Analysis of a Modified RC4 Algorithm

T.D.B Weerasinghe
MSc.Eng, BSc.Eng (Hons),
MIEEE, AMIE (SL)
Software Engineer
IFS R&D International, 363,
Udugama, Kandy, Sri Lanka

ABSTRACT

In this paper, analysis of a simply modified RC4 algorithm is presented. RC4 is the most widely used stream cipher and it is not considered as a cipher that is strong in security. Many alternatives have been proposed to improve RC4 key generation and pseudo random number generation but the thoughts behind this work is to try out a simple modification of RC4's PRGA, where we can mention like this:

Output = M **XOR** GeneratedKey **XOR j**

After having done the modification the modified algorithm is tested for its secrecy and performance and analyzed over the variable key length with respect to those of the original RC4. The results show that the modified algorithm is better than the original RC4 in the aspects of secrecy and performance.

General Terms

Symmetric Key Algorithms, Stream Cipher, RC4 KSG, RC4 PRGA

Keywords

Secrecy of RC4, Modified RC4

1. INTRODUCTION

RC4 is the most widely used stream cipher. In open literature there are a lot of modifications done in-order to improve the security and performance level of the particular algorithm. In this research the focus was to slightly alter the Output generation by adding one parameter into the XOR operation. The initial idea behind this change was aroused by the article published online by Likai Liu [7]. The intension behind this concept was to check the secrecy level and performance by doing the particular change and observe the results. Since Shannon's theories of secrecy of ciphers are not used regularly in the area in Cryptography (in open literature); the idea was to select them as the criterion for secrecy measurement. Performance is analyzed by measuring the encryption time of each cipher. Both secrecy and performance are analyzed over the variable key lengths which varied from 64 bits to 1856 bits. (Key lengths of RC4 can vary from 40 bits to 2048 bits). All the algorithms were written in Java as well as the time calculation (in microseconds) and secrecy measurements are also done using two separate Java programs. To generate a random alpha numeric character key, random character generation in Java is used!

2. RC4 ALGORITHM

RC4 is the widely used stream cipher. In this section, the original RC4 is described in a nutshell. The following description is illustrated from a research work done by Yassir Nawaz et. al. [1]

The RC4 algorithm has of two major parts: The key scheduling algorithm (KSA) and the pseudo-random generation algorithm (PRGA).

l - Length of the initial key in bytes
N - Size of the array *S* or the S-box in words.

Normally RC4 is used with a $n = 8$ and array size $N = 2^8$.

In the first phase of RC4 operation an identity permutation (0, 1..., *N*-1) is loaded in the array S. A secret key *K* (initial key) is then used to initialize *S* to a random permutation by shuffling the words in *S*. During the second phase, PRGA produces random words from the permutation in *S*.

An iteration of the PRGA loop produces one output word that constructs the running key stream (generated key stream). The keystream is bit-wise XORed with the plaintext to obtain the ciphertext. [1]

```
for i = 0 to (N-1)
   S[i] = i;
j = 0;
for i = 0 to (N-1)
   j = (j + S[i] + K[i mod l]) mod N;
   swap (S[i], S[j])
i = 0, j =0;
Output Generation Loop:
   i = (i+1) mod N;
   j = (j+S[i]) mod N;
   swap (S[i], S[j]);
   Output = S[(S[i] + S[j]) mod N];
```

The general structures of KSA and PRGA are shown in the above figrue. The original RC4 can be illustrated as follows because n = 8 and $N = 2^8$ (256)

KSA:

```
for i = 0 to 255
   S[i] = i;
j=0
for i = 0 to 255
```

```
  j = (j+S[i]+K[i mod l]) mod 256;
  swap S[i] and S[j];
```

PRGA :

```
i = 0, j=0;
for x = 0 to L-1
  i = (i+1) mod 256;
  j = (j+S[i]) mod 256;
  swap S[i] and S[j];
  GeneratedKey = S[ (S[i] + S[j]) mod 256] ;
  Output = M[x] XOR GeneratedKey;
```

Where 'M' is the plain text message and 'L' is its length. [4]

3. MODIFIED RC4

Here is the pseudo code of the modified RC4:

KSA:

```
for i = 0 to 255
  S[i] = i;
j=0
for i = 0 to 255
  j = (j+S[i]+K[i mod l]) mod 256;
  swap S[i] and S[j];
```

PRGA:

```
i = 0, j=0;
for x = 0 to L-1
  i = (i+1) mod 256;
  j = (j+S[i]) mod 256;
  swap S[i] and S[j];
  GeneratedKey = S[ (S[i] + S[j]) mod 256] ;
  Output = M[x] XOR GeneratedKey XOR j;
```

Where 'M' is the plaintext message and 'L' is its length. [4]

4. SECRECY OF CIPHERS

4.1 Definitions related to secrecy

Entropy:

Entropy of a message X, called H(X), is the minimum number of bits needed to encode all possible occurrences (meanings) of the message, assuming all messages are equally likely. [5]

- Entropy of a given message X is defined by the weighted average:

$$H(X) = -\sum_{1}^{n} P(x_i) \log P(x_i)$$

Uncertainty:

Uncertainty of a message is the number of plaintext bits that must be recovered when the message is scrambled in cipher text in order to learn the plaintext. The uncertainty of a message is measured by its entropy. [5] Higher the number of bits, higher the uncertainty.

Equivocation:

Equivocation is the uncertainty of a message that can be reduced by given additional information. [5]

- Equivocation is the conditional entropy of X given Y:

$$H_Y(X) = \sum \{X,Y\} P(X,Y) \log_2 [P_Y(X)]$$
$$H_Y(X) = \sum \{Y\} P(Y) \sum \{X\} P_Y(X) \log_2 [P_Y(X)]$$

Secrecy of ciphers:

Secrecy of a cipher is calculated in terms of the key equivocation $H_c(K)$ of a key K for a given cipher text C; that is the amount of uncertainty in K given C: [5]

$$H_c(K) = \sum \{C\} P(C) \sum \{K\} P_c(K) \log_2 [P_c(K)]$$

Note: This is the equation used in the secrecy calculation in this research and it was used in some of my previous work [2], [3] and all the above definitions were derived from theories of Shannon related to entropy and secrecy. Claude Elwood Shannon [April 30, 1916 – February 24, 2001] is called the Father of Information Theory.

All the above equations/definitions are illustrated from the lecture notes of Dr.Issa Traore of the University of Victoria, British Columbia, Canada.

URL: www.ece.uvic.ca/~itraore/elec567-04/notes/elec6704-6-2.pdf

4.2 Calculation of the secrecy of ciphers

How the calculation is done in the program:

- Consider the highlighted part first: That is the entropy of K given the relevant cipher. (This cipher has come due to this key)
  - Calculate how often each key byte has appeared in the key.
  - And then calculate the probability of each byte appears (given the cipher) in the key and get the summation of Pc(K) * $\log_2 P_c(K)$.
- Then consider the other part: Calculating P(C) and the summation.
  - Calculate how often each cipher byte has appeared in the cipher text.
  - And then calculate the probability of each byte appears in the key and get the summation (for all possibilities of the cipher bytes). This cipher is related to the above key; i.e. this cipher is obtained by encrypting plain text with the above key. Then get the multiplication of the highlighted part and

P(C) is calculated and finally the summation of all possibilities is calculated.

5. RESEARCH METHOD

5.1 Method for analyzing secrecy

This is done in two categories.

1. With a variable key length. (then the input text is fixed; considered as a password)

2. With a variable data size. (then the key size is kept fixed – 128 bits)

Category 1:

Secrecy values of ciphertexts related to different key lengths ranging from 64 bits to 1856 bits are obtained and the average value is calculated. For each key length, to normalize the results, experiment is carried-out for 100 random keys. For example, if you consider a 64 bit key, then required values are taken for 100 different 64 bit keys. Here, the input is a constant/fixed value as the variable is the key length. After calculations the comparison of the average secrecy values of both RC4 and modified RC4 ciphertexts is possible, over the length of the random key.

Used input: #THARINDU_WEERASINGHE#

Thus, the secrecy analysis is done in the first category!

Category 2:

- In this category the key length is fixed. (128 bits) Data size has been varied from 10 KB to 100KB. Here, at each instance the input is given as a text file containing alpha numeric values.

5.1.1 *Java method to calculate secrecy:* [6]

```
public static double calculateSecrecy(byte[]
key, byte[] cipher, int start)

{

        double entropy = 0;

        double secrecy = 0;

        final int[] countedKey =
countByteDistribution(key, start, key.length-
1);

         final int[] countedCipher =
countByteDistribution(cipher, start,
cipher.length-1);

for (int i=0;i<256;i++)

{

        final double p_k = 1.0 * countedKey[i]
/ key.length;

        final double p_c = 1.0 *
countedCipher[i] / cipher.length;

        if (p_k > 0)

        {

                entropy += p_k * log2(p_k);

                secrecy += -p_c * entropy;

        }

}

return secrecy;

}
```

5.2 Method for analyzing performance

This is also done in two categories.

1. With a variable key length. (then the input text is fixed; considered as a password)

2. With a variable data size. (then the key size is kept fixed – 128 bits)

Category 1:

The encryption times (in microseconds) for different key lengths (64 bits to 1856 bits) are measured and the normalization is done similarly as in the secrecy analysis. By comparing the average encryption times of both RC4 and modified RC4 over the length of the random key, the performance analysis is done!

Category 2:

- In this category the key length is fixed. (128 bits) Data size has been varied from 10 KB to 100KB. Here, at each instance the input is given as a text file containing alpha numeric values similarly as in the secrecy analysis. Encryption time is measured under the above normalization mechanism.

5.3 Method of generating the initial key

With the help of `java.security.SecureRandom` and `java.math.BigInteger`, streams of random alphanumeric characters are generated.

Example code:

```
new BigInteger(40, random).toString(32);
```

Need to change the first parameter of the `BigInteger` according to the required key length in bytes. E.g. If you need a random key having a length of 40bytes then you need to input 200 there. See below for the example:

```
new  BigInteger(200,  random).toString(32);
```

5.4 Method of analyzing the overall results

The idea behind the research was to modify the existing RC4 algorithm in order to check how it responds towards the secrecy of the cipher generated as well as the overall performance of the algorithm. Since the secrecy calculation is hardly used to evaluate security level of a cipher in open literature, the focus was to check the secrecy of the generated cipher text in order to evaluate the modified algorithm over the original one! The variation of the secrecy over the different key sizes is studied to give the result of the research with respect to the secrecy. Initial thought (prior to the outcome) was the secrecy level of the modified cipher should be higher than that of the original cipher because the modification leads to an additional operation. The average secrecy is calculated by running the

Performance is measured by calculating the average encryption time. All the tests were carried out in a machine which has the following configuration:

- Intel® Core™ i3 CPU, M370 @ 2.40 GHz
- 1.86 GB usable RAM
- MS Windows 7 Home Basic (32 bit)

6. RESULTS AND ANALYSIS

Table 1. Average Secrecy Value Vs Key Length

Key Length/Bits	Average Secrecy of RC4	Average Secrecy of modified RC4
64	0.032519153	0.021913237
320	0.110192092	0.153905248
576	0.180718271	0.190528341
832	0.15384488	0.198225548
1088	0.148745008	0.204488399
1344	0.178802549	0.176788808
1600	0.1718984	0.219518034
1856	0.191389199	0.22001954

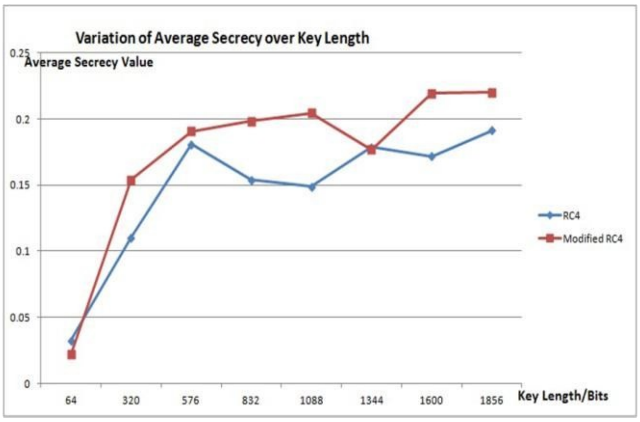

Fig 1: Average Secrecy Value Vs Key Length

Table 2. Average Encryption Time Vs Key Length

Key Length/Bits	Average Encryption Time of RC4/ µs	Average Encryption Time of modified RC4/ µs
64	101.4	97.3
320	84.1	74.2
576	85.7	76.5
832	88.2	78.7
1088	90.8	76.1
1344	91.1	77.6
1600	83.1	76.3
1856	85.1	80.3

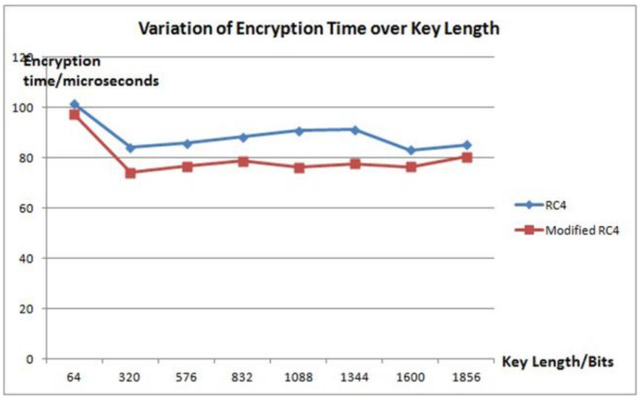

Fig. 2: Average Encryption Time Vs Key Length

Table 3. Average Secrecy Value Vs Data Size

Data Size/KB	Average Secrecy of RC4	Average Secrecy of modified RC4
10	0.2032	0.2021
20	0.20658	0.20756
30	0.2016	0.20127
40	0.20759	0.2125
50	0.20879	0.2145
60	0.20333	0.20656
70	0.21173	0.21416
80	0.20387	0.2033
90	0.20134	0.20418
100	0.2029	0.20557

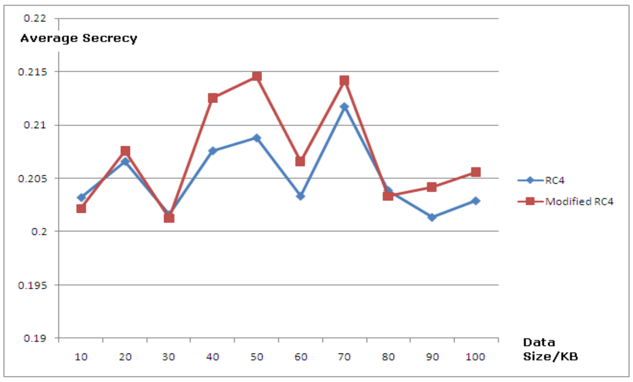

Fig. 3: Average Secrecy Value Vs Data Size

Table 4. Average Encryption Time Vs Data Size

Data Size/KB	Average Encryption Time of RC4/ µs	Average Encryption Time of modified RC4/ µs
10	2272	2146
20	4791	4059
30	6600	6393
40	7417	7173
50	8191	8227
60	9959	9351
70	9717	9616
80	10507	10389
90	11490	11777
100	12153	11997

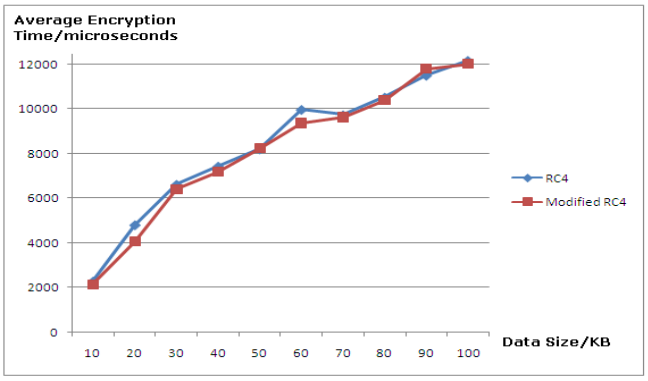

Fig. 4: Average Encryption Time Vs Data Size

7. CONCLUSIONS AND FUTURE WORK

Conclusion with respect to password type inputs:

As shown by the Fig 1, modified RC4 has better average secrecy than that of the original RC4. Both curves does not have smooth variation because of the limited number of samples, 100 to be précised (sample keys) taken for the experiment. But if we can test this for larger number of sample then the curve will have a smooth variation. If this is further explained if we take one instance of the experiment: consider the initial key length is 40 bits then we do have 2^40 number of possible keys. If we are to take the average secrecy of the instance where we have the key length of 40 bits then we need to get the secrecy values for all 2^40 keys which is unrealistic. So, in this research 100 sample keys are considered. (For measuring both secrecy and performance)

As far as the performance is concerned, the modified RC4 has the upper hand of the original RC4. By looking at the graph shown in the Fig 2, it is proved that the modification has lead to decrease the encryption time; hence the higher performance is achieved.

Conclusion with respect to the input messages that have larger data sizes:

As depicted in Fig. 3 the secrecy of the modified RC4 against the data size is more often than not, higher than that of original RC4. But again the measurements and calculations are done for 100 different keys (note: in this experiment the key size is taken as the most common size used 128 bits) and the curve could have been smoother if the numbers of samples are higher. (E.g. 10000 samples)

When it comes to the performance, it is obvious that the ‘modification’ has improved the performance (refer: Fig.4)

General conclusion:

Thus, a conclusion can be made upon the evident results, as the simple modification has done an enormous improvement of the RC4 (regardless of the limited number of samples taken – even for the limited number of keys modified RC4 gave good results, thus it is clearly obvious that if more samples were taken the results could have been much better. So, the simple modification in made RC4 to give better secrecy and performance simultaneously.

Suggested future work:

Same known plaintext attack for the modified RC4 and original RC4 and evaluate the tolerance levels.

REFERENCES

[1] Yassir Nawaz and Kishan Chand Gupta and Guang Gong. 2005. A 32-bit RC4-like Keystream Generator. In Cryptology ePrint Archive: Report 2005/175.

[2] T.D.B Weerasinghe. 2012. Analysis of a Hybrid Cipher Algorithm for Data Encryption. In IFRSA's International Journal of Computing, VOL.2 No.2, 397-401

[3]T.D.B Weerasinghe. 2012. Secrecy and Performance Analysis of Symmetric Key Encryption Algorithms. In IAES International Journal of Information and Network Security, VOL.1 No.2, 77-87

[4] Allam Mousa and Ahmad Hamad. 2006. Evaluation of the RC4 Algorithm for Data Encryption. In International Journal of Computer Science and Applications, VOL.3, No.2, 44-56

[5] Lecture notes of Dr.Issa Traore of the University of Victoria, British Columbia, Canada, related to secrecy of ciphers. URL: www.ece.uvic.ca/~itraore/elec567-04/notes/elec6704-6-2.pdf

[6] http://web17.webbpro.de/index.php?page=entropy

[7] http://lifecs.likai.org/2011/12/evaluating-rc4-as-pseudo-random-number.html