Fast Pseudorandom Generator based on Packed Matrices

JOSE-VICENTE AGUIRRE[1], RAFAEL ÁLVAREZ[2],
LEANDRO TORTOSA[3], ANTONIO ZAMORA[4]
Dpt. of Computer Science and Artificial Intelligence
University of Alicante
Campus de San Vicente, Ap. Correos 99, 03080 Alicante
SPAIN

This research was partially supported by the Spanish grant GV06/018

Abstract: - Pseudorandom generators are a basic foundation of many cryptographic services and information security protocols. We propose a modification of a previously published matricial pseudorandom generator that significantly improves performance and security. The resulting generator is successfully compared to world class standards.

Key-Words: - Pseudorandom Generator, Stream Ciphers, Binary Matrices, Cryptography, Security.

1 Introduction

Most cryptographic systems are based on unpredictable quantities. The keys, prime numbers or challenge values of many cryptosystems need to be unpredictable enough so that the probabilities of all different values are more or less the same, making impossible any search optimization based on the reduction of the key space to the most probable values. These are obtained from random sequences that can be either truly random or pseudorandom, meaning that they are generated deterministically but appear to be random enough for practical use.

A truly random generator is based on a natural source of randomness. This source is sampled and then postprocessed to make it free of biasing and skewing.

A pseudorandom generator is a completely deterministic algorithm, in the sense that the sequence it generates is a function of its inputs and, unlike a truly random generator, its output can be reproduced. This means that we only need the seed (the input to the pseudorandom generator) in order to regenerate the complete output sequence. The output sequence is much longer than the seed and it is not really random, it is just undistinguishable from a real random sequence (see [5, 7, 8]).

For security applications we need to produce sequences with large periods, high linear complexities and good statistical properties. Several statistical tests are applied; these tests include checking the frequency of single bits, pairs of bits and of other bit patterns. Also, the autocorrelation and the linear complexity of the sequence are used.

In this paper we propose a modification to a previously published pseudorandom generator based on block upper triangular matrices that allows improving performance and security by using word packed matrices, implementing over Z_2 and introducing new extraction and key scheduling mechanisms.

2 Preliminaries

2.1 Original Generator

The generator is based on the powers of a block upper triangular matrix (BUTM) defined over Z_p, with p prime. As we take the different powers of a BUTM, we have as a result a sequence of matrices of very long period that has great properties in terms of randomness. Each element of the sequence (each BUTM) can be processed to obtain a series of values that produce an output sequence with great statistical values. This scheme is simple enough to be really fast but incorporates enough complexity to present great cryptographic properties. For more details see [1, 2].

2.2 Packed Matrices

The concept of word packed matrices is essential for the optimized implementation of the generator over Z_2. Packed matrices allow adding and multiplying binary matrices just by performing binary operations between processor registers, which is very efficient.

We define a matrix, whose elements lie in Z_2, as a word packed matrix if one of its dimensions (rows or columns) is packed as word sized groups of bits.

Operations involving packed matrices are equivalent to those between conventional matrices since packed matrices are, essentially, just a way of storing the elements of the matrix so that the computations required can be efficiently implemented

as binary operations between processor registers. Nevertheless, they present certain peculiarities of their own that must be taken into account.

The addition of packed matrices must be done between matrices of the same type, be them packed by rows or by columns. Although they could be unpacked and operated normally, the optimal way is to perform a XOR operation word by word.

The product operation between packed matrices is a little more complex than the addition. The product must be done between matrices of different types and with compatible sizes. The multiplicand has to be a row packed matrix, while the matrix corresponding to the multiplier must be packed by columns.

3 Description

3.1 Implementation

During each iteration, the following operations have to be performed:

$$\begin{aligned} E &= AX^{(h-1)} + XB^{h-1}, \\ X^{(h)} &= E, \\ F &= BB^{h-1}, \\ B^h &= F. \end{aligned}$$

It can be observed that $X^{(h)}$ has to be computed for each iteration but, B^h is also required, this forces to keep in memory the original matrices, X or B, and their power, $X^{(h)}$ or B^h. It is also necessary to employ temporary matrices E and F since the same matrix cannot be employed as source and destination at the same time.

Considering the peculiarities of the product operation between packed matrices we can identify the following matrices and types:

- A has to be a row packed matrix,
- B has to be a row packed matrix,
- B^h has to be a column packed matrix,
- X has to be a row packed matrix,
- $X^{(h)}$ has to be a column packed matrix,
- E and F are temporary column packed matrices.

Although the product operation between word packed matrices generates sparse bits instead of words, these bits can be repacked into the desired format (rows or columns) without a significant performance hit.

3.1.1 Parameters

Besides determining the format for each matrix, their sizes must also be decided for the correct operation of the implementation.

Several sizes and the number of digits of the corresponding period are shown in table 1. The option that appears to be more adequate is the r=64, s=48 since the word size is 32 bits in this case and 64 requires exactly 2 words. Moreover, the order obtained is excellent, allowing the resulting generator to be useful for a wide spectrum of applications.

r	s	digits
15	8	06
31	8	11
47	8	16
23	16	11
31	16	14
47	16	18
47	32	23
63	32	28
64	48	33
80	48	38
95	48	43
96	53	44

Table 1. Periods for different sizes on p=2.

3.2 Key scheduling

In order to augment security, the generator performs a key scheduling operation by following these steps:

1. Iterate the generator 64 times.
2. Collapse all words in $X^{(h)}$ by XORing them together.
3. Elevate A and B to the value contained in that word using a fast exponentiation algorithm.

A good starting point for the generator is achieved in this way (see [4]). This operation has only to be performed when the key changes.

3.3 Bit extraction

We have designed an extraction scheme that takes advantage of the word packed structure of the $X^{(h)}$ matrix in order to achieve the maximum possible efficiency.

The $X^{(h)}$ matrix contains 2 rows with 48 words of 32 bits each

	Original	Optimized	BBS	AES	RC4	Correction
Frequency	1.7716	0.0450	1.0646	1.0268	1.1035	2.7060
Serial	3.6994	0.0564	1.5252	1.4981	2.2746	4.6050
Poker 8	254.03	270.53	249.00	254.91	276.35	284.30
Poker 16	65377	65858	65607	65650	65681	65999
Runs	15.6624	14.6603	16.1032	15.9688	15.7268	23.5418
AutoCorr.	0.7967	0.7895	0.7978	0.7984	0.7972	1.2820
Lin. Comp.	10000	10001	10000	10000	10000	>=10000
Time	4.8316	0.0527	21.3425	0.2479	0.0170	-

Table 2. Results for the optimized generator.

$$X^{(h)} = \begin{bmatrix} \omega_{1,1} & \omega_{1,2} & \cdots & \omega_{1,48} \\ \omega_{2,1} & \omega_{2,2} & \cdots & \omega_{2,48} \end{bmatrix}.$$

The extraction mechanism consists in extracting a word per each column of $X^{(h)}$ applying a non linear function to certain words of the matrix.

For this purpose, the following non linear functions are defined

$$F_1(x,y,z) = (x \wedge y) \vee (\neg x \wedge z),$$
$$F_2(x,y,z) = (x \wedge z) \vee (y \wedge \neg z),$$
$$F_3(x,y,z) = y \oplus (x \vee \neg z).$$

These functions are applied in the following way

$$\gamma^i = \omega_{1,i} + F_1(\omega_{1,i+1}, \omega_{2,i+1}, \omega_{2,i}) + F_2(\omega_{2,i}, \omega_{1,i+1}, \omega_{2,i+1}) + F_3(\omega_{2,i+1}, \omega_{2,i}, \omega_{1,i+1})$$

obtaining a word of 32 bits, γ^i, for each column which produces a total of 48x32=1536 bits of output per iteration.

Combining boolean operations like OR, AND, XOR o NOT with the addition modulo 2^{32} prevents attacks that are targeted at deducing a part of the seed from a certain amount of output sequence. Moreover, 1536 output bits are extracted nonlinearly from a total of 3072 bits per iteration; making brute force attacks especially expensive (see [4]).

4 Results

The results are shown in table 2. The optimized generator with r=64, s=48 is compared with the original version, and other reference algorithms like Blum Blum Shub [3], AES [8] in output feedback mode and RC4 [6] working as pseudorandom generators.

The analysis includes the statistical tests described in [5], the linear complexity and the execution time. These results are the average values obtained in a series of 1000 different sequences of 20000 bits in length.

A test is considered to be successful if the result obtained is less than the correction value, except in the case of the linear complexity, where the expected value is $n/2$, being n the length of the sequence.

The tests have been performed using the same processor, compiler and optimizing options for all implementations, in order to make the comparison as fair as possible. The implementations have been done using standard C, avoiding assembler or special instruction sets.

4.1 Performance

The proposed optimization achieves a significant performance improvement of over two orders of magnitude compared to the original version. With this excellent result, the keystream generator lies within the same order of magnitude of the standard RC4, and is much faster than the rest of the reference algorithms studied.

We consider that, in order to achieve a significant performance increase over this optimized version over Z_2, the use of extended multimedia and vector instructions (MMX, SSE, etc.) would be required hampering the portability to other architectures not supporting this instruction sets. Nevertheless, the recent 64 bits microprocessors would allow a direct

performance increase by utilizing the bigger registers. This does not occur with algorithms like RC4 [6] that, by design, cannot take advantage of more powerful architectures directly.

4.2 Randomness

The implementation over Z_2 based on word packed matrices does not only provide a very satisfactory performance level, it also produces sequence of excellent quality in terms of randomness.

It can be observed that the optimized generator maintains the quality of original generator and achieves comparable and, sometimes, better results than the reference algorithms.

5 Conclusions

We have proposed a modification to a previously published pseudorandom generator that achieves a performance improvement of two orders of magnitude.

This optimization is based on an implementation over Z_2 and the use of packed matrices allows performing most calculations with native binary operations between processor registers. Moreover, the word parking system can take advantage of more powerful processors with bigger registers, like the new 64 bit CPUs, directly unlike most other algorithms.

The generator produces sequences of great quality in terms of randomness, comparable to world class standard reference algorithms, allows seeds up to 3072 bits in size, a long period and the extraction and key scheduling mechanisms provide more non linearity and security.

References:

[1] Álvarez, R., Climent, J.J., Tortosa, L., Zamora, A. An Efficient Binary Sequence Generator with Cryptographic Applications. Appl. Mathematics and Computation 167-1 (2005) 16-27

[2] Álvarez, R., Tortosa, L., Vicent, J. F., Zamora, A. An Integral Security Kernel. Transactions on Business and Economics 1-3 (2004) 241-246

[3] Blum, L., Blum, M., Shub, M. A Simple Unpredictable Pseudorandom Number Generator. SIAM J. Comput. vol. 15 (1986) 364-383

[4] Kelsey, J., Schneier, B., Wagner, D., Hall, C. Cryptanalitic Attacks on Pseudorandom Number Generators. Fast Software Encryption, Fifth International Workshop. Springer-Verlag, (1998) 168-188

[5] Menezes, A., van Oorschot, P., Vanstone, S. Handbook of Applied Cryptography. CRC Press, Florida (2001)

[6] Rivest, R. The RC4 Encryption Algorithm. RSA Data Security, Inc. (1992)

[7] Schneier, B. Applied Cryptography Second Edition: protocols, algorithms and source code in C. John Wiley and Sons, New York (1996)

[8] Stallings, W. Cryptography and Network Security: Principles and Practice. Fourth Edition. Prentice Hall, New Jersey (2006)