An Optimized Pseudorandom Generator using 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
{jaguirre[1], ralvarez[2], tortosa[3], zamora[4]}@dccia.ua.es

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

Abstract: - Most cryptographic services and information security protocols require a dependable source of random data; pseudorandom generators are convenient and efficient for this application working as one of the basic foundation blocks on which to build the required security infrastructure. We propose a modification of a previously published matricial pseudorandom generator that significantly improves performance and security by using word packed matrices and modifying key scheduling and bit extraction schemes. The resulting generator is then successfully compared to world class standards.

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

1 Introduction

During the latest years, the internet has become extremely popular. Everybody seems to be present in one way or another: businesses setup facilities for electronic commerce, governments provide services, and even individuals create sites of the most diverse subjects.

Despite the immense possibilities, the Internet and the Web are vulnerable to malicious attacks. The harmful results of these attacks are multiplied by the fact that an attack not only implies a loss of service and money, or an inconvenience to the end user; it also implies damage to the business reputation that not many can afford. Businesses have taken this into account, increasing the demand of more secure services.

Various approaches to internet security have been considered, differing in the scope of applicability and the location within the TCP/IP stack. The IPSec system is a general purpose solution placed below the TCP protocol so it is transparent to end users and applications; it includes a filtering system allowing that only the required traffic is processed by IPSec. The SSL (Secure Socket Layer) and TLS (Transport Layer Security) systems are also relatively general purpose, but are placed right over the TCP protocol, providing the choice of being implemented either within the protocol suite (and accessible to all applications) or by specific packages such as browsers and http servers. Another approach is placing the security system within the application, tailoring it to the specific needs and peculiarities of that given application; being SET (Secure Electronic Transaction) the most popular example of this approach.

All of these security systems use cryptography; generally using public key cryptosystems to establish a session key and private key (symmetric) cryptosystems to transfer data securely between both parties using that session key.

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. The system must be designed to prevent any observation or manipulation by the opponent and also any non random external interference (such as electromagnetic radiation from periodic sources, etc.). Proper function should be tested periodically ensuring that the previous conditions hold true. Natural sources of randomness include the thermal noise of a resistor, noise input from a microphone or a camera, particle emission time during radioactive decay, etc; software based events can also be useful, such as the system clock, the elapsed time between keystrokes, mouse moves or network packets, operating system statistics, etc.

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.

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.

Most available cryptographic generators are based on linear feedback shift registers (LFSRs). They are so popular because they can be easily implemented in hardware, they produce sequences of large periods with good statistical properties, and have a simple structure that can be analyzed easily.

LFSRs by themselves are not very secure, but because they are so efficient they are commonly enhanced with other techniques to improve their cryptographic properties.

Another popular generator is the Blum Blum Shub (BBS) generator (see [18]). Other generators combine a block cipher in different ways to obtain a cryptographically secure random bit sequence, such as the pseudorandom generator specified in ANSI X9.17 which performs three triple DES encryptions per iteration.

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

In this section we describe the required prerequisites for the correct comprehension of our proposal.

2.1 Original Generator

Our 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.

Consider the block upper triangular matrix M defined as

$$M = \begin{bmatrix} A & X \\ \mathbf{O} & B \end{bmatrix}, \qquad (1)$$

whose entries lie in Z_p, where A is an $r \times r$ matrix, B is an $s \times s$ matrix, X is an $r \times s$ matrix, and O denotes the $s \times r$ zero matrix.

The following result, which is the base of the generator, establishes the expression of the different powers of matrix M. It also defines matrix $X^{(h)}$ in terms of A, B and X.

Theorem. Let M be the block upper triangular matrix given by (1).Taking h as a non negative integer then

$$M^h = \begin{bmatrix} A^h & X^{(h)} \\ \mathbf{O} & B^h \end{bmatrix}, \qquad (2)$$

where

$$X^{(h)} = \begin{cases} 0 & if\ h=0, \\ \sum_{i=1}^{h} A^{h-i} X B^{i-1} & if\ h \geq 1. \end{cases} \qquad (3)$$

Also, if $0 \leq t \leq h$ *then*

$$X^{(h)} = A^t X^{(h-t)} + X^{(t)} B^{h-t}. \qquad (4)$$

In order to generate the pseudorandom bit sequence, matrices *A* and *B* are fixed and matrix *X* is randomly chosen, constituting the seed of the sequence. Next expression (4) is applied to obtain the following succession of matrices:

$$X^{(2)}, X^{(3)}, X^{(4)}, \dots \qquad (5)$$

For each matrix $X^{(h)}$ a bit extraction operation is determined obtaining a sequence of bits like

$$b^2, b^3, b^4, \dots \qquad (6)$$

One of the basic properties that every pseudorandom sequence should hold is that its period must be very long (at least a period of 2^{128}). The key for obtaining long periods for the sequence given by (6) is constructing matrices *A* and *B* as companion matrices to primitive polynomials so the period can be guaranteed to be at least

$$lcm(p^r - 1, p^s - 1).$$

The value of *p* or the sizes of *A* and *B* need not be very large in order to achieve long periods. For more information see [21, 23].

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

The following operations have to be performed per iteration:

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

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.

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 with p=2.

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.

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 [5]). 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

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

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

α	Monobit	Serial	Poker	Runs	Autocorrelation
0.001	10.830	13.820	330.5	51.18	3.090
0.005	7.870	10.590	316.9	45.56	2.576
0.010	6.635	9.210	310.5	42.98	2.326
0.025	5.024	7.378	301.1	39.36	1.960
0.050	3.842	5.992	293.2	36.42	1.645
0.100	2.706	4.605	284.3	33.20	1.282

Table 2. Threshold values for the statistical tests

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

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. For more information see [20, 22, 26-28].

4 Results

The results are shown in table 3. The optimized generator with *r*=64, *s*=48 is compared with the original version, and other reference algorithms like Blum Blum Shub [18], AES [19] in output feedback mode and RC4 [25] working as pseudorandom generators.

The generator has been checked with five different statistical tests (frequency tests plus autocorrelation tests) and with the linear complexity of the sequence.

The monobit test statistic is a single bit frequency test designed to check that the number of zeros and ones of the sequence is more or less the same.

The test is expressed by the following equation, with *n* being the total number of bits in the sequence and n_0 and n_1 being the number of zero bits and one bits respectively:

$$X_1 = \frac{(n_0 - n_1)^2}{n}.$$

This test follows a χ^2 distribution with 1 degree of freedom.

The serial test checks that the frequency of pairs of bits (00, 01, 10 and 11) is also about the same. As before, *n* is the total number of bits, n_0 is the number of zero bits, n_1 is the number of one bits and n_{00}, n_{01}, n_{10} and n_{11} are the number of occurrences of such pairs allowing them to overlap.

$$\begin{aligned}X_2 &= a - b, \\ a &= \frac{4}{n-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2), \\ b &= \frac{2}{n}(n_0^2 + n_1^2) + 1.\end{aligned}$$

This follows a χ^2 distribution with 2 degrees of freedom.

The poker test checks that patterns of length *m* appear the same number of times in the sequence without overlapping. The value *m* is obtained with $\left\lfloor \frac{n}{m} \right\rfloor = 5 \cdot 2^m$, *k* is taken as $k = \left\lfloor \frac{n}{m} \right\rfloor$ and n_i is the number of occurrences of the pattern *i*.

$$X_3 = \frac{2^m}{3}\left(\sum_{i=1}^{2^m} n_i^2\right) - k.$$

This follows a χ^2 distribution with $2^m - 1$ degrees of freedom.

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

Table 3. Results for the optimized generator.

A run is a pattern of all zeros or all ones; a block is a run of ones and a gap is a run of zeros. The expected number of runs of length i is $e_i = (n - i + 3)/2^{i+2}$. The value of k is taken as the largest integer i for which $e_i \geq 5$ and is the length limit for which runs are accounted; finally, B_i and G_i are the number of blocks and gaps of length i (up to length k) of the sequence. The runs test is expressed by:

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i};$$

following a χ^2 distribution with $2k - 2$ degrees of freedom.

The autocorrelation checks for coincidences between the sequence and itself displaced d bits. Taking $A(d)$ as the amount of bits not equal between the sequence and itself displaced by d bits, we have:

$$X_5 = \frac{2\left(A(d) - \frac{n-d}{2}\right)}{\sqrt{n-d}};$$

which follows a N(0,1) distribution.

The linear complexity of a sequence of bits is the length of the shortest LFSR that can generate that sequence. The expected linear complexity for a random sequence is $n/2$ being n the length of the sequence. If we plot the linear complexity of a sequence against its length (for $n = 1, 2, 3, \ldots$) we have the linear complexity profile which should follow closely the line $n/2$ if the sequence is random. The linear complexity of a sequence can be computed using the Berlekamp-Massey algorithm.

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 correction values can be found in table 2.

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 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 Integral Kernel

Most protocols in digital business employ symmetric cryptography to transfer large quantities of data, while asymmetric cryptography is used to swap session keys, digital signatures, etc. Additionally, hash functions can be used in order to improve efficiency and data integrity.

Our proposal can be integrated with these basic components, obtaining a security kernel which can be the basis of many protocols. Since cryptographic algorithms are extremely diverse in nature, scope and requirements, this integration is highly beneficial since it allows for cheap mass production, and ease of design of new secure systems which could use the kernel as a black box.

The cryptographic kernel is based on the powers of a block upper triangular matrix, which is a very flexible technique. It can be adjusted to satisfy memory and speed requirements and be implemented successfully either in hardware or software. Another advantage is that the same basic mathematical scheme can be used to build private key cryptosystems, public key cryptosystems and hash functions. Therefore, we only require implementing this technique once in order to provide these three types of algorithms, integrating a full cryptographic kernel in a single low cost device. This is a remarkable new concept that shows how useful this technique can be in cryptography.

5.1 The Symmetric Component

To cipher large amounts of information efficiently, we need a private key cryptosystem. For that purpose, we can build a stream cipher using the mathematical base of the kernel by taking advantage of its great randomness properties as shown previously. We first create a good pseudorandom generator and, once we have that, we use it as the keystream generator in a Vernam cipher scheme, taking the seed of the generator as the key of the stream cipher. This pseudorandom generator can also be used to generate general purpose random numbers such as session keys, challenge values, etc.

Once we have a proper keystream, ciphering the plaintext is as simple as XORing the keystream with it bit by bit. In order to decipher we XOR the keystream again with the ciphertext and retrieve the original plaintext. The seed of the generator is shared by both parties so that they can reproduce correctly the keystream.

5.2 The Asymmetric Component

Defining the operator $\otimes$ as

$$X^{(a)} \otimes X^{(b)} = X^{(a+b)}, \quad (9)$$

set $G = \{X^{(0)}, X^{(1)}, X^{(2)}, X^{(3)}, \ldots\}$ has a finite group structure and its order can be taken as large as needed to make our scheme secure.

The key exchange scheme between two users *U* and *V*, proposed for our kernel, is:

1. *U* and *V* accord values for *p, n, A, B* and *X*
2. *U* generates a random number *k* and computes A^k, B^k and $X^{(k)}$
3. *V* generates a random number *m* and computes A^m, B^m and $X^{(m)}$
4. The numbers *k* and *m* are respectively the private keys of *U* and *V*
5. The pairs $(X^{(k)}, B^k)$ and $(X^{(m)}, B^m)$ are respectively the public keys of *U* and *V*
6. *U* computes $X^{(k+m)} = A^k X^{(m)} + X^{(k)} B^m$
7. *V* computes $X^{(m+k)} = A^m X^{(k)} + X^{(m)} B^k$

With this scheme users *U* and *V* share matrix $X^{(k+m)}$ in *G*.

The computation of A^k, A^m, B^k, B^m, $X^{(k)}$ and $X^{(m)}$ can be done efficiently adapting the existing quick exponentiation algorithm in Z_p.

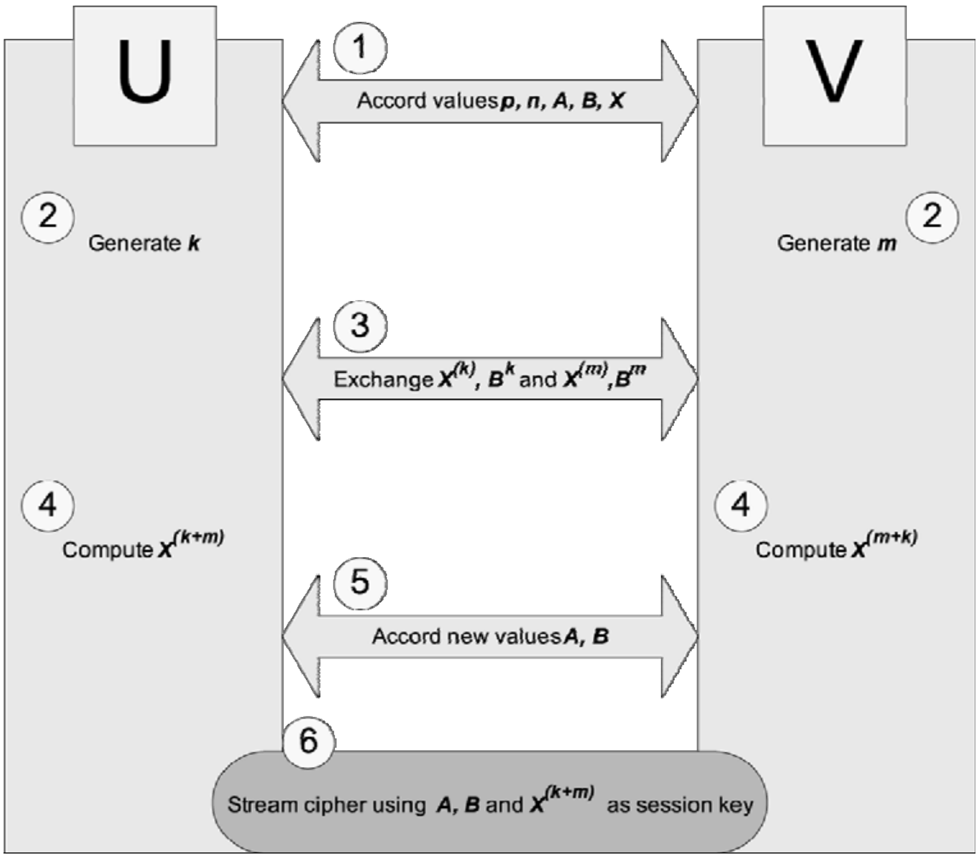
U
V
1
Accord values p, n, A, B, X
2
Generate k
2
Generate m
3
Exchange $X^{(k)}$, B^k and $X^{(m)}$, B^m
4
Compute $X^{(k+m)}$
4
Compute $X^{(m+k)}$
5
Accord new values A, B
6
Stream cipher using A, B and $X^{(k+m)}$ as session key

Figure 1. Secure communication scheme.

It is computationally infeasible, for an attacker, to know the shared key $X^{(m+k)}$ without the previous knowledge of k and m, because the problem the attacker would be facing is in the order of complexity of the discrete logarithm problem.

The scheme described previously can be adapted to perform digital signature using a similar technique to the ElGamal cryptosystem. Since the kernel requires little resources, it is suitable for low power or low cost environments. For more information see [10, 12, 13, 24].

5.3 The Hash Component

Taking the mathematical base of the kernel we can also build a hash function. We can use the pseudorandom generator as a diffusion and compression mechanism accumulating its results over a fixed length register. The stream cipher proposed can be also adapted to perform a hash function in the way shown in [27].

5.4 Applications

This integral security kernel can be used by any digital business protocol requiring security at any level, like A/V content distribution systems, anonymous peer to peer systems, certified email systems, online payment systems, etc. It can be implemented on any platform (PC, dedicated hardware, PDA, latest generation of cell phones, smart cards) and data transport system (Internet, wireless networks, satellites, terrestrial digital transmissions, etc.), being capable of adapting to the technological evolutions in the communications sector. Application examples can be seen in [2-9, 11, 14].

It is efficient and easy to implement either in hardware or software and requires very little resources, making possible its implementation in a wide spectrum of devices, especially those of low cost. In this way, confidentiality (ciphered information), integrity (no alteration warranty) and authentication (identity verification using digital signature) are assured in the communications.

For each of the communication channels established between the different parties (see figure

1), we need to guarantee confidentiality, authentication, and information integrity. For that purpose we require the usage of symmetric and asymmetric cryptography, a random generator and a hash function (operations offered by the proposed kernel).

In this way, the kernel provides all the means for a secure communication between two parties:

- First, both parties must establish values for p, n, A, B and X.
- Then, U generates a random k of sufficient length, V generates m in the same way.
- U sends V values $X^{(k)}$ and B^k, V does the same sending U the values $X^{(m)}$ and B^m.
- U computes $X^{(k+m)}$ and V computes $X^{(m+k)}$, since both parties reach the same result they now share this secret key.
- U and V can agree on new values for A and B.
- Taking the new A and B, along with $X^{(k+m)}$, we have the session key for our secure channel, using the kernel's stream cipher.

6 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.

Block upper triangular matrices can also be used to implement other cryptographic primitives, integrating a security kernel with many applications for low power / low cost secure solutions.

References:

[1] Aguirre, J-V., Alvarez, R., Tortosa, L., Zamora, A. Fast Pseudorandom Generator based on Packed Matrices. WSEAS Information Security and Privacy (2007) 98-101

[2] Aguirre, J-V., Alvarez, R., Tortosa, L., Zamora, A. Secure Lightweight P2P Multiconferencing. WSEAS Transactions on Communications, vol. 6-1 (2007) 195-200

[3] Aguirre, J-V., Alvarez, R., Sanchez, J., Zamora, A. Broadcast Multiplexing and Subchanneling for Secure P2P Multiconferencing. WSEAS Transactions on Computers, vol. 6-3 (2007) 522-527

[4] Aguirre, J-V., Alvarez, R., Noguera, J-V., Zamora, A. A Secure Remote Database Backup System. WSEAS Artificial Intelligence, Knowledge Engineering and Databases (2006) 43-46

[5] Aguirre, J-V., Alvarez, R., Noguera, J-V., Zamora, A. A Database Backup System with Secure Remote Data Transmission. WSEAS Transactions on Information Science Applications, vol. 4-3 (2006) 796-801

[6] Aguirre, J-V., Alvarez, R., Noguera, J., Tortosa, L., Zamora, A. Secure VoIP and Instant Messaging on Small PDA Devices. WSEAS Transactions on Computers, vol.5-1 (2006) 171-176

[7] Aguirre, J-V., Alvarez, R., Sanchez, J., Zamora, A. Silence Detection in Secure P2P VoIP Multiconferencing. WSEAS Information Security and Privacy (2006) 11-14

[8] Aguirre, J-V., Alvarez, R., Tortosa, L., Zamora, A. Lightweight Peer-to-Peer Secure Multi-Party VoIP Protocol. WSEAS Information Security and Privacy (2006) 7-10

[9] Aguirre, J-V., Alvarez, R., Noguera, J-V., Tortosa, L., Zamora, A. A Viability Analysis of a Secure VoIP and Instant Messaging System on a Pocket PC. WSEAS Information Security and Privacy (2005) 218-223

[10] Alvarez, R., Martinez, F-M., Vicent, J.F., Zamora, A. A New Public Key Cryptosystem based on Matrices. WSEAS Information Security and Privacy (2007) 36-39

[11] Alvarez, R., Oliver, J., Vicent, J., Zamora, A. Improving GSM Security for Voice and Text Data Transmission. WSEAS Transactions on Computers, vol. 5-1 (2006) 165-170

[12] Alvarez, R., Tortosa, L., Vicent, J-V., Zamora, A. Block Upper Triangular Matrices for Authentication and Integrity. WSEAS Transactions on Mathematics, vol.4-4 (2005) 339-346

[13] Alvarez, R., Tortosa, L., Vicent, J-F., Zamora, A. A Public Key Cryptosystem based on Block Upper Triangular Matrices. WSEAS Information Security and Privacy (2005) 163-168

[14] Alvarez, R., Oliver, J., Vicent, J-F., Zamora, A. Secure Communication System over a GSM Network. WSEAS Transactions on Computers, vol.5-1 (2005) 171-176

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

[16] Álvarez, R., Climent, J.J., Tortosa, L., Zamora, A. Un generador matricial de claves frente a Blum Blum Shub, RECSI’04 (2004) 113-123

[17] Álvarez, R., Climent, J.J., Tortosa, L., Zamora, A. A Pseudorandom Bit Generator Based on Block Upper Triangular Matrices. LNCS Web Engineering, vol. 2722 (2003) 299-300

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

[19] Daemen, J., Rijmen, V. The Design of Rijndael. Springer-Verlag (2002)

[20] 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

[21] Lidl, R., Niederreiter, H. Introduction to Finite Fields and their Applications. Cambridge University Press, Cambridge (1994)

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

[23] Odoni, R. W. K., Varadharajan, V., Sanders, P. W.: Public Key Distribution in Matrix Rings. Elec. Letters, vol. 20 (1984) 386-387

[24] Rivest, R., Shamir, A., Adleman, L. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. ACM Communications, vol. 21 (1978) 120-126

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

[26] Rueppel, R. A. Analysis and Design of Stream Ciphers. Springer-Verlag, Berlin (1986)

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

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