Attacking Power Generators Using Unravelled Linearization: When Do We Output Too Much?[⋆]

Mathias Herrmann, Alexander May

Horst Görtz Institute for IT-Security
Faculty of Mathematics
Ruhr University Bochum, Germany
`mathias.herrmann@rub.de, alex.may@rub.de`

Abstract. We look at iterated power generators $s_i = s_{i-1}^e \bmod N$ for a random seed $s_0 \in \mathbb{Z}_N$ that in each iteration output a certain amount of bits. We show that heuristically an output of $(1 - \frac{1}{e}) \log N$ most significant bits per iteration allows for efficient recovery of the whole sequence. This means in particular that the Blum-Blum-Shub generator should be used with an output of less than half of the bits per iteration and the RSA generator with $e = 3$ with less than a $\frac{1}{3}$-fraction of the bits.
Our method is lattice-based and introduces a new technique, which combines the benefits of two techniques, namely the method of linearization and the method of Coppersmith for finding small roots of polynomial equations. We call this new technique *unravelled linearization*.

Keywords: power generator, lattices, small roots, systems of equations

1 Introduction

Pseudorandom number generators (PRGs) play a crucial role in cryptography. An especially simple construction is provided by iterating the RSA function $s_i = s_{i-1}^e \bmod N$ for an RSA modulus $N = pq$ of bit-size n and a seed $s_0 \in \mathbb{Z}_N$. This so-called power generator outputs in each iteration a certain amount of bits of s_i, usually the least significant bits. In order to minimize the amount of computation per iteration, one typically uses small e such as $e = 3$. With slight modifications one can choose $e = 2$ as well when replacing the iteration function by the so-called absolute Rabin function [3, 4], where $s^2 \bmod N$ is defined to be $\min\{s^2 \bmod N, N - s^2 \bmod N\}$, N is a Blum integer and s_0 is chosen from $\{0, \dots, \frac{N-1}{2}\}$ with Jacobi symbol $+1$.

It is well-known that under the RSA assumption one can safely output up to $\Theta(\log n) = \Theta(\log \log N)$ bits per iteration [1, 8]. At Asiacrypt 2006, Steinfeld, Pieprzyk and Wang [14] showed that under a stronger assumption regarding the optimality of some well-studied lattice attacks, one can securely output $(\frac{1}{2} -$

⋆ This research was supported by the German Research Foundation (DFG) as part of the project MA 2536/3-1 and by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

$\frac{1}{e} - \epsilon - o(1))n$ bits. The assumption is based on a specific RSA one-wayness problem, where one is given an RSA ciphertext $c = m^e \bmod N$ together with a certain fraction of the plaintext bits of m, and one has to recover the whole plaintext m. We call this generator the SPW generator. The SPW generator has the desirable property that one can output a constant fraction $\Omega(\log N)$ of all bits per iteration. Using an even stronger assumption, Steinfeld, Pieprzyk and Wang could improve the output size to $(\frac{1}{2} - \frac{1}{2e} - \epsilon - o(1))n$ bits.

A natural question is whether the amount of output bits of the SPW generator is maximal. Steinfeld et al.'s security proof uses in a black-box fashion the security proof of Fischlin and Schnorr for RSA bits [8]. This proof unfortunately introduces a factor of $\frac{1}{2}$ for the output rate of the generator. So, Steinfeld et al. conjecture that one might improve the rate to $(1 - \frac{1}{e} - \epsilon)n$ using a different proof technique. Here, ϵ is a security parameter and has to be chosen such that performing $2^{\epsilon n}$ operations is infeasible. We show that this bound is essentially the best that one can hope for by giving an attack up to the bound $(1 - \frac{1}{e})n$.

In previous cryptanalytic approaches, upper bounds for the number of output bits have been studied by Blackburn, Gomez-Perez, Gutierrez and Shparlinski [2]. For $e = 2$ and a class of PRGs similar to power generators (but with prime moduli), they showed that provably $\frac{2}{3}n$ bits are sufficient to recover the secret seed s_0. As mentioned in Steinfeld et al., this bound can be generalized to $(1 - \frac{1}{e+1})n$ using the heuristic extension of Coppersmith's method [7] to multivariate equations.

Our contribution: We improve the cryptanalytic bound to $(1-\frac{1}{e})n$ bits using a new heuristic lattice-based technique. Notice that the two most interesting cases are $e = 2, 3$, the Blum-Blum-Shub generator and the RSA generator. For these cases, we improve on the best known attack bounds from $\frac{2}{3}n$ to $\frac{1}{2}n$ and from $\frac{3}{4}n$ to $\frac{2}{3}n$, respectively. Unfortunately — similar to the result of Blackburn et al. [2] — our results are restricted to power generators that output most significant bits in each iteration. It remains an open problem to show that the bounds hold for least significant bits as well.

Our improvement comes from a new technique called *unravelled linearization*, which is a hybrid of lattice-based linearization (see [13] for an overview) and the lattice-based technique due to Coppersmith [7]. Let us illustrate this new technique with a simple example. Assume we want to solve a polynomial equation $x^2 + ax + b = y \bmod N$ for some given $a, b \in \mathbb{Z}_N$ and some unknowns x, y. This problem can be considered as finding the modular roots of a univariate polynomial $f(x) = x^2 + ax + b$ with some error y.

It is a well-known heuristic that a linear modular equation can be easily solved by computing a shortest lattice vector, provided that the absolute value of the product of the unknowns is smaller than the modulus [13]. In order to linearize our equation, we substitute $u := x^2$ and end up with a linear equation in u, x, y. This can be solved whenever $|uxy| < N$. If we assume for simplicity that the unknowns x, y are of the same size, this yields the condition $|x| < N^{\frac{1}{4}}$.

However, in the above case it is easy to see that this linearization is not optimal. A better linearization would define $u := x^2 - y$, leaving us with a linear equation in u, x only. This yields the superior condition $|x| < N^{\frac{1}{3}}$. So one benefits from the fact that one can easily glue variables together, in our case x^2 and y, whenever this does not change the size of the larger variable. In our example this would also work when y had a known coefficient c of size $|c| \approx |y|$.

The main benefit from the attack of Blackburn et al. [2] comes from a clever linearization of the variables that occur in the case of power generators. While on the one hand such a linearization of a polynomial equation offers some advantages, on the other hand we lose the algebraic structure. Performing e.g. the substitution $u := x^2$, one obtains a linear equation in u, x, y but the property that u and x are algebraically dependent — one being the square of the other — is completely lost. Naturally, this drawback becomes more dramatic when looking at higher degree polynomials.

As a consequence, Coppersmith [6, 5, 7] designed in 1996 a lattice-based method that is well-suited for exploiting polynomial structures. The underlying idea is to additionally use algebraic relations before linearization. Let us illustrate this idea with our example polynomial $f(x, y) = x^2 + ax + b - y$. We know that whenever f has a small root modulo N, then also $xf = x^3 + ax^2 + bx - xy$ shares this root. Using xf as well, we obtain *two* modular equations in five unknowns x^3, x^2, x, y, xy. Notice that the unknowns x^2 and x are re-used in the second equation which reflects the algebraic structure. So even after linearizing both equations, Coppersmith's method preserves some polynomial structure. In addition to multiplication of f by powers of x and y — which is often called shifting in the literature — one also allows for powers f^i with the additional benefit of obtaining equations modulo larger moduli N^i.

When we compute the enabling condition with Coppersmith's method for our example $f(x, y)$ using an optimal shifting and powering, we obtain a bound of $|x| < N^{\frac{1}{3}}$. So the method yields a better bound than naive linearization, but cannot beat the bound of the more clever linearization with $u := x^2 - y$. Even worse, Coppersmith's method results in the use of lattices of much larger dimension.

To summarize, linearization makes use of the similarity of coefficients in a polynomial equation, whereas Coppersmith's method basically makes use of the structure of the polynomial's monomial set.

Motivation for unravelled linearization: Our new technique of *unravelled linearization* aims to bring together the best of both worlds. Namely, we allow for clever linearization but still exploit the polynomial structure. *Unravelled linearization* proceeds in three steps: linearization, basis construction, and unravellation. Let us illustrate these steps with our example $f(x, y)$, where we use the linearization $u := x^2 - y$ in the first step. In this case, we end up with a linear polynomial $g(u, x)$. Similar to Coppersmith's approach, in the second step we use shifts and powers of this polynomial. E.g., g^2 defines an equation in the unknowns u^2, ux, x^2, u, x modulo N^2. But since we start with a *linear* polynomial g, this alone will not bring us any benefits, because the algebraic structure got lost in the linearization process from f to g.

Therefore, in the third step we partially unravel the linearization for g^2 using the relation $x^2 = y + u$. The unravelled form of g^2 defines a modular equation in the unknowns u^2, ux, y, u, x, where we basically substitute the unknown x^2 by the unknown y. Notice here, that we can reuse the variable u which occurs in g^2 anyway. This substitution leads to a significant gain, since y is much smaller in size than x^2.

In the present paper, we elaborate on this simple observation that *unravelling of linearization* brings benefits to lattice reduction algorithms. We use the equations that result from the power generator as a case study for demonstrating the power of *unravelled linearization*, but we are confident that our new technique will also find new applications in various other contexts.

The paper is organized as follows. In Section 2 we will fix some very basic notions for lattices. In Section 3 we define our polynomials from the power generator with $e = 2$ and give a toy example with only two PRG iterations that illustrates how *unravelled linearization* works. This already leads to an improved bound of $\frac{7}{11}n$. In Section 4 we generalize to arbitrary lattice dimension (bound $\frac{3}{5}n$) and in Section 5 we generalize to an arbitrary number of PRG iterations (bound $\frac{1}{2}n$). In Section 6 we finally generalize to an arbitrary exponent e. Since our attacks rely on Coppersmith-type heuristics, we verify the heuristics experimentally in Section 7.

2 Basics on Lattices

Let $\mathbf{b_1}, \dots, \mathbf{b_d} \in \mathbb{Q}^d$ be linearly independent. Then the set

$$L := \left\{ \mathbf{x} \in \mathbb{Q}^d \mid \mathbf{x} = \sum_{i=1}^{d} a_i \mathbf{b_i}, a_i \in \mathbb{Z} \right\}$$

is called a lattice L with basis matrix $B \in \mathbb{Q}^{d \times d}$, having the vectors $\mathbf{b_1}, \dots, \mathbf{b_d}$ as row vectors. The parameter d is called the lattice dimension, denoted by $\dim(L)$. The determinant of the lattice is defined as $\det(L) := |\det(B)|$.

The famous LLL algorithm [10] computes a basis consisting of short and pairwise almost orthogonal vectors. Let $\mathbf{v_1}, \dots, \mathbf{v_d}$ be an LLL-reduced lattice basis with Gram-Schmidt orthogonalized vectors $\mathbf{v_1^*}, \dots, \mathbf{v_d^*}$. Intuitively, the property of pairwise almost orthogonal vectors $\mathbf{v_1}, \dots, \mathbf{v_d}$ implies that the norm of the Gram-Schmidt vectors $\mathbf{v_1^*}, \dots, \mathbf{v_d^*}$ cannot be too small. This is quantified in the following theorem of Jutla [9] that follows from the LLL paper [10].

Theorem 1 (LLL). *Let L be a lattice spanned by $B \in \mathbb{Q}^{d \times d}$. On input B, the L^3-algorithm outputs an LLL-reduced lattice basis $\{\mathbf{v_1}, \dots, \mathbf{v_d}\}$ with*

$$\|\mathbf{v_i^*}\| \geq 2^{\frac{1-i}{4}} \left(\frac{\det(L)}{b_{max}^{d-i}} \right)^{\frac{1}{i}} \qquad \textit{for } i = 1, \dots, d$$

in time polynomial in d and in the bit-size of the largest entry b_{max} of the basis matrix B.

3 Power Generators with $e = 2$ and Two Iterations

Let us consider power generators defined by the recurrence sequence

$$s_i = s_{i-1}^e \bmod N,$$

where N is an RSA modulus and $s_0 \in \mathbb{Z}_N$ is the secret seed.

Suppose that the power generator outputs in each iteration the most significant bits k_i of s_i, i.e. $s_i = k_i + x_i$, where the k_i are known for $i \geq 1$ and the x_i are unknown.

Our goal is to recover all x_i for a number of output bits k_i that is as small as possible. In other word, if we define $x_i < N^\delta$ then we have to find an attack that maximizes δ.

Let us start with the most simple case of two iterations and $e = 2$. The best known bound is $\delta = \frac{1}{3}$ due to Blackburn et al. [2]. We will later generalize to an arbitrary number of iterations and also to an arbitrary e.

For the case of two iterations, we obtain

$$s_1 = k_1 + x_1 \quad \text{and} \quad s_2 = k_2 + x_2,$$

for some unknown s_i, x_i. The recurrence relation of the generator $s_2 = s_1^2 \bmod N$ yields $k_2 + x_2 = (k_1 + x_1)^2 \bmod N$, which results in the polynomial equation

$$x_1^2 - x_2 + \underbrace{2k_1}_{a} x_1 + \underbrace{k_1^2 - k_2}_{b} = 0 \bmod N.$$

Thus, we search for small modular roots of $f(x_1, x_2) = x_1^2 - x_2 + ax_1 + b$ modulo N.

Let us first illustrate our new technique called *unravelled linearization* with a small-dimensional lattice attack before we apply it in full generality in Section 4.

Step 1: Linearize $f(x_1, x_2)$ into g.
We make the substitution $u := x_1^2 - x_2$. This leaves us with a linear polynomial $g(u, x_1) = u + ax_1 + b$.
Step 2: Basis construction.
Defining standard shifts and powers for g is especially simple, since g is a linear polynomial. If we fix a total degree bound of $m = 2$, then we choose g, xg and g^2.

Let $X := N^\delta$ be an upper bound for x_1, x_2. Then $U := N^{2\delta}$ is an upper bound for u. The choice of the shift polynomials results in a lattice L spanned by the rows of the lattice basis B depicted in Figure 1.

Let (u_0, x_0) be a root of g. Then the vector $\mathbf{v} = (1, x_0, x_0^2, u_0, u_0x_0, u_0^2, k_1, k_2, k_3)B$ has its right-hand three last coordinates equal to 0 for suitably chosen $k_i \in \mathbb{Z}$. Hence we can write $\mathbf{v}$ as $\mathbf{v} = (1, \frac{x_0}{X}, \ldots, \frac{u_0^2}{U^2}, 0, 0, 0)$. Since $|u_0| \leq U$ and $|x_0| \leq X$, we obtain $\|\mathbf{v}\| \leq \sqrt{6}$.

$$\begin{array}{ccccccccc}
 & & & & & & \boldsymbol{g} & \boldsymbol{x_1 g} & \boldsymbol{g^2} \\
1 & & & & & & b & & b^2 \\
 & \frac{1}{X} & & & & & a & b & ab \\
 & & \frac{1}{X^2} & & & & & a & a^2 \\
 & & & \frac{1}{U} & & & 1 & & b \\
 & & & & \frac{1}{UX} & & & 1 & a \\
 & & & & & \frac{1}{U^2} & & & 1 \\
 & & & & & & N & & \\
 & & & & & & & N & \\
 & & & & & & & & N^2
\end{array}$$

Fig. 1: After linearization and standard shifts and powers for $m = 2$.

To summarize, we are looking for a short vector $\mathbf{v}$ in the 6-dimensional sublattice $L' = L \cap (\mathbb{Q}^6 \times 0^3)$ with $\|\mathbf{v}\| \leq \sqrt{\dim(L')}$. Let $\mathbf{b_1}, \ldots, \mathbf{b_6}$ be an LLL-reduced basis of L' with orthogonalized basis $\mathbf{b_1^*}, \ldots, \mathbf{b_6^*}$. Coppersmith [7] showed that any vector $\mathbf{v} \in L'$ that is smaller than $\mathbf{b_6^*}$ must lie in the sub-space spanned by $\mathbf{b_1}, \ldots, \mathbf{b_5}$, i.e. $\mathbf{v}$ is orthogonal to $\mathbf{b_6^*}$. This immediately yields a coefficient vector of a polynomial $h(u, x_1)$, which has the same roots as $g(u, x_1)$, but over the integers instead of modulo N. Assume that we can find two such polynomials h_1, h_2, then we can compute all small roots by resultant computation provided that h_1, h_2 do not share a common divisor. The only heuristic of our method is that the polynomials h_1, h_2 are indeed coprime.

By the LLL-Theorem (Theorem 1), an orthogonalized LLL-basis contains a vector $\mathbf{b_d^*}$ in L' with $\|\mathbf{b_d^*}\| \geq c(d) \det(L')^{\frac{1}{d}}$, where $c(d) = 2^{\frac{1-d}{4}}$. Thus, if the condition

$$c(d) \det(L')^{\frac{1}{d}} \geq \sqrt{d}$$

holds, then $\bar{\mathbf{v}} = (1, \frac{x_0}{X}, \ldots, \frac{u_0^2}{U^2})$ will be orthogonal to the vector $\mathbf{b_d^*}$.

Since $\det(L')$ is a function of N, we can neglect $d = \dim(L')$ for large enough N. This in turn simplifies our condition to

$$\det(L') \geq 1.$$

Moreover, one can show by a unimodular transformation of B that $\det(L') = \det(L)$.

For our example, the enabling condition $\det(L) \geq 1$ translates to $U^4 X^4 \leq N^4$. Plugging in the values of $X := N^\delta$ and $U := N^{2\delta}$, this leads to the condition $\delta \leq \frac{1}{3}$. Notice that this is exactly the condition from Blackburn et al. [2]. Namely, if the PRG outputs $\frac{2}{3}n$ bits per iteration, then the remaining $\frac{1}{3}n$ bits can be found in polynomial time.

We will now improve on this result by unravelling the linearization of g.

Step 3: Unravel g's linearization.
We unravel the linearization by back-substitution of $x_1^2 = u + x_2$. This slightly changes our lattice basis (see Fig. 2).

$$\begin{array}{cccccc|ccc}
 & & & & & & \boldsymbol{g} & \boldsymbol{x_1 g} & \boldsymbol{g^2} \\
1 & & & & & & b & & b^2 \\
 & \frac{1}{X} & & & & & a & b & ab \\
 & & \frac{\mathbf{1}}{\mathbf{X}} & & & & & a & a^2 \\
 & & & \frac{1}{U} & & & 1 & a & a^2+b \\
 & & & & \frac{1}{UX} & & & 1 & a \\
 & & & & & \frac{1}{U^2} & & & 1 \\
 & & & & & & N & & \\
 & & & & & & & N & \\
 & & & & & & & & N^2
\end{array}$$

Fig. 2: After unravelling the linearization.

The main difference is that the determinant of the new lattice L_u increases by a factor of X. Thus our enabling condition $\det(L_u) \geq 1$ yields $U^4X^3 \leq N^4$ or equivalently $\delta \leq \frac{4}{11}$. This means that if the PRG outputs $\frac{7}{11}n$ of the bits in each of two iterations, then we can reconstruct the remaining $\frac{4}{11}n$ bits of both iterations in polynomial time. This beats the previous bound of $\frac{1}{3}n$.

We would like to stress again that our approach is heuristic. We construct two polynomials h_1, h_2. [1] The polynomials h_1, h_2 contain a priori three variables x_1, x_2, u, but substituting u by $x_1^2 - x_2$ results in two bivariate polynomials h_1', h_2'. Then, we hope that h_1' and h_2' are coprime and thus allow for efficient root finding. We verified this heuristic with experiments in Section 7.

4 Generalization to Lattices of Arbitrary Dimension

The linearization step from $f(x_1, x_2)$ to $g(u, x_1)$ is done as in the previous section using $u := x_1^2 - x_2$. For the basis construction step, we fix an integer m and define the following collection of polynomials

$$g_{i,j}(u, x_1) := x_1^j g^i(u, x_1) \qquad \text{for } i = 1, \ldots, m \text{ and } j = 0, \ldots, m - i. \qquad (1)$$

In the unravelling step, we substitute each occurrence of x_1^2 by $u + x_2$ and change the lattice basis accordingly. It remains to compute the determinant of the resulting lattice. This appears to be a non-trivial task due to the various back-substitutions. Therefore, we did not compute the lattice determinant as a function of m by hand. Instead, we developed an automated process that might be useful in other contexts as well.

We observe that the determinant can be calculated by knowing first the product of all monomials that appear in the collection of the $g_{i,j}$ after unravelling,

[1] The polynomial h_2 can be constructed from $\mathbf{b}^*_{\mathbf{d-1}}$ with a slightly more restrictive condition on $\det(L)$ coming from Theorem 1. However, in practical experiments the simpler condition $det(L) \geq 1$ seems to suffice for h_2 as well. In the subsequent chapters, this minor detail is captured by the asymptotic analysis.

and second the product of all N. Let us start with the product of the N, since it is easy to compute from Equation (1):

$$\prod_{i=1}^{m}\prod_{j=0}^{m-i} N^i = \prod_{i=1}^{m} N^{(m+1)i-i^2} = N^{\frac{1}{6}m^3+o(m^3)}.$$

Now let us bound the product of all monomials. Each variable x_1, x_2, u appears in the unravelled form of $g_{i,j}$ with power at most $2m$. Therefore, the product of all monomials that appear in all $\frac{1}{2}m^2 + o(m^2)$ polynomials has in each variable degree at most m^3. Thus, we can express the exponent of each variable as a polynomial function in m of degree 3 with rational coefficients — similar to the exponent of N.

But since we know that the exponents are polynomials in m of degree at most 3, we can uniquely determine them by a polynomial interpolation at 4 points. Namely, we explicitly compute the unravelled basis for $m = 1, \ldots, 4$ and count the number of variables that occur in the unravelled forms of the $g_{i,j}$. From these values, we interpolate the polynomial function for arbitrary m.

This technique is much less error-prone than computing the determinant functions by hand and it allows for analyzing very complicated lattice basis structures. Applying this interpolation process to our unravelled lattice basis, we obtain $\det(L) = X^{-p_1(m)}U^{-p_2(m)}N^{p_3(m)}$ with

$$p_1(m) = \frac{1}{12}m^3 + o(m^3), \quad p_2(m) = \frac{1}{6}m^3 + o(m^3), \quad p_3(m) = \frac{1}{6}m^3 + o(m^3).$$

Our condition $\det(L) \geq 1$ thus translates into $\frac{5}{12}\delta \leq \frac{1}{6}$ resp. $\delta \leq \frac{2}{5}$. Interestingly, this is exactly the bound that Blackburn et al. [2] conjectured to be the best possible bound one can obtain by looking at two iterations of the PRG.

In the next section, we will also generalize our result to an arbitrary fixed number of iterations of the PRG. This should intuitively help to further improve the bounds and this intuition turns out to be true. To the best of our knowledge, our attack is the first one that is capable of exploiting more than two equations in the contexts of PRGs.

5 Using an Arbitrary Fixed Number of PRG Iterations

We illustrate the basic idea of generalizing to more iterations by using three iterations of the generator before analyzing the general case.

Let $s_i = k_i + x_i$ for $i = 1, 2, 3$, where the k_i are the output bits and the x_i are unknown. For these values, we are able to use two iterations of the recurrence relation, namely

$$s_2 = s_1^2 \bmod N \qquad s_3 = s_2^2 \bmod N$$

from which we derive two polynomials

$$f_1 : \underbrace{x_1^2 - x_2}_{u_1} + \underbrace{2k_1}_{a_1} x_1 + \underbrace{k_1^2 - k_2}_{b_1} = 0 \bmod N$$
$$f_2 : \underbrace{x_2^2 - x_3}_{u_2} + \underbrace{2k_1}_{a_2} x_2 + \underbrace{k_2^2 - k_3}_{b_2} = 0 \bmod N.$$

We perform the linearization step $f_1 \to g_1$ and $f_2 \to g_2$ by using the substitutions $u_1 := x_1^2 - x_2$ and $u_2 := x_2^2 - x_3$.

In the basis construction step, we have to define a collection for the polynomials $g_1(u_1, x_1)$ and $g_2(u_2, x_2)$ using suitable shifts and powers. We will start by doing this in some generic but non-optimal way, which is depicted in Figure 3 for the case of fixed total degree $m = 2$ in g_1, g_2. In this basis matrix for better readability we leave out the left-hand diagonal consisting of the inverses of the upper bounds of the corresponding monomials.

$$\begin{array}{c|ccccccccc}
 & \boldsymbol{g_1} & \boldsymbol{x_1 g_1} & \boldsymbol{g_1^2} & \boldsymbol{g_2} & \boldsymbol{x_2 g_2} & \boldsymbol{g_2^2} & \boldsymbol{x_2 g_1} & \boldsymbol{x_1 g_2} & \boldsymbol{g_1 g_2} \\
1 & b_1 & & b_1^2 & b_2 & & b_2^2 & & & b_1 b_2 \\
x_1 & a_1 & b_1 & a_1 b_1 & & & & & b_2 & a_1 b_2 \\
x_2 & & a_1 & a_1^2 & a_2 & b_2 & a_2 b_2 & b_1 & & a_2 b_1 \\
x_3 & & & & & a_2 & a_2^2 & & & \\
u_1 & 1 & a_1 & a_1^2 + b_1 & & & & & & b_2 \\
u_1 x_1 & & 1 & a_1 & & & & & & \\
u_1^2 & & & 1 & & & & & & \\
u_2 & & & & 1 & a_2 & a_2^2 + b_2 & & & b_1 \\
u_2 x_2 & & & & & 1 & a_2 & & & \\
u_2^2 & & & & & & 1 & & & \\
x_1 x_2 & & & & & & & a_1 & a_2 & a_1 a_2 \\
u_1 x_2 & & & & & & & 1 & & a_2 \\
u_2 x_1 & & & & & & & & 1 & a_1 \\
u_1 u_2 & & & & & & & & & 1 \\
 & N & & & & & & & & \\
 & & N & & & & & & & \\
 & & & N^2 & & & & & & \\
 & & & & N & & & & & \\
 & & & & & N & & & & \\
 & & & & & & N^2 & & & \\
 & & & & & & & N & & \\
 & & & & & & & & N & \\
 & & & & & & & & & N^2
\end{array}$$

Fig. 3: Generic lattice basis for 2 polynomials

The reader may verify that the bound obtained from this collection of polynomials is $\delta \leq \frac{4}{11} \approx 0.364$, which is exactly the same bound as in our starting example in Section 3. A bit surprisingly, our generic lattice basis construction does not immediately improve on the bound that we derived from a single polynomial.

It turns out, however, that we improve when taking just a small subset of the collection in Fig. 3. If we only use the shifts $g_1, x_1 g_1, g_1^2$ and additionally g_2, then we obtain a superior bound of $\delta \leq \frac{5}{13} \approx 0.385$. The reason for the improvement comes from the fact that the monomial x_2 of g_2 can be reused as it already appeared in the shifts $x_1 g_1$ and g_1^2.

For the asymptotic analysis, we define the following collection of polynomials

$$g_{i,j,k} := x_1^k g_1^i g_2^j \qquad \text{for} \begin{cases} i = 0, \ldots, m \\ j = 0, \ldots, \left\lfloor \frac{m-i}{2} \right\rfloor \\ k = 0, \ldots, m - i - 2j \end{cases} \qquad \text{with } i + j \geq 1.$$

The intuition behind the definition of this collection of polynomials follows the same reasoning as in the example for $m = 2$. We wish to keep small the number of new monomials introduced by the shifts with g_2. Notice that the monomials x_2^i for $i = 0, \ldots \left\lfloor \frac{m}{2} \right\rfloor$ already appeared in the g_1 shifts — since we back-substituted $x_1^2 \to u_1 + x_2$. Therefore, it is advantageous to use the g_2 shifts only up to $\left\lfloor \frac{m}{2} \right\rfloor$.

With the interpolation technique introduced in Section 4, we derive a bound of $\delta \leq \frac{6}{13}$ for the case of 2 polynomials, i.e. three output values of the generator.

5.1 Arbitrary Number of PRG Iterations

Given $n + 1$ iterations of the PRG, we select a collection of shift polynomials following the intuition given in the previous section:

$$g_{i_1,\ldots,i_n,k} := x_1^k g_1^{i_1} \ldots g_n^{i_n}$$

$$\text{for} \begin{cases} i_1 &= 0, \ldots, m \\ i_2 &= 0, \ldots, \left\lfloor \frac{m-i_1}{2} \right\rfloor \\ \vdots & \\ i_n &= 0, \ldots, \left\lfloor \frac{m - \sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor \\ k &= 0, \ldots, m - \sum_{j=1}^{n} 2^{j-1} i_j \end{cases} \qquad \text{with } i_1 + \ldots + i_n \geq 1.$$

To perform the asymptotic analysis we need to determine the value of the determinant of the corresponding lattice basis. This means, we have to count the exponents of all occurring monomials in the set of shift polynomials. We would like to point out that because of the range of the index k, the shifts with x_1^k do not introduce additional monomials over the set defined by the product of the g_i alone. For this product the monomials can be enumerated as follows (see Appendix A for a proof):

$$x_1^{a_1} \ldots x_n^{a_n} u_1^{i_1 - a_1} \ldots u_{n-1}^{i_{n-1} - a_{n-1}} u_n^{i_n - 2b_n - a_n} x_{n+1}^{b_n}$$

$$\text{with}\begin{cases} i_1 = 0, \ldots, m & a_1 = 0,1 \\ i_2 = 0, \ldots, \left\lfloor \frac{m-i_1}{2} \right\rfloor & a_2 = 0,1 \\ \vdots & \vdots \\ i_n = 0, \ldots, \left\lfloor \frac{m-\sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor & a_n = 0,1 \\ b_n = 0, \ldots, \left\lfloor \frac{i_n - a_n}{2} \right\rfloor . \end{cases}$$

We are only interested in the asymptotic behavior, i.e. we just consider the highest power of m. We omit the floor function as it only influences a lower order term. Analogously, we simplify the exponents of u_j by omitting the value a_j, since it is a constant polynomial in m. Furthermore, for the same reason the contribution to the determinant of all x_i with $i \leq n$ can be neglected.

To derive the final condition, we have to compute the polynomials $p_j(m)$ of the following expression for the determinant (resp. the coefficients of the highest power of m):

$$\det(L) = X_{n+1}^{-p_x(m)} U_1^{-p_1(m)} \ldots U_n^{-p_n(m)} N^{p_N(m)}.$$

It seems to be a complicated task to compute these polynomials explicitly. Therefore, we follow a different approach and compute the sizes of their leading coefficients in relation to each other. This turns out to be enough to derive a bound on the sizes of the unknowns. In Appendix B we explain how to derive the following expressions for the polynomials:

$$p_j(m) = \frac{1}{2^{j-1}} p_1(m) \text{ for } j \leq n, \quad p_x(m) = \frac{1}{2^n} p_1(m), \quad p_N(m) = \frac{2^n - 1}{2^{n-1}} p_1(m),$$

where we again omit low order terms. We use these expressions in the enabling condition $\det(L) \geq 1$ and plug in upper bounds $X_{n+1} \leq N^\delta$ and $U_i \leq N^{2\delta}$. It is sufficient to consider the condition for the exponents:

$$\delta \frac{1}{2^n} p_1(m) + 2\delta \sum_{j=1}^{n} \frac{1}{2^{j-1}} p_1(m) \leq \frac{2^n - 1}{2^{n-1}} p_1(m).$$

Simplifying this condition and solving for δ, we obtain

$$\delta \leq \frac{2^{n+1} - 2}{2^{n+2} - 3},$$

which converges for $n \to \infty$ to $\delta \leq \frac{1}{2}$.

6 Extending to Higher Powers

In the previous sections, we have considered PRGs with exponent $e = 2$ only, i.e. a squaring operation in the recurrence relation. A generalization to arbitrary exponents is straight forward.

Suppose the PRG has the recurrence relation $s_2 = s_1^e \bmod N$. Let, as in Section 3, the output of the generator be k_1, k_2, i.e. we have $s_1 = k_1 + x_1$ and $s_2 = k_2 + x_2$, for some unknown s_i, x_i.
Using the recurrence relation, this yields the polynomial equation

$$\underbrace{x_1^e - x_2}_{u} + ek_1x_1^{e-1} + \ldots + ek_1^{e-1}x_1 + \underbrace{k_1^e - k_2}_{b} = 0 \bmod N.$$

The linearization step is analog to the case where $e = 2$, however, the unravelling of the linearization only applies for higher powers of x_1, in this case x_1^e.
The collection of shift polynomials using n PRG iterations is

$$g_{i_1,\ldots,i_n,k} := x_1^k g_1^{i_1} \ldots g_n^{i_n}$$

$$\text{for} \begin{cases} i_1 & = 0, \ldots, m \\ i_2 & = 0, \ldots, \left\lfloor \frac{m-i_1}{e} \right\rfloor \\ \vdots & \\ i_n & = 0, \ldots, \left\lfloor \frac{m-\sum_{j=1}^{n-1} e^{j-1} i_j}{e^{n-1}} \right\rfloor \\ k & = 0, \ldots, m - \sum_{j=1}^{n} e^{j-1} i_j \end{cases} \qquad \text{with } i_1 + \ldots + i_n \geq 1.$$

Taking a closer look at the analysis in Appendix A and B shows that the generalization for arbitrary e is straightforward. Working through the analysis we obtain for arbitrary e an asymptotic bound for an arbitrary number of polynomials of $\delta \leq \frac{1}{e}$.

7 Experiments

Since our technique uses a heuristic concerning the algebraic independence of the obtained polynomials, we have to experimentally verify our results. Therefore, we implemented the unravelled linearization using SAGE 3.4.1. including the L^2 reduction algorithm from Nguyen and Stehlé [12]. In Table 1 some experimental results are given for a PRG with $e = 2$ and 256 bit modulus N.

polys	m	δ	exp. δ	dim(L)	time(s)
1	4	0.377	0.364	15	1
1	6	0.383	0.377	28	5
1	8	0.387	0.379	45	45
2	4	0.405	0.390	22	10
2	6	0.418	0.408	50	1250
3	4	0.407	0.400	23	5

Table 1: Experimental Results for $e = 2$

In the first column we denote the number of polynomials. The second column shows the chosen parameter m, which has a direct influence on how close we approach the asymptotic bound. On the other hand, the parameter m increases the lattice dimension and therefore the time required to compute a solution. The theoretically expected δ is given in the third column, whereas the actually verified δ is given in the fourth column. The last column denotes the time required to find the solution on a Core2 Duo 2.2 GHz running Linux 2.6.24.

It is worth mentioning that most of the time to find the solution is not spend on doing the lattice reduction, but for extracting the common root from the set of polynomials using resultant computations. The resultant computations yielded the desired solutions of the power generators.

Acknowledgement: We would like to thank Dan Bernstein for bringing this research topic to our attention during an Ecrypt meeting.

References

1. Michael Ben-Or, Benny Chor, and Adi Shamir. On the cryptographic security of single rsa bits. In *STOC*, pages 421–430. ACM, 1983.
2. Simon R. Blackburn, Domingo Gomez-Perez, Jaime Gutierrez, and Igor Shparlinski. Reconstructing noisy polynomial evaluation in residue rings. *J. Algorithms*, 61(2):47–59, 2006.
3. Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986.
4. Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13(4):850–864, 1984.
5. Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Maurer [11], pages 178–189.
6. Don Coppersmith. Finding a small root of a univariate modular equation. In Maurer [11], pages 155–165.
7. Don Coppersmith. Small solutions to polynomial equations, and low exponent rsa vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.
8. Roger Fischlin and Claus-Peter Schnorr. Stronger security proofs for rsa and rabin bits. *J. Cryptology*, 13(2):221–244, 2000.
9. Charanjit S. Jutla. On finding small solutions of modular multivariate polynomial equations. In *EUROCRYPT*, pages 158–170, 1998.
10. Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
11. Ueli M. Maurer, editor. *Advances in Cryptology - EUROCRYPT '96, International Conference on the Theory and Application of Cryptographic Techniques, Saragossa, Spain, May 12-16, 1996, Proceeding*, volume 1070 of *Lecture Notes in Computer Science*. Springer, 1996.
12. Phong Q. Nguyen and Damien Stehlé. Floating-point lll revisited. In *EUROCRYPT*, pages 215–233, 2005.
13. Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In Joseph H. Silverman, editor, *CaLC*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180. Springer, 2001.

14. Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. On the provable security of an efficient rsa-based pseudorandom generator. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2006.

A Describing the Set of Monomials

Theorem 1 *Suppose we have n polynomials of the form*

$$f_i(x_i, x_{i+1}) = x_i^2 + a_i x_i + b_i - x_{i+1}$$

and define the collection of polynomials

$$f_1^{i_1} \dots f_n^{i_n} \quad \text{for} \begin{cases} i_1 & = 0, \dots, m \\ i_2 & = 0, \dots, \left\lfloor \frac{m-i_1}{2} \right\rfloor \\ \vdots & \\ i_n & = 0, \dots, \left\lfloor \frac{m - \sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor. \end{cases}$$

After performing the substitutions $x_i^2 \mapsto u_i + x_{i+1}$, the set of all occurring monomials can be described as

$$x_1^{a_1} \dots x_n^{a_n} u_1^{i_1 - a_1} \dots u_{n-1}^{i_{n-1} - a_{n-1}} u_n^{i_n - 2b_n - a_n} x_{n+1}^{b_n}$$

$$\text{with} \begin{cases} i_1 = 0, \dots, m & a_1 = 0, 1 \\ i_2 = 0, \dots, \left\lfloor \frac{m-i_1}{2} \right\rfloor & a_2 = 0, 1 \\ \vdots & \vdots \\ i_n = 0, \dots, \left\lfloor \frac{m - \sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor & a_n = 0, 1 \\ b_n = 0, \dots, \left\lfloor \frac{i_n - a_n}{2} \right\rfloor. & \end{cases}$$

Proof. By induction: **Basic step:** $n = 1$
For one polynomial $f_1(x_1, x_2) = x_1^2 + a_1 x_1 + b_1 - x_2$ we perform the substitution $x_1^2 \mapsto u_1 + x_2$ to obtain $g_1(u_1, x_1) = u_1 + a_1 x_1 + b_1$. The set of all monomials that are introduced by the powers of $g_1(u_1, x_1)$ can be described as

$$x_1^{j_1} u_1^{i_1 - j_1} \quad \text{for} \begin{cases} i_1 & = 0, \dots, m \\ j_1 & = 0, \dots, i_1. \end{cases}$$

It remains to perform the substitution on this set. Therefore, we express the counter j_1 by two counters a_1 and b_1 and let $j_1 = 2b_1 + a_1$, i.e. we write the set as

$$(x_1^2)^{b_1} x_1^{a_1} u_1^{i_1 - 2b_1 - a_1} \quad \text{for} \begin{cases} i_1 & = 0, \dots, m \\ a_1 & = 0, 1 \\ b_1 & = 0, \dots, \left\lfloor \frac{i_1 - a_1}{2} \right\rfloor. \end{cases}$$

Imagine that we enumerate the monomials for fixed i_1, a_1 and increasing b_1, and simultaneously perform the substitution $x_1^2 \mapsto u_1 + x_2$. The key point to notice is that all monomials that occur after the substitution, i.e. all of $(u_1 + x_2)^{b_1} x_1^{a_1} u_1^{i_1-2b_1-a_1}$, have been enumerated by a previous value of b_1, except for the single monomial $x_2^{b_1} x_1^{a_1} u_1^{i_1-2b_1-a_1}$.
Thus, the set of monomials after the substitution can be expressed as

$$x_2^{b_1} x_1^{a_1} u_1^{i_1-2b_1-a_1} \quad \text{for} \begin{cases} i_1 &= 0, \ldots, m \\ a_1 &= 0, 1 \\ b_1 &= 0, \ldots, \left\lfloor \frac{i_1-a_1}{2} \right\rfloor. \end{cases}$$

This concludes the basic step.

Inductive Step: $n-1 \to n$
Suppose the assumption is correct for $n-1$ polynomials. By the construction of the shift polynomials and the induction hypothesis, we have the set of monomials

$$\underbrace{x_1^{a_1} \ldots x_{n-1}^{a_{n-1}} u_1^{i_1-a_1} \ldots u_{n-2}^{i_{n-2}-a_{n-2}} u_{n-1}^{i_{n-1}-2b_{n-1}-a_{n-1}} x_n^{b_{n-1}}}_{\text{Hypothesis}} \underbrace{x_n^{j_n} u_n^{i_n-j_n}}_{f_n}$$

$$\text{for} \quad \begin{cases} i_1 = 0, \ldots, m & a_1 = 0, 1 \\ i_2 = 0, \ldots, \left\lfloor \frac{m-i_1}{2} \right\rfloor & a_2 = 0, 1 \\ \vdots & \vdots \\ i_{n-1} = 0, \ldots, \left\lfloor \frac{m-\sum_{j=1}^{n-2} 2^{j-1} i_j}{2^{n-2}} \right\rfloor & a_{n-1} = 0, 1 \\ b_{n-1} = 0, \ldots, \left\lfloor \frac{i_{n-1}-a_{n-1}}{2} \right\rfloor & \\ i_n = 0, \ldots, \left\lfloor \frac{m-\sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor & j_n = 0, \ldots, i_n. \end{cases}$$

By adding the n-th polynomial, we also get the new relation $x_n^2 = u_n + x_{n+1}$. Before performing the substitutions, however, we have to take a closer look at the powers of x_n. The problem seems to be that we have a contribution from the n-th polynomial as well as from some previous substitutions. It turns out that this can be handled quite elegantly. Namely, we will show that all occurring monomials are enumerated by just taking $b_{n-1} = 0$.
Consider the set of monomials for $b_{n-1} = c$ for some constant $c \geq 1$:

$$x_1^{a_1} \ldots u_{n-1}^{i_{n-1}-2c-a_{n-1}} x_n^{j_n+c} \quad \text{for } j_n \in \{0, \ldots, i_n\}.$$

Exactly the same set of monomials is obtained by considering the index $i'_{n-1} = i_{n-1} - 2$ and $b_{n-1} = c - 1$. Notice that in this case the counter i'_n, which serves as an upper bound of j'_n, runs from 0 through

$$\begin{aligned} \left\lfloor \frac{m - \sum_{j=1}^{n-2} 2^{j-1} i_j - 2^{n-2} i'_{n-1}}{2^{n-1}} \right\rfloor &= \left\lfloor \frac{m - \sum_{j=1}^{n-2} 2^{j-1} i_j - 2^{n-2} i_{n-1} + 2^{n-1}}{2^{n-1}} \right\rfloor \\ &= i_n + 1. \end{aligned}$$

Thus, we have the same set of monomials as with $b_{n-1} = c - 1$:

$$x_1^{a_1} \dots u_{n-1}^{i'_{n-1}-2(c-1)-a_{n-1}} x_n^{j'_n+(c-1)} \text{ for } j'_n \in \{0, \dots, i'_n\}.$$

Iterating this argument, we conclude that all monomials are enumerated by $b_{n-1} = 0$.

Having combined the occurring powers of x_n, we continue by performing an analog step as in the basic step: introduce a_n and b_n representing j_n. This leads to

$$x_1^{a_1} \dots u_{n-1}^{i_{n-1}-a_{n-1}} (x_n^2)^{b_n} x_n^{a_n} u_n^{i_n-2b_n-a_n}$$

$$\text{for } \begin{cases} i_1 = 0, \dots, m & a_1 = 0, 1 \\ i_2 = 0, \dots, \left\lfloor \frac{m-i_1}{2} \right\rfloor & a_2 = 0, 1 \\ \vdots & \vdots \\ i_{n-1} = 0, \dots, \left\lfloor \frac{m-\sum_{j=1}^{n-2} 2^{j-1} i_j}{2^{n-2}} \right\rfloor & a_{n-1} = 0, 1 \\ i_n = 0, \dots, \left\lfloor \frac{m-\sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor & a_n = 0, 1 \\ b_n = 0, \dots, \left\lfloor \frac{i_n-a_n}{2} \right\rfloor . \end{cases}$$

Finally we substitute $x_n^2 = u_n + x_{n+1}$. Using the same argument as in the basic step, we note that new monomials only appear for powers of x_{n+1}.

B Relations among Exponent Polynomials

For the determinant computation we need to sum up the exponents of the occurring monomials. Take for example u_ℓ with $\ell < n$: using the description of the set from Appendix A, we need to compute

$$\sum_{i_1=0}^{m} \sum_{a_1=0}^{1} \sum_{i_2=0}^{\left\lfloor \frac{m-i_1}{2} \right\rfloor} \sum_{a_2=0}^{1} \cdots \sum_{i_n=0}^{\left\lfloor \frac{m-\sum_{j=1}^{n-1} 2^{j-1} i_j}{2^{n-1}} \right\rfloor} \sum_{a_n=0}^{1} \sum_{b_n=0}^{\left\lfloor \frac{i_n-a_n}{2} \right\rfloor} (i_\ell - a_\ell) \, .$$

We will step by step simplify this expression using the fact that in the asymptotic consideration only the highest power of the parameter m is important.

In the first step we notice that we may remove the $-a_\ell$ from the summation, because a_ℓ does not depend on m, while i_ℓ does. Therefore, the a_ℓ just affects lower order terms. With the same argument we can omit the a_n in the upper bound of the sum over b_n. Further, the floor function in the limit of the sums does only affect lower order terms and therefore may be omitted. Next, we can move all the sums of the a_i to the front, since they are no longer referenced anywhere, and replace each of these sums by a factor of 2, making altogether a global factor of 2^n.

For further simplification of the expression, we wish to eliminate the fractions that appear in the bounds of the sums. To give an idea how to achieve this, consider the expression

$$\sum_{i_1=0}^{m} \sum_{i_2=0}^{\frac{m-i_1}{2}} i_2.$$

Our intuition is to imagine an index i'_2 of the second sum that performs steps with a width of 2 and is upper bounded by $m - i_1$. To keep it equivalent, we have to compute the sum of over all integers of the form $\left\lfloor \frac{i'_2}{2} \right\rfloor$. However, when changing the index to i'_2, the sum surely does not perform steps with width 2. I.e. we count every value exactly twice. Thus, to obtain a correct reformulation, we have to divide the result by 2. Note that asymptotically we may omit the floor function and simply sum over $\frac{i'_2}{2}$.

In the same way we are able to reformulate all sums from i_1 to i_n. For better readability we replaced i'_j with i_j again.

$$2^n \cdot \frac{1}{2} \cdot \frac{1}{4} \cdot \ldots \cdot \frac{1}{2^{n-1}} \sum_{i_1=0}^{m} \sum_{i_2=0}^{m-i_1} \cdots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \sum_{b_n=0}^{\frac{i_n}{2^n}} \frac{1}{2^{\ell-1}} i_\ell. \qquad (2)$$

It seems to be a complicated task to explicitly evaluate a sum of this form. Therefore, we follow a different approach, namely we relate the sums over different i_ℓ to each other. We start with the discussion of a slightly simpler observation:

Sums of the form $\sum_{i_1=0}^{m} \sum_{i_2=0}^{m-i_1} \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} i_\ell$ are equal for all $\ell \leq n$.

An explanation can be given as follows. Imagine the geometric object that is represented by taking the i_j as coordinates in an n-dimensional space. This set describes an n-dimensional simplex, e.g. a triangle for $n = 2$, a tetrahedron for $n = 3$, etc. Considering its regular structure, i.e. the symmetry in the different coordinates, it should be clear that the summation over each of the i_ℓ results in the same value.

In the sum of Equation (2) there is an additional inner summation with index b_n and limit $i_n/2^n$. For the indices $\ell < n$ this innermost sum is constant for all values of ℓ and thus with the previous argumentation the whole sums are equal for all $\ell < n$. We only have to take care of the leading factors, i.e. the powers of 2 that came from replacing the summation variables.

This gives us already a large amount of the exponent polynomials in the determinant expression. Namely, we are able to formulate the polynomials p_ℓ (which is the sum over the i_ℓ) in terms of p_1 for all $\ell < n$. The difference is exactly the factor $\frac{1}{2^{\ell-1}}$ that has been introduced when changing the index from i_ℓ to i'_ℓ.

For the exponent polynomial of the variable u_n, however, we have to be careful because we do not compute the summation of $i_n - a_n$, but of $i_n/2^{n-1} - 2b_n - a_n$ instead ($i_n/2^{n-1}$ since we changed the summation index i_n). The value

$-a_n$ can be omitted with the same argument as before. To derive a relation of p_n to p_1, we start by evaluating the inner sums:

$$p_1 : \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \sum_{b_n=0}^{\frac{i_n}{2^n}} i_1 = \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \frac{i_n}{2^n} i_1$$

$$p_n : \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \sum_{b_n=0}^{\frac{i_n}{2^n}} \left(\frac{i_n}{2^{n-1}} - 2b_n \right) = \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \left(\frac{i_n^2}{2^{2n-1}} - 2\frac{1}{2}\frac{i_n^2}{2^{2n}} \right)$$

$$= \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \frac{i_n^2}{2^{2n-1}}.$$

Notice that once again, for the asymptotic analysis we have only considered the highest powers.

Because of the previously mentioned symmetry between i_1 and i_n, we finally derive $p_n = \frac{1}{2^{n-1}} p_1$. The same argument can be used to derive the bound on the variable x_{n+1} for which we have to compute the sum

$$p_x : \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \sum_{b_n=0}^{\frac{i_n}{2^n}} b_n = \ldots \sum_{i_n=0}^{m-\sum_{j=1}^{n-1} i_j} \frac{i_n^2}{2^{2n}}.$$

The multiplicative relation between p_1 and p_x is therefore $p_x = \frac{1}{2^n} p_1$.

Finally, to compute the exponent of N in the determinant, we have to sum up all exponents that occur in the enumeration of the shift polynomials given in Section 5.1. The simplifications are equivalent to the ones used before and we obtain:

$$p_N = \sum_{\ell=1}^{n} \left(\frac{1}{2} \cdot \frac{1}{4} \cdot \ldots \cdot \frac{1}{2^{n-1}} \sum_{i_1=0}^{m} \ldots \sum_{i_n=0}^{m-\sum_{j=0}^{n-1} i_j} \sum_{k=0}^{m-\sum_{j=0}^{n} i_j} \frac{1}{2^{\ell-1}} i_\ell \right).$$

We first note that for $\ell < n$ we may write

$$\ldots \frac{1}{2^{\ell-1}} i_\ell \sum_{i_n=0}^{c} \sum_{k=0}^{c-i_n} 1 \qquad \text{with } c = m - \sum_{j=0}^{n-1} i_j.$$

This is asymptotically equivalent to

$$\ldots \frac{1}{2^{\ell-1}} i_\ell \sum_{i_n=0}^{c} \sum_{k=0}^{i_n} 1 = 2^n \cdot \ldots \frac{1}{2^{\ell-1}} i_\ell \sum_{i_n=0}^{c} \sum_{k=0}^{\frac{i_n}{2^n}} 1 = \frac{1}{2^{\ell-1}} p_1.$$

For $\ell = n$ we argue again that the summations for different i_ℓ behave the same way. Thus it follows $\frac{1}{2} \cdot \frac{1}{4} \cdot \ldots \cdot \frac{1}{2^{n-1}} \sum_{i_1=0}^{m} \ldots \sum_{i_n=0}^{m-\sum_{j=0}^{n-1} i_j} \sum_{k=0}^{m-\sum_{j=0}^{n} i_j} \frac{i_n}{2^{n-1}} = \frac{1}{2^{n-1}} p_1$. Summing up, we obtain

$$p_N = (1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{n-1}}) p_1 = \frac{2^n - 1}{2^{n-1}} p_1.$$