An efficient pseudo-random generator provably as secure as syndrome decoding

Jean-Bernard Fischer[1] and Jacques Stern[2]

[1]Thomson Consumer Electronics R&D France
Parc d'innovation n° 1, B.P. 120, 67403 Illkirch Cedex, France
fischerj@tce-rdf.fr

[2]Ecole Normale Supérieure, Laboratoire d'informatique
45, rue d'Ulm, 75230 Paris Cedex 05, France
Jacques.Stern@ens.fr

Abstract. We show a simple and efficient construction of a pseudo-random generator based on the intractability of an NP-complete problem from the area of error-correcting codes. The generator is proved as secure as a hard instance of the syndrome decoding problem. Each application of the scheme generates a linear amount of bits in only quadratic computing time.

1 Introduction

A pseudo-random generator is an algorithm producing strings of bits that look random. The concept of "randomly looking" has been formalized by Blum and Micali [4] within the framework of complexity theory. Yao [22] has shown that the existence of a one-way permutation is sufficient to construct a pseudo-random generator. Subsequently, a long series of deep articles led to the conclusion that the existence of a one-way function is equivalent to the hypothesis that a pseudo-random generator exists [15, 10, 14]. However, the theoretical constructions proposed in these articles are often impractical.

Several schemes have been proposed which have a "proven security", i.e. based on the difficulty of well known problems like factorization [21, 3, 1, 18] or the discrete logarithm [4, 16, 12]. But these propositions suffer from a relatively slow computing rate (i.e. they need much computation per generated bit). For example, outputting a single bit for the BBS generator takes quadratic time, and cubic time for the RSA based generators. This can be slightly enhanced to a $log_2(n)$ output.

Therefore, it is interesting to study new schemes based on difficult problems not related to number theory. In an early work [9], Goldreich, Krawczyk and Luby established that the existence of a pseudo-random generator could be based on hard problems from the theory of error-correcting codes. Unfortunately, their construction was a bit intricate. In the same vein, Impagliazzo and Naor [13] devised an elegant construction of a pseudo-random generator based on the subset-sum problem. Unfortunately, many researchers think that the underlying problem is computationnally rather weak. We present a new scheme based on the syndrome decoding problem, which is believed to be hard on most instances. Our scheme is extremely simple and achieves quadratic time with respect to the security parameters for producing a linear amount of random bits.

1.1 The Syndrome Decoding problem

In the field of error correcting codes, one of the basic open problems is to find efficient algorithms for the decoding of random linear codes. Codes with a constant information rate and correcting a constant fraction of bits are particularly interesting.

A (n, k, d) binary linear code is a subspace of $\{0, 1\}^n$ of size 2^k where every non zero word has weight at least d. It's information rate is k/n and it can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. It can be defined by its parity check matrix which is a n-by-$(n-k)$ binary matrix with the property that, for each vector of the code, the product (mod 2) of the matrix by the vector is zero; this product can actually be computed for any vector and is called the syndrome. If the vector is not in the code, the syndrome is the sum of the columns of the matrix corresponding to positions where one bits are located and is non zero.

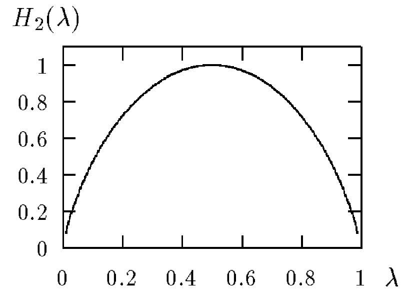
Random linear codes are defined by a random parity check matrix. For such codes, no efficient algorithm is known for finding the closest code word to a vector, given its syndrome. It is also difficult to find a word of given weight from his syndrome's value [2]. This is called the **syndrome decoding problem**.

These problems are NP-complete. For further information, we refer to the books by McWilliams and Sloane [17] for error correcting codes, and by Garey and Johnson [7] for NP-complete problems. The syndrome decoding problem is **NP-hard**; see Berlekamp, McEliece and van Tilborg [2] for a proof. It can be stated as follows [7]:

Instance: An $m \times n$ binary matrix $A = (a_{ij})$, a binary non-null vector $y = (y_1, \ldots, y_m)$, and a positive integer w.
Question: Is there a binary vector $x = (x_1, \ldots, x_n)$ with no more than w 1's such that, for $1 \leq j \leq m$, $\sum_{i=1}^{n} x_i \cdot a_{ij} \equiv y_j \pmod 2$?

Comment: The variant in which we ask for an x with *exactly* w 1's is NP-complete, even for $y = (0, 0, \ldots, 0)$. If we drop the word exactly, the question becomes open.

NP-hardness ensures that there is no polynomial-time algorithm for solving the problem in the worst case; however many NP-complete problems can be efficiently solved in the average case. So we have to study the hardness of a random instance. This leads to practical issues that have been extensively studied by various people, as reviewed by Chabaud [6, 19]. Algorithms known to solve the syndrome decoding problem are all probabilistic, so that we will speak about the probability of finding a word. They all have a computing complexity that grows exponentially in the size of n. They also give strong evidence that, in the case of random codes, the problem is the hardest for weights in the neighbourghood of the Gilbert-Warshamov bound corresponding to the dimensions of the code. The Gilbert-Warshamov bound λ of a (n, k, d) binary code is defined by the relation $1 - k/n = H_2(\lambda)$ where $H_2(x) = -x\log_2 x - (1-x)\log_2(1-x)$.



What comes out of the analysis is that it is relatively easy to find words of very low weight and given syndrome, but that the probability of finding one increases exponentially with the weight of the word (these algorithms are all probabilistic). However, the number of words having a given syndrome also grows exponentially with the weight; the point where there is an average of one word per syndrome value is the Gilbert-Warshamow bound. Finally, given a random vector with weight below the bound, the probability for it of having a pre-image is exponentially small in the weight.

We reach the following experimental conclusion: the difficulty of finding a word is a function of its weight that is growing until the Gilbert-Warshamov bound, and is then decreasing. So we can define a hard instance of syndrome decoding as an instance where the weight of the vectors is close to the Gilbert-Warshamov bound.

1.2 Formal version of the intractability assumption

In the sequel, we will only consider subsets D_n such that, for suitable functions $k(n)$ and $l(n)$, D_n is included in $\{0,1\}^{l(n)}$ and is the one-one image of $\{0,1\}^{k(n)}$ by a polynomial-time function. This is a particular case of what is called polynomially-samplable [8].

We now define what we mean by hard (this is essentially a simplified version of Goldreich [8]):

Definition 1 *A collection of functions $\{f_n : D_n \mapsto \{0,1\}^{k(n)}\}$ is called* ***strongly one-way*** *if the following two conditions hold:*

- *there exists a polynomial-time algorithm F that, on input $x \in D_n$, always outputs $f_n(x)$.*
- *for every probabilistic polynomial-time algorithm A, every $c > 0$ and all sufficiently large n's,*

$$\Pr(A(f_n(X_n)) \in f_n^{-1} f_n(X_n)) < \frac{1}{n^c}$$

where X_n is a random variable uniformely distributed over D_n.

We now consider a very general collection of functions related to the syndrome decoding problem.

Definition 2 *Let ρ be in $]0,1[$; let w and w' be two integer functions such that $w(n) \leq w'(n) < n$. The SD(ρ, w, w') collection is the set of functions $\{f_n\}$ such that:*
$D_n = \{(M,x), M \in \lfloor \rho n \rfloor \times n, x \in \{0,1\}^n / w(n) \leq |x| \leq w'(n)\}$
$$\begin{array}{rrcl} f_n: & D_n & \longrightarrow & \{0,1\}^{\lfloor \rho n \rfloor \cdot (n+1)} \\ & (M,x) & \longmapsto & (M, M \cdot x) \end{array}$$

As we have seen in the previous section, instances of the problem where $w(n)$ and $w'(n)$ are very small or close to $n/2$ are not hard, so that we want to restrict the collection to the instances where the weight of x is near the Gilbert-Warshamov bound. Clearly, it is not known if such a collection is one-way, but we have seen that, despite extensive research, no efficient inversion algorithm has been found. So we make the assumption that the collection is one-way for weights slightly below the Gilbert-Warshamov bound.

Intractability assumption 1 *Let ρ be in $]0,1[$; let λ be defined by $\rho = H_2(\lambda)$ and $\lambda < 1/2$. Then, for any positive real ϵ, if we set $w(n) = \left\lfloor \frac{\lambda}{1+\epsilon} n \right\rfloor$ and $w'(n) = \lfloor \lambda n \rfloor$, the SD($\rho, w, w'$) collection is strongly one-way.*

Note: D_n has to be polynomially-samplable. We will discuss this point later on.

Of particular interest to us is the case where the weight of x is fixed and is a constant fraction of n, i.e. $w(n) = w'(n) = \lfloor \delta n \rfloor$ for some δ in $]0,1[$. The corresponding intractability assumption reads as follows:

Intractability assumption 2 *Let ρ be in $]0,1[$. Then, for all δ in $]0,1/2[$ such that $H_2(\delta) < \rho$, the SD(ρ, δ, δ) collection of functions is strongly one-way.*

We now write *SD*(ρ, δ) to refer to our second assumption with fixed parameters ρ and δ. We will occasionnally omit the parameters thus writing *SD* in place of *SD*(ρ, δ). The corresponding function will be denoted by $f_n^{\rho,\delta}$ or simply f_n and has domain $D_n^{\rho,\delta} = \{(M,x), M \in \lfloor \rho n \rfloor \times n, x \in \{0,1\}^n / |x| = \delta)\}$

Cryptographic applications of the syndrome decoding problem have appeared in an identification scheme devised by the second named author [20].

Note that if $H_2(\lambda) = \rho$ and $\delta < \frac{\lambda}{2}$, the intractability of *SD*(ρ, δ) is a particular case of the difficulty of decoding below half the minimum distance. Thus our assumption is stronger than the usual decoding assumptions (see [9]). The construction that appears in section 2 can equally be based on the hardness of decoding but, when we come to practical issues, codes of a larger dimension will be needed. This is why we chose to work with our SD assumption.

2 A pseudo-random generator based on the syndrome decoding problem

2.1 Construction of the generator

Our goal is to construct an efficient pseudo-random generator based on a hard problem, so that the generator inherits the hardness of the problem. We have defined a hard instance of the syndrome decoding problem and a collection of functions $SD(\rho, \delta)$ one-way under the second intractability assumption.

Let us first note that $f_n^{\rho,\delta}$ expands its input: the size of the input set is $2^{\lfloor \rho n \rfloor \cdot n} \cdot \binom{n}{\delta n}$, the size of the image set is $2^{\lfloor \rho n \rfloor \cdot n} \cdot 2^{\lfloor \rho n \rfloor}$; since $H_2(\delta) < \rho$, there exists a positive real ϵ such that $(1+\epsilon)\log_2 \binom{n}{\delta n} = \lfloor \rho n \rfloor$. That means that for a sufficiently large n, $f_n^{\rho,\delta}$ expands its input by some linear amount of bits. We note also that $f_n^{\rho,\delta}$ can be computed in time $O(n^2)$, since it is exactly the binary product of a $\lfloor \rho n \rfloor \times n$ matrix with a $\lfloor \rho n \rfloor$ column vector.

We construct the generator $G_{\rho,\delta}$ in the following way:

Input: $(M, x) \in D_n^{\rho,\delta}$
Output: $f_n^{\rho,\delta}(M, x) = (M, M \cdot x)$

Following a standard construction, we also consider an iterative generator $g_{\rho,\delta}$ that, on input (M, x), outputs as many bits as we like. To perform this iteration, we need an efficient algorithm that computes a vector of size n and weight δ from a $\log_2 \binom{n}{\delta n}$ bit number. For the time being, we consider such an algorithm A as granted and we will describe one in the next section. We get the following:

Input: $(M, x) \in D_n^{\rho,\delta}$

1. compute $y = M \cdot x$
2. separate y in two bit strings y_1 and y_2, y_1 being the first $\left\lceil \log_2 \binom{n}{\delta n} \right\rceil$ bits of y and y_2 the remaining bits.
3. **output** y_2
4. set $x \leftarrow A(y_1)$ and goto 1.

2.2 An algorithm for generating all words of given weight

The first and second intractability assumptions require a polynomial-time algorithm that samples the set of vectors of given weight. Such an algorithm is also needed in our scheme for the pseudo-random generator, but furthermore, it has to be computable in quadratic time to preserve the efficiency of the generator.

We will use an algorithm inspired by Guillot [11]. This algorithm is said quadratic, takes as input a word of exactly $\log_2 \binom{n}{w}$ bits and outputs a word of length n and weight $\lfloor \delta n \rfloor$.

However, in his paper, Guillot uses a heuristic approximation to justify the computing time $O(n^2)$. He assumes that multiplication and division of a large number (of size n) by a small number (of size $\log_2 n$) are linear because the small number has typically the size of a computer word. This argument is not enough for a more formal approach.

In the Turing machine model, the algorithm is "almost" quadratic since it has a of complexity $O(n^2 \log n)$. However the quadratic time bound holds in the model corresponding to random machines with logarithmic cost. See the appendix for a proof.

We now present the algorithm for generating words of given weight. It is based on an efficient way of outputting a word of length n and weight w given its index in the lexicographic ordering (see [11]).

We write the lexicographic enumeration algorithm in the following way:

```
Input: $i, n, w$

1. $c \leftarrow \binom{n}{w}$
2. while $n > 0$ do
   (a) $c' \leftarrow c \cdot \frac{n-w}{n}$
   (b) if $i \leq c'$
         – output 0
         – $c \leftarrow c'$
   (c) else
         – output 1
         – $i \leftarrow i - c'$
         – $c \leftarrow c \cdot \frac{w}{n}$
   (d) $n \leftarrow n - 1$
```

¿From the lemma, we note that the initial computation of $\binom{n}{w}$ takes quadratic time. The two cases in the while loop respectively take care of words where a zero (resp. a one) comes first: entering the loop with the initial values of the variables, we find $\binom{n-1}{w}$ words starting with 0, the remaining with 1. The same happens in subsequent iterations, mutatis mutandis.

2.3 Security of the scheme

Goldreich and Levin [10] have shown that the inner product is a hard bit for any one-way function. Recall that, if $x, r \in \{0,1\}^n$, the inner product $r \odot x$ is the parity of the number of positions where the bits of x and of r are both 1 ($r_i = x_i = 1$). The hardness result reads as follows:

Theorem 1 *[10] Let $f : D_n \subset \{0,1\}^{l(n)} \to \{0,1\}^{k(n)}$ be a one-way function. For every polynomial-time algorithm A, every polynomial p and all but finitely many n's,*

$$\Pr(A(f(x), r) = r \odot x) < \frac{1}{2} + \frac{1}{p(n)}$$

where the probability is taken over uniformly chosen $x \in D_n$ and $r \in \{0,1\}^{l(n)}$.

An algorithm is a pseudo-random generator if its output distribution is polynomial-time indistinguishable from a truly random distribution. Two distributions X and Y are indistinguishable in polynomial-time if, for every probabilistic polynomial-time algorithm D and every polynomial p, $|Prob(D(x) = 1) - Prob(D(y) = 1)| < \frac{1}{p(n)}$ where probabilities are taken over X and Y. We will prove that the generator $G_{\rho,\delta}$ is pseudo-random under intractability assumption 2.

Theorem 2 *If the SD(ρ, δ) collection is one-way, then $G_{\rho,\delta}$ is pseudo-random.*

Remark: By standard arguments, the same result extends to the iterative generator $g_{\rho,\delta}$.

Proof. This proof is inspired from Impagliazzo and Naor [13]. Let $G_{\rho,\delta}$ be the generator with matrix $M \in 2^{\lfloor \rho n \rfloor \times n}$ and weight $w = \lfloor \delta n \rfloor$ as defined in 2.1. If $G_{\rho,\delta}$ is not pseudo-random, we can build a distinguisher that accepts (i.e. outputs 1) with a different probability a string generated by $G_{\rho,\delta}$ and a random string. We can then use this distinguisher to predict the inner product of r and s with an advantage better then $\frac{1}{poly(n)}$ from given values of r and $G_{\rho,\delta}(s)$. Using the Goldreich-Levin theorem, we have a contradiction to the one-wayness of the corresponding $f_n^{\rho,\delta}$ function.

When we think of r and s as defining subsets of the matrix columns (the i-th column of the matrix is in the subset defined by r (resp. s) if the i-th bit of r (resp. s) is 1), we see that $r \odot s$ is the parity of the intersection. The idea is to use the distinguisher to predict this parity.

The distinguisher is fed with a matrix M and a binary word t of size $\lfloor \rho n \rfloor$. Without loss of generality, we can assume that, for infinitely many indices, the distinguisher

- outputs 1 with probability at least $\frac{1}{2} + \frac{1}{p(n)}$ if t is the product of the matrix with a word of weight w;
- outputs 1 with probability almost exactly $\frac{1}{2}$ if t is chosen uniformly in $\{0,1\}^{\lfloor \rho n \rfloor}$.

Let M in $\lfloor \rho n \rfloor \times n$ be a matrix, where c_i is the i-th column of M: $M = (c_i)_{1 \le i \le n}$. Given $r = (r_1, \ldots, r_n) \in \{0,1\}^n$ and $x \in \{0,1\}^{\lfloor \rho n \rfloor}$ both chosen at random, we construct a new matrix $M_x^r = (c_i')_{1 \le i < n}$ such that:

- if $r_i = 1$, the i-th column of M_x^r is the bitwise xor of c_i and x (i.e. $c_i' = c_i \oplus x$);
- if $r_i = 0$, the i-th column of M_x^r is c_i.

We let u denote a possible output of the pseudo-random generator, i.e $M \cdot s = u$ for some s. Let $k = |r \cap s|$ be the number of locations where the bits of r and s are both 1, and ϵ be the parity of k. As noted above, this is equivalent to saying that ϵ is the inner product of r and s. Let $\epsilon \cdot x = x$, if $\epsilon = 1$ and $0^{\lfloor \rho n \rfloor}$ otherwise. We observe that the result of $M_x^r.s$ is $u \oplus \epsilon \cdot x$, as established by the following computation:

$$\begin{aligned} M_x^r.s &= \textstyle\bigoplus_{c'_i \in s} c'_i \\ &= (\textstyle\bigoplus_{c_i \in s} c_i) \oplus (\textstyle\bigoplus_{c_i \in r \cap s} x) \\ &= u \oplus \epsilon \cdot x \end{aligned}$$

On input $(M_x^r, u \oplus \epsilon \cdot x)$, the distinguisher sees exactly the same distribution as on input (M, u). But then, the distinguisher outputs 1 on input $(M_x^r, u \oplus \epsilon \cdot x)$ with probability at least $\frac{1}{2} + \frac{1}{p(n)}$. On the other hand, if we replace ϵ by $\overline{\epsilon}$, then, due to the randomness of x, the distinguisher is fed with uniformly distributed inputs. The distinguisher outputs 1 with probability 1/2. This leads us to the following consruction:

Input: $M \in \{0,1\}^{\lfloor \rho n \rfloor \times n}, u \in \{0,1\}^{\lfloor \rho n \rfloor}, r \in \{0,1\}^n$

- choose a random $x \in \{0,1\}^{\lfloor \rho n \rfloor}$ and a random $\theta \in \{0,1\}$ (θ is our guess about the inner product)
- feed the distinguisher with $(M_x^r, u \oplus \theta \cdot x)$
- if the distinguisher answers 1, **output** θ, else **output** $\overline{\theta}$.

Theorem 3 *Our algorithm predicts the inner product with probability at least* $\frac{1}{2} + \frac{1}{2p(n)}$.

Proof. If we have guessed ϵ, our predictor is correct if the the distinguisher outputs 1. We have seen that this is the case with probability at least $\frac{1}{2} + \frac{1}{p(n)}$.

If we have not guessed ϵ correctly, the predictor sees the input as a totally random distribution. It then outputs 1 with probability almost $\frac{1}{2}$. Since both cases happen with probability $\frac{1}{2}$, the overall probability of the algorithm to predict the inner product is at least $\frac{1}{2} + \frac{1}{2p(n)}$. $\square$

3 Performances

The computation of the product of the matrix by an vector is done solely by using logical operations like AND, XOR and PARITY, which leads to very fast implementations. The matrix can be either extensively described bit per bit, or defined as the output of a pseudo-random generator so that the matrix description remains small in size. This can be done very simply by using a congruential generator with different seeds for each line.

The main bottleneck in the algorithm comes from the sampling algorithm that computes a binary vector of given length and weight. We have to use multiplications and divisions, and that is costly in comparison with the previous operations. However, such computations can be greatly speeded up by doing some precomputations, like a table of the binomials that will be used. That leaves us with only substractions and comparisons, yelding a very fast computation.

Recent attacks on cryptosystems based on error-correcting codes [5] have shown that a $(512, 256)$ random linear code can be decoded up to half its minimum distance 58. That means that words of weight up to 30 can be found given their syndrome. So we can choose a weight between 50 and 57 with the assurance of a very good security.

The gain of the generator is $\rho n - \left\lceil \log_2 \binom{n}{\lfloor \delta n \rfloor} \right\rceil$. For a value of ρ fixed to $1/2$, we have the following results:

n	512	512	512	728	728	1024	1024
$\lfloor \delta n \rfloor$	56	55	50	78	71	110	100
gain	5	8	23	11	32	12	43

As seen in the table, $n = 512$ and $\lfloor \delta n \rfloor = 55$ yields one byte per iteration, which looks very attractive.

The scheme as described in section 2 can be improved by precomputing the binomial coefficients. By thus constructing a table of binomials, we get rid of costly multiplications and divisions. The resulting scheme makes only use of comparisons, substractions of n-bit numbers and multiplications of a binary matrix with an n-bit vector. Note that those operations are very fast since they require only logical or fast instructions. The memory cost of a table of the binomial coefficients for the $(512, 256, 55)$ scheme is the following: we need 512×55 entries of size 256 bits, which makes a total of 880 kBytes of memory. This is clearly not an issue for todays computers.

On a SUN Sparc10 station, our implementation using a precomputed binomial table achieves an output rate of 3500 bits per second. RSA with a 512 bit modulus and small exponent is rated at about 0.005 sec per encryption using chinese remainders. If we output $\log_2(n) = 9$ bits per application of the scheme, the output rate is about 1800 bits per second.

4 Conclusion

We have shown a construction for a pseudo-random generator with proven security. This generator is very efficient and simple, and its implementation is fairly straightforward. We hope that this work will encourage research of alternative solutions to number theory.

Acknowledgements

We thank Oded Goldreich and Mike Luby for discussions on the subject of this paper. Also, after this paper has been accepted for EUROCRYPT, we were informed that Ramarathnan Venkatesen had considered a similar construction. His work will appear elsewhere.

A Appendix

Lemma 1 *The product of an n-bit integer by an $\log_2 n$-bit integer is computable in linear time. The same property holds for the division.*

Remark: as will be seen from the proof, this actually holds in a model corresponding to random access machines with logarithmic cost. It is unclear that the result carries over in the Turing machine model.

Proof. Let x be a n-bit integer, y a $\log_2 n$-bit integer, and α a positive real smaller than $1/2$.

We write x and y in base $2^t = 2^{\lfloor \alpha \log_2 n \rfloor}$:

$$x = \sum_{i=0}^{\frac{n}{t}} x_i \cdot 2^{ti} \quad , \quad y = \sum_{j=0}^{\frac{\log_2 n}{t}} y_j \cdot 2^{tj}$$

Then $x \cdot y = \sum_{i,j} x_i \cdot y_j \cdot 2^{t(i+j)}$.

So the multiplication requires $\lceil \frac{n}{t} \rceil \times \left\lceil \frac{\log_2 n}{t} \right\rceil \simeq O(\frac{n}{\log_2 n})$:

- multiplications of two t-bit integers;
- shiftings of t-bit integers (the multiplication by $2^{t(i+j)}$);
- additions of two t-bit integers.

Addition and shifting are done in time linear in the length of the input, which is, in our case, in $O(\log_2 n)$.

It remains to prove that multiplication of two $log_2 n$-bit integers can be done within the same complexity bound. First, we construct once for all a table of all the possible products $x_i \cdot y_j$. Since x_i and y_j are in $\{0, .., 2^t\}$, the size of the table is at most $2^{\alpha \log_2 n} \times 2^{\alpha \log_2 n}$. Using the quadratic-time multiplication, the table can be computed in time $(2^{\alpha \log_2 n})^2 (\alpha \log_2 n)^2$; and since $\alpha < \frac{1}{2}$, this construction is done in time $O(n)$. Table look-up is performed in $O(\alpha \log_2 n + \alpha \log_2 n + \alpha \log_2 n) \simeq O(\log_2 n)$, so the complexity of our multiplication is also $O(\log_2 n)$.

Finally, the multiplication of an n-bit integer by an $\log_2 n$-bit integer requires the construction of a table done in $O(n)$ and $O(\frac{n}{\log_2 n})$ operations which are done in $O(\log_2 n)$, so that the whole scheme has a complexity of $O(n)$.. $\square$

References

1. Alexi, W., Chor, B., Goldreich, O., Schnorr, C. P.: Rsa and rabin functions: certain parts are as hard as the whole. SIAM J. Computing **17** (1988) 194–209.
2. Berlekamp, E. R., McEliece, R. J., van Tilborg, H. C. A.:. On the inherent intractability of certain coding problems. In IEEE Trans. Information Theory (1978) IEEE pp. 384–386.
3. Blum, L., Blum, M., Shub, M.: A simple unpredictible pseudo-random number generator. SIAM J. Computing **15** (1986) 364–383.

4. Blum, M., Micali, S.: How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. Computing **13** (1984) 850–863.
5. Canteaut, A., Chabaud, F.:. A general improvement of the previous attacks on McEliece's cryptosystem. Unpublished.
6. Chabaud, F.:. On the security of some cryptosystems based on error-correcting codes. In Advances in Cryptology: Proc. of EUROCRYPT'94 (1994) LNCS Springer-Verlag.
7. Garey, M. R., Johnson, D. S.:. Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Co 1979.
8. Goldreich, O.:. Foundations of cryptography (Fragments of a book). Weizmann Institut of Science 1995.
9. Goldreich, O., Krawczyk, H., Luby, M.:. On the existence of pseudo-random generators. In Proc. 29th Symp. on Foundations of Computing Science (1988) IEEE pp. 12–24.
10. Goldreich, O., Levin, L. A.:. Hard core predicate for any one-way function. In Proc. 21st Symp. on Theory of Computing (1989) ACM press pp. 25–32.
11. Guillot, P.:. Algorithmes pour le codage à poids constant. Unpublished.
12. Håstad, J., Schrift, A. W., Shamir, A.: The discrete logarithm modulo a composite hides $o(n)$ bits. J. of Computing and Systems Science **47** (1993) 376–404.
13. Impaggliazzo, R., Naor, M.:. Efficient cryptographic schemes provably as secure as subset sum. In Proc. 30th Symp. on Foundations of Computing Science (1989) IEEE pp. 236–241.
14. Impagliazzo, R., Levin, L. A., Luby, M.:. Pseudo-random generation from any one-way functions. In Proc. 21st Symp. on Theory of Computing (1989) ACM press pp. 12–24.
15. Levin, L. A.:. One-way functions and pseudo-random generators. In Proc. 21st Symp. on Theory of Computing (1985) ACM pp. 363–365.
16. Long, D. L., Wigderson, A.: The discrete log hides $o(\log n)$ bits. SIAM J. Computing **17** (1988) 363–372.
17. McWilliams, F. J., Sloane, N. J. A.:. The theory of error-correcting codes. North-Holland 1977.
18. Micali, S., Schnorr, C. P.:. Efficient, perfect random number generators. In Advances in Cryptology, Proc. of CRYPTO'88 (1988) vol. 576 of LNCS Springer Verlag.
19. Stern, J.:. A method for finding codewords of small weight. In Lecture Notes in Computer Science, Coding Theory and Applications vol. 388. Springer 1989 pp. 106–113.
20. Stern, J.:. A new identification scheme based on syndrome decoding. In Advances in Cryptology, Proc. of CRYPTO'93 (1993) vol. 773 of LNCS Springer-Verlag pp. 13–21.
21. Vazirani, U. V., Vazirani, V. V.:. Efficient and secure pseudo-random sequences from slightly-random sources. In Proc. 25th Symp. on Foundations of Computing Science (1984) IEEE pp. 458–463.
22. Yao, A. C.:. Theory and application of trapdoor functions. In Proc. 25th Symp. on Foundations of Computing Science (1982) IEEE pp. 80–91.