*International Journal of Advances in Computer Networks and Its Security– IJCNS*
*Volume 5: Issue 1 [ISSN : 2250-3757]*
*Publication Date : 30 April, 2015*

# Encryptor resulted from beating the 7z for incompressible files

[ João Silvestre – Coimbra, Portugal]

***Abstract*— The generation of incompressible files to 7z led to a new encryption algorithm. In this paper is presented the new encryption algorithm and the results of testing the new encryption technique, named Encryptor. The new algorithm, is a private algorithm, with only 64 bits in the state variable, but can be extended to other lengths. The Encryptor is protected from password attacks by a very long hash algorithm.**

**The inferred characteristics of the algorithm are of infinite length sequence, and therefore can not be attacked by the sequence. The characteristics of sequence are Gold code. Hitting the mark is only a result that can decrypt, everything else is noise. The algorithm is not a maximum length sequence, but a random length sequence. For example a sequence with four bytes repeats almost at four Gigabytes of data. The statistics of data at various lengths is flat.**

**The brute force attack can take $2^{64}$ executions of the Encryptor that is a sequential algorithm and takes to long to attack. The only way to attack the Encryptor is the DNA algorithm but has the problem of processing 16Exabytes.**

**The implementation of this new algorithm is very fast and can be suitable for hardware implementations for its simplicity.**

***Keywords*— encryption, Gold code**

## I. Introduction

The generation of pseudo random sequences (PRN) uses characteristic polynomials to define the points of sequence feedback. The polynomials typically chosen, generate maximum length sequences 2n -1, where n is the number of bits in the polynomial of the sequence. The random number generator of the C programming language was used to try to generate incompressible files, however 7z could build a dictionary and compress the file very significantly. In another attempt we used a characteristic polynomial that generates maximum length sequences to generate a pseudo-random number file, again 7z could build a dictionary and compress the file very significantly. Then I consulted my notes on Analog and Digital Communication Systems in the theme of PRN sequences generation and the generation of Gold codes. The result was the encryption of the random number generator of the C programming language, generating two sequences and making the XOR of the two to try to generate a Gold code. This result was then divided by a cousin number to generate a file of pseudo random numbers, so was beaten 7z that could not compress the file.

## II. Detailed Encryptor study

The encryption algorithm resulting from this study is presented below. A more detailed study of the algorithm robustness for presenting this work, indicates that the sequence was estimated to be $2^{64}$, actually has a lower strength. By studying the C random number generator, a sequence of two random numbers repeated in $2^{31}$, however checking the real sequence and with different seed, it is found that repeats after 1Giga, or 30 bits. The PRN sequence generator has two state variables, a 32-bit variable and another 32-bit variable of the random number generator that corresponds to 30 bits, resulting in a sequence that is estimated to be between $2^{62}$ and $2^{64}$. Figure 1 shows the statistics of the length of the sequence of two random numbers.

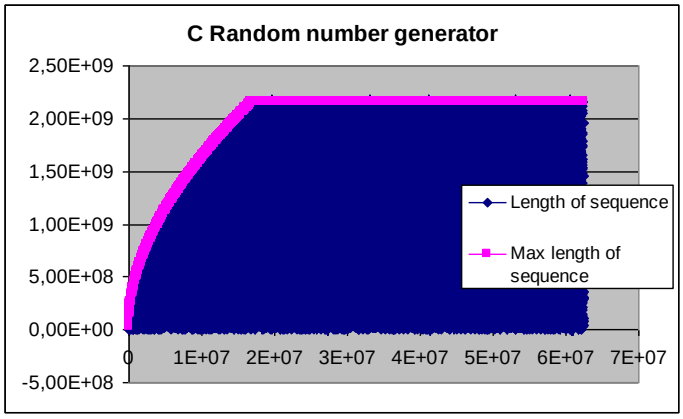


Figure 1. – Sequence length of two random numbers (sampled).

The C code for the encryption is the following:

```
for (i=0;i<bytes;i++)
      {
       key = ((rand()<<16)^rand()^key^keyMSB);
       encrypted_buffer[i] = buffer_data[i]^(key%255);
       keyMSB = key>>16;
      }
```

Authors Name/s per Affiliation *(Author)*
line 1 ISEC / Instituto Politécnico de Coimbra
line 2: Portugal

Of course it is easy to decrypt if we know the seed of the random number generator. The first step is to seed the random number generator based on password keys. In the second step we need to hash the password keys before start to encrypt. There is a initialization process of the state variable by the password keys, and the next step a very long random key hash.

```
length = key;
for ((register unsigned int)i=0;i<(register unsigned int)length;i++)
        {
          (register unsigned int)key = ((rand()<<16)^rand()^key^keyMSB);
          (register unsigned int)keyMSB = key>>16;
        }
```

The proposed algorithm corresponds to a multiplication (XOR) of two random sequences to form a Gold code. The feedback points do not correspond to a particular polynomial as is classically used, but random feedback points. As a result the sequence does not have maximum length but have random length. It is therefore a completely deterministic random sequence, which sequence is known only by the sender and by the receiver.

What is the hidden message in the sequence of numbers in PI? There will be something hidden there?

The Encryptor is just noise, there will be some hidden message? Sure there are, but to demodulate is necessary to know the sequence of the noise and only the sender and the receiver is in the know.

For a good encryption, the state variable of the sequence was divided by a prime number and the rest of the division used to make the XOR byte by byte with the message.

## A. *Unraveling the secrets of Encryptor.*

Studying the Encryptor sequence it appears that the statistical sequence of bytes is flat as shown in Figure 2, as well as the statistics of the two bytes sequence (Figure 3), lacking only the multiple N*256-1 due to the rest of the division by 255.

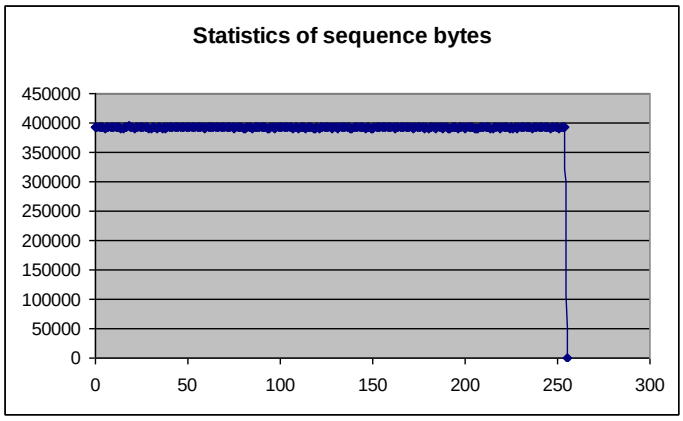


Figure 2. – Flat statistic of sequence bytes.

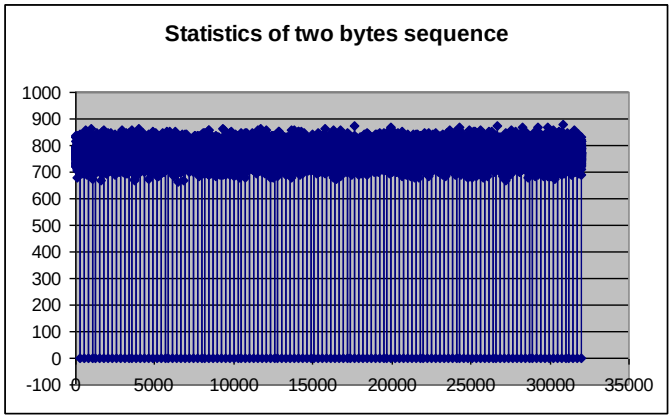


Figure 3. – Flat statistic of two bytes sequence.

The statistic of the three bytes sequence, shown in Figure 4, that shows the statistic is apparently flat. In the graph were used four gigabytes of data, but would need to be much more to get good statistics. In the graph arises suspicion that the sequences have not equal probability and the following study of length of 4 bytes sequences shows that are not random but pseudo-random numbers. The zeros of the graph are the multiple N*256-1 due to the rest of the division by 255.

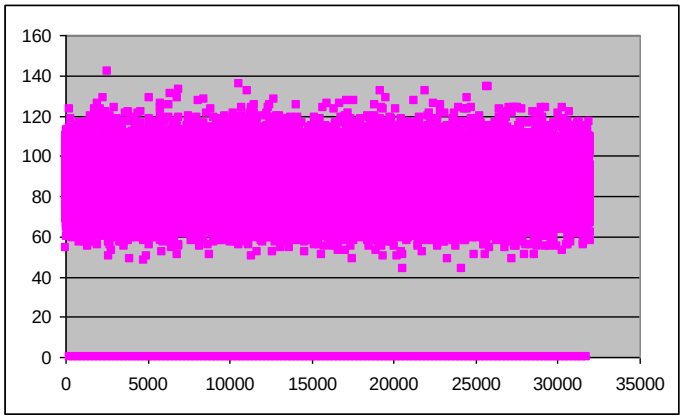


Figure 4. – Flat statistic of three bytes sequence.

Although the statistical of the bytes being flat (equal probability) and two-byte statistical also be flat, lacking only the multiple N*256-1, it is verified that the sequence length of 5 bytes is equal to the sequence length of 4 bytes, are not found all sequences, in 4,29E9 sequences of 4 bytes are only present 4,21E6 sequences that repeat after 1,07E9 bytes length ($2^{30}$). Then a problem is identified, the Encryptor has only 30 bits, each random number is only 15 bits and are used two random numbers means that the algorithm has less 2 bits. Although the two 32 bits state variables, it was thought that the result would be a complexity of 64 bits, but the algorithm generates a sequence of fixed length of only 30 bits ($2^{30}$).

What saves the algorithm is the initialization of the state variable and random number generator with the password keys. This process generates a different 30 bits sequence (One Gigabyte sequence) for each different password, tested and proven. Thus by statistical analysis of the 4 bytes sequences, are added over 10 bits by initialization with password keys. The initial hash algorithm added indeterminacy in positioning of sequence. It is estimated that the robustness of Encryptor is between 40 bits and 64 bits contrary to what was thought to be between 62 and 64 bits. Analyzing the sequences for various initializations it is found that different and interferents sequences are generated, i.e. different sequences have some equal patterns of 4 bytes, this is due to the two state variables of the algorithm the 32 bits state variable and the random number generator state variable. So is estimated to be more than 10 bits of complexity added due to initialization. The probability is that the algorithm can generate four Giga sequences of one Giga bytes, meaning 62 bits of complexity, although the typical password reduce this complexity by 4 or 60 bits, but does not mean that someone can use a more complex password or more 2 bits.

Before this detailed study by earlier work it was thought that the length of the sequence was random as shown in Figure 5. The 3 bytes sequence repeats after a maximum of 426E6 bytes.

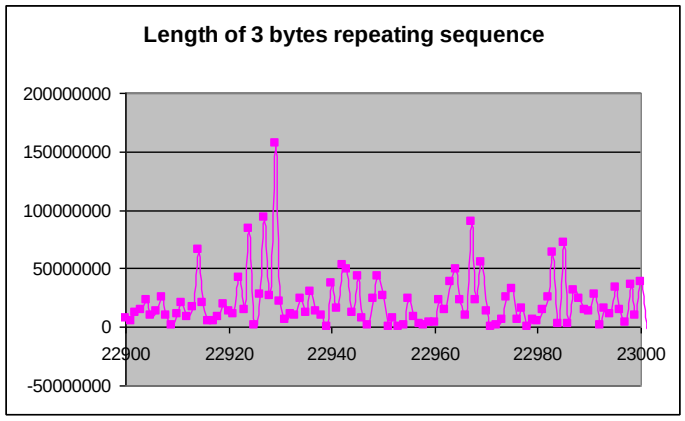


Figure 5. – Length of 3 bytes repeating sequence.

In detailing the study it was found that a sequence of four bytes is repeated after one gigabytes, as well as the sequence of five bytes, etc.

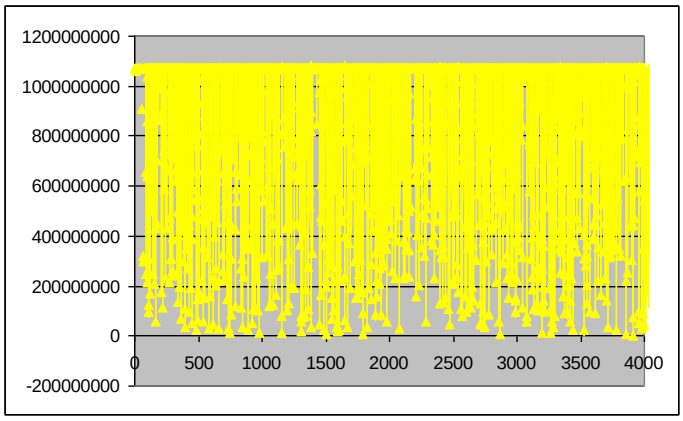


Figure 6. – Length of 4 bytes repeating sequence.

Figure 6 shows the sequence of repeating the 4 bytes sequence length and the random sequence length detail of repeating the 4 bytes sequence length. The graph shows the data of the sequence length after the first gigabyte. As can be seen four bytes are not enough to identify a possible start of the sequence, however the study done with 5 bytes (to detect sequences of one Tera failed got a Giga) showing that the 5 bytes sequence has not random length, but has a fixed length of one Gigabytes.

Finally a spectrogram was made and it was concluded that the Encryptor is a good white noise generator, the messages to encrypt will suffer a spread spectrum to be confused with white noise.

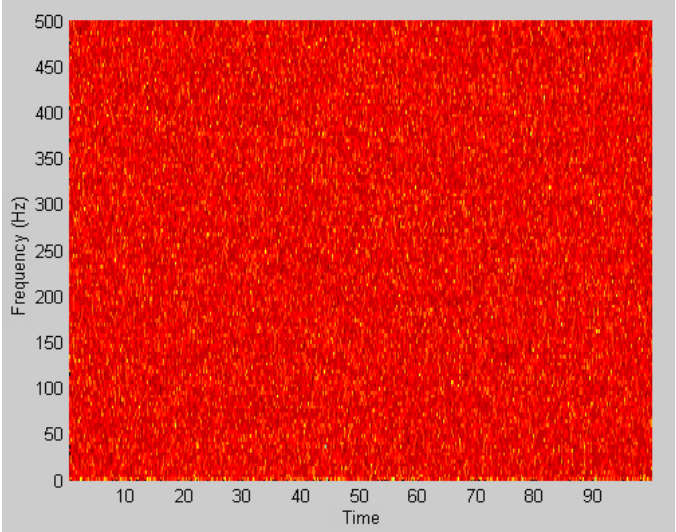


Figure 7. – Spectrogram of a Encrypted message.

Figure 8 shows the autocorrelation of a Encrypted message and the figure indicates that the Encrypted message is white noise as expected. The Encryptor is not a Gold code is white noise with better autocorrelation characteristics.

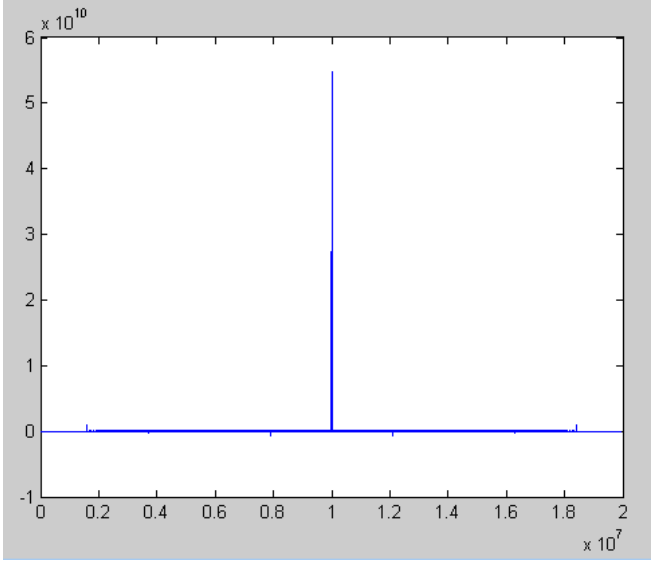


Figure 8. – Autocorrelation of a Encrypted message.

## Conclusions

The myth that Encryptor was very robust with only 64 bits was scrapped. The Encryptor remains good with strength of the order of 64 bits but the sequence has only 30 bits ($2^{30}$ in length). The improvements that can be made, a hash in the positioning of the encrypted file, so that it can not be attacked by knowing the file header, thus adding some garbage to the result file, or scrambling the file and then encrypt. Another improvement will be to move the state variable to 64 bits and eliminate the fixed zero bits.

## References

[1] Bruce Schneier, Applied cryptography : protocols, algorithms, and source code in C, 2nd ed, John Wiley & Sons.

About Author (s):

João Silvestre obtained his Licenciatura end MSc degrees in 1990 and 1995, from Coimbra University. Silvestre is now a full time invited Adjunct Professor at Instituto Superior de Engenharia de Coimbra, in the Electrotechnical Department, teaching mainly electronics topics. His current research interests include power electronics, fast digital electronics, signal processing and electrical vehicles.