

## Abstract

We make a new proposal about how to use in an effective way a CSPRBG (Computationally Secure Pseudo Random Bit Generator) for cryptographic purposes. We introduce the definitions of TCSPRBG (Typical CSPRBG) and SCSPRBG (Special CSPRBG). In particular the definition of SCSPRBG synthesizes in a simple way our proposal of how to modify a CSPRBG in order to achieve a higher throughput rate, while retaining some essential features of its computational security.

We then summarize which should be, in our opinion, a "standard way" to use a CSPRBG for cryptographic purposes. We eventually present as an application, a particular SCSPRBG for which we can achieve throughput rates greater than 100 Mbits/sec on current mobile devices.

# A proposal of a faster variant of known provably secure PRBGs

A. Corbo Esposito and F. Didone

December 24, 2013

## 1 Introduction

In this paper we propose an encryption scheme based on a CSPRBG that we believe has some innovative features.

It is well known that a CSPRBG can be used to communicate encrypted data. In particular the computational work needed to ensure cryptographic security can be carried out outside the temporal interval of encryption-decryption of data: in fact the encryption-decryption phase consists of two xor with a same piece of the string produced by CSPRBG, therefore its execution time is negligible. We describe this aspect saying that the encryption method has an execution time with zero latency, just to highlight that this scheme can be applied to real-time communications. Moreover it is clear that the maximum rate allowed for communication is only bounded by the throughput rate of the CSPRBG. We can even replace the word "maximum" with word "average" in the previous statement if we use some appropriate buffering algorithm.

We first take into account the existence of PRBGs for which security results have been proved. In particular we consider RSAPRBG and QUAD, for which security results have been proved in papers [RSA], [QUAD1], [QUAD2]. In the paper [ARTICOLO2] we carried out a Java software implementation of both these PRBGs putting in place all knowledge and tricks we know, obtaining in both cases a throughput rate of several Mbit/sec (see [], pag. 4); such results seem to be better of the ones forecasted in [], pag. 4 for RSAPRBG and somewhat not bad for QUAD, giving the results obtained in [], pag. 4 for a hardware implementation of QUAD on a FPGA.

Such throughput rates, nevertheless, in our opinion are insufficient for some kind of applications (namely encryption of audio/video data), since while they could be improved by a hardware implementation, they are actually obtained used with CPU load of 100

In order to obtain a faster variant of these CSPRBGs we introduce the definitions of two particular classes of them, namely the class TCSPRBG that turns out to contain both RSAPRBG and QUAD, and the class SCSPRBG; a PRBG belonging to this class starts from a correspondent TCSPRBG and uses one more one-way function  $w$  as a "local" expander of the bit stream, in order to obtain a higher throughput rate while basically maintaining the same security (even if function  $w$  could be inverted in some case, this does not compromise the security of the bit stream, see proposition X.X).

To give a precise idea of the throughput rates we can obtain, we consider a particular choice for the function  $w$ . We call it the `pick` function. Again paper [1] provides numerical results of a java software implementation of this function, see [1].

In the final section we show how the inversion of the `pick` function is the inversion of a (very particular) quadratic system in  $\text{GF}(2)$ . Moreover we also clarify some relations between the `pick` function and the function  $f$  used in QUAD.

Then we develop some strategies to invert the `pick` function. We start from the simplest ones or from some adaptation of known strategies for inversion of quadratic systems to our particular case, trying our best to exploit particularities of the `pick` function to make them faster. The best time estimated for the inversion `pick` function seem to confirm that such inversion is practically not possible for the choice of parameters we have chosen in [1].

We anyway aren't able to obtain sharp lower bounds for the time needed for inversion.

It seems to us however that the combined effect of proposition X.X and the estimated times for the inversion of the `pick` function is that an SCSPRBG based on a correspondent TCSPRBG given by RSAPRBG or QUAD coupled with the choice of `pick` function as  $w$  deserves some attention.

Organization. Section 2 contains notations and definitions. In the section 3 we introduce the definitions of TCSPRBG and SCSPRBG along with some comments and remarks. In the section 4 we describe the method; in the section 5 we discuss the applications of the method on existing mobile devices.

## 2 Preliminary notations and definitions

Notation. We use  $i \oplus j$  to denote the bitwise xor of integers  $i$  and  $j$  and  $||$  to denote the concatenation of two sequences.

### 2.1 One way function

There are some types of functions that play a significant roles in cryptography. One of these types is the one way function. We adopt for our purposes the following definitions of one-way function from [12].

**Definition 2.1 (Strong one way function)** A function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  with  $m = O(n)$  is called (strongly) one way function if the following two condition hold:

1. Easy to compute: There exists a deterministic polynomial time algorithm  $A$  such that on input  $x$  algorithm  $A$  outputs  $f(x)$  (i.e.  $A(x) = f(x)$ ).
2. Hard to invert: For every probabilistic polynomial time algorithm  $A'$ , every positive polynomial  $p(\cdot)$  and all sufficiently large  $n$ 's,

$$\Pr[A'(f(U_n)) \in f^{-1}(f(U_n))] < \frac{1}{p(n)}$$

where  $U_n$  denotes a random variable uniformly distributed over  $\{0, 1\}^n$ .

**Definition 2.2 (Weak one way function)** A function  $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$  with  $m = O(n)$  is called (weak) one way function if the following two condition hold:

1. Easy to compute: There exists a deterministic polynomial time algorithm  $A$  such that on input  $X$  algorithm  $A$  outputs  $f(x)$  (i.e.  $A(x) = f(x)$ ).
2. Slightly hard to invert: There exists a polynomial  $p(\cdot)$  such that for every probabilistic polynomial time algorithm  $A'$  and all sufficiently large  $n$ 's:

$$Pr[A'(f(U_n)) \notin f^{-1}(f(U_n))] > \frac{1}{p(n)}$$

## 2.2 Random bit generator

**Definition 2.3** A random bit generator is an algorithm with outputs a sequence of statistically independent and unbiased binary digits.

**Remark 2.1** It is possible to use a random bit generator to generate random numbers. For example a integer number in the interval  $[0, n]$  can be obtained by generating a random bit sequence of length  $\lceil \log_2 n \rceil + 1$  and converting it to an integer.

A true random bit generator requires a naturally occurring source of randomness. It is difficult to design a hardware device or software program that produces uncorrelated bit sequences exploiting this randomness. Moreover this kind of generator is influenced by external features so it must be periodically tested.

There are two kind of true random bit generator:

- hardware-based generator (that exploits the randomness which occurs in some physical phenomena);
- software-based generator (that can be based on the system clock or on the content of input/output buffers, and this is more difficult to design than previous).

## 2.3 Pseudo random bit generator

**Definition 2.4** A pseudo-random bit generator (PRBG) is a deterministic algorithm which, given a truly-random binary sequence of length  $k$ , outputs a binary sequence of length  $l \gg k$  which appears to be random. The input to the PRBG is called seed and the output is called a pseudo-random bit sequence.

The output to the PRBG is not random because the number of output sequences is a small fraction,  $2^k/2^l$  of all binary sequences of length  $l$ .

The idea is to consider a small truly-random sequence and to expand it in a longer sequence. In this way a possible adversary can not easily distinguish between output sequences of the PRBG and truly-random sequences of length  $l$ . Therefore there are statistical tests that represent necessary conditions but not sufficient so that a generator may be secure. In fact it is impossible to prove mathematically that the output of a generator is truly-random. Tests are probabilistic.

A minimum security condition for a random sequence generator is that the length  $k$  of the random seed should be sufficiently large so that a search over  $2^k$  elements is infeasible for the adversary.

The seed must have two properties:

- the output sequence of PRGB must be statistically indistinguishable from the truly-random sequence;
- output bits must be unpredictable by the adversary that has limited computational resources.

We now have to specify what "appears to be random" mean.

**Definition 2.5** A PRGB is said to pass all polynomial-time statistical tests, and therefore can be considered as a cryptographic secure PRGB, if no polynomial-time algorithm can distinguish between an output sequence of the generator and truly-random sequence with probability significantly greater than  $1/2$ .

**Definition 2.6** A pseudo-random bit generator is said to pass the next-bit test if there is no polynomial-time algorithm which, on input of the first  $l$  bits of an output sequence  $s$ , can predict the  $(l + 1)^{st}$  bit of  $s$  with probability significantly greater than  $\frac{1}{2}$ .

**Remark 2.2** A pseudo-random bit generator passes the next-bit test if and only if it passes all polynomial-time statistical tests.

**Definition 2.7** A PRBG that passes the next-bit test (possibly under some plausible but unproved mathematical assumption such as the intractability of factoring integers) is called a cryptographically secure pseudo-random bit generator (CSPRBG).

### 3 Typical CSPRBG

**Definition 3.1** A typical-CSPRBG (denoted with TCSPRBG) is defined as follows. Computed the initial value  $x_0$  from a true random seed  $u_0$  after an initialization phase, let  $f$  be an one-way invertible function and  $g$  be an one-way not (invertible) function. Compute:

$$x_{i+1} = f(p_1, \dots, p_k, x_i) \quad \forall i = 0, 1, \dots$$

where  $p_1, \dots, p_k$  are fixed and known parameters,

$$y_i = g(x_i) \quad \forall i = 1, 2, \dots$$

The output sequence to pseudo-random generator is:

$$s = y_1 || y_2 || \dots$$

where  $||$  denotes the concatenation.

The output sequence depend on the properties of the one-way function used therefore it may be necessary  $y_i$  only keeps some bits of the output values  $x_i$  in order to remove possible correlation between successive values. Therefore the function  $g$  is typically a projection.

Definition 3.2 The TCSPRBG inversion problem is the following: given  $y_1||y_2||\dots||y_n$ , find  $x_{n+1}$ .

Definition 3.3 The TCSPRBG partial inversion problem is the following: given  $y_1||y_2||\dots||y_n$ , find  $y_{n+1}$ .

Remark 3.1 The idea behind previous definition is the following:

- if we adversaly solve the inversion problem defined by 3.2 we clearly can predicted all the future output of the PRBG, and the security in this case is completaly broken;
- if we adversaly can solve for some values of  $n$  (but not for every  $n$ ) the partial inversion problem it can still predicted part of the future output of the PRBG and security partial broken.

Remark 3.2 The "CS" in "TCSPRBG" stands for cryptographically secure. There can not be an unconditioned proof of this feature (the same is true for the existence of a one-way function, see [12]). All existing security proof are conditioned to conjectures of the following types:

- $P \neq NP$ ;
- on unsolvability of well known problems in less than a certain certain time.

Two examples of TCSPRBG we consider in the present thesis are:

- a variant of the Micali Schnorr RSA PRG studied by Steinfeld, Pieprzyk and Wang in [5];
- QUAD [9],[10].

The strategy that is followed to prove that such PRBGs are computational secure is the following:

- a necessary premise to solve any inversion problem is that sequence  $y_1||y_2..||y_n$  is distinguishable from a random sequence;
- if such sequence (for  $n$  not too large) is distinguishable from a random sequence then some well known hard problem could be solved more effciently than it is actually known.

In order to give an idea of how secure these CSPRBG are, we report the statement of the main security results obtained by these authors respectively for the RSA PRBG and for the QUAD:

Theorem 3.1 For all  $n \geq 2^9$ , any  $(T, \delta)$  distinguisher  $D$  for  $(n, e, r, l)$ -RSAPRG can be converted into a  $(T_{INV}, \epsilon_{INV})$  inversion algorithm  $A$  for the  $(n, e, r, w)$ -CopRSA problem (with  $w = 3 \log(2l/\delta) + 5$ ) with:

$$T_{INV} = C_S(T + O(l/r \log(e)n^2)) \quad (1)$$

where:

$$C_S = 64(l/\delta)^2 n \log(n)$$

and:

$$\epsilon_{INV} = \delta/9 - 4/2^{n/2}$$

Theorem 3.2 Let  $L = \lambda(k-1)n$  be the number of key-stream bits produced by in time  $\lambda T_S$  using  $\lambda$  iterations of our construction. Suppose there exists an algorithm  $A$  that distinguishes the  $L$ -bit key-stream sequence associated with a known randomly chosen system  $S$  and an unknown randomly chosen initial internal state  $x \in \{0, 1\}^n$  from a random  $L$ -bit sequence in time  $T$  with advantage  $\epsilon$ . Then there exists an algorithm  $C$ , which given the image  $S(x)$  of a randomly chosen (unknown)  $n$ -bit value  $x$  by a randomly chosen  $n$ -bit to  $m$ -bit quadratic system  $S$  produces a preimage of  $S(x)$  with probability at least  $\frac{\epsilon}{2^{3\lambda}}$  over all possible values of  $x$  and  $S$  in time upper bounded by  $T'$ .

$$T' = \frac{2^7 n^2 \lambda^2}{\epsilon^2} \left( T + (\lambda + 2)T_S + \log \left( \frac{2^7 n \lambda^2}{\epsilon^2} \right) \right) + \frac{2^7 n \lambda^2}{\epsilon^2} T_S$$

A very rough interpretation of these theorems can be given us follows:

- theorem 3.1 states that if the sequence produced by RSAPRBG is distinguishable (in a certain time) from a random sequence then an "RSA type" problem (the precise definition of  $(n, e, r, w)$ -CopRSA is given in definition 4.2 of [5]) could be solved in less time than the best known attack, i.e. the Coppersmith attack. For the choice  $n = 6144$  this implies that an output up to  $2^{32}$  bits should be secure given the actual computer technology ( $2^{70}$  instruction needed), see table 1 of [5];
- theorem ?? states that in the sequence produced by QUAD is distinguishable from a random sequence, than a quadratic system in GF2 should be solved in less time than expected by the best known algorithms based on Groebner bases, resulting contradiction ( $n > 350$ ).

One point in favour of RSAPRBG is that the practical implementation of QUAD is proposed for  $n = 160$ , while one point in favour of QUAD is that the solution of quadratic systems in GF2 is known to be NP-complete.

With regard to the applications of this type of generator it is necessary to take into account the following features:

1. conjectures that are assumed to obtain security results (e.g. intractability of integer factorization can no longer be true if quantum computers will be realized and introduced in the market);
2. parameters for which the security of these families of generators is guaranteed;

3. the maximum bit-length of output sequence that can be made public so that the system continues to be safe;
4. system throughput. In practice this assessment can not be purely theoretical since it is impossible to anticipate all the operations performed by a generic machine (data movement operations, memory management operations,...).

### 3.1 Special TCSPRBG

Now we consider a modification of TCSPRBG. We call it Special TCSPRBG (SCSPRBG for short).

Definition 3.4 A SCSPRBG is defined as follows. Let  $f$  and  $g$  be two function as in the definition 3.1. Let:

$$x_{i+1} = f(p_1, \dots, p_k, x_i)$$

since the parameters  $p_1, \dots, p_k$  are fixed and known usually we write  $f(x_i)$ .

$$y_i = g(x_i)$$

$$z_i = w(y_i)$$

The function  $w$  is a weakly one way function such that:

$$- \#_{bit}(z_i) \gg \#_{bit}(y_i)$$

where  $\#_{bit}(\cdot)$  denotes the number of bits of the sequence. The output sequence of generator is  $z_1 || z_2 || \dots$

Definition 3.5 The TCSPRBG defined above is set correspondent to the SCSPRBG that uses the same function  $f$  and the same function  $g$ .

We say that a TCSPRBG and a SCSPRBG are correspondent if they use the same function  $f$  and the same function  $g$ .

Definition 3.6 The SCSPRBG inversion problem is defined as follows: given  $z_1 || z_2 || \dots || z_n$  find  $x_{n+1}$ .

Definition 3.7 The SCSPRBG partial inversion problem is defined as follows: given  $z_1 || z_2 || \dots || z_n$  find  $y_{n+1}$ .

Remark 3.3 The SCSPRBG inversion problems are obviously not easier than the inversion problems for the correspondent TCSPRBG.

Definition 3.8 The SCSPRBG sub-partial inversion problem is defined as the following: given  $z_1 || z_2 || \dots || z_n$  and a subset of the bits of  $z_{n+1}$ , find  $z_{n+1}$ .

Remark 3.4 This problem can be much easier than the previous if  $w$  is not carefully chosen. To understand this, since the number of bits of  $z(n+1)$  is larger than the number of bits of  $y(n+1)$ , a partial knowledge of  $z(n+1)$  could be sufficient to determine  $y(n+1)$  and to compute  $z(n+1) = w(y(n+1))$ . Moreover the previous definition emphasizes that if one adversary can solve the sub-partial inversion problem he actually gains some information.

Remark 3.5 On the other side if one adversary is not able to solve the inversion and partial inversion problem for the SCSPRBG it is reasonable to assume that the sub-partial inversion problem is not easier than the inversion problem for the function  $w$  (i.e. given  $z(n+1)$ , find  $y(n+1)$ ). Therefore if we build an SCSPRBG correspondent to a TCSPRBG (for example correspondent to RSAPRBG or QUAD) we only study the difficulty of the inversion of function  $w$ .

Remark 3.6 The throughput rate of a SCSPRBG can be much higher than the throughput of correspondent TCSPRBG, due to use of the weakly one way function  $w$ .

Next section will be devoted to find good candidate for the choice of function  $w$ .

## 4 The function pick $(k, m)$

Definition 4.1 Let  $k, m$  be numerical and let  $l = m \cdot 2^{k-1}$ . Let now  $M$  be a public known matrix of random bits with  $l$  rows and  $2l$  (i.e.  $m \cdot 2^k$ ) columns. We call pick  $(k, m)$  the following function that transforms a string of  $mk$  bits in a string of  $l$  bits. The input of  $mk$  bits is divided in  $m$  segments of  $k$  bits. The  $m \cdot 2^k$  columns of  $M$  are arranged in  $m$  blocks of  $2^k$  columns each. Each segment of  $k$  bits is used to choose ("pick") column among the  $2^k$  columns in the each block.

The output string is the xor of the chosen columns (and for each block).

Remark 4.1 The ratio between the length of the output string and the input one is obviously  $2^{k-1}/k$ .

Remark 4.2 Let assume that we have a TCSPRBG that, at each iteration outputs  $m \cdot k$  bits. Then, if the pick  $(k, m)$  function turns out to be a weakly one way function than it can be used as the function  $w$  in the definition 3.4.

Definition 4.2 The pick  $(k, m)$  inversion problem is defined as follows: given the matrix  $M$  and the output string find the input string.

We give some estimate of the difficulties of this inversion problem. We first show that to solve the inversion problem is equivalent to solve a system with  $m \cdot 2^k$  unknowns,  $l + m$  linear equations and  $m \cdot \binom{2^k}{2}$ .

Let  $a_{h,i,j}$  the element of the Matrix  $M$  of row  $1 < h < l$  and column  $0 < j < 2^k - 1$  in the block  $1 < i < m$ . We denote with  $x = (j_1, \dots, j_m) \in (\mathbb{Z}_{\neq \tau})^m$  the input string and  $y = (y_1, \dots, y_l) \in (\mathbb{Z}_{\neq \tau})^m$  the output string. There exist  $m$  linear relationship:

$$\sum_{i=1}^m \sum_{j=0}^{2^k-1} a_{h,i,j} \cdot q_{i,j} = y_h \quad (2)$$

with  $1 \leq h \leq l$  and where  $q_{i,j}$  are  $m \cdot 2^k$  variables in  $\mathbb{Z}/2$ . We have for all  $1 \leq i \leq m$ :

$$q_{i,j} = 0 \quad j \neq j_i \quad j = j_i$$

Therefore  $\forall 1 \leq i \leq m$  an  $q_{i,j}$  is equal to 1.

This corresponds to accompany the system of linear equations ref eqLin with the following quadratic equations:

$$q_{i,j}q_{i,l} = 0 \quad 0 \leq j \leq l \leq 2^k - 1 \quad \sum_{j=0}^{2^k-1} q_{i,j} = 1 \quad 1 \leq i \leq m \quad (3)$$

We see that an iteration of the QUAD can be traced back to a case in a standard way (much) of a particular iteration of the problem pick. For simplicity we consider only iterations QUAD doubling the number of bits (i.e.  $k = 2$ ). In the QUAD iteration we have  $n$  bits and a matrix with  $2n$  rows and  $1 + n + \binom{n}{2}$  columns. In this case we call  $x = (x_1, \dots, x_n) \in (\mathbb{Z}/2)^n$  the input string and  $y = (y_1, \dots, y_{2n}) \in (\mathbb{Z}/2)^{2n}$  the output string; the relationship between the two strings are given by the  $2n$  equazioni quadratiche:

$$\sum_{1 \leq i < j \leq n} a_{i,j}^h x_i x_j + \sum_{i=1}^n b_i^h x_i = y_h$$

with  $a_{i,j} \in A$ ,  $b_i \in B$ , where  $A$  is the matrix with  $2n$  rows and  $\binom{n}{2}$  columns and  $B$  a matrix with  $2n$  rows and  $n$  columns (for simplicity we don't consider the columns  $c_i^h$ ). We suppose that  $n$  is divisible by  $2k$  (where  $k$  is the variable in the pick definition). Let:

$$m = \frac{n}{2k} + \frac{2n}{2k} \left( \frac{n}{2k} - 1 \right) = \frac{n(n-k)}{2k^2}$$

We divide the index  $1, \dots, n$  in  $\frac{n}{2k}$  blocks  $\sigma_1, \dots, \sigma_{\frac{n}{2k}}$  di  $2k$  indeces. Every block is divided in two sub-blocks with the same number of indeces such that  $\sigma_i = \tau_i \cup \phi_i$ .

$$\begin{aligned} & \sum_{r=1}^{n/(2k)} \left( \sum_{i \in \sigma_r} b_i^h x_i + \sum_{i < j \text{ con } i, j \in \sigma_r} a_{i,j}^h x_i x_j \right) + \\ & + \sum_{1 \leq r < s \leq k}^{n/(2k)} \left( \sum_{i \in \sigma_r} \sum_{j \in \sigma_s} a_{i,j}^h x_i x_j \right) + \\ & + \sum_{1 \leq r < s \leq k}^{n/(2k)} \left( \sum_{i \in \sigma_r} \sum_{j \in \tau_s} a_{i,j}^h x_i x_j \right) + \\ & + \sum_{1 \leq r < s \leq k}^{n/(2k)} \left( \sum_{i \in \tau_r} \sum_{j \in \sigma_s} a_{i,j}^h x_i x_j \right) + \end{aligned}$$

$$+ \sum_{1 \leq r < s \leq k}^{n/(2k)} \left( \sum_{i \in \tau_r} \sum_{j \in \tau_s} a_{i,j}^h x_i x_j \right) +$$

We fix  $r$ , the result of sum ( or xor) depends by only values of  $\{(x_i), i \in \sigma_r\} \in (\mathbb{Z}/2)^{2^k}$ . therefore we can rewrite:

$$\sum_{r=1}^{n/(2k)} \sum_{j=1}^{2^k} a_{h,r,j} \cdot q_{r,j}$$

where  $a_{h,r,j}$  are appropriate values and for all  $1 \leq r \leq \frac{n}{2k}$  one of  $q_{r,j}$  is equal 1. Likewise each of four sum are rewrite as follows:

$$\sum_{r=\frac{\rho_1}{2k}+1}^{\rho_1+\frac{n}{2k}} a_{h,r,j} q_{r,j} \dots \sum_{r=\rho_3\frac{n}{2k}+1}^{\rho_4+\frac{n}{2k}}$$

where  $\rho_{s+1} - \rho_s = \frac{n}{2k} \left( \frac{n}{2k} - 1 \right)$  where  $\rho_0 = 0$  e  $s = 1, 2, 3, 4$  and where for all values of  $r$  exists a only  $j = j(r)$  tale che  $q_{r,j}$ .

## 5 Efficient encryption method based on a SCSPRBG

In this section we present how to use a TCSPRBG or a SCSPRBG encrypt communications.

Suppose two users, Alice and Bob, want to communicate securely. The communication between two users can be of two types:

- one-way communication
- two-way communication

In an one way communication only one of the two users can send the message and the other can only receive it, while in a two-way communication each user can both transmit and receive messages.

Moreover a two way-communication can be:

- symmetrical (it is a communication system in which the speed or quantity of data is the same in both directions, averaged over time, i.e. telephone);
- asymmetrical (it is a communication system in which the data speed or quantity differs in one direction as compared with the other direction, averaged over time, i.e. ADSL).

The encryption method works in both cases in almost the same way.

For simplicity's sake we consider an one-way communication: suppose Alice want to send a message  $M$  to Bob. Both use the same TCSPRBG (or SCSPRBG) characterize by a private seed,  $x_0$ , and by some known parameters,  $p_1, \dots, p_k$ .

Remark 5.1 In the case of two-way communication, the two users must generate two sequence TCSRPG (or SCSPRPG), one used to send a message from Alice to Bob, the other used to send a message from Bob to Alice.

Then first two users exchange the seed securely (in fact it must be known to only two users). Now Alice and Bob parallel generate a long sequence TCSRPG (or SCSPRPG), denoted by  $S$ . The  $S$  sequence is managed in FIFO mode (first in, first out), i.e. the first bits products are used to encrypt messages while the new bits products are positioned in the line.

They use pieces of  $S$  sequence, denoted with  $PRB_i$ , as keys for symmetric key encryption algorithm which will protect the communication (note that a symmetric encryption algorithm is computationally more advantageous than a public key encryption algorithm).

Suppose Alice wishes to send to Bob the message  $M$  securely. The message  $M$  is divided into  $k$  sequences:

$$M = M_1 || M_2 || \dots || M_k$$

each sequence  $M_i$  is encrypted with a piece  $PRB_i$  of the sequence  $S$  (having the same bit-length of  $M_i$ ) through a xor operation:

$$ALICE : enc(M_i) = M_i \oplus PRB_i \quad \forall i = 1, \dots, k$$

Alice sends the encrypted message to BOB. He is able to decrypt it through another simple xor operation because he already has the sequence  $PRB_i$ :

$$BOB : dec(enc(M_i)) = enc(M_i) \oplus PRB_i = M_i \quad \forall i = 1, \dots, k$$

Encryption and decryption time are negligible. Moreover the parallel computation of the chunk  $PRB_i$  that we used to xor the message  $M_i$  can be done before its use.

Remark 5.2 This scheme amounts to consider the output sequence of a TCSRPG (SCSPRPG) as a key-stream for a stream cypher.

Remark 5.3 No padding schemes are required for the message  $M$  and such type of attacks can not be constructed.

Remark 5.4 A possible drawback of the scheme is that the actual bit production rate can be very different for Alice and Bob.

Remark 5.5 The scheme works until the rate with the bits are produced and added to the sequence  $s(x_0)$  is greater than the rate at which the bits are taken to encrypt (decrypt) messages.

This method is secure if:

- the key  $PRB_i$  is used only once;
- not a single value  $x_i$  can be fully recovered by an attacker;
- the sequence  $S$  is a PRBG.

Therefore it is necessary that Alice and BOB communicate through a *CSPRBG*.

Two users must share a common seed  $x_0$  to generate the pseudo-random sequence  $S$  both for a *TCSPRBG* that for a *SCSPRBG*. There are two problems:

1. to exchange the seed  $x_0$  securely;
2. to preserve the seed  $x_0$  securely until it is used.

A possible solution for the first problem is that users exchange the seed personally. Another solution is to use a public encryption algorithm (as RSA) to exchange the seed. If the seed is exchanged through RSA algorithm it is secure if:

- quantum computers not exist;
- authentication is made.

Remark 5.6 If parts of the seed are encrypted via RSA algorithm and sent in a cross-way between Alice and Bob we can observe that attacker should break both RSA-keys of Alice and Bob.

Remark 5.7 In the present work we don't consider in detail the following problems:

- how to produce and share truly random seed between Alice and Bob;
- authentication problems (Man in the middle);
- moreover we only generically address the problem of secret data protection for Alice and Bob.

Remark 5.8 We prefer to use RSA algorithm rather than elliptic curves for various practical reasons:

- for simplicity's sake;
- RSA cryptanalysis has been most studied than that of elliptic curves.
- we believe that the actual implementation of RSA, given its simplicity, can be less prone to fatal mistakes.

The second problem can be solved generating and exchanging the seed shortly before its use. In this case all stages of generation of the seed must be protected too.

Remark 5.9 In this work we will not consider in details these two problems or the problem to generate really random sequences. We will only focus on the use and efficiency of a *TCSPRBG* and a *SCSPRBG*.

## 6 An example of an efficient SCSPRBG

In this section we want to give an example of an SCSPRBG for which can be given some security results. Since the inversion problem (see def.3.6) for a SCSPRBG is not easier than the inversion problem for the correspondent TCSPRBG, the security results that are valid for the correspondent TCSPRBG still hold.

The hardness of SCSPRBG partial inversion problem (see def. 3.8) is directly connected with the choice of the function  $w$ . We can think of many examples to construct the  $w$  function as we require that it is only a weakly one-way function. We outline one example that was inspired to us by QUAD.

The crypto-analysis of such example is far from complete, but through it we want to illustrate a basic rule, in our opinion, to choose the  $w$  function, i.e. the simplicity; in fact if the function structure is relatively simple its safety will be easier to prove. Furthermore in the opposite case it will be easier to build an attack that proves its insecurity, thus prompting the need to change the choice of the function.

### 6.1 Possible $w$ function

We suppose to have a system of  $n$  linear equations in  $2n$  unknowns in  $GF(2)$ , with  $n$  sufficiently large (each coefficient is represented by a bit).

We suppose:

$$2n = 2^l = 2^h 2^{l-h}$$

with  $h < l$ .  $a_{ij}$  denotes the  $j$ -th coefficient of  $i$ -th equation;  $A^j$  denotes the  $j$ -th column of the matrix  $A$ .

We split up the column in  $2^{l-h}$  blocks, each containing  $2^h$  columns. Now we choose one column for each block and compute the xor of the selected columns:

$$A^{i_1} \oplus \dots \oplus A^{i_s} \tag{4}$$

where  $s = 2^{l-h}$  and  $i_j = (j-1)2^h + 1 + r_j$  where  $r_j$  is an integer such that  $0 \leq r_j \leq 2^{h-1}$ .

Then to make that choice, we must specify  $r_j$  for  $1 \leq j \leq 2^{l-h}$ ; each  $r_j$  consists of  $h$  bits, totalling  $h2^{l-h}$  bits. This sequence represents the input of the  $w$  function and the result of the operation 4 represents the output.

Before making an assessment of the safety of such system, we consider its computational efficiency. The system requires  $\frac{2^{l-h}-1}{d}$  xor/bit where  $d$  is the number of bits of a word (i.e.  $d = 32$  or  $d = 64$ ). The multiplication factor  $R$  is the ratio between the bit length of output sequence and the bit length of the input sequence:

$$R = \frac{2^{l-1}}{h2^{l-h}} = \frac{2^{h-1}}{h}$$

**Remark 6.1** We consider a numerical example. if  $2n = 8192$  and  $h = 6$  we need only 4 xor/bit for  $d = 32$ , and we obtain  $R = 32/6$ .

The computation of  $w$  is then at least 25 times faster than QUAD (see equation ??). To compute the speed of the SCSPRBG we have to take into a count the time spent for the iteration of  $f$  (the time spent in the computation of  $g$  is negligible).

We suppose that an iteration of CSPRBG (considering only the function  $f$  and  $g$ ) costs  $T_1$  cycles/bit and that the function  $w$  costs  $T_2$  cycles/bit with  $R$  as multiplication factor. Each bit of first iteration generates  $R$  bit used for  $w$  function; therefore the cost is:

$$\left(\frac{T_1}{R} + T_2\right) \text{ cycles/bit} \quad (5)$$

In the case of RSAPRG authors declare a value  $T_1 \sim 3000$  cycles/bit (see [5]) while in the case of QUAD it is reasonable to estimate 200-400 cycles/bit. Therefore even if we use QUAD as correspondent TCSRPG we can neglect  $T_2$  with respect to  $T_1/R$  and we can conclude that the speed of our system is roughly 5 times greater than QUAD.

**Remark 6.2** We want to compare the amount of memory required by the system described above rather to a MQ system. For instance for  $n = 8192$ , a system of this type requires a memory of  $\sim 4$  Mbytes. We remember that a MQ system with 320 equations in 160 unknowns requires 1/2 Mbytes of memory. Therefore in our case we need a storage space a 8 times bigger than a MQ system.

**Remark 6.3** We can iterate the function  $w$  a few times. This can be done in order to get a higher speed, even if  $j$  is rather low, e.g.  $j = 2$  or  $j = 3$ , we can achieve a much higher speed than the correspondent TCSRPG. Such speed can be computed in a similar way to 5, but in any case will obviously be bounded by  $T_2$  cycles/bit.

## 6.2 Crypto-analysis considerations

We have just started the study of the proposed system above, but we want to mention some considerations for completeness.

We analyzed is how difficult it is to invert  $w$ ; more precisely the problem is the following:

- given  $A^{i_1} \oplus \dots \oplus A^{i_s}$ , find the sequence  $r_1 || \dots || r_s$ .

This problem does not correspond to the definition of partial inversion problem we give (see def. 3.8) but it seems to us that its computational complexity is a good indicator for the computational complexity of the partial inversion problem.

We observe that the choice of exactly one column of each block of length  $2^h$  corresponds to the addition of the following equation for each block:

$$\sum_{i \in \text{block}} x_i = 1 \quad (6)$$

$$x_i x_j = 0 \quad \forall i, j \in \text{block} \quad (7)$$

The equations 6 are  $2^{l-h}$  linear equations (one for each block) and 7 are  $2^{l-1}(2^h - 1)$  quadratic equations. We observe that the equations 7 are not independent. Equations (7) can be replaced by the following minimal set of quadratic equations for each block:

$$x_i x_j = 0 \quad \forall i, j \in \text{block} : i + j = 0 \text{ in } GF(2) \quad (8)$$

Equations 8 give that in a single block there is at most one  $x_i$  equal to 1 when the index  $i$  is odd and at most one  $x_i$  equal to 1 when the index is even. Then the equation 6 gives us that exactly one  $x_i$  is equal to 1 for each block. The total number of equations 8 is  $2^{l-1}(2^{h-1} - 1)$ . Therefore we get an MQ system with this particular structure:

- $2^l$  unknowns;
- $2^{l-1} + 2^{l-h}$  linear equations;
- $2^{l-1}(2^{h-1} - 1)$  quadratic equations.

The best way to reduce the complexity of the system seems to us to use Gauss elimination to possible eliminate  $2^{l-1} + 2^{l-h}$  unknowns we will then be left with a system with the following structure:

- $2^{l-1} - 2^{l-h}$  unknowns;
- $2^{l-1}(2^{h-1} - 1)$  equations of second degree.

### 6.3 Three strategies to find the solution of the system

Now we want to illustrate three strategies to find the solution of our system. We assume that we are left with the first  $2^{l-1} - 2^{l-h}$  unknowns.

#### 6.4 First strategy

A straightforward one is the following:

1. For each block of left unknowns we have to choice one variable to be put equal to one. This gives  $2^{h(2^{l-h-1}-2^{l-h})}$  possibilities.
2. To compute the other variables and verify if they too respect the equations 8; in this case (very probably) this is the solution of the system;
3. Otherwise to go to step 1) until we find the solution.

For clarity, we consider the numerical example of the remark 6.1 . If  $n = 4096$  and  $h = 6$ , the step 1) given  $2^{372}$  choices. The step 3) requires the xor of 61 words to compute a number of unknowns equal to 32 (64 if the machine works in 64 bits).

It is very likely that the values of these first 32 unknowns do not respect the 8. Therefore we can estimate, although not accurately, the expected time to find the solution in  $5,9 \cdot 10^{113}$  machine cycles.

#### 6.5 Second strategy

The third strategy is consisted in solving the system applying the XL relinearization algorithm by [17]. In the case of the numerical example of remark 6.1 this algorithm produces a system of  $\left( 3\,968 \right.$

12) linear equations, obtaining a computational complexity of  $\left( 3968 \right)^{2,37} = 5,7 \cdot 10^{81}$ . So this strategy seems to be the best one.

We insisted that the particular structure of equations 8 can speed up above strategies, nevertheless the estimates seem to be quite reassuring.

### 6.6 Third strategy

Another possible strategy is to use linear equations to eliminate some of the variables  $x_i$  and derive the remaining  $x_i$  by using quadratic equations. In this case we have 8192 unknowns and 4224 (4096+128) linear equations. Therefore we can delete (most) 4224 unknown. We remain with 3968 unknown and  $64 \times 63 \times 64$  quadratic equations. We use the Brdet estimate. In this case  $k \simeq 65$  and  $D \simeq 7.6$ . Let  $D=7$  the estimate operations number to solve the system is  $\left( 3968 \right)^{2,37} \simeq 8.4 \cdot 10^{50}$  that is less of the previous strategies.

### 6.7 Fourth strategy

Another strategy to solve the system is probabilistic type. More precisely we can fix arbitrarily in each of the 128 blocks by 64 variables, 31 variables equal. Now we have a sufficient number of equations to determinate all variables. It is easy to controll if a variable is equal 1. The success probability is equal to  $\left( \frac{31}{64} \right)^{128} = 1,5 \cdot 10^{-37} = (6,6) \cdot 10^{36})^{-1}$ .

If the computational cost of a single verification is  $10^4$ , we obtain a probability to solve the system equal to  $\simeq 1 - 1/e$  with a computational complexity of  $6,6 \cdot 10^{40}$ .

## References

- [1] A. Menezes, P: van Oorschot, S: Vanstone. Handbook of Applied Cryptography, 1996.
- [2] Aviezri S. Fraenkel and Yaacov, Complexity of solving algebraic equations. Inf. Process. Lett., 10(4/5):178-179,1980.
- [3] Michael R. Garey and David S. Johnson. Computers and Intractability: A guide to the Theory of NP-Completeness , chapter 7.2 Algebraic Equations over  $GF(2)$ . W H Freeman & Co, 1979.
- [4] Jacques Patarin and Louis Goubin. Asymmetric cryptography with s-boxes. In ICICS, pages 369-380, 1997.

- [5] Ron Steinfield and Josef Pieprzyk and Huaxiong Wang. On the Provable Security of an Efficient RSA-Based Pseudorandom, 2006.
- [6] R. Fischlin and C.P. Schnorr. Stronger Security Proofs for RSA and Rabin Bits, 1999.
- [7] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. Cryptology*, 10:223-260, 1997.
- [8] A.K. Lenstra. Unbelievable Security: Matching AES Security Using Public Key Systems. In *Asiacrypt 2001*, volume 2248 of LNCS, pages 67-86, Berlin, 2001. Springer-Verlag.
- [9] C. Berbain, H. Gilbert and J. Patarin. QUAD: A Practical Stream Cipher with Provable Security. S. Vaudenay(Ed.): *EUROCRYPT 2006*, LNCS 4004, pp. 109-128, 2006.
- [10] C. Berbain, H. Gilbert and J. Patarin. QUAD: A multivariate stream cipher with provable security, 2009.
- [11] Richard P. Brent and Paul Zimmermann: *Modern Computer Arithmetic*, October 2010.
- [12] Oden Goldreich: *Foundations of Cryptography. Basic Techniques*, 2003.
- [13] Gregory V. Bard: *Algebraic Cryptanalysis*, 2009.
- [14] Magali Bardet. *Étude des systèmes algébriques surdéterminés. applications aux codes correcteurs et à la cryptographie*. PhD, Université Paris IV, 2004.
- [15] Nicolas Courtois, Louis Goubin, Willi Meier, and Jean-Daniel Tacier. Solving underdefined systems of multivariate quadratic equations. In *Public Key Cryptography*, pages 211-227, 2002.
- [16] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, 1997.
- [17] Marc Joye (Ed.): *Topics in Cryptology-CT-RSA 2003*
- [18] Karatsuba, Anatolii A. and Ofman, Yuri. 1962. Multiplication of multi-digit numbers on automata (in Russian). *Doklady Akad. Nauk SSSR*.
- [19] Toom, Andrei L. 1963. The complexity of a scheme of functional elements realizing the multiplication of integers (in Russian).
- [20] A. Corbo Esposito, F. Didone: Some improvements for the implementation of subquadratic algorithms for modular reduction, (pre-print 2012).
- [21] A. Corbo Esposito, F. Didone: Non recursive implementation of Karatsuba algorithm for multiplication and square, (pre-print 2012).

- [22] A. Corbo Esposito, P. Di Meo, F. Didone, V. Massaro: Numerical results for implementation of integer multiplication and modular reduction in Java big integer class, Internal report university Cassino e del Lazio Meridionale, 2012.
- [23] A. Corbo Esposito, F. Didone: Sviluppo di un metodo di crittazione crittograficamente sicuro con tempo di latenza nullo, Tesi di laurea specialistica in Ingegneria delle Telecomunicazioni, 2009.