Time/Memory/Data Tradeoffs for Variants of the RSA Problem

Pierre-Alain Fouque, Damien Vergnaud, Jean-Christophe Zapalowicz

Time/Memory/Data Tradeoffs for variants of the RSA Problem

Pierre-Alain Fouque[1], Damien Vergnaud[2], and Jean-Christophe Zapalowicz[3]

[1] University of Rennes 1
pierre-alain.fouque@ens.fr
[2] École normale supérieure, Paris, France[†]
[3] INRIA Rennes
jean-christophe.zapalowicz@inria.fr

Abstract. In this paper, we study the security of the Micali-Schnorr pseudorandom number generator. The security of this cryptographic scheme is based on two computational problems which are variants of the RSA problem. The RSA problem essentially aims at recovering the plaintext from a random ciphertext. In the analysis of the Micali-Schnorr pseudorandom generator, we are interested in instances of this problem where the plaintext is small and where the ciphertext is not entirely known. We will describe time / memory tradeoff techniques to solve these hard problems which provides the first analysis of this pseudorandom generator 25 years after its publication.

Keywords: Micali-Schnorr generator, Time/Memory/Data tradeoff

1 Introduction

In this paper we study two cryptographic computational problems related to the RSA problem. Given a modulus N, product of two large prime numbers, and an odd exponent e, coprime to $\varphi(N)$ the order of the multiplicative group $\mathbb{Z}_N^*$, the RSA problem consists in recovering the plaintext $m \in \mathbb{Z}_N^*$ from a random ciphertext $c = m^e \bmod N$. The variants we look at consider particular instances of this problem where the plaintext is small or where the plaintext is small and only a part of the ciphertext is known. These two problems appear to be related to the security of a pseudorandom generator proposed by Micali and Schnorr [21].

A pseudorandom generator is a deterministic polynomial time algorithm that expands short seeds (made of truly random bits) into longer bit sequences, whose distribution cannot be distinguished from uniformly random bits by a computationally bounded algorithm. Pseudorandom number generators are probably the most basic cryptographic primitive: they are widely used for block ciphers, public-key encryption, digital signatures, keystream generation and as passwords sources. It is well-known that pseudorandom generators exist if and only if one-way functions exist [17] (though this generic construction is highly inefficient).

[†] ENS, CNRS & INRIA – UMR 8548

The first practical generator with proven security was proposed by Blum and Micali [6]. The Blum-Micali generator outputs only one bit per iteration – which costs one exponentiation modulo a large prime p – and its security is based on the intractability of the discrete logarithm problem in $\mathbb{Z}_p^*$. It has been shown (e.g. [18]) that the generator remains secure if one outputs $O(\log \log p)$ bits per iteration. Another line of pseudorandom generators is based on factoring-like assumptions (e.g. [5, 21, 24]). The BBS generator was introduced by Blum, Blum and Shub in [5] and proven secure under the assumption that deciding quadratic residuosity modulo a composite Blum[1] integer N is hard. The generator works by repeatedly squaring modulo N a random seed in $\mathbb{Z}_N^*$ and outputs (at each iteration) the least significant bit of the current value. Similarly, the RSA generator works by iterating the RSA encryption mapping $v \mapsto v^e \bmod N$ (for a public RSA modulus N and a public exponent e coprime to $\varphi(N)$) on a secret random initial seed value $v_0 \in \mathbb{Z}_N^*$ to compute the intermediate state values $v_{i+1} = v_i^e \bmod N$ (for $i \in \mathbb{N}$) and outputting the least significant bit of the state value at each iteration. In [1] Alexi *et al.* showed that one can output up to $O(\log \log N)$ bits per iteration of the BBS generator and the RSA generator. The actual number of bits that can be output depends on the concrete parameters adopted but the generators are considered too slow for most applications.

Another line of number-theoretic pseudorandom generators sacrifices provable security for efficiency. Gennaro [15] suggested a discrete-logarithm based generator that outputs $O(\log p)$ bits per modular exponentiation in $\mathbb{Z}_p^*$ but its security is based on a strong and not so well-studied "discrete logarithm with short exponents" assumption. Steinfeld, Pieprzyk and Wang [24] showed, that assuming the hardness of a strong variant of the RSA inversion problem modulo the integer N, one can securely output as much as $(1/2 - 1/e) \log N$ bits in the RSA generator. On the other hand, Herrmann and May [20] showed heuristically (using Coppersmith methods [10, 11]) that an output of $(1 - 1/e) \log N$ most significant bits per iteration allows for efficient recovery of the whole sequence.

Micali and Schnorr [21] proposed a variant of the RSA generator that on a secret random initial seed value $x_0 \in \mathbb{Z}_N^*$ computes the intermediate values $v_i = x_i^e \bmod N$ and outputs, for some $k \in \mathbb{N}$, the k least significant bits of v_i. But the successor x_{i+1} of x_i is formed from a separate part of v_i, the remaining most significant bits (contrary to the *incestuous* RSA generator where $x_{i+1} = v_i$). The security of Micali-Schnorr pseudorandom generator relies on the (strong) assumption that the distribution of $x^e \bmod N$ for random k-bit integers is indistinguishable from the uniform distribution on $\mathbb{Z}_N^*$. The generator is insecure if $(1 - 1/e) \log N$ least significant bits are output per iteration but no better attack was proposed since its proposal 25 years ago. It remains open to know what is the maximum quantity of information that can be output per iteration allowing the generator to be efficient but still secure against potential attackers.

Our Techniques As dynamic programming, time / memory tradeoffs is a well-known technique to reduce the time complexity of a problem using memory.

[1] N is a Blum integer if $N = pq$ with p and q primes and $p, q \equiv 3 \bmod 4$

Shamir and Schroeppel in [23] have described such algorithms for specific NP-complete problems such as knapsack problems. In cryptography, this technique has been used many times to analyze the security of symmetric primitives such as block ciphers or stream ciphers and some computational problems such as the baby-step giant-step algorithm to compute discrete logarithms. Basically, some computations can be done independently of other resources. For instance, using the public key the adversary can precompute some values and store a small fraction of these values in the offline phase. Then, the adversary gets some ciphertexts and his goal can be to recover the secret key.

In [19] Hellman described a technique to invert random looking functions. This technique has been rigorously studied in [13] by Fiat and Naor to work for any functions and rigorous lower bounds have been given in [3] by Barkan, Biham and Shamir. Oeschlin in [22] described a variant of Hellman tradeoff, but this variant has been show equivalent to Hellman tradeoff by Barkan *et al.* since many heuristics can be applied to Hellman technique. Finally, Babbage [2] and Golic [16], then Biryukov and Shamir [4] presented tradeoff for stream cipher by using more or less data. This resource is a crucial parameter in cryptanalysis and it is important to present attacks using as low data complexity as possible.

Our contributions. In this paper, we use time/memory/data tradeoff techniques to propose algorithms for two computational problems related to the security of the Micali-Schnorr pseudorandom generator. The algorithms are decomposed into two phases: the preprocessing one where the attacker constructs large hash tables using the structure of the focused cryptosystem, and the real-time phase where it uses the data produced by the cryptosystem and the hash tables to retrieve the secrets. The three tradeoffs algorithms we describe are similar to the tradeoffs for stream ciphers. However, in order to construct such algorithms, we need to specify the function f we used. For stream ciphers, the main idea is to execute from a hidden state the generator in order to have at least $\log S$ bits of output if the state is of size S. Here, we decide to truncate the output value. It is a bit weird to define f in such a way since the iteration of such functions is no more related to the iteration of the generator. However, the only things we need is to cover the space in such a way that the inversion will be possible. This choice of function f is suitable for Micali-Schnorr generator but does not work for the BBS or the RSA generator. Moreover, in order to prove that the many Hellman tables algorithm works (our third algorithm), we need to prove that each table uses an independent function. We provide such claim in the analysis of the third algorithm. Indeed, this independence assumption is in fact the tricky part of the analysis and Hellman paper relies on heuristic in order to provide lower bounds on the time complexity of his scheme. Using a computational argument we prove that the considered functions are independent.

Our algorithms do not contradict the strong assumption used for the Micali-Schnorr pseudorandom generator. They can be applied even though only a small part of the generator is output at each iteration. Moreover, we will show that once one value is recovered using the algorithms we describe for the first problem, then we are also able to retrieve the seed by using another time/memory tradeoff.

Finally, even if our algorithms beating the bound remain exponential, we achieve to decrease the constant and that can be very interesting in cryptography (for example, in the case of the factorization).

Organization of the paper. In Section 2, we present the first problem we look at and basics about the Micali-Schnorr pseudorandom generator. We explain why the problem is easy for some small parameters. In Section 3, we describe three time/memory algorithms for solving the first problem using different tradeoffs. In Section 4, we show other tradeoffs to recover the seed of the generator.

2 Micali-Schnorr Pseudorandom Generator

The Micali-Schnorr pseudorandom generator is defined by the recursive sequence $(v_i = x_{i-1}^e \mod N)$ for $i \geq 1$, with (e, N) the RSA public key, $x_0 \in [0, 2^r[$ the secret seed of size[2] r and $v_i = 2^k x_i + w_i$. At each iteration, this generator outputs the k least significant bits of v_i, denoted by w_i. In addition, denoting n the size of the modulus N, only x_i of size $r = n - k$, unknown, is reused for the next iteration. Since the generator outputs $O(k/\log e)$ bits per multiplication, one wants k to be as large as possible and e to be as small as possible.
This pseudorandom generator is proven secure under the following assumption:

Assumption 1 *The distribution of $x^e \mod N$ for random r-bit integers is indistinguishable by all polynomial-time statistical tests from the uniform distribution of elements of $\mathbb{Z}_N^*$.*

Description of the problem. Let (e, N) the RSA public key with N of size n. Using the equality $v_i = 2^k x_i + w_i$ where $v_i \in \mathbb{Z}_N$, $w_i \in [0, 2^k)$ and $x_i \in [0, 2^r)$, we consider the recurrence sequence

$$\forall i \geq 1, \quad v_i = x_{i-1}^e \mod N \tag{1}$$

Given (e, N, r), $\{w_1, \cdots, w_j\}$ with $j \in \mathbb{N}$, the problem consists in retrieving one value x_c with $c \in \{0, \cdots, j-1\}$.

For an attacker, finding one of the values x_i using some iterations of the Micali-Schnorr pseudorandom generator will lead to infer its next outputs. The difficulty of the above problem depends highly on the value of r. Figure 1 sums up this hardness, with a transition value equal to n/e. We first explain why it is easy to solve the problem when the size of r is less than n/e.

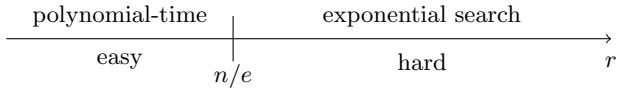

Fig. 1. Difficulty of the problem depending of the value r

[2] Throughout the paper, the *size* of an integer is (an upper-bound of) its bit-size.

Theorem 1. *Suppose that the value x_0 of size r is odd. If $r \leq n/e$, given (e, N) and w_1, there exists a polynomial-time algorithm which retrieves the value x_0.*

Proof. If $r \leq n/e$, the modular reduction is not performed in Equation 1, so $v_1 = x_0^e$ over the integers and using $v_1 = 2^k x_1 + w_1$, one has the following modular equation:

$$x_0^e = w_1 \mod 2^k$$

where all the values except x_0 are known. We now use the well-known Hensel's lifting lemma to retrieve this secret value.

Lemma 1 (Hensel's lifting lemma). *Let p be a prime and c be a positive integer. One denotes f a polynomial having a root x modulo p^c which satisfies:*

$$f(x) = 0 \mod p^c \quad \text{and} \quad f'(x) \neq 0 \mod p$$

Then, one can lift x to obtain an unique nontrivial root $x^* \in [0, p^{c+1})$ verifying:

$$f(x^*) = 0 \mod p^{c+1} \quad \text{and} \quad x^* = x \mod p^c$$

With Lemma 1, by using $f(x) = x^e - w_1$, one can reconstruct bit per bit x_0 looking at the powers of 2. The value x^* can be efficiently computed by $x^* = x + \lambda \cdot 2^c$ where $\lambda = -\frac{f(x)}{2^c} \cdot (f'(x))^{-1} \bmod 2$. □

Note that if the value x_0 is even, one loses the uniqueness of the lift. However, computing $x^e - w_1 \mod 2^k$ for each candidate x of size r can suffice to retrieve this value; else one tests another output w_i of the generator.
Another possibility to retrieve the seed consists in raising w_1 to the power $e^{-1} \bmod 2^{k-1}$ (notice that e is odd). However the complexity of Hensel lifting is linear in the size of the root, contrary to this exponentiation.
To avoid this simple algorithm but to remain efficient, i.e to output a maximum of bits per iteration, the parameter k has to be smaller than $\lfloor n(1 - \frac{1}{e}) \rfloor$. Finally, it seems hard to find a polynomial-time algorithm if $r > n/e$, for example by using Coppersmith techniques, which are techniques bases on lattice reduction to find small modular roots.

3 Solving the Problem using Time/Memory/Data Tradeoffs

For now, we consider the problem in the case where r is larger than n/e and we will present three similar algorithms that use different tradeoffs in order to solve the problem. These algorithms use the fact that only the hidden information, i.e the value x_i of a relatively small size r, is recycled for the next iteration contrary to some other pseudorandom generators as the BBS or the RSA ones. We denote the five key parameters as follows:

- 2^r represents the cardinality of the search space.
- P represents the time required by the preprocessing phase of the algorithm.
- M represents the quantity of access memory required for the algorithm.
- T represents the time required by the online phase of the algorithm.
- D represents the quantity of data required for the algorithm.

3.1 First algorithm

The first algorithm is quite simple to explain and to implement but not really efficient. The preprocessing phase consists in storing the couples $(x, LSB_k(x^e \mod N))$ for some different values of x in a hash table. During the online phase, one tests for each value w_i if it appears in the hash table. For example, it will work by taking $M = 2^{r/3}$ and $D = T = 2^{2r/3}$ or even $M = T = D = 2^{r/2}$. The proof is given in the full version of this paper.

Theorem 2. *Given (e, N) and D consecutive values $w_1, \cdots, w_D$, there exists an algorithm which retrieves one of the values $x_0, \cdots, x_{D-1}$ in time T, by using M random access memory such that $TM = O(2^r)$ with $D = O(T)$.*

3.2 Second algorithm using one Hellman's table

The two next algorithms are based on [19, 4]. Hellman then Biryukov and Shamir have proposed different attacks using tradeoffs for breaking block ciphers and stream ciphers. We define a special function in order to apply these attacks for solving our problem, and thus for the Micali-Schnorr pseudorandom generator. This second algorithm gives the same tradeoff as the first one, but need less data: for $M = 2^{r/3}$ and $T = 2^{2r/3}$, it just requires a bit more than $2^{r/3}$ data.

Theorem 3. *Given (e, N) and D consecutive values $w_1, \cdots, w_D$, there exists an algorithm which retrieves one of the values $x_0, \cdots, x_{D-1}$ in time T, by using M random access memory such that $TM = O(2^r)$ with $T \leq D^2$.*

Proof. **Algorithm.** Let f be the function defined by $f(x) = LSB_r(x^e \mod N)$ where $LSB_r(x)$ represents the r least significant bits of x. The preprocessing phase consists in computing for m random different values $x_0^1, \cdots, x_0^m$ the values $f^t(x_0^1), \cdots, f^t(x_0^m)$ with $m, t \in \mathbb{N}$ and where f^t means that the function f is iterated t times. The construction of a hash table containing the $f^t(x_0^i)$ as keys and the x_0^i as associated values, for $i \in \{1, \cdots, m\}$, concludes this phase.
The algorithm in online phase works as follows:

1. One selects a known value w_j for $j > 0$.
2. One considers only the r least significant bits of w_j, denoted by z_j.
3. For $i \in \{0, \cdots, t\}$, one tests if $f^i(z_j)$ is a key of the hash function. If the $t + 1$ tests fail, one selects the next known value and restarts the algorithm.
4. If a test succeeds, denoting the associated value x_0^c, one has:

$$f^{t-i}(x_0^c) = z_j = f(\underbrace{f^{t-i-1}(x_0^c)}_{X})$$

X is a value of size r that corresponds with high probability to the hidden part of the generator at the previous iteration. A simple verification consists of the computation of the value $X^e \mod N$.

Value	Hellman's Matrix for our algorithm	Key
$\boldsymbol{x_0^1}$	$\xrightarrow{f} f(x_0^1) \xrightarrow{f} \cdots \xrightarrow{f} f^{t-1}(x_0^1) \xrightarrow{f}$	$\boldsymbol{f^t(x_0^1)}$
$\vdots$		$\vdots$
$\boldsymbol{x_0^c}$	$\underbrace{\xrightarrow{f} \cdots \xrightarrow{f} X}_{\text{step 4}} \xrightarrow{f} \underbrace{z_j \xrightarrow{f} \cdots \xrightarrow{f}}_{\text{step 3}}$	$\boldsymbol{f^i(z_j)}$
$\vdots$		$\vdots$
$\boldsymbol{x_0^m}$	$\xrightarrow{f} f(x_0^m) \xrightarrow{f} \cdots \xrightarrow{f} f^{t-1}(x_0^m) \xrightarrow{f}$	$\boldsymbol{f^t(x_0^m)}$

Table 1. Computation of our algorithm using a hash table

Table 1 gives an overview of the algorithm by manipulating the hash table and using the function f.

Complexity. The number of different values in this table can be estimated as follows (the end value of each chain is not counted):

$$E(\#\{f^j(x_0^i), 1 \leq i \leq m, 0 \leq j < m\}) = \sum_{i=1}^{m} \sum_{j=0}^{t-1} Pr[A_{i,j}]$$

where $A_{i,j}$ the event $[f^j(x_0^i) \notin \{f^{j'}(x_0^{i'}), i' < i \text{ or } j' < j\}]$. Note that $A_{i,j} \subseteq A_{i,j-1}$ (since $f^j(x_0^i) = f^{j-1}(x_0^i)$). Moreover, we have the following property:

$$Pr[A_{i,j}|A_{i,j-1}] \geq 1 - \frac{it}{2^r} \Rightarrow Pr[A_{i,j}] \geq \big(1 - \frac{it}{2^r}\big)^{j+1}$$

Hence, the probability p that the value we search is in the Hellman's table is greater than $2^{-r} \sum_{i=1}^{m} \sum_{j=0}^{t-1} (1 - \frac{it}{2^r})^{j+1}$.
By denoting D the number of known values of the recurrence, the time required by the preprocessing phase of the algorithm P is equal to $O(mt)$, the memory M and the time of the algorithm T are defined by $M = O(m)$ and $T = O(Dt)$. For each known value of the recurrence, one has a probability p depending on the size of the table to success, and thus we need $Dp = O(1)$. If $mt^2 \ll 2^r$ then $p \approx 2^{-r}mt$. Consequently we obtain the tradeoff $Dt \cdot m = O(2^r)$, i.e $TM = 2^r$ with $T \leq D^2$ (due to $mt^2 \ll 2^r$).
As the table gets larger, some chains will eventually collide and merge due to the birthday paradox. So it may be preferable to use many small tables, that is the next attack. Finally note that, at step 4, each value $f^i(z_j)$ may have multiple predecessors, hence there is a small probability that $f^{t-i}(x_0^c)$ will not be equal to z_j. In this case, one tries an other output but it is clear that these "false alarms" will increase the complexity by only a small constant factor. □

3.3 Third algorithm using many Hellman tables

This last algorithm which uses more tables, proposes then another repartition between the memory, the time and the data. For example, for $M = 2^{5r/8}, T = 2^{r/2}$ and $D = 2^{r/8}$, this tradeoff is preferable compared to the first two algorithms.

This set of tables covers a larger fraction of the possible output values and consequently, the online phase need less data. Each table requires a specific function and, in order to cover different independent output values, the functions need to be independent. In [19], Hellman rigorously calculated a lower bound on the expected coverage of images by a *single* table which is essentially the same analysis we did in the previous algorithm. However, the analysis for the full scheme (with *many* tables) is highly heuristic and is based on the unjustifiable assumption that many simple variants of f are independent of each other. Fiat and Naor in [13] propose to use k-wise independent functions in order to propose an algorithm to invert *any* function, while Hellman assumes that the function is random. In order to replace the heuristic, one could think of using independent functions for each table by computing $g_i = h_i \circ f$, where $\{h_i\}_i$ is a family of k-wise independent functions. The main drawback is that the number of such functions we need is exponential and it is not easy to construct such functions. Here, we want to avoid Hellman heuristic while similar heuristic could be made. For instance, we could define many functions by considering any r bits among the $n-r$ output bits which will give us $\binom{n-r}{r}$ different functions. However, many functions will have the same subset of bits and we cannot assume independence between them. The analysis of the algorithm is based on the following hypothesis:

Assumption 2 *Denoting $f(x) = LSB_r(x^e \mod N)$, the distribution of $f(x)$ for random r''-bit integers ($r'' \geq r$) x is indistinguishable by all polynomial-time statistical tests from the uniform distribution of integers in $[0, 2^r)$.*

Theorem 4. *Given (e, N) and D consecutive values $w_1, \cdots, w_D$, there exists an algorithm which retrieves one of the values $x_0, \cdots, x_{D-1}$ in time T, by using M random access memory such that $TM^2D^2 = O(2^{2r})$ with $D^2 \leq T \leq 2^r$.*

Proof. **Algorithm.** The third algorithm is similar to the second one but, instead of using a single table, one uses $\ell = t/D$ hash tables of size mt (assuming that $t > D$). First, one has to find which table covers the output value. Then one applies the second algorithm. Consequently, the search of the table requires to look for each value in all tables in parallel.

Complexity. Using the same analysis as for the second algorithm, we know that we cover a fraction $mt/2^r$ of the output values with one table. Now, using ℓ tables, we want to prove that the number of output values we cover is $mt\ell$. To prove such result, we have to solve the independence problem, namely that to describe independent functions for each table so that we are still able to invert f. First of all, the whole output of the $n-r$ least significant bits of $x^e \mod N$ can be used. But this only allows us to construct a constant number of functions. Our second idea is to use the fact that f is a random function or that its outputs are indistinguishable from the uniform distribution (Assumption 2). By using

ℓ random and independent values $z_i \in [0, 2^{r'})$, we can define ℓ functions as $g_i(x) = f(x + z_i \cdot 2^r)$ for $i \in \{1, \cdots, \ell\}$. We claim that this set of functions is independent, otherwise assumption 2 will be wrong for $r'' = r + r'$.
Using the same notations as in the previous proof, the probability p that the value we search is in one of the ℓ Hellman's tables is greater than $1 - \left(1 - 2^{-r} \sum_{i=1}^{m} \sum_{j=0}^{t-1} (1 - \frac{it}{2^r})^{j+1}\right)^{\ell}$ and $p \approx 2^{-r} mt\ell$ if $mt^2 \ll 2^r$. We clearly have $M = O(m\ell)$, $T = O(Dt\ell)$ and $P = O(mt\ell)$. For $\ell = t/D$, we obtain the tradeoff $TM^2D^2 = O(2^{2r})$ with $D^2 \leq T \leq 2^r$. □

Remark 1. This tradeoff is less constraining than Hellman tradeoff: we only need that one value is in one table and not that a particular value.

4 Inverting RSA for Small Plaintext Problem

By using one of the previous algorithms, one knows the value of a hidden part of the generator denoted x_i for $i \geq 0$. We now present two different ways to invert the Micali-Schnorr generator, i.e to retrieve the secret seed x_0.

Description of the problem. Let (e, N) be an RSA public key with N of size n and an integer $r \leq n$. Given (N, e, r) and $y = x^e \bmod N$ for $x \in [0, 2^r)$, the problem consists in recovering x.

Remark 2. This problem is well-known to be solvable in polynomial time when $r \leq n/e$ since as before the equality holds over the integers.

4.1 Multipoint evaluation of univariate polynomials

Let $P(x) \in \mathbb{Z}_N[x]$ be a polynomial of degree less than $n = 2^k$. The multipoint evaluation problem is the task of evaluating P at n distinct points $\alpha_0, \ldots, \alpha_{n-1} \in \mathbb{Z}_N$. Using Horner's rule, it is easy to propose a solution that uses $O(n^2)$ addition and multiplication in $\mathbb{Z}_N$ but it is well-known that one can propose an algorithm with quasi-linear complexity $\tilde{O}(n)$ operations in $\mathbb{Z}_N$ using a divide-and-conquer approach [8, 14].

Let $P_0 = \prod_{\ell=0}^{n/2-1} (x - \alpha_\ell)$ and $P_1 = \prod_{\ell=n/2}^{n-1} (x - \alpha_\ell)$ and let us define $R_0 = P \bmod P_0$ and $R_1 = P \bmod P_1$. We have $R_0(\alpha_i) = P(\alpha_i)$ for all $i \in \{0, \ldots, n/2 - 1\}$ and $R_1(\alpha_i) = P(\alpha_i)$ for all $i \in \{n/2, \ldots, n-1\}$ and this gives immediately a recursive algorithm (*i.e.* compute P_0, P_1, R_0, R_1 and reduce the problem to the multipoint evaluation of R_0 and R_1 of degree $n/2 = 2^{k-1}$).

Let $A_i(x) = (x - \alpha_i)$ for $i \in \{0, \ldots, n-1\}$ and $P_{i,j} = A_{j2^i} A_{j2^i+1} \ldots A_{j2^i+2^i-1}$ for $i \in \{0, \ldots, k\}$ and $0 \leq j < 2^{k-i}$. We have $P_{0,j} = A_j$ and $P_{i+1,j} = P_{i,2j} P_{i,2j+1}$ so for $i \in \{0, \ldots, k\}$ we can compute recursively all polynomials $P_{i,j}$ and $0 \leq j < 2^{k-i}$ in $2^{k-i-1} O(M(2^i)) = O(M(n))$ operations in $\mathbb{Z}_N$ where $M(i)$ denotes the arithmetic complexity to compute the product of two polynomials of degree i in $\mathbb{Z}_N[x]$. Overall, the computation of all polynomials $P_{i,j}$ requires $O(M(n) \log n)$ operations in $\mathbb{Z}_N$ using a tree.

The polynomials R_0 and R_1 can be computed using $O(M(n))$ operations in $\mathbb{Z}_N$ (using a Newton inversion), hence the complexity $T(n)$ of the recursive algorithm satisfies $T(n) = 2T(n/2) + O(M(n))$ and therefore $T(n) = O(M(n) \log n)$.

The multipoint evaluation of univariate polynomials has found numerous application in cryptanalysis (*e.g.* [12, 9]). In our case, it is clear that using this technique will lead to retrieve the seed. For example, using the same notations as in preliminaries, suppose that we know the value of v_i and want to retrieve the value of x_{i-1} of the generator. That can be done by multipoint evaluating the polynomial of degree $e(2^{r/2}+1)$:

$$P(X) = (X^e - v_i)((X+1)^e - v_i)((X+2)^e - v_i)...((X+2^{r/2})^e - v_i) \mod N$$

on the points $k \cdot 2^{r/2}$ for $k = 0, \cdots, 2^{r/2}$ in order to find k_c such that $P(k_c \cdot 2^{r/2}) = 0 \bmod N$. Then, one searches the value of x_{i-1} on the form $k_c \cdot 2^{r/2} + \ell$ for $\ell = 0, \cdots, 2^{r/2}$. This technique requires $\tilde{O}(e \cdot 2^{r/2})$ operations in $\mathbb{Z}_N$. Its complexity is linear in e but, as mentioned above, e is chosen as small as possible in practice. Moreover, one has to store the first tree, i.e $2^{r/2}$ polynomials.

Remark 3. This algorithm can be applied to attack the RSA encryption system when used to encrypt a short secret key of a symmetric cipher. Our algorithm is slightly less efficient than the one in [7] but it always succeeds (whereas recovering a 40-bit plaintext for instance is successful only with probability 0.39 in [7]).

4.2 Coppersmith's method

Another technique is based on the well-known Coppersmith's method for the case of a modular univariate polynomial. In 1996, Coppersmith introduced lattice-based techniques for finding small roots on univariate and bivariate polynomial equations in polynomial time [11, 10]. Some recalls are done in the full version. In our case, starting from the equation $x_{i-1}^e = v_i \mod N$, we can define the following modular univariate polynomial f as $f(x) = x^e - v_i \mod N$. The value x_{i-1} represents a small modular root of this polynomial. However, our root of size r is not enough small for this technique which requires the root to be less than $N^{1/e}$, i.e $r < n/e$ (see [11]). To circumvent this problem, one can guess j bits of x in order to have $r - j < n/e$ and then apply Coppersmith's method for each guess. Instead of f, one uses the polynomial g of degree e:

$$g(x) = (\lambda + x)^e - v_i \mod N$$

with λ the guessed value of j bits. The truncated value of x_{i-1} denoted by x_{i-1}^{tr} is a small modular root of g. Its degree being the same, the asymptotic condition on the size of the root remains the same. The followng theorem, proved in the full version, establishes the condition on the bound:

Theorem 5. *Using the set of polynomials* $\{x^j g^i | j \leq e-1 \wedge i \leq p-1\} \cup \{g^p\}$ *with* $p \in \mathbb{N}$*, Coppersmith's method will return* x_{i-1}^{tr} *as long as* $x_{i-1}^{tr} < N^\delta$ *with:*

$$\delta = \frac{\sum_{i=1}^{p}(e-1)(i-1)+i}{\sum_{i=1}^{ep} i} = \frac{ep - e + 2}{e^2 p + 2}$$

Starting from a basis $(b_1, \cdots, b_w)$ of a lattice of $\mathbb{Z}^m$, this technique works in complexity $O(w^4 m \log B(w + \log B))$ with $B = \max_{1 \leq i \leq w} \|b_i\|$, and we have to store the lattice of size $w + m$. By denoting $x_{i-1}^{tr} < N^{\frac{1}{e} \cdot (1-\epsilon)}$ with $0 < \epsilon < 1$, one can determine the minimal value for p in order to retrieve the root, i.e $p = \frac{e-1-\epsilon}{e\epsilon} = O(1/\epsilon)$. The number of polynomials w being equal to $e(p-1)+1 = O(\frac{e}{\epsilon})$, those of monomials m being equal to $ep + 1 = O(\frac{e}{\epsilon})$ and $\log B = n$, this technique will require $O(2^{r - \frac{n}{e} \cdot (1+\epsilon)}(\frac{e^6 n}{\epsilon^6} + \frac{e^5 n^2}{\epsilon^5}))$ in time and $O(\frac{e}{\epsilon})^2$ in memory.

5 Conclusion

In this paper, for the first time, we have shown that, for all recommended parameters, we are able to predict the Micali-Schnorr pseudorandom generator faster than by an exhaustive search by using time/memory tradeoff or time/memory/data tradeoff attacks. These attacks are feasible only because of the specificity of this generator that uses only a small number to iterate and it remains a open problem to design a time/memory tradeoff algorithm able to infer sequences produced by the BBS or the RSA generator (in the range of parameters not covered by Herrmann and May techniques [20]).

We have also proposed three techniques (the last one is explained in the full version) to reverse the generator and retrieve the generator seed. An interesting open question is to decrease the memory requirement of our algorithms. The ρ or λ methods for factoring and discrete logarithms (which were invented by Pollard) use pseudorandom walks and require polynomial (or even constant) memory rather than exponential as in our time/memory tradeoffs. They can be applied to attack Gennaro's efficient pseudorandom generator based on the discrete logarithm [15] but it remains open to adapt this approach to predict the Micali-Schnorr pseudorandom generator.

Acknowledgements. This work was supported in part by the French ANR-12-JS02-0004 ROMAnTIC Project.

References

1. W. Alexi, B. Chor, O. Goldreich, and C.-P. Schnorr, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput. **17** (1988), no. 2, 194–209.
2. S. Babbage, *A space/time tradeoff in exhaustive search attacks on stream ciphers*, IEE Conference Publication - European Convention on Security and Detection, vol. 408, 1995.
3. E. Barkan, E. Biham, and A. Shamir, *Rigorous bounds on cryptanalytic time/memory tradeoffs*, Advances in Cryptology – Crypto 2006 (C. Dwork, ed.), LNCS, vol. 4117, Springer, August 2006, pp. 1–21.
4. A. Biryukov and A. Shamir, *Cryptanalytic time/memory/data tradeoffs for stream ciphers*, Advances in Cryptology – Asiacrypt 2000 (T. Okamoto, ed.), LNCS, vol. 1976, Springer, December 2000, pp. 1–13.
5. L. Blum, M. Blum, and M. Shub, *A simple unpredictable pseudo-random number generator*, SIAM J. Comput. **15** (1986), no. 2, 364–383.

6. M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM J. Comput. **13** (1984), no. 4, 850–864.
7. D. Boneh, A. Joux, and P. Q. Nguyen, *Why textbook ElGamal and RSA encryption are insecure*, Advances in Cryptology – Asiacrypt 2000 (T. Okamoto, ed.), LNCS, vol. 1976, Springer, December 2000, pp. 30–43.
8. A. Bostan, P. Gaudry, and É. Schost, *Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator*, SIAM J. Comput. **36** (2007), no. 6, 1777–1806.
9. Y. Chen and P. Q. Nguyen, *Faster algorithms for approximate common divisors: Breaking fully-homomorphic-encryption challenges over the integers*, Advances in Cryptology – Eurocrypt 2012, LNCS, Springer, 2012, pp. 502–519.
10. D. Coppersmith, *Finding a small root of a bivariate integer equation; factoring with high bits known*, Advances in Cryptology – Eurocrypt'96 (U. M. Maurer, ed.), LNCS, vol. 1070, Springer, May 1996, pp. 178–189.
11. ———, *Finding a small root of a univariate modular equation*, Advances in Cryptology – Eurocrypt'96 (U. M. Maurer, ed.), LNCS, vol. 1070, Springer, May 1996, pp. 155–165.
12. J.-S. Coron, A. Joux, A. Mandal, D. Naccache, and M. Tibouchi, *Cryptanalysis of the RSA subgroup assumption from TCC 2005*, PKC 2011: 14th International Workshop on Theory and Practice in Public Key Cryptography(D. Catalano, N. Fazio, R. Gennaro, and A. Nicolosi, eds.), LNCS, vol. 6571, Springer, March 2011, pp. 147–155.
13. A. Fiat and M. Naor, *Rigorous time/space trade-offs for inverting functions*, SIAM J. Comput. **29** (1999), no. 3, 790–803.
14. C. Fiduccia, *Polynomial evaluation via the division algorithm: The Fast Fourier Transform revisited*, 4th Annual ACM Symposium on Theory of Computing (P. Fischer, H. P. Zeiger, J. Ullman, and A. Rosenberg, eds.), ACM, 1972, pp. 88–93.
15. R. Gennaro, *An improved pseudo-random generator based on the discrete logarithm problem*, Journal of Cryptology **18** (2005), no. 2, 91–110.
16. J. D. Golic, *Cryptanalysis of alleged A5 stream cipher*, Advances in Cryptology – Eurocrypt'97 (W. Fumy, ed.), LNCS, vol. 1233, Springer, May 1997, pp. 239–255.
17. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby, *A pseudorandom generator from any one-way function*, SIAM J. Comput. **28** (1999), no. 4, 1364–1396.
18. J. Håstad and M. Näslund, *The security of all RSA and discrete log bits*, J. ACM **51** (2004), no. 2, 187–230.
19. M. E. Hellman, *A cryptanalytic time-memory trade-off*, IEEE Transactions on Information Theory **26** (1980), no. 4, 401–406.
20. M. Herrmann and A. May, *Attacking power generators using unravelled linearization: When do we output too much?*, Advances in Cryptology – Asiacrypt 2009 (M. Matsui, ed.), LNCS, vol. 5912, Springer, December 2009, pp. 487–504.
21. S. Micali and C.-P. Schnorr, *Efficient, perfect polynomial random number generators*, Journal of Cryptology **3** (1991), no. 3, 157–172.
22. P. Oechslin, *Making a faster cryptanalytic time-memory trade-off*, Advances in Cryptology – Crypto 2003 (D. Boneh, ed.), LNCS, vol. 2729, Springer, August 2003, pp. 617–630.
23. R. Schroeppel and A. Shamir, *A* $T = O(2^{n/2})$, $S = O(2^{n/4})$ *algorithm for certain NP-complete problems*, SIAM J. Comput. **10** (1981), no. 3, 456–464.
24. R. Steinfeld, J. Pieprzyk, and H. Wang, *On the provable security of an efficient RSA-based pseudorandom generator*, Advances in Cryptology – Asiacrypt 2006 (X. Lai and K. Chen, eds.), LNCS, vol. 4284, Springer, December 2006, pp. 194–209.