
end loop
end
```

Important design considerations for a stream cipher are given below:

1) The encryption sequence should have a large period. A pseudorandom number generator uses a function that produces a deterministic stream of bits that eventually repeats. The longer the period of repeat the more difficult it will be to do cryptanalysis.
2) The keystream should approximate the properties of a true random number stream as close as possible. For example, there should be an approximately equal number of 1s and 0s. If the keystream is treated as a stream of bytes, then all of the 256 possible byte values should appear approximately equally often. The more random appearing the keystream is, the more randomized the ciphertext is, making cryptanalysis more difficult
3) Note from Figure 1 that the output of the pseudorandom number generator is conditioned on the value of the input key. To guard against brute-force attacks, the key needs to be sufficiently long. The same considerations as apply for block ciphers are valid here. Thus, with current technology, a key length of at least 128 bits is desirable.

The basic working of the algorithm is shown below:



The steps for **RC4** encryption algorithm are:

1) Get the data to be encrypted and the selected key.
2) Create two string arrays.
3) Initiate one array with numbers from 0 to 255.
4) Fill the other array with the selected key.
5) Randomize the first array depending on the array of the key.
6) Randomize the first array within itself to generate the final key stream

ATTACKS ON RC4

Many cryptanalysis of RC4 emerged after the algorithm was made public in 1994. The cryptanalysis was divided into two main parts, analysis of the initialization of RC4 which focuses on the initialization of KSA and analysis of the key stream generation which focuses on the internal state and the round operation of PRGA. The most serious weakness of RC4 that the probability of a zero output byte at the second round is twice as large as expected. Fluhrer et al.[3,8] have shown that the RC4 could be attacked completely if we can know portion of the secret key. There is a Probabilistic correlation between the secret information and the public information

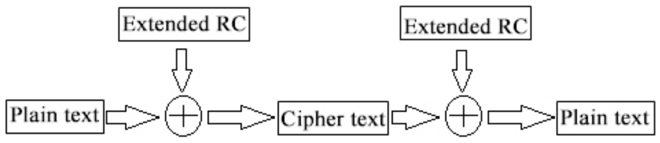
Crypto analysts discovered the secret key from the initial state table using biases in the first entries of the table. They created some equations by the initial state table. Thy guess some of the bytes of the secret key and they obtain the rest of the key by using these equations. In 2007, Klein[8] presented a statistical relation between any output byte and the value of SOl at the time of the output generation. Violeta Tomasevi and Slobodan Bojani [10] introduced an abstraction in form of general conditions for cryptanalytic managing of the information about the current state of the RC4 stream cipher. The general conditions based strategy is used to favor more promising values that should be assigned to unknown entries in the RC4 table. Crypto analysts have formally proved that only a known elements of the S-box along with two index-pointers cannot predict more than a output bytes in the next N rounds. They have also designed an efficient algorithm to deduce certain special RC4-states known as Non-fortuitous Predictive States.

3. Proposed System

The design considerations of **Extended RC4** are as follows:

1) Suitable for hardware or software. Uses only primitive computational operations commonly found on microprocessors.
2) Simple and Fast. Use simple algorithm, which is easy to implement and eases the task of determining the strength of the algorithm.
3) 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.
4) The encryption sequence should have a large period. The longer the key, the longer it takes for a brute force attack and more difficult to do the cryptanalysis.

5) Low memory requirement. To make it suitable for devices with restricted memory



The whole system operation is shown below:
The extended RC4 also uses the same architecture as the existing RC4, but KSA and PRGA structure have some variations.

PRNGs (Pseudo Random Number Generators) used here 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 analyze, since analysis could penetrate the cryptographic system.
4) Capable of producing a good distribution of values.

EXTENDED RC4 Algorithm logic is

```
while GeneratingOutput:
 i := i + 1
 a := S[i]
 j := j + a
 b := S[j]
 S[i] := b (Swap S[i] and S[j])
 S[j] := a
 c := S[i<<5 ⊕ j>>3] + S[j<<5 ⊕ i>>3]
 output (S[a+b] + S[c⊕0xAA]) ⊕ S[j+b]
endwhile
```

The above algorithm can be further modified to reduce time complexity where all arithmetic is performed using modulo 256 as follow:

```
while GeneratingOutput:
 i := i + w
 j := k + S[j + S[i]]
 k := k + i + S[j]
 swap values of S[i] and S[j]
 output S[j + S[i + S[z + k]]]
endwhile
```

Average encryption times were calculated after having obtained encryption times in several similar number of experiments for each plaintext data size. With the average encryption times, the throughputs were calculated. With these two results, performance of each algorithm can be analyzed.
Secrecy of ciphers according to the theories of Shannon. With this analysis, a basic idea on the security level of the algorithms has been obtained. Each algorithm was tested for data sizes ranging from 10KB to 100KB. Average secrecy values were calculated after having obtained secrecy values in several similar no. of experiments for each plaintext data size.
To compute the average time of an encryption, I simply surrounded my encryption loop (successive encryption of the same plaintext with the same key) with timers:

```
long t1 = System.nanoTime();
for (int i = 0; i < m; ++ i)
cipher.encrypt ();
cipher.print();
long t2 = System.nanoTime();
```

The time for one encryption is then easily computed by subtracting t1 to t2 and dividing the results by m (the number of iteration).
The average throughput can be calculated by the dividing the appropriate data size by the encryption time in seconds. It is obtains in KBps.

4. FPGA Implementation

The steps in FPGA implementation are as follows:

The architectural-design phase is surprisingly similar for microprocessor and FPGA. It’s not unusual to perform a "first cut" at programming in a pseudo-language that can be translated into and refined as a specific language, say assembly, C++, or JAVA. This architectural design was in java language then translate it to Verilog for an FPGA. Architectural issues could fill a book; therefore have to focus on development issues.

Editing: can use any editor to prepare fpga.v files.

Compiling of a program for the microprocessor combines the edited files and builds a logically correct sequence of bits that are used to control the sequencing of logical gates. These gates write data onto buses, into latches and registers, out ports, and across channels. The gates have fixed relationships designed to accomplish fixed functions. The assembly-language instructions represent these functions. Thus microprocessor compilers either produce assembly-language programs that are then assembled into bit patterns or directly produce the bits to drive the gates and fill the registers and the memories.

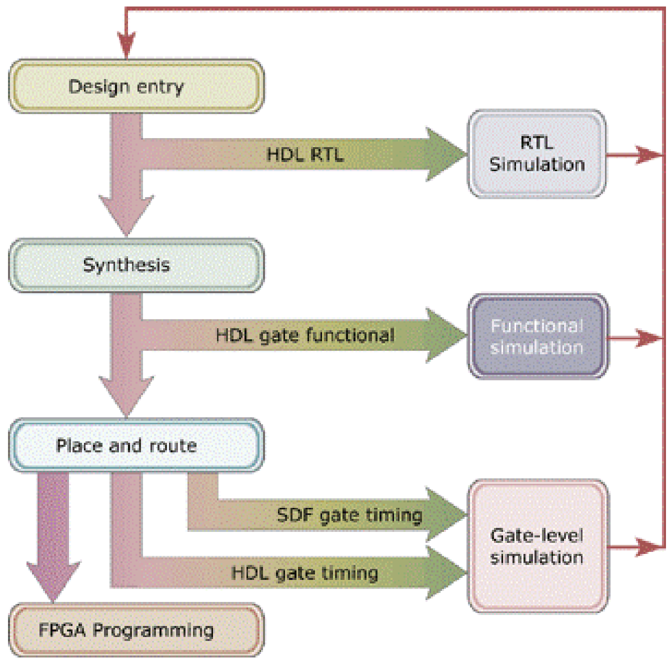
Linking: The bit-based outputs of the microprocessor compilation process typically don't directly control gates but must be connected to other bit patterns. This is true because most programs run under the control of an operating system and must be connected, or linked, to the operating system. In fact, the location in memory of the actual compiled bits is usually unknown and not determined until linking and loading is completed. Further, there may be programs existing in a library that must also be linked to the compiled program before a useful product exists.

Loading: Finally, just as embedded programs are often embedded in physical ROM, flash, or downloaded live, FPGA programs (compiled, synthesized, placed, and routed) must be embedded in the physical FPGAs. The actual programming file may be a .HEX or similar. Programmers typically download or burn the bits from these files into the hardware. If nonvolatile, this is a one-time proposition. If not, it's a download-at-power-up proposition. Many variations exist with FPGAs as with microprocessor-based embedded systems, but in the end, in a functioning microprocessor-based product, the bits compiled, linked, and loaded must "get into" the physical memory to control the gates of the processor, and in an FPGA-based functioning product, the bits compiled, synthesized, placed, and routed, must "get into" the FPGA, to implement the gates of the system.

Debugging programs: All experienced programmers know that complex programs, even subprograms, don't run correctly the first time. In embedded systems program development, usually uses debuggers, simulators, and emulators to fix this. These tools enable the user to step through the program execution and observe the effects on flags, register contents, memory locations, and so on, and to try to match what we expect at a given place and time with what we see, in the simulator or emulator

Testbenches in FPGA: Most FPGA systems are standalone systems connected to the real world, and functioning in interaction with the real world. Therefore a large part of debugging and test is concerned with simulating the real world to which the FPGA will attach. In reference to the way logic circuits were debugged historically, this is called the testbench and is considered an integral whole—in other words, it can be compiled as a whole. Simulation typically steps through the operation of the testbench, which stimulates the FPGA. The simulation process observes the transformations and translations of signals as they propagate through the FPGA from the input pins and provides responses that eventually reach an output pin.
Netlist: Although it's not really a part of the process, it's worthwhile to understand that the output of the FPGA design process is a *netlist* or list of nets or wires that connect gate outputs to other gate inputs

The FPGA programming process is as follows



5. Results

The extended RC4 algorithm is designed and then implemented in the FPGA. Overall results of average encryption time and throughput over data size can be illustrated by using the result table and graphs as follows:

Table 1: Average Encryption Time Vs Data Size

Data size(KB)	Average Encryption Time(µs)	
	RC4	Extended
10	3140	2630
20	6008.5	3015.5
30	7558.8	3782
40	8993	4380
50	10728.5	6300.4
60	11833	7800.5
70	13142.5	8056
80	14590.5	8210
90	16257	8412.6
100	17084	8700

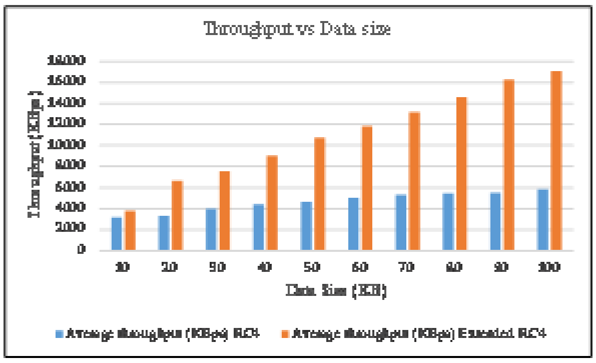


Figure 1: Average Encryption Time (in microseconds) over Data Size. Average was taken out of similar no. of experiments for each data size. Efficient RC4 cipher proposed in this paper shows the lesser encryption time and the Original RC4 cipher shows the highest encryption time almost for every data size from 10 KB to 100KB.

From the graph it is clear that the extended RC4 is far better than the RC4 algorithm in the encryption speed

Table 2: Average Throughput Vs Data Size

Data size(KB)	Average throughput (KBps)	
	RC4	Extended
10	3184.72	3802.22
20	3328.61	6633.49
30	3969.02	7558.78
40	4447.82	8993.17
50	4660.21	10729.13
60	5070.21	11833.89
70	5326.27	13142.45
80	5483.02	14590.99
90	5536.35	16256.198
100	5853.43	17084.11

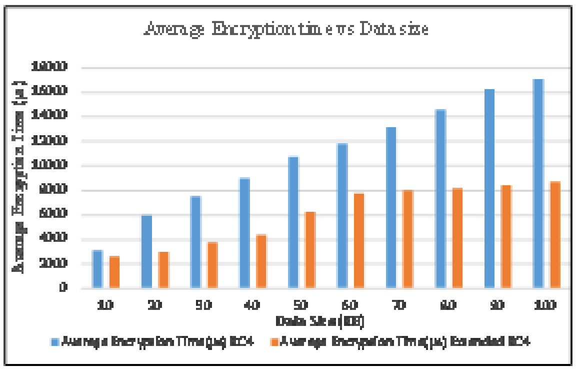


Figure 2: Average Throughput (in KBps) over Data Size. Throughput was calculated by dividing the appropriate data size by the encryption time in seconds. Thus KBps values

were obtained. The highest throughput was achieved by the Efficient RC4 proposed

Conclusion

Firstly, this paper introduced RC4 stream cipher and several attacks on it. Based on the weakness of the relations between the states of S-box in RC4, we present an extended RC4 in this paper. The new algorithm has destroyed the relations. The new algorithm enhanced the security of RC4, and it is faster than RC4. However, whether the improved RC4 has other loopholes remains to be tested.

Higher values of secrecy are shown by the new stream cipher, which means the randomness of the cipher is higher than that of other, which is a feature of a good cipher.

Reference

[1] J. Xie, X. Pan, ―An Improved RC4 Stream Cipher‖, 2010 International Conference on Computer Application and System Modeling, (ICCASM 2010), pp. (V7) 156-159, 2010

[2] R. A. Rueppel. *Analysis and Design of Stream Ciphers*. Springer- Verlag, 1986

[3] Rivest, R., *The RC4 Encryption Algorithm. RSA Data Security*. Inc., March, 1992.

[4] RC4 algorithm for WLAN WEP protocol" IEEE Transactions on control and decision conference , 2010 .

[5] Yao Yao Jiang Chong , Wang Xingwei " Enhancing RC4 algorithm for WLAN WEP Protocol" 978-1-4244-5182-1/10/$26.00 © 2010 IEEE

[6] Stallings et al,"Computer security: principles and practice", Upper Saddle River, N.J, Prentice Hall,2008.

[7] Rick Wash, Lecture Notes on Stream Ciphers & RC4

[8] A. Klein, ―Different attacks on the RC4 stream cipher‖, Lecture Notes, Department of Pure Mathematics and Computer Algebra, Ghent University, Belgium.

[9] S. S. Gupta, A. Chattopadhyay, K.Sinha, S. Maitra, B. Sinha, ―High Performance Hardware Implementation for RC4 Stream Cipher‖, *Computers, IEEE Transactions on*, vol.62, no.4, pp.730,743, April 2013 doi: 10.1109/TC.2012.19

[10] G. Paul, S. Maitra, ―RC4 State In formation at Any Stage Reveals the Secret Key‖, Proceedings of Selected Areas in Cryptography. LNCS, vol. 4876, no., pp. 260,377, Springer, Heidelberg 2007.

[11] Mantin, A. Shamir, "A Practical Attackon Broadcast RC4,"FastSoftware Encryption 2001 (M.Matsui,ed.), vol.2355 of LNCS, pp.152-164, Springer-Verlag, 2001

[12] Hardware Implementation of Modified RC4 Stream Cipher Using FPGA Jaya Dofe1, Manish Patil2 Dept. of Electronics, PG Student Maharashtra Academy of Engineering, Alandi, Pune University of Pune

[13] Hardware Implementation Of The Rc4 Stream Cipherp. Kitsos, G. Kostopoulos, N. Sklavos, and O. Koufopavlou,VLSI Design Laboratory,Electrical and Computer Engineering Department,University of Patras, Patras, Greece

[14] Hardware Implementation of RC4A Stream Cipher International Journal of Cryptology Research 1(2): 225-233 (2009), Abdullah Al Noman, Roslina Mohd. Sidek and Abdul Rahman Ramli

[15] Hardware Implementation of High Speed RC4 Algorithm in FPGA,International Journal of Computer Applications (0975 – 8887) Volume 83 – No4, December 2013

Author Profile

Mr. Nikhil Sankar, received his b. tech degree in electronic and communication engineering from Malabar College of Engineering and technology affiliated to Calicut University, Kerala. Now he is pursuing M.E in VLSI Design from Hindusthan institute of technology affiliated to Anna University, Chennai. His area of interests is VLSI Design and cryptography.

Mrs. K. Kalpana received her M.E degree in VLSI Design from Karpagam University,Coimbatore, TamilNadu and B.E degree in electronics and communication Engineering From Periyar Maniay-ammai College of Engineering and Technology for Women, Affiliated to Bharathidhasan University, Thanjavur, Tamilnadu. She has more than a decade of teaching experience in various Engineering colleges in Tamil Nadu. Currently she is working as an Assistant Professor in the department of ECE at Hindusthan Institute of Technology, Coimbatore. Her area of interests is image processing, Signal Processing and VLSI Design. She published around 5 papers in refereed conferences and journals.