Cryptographically Secure Pseudo-Random Bit Generator for RFID Tags

Pedro Peris-Lopez
Security Lab
Faculty of EEMCS
Delft University of Technology
Delft, The Netherlands
P.PerisLopez@tudelft.nl

Enrique San Millán
Depart. of Electrical Engineering,
University Carlos III of Madrid
Leganés, Spain
quique@ing.uc3m.es

Jan C.A. van der Lubbe
Security Lab
Faculty of EEMCS
Delft University of Technology
Delft, The Netherlands
J.C.A.vanderLubbe@tudelft.nl

Luis A. Entrena
Depart. of Electrical Engineering,
University Carlos III of Madrid
Leganés, Spain
entrena@ing.uc3m.es

Abstract

In RFID protocols, random numbers are mainly required to anonymize tag answers in order to guarantee the privacy of the owner of the transponder. Our analysis looks at the feasibility of RFID tags for supporting Cryptographically Secure Pseudorandom Number Generators (CS-PRNG) on their limited chip. Specifically, we study the implementation of the Blum-Blum-Shub (BBS) pseudorandom number generator for security levels 2^{32} (160 bits) and 2^{64} (512 bits) respectively, these values being suitable for many RFID applications but not for standard security applications.
Keywords– *RFID, privacy, randomness, CS-PRNG, BBS*

1. Introduction

Radio Frequency Identification (RFID) is an automatic identification technology in which a small transponder (tag), attached to an object (i.e. person, animal or product), receives and responds to radio-frequency queries from a transceiver (reader). Tags usually respond with a constant value which facilitates their association with their holders. An attacker may track a user's movements, putting location privacy at risk. The inclusion of random numbers in tag answers may deter such attacks. In reality, however, mechanisms for random generation are often not as well designed as could be expected [7].

2. Blum-Blum-Shub

Cryptographically Secure Pseudorandom Number Generators (CS-PRNG) fall into two main groups: 1) those based on standard cryptographic primitives; 2) those based upon mathematical problems thought to be hard (i.e there is no algorithm bounded by a running time of the form $O(n^k)$). In this paper, we focus on the Blum-Blum-Shub (BBS) generator whose security depends on the fact that integer factorization is generally assumed to be intractable [2]. A pseudorandom bit sequence $\{z_1, z_2, ..., z_l\}$ of length l is generated as described below:

1. Setup

1.1. Generate two large secret random (and distinct) primes p and q each congruent to 3 modulo 4, and compute $n = p \cdot q$.

1.2. Initialize the seed s randomly in the interval $[1, n-1]$ such that $gcd(s, n) = 1$ and compute $x_0 = s^2 \mod n$.

2. Random Bit Generation

For i from 1 to l:

2.0 $x_i = x_{i-1}^2 \mod n$

2.1 $z_i =$ least significant bit of x_i

The generation of each pseudo-random bit $\{z_i\}$ requires one modular squaring. We can increase the efficiency of

the generator by extracting the j least significant bits of x_i (step 2.1), where $j = c \cdot \lg \lg n$ and c is a constant. This version of the generator is also cryptographically secure if n is sufficiently large [9].

2.1. Motivation

In general terms, the hardware demands and temporary requirements of this generator, mainly associated with the use of modular multiplications, make the primitive inadequate for constrained RF devices. The assertion is correct when we demand the same security for RFID applications as for conventional cryptographic applications (i.e. electronic banking). Following NIST recommendations, a module n of 2048 bits length has to be used for cryptographic applications from 2009 to 2010 [4]. For that length of module, BBS implementation far exceeds the capabilities of most RF transponders (e.g. contacless smart cards, RFID tags, etc.). However, practical security values in the range of $[2^{32}, 2^{64}]$ are adequate for many RFID applications. For example, EPC Class-1 Generation-2 RFID tags, which are widespread around the world and read in their millions every day, can offer a maximum security level of 2^{32} [5].

As mentioned previously, the security of BSS is linked to integer factorization problem. The expected running time of the elliptic curve factoring algorithm in the hardest case, when n is a product of two primes (p and q) of roughly the same size, is $L_n[1/2, 1]$. Alternatively, the quadratic sieve algorithm may be used, having the same expected running time but being much more efficient [10].

Definition Let $\mathcal{A}$ be an algorithm whose inputs are either elements of a finite field F_q, or an integer q. $\mathcal{A}$ is a subexponential-time algorithm when its expected running time is of the form:

$$L_q[\alpha; c] = O(e^{((c+o(1))(\ln q)^{\alpha}(\ln \ln q)^{1-\alpha})}) \qquad (1)$$

where c is a positive constant, and α is a constant satisfying $0 < \alpha < 1$.

From Equation 1, and fixing $\alpha = 0.5$ and $c = 1$, a length of 160 bits or 512 bits is required for satisfying a security level of 2^{32} or 2^{64} respectively. These values are significantly less demanding than NIST recommendations which are suitable for conventional cryptographic applications. However, RFID applications are often less demanding in terms of security, due to the need to achieve a balance between operativity, price and security.

Main Contribution: This paper makes a detailed analysis of the feasibility of implementing the BBS generator for 160 bits and 512 bits length. It is the first –to the best of our knowledge– to discuss the use of CS-PRNG in RFID tags by which security levels are relaxed in comparison with

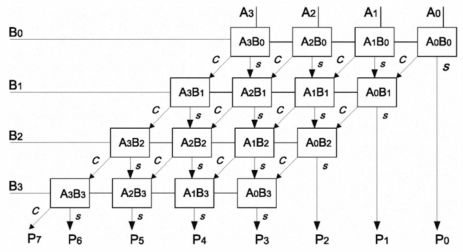


Figure 1. Combinational Multiplier Architecture

standard cryptography whilst maintained at a suitable level ($[2^{32}, 2^{64}]$) for intended applications. Specifically, different alternatives have been considered with the aim of optimizing the resources consumed. As the end of our study, the reader is provided with an estimation of the circuit area required and the time consumed in each generation for the four architectures analyzed.

3. Modular Multiplication

The implementation of the BBS algorithm is based on modular squaring, which can be computed by modular multiplication. Different algorithms have been proposed to obtain the modular multiplication of two numbers ($A \times B \mod M$) [14].

Specifically, we study the hardware implementations for the following algorithms: 1) Classical combinational multiplier, with Barrett's reduction; 2) Classical shift and add multiplication, with Barrett's reduction; 3) Karatsuba's multiplication, with Barrett's reduction; 4) Montgomery's direct modular multiplication, iterative architecture.

3.1. Classical combinational multiplication

The fastest implementation of a multiplier is achieved using combinational logic only. Considering the partial multiplications that are needed, the multiplier can be implemented with logic and gates for each partial multiplication, and adders to perform the addition of the partial multiplications. The reader should note that this approach has the disadvantage that the multiplier requires a lot of hardware. The architecture required for 4-bit unsigned binary numbers is shown in Fig. 1.

3.2. Classical shift and add multiplication

To simplify the hardware necessary for partial products, these can be stored in a registry, and multiplications can be obtained in several clock cycles. To optimize the number of

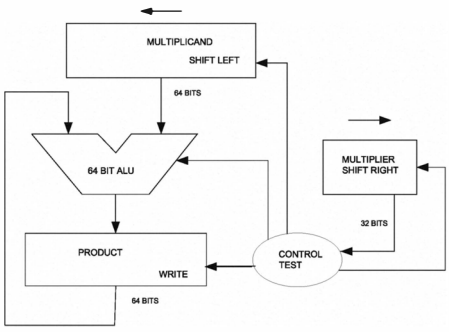


Figure 2. Shift and Add Multiplier Architecture

clock cycles consumed, the partial multiplications considered are the multiplicand by each bit of the multiplier (i.e. there are as many partial multiplications as bits had by the multiplier). As multiplication by two with binary numbers can be implemented as a left shift, the partial multiplications can be implemented with just a shift left register. An adder is necessary for the addition of the partial results, and some control logic completes the architecture of this multiplier as illustrated in Fig. 2. This sequential multiplier uses very few hardware resources, but it is slow because many clock cycles are consumed, especially if we consider big operands.

3.3. Karatsuba-Ofman's

Karatsuba-Ofman's [8] is a much more efficient algorithm in terms of its area/time factor. It is based on a divide-and-conquer strategy. A $2n$-digit integer multiplication is reduced to two n-digits multiplications, one $(n+1)$-$digits$ multiplication, two n-digits subtractions and two $2n$-digit additions.

We consider the product $X \times Y$ of 2 integers, X and Y. These integers can be split into two halves (i.e. X_H, X_L, and Y_H, Y_L). Let n be the number of bits of each of these halves: $X = X_H \cdot 2^n + X_L$ and $Y = Y_H \cdot 2^n + Y_L$.

The product $P = X \times Y$ can be obtained by computing four n-bit multiplications:

$$\begin{aligned} P &= X \times Y = (X_H \cdot 2^n + X_L)(Y_H \cdot 2^n + Y_L) = \\ &= 2^{2n}(X_H Y_H) + 2^n(X_H Y_L + X_L Y_H) + X_L Y_L \quad (2) \end{aligned}$$

Finally, the computation of Equation 4 can be improved by applying the equivalency shown below:

$$X_H Y_L + X_L Y_H = (X_H + X_L)(Y_H + Y_L) - X_H Y_H - X_L Y_L \quad (3)$$

Summarizing, the $2n$-bits multiplication $(X \times Y)$ can be thus reduced to three n-bit multiplications, $X_L Y_L$ and $(X_H + X_L)(Y_H + Y_L)$ by applying Karatsuba-Ofman's algorithm.

The hardware implementation that we obtain for this algorithm uses a shift and add multiplier (see Fig. 2) for the reduced multiplications. Shift registers are used for the multiplication by 2^{2n} and 2^n, and the rest of the logic can be implemented with combinational logic (i.e. an adder/subtractor and some control logic).

The multiplication algorithms considered above can be converted into modular multipliers by performing a reduction operation after the multiplication. To do this, we can use Barrett's algorithm that has been implemented in hardware too.

3.4. Barret's Reduction

This algorithm is used to obtain the remainder of an integer division [1]. The basic idea of the algorithm consists of successively shifting and subtracting the module until the remainder is non-negative and smaller than the module. Several optimizations can be applied to this algorithm, given that a constant module is going to be used for all the operations in the system. In this case, Barrett's reduction method requires the pre-computation of one parameter $\mu = [\frac{2^{2n}}{M}]$, which does not change as long as the modulo remains constant. The reduction then takes the form $R = N - [[\frac{N}{2^n}]\frac{\mu}{2^n}]M$, which requires two n-bit multiplications and one n-bit subtraction; a total of three multiplications and one subtraction.

Specifically, the following optimized algorithm is employed in our experimentation, which consumes n clock cycles:

```
--------------------------
algorithm Reduction (P, M)
    int R_0 = P
    int N = LeftShift(M, n)
    1: for i = 0 to n
    2:  R_i = R_{i-1} - N;
    3:  if R < 0 then
    4:      R_i = R_{i-1}
    5:  N = RightShift(N)
    return R_i;
end Reduction
--------------------------
```

where P is the result of the multiplication between the n-bits operands A and B and where M is the modulo.

3.5. Montgomery

This algorithm interleaves multiplications and reduction operations to obtain the result of the modular multiplication in just once step [12], unlike the algorithms shown previously, which require two steps, multiplication and reduction. Basically, a series of additions are computed to obtain the modular product. In [13], an iterative sequential architecture and a systolic-based architecture are analyzed to

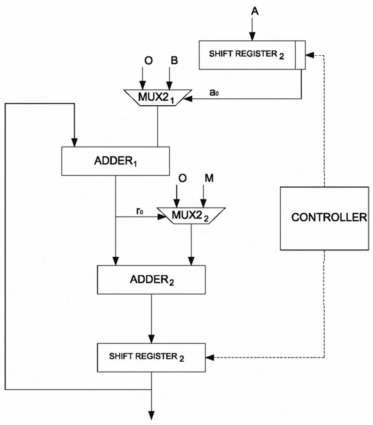


Figure 3. Montgomery Iterative Sequential Architecture

implement the modular multiplication using the fast Montgomery algorithm. We focus on the sequential architecture –displayed in Fig.3– because our operands are smaller than 512 bits.

Let A and B be two arbitrary integers and M the module, where n is their size in its binary representation ($Q = \sum_{i=0}^{n-1} q_i \times 2^i$ for $Q \in \{A, B, M\}$). These values must satisfy two pre-conditions: 1) the module M has to be relatively prime to the base (i.e. $gcd(M, 2) = 1$ for $radix = 2$); 2) the multiplicand A and the multiplier B must be smaller than the module M. The algorithm used in the multiplication is given below:

```
-------------------------------
 algorithm Montgomery (A, B, M)
     int R = 0
     1: for i = 0 to n-1
     2:  R = R + a_i x B;
     3: if r_0 = 0 then
     4:       R = R div 2
     5: else
     6:       R = (R+M) div 2
     return R;
 end Montgomery
-------------------------------
```

The result obtained after the computations of n iterations is $R = A * B * 2^{-n} \bmod M$. An extra Montgomery modular multiplication by the constant $C = 2^n \bmod M$ is thus required to obtain the right result. So the modular multiplication is computed using the algorithm below:

```
---------------------------------
algorithm ModularMult (A, B, M, n)
    const C := 2^n mod M;
    int R := 0;
    R := Montgomery (A, B, M)
    return Montgomery (R, C, M);
end ModularMult
---------------------------------
```

Note that the algorithm is efficient if M is constant, which is typical in cryptographic algorithms that use modular multiplications. In this case, C is a constant that can be pre-computed.

3.6. Hardware Implementation

We study the hardware implementation for the four modular multiplications algorithms described in section above. The different architectures are described in VHDL description language. The design uses generic parameters, so that it can be easily synthesized for different sizes of the variables ($n = \{2^2, 2^3, ..., 2^{10}\}$). The Modelsim tool is used in simulation to test the correctness of the hardware descriptions [11]. The synthesis of the circuits is obtained using Xilinx Foundation [15].

To estimate area and delay, we use a Xilinx programmable device. We chose one with small CLBs (Look-Up Tables) so that correspondence with Gates Equivalent (GE) is as close as possible. A Spartan-3 FPGA device, specifically the XC3S1500L (4-input LUTs), was used for this. Table 1 shows area (LUTs), maximum operation frequency (MHz) and the number of clock cycles consumed in each generation for each architecture and for different bit length of the variables.

Of these four architectures, classical combination architecture can be discarded because of the excessive circuit area demanded for implementation. To facilitate comparison of the remaining alternatives, we calculate the area/time factor at the maximum operating frequency. The results are summarized in diagrams 4a) and 4b) in Figure 4. We observe that Montgomery offers the best results both for low bit lengths and for high bit lengths. According to architectures that require a reduction after the multiplication, shift and add architecture offers better performance than Karatsuba architecture. It consumes the same number of clock cycles and has approximately the same maximum frequency approximately but Karatsuba architecture demands a higher circuit size.

4. Hardware Implementation of Blum-Blum Shub

In Blum-Blum Shub generator, the generation of an output (i.e. one or j random bits) can be obtained with a modular multiplication. So, only a modular multiplier and some basic control logic are needed. In particular, we analyze different architecture implementations of the BBS generator, considering the different modular multiplications discussed in the previous section.

We use the same methodology, software and FPGA as described in Section 3. However, we now focus our analysis on module sizes of 160 and 512 bits exclusively. Additionally, we set operating frequency at 100 KHz, because despite being a value considerably inferior to the maximum

Table 1. Hardware Implementation of Modular Multiplication

n (bits)	Area (LUT)	Fmax (MHz)	Clock Cycles
4	124	154	5
8	291	105	9
16	724	77	17
32	1979	55	33
64	6129	41	65
128	20550	35	129
256	72639	25	257
512	282892	14	513
1024	126220	8	1025

A. Comb. + Barret

n (bits)	Area (LUT)	Fmax (MHz)	Clock Cycles
4	148	168	8
8	299	132	16
16	573	112	32
32	1131	88	64
64	2311	62	128
128	4616	41	256
256	7829	25	512
512	19686	14	1024
1024	36053	8	2048

B. Shift and Add + Barret

n (bits)	Area (LUT)	Fmax (MHz)	Clock Cycles
4	303	168	6
8	579	131	12
16	1102	112	24
32	2221	88	48
64	4366	62	96
128	8422	41	192
256	14107	25	384
512	38978	14	768
1024	76112	8	1536

C. Karatsuba + Barret

n (bits)	Area (LUT)	Fmax (MHz)	Clock Cycles
4	59	122	9
8	115	113	17
16	225	96	33
32	442	88	65
64	880	59	129
128	1756	42	257
256	3484	24	513
512	6966	14	1025
1024	14035	8	2049

D. Montgomery Iterative

Table 2. Hardware Implementation of BBS

		Area (LTUs)	Time (μsec.) @ 100 KHz
Comb. Classic + Barret	160 bits	22,550	1,850
	512 bits	284,892	5,270
Shift and Add + Barret	160 bits	6,176	3,700
	512 bits	21,780	11,840
Karatsuba + Barret	160 bits	10,433	2,900
	512 bits	40,052	9,280
Montgomery Iterative	160 bits	3,052	3,563
	512 bits	9,953	11,200

operation frequency, it is the typical operation frequency of RFID tags [6]. The experimental results are summarized in Table 2.

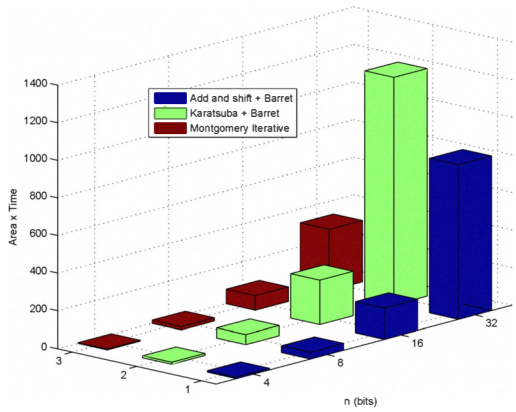
The results show very similar execution times for all the algorithms. This is because the same operation frequency is used for all, and the advantage that some of these algorithms have of being able to work at higher frequencies cannot be exploited. The algorithms that perform the modular multiplication in two steps are disadvantaged by having to use Barrett's reduction which requires about n clock cycles. As the delay is not a differentiated factor, the choice of algorithm will generally depend on the area they occupy. In this respect, we can see that the Montgomery iterative approach produces the best results.

Finally, the chart in Fig. 5 compares the area/time product of the four implementations. Montgomery iterative architecture offers the most efficient results. The next most efficient is the architecture based on shift and add multiplication with Barret reduction which increases this factor by about 50%. The other two approaches are lest efficient in this aspect.

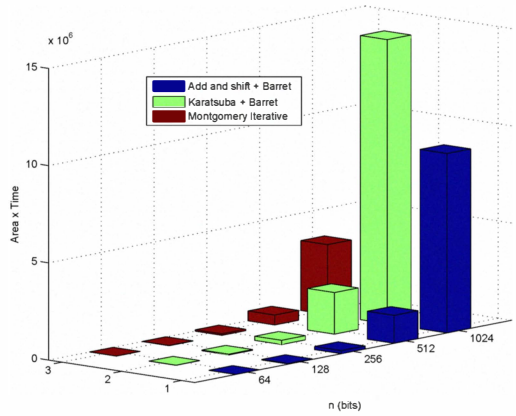
5. Conclusions

In this paper, we examine several architectures for the implementation of the BBS pseudorandom number generator. Generally, the BBS generator (i.e. 768 or 1024 bits) is very demanding in terms of circuitry and computation time. In [3], Blum and Paar presented an efficient FPGA implementation for these module lengths, which proves the excessive circuit area required for their implementation.

However, with regard to security, many RFID applications are not as exigent as standard cryptographic applications. Specifically, a security level of 2^{32} or 2^{64} may be an adequate value for a considerable number of applications. For this reason, in this paper, we analyzed the implementation of the BBS generator with a modulo of 160 and 512 bits respectively. For each one of these modules, we implemented different architectures, measuring the area (LUTs) and the consumed time per generation (μsec.). The deciding factor in the selection of the most convenient architecture is the circuit area because similar execution times are obtained with the different alternatives studied. The Montgomery iterative architecture is the most efficient and we recommend its use for constrained devices such as RFID tags or sensor networks nodes. A comparison of our results with [3] is not possible as the authors used a completely different method of estimation, estimating area by the number of CLB. Additionally, they used a FPGA that has 3 LUTs and some additional logic in each CLB. This FPGA is very different from



(a) Area x Time Factor (4 – 32 bits)



(b) Area x Time Factor (64 – 1048 bits)

Figure 4. Area x Time Factor of Modular Multiplication

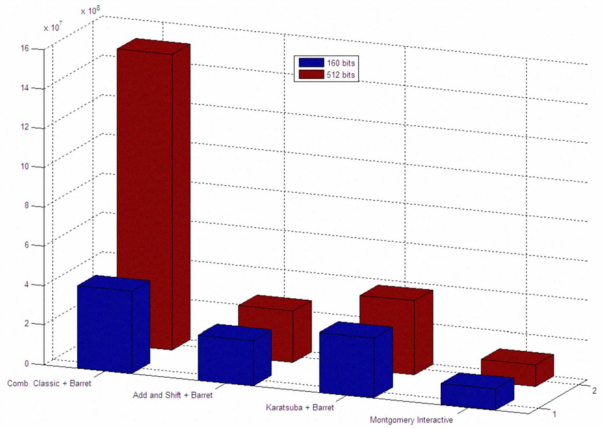


Figure 5. Area x Time Factor of BBS Implementation

ours, which is much smaller an has simpler CLBs. Instead of the number of CLBs, we provide the number of LUTs which is a much more convenient means of estimation of the combinational area of the circuit.

References

[1] P. Barrett. Implementing the Rivest, Shamir and Aldham public-key encryption algorithm on standard digital signal processor. In *Proceedings of CRYPTO'86*, volume 263 of *LNCS*, pages 311–323. Springer-Verlag, 1986.

[2] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986.

[3] T. Blum and C. Paar. Montgomery modular exponentiation on reconfigurable hardware. In *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, pages 70–77. IEEE Computer Society, 1999.

[4] ECRYPT. Yearly report on algorithms and keysizes (2007-2008). D.SPA.28 Rev. 1.1 IST-2002-507932 ECRYPT, National Institute of Technology, 2008.

[5] EPCGlobal. EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860-960 MHz Version. 1.2.0, 2008.

[6] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. In *Proc. on Information Security*, volume 152, pages 13–20. IEEE Computer Society, 2005.

[7] F. D. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Wichers Schreur, and B. Jacobs. Dismantling mifare classic. In *Proc. of ESORICS'08*, volume 5283 of *LNCS*, pages 97–114. Springer, 2008.

[8] D. E. Knuth. *The art of computer programming: seminumerical algorithms*, volume 2, 2nd edition. Addison-Wesley, Reading, Mass, 1981.

[9] A. Menezes, P. v. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, chapter Pseudorandom Bits and Sequences. CRC Press, 1996.

[10] A. Menezes, P. v. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*, chapter Number-Theoretic Reference Problems. CRC Press, 1996.

[11] Mentor Graphics, ModelSim. http://www.model.com.

[12] P. Montgomery. A signed binary multiplication technique. *Modular Multiplication without trial division*, 44:519–521, 1985.

[13] N. Nedjah and L. de Macedo Mourelle. Two hardware implementations for the Montgomery modular multiplication: sequential versus parallel. In *Proceedings of the 15th Symposium on Integrated Circuits and Systems Design*, pages 3–8, 2002.

[14] N. Nedjah and L. de Macedo Mourelle. A review of modular multiplication methods ands respective hardware implementation. *Informatica (Slovenia)*, 30(1):111–129, 2006.

[15] Xilinx, ISE Foundation Software. http://www.xilinx.com.