Pseudo-Random Generator Based on Chinese Remainder Theorem

Jean Claude Bajard and Heinrich Hördegen

LIRMM Univ. Montpellier 2 UMR CNRS, France

ABSTRACT

Pseudo-Random Generators (PRG) are fundamental in cryptography. Their use occurs at different level in cipher protocols. They need to verify some properties for being qualified as robust. The NIST proposes some criteria and a tests suite which gives informations on the behavior of the PRG. In this work, we present a PRG constructed from the conversion between further residue systems of representation of the elements of $GF(2)[X]$. In this approach, we use some pairs of co-prime polynomials of degree k and a state vector of $2k$ bits. The algebraic properties are broken by using different independent pairs during the process. Since this method is reversible, we also can use it as a symmetric crypto-system. We evaluate the cost of a such system, taking into account that some operations are commonly implemented on crypto-processors. We give the results of the different NIST Tests and we explain this choice compare to others found in the literature. We describe the behavior of this PRG and explain how the different rounds are chained for ensuring a fine secure randomness.

Keywords: Pseudo-Random Generator, Cryptography, Chinese Remainder Theorem

1. PRESENTATION

Pseudo-random generators are very useful in computer science, most of the time only they are used for obtaining a uniform repartition of the values, as in a Monte-Carlo algorithm. We can find a nice inventory of this first generation of generators in the well known book of D. Knuth.[1] Some theoretical results can be found in a paper of J.C. Lagarias.[2]

Recently with the flowering of the cryptography, random and pseudo-random generators are used in many cryptographic applications like the Diffie-Hellman key exchange.[3] In some case we need a determinist approach for example in a Vernam cipher[4] where a message is xored with the output of a random generator. For that, only a pseudo-random generator allows to decipher. In this case, a known seed is needed to initialize the generator, this seed is the secret key. In most of the cases, as true random generators are not easy to built, and are often slow, pseudo-random ones are preferred.

But for a use in cryptography, a pseudo-random generator must be secure. In other terms, if one output is known, it must be very difficult to predict the next value. Thus a notion of cryptographically secure pseudorandom generator must be defined.[5] A. Shamir introduced[6] the first pseudo-random generator in respect to this notion. In the RFC 1750,[7] they proposed some criteria for these generators in cryptography. A good survey of the use of pseudo-random generators in cryptography is given in the Handbook of Applied Cryptography.[8]

As suggested in the appendix 3 *Random Number Generation for the DSA* of the FIPS 186-2,[9] a cipher algorithm can be used for constructing a pseudo-random generator. It is the case of the the generator proposed in the ANSI X9.17, which uses the Data Encryption Standard (DES). The method proposed in this paper is based on a protocol which can be used for ciphering. But, as it is not the purpose of this paper we do not discuss of that, we just give an analyze the secure randomness of the depicted process. For studying the behaviour of our approach we use a suite of test proposed by the NIST.[10] One of the first series of tests was proposed in.[11] Then, one of the most popular the DIEHARD was constructed.[12] In 2001, the NIST proposed a new statistical test suite in the SP 800-22.[10] A recent software, TestU01, is now accessible on the web.[13] But as we explain in section 4, our choice is devoted to the NIST suite.

Thus, we first introduce a representation based on the Remainder Chinese Theorem, called Residue Systems. Then we depict the basic round of our approach, giving the cipher and decipher process. In the analyze of the random behaviour, we justify a double-round approach for a better answer to the tests. We give some examples for illustrating our discussion. To end we conclude giving some element of perspectives.

Jean Claude Bajard is moving to LIP6 Paris in September 2009

2. INTRODUCTION TO THE RESIDUE SYSTEMS OF REPRESENTATION

In,[14] a residue representation of the elements of $GF(2^k)$ is introduced using a set of trinomials. For our purpose, we generalize this system with a set of co-prime polynomials.

We define the residue representation of an element A of $GF(2)[X]$ of degree lower than k as a set $(a_1,...,a_n)$ of polynomials of degre lower than r, such that $a_i(X) \equiv A(X) \pmod{p_i}$ for all $i = 1,...,n$ with $(p_1,...,p_n)$ a set of co-prime polynomials of degre r with $n \times r \geq k$. In this paper we consider that the p_i have the same degree r but this can be generalized.

The conversion to the standard polynomial representation is obtained, in the context of this paper, with the Newton interpolation. We begin to construct an intermediate representation $(\hat{a}_1,...,\hat{a}_n)$ with:

$$\begin{cases} \hat{a}_1 & = a_1 \\ \hat{a}_2 & = (a_2 - \hat{a}_1) \times (p_1)^{-1}_{p_2} \bmod p_2 \\ \vdots & \\ \hat{a}_n & = \left(\cdots (a_n - \hat{a}_1) \times (p_1)^{-1}_{p_n} \cdots - \hat{a}_{n-1}\right) \times (p_{n-1})^{-1}_{p_n} \big) \bmod p_n \end{cases} \tag{1}$$

Then, the polynomial representation of A is given by:

$$A(X) = \hat{a}_1 + \hat{a}_2 p_1 + \hat{a}_3 p_1 p_2 + \cdots + \hat{a}_{n-1} p_1 \cdots p_{n-1} \tag{2}$$

It is known that if the p_i are trinomials, then it is possible to define a sub-quadratic algorithm[14] for the multiplication in $GF(2^k)$. During experimentations on this algorithm, we had noted some random properties that we can use for the construction of a pseudo-random generator.

3. PROPOSITION OF A PSEUDO-RANDOM GENERATOR BASED ON CHINESE RAMAINDER THEOREM

The main idea is to consider a state E as a polynomial with $2k$ coefficients in $GF(2)$, and two residue bases (p_1, p_2) and (p_3, p_4) where the p_i are polynomials of degree k in $GF(2)$.

Algorithm 1 Basic Round of Residue Pseudo-Random Process

Require: a state E ($2k$ bits vector), and two residue bases (p_1, p_2) and (p_3, p_4).
Ensure: new state $\tilde{E}$ ($2k$ bits vector)
  Evaluate $e_1 = E \bmod p_1$ and $e_2 = E \bmod p_2$
  Construct the new state $\tilde{E} = e_1 + \left((e_2 - e_1) \times (p_3)^{-1}_{p_4} \bmod p_4\right) \times p_3$

We note that the two bases (p_1, p_2) and (p_3, p_4) are not linked. So, we insure here a discontinued process. Attackers have to find the two bases for recovering E. We consider that the bases represent the seeds of our system with the initial state E. The probability[15] that two random polynomials of degree r are co-prime is $\frac{1}{2}$. Thus, the construction of the two bases is easy.

When the two bases are known, the process is reversible. Thus, we can use also this method for ciphering. Indeed, $\tilde{E} \bmod p_3 = e_1$ and $\tilde{E} \bmod p_4 = e_2$

Algorithm 2 Reversed Basic Round of Residue Pseudo-Random Process

Require: a ciphered state $\tilde{E}$, and two residue bases (p_1, p_2) and (p_3, p_4).
Ensure: original state E
  Evaluate $e_3 = \tilde{E} \bmod p_3$ and $e_4 = \tilde{E} \bmod p_4$
  Construct the new state $E = e_3 + \left((e_4 - e_3) \times (p_1)^{-1}_{p_2} \bmod p_2\right) \times p_1$

EXAMPLE 1. *We consider two bases*

$$
\begin{aligned}
p_1 &= 1+x^3+x^{19}+x^{22}+x^{23}+x^{26}+x^{27}+x^{28}+x^{31}+x^{32}\\
p_2 &= 1+x^1+x^2+x^6+x^9+x^{19}+x^{22}+x^{26}+x^{28}+x^{31}+x^{32}\\
p_3 &= 1+x^2+x^9+x^{18}+x^{23}+x^{26}+x^{28}+x^{29}+x^{31}+x^{32}\\
p_4 &= 1+x^1+x^3+x^7+x^{11}+x^{20}+x^{25}+x^{26}+x^{27}+x^{31}+x^{32}
\end{aligned}
$$

The initial state E is given in binary least significant degree first (the polynomial form is given):

$$
\begin{aligned}
E &= 0100000000000000000000000000000000000000000000000000000000000100\\
E &= x^1+x^{61}
\end{aligned}
$$

Using Algorithm 1 we obtain the ciphered value $\tilde{E}$:

$$\tilde{E} = 1110111001111110010011111110101000111101000010110110000100110001$$

Then, we recorver the original value with Algorithm 2:

$$E = 0100000000000000000000000000000000000000000000000000000000000100$$

We note that the reversibility of the process can be a drawback of this generator. But,it is the case of most of the standards, as ANSI X9.17 which uses a DES. The bases and the initial state must be secret.

4. ANALYSIS OF THE RANDOMNESS

For the analysis of the randomness of our proposal we had used the tests[10] proposed by the NIST. This suite includes some tests due to Maurer[11] or Marsaglia with the Diehard tests.[12] We prefer the NIST suite to the more recent Testu01[13] for two reasons: the first one is that is a standard in the world of cryptography, the second one is due to the facility of use. Testu01 can be, perhaps, better for a very precise study. But our goal is just to propose an approach regarding the recommendations of the NIST standard.

4.1 Summarized description of the NIST suite

This testing tool is composed of fifteen evaluations. For each test, two things are verified: the first the direct answer to the test giving a p-value for each tested blocks, the second one is the uniform repartition of this results in respect to a true random generator. The ciphered file is split in m sequences of n bits which are evaluated and the m result are distributed in ten different intervals available for the p-values, this distribution must be uniform as it would be the case for a true random generator.

1. The Frequency (Monobit) Test: it consider that the number of 1 must be equal to the number of 0. It is a very simple test which is fundamental for a random generator.
2. Frequency Test within a Block: it is inspired of the previous one, each sequence is decomposed in N blocks of M bits, and for each sequence it is verified how (using a χ^2 function like in most of the tests) the distribution of the number of 1 in each block satisfies a repartition observed for true generators. The interest is to verify that each word of M bits occurs in a good proportion. The standard recommends $M = 128$.
3. The Cumulative Sums (Cusums) Test: it measure the variation of the cumulative sum (-1 for the 0s and $+1$ for the 1s) in the sequence. For this, it evaluate the maximum of this sum computed in one way (forward) and then in the opposite way(backward). These results are compared to the expected value for a random sequence.
4. The Runs Test: it verifies that successive 1s or 0s, occur in respect of a random sequence. It tests, for that, the frequencies of the oscillations between 0 and 1.

5. Test for the Longest-Run-of-Ones in a Block: it is the same kind of test as the previous one but applied on blocks of size M. Here the recommended size for $n = 10^6$ is $M = 10^4$.

6. The Binary Matrix Rank Test: in this test, the sequences are split in 32×32 binary matrices. For each matrix, it measure the rank (in other word the number of independent vectors). It is compared to the expected result for a random sequence. This test is directly inspired of the Diehard.

7. The Discrete Fourier Transform (Spectral) Test: the idea of this test is to analyze each sequence as a signal. A good randomness is insured if no high picks are detected. But, as in a random sequence every suite can occur, this test verifies that the sequences respect a repartition assumed by a true random generator.

8. The Non-overlapping Template Matching Test: this test uses a sliding window for finding patterns, if the pattern is found then the window jumps to the next bit following the pattern. The number of patterns turns around 147 for a window of 9 bits, they are issued from a template library of non-periodic patterns. The results are given for each pattern. For statistical test, each sequence is cut in large blocks (for example 8 blocks).

9. The Overlapping Template Matching Test: this test is similar to the previous one. Here, when the pattern is found, the window does not jump but slides of one bit. Only the all ones pattern is tested.

10. Maurer's "Universal Statistical" Test: it deals with a research of similar patterns, as for a compression algorithm like Lempel-Ziv. It cumulates the logarithms of the distance between identical blocks (the length of the block is L, for example $L = 7$), if some blocks appear frequently this sum will be smaller. Then the results are compared to the ones expected for a random sequence.

11. The Approximate Entropy Test: this test, as the serial one, is close to the frequencies tests. Here, it measures the entropy of the occurrences of m-bits words and compares it to the one of $(m+1)$-bits words. Then, it evaluates how it corresponds to the suited results.

12. The Random Excursions Test: For this test, a sequence of cumulative sums is formed from a random sequence, successively adding +1 for 1s and -1 for 0s to the previous sum. A cycle of the obtained sequence is a subsequence starting and ending with 0. For each cycle, the number of occurrences of the values $-4, -3, -2, -1, +1, +2, +3, +4$ is counted and compared to the one expected for real random sequences.

13. The Random Excursions Variant Test: this variant is very close to the previous tests. The number of tested values of the cumulative sum is larger, from -9 to $+8$.

14. The Serial Test: this test is close to the frequency tests. It studies the variation of the frequencies of all the pattern of length M, $M-1$ and $M-2$ (for example $M = 7$) using derivation of first and second degree. Then it compares its results to expected frequencies.

15. The Linear Complexity Test: in this test each sequence is split into blocks of length M (e.g. $M = 500$) and using Berlekamp-Massey algorithm it finds the shorter LFSR (Linear Feedback Shift Register) which can construct this block. So, longer is the LSFR, better it is. The results for each sequence is then compared to an expected result.

4.2 Comments on the results obtained by the Residue Approach

The simple use of Algorithm 1 with two bases was not convincing, and it fails the tests. Hence, we had tested a double round algorithm with four bases, which passes successfully the tests. The results are similar to the ones obtained with a ciphered GPG file.

Algorithm 3 Double Round of Residue Pseudo-Random Process

Require: a state E, and four residue bases (p_1, p_2), (p_3, p_4) and (p_5, p_6), (p_7, p_8).
Ensure: new state $\tilde{E}$
Evaluates $e_1 = E \bmod p_1$ and $e_2 = E \bmod p_2$
Constructs the new state $\tilde{E} = e_1 + \left((e_2 - e_1) \times (p_3)^{-1}_{p_4} \bmod p_4\right) \times p_3$
Evaluates $e_5 = E \bmod p_5$ and $e_6 = E \bmod p_6$
Constructs the new state $\tilde{E} = e_5 + \left((e_6 - e_5) \times (p_7)^{-1}_{p_8} \bmod p_8\right) \times p_7$

EXAMPLE 2. *We give here the partial results of a NIST suite of tests, made on a ciphered file obtained with Algorithm 3 with the bases and initial state given in Example 1, completed with the bases:*

$$
\begin{aligned}
p_5 &= 1+x^{12}+x^{13}+x^{18}+x^{19}+x^{20}+x^{21}+x^{22}+x^{30}+x^{32}\\
p_6 &= 1+x^{5}+x^{8}+x^{15}+x^{19}+x^{21}+x^{23}+x^{25}+x^{30}+x^{31}+x^{32}\\
p_7 &= 1+x^{2}+x^{3}+x^{4}+x^{23}+x^{27}+x^{28}+x^{29}+x^{30}+x^{32}\\
p_8 &= 1+x^{5}+x^{6}+x^{7}+x^{13}+x^{23}+x^{25}+x^{26}+x^{29}+x^{31}+x^{32}
\end{aligned}
$$

This test used a file of 800 000 000 bits which was split in 800 sequences of 1 000 000 bits. The columns `C1` *to* `C10` *represent the number of p-values belonging to the intervals possible from* 0...0.1 *to* 0.9...1. *The test is considered as successful if the p-value is greater than* 0.001. *The column* `P-Value` *correspond to the p-value of the repartition in the columns C-i, if we get 80 in each of the column, the repartition is uniform and this p-value is equal to 1. The column* `PROPORTION` *gives the ratio of successful tests.*

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

generator is <ChiffreBis1>

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
86	85	67	68	90	83	83	92	68	78	0.366918	0.9875	Frequency
79	70	66	83	82	87	87	77	80	89	0.717205	0.9888	BlockFrequency
90	79	81	75	83	83	69	77	88	75	0.871642	0.9888	CumulativeSums
90	81	72	72	88	74	70	81	99	73	0.311542	0.9875	CumulativeSums
90	72	93	71	78	88	77	75	70	86	0.519103	0.9938	Runs
77	82	94	77	75	75	77	71	75	97	0.494392	0.9875	LongestRun
800	0	0	0	0	0	0	0	0	0	0.000000 *	0.0000 *	Rank
11	47	61	68	81	95	115	110	109	103	0.000000 *	1.0000	FFT
93	83	78	77	78	75	87	95	73	61	0.255705	0.9925	NonOverlappingTemplate
86	94	79	66	99	83	82	63	66	82	0.063815	0.9912	NonOverlappingTemplate
95	87	87	78	59	68	66	97	69	94	0.010891	0.9900	NonOverlappingTemplate
88	81	87	78	74	85	82	77	78	70	0.930026	0.9900	NonOverlappingTemplate
73	79	84	89	80	79	86	65	82	83	0.809707	0.9912	NonOverlappingTemplate
...												
86	88	73	71	78	91	85	85	62	81	0.425817	0.9925	NonOverlappingTemplate
92	73	73	94	82	85	74	84	67	76	0.455937	0.9938	OverlappingTemplate
84	75	84	86	92	70	76	77	79	77	0.863690	0.9900	Universal
81	71	79	82	84	78	78	80	84	83	0.995373	0.9875	ApproximateEntropy
60	47	59	37	49	51	47	50	51	35	0.208652	0.9877	RandomExcursions
53	45	44	48	59	54	45	45	42	51	0.776784	0.9959	RandomExcursions
...												
51	62	40	36	51	45	46	49	52	54	0.337162	0.9918	RandomExcursions
51	55	49	40	50	53	44	53	43	48	0.872295	0.9938	RandomExcursionsVariant
47	53	52	55	39	56	39	48	36	61	0.157031	0.9856	RandomExcursionsVariant
46	50	54	48	58	51	43	43	47	46	0.888137	0.9774	RandomExcursionsVariant
...												
49	49	54	45	52	42	60	36	48	51	0.509162	0.9877	RandomExcursionsVariant
73	81	68	72	91	77	92	85	75	86	0.562080	0.9925	Serial
90	70	77	83	80	67	79	83	87	84	0.762209	0.9850	Serial
390	52	38	57	39	39	59	36	40	50	0.000000 *	0.5625 *	LinearComplexity

The results obtained for different bases and initial states, all selected randomly, are similar to those of the Example 2.

If we compare with a file of the same size ciphered with gpg using an `AES 256`, we observe similar results excepted for the Rank (see Example 3). In the two cases, the proposed approach and AES, most of the tests are validated. For the `FFT` and the `LinearComplexity` the results are very similar and can be explained. For the FFT all the tests are too successful, in the sense that no signal occurred. It is relatively understandable, a pseudo-random generator is constructed to reach this point. Now about the `LinearComplexity`, the fact that this kind of generators is determinist, gives to the blocks the possibility to be generated by short LSFR. We can assume that the two tests will give the same kind of result for the majority of the pseudo-random generators. Hence only the Rank test stay, with the first experimentations, a problem for residue pseudo-random generator. We must focus our attention on this point, and analyze more deeply the results obtained for this test.

EXAMPLE 3. *We give here the results obtained of a file ciphered with* `gpg` *using an* `AES 256`*. The size of the file is of 800 000 000 bits, split in 800 sequences of 1 000 000 bits.*

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
83	81	79	79	85	85	80	70	71	87	0.932904	0.9912	Frequency
61	91	82	84	79	76	72	85	87	83	0.501755	0.9862	BlockFrequency
80	80	80	85	88	73	75	75	85	79	0.975801	0.9912	CumulativeSums
86	72	77	85	93	68	77	78	86	78	0.714660	0.9900	CumulativeSums
86	79	69	81	66	86	70	91	88	84	0.470189	0.9875	Runs
78	80	83	84	63	75	75	84	89	89	0.655334	0.9875	LongestRun
83	79	81	62	82	76	70	98	84	85	0.330628	0.9912	Rank
31	43	48	89	97	91	106	93	98	104	0.000000 *	1.0000	FFT
71	81	67	77	76	83	92	91	88	74	0.521600	0.9888	NonOverlappingTemplate
...												
91	68	94	79	89	71	81	77	74	76	0.477392	0.9825	NonOverlappingTemplate
80	81	66	84	83	74	70	86	84	92	0.644928	0.9925	OverlappingTemplate
85	78	86	72	80	95	74	69	69	92	0.375313	0.9875	Universal
77	74	86	76	74	92	69	86	84	82	0.771953	0.9850	ApproximateEntropy
61	50	38	49	59	46	63	42	38	51	0.085481	0.9819	RandomExcursions
...												
43	63	49	42	50	51	50	48	54	47	0.692175	0.9980	RandomExcursionsVariant
...												
43	52	58	60	53	41	52	40	48	50	0.479706	0.9960	RandomExcursionsVariant
89	62	83	83	75	91	82	70	75	90	0.352513	0.9862	Serial
79	72	76	82	83	80	71	76	86	95	0.774372	0.9912	Serial
389	47	50	42	38	52	44	51	52	35	0.000000 *	0.5600 *	LinearComplexity

Thus, we show in the Example 4 the statistics obtained during a Rank test of our approach. The sequence is split in 32×32 matrices, in a sequence of 1 000 000 bits that represents 976 matrices. we observe that no one has the maximal rank of 32 but most of them have a rank of 31. This point is discussed in the paper introducing TestU01.[13] They explain that Diehard can fail with some good pseudo-random generators, and that was one of the reasons they excluded this test from TestU01. For our purpose, we tested with bases of 64 bits, and we verify that the Rank test was passed with success. That confirms the discussion given in TestU01 about this Diehard test.

EXAMPLE 4.

```
RANK TEST
---------------------------------------------
COMPUTATIONAL INFORMATION:
---------------------------------------------
```

```
(a) Probability P_32 = 0.288788
(b)             P_31 = 0.577576
(c)             P_30 = 0.133636
(d) Frequency   F_32 = 0
(e)             F_31 = 716
(f)             F_30 = 260
(g) # of matrices    = 976
(h) Chi^2            = 451.716918
(i) NOTE: 576 BITS WERE DISCARDED.
---------------------------------------------
FAILURE p_value = 0.000000
```

EXAMPLE 5. *In the example we consider a file of* 87752704 *bits ciphered with the residue approach using bases of* 64 *bits. The file is split in* 103 *sequences of* 851968 *bits representing* 832 *matrices* 32×32.

The elements of the bases are presented in binary with the least significant coefficient first.

$$
\begin{array}{rcl}
p_1 & = & 1001001011100001000100110011100110010010111000010001001100111 0011 \\
p_2 & = & 1110001001001100100100111110100111100010010011001001001111101 0011 \\
p_3 & = & 1010000000100000000010000100101101101000000010000000001000010010 11011 \\
p_4 & = & 1101000100010000000001000011100011101000100010000000001000011100 011 \\
p_5 & = & 1001010110001100001111100000000101000111001001100000011111000001 01 \\
p_6 & = & 1000010010000000100010101010100111000000100100110000111010011000 101 \\
p_7 & = & 1111010110010100100110100011010110111000000111000100000100011 1101 \\
p_8 & = & 1000011100100100100000010110010111000000001011000010111111010111 11 \\
 & & \\
E & = & 0100001000000001000111010100010010110100100000001000011000010010 11 \\
 & & 1011001100110010011110001010100000011010001001101010011001100101
\end{array}
$$

With these bases and this value for E, we had obtained the following results.

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
generator is <Chiffre64>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
 ------------------------------------------------------------------------------
  16  10  11  11  14  12   6   4   6  13  0.141256   0.9806    Rank
  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

To end, we must confess that the choice of the bases is crucial for the approach proposed in this paper. During our experimentations, we had observed some very bad results with the bases composed only with trinomials.

5. DISCUSSIONS AND CONCLUSIONS

In this paper we introduce a new approach for constructing a pseudo-random generator available in cryptography. This method uses polynomial multiplications and modular reductions found in most of the crypto-processor. It can be useful in particular for hardwares dedicated to Elliptic Curves Crypto systems. We had shown its good behaviour through the different tests passed with success. In the perspectives, we must characterize the well suited residue bases. We have seen that some, like all trinomials bases, are not well adapted. But some parse bases, with some regular structure are fine. The number of pairs of co-prime polynomial is huge, that lets us thinking that we can propose a large choice of bases. A last point to study is the ciphering properties of this approach.

REFERENCES

[1] Knuth, D., [*The Art of Computer Programming Seminumerical Algorithms*], ch. 3, Addison-Wesley, 2nd ed. (1981).

[2] Lagarias, J., "Pseudorandom number generators in cryptography and number theory," in [*Cryptology and Computational Number Theory,volume 42 of Proceedings of Symposia in Applied Mathematics*], Pomerance, C., ed., American Mathematical Society (1990).

[3] Diffie, W. and Hellman, M., "New directions in cryptography," *IEEE Transactions on Information Theory* **22** (1976).

[4] Vernam, G., "Cipher printing telegraph systems for secret wire and radio telegraph communications," *Journal of the American Institute for Electrical Engineers* **55** (1924).

[5] Blum, M. and Micali, S., "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing* **13** (1984).

[6] Shamir, A., "On the generation of cryptographically strong pseudorandom sequences," *ACM Transactions on Computer Systems* **1** (1983).

[7] Eastlake, D., Crocker, S., and Schiller, J., "Randomness requirements for security," tech. rep., RFC 1750 (1994).

[8] Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A., [*Handbook of Applied Cryptography*], ch. 5, CRC Press, fifth ed. (2001).

[9] NIST, "Digital signature standard (dss)," Tech. Rep. FIPS 186-2, NIST Federal Information Processing Standards (2000).

[10] NIST, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Tech. Rep. SP 800-22, NIST Special Publication (2001).

[11] Maurer, U., "A universal statistical test for random bit generators," *J. Cryptology* **5** (1992).

[12] Marsaglia, G., "Diehard: a battery of tests of randomness." See http://stat.fsu.edu/geo/diehard.html (1996).

[13] L'Ecuyer, P. and Simard, R., "Testu01: a c library for empirical testing of random number generators," *ACM Trans. on Mathematical Software* **33** (2007).

[14] Bajard, J., Imbert, L., and Jullien, G. A., "Parallel montgomery multiplication in $GF(2^k)$ using trinomial residue arithmetic," in [*17th IEEE symposium on Computer Arithmetic (ARITH 17)*], Pomerance, C., ed., IEEE Computer Society (2005).

[15] Ivchenko, G. I. and Medvedev, Y. I., "Random polynomials over a finite field," *Discrete Mathematics and Applications* **18** (2008).