An Algorithm for Fast Software Encryption

Zhaneta Tasheva

Abstract: *An algorithm for fast software encryption is proposed in this paper. It is based on the architecture of new pseudo random number generator (PRNG), named Self–Shrinking p–adic Generator (SSPG). In the paper first, the basic SSPG architecture and algorithm are recalled. Then, the software implementation in Visual C++ environment is presented. Finally, the results of some images and texts, encrypted with SSPG sequence, are discussed.*

The proposed algorithm is suitable for fast software encryption because it allows generating uniform, scalable and unpredictable pseudo random sequences with large period.

Key words: *Cryptography, Encryption Algorithm, Stream Ciphers, FCSRs, PRNG.*

INTRODUCTION

The need of software-oriented stream ciphers in modern communication and information systems has rapidly grown during the last several years. One of the most used cryptographic systems is the binary additive stream cipher in witch the keystream, the plaintext and the ciphertext are basic binary sequences. The keystream is generated from a keystream generator, which takes a secret key as a seed, and produces a long pseudorandom sequence. The ciphertext is generated by bitwise modulo 2 additions of the keystream and the plaintext.

The main goal of software-oriented stream cipher design is to generate efficient pseudorandom sequence [1], [5], [6] whit fast software algorithm and with truly random sequences properties. To satisfy this need an algorithm for fast software encryption with new pseudo random number generator, named Self–Shrinking *p*–adic Generator (*SSPG*) [7] is proposed and the analysis of encrypted with *SSPG* images and texts are given in this paper.

THE SELF - SHRINKING P-ADIC GENERATOR ALGORITHM

In this section the basic architecture of a new Self-Shrinking *p*–adic Generator and some basic *SSPG* properties will be recalled.

A. THE *SSPG* ARCHITECTURE

In contrast with the classic self-shrinking generator [4] the *SSPG* architecture (Fig. 1) uses a *p*-adic *FCSR* instead of *LFSR*. This allows the generator to produce a number in the range 0 to $p-1$ in one step ($a_i = [0, 1, \ldots, p-1]$). The self-shrinking *p*-adic generator selects a portion of the output *p*-adic *FCSR* sequence by controlling of the *p*-adic *FCSR* itself using the following algorithm.

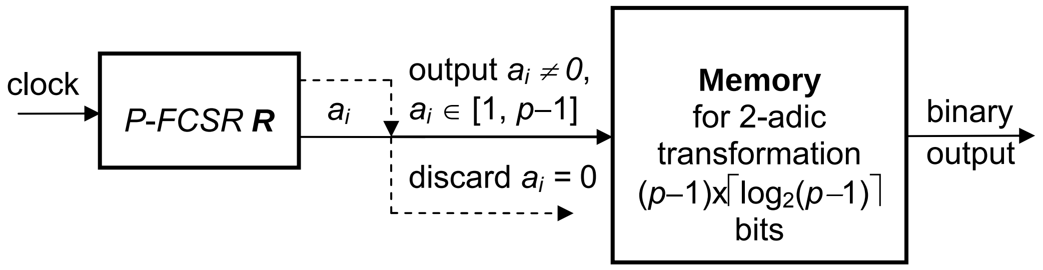

clock
P-FCSR R
a_i
output $a_i \neq 0$, $a_i \in [1, p-1]$
discard $a_i = 0$
Memory for 2-adic transformation $(p-1)\times\lceil\log_2(p-1)\rceil$ bits
binary output

Fig. 1. *Self-Shrinking p-adic Generator*

Definition 1: The algorithm of the Self-Shrinking *p*-adic Generator (Fig. 1) consists of the following steps:

1. The p-adic *FCSR R* is clocked with clock sequence with period τ_0.
2. If the p-adic *FCSR* output number is not equal to 0 ($a_i \neq 0$), the output bit forms a part of the p-adic *SSPG* sequence. Otherwise, if the output number of the p-adic *FCSR* is equal to 0 ($a_i = 0$), the p-adic output number of *SSPG* is discarded.
3. The shrunken p-adic *SSPG* output sequence is transformed in 2-adic sequence in which every p-adic number is presented with $\lceil \log_2 (p-1) \rceil$ binary digits, where $\lceil x \rceil$ is the smallest integer which is greater or equal to x. Every output number i from 1 to p–1 of p-adic *SSPG* sequence is depicted with p–adic expansion of the number:

$$i - 1 + \frac{2^{\lceil \log_2 (p-1) \rceil} - (p-1)}{2}. \tag{1}$$

The binary presentations of p-adic shrunken *SSPG* output numbers with various prime p from 3 to 17 according to (3) are shown in Table 1.

Table 1: *Binary presentation of p-adic SSPG output*

***p*-adic number**	**Binary presentation of *p*-adic number**					
	***p* = 3**	***p* = 5**	***p* = 7**	***p* = 11**	***p* = 13**	***p* = 17**
1	0	00	001	0011	0010	0000
2	1	01	010	0100	0011	0001
3	–	10	011	0101	0100	0010
4	–	11	100	0110	0101	0011
5	–	–	101	0111	0110	0100
6	–	–	110	1000	0111	0101
7	–	–	–	1001	1000	0110
8	–	–	–	1010	1001	0111
9	–	–	–	1011	1010	1000
10	–	–	–	1100	1011	1001
11	–	–	–	–	1100	1010
12	–	–	–	–	1101	1011
13	–	–	–	–	–	1100
14	–	–	–	–	–	1101
15	–	–	–	–	–	1110
16	–	–	–	–	–	1111

The proposed *SSPG* uses the generalization of 2-adic *FCSRs* [2], [3], [5] with stage contents and feedback coefficients in Z/(p) where p is a prime number, not necessarily 2.

The unlinearity in proposed *SSPG* is guaranteed by the fact that it is unknown at which positions the *FCSR*-sequence is shrunken. As a result the linear algebraic structure of the original *FCSR*-sequence is destroyed. The software *SSPG* implementation is very fast because the pseudorandom generator produces $\lceil \log_2 (p-1) \rceil$ binary digits in one step.

B. The *SSPG* Properties

In this subsection some theorem for S*SPG* properties will be recalled. The proofs of them can be found in [7].

Theorem 1: The period of the self-shrunken p-adic generator realized by maximum length p-adic *FCSR* of length L and connection integer q is:

$$T_{SSPG} = T^{*} . \lceil \log_2 (p-1) \rceil, \tag{2}$$

where T^* is the number of output p-adic *FCSR* numbers different from 0.

Theorem 2: The self-shrunken output *SSPG* sequence generated by maximum length *p*-adic *FCSR* of length *L* and connection integer *q* is a balanced sequence. Morover:

1. If the prime *p* can be present in form $p = 2^n + 1$, the numbers of 0s and 1s in the self-shrunken output *SSPG* sequence are balanced and equal to:

$$N_{0s} \approx N_{1s} \approx \left\lceil \frac{q-1}{2^n+1} \right\rceil .2^{n-1}.n\,. \tag{3}$$

2. if the prime *p* satisfies the inequality:

$$p < 2^{\lceil \log 2\,(p-1) \rceil} + 1 \tag{4}$$

the numbers of 0s and 1s in the self-shrunken output *SSPG* sequence are balanced and equal to:

$$N_{0s} \approx N_{1s} \approx \left\lceil \frac{q-1}{2^n+1} \right\rceil . \frac{\lceil \log_2(p-1) \rceil .(p-1)}{2}\,. \tag{5}$$

As one can see from Theorem 1 and Theorem 2 the proposed algorithm generates sequences with following properties:

- The sequences are **balanced;**
- They have **large period**;
- There isn't **possibility to appear subsequences** of $2\lceil \log 2\,(p-1) \rceil$ consecutive 1s or 0s in sequences.

FAST ENCRYPTION WITH SOFTWARE IMPLEMENTED *SSPG*

The software implementation of *SSPG* is realized in Visual C++ environment. It uses the base template class FCSR <class T> which provide a straightforward way of abstracting type information. The template class FCSR <class T> models the *p*–adic *FCSR* definition 2 [3].

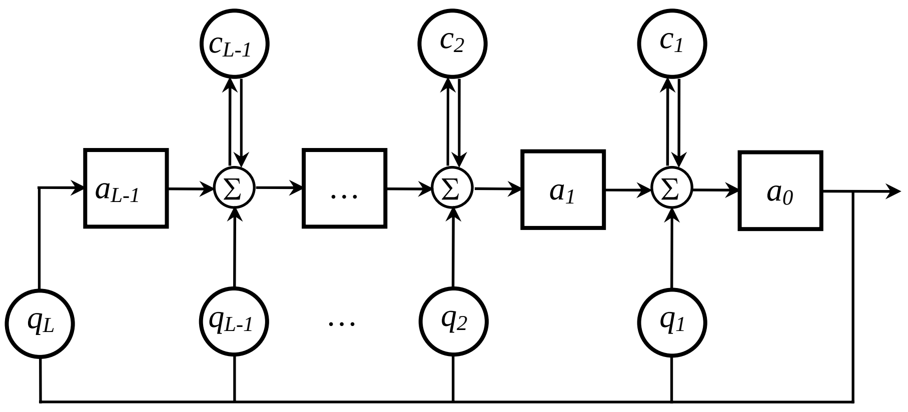


Fig. 2. *Galois FCSR*

Definition 2: A *p-adic feedback with carry shift register* with Galois architecture of length *L* (Fig. 3) consists of *L stages* (or *delay element*s) numbered *0, 1, …, L-1*, each capable to store one *p*-adic (0, 1, …, *p*-1) number and having one input and one output; and a clock which controls the movement of data. During each clock cycle the following operations are performed:

1. The content of stage 0 is output and forms part of the *output sequence*;
2. The sum modulo *p* after stage *i* is passed to stage *i - 1* for each *i*, $1 \le i \le L-1$;
3. The output of the last stage 0 is introduced into each of the tapped cells simultaneously, where it is fully added (with carry) to the contents of the preceding stages.

The $q_1, q_2, \ldots, q_L$ are the *feedback multipliers* and the cells denoted with $c_1, c_2, \ldots, c_{L-1}$ are the *memory* (or "*carry*") bits. If

$$q = -1 + q_1 p + q_2 p^2 + \ldots + q_L p^L \tag{6}$$

is the base p expansion of a positive integer:

$$q \equiv -1 \pmod{p}, \tag{7}$$

then q is a connection integer for a *FCSR* with feedback coefficients $q_1, q_2, \ldots, q_L$ in Z/(p).

With each clock cycle, the integer sums:

$$\sigma_j = a_j + a_0 q_j + c_j \tag{8}$$

is accumulated.

At the next clock cycle this sum modulo p

$$a'_{j-1} = \sigma_n \pmod{p} \tag{9}$$

is passed on to the next stage in the register, and the new memory values are:

$$c'_j = \sigma_n \ (\mathrm{div}\ p). \tag{10}$$

The template class FCSR <class T> has seven class–members (Fig. 3) which define accordingly the initial state and the memory of the *FCSR*, its current state and memory, its feedback polynomial, the size and the prime p of GF(p). The method Shift( ) (Fig. 4) realizes the basic *FCSR* operations (8) ÷ (10) performed during one clock cycle and the method Reset( ) resets the *FCSR* back to its initial state and memory.

```
template<class T> class FCSR {          // Feedback with Carry Shift Register Template Class
  public:
    ….
  private:
    T *initial_state;       // the initial state of the FCSR
    T *initial_memory;      // the initial memory of the FCSR
    T *state;               // the current state of the FCSR
    T *memory;              // the current memory state of the FCSR
    T *polynomial;          // the feedback polynomial of the FCSR
    int size;               // size of the FCSR
    int prime;              // prime p in GF(p)
};
```

Fig. 3: *Declaration of template class FCSR*

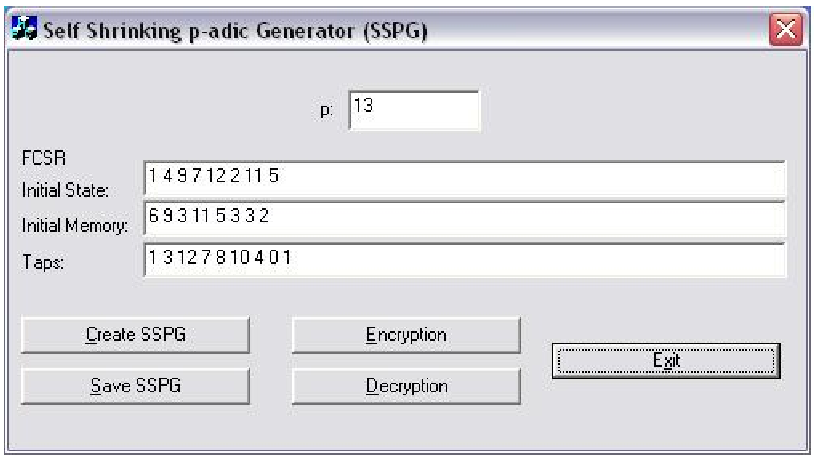
```
template <class T> T FCSR<T>::Shift( ) {          // Method for Shift FCSR operation
  T temp = state[0];
   for (int i = 1; i < size - 1; i++) {
            int sum = state[i] + temp*polynomial[i]+memory[i];
            state[i-1] = sum % prime;
            memory[i] = sum / prime;
   }
  state[size-1]=(temp*polynomial[size+1])%prime;
  return (temp);
}
```

Fig. 4: *Definition of method Shift ( )*

Using templates, the design class FCSR can operate on data of many types and the code is generated for a template class and its functions only when they are instantiated.

The software implementation "Self-Shrinking *p*–adic Generator" (Fig. 5) realizes *SSPGs* with primes $p \in [3, 17]$. The software application opportunities are:

1. The *SSPG* seed has the length up to $3*256*\lceil \log_2 p \rceil$ bits for the initial state, the initial memory and the feedback polynomial of the control *p*–adic *FCSR*. The coefficient $\lceil \log_2 p \rceil$ is used to present every *p*–adic number (for example the possible numbers in GF(5) are 0, 1, 2, 3 and 4).
2. The *SSPG* seed may be changed by modifying the feedback polynomial, initial state and initial memory of *p*–adic *FCSR*.
3. The output text file of *SSPG* pseudo random number sequence with arbitrary bit length, defined by user can be saved.
4. The arbitrary files (.txt, .bmp, .bin, *.*) can be encrypted and decrypted with *SSPG*.



p: 13

FCSR
Initial State: 1 4 9 7 12 2 11 5
Initial Memory: 6 9 3 11 5 3 3 2
Taps: 1 3 12 7 8 10 4 0 1

Fig. 5: *Software application "Self–Shrinking p–adic Generator"*

The properties of *SSPG* cryptographic generator were been analyzing by means of 300 text files and 300 color BMP images encrypted with *SSPG* sequences with different seeds. The two used *SSPG*s with its polynomials are presented in Table 1 and the corresponding texts and images, encrypted with these *SSPG*s, are given in Table 2 and on Fig. 6.

Table 1: *Polynomials for SSPG elements*

PRNG	*SSPG Elements*	*Polynomials*
SSPG1 *p = 13*	Initial State	1 4 9 7 12 2 11 5
	Initial Memory	6 9 3 11 5 3 3 2
	Taps	1 3 12 7 8 10 4 0 1
SSPG2 *p =17*	Initial State	1 14 9 7 10 12 1 5
	Initial Memory	6 9 13 10 5 3 0 2
	Taps	1 3 10 7 8 9 14 10 1

Table 2: *Original and Encrypted with SSPGs texts*

Original text	International Conference on Computer Systems and Technologies - CompSysTech' 2005
Encrypted with *SSPG1* text	I□□чsL□S□5 V'□%щVЪэрUеПу¶(‡пґл6цЁМНЕ†$(ц° ·□У"Ющ ¬«эчV1(р□)cj@¬t□Юґ5Ошщ:К'ІинҐЂ
Encrypted with *SSPG2* text	IюOjЮзП□єМ©'SrЛiqQfиќреЙЦп®□У?ё;□цгµЧ□s{-.‡®еЩ3+љ6,□:« 3ЯєNxЪDFД"‹Ѓ{ф9ьр□л®¤□©4JB

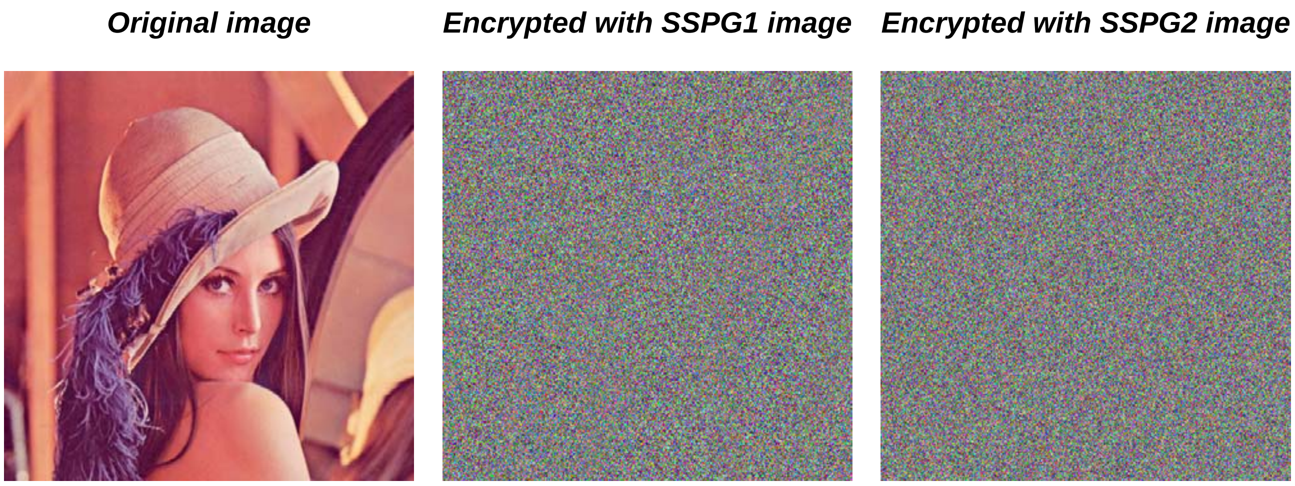


Fig. 6: *Original and Encrypted with SSPGs images*

CONCLUSIONS

The analysis of all 300 encrypted images and 300 encrypted text files with developed algorithm for software encryption leads to the following conclusions:

1. The generated pseudo random sequences are **balanced;** have **large period**; and the possibility to appear **long subsequences** of consecutive 1s or 0s is excluded.

2. The images and text are **well encrypted**, i.e. no one can identify them.

3. The values of three basic colours components R (red), G (green) and B (blue) from 0 to 255 **appear with approximately equal probability** (shown on Fig. 6).

4. The values of output *SSPG* sequence are **uniformly distributed**.

5. The proposed algorithm is **fast** because in one clock step the generator forms $\lceil \log_2 (p-1) \rceil$ binary digits.

REFERENCES

[1] D. Coppersmith, H. Krawczyk, Y. Mansour, "The Shrinking Generator", *Proceedings of Crypto 93*, Springer-Verlag, pp. 22-39, 1994.

[2] A. Klapper, M. Goresky, "2-adic Shift Register. Fast Software Encryption", *Second International Workshop,* (Lecture Notes in Computer Science, vol. 950, Springer Verlag, N. Y.,) pp. 174–178, 1994.

[3] A. Klapper, M. Goresky, "Feedback Shift Registers, 2-adic Span, and Combiners With Memory.", *Journal of Cryptology*, Volume 10, Number 2, 1997, pp. 111-147, http://www.math.ias.edu/~goresky/pdf/2adic.jour.pdf

[4] W. Meier, O. Staffelbach, "The Self-Shrinking Generator", *Proceedings of Advances in Cryptology*, EuroCrypt '94, Springer-Verlag, pp. 205-214, 1998.

[5] P. van Oorshot, A. Menezes, S. Vanstone, "*Handbook of Applied Cryptography*", CRC Press, 1997.

[6] Schneier, "*Applied Cryptography*", John Wiley & Sons, New York, 1996.

[7] Zh. N. Tasheva, B. Y. Bedzhev, B. P. Stoyanov, "Self-Shrinking p-adic Cryptographic Generator", accepted for *XL International Scientific Conference on Information, Communication and Energy Systems and Technologies ICEST 2005*, June 29 – July 1, 2005, Nish, Serbia and Montenegro.

ABOUT THE AUTHOR

Assistant Prof. Eng. PhD. Zhaneta N. Tasheva, NMU "V. Levski", Faculty of Artillery and Air Defence, Shoumen, Bulgaria, Phone: +359 54 5 23 71, e-mail: tashevi86@yahoo.com.