Easing Coppersmith Methods Using Analytic Combinatorics: Applications to Public-Key Cryptography with Weak Pseudorandomness

Fabrice Benhamouda[1](✉), Céline Chevalier[2](✉), Adrian Thillard[1,3](✉), and Damien Vergnaud[1](✉)

[1] ENS, CNRS, INRIA, and PSL, Paris, France
{fabrice.ben.hamouda,damien.vergnaud}@ens.fr, adrianthillard@gmail.com
[2] CRED, Université Panthéon-Assas, Paris, France
celine.chevalier@ens.fr
[3] ANSSI, Paris, France

Abstract. The *Coppersmith methods* is a family of lattice-based techniques to find small integer roots of polynomial equations. They have found numerous applications in cryptanalysis and, in recent developments, we have seen applications where the number of unknowns and the number of equations are non-constant. In these cases, the combinatorial analysis required to settle the complexity and the success condition of the method becomes very intricate.

We provide a toolbox based on *analytic combinatorics* for these studies. It uses the structure of the considered polynomials to derive their generating functions and applies complex analysis techniques to get asymptotics. The toolbox is versatile and can be used for many different applications, including multivariate polynomial systems with arbitrarily many unknowns (of possibly different sizes) and simultaneous modular equations over different moduli. To demonstrate the power of this approach, we apply it to recent cryptanalytic results on number-theoretic pseudorandom generators for which we easily derive precise and formal analysis. We also present new theoretical applications to two problems on RSA key generation and randomness generation used in padding functions for encryption.

Keywords: Coppersmith methods · Analytic combinatorics · Cryptanalysis · Pseudorandom generators · RSA key Generation · Encryption padding

1 Introduction

Many important problems in (public-key) cryptanalysis amount to solving polynomial equations with partial information about the solutions. In 1996, Coppersmith introduced two celebrated lattice-based techniques [11–13] for finding small roots of polynomial equations. They have notably found many important

C.-M. Cheng et al. (Eds.): PKC 2016, Part II, LNCS 9615, pp. 36–66, 2016.
DOI: 10.1007/978-3-662-49387-8_3

applications in the cryptanalysis of the RSA cryptosystem (see [27] and references therein). The first technique works for a univariate modular polynomial whereas the second one deals with a bivariate polynomial over the integers. In these methods, a family of polynomials is first derived from the polynomial whose roots are wanted; this family naturally gives a lattice basis and short vectors of this lattice possibly provide the wanted roots. Since 1996 many generalizations of the methods have been proposed to deal with more variables (e.g., [8,20,22]) or multiple moduli (e.g., [28–30]).

Most of the applications of the Coppersmith methods in cryptanalysis involve a constant number of multivariate polynomial equations in a constant number of variables. However, in recent developments, we have seen applications of the methods where the number of unknowns is non-constant (e.g., [2,18,29]). These applications typically involve a number-theoretic pseudorandom number generator that works by iterating an algebraic map on a secret random initial seed value and outputting the state value at each iteration. It has been shown that in many cases Coppersmith's methods can be applied to recover some secret value. The difficulty comes from the fact that the polynomial system to solve involves all iterates of the pseudorandom generator. It is very tedious to analyze the attack complexity (i.e., the dimension of the lattice derived from the polynomial system whose roots are wanted) and its success condition (i.e., the total degrees of the polynomials and monomials families used in the lattice construction). For instance in [2,18], this analysis is a bit loose; it uses a nice simplifying trick in order to analyze the condition of success but does not permit to estimate the attack complexity. The main intent of this paper is to promote the use of *analytic combinatorics* in order to perform these computations. In order to demonstrate the power of this approach, we apply it to known cryptanalytic results [2] for which we easily derive precise and formal analysis. We also present new theoretical applications to two problems that were left open in [16] on RSA key generation and randomness generation used in padding functions for encryption.

Prior Work. As illustrations of our toolbox, we apply it to the following problems from the literature:

- *Number-theoretic pseudorandom generators.* A *pseudorandom generator* is a deterministic algorithm that maps a random seed to a longer string that cannot be distinguished from uniformly random bits by a computationally bounded algorithm. As mentioned above, a *number-theoretic* pseudorandom generator iterates an algebraic map F over a residue ring $\mathbb{Z}_N$ on a secret random seed $v_0 \in \mathbb{Z}_N$ and computes the intermediate states $v_{i+1} = F(v_i) \bmod N$ for $i \in \mathbb{N}$. It outputs (some consecutive bits of) the state value v_i at each iteration. The well-known *linear congruential generator* corresponds to the case where F is an affine function. It is efficient and has good statistical properties but Boyar [10] proved that one can recover the seed in time polynomial in the bit-size of M and this is also the case even if one outputs only the most significant bit of each v_i (see [9,23,31]). In [2], Bauer, Vergnaud, and Zapalowicz studied the security of number-theoretic generators for rational map F and proposed attacks based on Coppersmith's techniques showing that for low degree F the generators are polynomial time predictable if sufficiently many consecutive bits of the v_i's are revealed (see also [5,6]). Their lattice constructions are intricate and the analysis of their attacks is complex.

– *Key generation and Paddings from weak pseudorandom generator.* The former attacks assume that the adversary has direct access to sufficiently many consecutive bits of a certain number of outputs. However, it may be possible that using such a generator in a cryptographic protocol does not make the resulting protocol insecure. For instance, in [25], Koshiba proved that the linear congruential generator can be used to generate randomness in the ElGamal encryption scheme (based on some plausible assumption). This security results holds because the adversary against ElGamal encryption scheme does not have access to the actual outputs of the generator. *A contrario*, in 1997, Bellare et al. [4] broke the Digital Signature Algorithm (DSA) when the random nonces used in signature generation are computed using a linear congruential generator. Recently, Fouque et al. [16] analyzed the security of public-key schemes when the secret keys are constructed by concatenating the outputs of a linear congruential generator. They obtained a time/memory tradeoff on the search for the seed when such generators are used to generate the prime factors of an RSA modulus (using multipoint polynomial evaluation). They left open the problem to extend it to different scenarios, such as the generation of randomness used in padding functions for encryption and signatures.

Technical Tools. In *Coppersmith methods*, one usually considers an irreducible multivariate polynomial f defined over $\mathbb{Z}$, having a small root $\boldsymbol{x}$ modulo a known integer N and one generates a collection of polynomials having $\boldsymbol{x}$ as a modular root (usually, multiples and powers of f are chosen). The problem of finding $\boldsymbol{x}$ can be reformulated by constructing a matrix using the collection of polynomials (see Sect. 2). The methods succeed (heuristically) if some conditions on the matrix hold and these conditions can be checked by enumerating the polynomials involved in the collection and the total degree of the monomials appearing in the collection. The success condition is usually stated as a bound $\boldsymbol{x} < N^{\delta}$ where δ is an asymptotic explicit constant derived from the combinatorial analysis. However, in order to actually reach this bound in practice, the constructed matrix is of huge dimension and the computation which is theoretically polynomial-time becomes in practice prohibitive[1]. These attacks based on this method are obviously strong evidence of a weakness in the underlying cryptographic scheme and there exist method that makes it possible to use matrices of reasonable dimension

[1] Following Lipton's terminology we can often qualify as *galactic* the resulting polynomial-time algorithm for the asymptotic value of δ [26].

(e.g., by performing an exhaustive to retrieve a small part of $\boldsymbol{x}$ and Coppersmith technique with a smaller matrix to retrieve the other (bigger) part).

The combinatorial analysis in Coppersmith methods is usually easy to perform but as mentioned above it can be very intricate if one considers multivariate polynomial equations in a non-constant number of variables. *Analytic combinatorics* is a celebrated technique — which was mostly developed by Flajolet and Sedgewick [15] — of counting combinatorial objects. It uses the structure of the objects considered to derive their generating functions and applies complex analysis techniques to get asymptotics.

Contributions. The main contribution of the paper is to provide a toolbox based on analytic combinatorics for the study of the complexity and the success condition of Coppersmith methods. The toolbox is versatile and can be used for many different applications, including multivariate polynomial systems with arbitrarily many unknowns (of possibly different sizes) and simultaneous modular equations over different moduli.

In order to illustrate the usefulness of this toolbox, we then revisit the analysis of previous cryptanalytic results from the literature on number-theoretic pseudorandom generators [2]. In particular, we precise the complexity analysis of the attacks described in [2] by giving generating functions and asymptotics for the dimension of the matrix involved in the attack. We provide a complete analysis of the success condition of the attacks described in [2,29]. The technique uses simple formal manipulation on the generating functions and are readily done using any computer algebra system. In particular, this shows that the toolbox is very generic and can be applied in many settings (and does not require any clever tricks).

Eventually, we provide new applications of the toolbox to RSA key generation and encryption paddings from weak pseudorandom generator. We improve Fouque *et al.* time/memory tradeoff attack and we propose a (heuristic) polynomial-time factorization attack when the RSA prime factors are constructed by concatenating the outputs of a linear congruential generator. Our attack applies when the primes factors are concatenation of three (or more) consecutive outputs of the generator, i.e., when the seed is at most $N^{1/6}$ (for which the time/memory tradeoff attack has the prohibitive complexity $O(N^{1/12})$). The attack is theoretical since it makes use of a matrix of large dimension. Following their suggestion, we also apply our toolbox to the setting of the randomness generation used in padding functions for encryption. To illustrate our technique, we consider RSA Encryption with padding as described in PKCS#1 v1.5; it has been known to be insecure since Bleichenbachers chosen ciphertext attack [7] but, unfortunately, this padding is still in used about everywhere (e.g., TLS, XML encryption standard, hardware token, ...). We consider several scenario, namely linear congruential generator (LCG), truncated LCG, and LCG used in n consecutive ciphertext. We apply our toolbox to all of them and for an RSA modulus N with a public exponent e and a LCG with modulus M, our attacks are polynomial-time in $\log(N)$ for the following (asymptotic) M's:

Key generation with LCG	PKCS#1 v1.5		
	LCG	Truncated LCG	LCG and multiple ciphertexts
$M \leqslant N^{1/6}$	$M < N^{1/e}$	$M < N^{1/e}$	$M < N^{n/e}$

2 Coppersmith Methods

In this section, we give a short description of Coppersmith method for solving a multivariate modular polynomial system of equations over multiple moduli. We refer the reader to [22,30] for details and proofs.

Problem Definition. Let $f_1(y_1, \ldots, y_n), \ldots, f_s(y_1, \ldots, y_n)$ be irreducible multivariate polynomials defined over $\mathbb{Z}$, having a root $(x_1, \ldots, x_n)$ modulo respective known integers $N_1, \ldots, N_n$, that is $f_i(x_1, \ldots, x_n) \equiv 0 \mod N_i$. This root is *small* in the sense that each of its components is bounded by a known value, namely $|x_1| < X_1, \ldots, |x_n| < X_n$. We need to bound the sizes of X_i (for $i \in \{1, \ldots, n\}$) allowing to recover the desired root in polynomial time.

Polynomials Collection. In a first step, for each modulus N_i, one generates a collection of polynomials $\{\tilde{f}_{i,1}, \ldots, \tilde{f}_{i,r(i)}\}$ having $(x_1, \ldots, x_n)$ as a root modulo N_i. Usually, multiples and powers of the original polynomial f_i are chosen, namely $\tilde{f}_{i,j} = y_1^{k_{i,j,1}} \cdots y_n^{k_{i,j,n}} f_i^{k_{i,j,\ell}}$ for some integers $k_{i,j,1}, \ldots, k_{i,j,n}, k_{i,j,\ell}$. By construction, such polynomials satisfy the relation $\tilde{f}_{i,j}(x_1, \ldots, x_n) \equiv 0 \mod N_i^{k_{i,j,\ell}}$, i.e., there exists an integer $c_{i,j,k}$ such that $\tilde{f}_{i,j,k}(x_1, \ldots, x_n) = c_{i,j,k} N_i^{k_{i,j,\ell}}$. If some moduli N_i are equals, one can also consider multiples and powers of products of the corresponding original polynomials f_i.

From now, we denote for each $i \in \{1, \ldots, s\}$, the polynomials $\{\tilde{f}_{i,1}, \ldots, \tilde{f}_{i,r(i)}\}$ constructed as above. Considering the union of such sets if some moduli N_i are equals, we can assume without loss of generality that the moduli N_i are *pairwise distinct* and even *pairwise coprime*. Let us denote as $\mathscr{P}$ the set of all the polynomials and $\mathscr{M}$ the set of monomials appearing in the collection $\mathscr{P}$. In the paper, we use the following essential condition for the method to work: for each $i \in \{1, \ldots, s\}$, the polynomials $\{\tilde{f}_{i,1}, \ldots, \tilde{f}_{i,r(i)}\}$ are *linearly independent*.

Matrix Construction. The problem of finding small modular roots of these polynomials can now be reformulated in a vectorial way. Indeed, each polynomial from our chosen collection can be expressed as a vector over $\mathbb{Z}^t$ by extracting its coefficients and putting them into a vector with respect to a chosen order on $\mathscr{M}$. We hence construct a matrix $\mathfrak{M}$ as follows and we define as $\mathcal{L}$ the lattice generated by its rows:

$$\mathfrak{M} = \begin{array}{cccc|ccc|c}
 & & & & \tilde{f}_{1,1} & \cdots & \tilde{f}_{s,r(s)} & \\
 & & & & \downarrow & \cdots & \downarrow & \\
1 & & & & & & & 1\\
 & X_1^{-1} & & & & \bigstar & & y_1\\
 & & \ddots & & & & & \vdots\\
 & & & X_1^{-a_1}\dots X_n^{-a_n} & & & & y_1^{a_1}\dots y_n^{a_n}\\
\hline
 & & 0 & & N_1^{k_{1,1,\ell}} & & & \\
 & & & & & \ddots & & \\
 & & & & & & N_s^{k_{s,r(s),\ell}} &
\end{array}$$

On that figure, every row of the upper part is related to one monomial of $\mathscr{M}$ (we assume in the figure that $\mathscr{M}$ contains 1, y_1, and $y_1^{a_1}\dots y_n^{a_n}$ among other monomials). The left-hand side contains the bounds on these monomials (e.g., the coefficient $X_1^{-1}X_2^{-2}$ is put in the row related to the monomial $y_1y_2^2$). The right-hand side is formed by all vectors coming from the union of the collections $\{\tilde{f}_{i,1},\dots,\tilde{f}_{i,r(i)}\}$.

A Short Vector in a Sublattice. Let us now consider the row vector

$$r_0 = (1, x_1, \dots, x_1^{a_1}\dots x_n^{a_n}, -c_1, \dots, -c_r).$$

By multiplying this vector by the matrix $\mathfrak{M}$, one obtains:

$$s_0 = \left(1, \left(\frac{x_1}{X_1}\right), \dots, \left(\frac{x_1}{X_1}\right)^{a_1} \cdots \left(\frac{x_n}{X_n}\right)^{a_n}, 0, \dots, 0\right).$$

By construction, this vector which, *in some sense*, contains the root we are searching for, belongs to $\mathcal{L}$ and its norm is very small. Thus, the recovery of a small vector in $\mathcal{L}$, will likely lead to the recovery of the desired root $(x_1,\dots,x_n)$. To this end, we first restrict ourselves in a more appropriated subspace. Indeed, noticing that the last coefficients of s_0 are all null, we know that this vector belongs to a sublattice $\mathcal{L}'$ whose last coordinates are composed by zero coefficients. By doing elementary operations on the rows of $\mathfrak{M}$, one can easily construct that sublattice and prove that its determinant is the same as the one of $\mathcal{L}$.

Method Conclusion. From that point, one computes an LLL-reduction on the lattice $\mathcal{L}'$ and computes the Gram-Schmidt's orthogonalized basis $(b_1^\star,\dots,b_t^\star)$ of the LLL output basis $(b_1,\dots,b_t)$. Since s_0 belongs to $\mathcal{L}'$, this vector can be expressed as a linear combination of the $b_i^\star$'s. Consequently, if its norm is smaller than those of $b_t^\star$, then s_0 is orthogonal to $b_t^\star$. Extracting the coefficients appearing in $b_t^\star$, one can construct a polynomial p_1 defined over $\mathbb{Z}$ such that $p_1(x_1, \dots, x_n) = 0$. Repeating the same process with the vectors $b_{t-1}^\star, \dots, b_{t-n+1}^\star$ leads to the system $\{p_1(x_1, \dots, x_n) = 0, \dots, p_n(x_1, \dots, x_n) = 0\}$. Under the (heuristic) assumption that all created polynomials define an algebraic variety of dimension 0, the previous system can be solved (e.g., using elimination techniques such as Groebner basis) and the desired root recovered in polynomial time.

The conditions on the bounds X_i that make this method work are given by the following (simplified) inequation (see [30] for details):

$$\prod_{y_1^{k_1} \dots y_n^{k_n} \in \mathscr{M}} X_1^{k_1} \cdots X_n^{k_n} < \prod_i N_i^{\sum_{i=1}^n \sum_{j=1}^{r(i)} k_{i,j,\ell}} . \tag{1}$$

For such techniques, the most complicated part is the choice of the collection of polynomials, what could be a really intricate task when working with multiple polynomials.

3 Analytic Combinatorics

We now recall the analytic combinatorics results that we need in the remaining of this paper. We deliberately omit some of the formalism in order to simplify the techniques used. See [15] for more details. In the following, we denote by $|\mathcal{A}|$ the cardinal of a set $\mathcal{A}$.

3.1 Introduction

As explained in the former section, Coppersmith's method requires polynomials which are usually constructed as $f_\mathbf{k} = y_1^{k_1} \dots y_n^{k_n} f^{k_\ell}$ (with f being a polynomial of degree e in the variables $y_1, \dots, y_n$). In the following, we thus consider a set of polynomials looking like[2]

$$\begin{aligned}\mathscr{P} = \{f_\mathbf{k} = y_1^{k_1} \dots y_n^{k_n} f^{k_\ell} \mod N^{k_\ell} \mid 1 \leqslant k_\ell < t \\ \text{and } \deg(f_\mathbf{k}) = k_1 + \dots + k_n + k_\ell e < te\},\end{aligned}$$

where the notation $\mod N^{k_\ell}$ is only here to recall that the considered solution verifies $f_\mathbf{k} \equiv 0 \mod N^{k_\ell}$ (to make things clearer). We suppose that f is not just a monomial (i.e., is the sum of at least two distinct monomials) and therefore each $\mathbf{k}$ corresponds to a distinct polynomial $f_\mathbf{k}$.

The set of monomials appearing in the collection $\mathscr{P}$ will usually look like

$$\mathscr{M} = \{y_\mathbf{k} = y_1^{k_1} \dots y_n^{k_n} \mid 0 \leqslant \deg(y_\mathbf{k}) = k_1 + \dots + k_n < te\}.$$

[2] We only use one polynomial f and one modulus N for the sake of simplicity. Furthermore, this exact set $\mathscr{P}$ could actually not appear in the Coppersmith methods, as the polynomials are not linearly independent. However, it is easier to explain analytic combinatorics tools on this set $\mathscr{P}$. We show later, in Sect. 4 and throughout this paper, how to adapt these tools to useful variants of this set.

By construction, since $(x_1, \ldots, x_n)$ is a modular root of the polynomials $f_{\mathbf{k}}$, there exists an integer $c_{\mathbf{k}}$ such that $f_{\mathbf{k}}(x_1, \ldots, x_n) = c_{\mathbf{k}} N^{k_\ell}$ (see Sect. 2). Furthermore, this root is *small* in the sense that each of its components is bounded by a known value, namely $|x_1| < X_1, \ldots, |x_n| < X_n$. These considerations imply that for the final condition in Coppersmith's method (see Eq. (1)), one needs to compute the values

$$\psi = \sum_{f_{\mathbf{k}} \in \mathscr{P}} k_\ell \quad \text{and} \quad \forall i \in \{1, \ldots, n\}, \quad \alpha_i = \sum_{y_{\mathbf{k}} \in \mathscr{M}} k_i .$$

These values correspond to the exponent of N and X_i (for $i \in \{1, \ldots, n\}$) in Eq. (1) respectively.

For the sake of readability for the reader unfamiliar with analytic combinatorics, we first show how to compute the number of polynomials in $\mathscr{P}$ or $\mathscr{M}$ of a certain degree and then how to compute these sums ψ and α_i but only for polynomials in $\mathscr{P}$ or $\mathscr{M}$ of a certain degree. These computations are of no direct use for Coppersmith's method but are a warm-up for the really interesting computation, namely these sums ψ and a_i for polynomials in $\mathscr{P}$ or $\mathscr{M}$ up to a certain degree.

3.2 Combinatorial Classes, Sizes, and Parameters

A combinatorial class is a finite or countable set on which a size function is defined, satisfying the following conditions: (i) the size of an element is a non-negative integer and (ii) the number of elements of any given size is finite. Polynomials of a "certain" form and up to a "certain" degree can be considered as a combinatorial class, using a size function usually related to the degree of the polynomial.

In the following, we can consider the set $\mathscr{P}$ as a combinatorial class, with the size function $S_{\mathscr{P}}$ defined as $S_{\mathscr{P}}(f_{\mathbf{k}}) = \deg(f_{\mathbf{k}}) = k_1 + \ldots + k_n + k_\ell e$. In order to compute the sum of the k_ℓ as explained in Sect. 3.1, we define another function $\chi_{\mathscr{P}}$, called a *parameter* function, such that $\chi_{\mathscr{P}}(f_{\mathbf{k}}) = k_\ell$. This function will enable us, instead of counting "1" for each polynomial, to count "k_ℓ" for each polynomial, which is exactly what we need (see Sect. 3.4 for the details).

As for the monomials, we will also consider the set $\mathscr{M}$ as a combinatorial class, with the size function $S_{\mathscr{M}}$ defined as $S_{\mathscr{M}}(y_{\mathbf{k}}) = k_1 + \ldots + k_n$. In the case the bounds on the variables are equal ($X_1 = \ldots = X_n = X$), the parameter function corresponding to the exponent α_1 of X_1 in the final condition in Coppersmith's method will be set as $\chi_{\mathscr{M}}(y_{\mathbf{k}}) = k_1 + \ldots + k_n$. Otherwise, one will be able to define other parameter functions in case the bounds are not equal (see again Sect. 3.4).

3.3 Counting the Elements: Generating Functions

The counting sequence of a combinatorial class $\mathcal{A}$ with size function S is the sequence of integers $(A_p)_{p \geqslant 0}$ where $A_p = |\{a \in \mathcal{A}_p \mid S(a) = p\}|$ is the number of objects in class $\mathcal{A}$ that have size p. For instance, if we consider the set $\mathscr{M}$ defined in Sect. 3.1, we have the equality $M_1 = n$ since there are n monomials in n variables of degree 1.

Definition 1. *The ordinary generating function (OGF) of a combinatorial class $\mathcal{A}$ is the generating function of the numbers A_p, for $p \geqslant 0$, i.e., the formal[3] power series $A(z) = \sum_{p=0}^{+\infty} A_p z^p$.*

For instance, if we consider the set $\mathscr{M}^{(1)} = \{y_1^{k_1} \mid 1 \leqslant k_1 < t\}$ of the monomials with one variable, then one gets $M_p^{(1)} = 1$ for all $p \in \mathbb{N}$, implying that $M^{(1)}(z) = \sum_{p=0}^{+\infty} z^p = \frac{1}{1-z}$.

In the former example, we constructed the OGF $A(z)$ from the sequence of numbers A_p of objects that have size p. Of course, what we are really interested in is to do it the other way around. We now describe an easy way to construct the OGF, and we will deduce from this function and classical analytic tools the value of A_p for every integer p. We assume the existence of an "atomic" class, comprising a single element of size 1, here a variable, usually denoted as $\mathcal{Z}$. We also need a "neutral" class, comprising a single element of size 0, here 1, usually denoted as ε. Their OGF are $Z(z) = z$ and $E(z) = 1$. We show in Table 1 the possible admissible constructions that we will need here, as well as the corresponding generating functions.

One then recovers the formula $M^{(1)}(z) = \frac{1}{1-z}$ from $Z(z) = z$ and the construction $\textsc{Seq}(\mathcal{Z})$ to describe $\mathscr{M}^{(1)}$. Similarly, if we now consider the set $\mathscr{M}^{(2)} = \{y_{\mathbf{k}} = y_1^{k_1} y_2^{k_2} \mid 0 \leqslant k_1 + k_2 < t\}$ of the monomials with two variables, with the size function $S(y_{\mathbf{k}}) = k_1 + k_2$, then one gets $M^{(2)}(z) = M^{(1)}(z) \cdot M^{(1)}(z) = \frac{1}{(1-z)^2}$ from $\mathscr{M}^{(2)} = \mathscr{M}^{(1)} \times \mathscr{M}^{(1)}$. Finally, since $\frac{1}{(1-z)^2} = \sum_{p=1}^{+\infty} p z^{p-1}$, one gets, for all $p \geqslant 1$, $(M_2)_p = p + 1$, which is exactly the number of monomials with two variables of size p.

When the class contains elements of different sizes (such as variables of degree 1 and polynomials of degree e), the variables are represented by the atomic

Table 1. Combinatorics constructions and their OGF

	Construction	OGF
Atomic class	$\mathcal{Z}$	$Z(z) = z$
Neutral class	ε	$E(z) = 1$
Disjoint union	$\mathcal{A} = \mathcal{B} + \mathcal{C}$ (when $\mathcal{B} \cap \mathcal{C} = \emptyset$)	$A(z) = B(z) + C(z)$
Complement	$\mathcal{A} = \mathcal{B} \setminus \mathcal{C}$ (when $\mathcal{C} \subseteq \mathcal{B}$)	$A(z) = B(z) - C(z)$
Cartesian product	$\mathcal{A} = \mathcal{B} \times \mathcal{C}$	$A(z) = B(z) \cdot C(z)$
Cartesian exponentiation	$\mathcal{A} = \mathcal{B}^k = \mathcal{B} \times \dots \times \mathcal{B}$	$A(z) = B(z)^k$
Sequence	$\mathcal{A} = \textsc{Seq}(\mathcal{B}) = \varepsilon + \mathcal{B} + \mathcal{B}^2 + \dots$	$A(z) = \frac{1}{1-B(z)}$

[3] We stress that it is a "formal" series, i.e., with no need to worry about the convergence.

element $\mathcal{Z}$ and the polynomials by the element $\mathcal{Z}^e$, in order to take into account the degree of the polynomial f. If we consider for instance the set $\mathscr{P}^{(1,2)} = \{f_{\mathbf{k}} = y_1^{k_1} f^{k_\ell} \mid 1 \leqslant k_\ell < t \text{ and } \deg(f_{\mathbf{k}}) = k_1 + 2k_\ell < 2t\}$, with f a polynomial of degree 2, this set is isomorphic to $\textsc{Seq}(\mathcal{Z}) \times \mathcal{Z}^2\textsc{Seq}(\mathcal{Z}^2)$, since $\deg(f) = 2$. This leads to an OGF equals to

$$\frac{1}{1-z}\frac{z^2}{1-z^2} = \sum_{q=0}^{+\infty} qz^q \sum_{r=1}^{+\infty} rz^{2r} = \sum_{p=0}^{+\infty} \sum_{r=1}^{\lfloor p/2 \rfloor} (p-2r)rz^p\,,$$

which gives $P_p^{(1,2)} = \sum_{r=1}^{\lfloor p/2 \rfloor}(p-2r)r$, which is exactly the number of polynomials of degree p contained in the class.

3.4 Counting the Parameters of the Elements: Bivariate Generating Functions

As seen in the former section, when one considers a combinatorial class $\mathcal{A}$ of polynomials and computes the corresponding OGF $A(z)$, classical analytic tools enable to recover A_p as the coefficient of z^p in the OGF. As explained in the introduction of this section, however, Coppersmith's method requires a computation a bit more tricky, which involves an additional parameter. For the sake of simplicity, we describe this technique on an example.

For instance, consider our monomial set example $\mathscr{M}^{(2)}$, but now assume that $X_1 \neq X_2$. Our goal is to compute $\sum k_1$, where the sum is taken over all the monomials in $\mathscr{M}^{(2)}$ of size p. We set a parameter function[4] $\chi(y_{\mathbf{k}}) = k_1$ and we do not compute $M_p^{(2)}$ (for $p \geqslant 1$) anymore, but rather

$$\chi_p(\mathscr{M}^{(2)}) = \sum_{y_{\mathbf{k}} \in \mathscr{M}^{(2)} | S(y_{\mathbf{k}})=p} \chi(y_{\mathbf{k}}) = \sum_{y_{\mathbf{k}} \in \mathscr{M}^{(2)} | S(y_{\mathbf{k}})=p} k_1$$

where, informally speaking, instead of counting for 1, every monomial counts for the value of its parameter (here the degree k_1 in y_1).

The value $\chi_p(\mathscr{M}^{(2)})$ cannot be obtained by the construction of $\mathscr{M}^{(2)}$ as $\textsc{Seq}(\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z})$ that we used in the former section, since the two atomic elements $\mathcal{Z}$ do not play the same role (the first one is linked with the parameter, whereas the second one is not). The classical solution is simply to "mark" the atomic element useful for the parameter, with a new variable u: With this new parameter function, $\mathscr{M}^{(2)}$ is seen as $\textsc{Seq}(u\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z})$, defining the bivariate ordinary generating function (BGF)[5] $M_2(z,u) = \frac{1}{1-uz}\frac{1}{1-z}$. We remark that when we set $u = 1$, we get the original non-parameterized OGF. Informally speaking, the BGF of a combinatorial class $\mathcal{A}$ with respect to a size function S

[4] Note that it is possible to count the exponents of both X_1 and X_2 at once using two parameters, but it is usually easier to count them separately, which often boils down to the same computation. See concrete examples in Sect. 4.

[5] In complex cases, the marker u can be put to some exponent k, for instance if the parameter considered has a value equal to k for the atomic element.

and a parameter function χ is obtained from the corresponding OGF by replacing each z by $u^k z$ where k is the value of the parameter taken on the atomic element $\mathcal{Z}$. We then obtain $\chi_p(\mathcal{A})$ via the following result:

Theorem 2. *Assume $\mathcal{A}$ is a combinatorial class with size function S and parameter function χ, and assume $A(z,u)$ is the bivariate ordinary generating funtion for $\mathcal{A}$ corresponding to this parameter (constructed as explained above). Then, if we define*

$$\chi_p(\mathcal{A}) = \sum_{a \in \mathcal{A} | S(a)=p} \chi(a)$$

the ordinary generating function of the sequence $(\chi_p(\mathcal{A}))_{p \geqslant 0}$ is equal to the value $(\partial A(z,u)/\partial u)_{u=1}$, meaning that we have the equality

$$\left.\frac{\partial A(z,u)}{\partial u}\right|_{u=1} = \sum_{p=0}^{+\infty} \chi_p(\mathcal{A}) z^p \overset{def}{=} \chi(A)(z).$$

Coming back to our example, one then gets

$$\chi(M^{(2)})(z) = \sum_{p=0}^{+\infty} \chi_p(\mathscr{M}^{(2)}) z^p = \left.\frac{\partial M^{(2)}(z,u)}{\partial u}\right|_{u=1} = \frac{z}{(1-z)^3} = \sum_{p=1}^{+\infty} \frac{p(p-1)}{2} z^{p-1}.$$

meaning that $\chi_p(\mathscr{M}^{(2)}) = p(p+1)/2$ (remind that it is an equality on formal series). Finally, the sum of the degrees k_1 of the elements of size p is $p(p+1)/2$, which can be checked by enumerating them: $y_2^p, y_1 y_2^{p-1}, y_1^2 y_2^{p-2}, \ldots, y_1^{p-1} y_2, y_1^p$. It is easy to see that the result is exactly the same for X_2, without any additional computation, by symmetry.

3.5 Counting the Parameters of the Elements up to a Certain Size

We described in the former section a technique to compute the sum of the (partial) degrees of elements of size p, but how about computing the same sum for elements of size *up to p*? Using the notations of the former section, we want to compute

$$\chi_{\leqslant p}(\mathcal{A}) = \sum_{a \in \mathcal{A} | S(a) \leqslant p} \chi_p(a).$$

The naive way is to sum up the values $\chi_i(\mathcal{A})$ for all i between 0 and p:

$$\chi_{\leqslant p}(\mathcal{A}) = \sum_{i=0}^{p} \sum_{a \in \mathcal{A} | S(a)=i} \chi_i(a)\,,$$

but an easier way to do so is to artificially force all elements a of size less than or equal to p to be of size exactly p by adding enough times a dummy element y_0 such that $\chi(y_0) = 0$.

In our context of polynomials, the aim of the dummy variable y_0 is to homogenize the polynomial. If we consider again the set $\mathscr{M}^{(2)}$ of monomials of two variables y_1 and y_2, with size function equal to $S(y_{\mathbf{k}}) = k_1 + k_2$ and parameter function equal to $\chi(y_{\mathbf{k}}) = k_1$, and if we are interested in the sum of the degrees k_1 of the elements in this set of size *up to p*, we now describe this set as $\textsc{Seq}(u\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z})$, the last part being the class of monomials in the unique variable y_0. This variable is not marked, since its degree is not counted. One obtains the new bivariate generating function $M^{(2)}(z,u) = \frac{1}{1-uz}\frac{1}{(1-z)^2}$ and

$$\chi_{\leqslant}(M^{(2)})(z) = \sum_{p=0}^{+\infty} \chi_{\leqslant p}(\mathscr{M}^{(2)}) z^p = \left.\frac{\partial M^{(2)}(z,u)}{\partial u}\right|_{u=1} = \frac{z}{(1-z)^4}$$
$$= \sum_{p=2}^{+\infty} \frac{p(p-1)(p-2)}{6} z^{p-2},$$

meaning that $\chi_{\leqslant p}(\mathscr{M}^{(2)}) = p(p+1)(p+2)/6$ (remind that it is an equality on formal series). Finally, the sum of the degrees k_1 of the elements of size up to p (i.e., the exponent of X_1 in Coppersmith's method) is $p(p+1)(p+2)/2$, which can be checked by the computation

$$\sum_{i=0}^{p} \frac{i(i+1)}{2} = \frac{p(p+1)(p+2)}{6}.$$

Again, it is easy to see that the result is exactly the same for X_2, without any additional computation.

3.6 Asymptotic Values: Transfer Theorem

Finding the OGF or BGF of the combinatorial classes is usually an easy task, but finding the exact value of the coefficients can be quite painful. Coppersmith's method is usually used in an asymptotic way. Singularity analysis enables us to find the asymptotic value of the coefficients in an simple way, using the technique described in [15, Corollary VI.1(sim-transfer), p. 392]. Adapted to our context, their transfer theorem can be stated as follows:

Theorem 3 (Transfer Theorem). *Assume $\mathcal{A}$ is a combinatorial class with an ordinary generating function F regular enough such that there exists a value c verifying*

$$F(z) \underset{z\to 1}{\sim} \frac{c}{(1-z)^{\alpha}}$$

for a non-negative integer α. Then the asymptotic value of the coefficient F_n is

$$F_n \underset{n\to\infty}{\sim} \frac{cn^{\alpha-1}}{(\alpha-1)!}.$$

4 A Toolbox for the Cryptanalyst

We now describe how to use the generic tools recalled in the former section to count the exponents of the bounds $X_1, \ldots, X_n$ and of the modulo N (as in the previous section, we consider the simplified case with only one modulus N) on the monomials and polynomials appearing in Coppersmith's method (see Sect. 2). For the sake of simplicity, we describe the technique on several examples, supposedly complex enough to be easily combined and adapted to most of the useful cases encountered in practice.

4.1 Counting the Bounds for the Monomials (Useful Examples)

First Example. In this example, we consider

$$\mathscr{M} = \{{y_1}^{i_1} \cdots {y_m}^{m} \cdot {y_{m+1}}^{m+1} \cdots {y_n}^{i_n} \mid 1 \leqslant i_1 + \ldots + i_n < t\}$$

with the bounds $|y_i| < X$ for $1 \leqslant i \leqslant m$ et $|y_i| < Y$ for $m < i \leqslant n$. In order to obtain the exponent for the bound X, we consider the size function $S({y_1}^{i_1} \ldots {y_n}^{i_n}) = i_1 + \ldots + i_n$ and the parameter function $\chi_X({y_1}^{i_1} \ldots {y_n}^{i_n}) = i_1 + \ldots + i_m$.

We describe $\mathscr{M}$ as $\prod_{i=1}^{m} \textsc{Seq}(u\mathcal{Z}) \times \prod_{i=m+1}^{n} \textsc{Seq}(\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z}) \setminus \varepsilon$ (the last $\textsc{Seq}(\mathcal{Z})$ being for the dummy value y_0), which leads to the OGF

$$F(z, u) = \left(\frac{1}{1 - uz}\right)^{m} \left(\frac{1}{1 - z}\right)^{n-m+1} - 1.$$

The next step is to compute the partial derivative in u at $u = 1$:

$$\left.\frac{\partial F(z, u)}{\partial u}\right|_{u=1} = \left.\frac{mz}{(1 - uz)^{m+1}} \left(\frac{1}{1 - z}\right)^{n-m+1}\right|_{u=1} = \frac{mz}{(1 - z)^{n+2}}$$

and take the equivalent value when $z \to 1$:

$$\left.\frac{\partial F(z, u)}{\partial u}\right|_{u=1} \underset{z \to 1}{\sim} \frac{m}{(1 - z)^{n+2}},$$

which finally leads, using Theorem 3, to $\chi_{X,<t}(\mathscr{M}) \sim \frac{m(t-1)^{n+1}}{(n+1)!} \sim \frac{mt^{n+1}}{(n+1)!}$.

Finally, it is easy to see that if one denotes $\chi_Y({y_1}^{i_1} \ldots {y_n}^{i_n}) = i_{m+1} + \ldots + i_n$, one gets $\chi_{Y,<t}(\mathscr{M}) \sim \frac{(n-m)t^{n+1}}{(n+1)!}$. This set of monomials used in Coppersmith's method thus leads to the bound $X^{\frac{mt^{n+1}}{(n+1)!}} Y^{\frac{(n-m)t^{n+1}}{(n+1)!}}$. In the particularly useful case where $X = Y$, the bound becomes $X^{\frac{nt^{n+1}}{(n+1)!}}$ for all the monomials in n variables of degree up to t.

Second Example. In this example, we consider

$$\mathscr{M} = \{{y_1}^{i_1} \dots {y_n}^{i_n} \mid (i_1 = 0 \text{ or } i_2 = 0) \\ \text{and } 1 \leqslant i_3 \leqslant e \text{ and } 1 \leqslant i_1 + \ldots + i_n < t\}$$

with the bounds $|y_i| < X$ for $1 \leqslant i \leqslant n$. We use the size function $S({y_1}^{i_1} \dots {y_n}^{i_n}) = i_1 + \ldots + i_n$ and the parameter function $\chi({y_1}^{i_1} \dots {y_n}^{i_n}) = i_1 + \ldots + i_n$ (since the bound X is the same for all variables).

The first step is to split $\mathscr{M}$ into disjoint subsets. In our case, the three disjoint subsets correspond to $i_1 = i_2 = 0$, ($i_1 = 0$ and $i_2 \neq 0$) and ($i_1 \neq 0$ and $i_2 = 0$). Taking into account the dummy value y_0, we describe them as

$$(\mathcal{Z} + \ldots + \mathcal{Z}^e) \times \prod_{i=1}^{n-3} \textsc{Seq}(u\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z})$$

for the first one and

$$(u\mathcal{Z}) \times \textsc{Seq}(u\mathcal{Z}) \times (\mathcal{Z} + \ldots + \mathcal{Z}^e) \times \prod_{i=1}^{n-3} \textsc{Seq}(u\mathcal{Z}) \times \textsc{Seq}(\mathcal{Z})$$

for the two others (since the presence of y_1 or y_2 is mandatory). This leads to the OGF

$$\begin{aligned} F(z,u) &= \left(1 + \frac{uz}{1-uz} + \frac{uz}{1-uz}\right)(z + \ldots + z^e)\left(\frac{1}{1-uz}\right)^{n-3} \frac{1}{1-z} \\ &= \frac{1+uz}{(1-uz)^{n-2}} \frac{z + \ldots + z^e}{1-z}, \end{aligned}$$

which gives, after computations,

$$\left.\frac{\partial F(z,u)}{\partial u}\right|_{u=1} = \left.\frac{z((n-3)uz + n - 1)}{(1-uz)^{n-1}} \frac{z + \ldots + z^e}{1-z}\right|_{u=1} \underset{z\to1}{\sim} \frac{(2n-4)e}{(1-z)^n},$$

which finally leads to $\chi_{<t}(\mathscr{M}) \sim \frac{(2n-4)e(t-1)^{n-1}}{(n-1)!} \sim \frac{(2n-4)et^{n-1}}{(n-1)!}$, using Theorem 3.

4.2 Counting the Bounds for the Polynomials (Example)

We now consider the set

$$\mathscr{P} = \{f_{\mathbf{k}} = y_1^{k_1} \dots y_n^{k_n} f^{k_\ell} \mod N^{k_\ell} \mid 1 \leqslant k_\ell < t \\ \text{and } \deg(f_{\mathbf{k}}) = k_1 + \ldots + k_n + k_\ell e < te\}$$

with the bounds $X_1 = \ldots = X_n = X$ for the variables. In order to obtain the exponent for the modulus N, we consider the size function $S({y_1}^{k_1} \dots {y_n}^{k_n} f^{k_\ell}) = k_1 + \ldots + k_n + k_\ell$ and the parameter function $\chi_{N}({y_1}^{k_1} \dots {y_n}^{k_n} f^{k_\ell}) = k_\ell$.

For the sake of simplicity, we can consider $0 \leqslant k_\ell < t$ since the parameter function is equal to 0 on the elements $f_{\mathbf{k}}$ such that $k_\ell = 0$. We describe $\mathscr{P}$ as

$\prod_{i=1}^{n} \textsc{Seq}(\mathscr{Z}) \times \textsc{Seq}(u\mathscr{Z}^e) \times \textsc{Seq}(\mathscr{Z})$ (the last one being for the dummy value y_0), since only f needs a marker and its degree is e. This leads to the OGF

$$F(z,u) = \left(\frac{1}{1-z}\right)^{n+1} \frac{1}{1-uz^e}.$$

The next step is to compute the partial derivative in u at $u = 1$:

$$\left.\frac{\partial F(z,u)}{\partial u}\right|_{u=1} = \left.\frac{z^e}{(1-uz^e)^2}\left(\frac{1}{1-z}\right)^{n+1}\right|_{u=1} = \frac{z^e}{(1-z^e)^2}\left(\frac{1}{1-z}\right)^{n+1}$$

and take the equivalent value when $z \to 1$, using the formula $1 - z^e \sim e(1-z)$:

$$\left.\frac{\partial F(z,u)}{\partial u}\right|_{u=1} \underset{z \to 1}{\sim} \frac{1}{e^2(1-z)^{n+3}}\,,$$

which finally leads, using Theorem 3, to $\chi_{N,<te}(\mathscr{P}) \sim \frac{(te)^{n+2}}{e^2(n+2)!}$.

5 Number-Theoretic Pseudorandom Generators (Following [2])

As mentioned in the introduction, number-theoretic pseudorandom generators work by iterating an algebraic map F over a residue ring $\mathbb{Z}_N$ on a secret random initial seed value $v_0 \in \mathbb{Z}_N$ to compute the intermediate state values $v_{i+1} = F(v_i) \bmod N$ for $i \in \mathbb{N}$ and outputting (some consecutive bits of) the state value v_i at each iteration. In [2], Bauer *et al.* showed that such a pseudorandom generator defined by a known iteration polynomial function F can be broken under the condition that sufficiently many bits are output by the generator at each iteration (with respect to the degree of F).

Let $F(X)$ be a polynomial of degree d in $\mathbb{Z}_N[X]$ and let v_0 be a secret seed. As in [2], we assume that the generator outputs the k most significant bits of v_i at each iteration (with $k \in \{1, \dots, n\}$ where n is the bit-length of N). More precisely, if $v_i = 2^{n-k}w_i + x_i$, with $0 \leqslant x_i < 2^{n-k} = M = N^\delta$ which is unknown to the adversary and w_i is output by the generator. The adversary wants to recover x_i for some $i \in \mathbb{N}$ from consecutive values of the pseudorandom sequence (with M as large as possible). We have $v_{i+1} = F(v_i) \bmod N$ (for $i \in \mathbb{N}$) for a known polynomial F and $2^{m-k}w_{i+1} + x_{i+1} = F(2^{m-k}w_i + x_i) \bmod N$. We can therefore define explicitly a family of bivariate polynomials of degree d, $f_i(y_i, y_{i+1}) \in \mathbb{Z}_N[y_i, y_{i+1}]$, such that $f_i(x_i, x_{i+1}) = 0 \bmod N$, for $i \in \{0, \dots, n\}$ whose coefficients publicly depend on the approximations w_i, w_{i+1} and F's coefficients. The goal is to compute the (small) modular root $(x_0, x_1, \dots, x_n)$ of the polynomial system $\{f_0(y_0, y_1) = 0, \dots, f_n(y_n, y_{n+1}) = 0\}$ in polynomial time.

Description of the attack. In order to solve this system, Bauer *et al.* [2] applied Coppersmith method for multivariate modular polynomial system to the following collection of polynomials:

$$\mathscr{P} = \{{y_0}^j {f_0}^{i_0} \dots {f_n}^{i_n} \mid d(i_0 + di_1 + \dots + d^n i_n) + j \leqslant dm \wedge (i_0 + \dots + i_n > 0)\}$$

where $m \geq 1$ is a fixed integer. They showed that the set of monomials occurring in the collection is:

$$\{{y_0}^j {y_1}^{i_0} \dots {y_{n+1}}^{i_n} \mid d(i_0 + di_1 + \dots + d^n i_n) + j \leqslant dm\}.$$

To analyze their algorithm, Bauer *et al.* used a trick from [18] and only computes the quotient of the two quantities involved in Coppersmith success condition (1) (thanks to a fortunate simplification). In the following, we will use our toolbox to recompute (more) easily the bounds on these two quantities. We also obtain more precise estimates since our toolbox also permits to obtain the dimensions of the matrix used in Coppersmith method (and therefore the actual complexity of the attack).

Bound for the Polynomials. We consider the set $\mathscr{P}$ defined as

$$\{{y_0}^j {f_0}^{i_0} \dots {f_n}^{i_n} \mod N^{i_n} \mid d(i_0 + di_1 + \dots + d^n i_n) + j \leqslant dm \wedge i_0 + \dots + i_n > 0)\}$$

as a combinatorial class, with the size function $S_f({y_0}^j {f_0}^{i_0} \dots {f_n}^{i_n}) = d(i_0 + di_1 + \dots + d^n i_n) + j$ and the parameter function $\chi_f({y_0}^j {f_0}^{i_0} \dots {f_n}^{i_n}) = i_0 + \dots + i_n$. For the sake of simplicity, we can consider $i_0 + \dots + i_n \geqslant 0$ since the parameter function is equal to 0 on the elements such that $i_0 + \dots + i_n = 0$. We split the parameter functions into $(n+1)$ parts $\chi_{f,j}({y_0}^j {f_0}^{i_0} \dots {f_n}^{i_n}) = i_j$ (for $j \in \{0, \dots, n\}$), do the computation for each of them and sum the obtained asymptotic equivalents (and this can be done legitimately by computing the corresponding limits).

Let $j \in \{0, \dots, n\}$. Since the degree of each f_k is d^{k+1}, we consider $\mathscr{P}$ as

$$\underbrace{\textsc{Seq}(\mathcal{Z})}_{y_0} \times \prod_{\substack{k=0 \\ k \neq j}}^{n} \underbrace{\textsc{Seq}(\mathcal{Z}^{d^{k+1}})}_{f_k} \times \underbrace{\textsc{Seq}(u\mathcal{Z}^{d^{j+1}})}_{f_j} \times \underbrace{\textsc{Seq}(\mathcal{Z})}_{\text{dummy var.}} \setminus \underbrace{\textsc{Seq}(\mathcal{Z})}_{y_0} \times \underbrace{\textsc{Seq}(\mathcal{Z})}_{\text{dummy var.}}$$

which leads to the following generating function

$$F_j(u, z) = \frac{1}{1-z} \left(\prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{1}{1 - z^{d^{k+1}}} \right) \frac{1}{1 - uz^{d^{j+1}}} \frac{1}{1-z} - \frac{1}{1-z} \frac{1}{1-z}.$$

We take the partial derivative in u and then let $u = 1$:

$$\left. \frac{\partial F_j}{\partial u}(u, z) \right|_{u=1} = \left(\frac{1}{1-z} \right)^2 \times \left(\prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{1}{1 - z^{d^{k+1}}} \right) \times \frac{z^{d^{j+1}}}{(1 - z^{d^{j+1}})^2}.$$

We take the equivalent when $z \to 1$, using the formula $1 - z^n \sim n(1-z)$:

$$\left.\frac{\partial F_j}{\partial u}(u,z)\right|_{u=1} \underset{z\to 1}{\sim} \left(\frac{1}{1-z}\right)^2 \times \left(\prod_{\substack{k=0\\k\neq j}}^{n} \frac{1}{d^{k+1}(1-z)}\right) \times \frac{1}{(d^{j+1})^2(1-z)^2} .$$
$$\underset{z\to 1}{\sim} \frac{1}{(1-z)^{n+4}} \frac{1}{d^{(n+1)(n+2)/2} d^{j+1}}$$

Applying Theorem 3, one finally gets

$$\chi_{f,j,\leqslant dm}(\mathscr{P}) \sim \frac{1}{(n+3)!}(dm)^{n+3} \frac{1}{d^{(n+1)(n+2)/2} d^{j+1}} ,$$

which leads to

$$\chi_{f,\leqslant dm}(\mathscr{P}) \sim \left(\sum_{j=0}^{n} \frac{1}{d^{j+1}}\right) \frac{1}{(n+3)!}(dm)^{n+3} \frac{1}{d^{(n+1)(n+2)/2}} .$$

Bound for the Monomials. We consider the set $\mathscr{M}$ defined as

$$\{y_0{}^j y_1{}^{i_0} \dots y_{n+1}{}^{i_n} \mod M^{i_0+\dots+i_n} \mid d(i_0 + di_1 + \dots + d^n i_n) + j \leqslant dm\}$$

as a combinatorial class, with the size function $S_y(y_0{}^j y_1{}^{i_0} \dots y_{n+1}{}^{i_n}) = d(i_0 + di_1 + \dots + d^n i_n) + j$ and the parameter function $\chi_y(y_0{}^j f_0{}^{i_0} \dots f_n{}^{i_n}) = i_0 + \dots + i_n$. As before, we split the parameter functions into $(n+1)$ parts $\chi_{y,j}(y_0{}^j y_1{}^{i_0} \dots y_{n+1}{}^{i_n}) = i_j$ (for $j \in \{0, \dots, n\}$) and do the computation for each of them. As each y_k "counts for" d^k in the condition of the set, we consider $\mathscr{M}$ as

$$\underbrace{\textsc{Seq}(\mathcal{Z})}_{y_0} \times \prod_{\substack{k=1\\k\neq j}}^{n+1} \underbrace{\textsc{Seq}(\mathcal{Z}^{d^k})}_{y_k} \times \underbrace{\textsc{Seq}(u\mathcal{Z}^{d^j})}_{y_j} \times \underbrace{\textsc{Seq}(\mathcal{Z})}_{\text{dummy var.}} ,$$

which leads to the following generating function

$$G_j(u,z) = \frac{1}{1-z}\left(\prod_{\substack{k=1\\k\neq j}}^{n+1} \frac{1}{1-z^{d^k}}\right) \frac{1}{1-uz^{d^j}} \frac{1}{1-z} .$$

We take the partial derivative in u and then let $u = 1$:

$$\left.\frac{\partial G_j}{\partial u}(u,z)\right|_{u=1} = \left(\prod_{\substack{k=0\\k\neq j}}^{n+1} \frac{1}{1-z^{d^k}}\right) \times \left(\frac{1}{1-z}\right) \times \frac{z^{d^j}}{(1-z^{d^j})^2} .$$

We take the equivalent when $z \to 1$, using the formula $1 - z^n \sim n(1-z)$:

$$\left.\frac{\partial G_j}{\partial u}(u,z)\right|_{u=1} \underset{z\to 1}{\sim} \left(\prod_{\substack{k=0\\k\neq j}}^{n+1} \frac{1}{d^k(1-z)}\right) \times \left(\frac{1}{1-z}\right) \times \frac{1}{(d^j)^2(1-z)^2}$$
$$\underset{z\to 1}{\sim} \frac{1}{(1-z)^{n+4}} \frac{1}{d^{(n+1)(n+2)/2} d^j} .$$

Applying Theorem 3, one finally gets

$$\chi_{y,j,\leqslant dm}(\mathscr{M}) \sim \frac{1}{(n+3)!}(dm)^{n+3}\frac{1}{d^{(n+1)(n+2)/2}d^j},$$

which leads to

$$\chi_{y,\leqslant dm}(\mathscr{M}) \sim \left(\sum_{j=0}^{n+1}\frac{1}{d^j}\right)\frac{1}{(n+3)!}(dm)^{n+3}\frac{1}{d^{(n+1)(n+2)/2}}.$$

Condition. If we denote by $\mu = \chi_{f,\leqslant dm}(\mathscr{P})$ and $\xi = \chi_{y,\leqslant dm}(\mathscr{M})$, the condition for Coppersmith's method is $N^\mu > M^\xi$, i.e., $N^{\mu/\xi} > M$, where

$$\frac{\mu}{\xi} = \frac{\chi_{f,\leqslant dm}(\mathscr{P})}{\chi_{y,\leqslant dm}(\mathscr{M})} \sim \frac{\sum_{j=0}^{n}\frac{1}{d^{j+1}}}{\sum_{j=0}^{n+1}\frac{1}{d^j}} = \frac{\frac{1}{d}\frac{1-1/d^{n+1}}{1-1/d}}{\frac{1-1/d^{n+2}}{1-1/d}} \sim \frac{1}{d},$$

which leads to the expected bound $M < N^{1/d}$ that was given in [2], for which the algorithm (heuristically) outputs the the (small) modular root $(x_0, x_1, \dots, x_n)$ of the polynomial system $\{f_0(y_0, y_1) = 0, \dots, f_n(y_n, y_{n+1}) = 0\}$ in polynomial time.

Complexity. In order to compute the dimensions of the matrix used in Coppersmith methods, we have to compute the cardinality of the sets $\mathscr{P}$ and $\mathscr{M}$ (i.e., with the constant parameter functions $\chi_f = 1$ and $\chi_{y,j} = 1$). We obtain the generating functions

$$\frac{1}{1-z}\left(\prod_{k=0}^{n}\frac{1}{1-z^{d^{k+1}}}\right)\frac{1}{1-z} - \frac{1}{1-z}\frac{1}{1-z} \underset{z\to 1}{\sim} \frac{1}{(1-z)^{n+3}}\frac{1}{d^{(n+1)(n+2)/2}}$$

and

$$\frac{1}{1-z}\left(\prod_{k=1}^{n+1}\frac{1}{1-z^{d^k}}\right)\frac{1}{1-z} \underset{z\to 1}{\sim} \frac{1}{(1-z)^{n+3}}\frac{1}{d^{(n+1)(n+2)/2}}$$

for $\mathscr{P}$ and $\mathscr{M}$ (respectively). We thus obtain as above for the cardinality of both sets $\mathscr{P}$ and $\mathscr{M}$ (and therefore essentially for the dimensions of the matrix), the asymptotics

$$\frac{(dm)^{n+2}}{(n+2)!}\frac{1}{d^{(n+1)(n+2)/2}}.$$

Remark 4. A computer algebra program can compute the first coefficients of the formal series for μ and ξ and for the cardinality of the sets $\mathscr{P}$ and $\mathscr{M}$, for any given d and n. Therefore, given d, n, and $\log M/\log N$, it enables to compute the minimum value m such that the attack works (i.e., such that $\mu/\xi > \log M/\log N$, using the simplified condition, assuming the heuristic assumption holds) and then to compute the corresponding number of polynomials in $\mathscr{P}$ and of monomials in $\mathscr{M}$, which then yield the size of the matrix. For an example of such an analysis see end of Sect. 6.1.

6 New Applications

6.1 Key Generation from Weak Pseudorandomness

In [16], Fouque, Tibouchi and Zapalowicz analyzed the security of key generation algorithms when the prime factors of an RSA modulus are constructed by concatenating the outputs of a linear congruential generator. They proposed an (exponential-time) attack based on multipoint polynomial evaluation to recover the seed when such generators are used to generate one prime factor of an RSA modulus. In this section, we propose a new heuristic (polynomial-time) algorithm based on Coppersmith methods that allows to factor an RSA modulus when both its primes factors are constructed by concatenating the outputs of a linear congruential generator (with possible different seeds).

Let $M = 2^k$ be a power of 2 (for $k \in \mathbb{N} \setminus \{0\}$). For the ease of exposition, we consider a straightforward method to generate a prime number in which the key generation algorithm starts from a random seed modulo M, iterates the linear congruential generator and performs a primality test on the concatenation of the outputs (and in case of an invalid answer, repeat the process with another random seed until a prime is found). Let v_0 and w_0 be two random seeds for a linear congruential generator with public parameters a and b in $\mathbb{Z}_M$ that defines the pseudorandom sequences:

$$v_{i+1} = av_i + b \mod M \quad \text{and} \quad w_{i+1} = aw_i + b$$

for $i \in \mathbb{N}$. We assume that the adversary is given as input a (balanced) RSA modulus $N = p \cdot q$ where p and q are (kn)-bit primes where $p = v_0 + Mv_1 + \ldots + M^n v_n$ and $q = w_0 + Mw_1 + \ldots + M^n w_n$.

Description of the attack. The adversary is given as inputs the RSA modulus N and the generator parameters a and b and its goal is to factor N (or equivalently to recover one of the secret seed v_0 or w_0 used in the key generation algorithm). This can be done by solving the following multivariate system of polynomial equations over the moduli N and M with unknowns $v_0,\ldots,v_n,w_0,\ldots,w_n$:

$$\begin{cases} f = (v_0 + Mv_1 + \ldots + M^n v_n)(w_0 + Mw_1 + \ldots + M^n w_n) \equiv 0 \mod N \\ g_0 = v_1 - (av_0 + b) \equiv 0 \mod M \\ \quad \vdots \\ g_{n-1} = v_n - (av_{n-1} + b) \equiv 0 \mod M \\ h_0 = w_1 - (aw_0 + b) \equiv 0 \mod M \\ \quad \vdots \\ h_{n-1} = w_n - (aw_{n-1} + b) \equiv 0 \mod M. \end{cases}$$

In order to apply Coppersmith technique, the most complicated part is the choice of the collection of polynomials constructed from the polynomials that occur in this system. After several attempts, we choose to use the following polynomial family (parameterized by some integer $t \in \mathbb{N}$):

$$\begin{aligned}\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k} &= v_0^{i_0}\dots v_n^{i_n}\cdot w_0^{j_0}\dots w_n^{j_n}\cdot f^k \mod N^k\\ &\text{with } 1 \le k < t,\ (i_0 = 0 \text{ or } j_0 = 0)\\ &\text{and } \deg(\tilde{f}_{\dots}) = i_0 + \dots + i_n + j_0 + \dots + j_n + 2k < 2t\\ \tilde{g}_{i_0,\dots,i_n,j_0,\dots,j_n} &= g_0^{i_0}\dots g_{n-1}^{i_{n-1}}\cdot v_n^{i_n}\cdot h_0^{j_0}\dots h_n^{j_{n-1}}\cdot w_n^{j_n} \mod M^\ell\\ &\text{with } 1 \le \ell = i_0 + \dots + i_{n-1} + j_0 + \dots + j_{n-1}\\ &\text{and } \deg(\tilde{g}_{\dots}) = i_0 + \dots + i_n + j_0 + \dots + j_n < 2t.\end{aligned}$$

The moduli N and M are *coprime* (since N is an RSA modulus and M is a power of 2) and it is easy to see that the polynomials $\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k}$ on one hand and the polynomials $\tilde{g}_{i_0,\dots,i_n,j_0,\dots,j_n}$ on the other hand are linearly independent.

We have a system of modular polynomial equations in $2n+2$ unknowns and the Coppersmith method does not necessarily imply that we can solve the system of equations. As often in this setting, we have to assume that if the method succeeds, we will be able to recover the prime factors p and q from the set of polynomials we will obtain:

Heuristic 1. *Let $\mathcal{F}$ denote the polynomial set*

$$\mathscr{P} = \left\{\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k} \mid \begin{array}{c} 1 \le k < t,\ (i_0 = 0 \text{ or } j_0 = 0)\\ i_0 + \dots + i_n + j_0 + \dots + j_n + 2k < 2t \end{array}\right\}$$
$$\bigcup \left\{\tilde{g}_{i_0,\dots,i_n,j_0,\dots,j_n} \mid \begin{array}{c} 1 \le \ell = i_0 + \dots + i_{n-1} + j_0 + \dots + j_{n-1}\\ \deg(\tilde{g}_{\dots}) = i_0 + \dots + i_n + j_0 + \dots + j_n < 2t \end{array}\right\}.$$

We assume that the set of polynomials we get by applying Coppersmiths method with the polynomial set $\mathscr{P}$ define an algebraic variety of dimension 0.

Theorem 5. *Under Heuristic 1, given as inputs an RSA modulus $N = p \cdot q$ and the linear congruential generator parameters a and b such that $p = v_0 + Mv_1 + \dots + M^n v_n$ and $q = w_0 + Mw_1 + \dots + M^n w_n$. (where v_0 and w_0 are two random seeds and $v_{i+1} = av_i + b \mod M$ and $w_{i+1} = aw_i + b$ for $i \in \mathbb{N}$), we can recover the prime factors p and q in polynomial time in* $\log(N)$ *for any $n \geqslant 2$.*

Bounds for the Polynomials Modulo *N*. We consider the set

$$\begin{aligned}\mathscr{P}_f = \{\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k} &= v_0^{i_0}\dots v_n^{i_n}\cdot w_0^{j_0}\dots w_n^{j_n}\cdot f^k \mod N^k\\ &\mid 1 \le k < t,\ (i_0 = 0 \text{ or } j_0 = 0)\\ &\text{and } \deg(\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k}) = i_0 + \dots + i_n + j_0 + \dots + j_n + 2k < 2t\}\end{aligned}$$

as a combinatorial class, with the size function $S_f(\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k}) = i_0 + \dots + i_n + j_0 + \dots + j_n + 2k$ and the parameter function $\chi_f(\tilde{f}_{i_0,\dots,i_n,j_0,\dots,j_n,k}) = k$.

The degree of each variable $v_0, \ldots, v_n, w_0, \ldots, w_n$ is 1, whereas the degree of f is 1. For the sake of simplicity, we can consider $0 \leqslant k < t$ since the parameter function is equal to 0 on the elements $f_{\mathbf{k}}$ such that $k = 0$. We use the technique described in the second example of Sect. 4.2 to write $\mathscr{P}_f$ as a disjoint union of three sets (depending on the values i_0 and j_0) and consider it as

$$\underbrace{(\varepsilon + \mathcal{Z}\textsc{Seq}(\mathcal{Z}) + \mathcal{Z}\textsc{Seq}(\mathcal{Z}))}_{v_0, w_0} \times \prod_{k=1}^{n} \underbrace{\textsc{Seq}(Z)}_{v_k} \times \prod_{k=1}^{n} \underbrace{\textsc{Seq}(Z)}_{w_k} \times \underbrace{\textsc{Seq}(uZ^2)}_{f} \times \underbrace{\textsc{Seq}(Z)}_{\text{dummy var,}}$$

which leads to the following generating function:

$$F(u,z) = \left(1 + \frac{z}{1-z} + \frac{z}{1-z}\right) \frac{1}{(1-z)^{2n}} \frac{1}{1-uz^2} \frac{1}{1-z} = \frac{1+z}{(1-z)^{2n+2}} \frac{1}{1-uz^2}\,.$$

We take the partial derivative in u, then let $u = 1$, and finally take the equivalent when $z \to 1$:

$$\left.\frac{\partial F}{\partial u}(u,z)\right|_{u=1} = \frac{z^2}{(1-z)^{2n+4}(1+z)} \underset{z \to 1}{\sim} \frac{1}{2(1-z)^{2n+4}}.$$

Applying Theorem 3, since $2t \sim 2t - 1$, one finally gets

$$\chi_{f,<2t}(\mathscr{P}_f) \sim \frac{1}{2(2n+3)!}(2t)^{2n+3}.$$

Bounds for the Polynomials Modulo *M*. We consider the set

$$\begin{aligned}\mathscr{P}_g = \{\tilde{g}_{i_0,\ldots,i_n,j_0,\ldots,j_n} = g_0^{i_0} \ldots g_{n-1}^{i_{n-1}} \cdot v_n^{i_n} \cdot h_0^{j_0} \ldots h_n^{j_{n-1}} \cdot w_n^{j_n} \mod M^\ell \\ \mid 1 \le \ell = i_0 + \ldots + i_{n-1} + j_0 + \ldots + j_{n-1} \\ \text{and } \deg(\tilde{g}_{i_0,\ldots,i_n,j_0,\ldots,j_n}) = i_0 + \ldots + i_n + j_0 + \ldots + j_n < 2t\}\end{aligned}$$

as a combinatorial class, with the size function $S_g(\tilde{g}_{i_0,\ldots,i_n,j_0,\ldots,j_n}) = i_0 + \ldots + i_n + j_0 + \ldots + j_n$ and the parameter function $\chi_g(\tilde{g}_{i_1,\ldots,i_n,j_0,\ldots,j_n}) = i_0 + \ldots + i_{n-1} + j_0 + \ldots + j_{n-1}$. The degree of each polynomial g_k is 1, as well as the degrees of v_n and w_n. For the sake of simplicity, we can consider $0 \leqslant \ell$ since the parameter function is equal to 0 on the elements such that $\ell = 0$. We thus consider $\mathscr{P}_g$ as

$$\prod_{k=0}^{n-1} \underbrace{\textsc{Seq}(uZ)}_{g_k} \times \underbrace{\textsc{Seq}(Z)}_{v_n} \times \prod_{k=0}^{n-1} \underbrace{\textsc{Seq}(uZ)}_{h_k} \times \underbrace{\textsc{Seq}(Z)}_{w_n} \times \underbrace{\textsc{Seq}(Z)}_{\text{dummy var.}}$$

which leads to the following generating function:

$$G(u,z) = \frac{1}{(1-uz)^{2n}} \frac{1}{(1-z)^2} \frac{1}{1-z}.$$

We take the partial derivative in u, then let $u = 1$, and finally take the equivalent when $z \to 1$:

$$\frac{\partial G}{\partial u}(u,z)\bigg|_{u=1} = \frac{2nz}{(1-z)^{2n+4}} \underset{z\to 1}{\sim} \frac{2n}{(1-z)^{2n+4}}.$$

Applying Theorem 3, since $2t \sim 2t-1$, one finally gets

$$\chi_{g,<2t}(\mathscr{P}_g) \sim \frac{2n}{(2n+3)!}(2t)^{2n+3}.$$

Bounds for the Monomials Modulo *M*. We consider the set

$$\mathscr{M}=\{{v_0}^{i_0}\ldots{v_n}^{i_n}\cdot{w_0}^{j_0}\ldots{w_n}^{j_n} \mod M^\ell \mid 0 \leqslant \ell=i_0+\ldots+i_n+j_0+\ldots+j_n < 2t\}$$

as a combinatorial class, with the size function $S_x({v_0}^{i_0}\ldots{v_n}^{i_n}\cdot{w_0}^{j_0}\ldots{w_n}^{j_n}) = i_0+\ldots+i_n+j_0+\ldots+j_n$ and the parameter one $\chi_x({v_0}^{i_0}\ldots{v_n}^{i_n}\cdot{w_0}^{j_0}\ldots{w_n}^{j_n}) = i_0+\ldots+i_n+j_0+\ldots+j_n$. The degree of each variable x_k is 1. We thus consider $\mathscr{M}$ as

$$\prod_{k=0}^{n}\underbrace{\textsc{Seq}(uZ)}_{v_k}\times\prod_{k=0}^{n}\underbrace{\textsc{Seq}(uZ)}_{w_k}\times\underbrace{\textsc{Seq}(Z)}_{\text{dummy var,}}$$

which leads to the following generating function:

$$H(u,z) = \frac{1}{(1-uz)^{2n+2}}\frac{1}{1-z}.$$

We take the partial derivative in u, then let $u = 1$, and finally take the equivalent when $z \to 1$:

$$\frac{\partial H}{\partial u}(u,z)\bigg|_{u=1} = \frac{(2n+2)z}{(1-z)^{2n+4}} \underset{z\to 1}{\sim} \frac{2n+2}{(1-z)^{2n+4}}.$$

Applying Theorem 3, since $2t \sim 2t-1$, one finally gets

$$\chi_{x,<2t}(\mathscr{M}) \sim \frac{2n+2}{(2n+3)!}(2t)^{2n+3}.$$

Condition. If we denote by $\nu = \chi_{f,<te}(\mathscr{P}_f)$, $\mu = \chi_{g,<te}(\mathscr{P}_g)$ and $\xi = \chi_{x,<te}(\mathscr{M})$, the condition for Coppersmith's method is $N^\nu \cdot M^\mu > M^\xi$, where

$$\frac{\nu}{\xi-\mu} = \frac{\chi_{f,<te}(\mathscr{P}_f)}{\chi_{x,<te}(\mathscr{M})-\chi_{g,<te}(\mathscr{P}_g)} \underset{z\to 1}{\sim} \frac{\frac{1}{2(2n+3)!}(2t)^{2n+3}}{\frac{2n+2}{(2n+3)!}(2t)^{2n+3}-\frac{2n}{(2n+3)!}(2t)^{2n+3}} \underset{z\to 1}{\sim} \frac{1}{4}$$

which leads to the bound $M < N^{1/4}$ (and since N is an even power of M we obtain $M \leqslant N^{1/6}$ and thus $n \geqslant 2$).

Table 2. Bounds in Coppersmith (simplified) success condition (1)

n	t	1	2	3	4	5	6	7	8
2	Polynomial bound	4	38	186	654	1866	4602	10182	**20706**
	Monomial bound	6	48	216	720	1980	4752	10296	**20592**
3	Polynomial bound	6	68	402	1688	5682	**16340**		
	Monomial bound	8	80	440	1760	5720	**16016**		
4	Polynomial bound	10	152	1206	6704	**29416**			
	Monomial Bound	12	168	1260	6720	**28560**			
5	Polynomial bound	12	206	1842	**11486**				
	Monomial bound	14	224	1904	**11424**				

Remark 6. In the previous attack, we actually considered a very naive prime number generation algorithm. However, a prime number generation algorithm based on this (bad) design principle would probably use instead an incremental algorithm and output prime numbers $p = (v_0 + Mv_1 + \ldots + M^n v_n) + \alpha$ and $q = (w_0 + Mw_1 + \ldots + M^n w_n) + \beta$ for some α and β in $\mathbb{N}$. Thanks to the prime number theorem, these values are likely to be small and the previous algorithm can be run[6] after an exhaustive search of α and β.

Concrete bounds. The previous analysis leads to the bound $M < N^{1/4}$ when t goes to ∞. Actually to reach the (simplified) success condition (1) in Coppersmith method for $n \geqslant 2$, we need only small values of t as shown in Table 2.

Unfortunately, even if t is small, the constructed matrix is of huge dimension (since the number of monomials is quite large) and the computation which is theoretically polynomial-time becomes in practice prohibitive (for instance, for $n = 3$ and $t = 6$, the matrix is of dimension 6473). These attacks are netherthe-less good evidence of a weakness in this key generation scheme. For $n = 1$ (i.e., $M = N^{1/4}$), the polynomial time attack does not apply, but one may combine it with an exhaustive search to retrieve a small part of v_0, v_1, w_0 and w_1 to retrieve the other (bigger) part of the seeds.

6.2 PKCS#1 V1.5 Padding Encryption with Weak Pseudorandomness

PKCS#1 v1.5 describes a particular encoding padding for RSA encryption. Let N be RSA an modulus of byte-length k (i.e., $2^{8(k-1)} < N < 2^{8k}$, e be a public exponent coprime to the Euler totient $\varphi(N)$ and m be a message of ℓ-byte with $\ell < k - 11$. The PKCS#1 v1.5 padding of m is defined as follows:

1. A randomizer r consisting in $k-3-\ell \geqslant 8$ nonzero bytes is generated uniformly at random;

[6] Alternatively, one can also adapt the algorithm by adding unknowns for α and β to the multivariate modular polynomial system.

2. $\mu(m, r)$ is the integer converted from the octet-string:

$$\mu(m, r) = \mathtt{0002}_{16}||r||\mathtt{00}_{16}||m. \tag{2}$$

The encryption of m is then defined as $c = \mu(m, r)^e \mod N$. To decrypt $c \in \mathbb{Z}_N^*$, compute $c^d \mod N$ (where $ed \equiv 1 \mod \varphi(N)$), convert the result to a k-byte octet-string and parse it according to Eq. (2). If the string cannot be parsed unambiguously or if r is shorter than eight octets, the decryption algorithm $\mathcal{D}$ outputs $\perp$; otherwise, $\mathcal{D}$ outputs the plaintext m.

The PKCS#1 v1.5 padding has been known to be insecure for encryption since Bleichenbachers famous chosen ciphertext attack [7]. Several additionnal attacks were published since 1998 (e.g., [1,14,21]).

Fouque *et al.* [16] suggested to consider the setting of the randomness generation used in padding functions for encryption. In PKCS#1 v1.5 padding, the randomizer shall be pseudorandomly generated (according to the RFC which defines it [24]) and since it is still widely used in practice (e.g., TLS, XML Encryption standard, Hardware token...)n it seems interesting to investigate its security when the randomizer is constructed by concatenating the outputs of a linear congruential generator. We consider several scenarios (linear congruential generator, truncated linear congruential generators, multiple ciphertexts ...) and we apply our toolbox to all of them.

Scenario 1: Linear Congruential Generator. The first attack scenario can be seen as a *chosen distribution attack*. These attacks were introduced by Bellare et al. [3] to model attacks where an adversary can control the distribution of both messages and random coins used in an encryption scheme. We assume that the adversary can control the message (as in the classical notion of semantic security for public-key encryption schemes [17]) and that the randomizer used in the PKCS#1 v1.5 padding is constructed by concatenating the outputs of a linear congruential generator (with a seed picked uniformly at random). The adversary will choose two messages m_0 and m_1 of the same byte-length $\ell < k - 11$ (where k is the byte length of the RSA modulus N) and the challenger will pick at random a seed x_1 of byte-length ρ. It will compute

$$x_{i+1} = ax_i + b \mod M$$

for $i \in \{2, \dots, n-1\}$ where $n = (k-3-\ell)/\rho$ and $M = 2^{8\rho}$. The challenge ciphertext will be $c = \mu(m_b, r)^e \mod N$ where b is a bit picked uniformly at random by the challenger and the randomizer r is the concatenation of $x_1, \dots, x_n$. We have

$$\begin{aligned} \mu(m_b, r) &= \mathtt{0002}_{16}||r||\mathtt{00}_{16}||m_b \\ &= \mathtt{0002}_{16}||x_1||x_2||\dots||x_n||\mathtt{00}_{16}||m_b \\ &= (\tilde{\alpha_1}x_1 + \tilde{\alpha_2}x_2 + \dots + \tilde{\alpha_n}x_n + \tilde{\beta}) \end{aligned}$$

where this last expression is the integer converted from the octet-string with the $\tilde{\alpha_i}$'s are known public constant and $\tilde{\beta}$ is the integer converted from the string

m_b. If we divide c by $\tilde{\alpha_1}^e$, we obtain

$$c = (x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n + \beta)^e \mod N$$

where $\alpha_i = \tilde{\alpha_i}/\tilde{\alpha_1}$ for $i \in \{2, \ldots, n\}$ and $\beta = \tilde{\beta}/\tilde{\alpha_1}$.

Description of the attack. The adversary is therefore looking for the solutions of the following modular multivariate polynomial system: of *monic* polynomial equations:

$$\begin{cases} f = (x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n + \beta)^e \mod N \\ g_1 = x_1 - a x_2 + b \mod M \\ \quad\vdots \\ g_{n-1} = x_{n-1} - a x_n + b \mod M \end{cases}$$

where β can be derived easily from the value m_b. The attack consists in applying Coppersmith Method for multivariate polynomials with two moduli (see Sect. 2) to the two systems derived from the two possible values for m_b.

As above, the most complicated part is the choice of the collection of polynomials constructed from the polynomials that occur in this system. Our analysis brought out the following polynomial family (parameterized by some integer $t \in \mathbb{N}$):

$$\begin{aligned} \tilde{f}_{i_1,\ldots,i_n,j} &= x_1^{i_1} \ldots x_n^{i_n} \cdot f^j \mod N^j \\ &\quad \text{with } 1 \le j < t,\ 0 \le i_1 < e \text{ and } \deg(\tilde{f}_{\ldots}) = i_1 + \ldots + i_n + je < te \\ \tilde{g}_{i_1,\ldots,i_n} &= g_1^{i_1} \ldots g_{n-1}^{i_{n-1}} \cdot x_n^{i_n} \mod M^k \\ &\quad \text{with } 1 \le k = i_1 + \ldots + i_{n-1} \text{ and } \deg(\tilde{g}_{\ldots}) = i_1 + \ldots + i_n < te. \end{aligned}$$

As in the previous section, the moduli N and M are *coprime* (since N is an RSA modulus and M is a power of 2). Moreover, it is easy to see that the polynomials $\tilde{f}_{i_1,\ldots,i_n,j}$ on one hand and the polynomials $\tilde{g}_{i_0,\ldots,i_n}$ on the other hand are linearly independent. Indeed, these polynomials have distinct leading monomials and are monic.

We have a system of modular polynomial equations in n unknowns and the Coppersmith method does not necessarily imply that we can solve the system of equations. Thus, we also have to assume that if the method succeeds, we will be able to recover the seed x_1 from the set of polynomials we will obtain:

Heuristic 2. *Let $\mathscr{P}$ denote the polynomial set*

$$\mathscr{P} = \left\{ \tilde{f}_{i_1,\ldots,i_n,j} \mid \begin{matrix} 1 \le j < t,\ 0 \le i_1 < e \\ i_1 + \ldots + i_n + je < te \end{matrix} \right\} \bigcup \left\{ \tilde{g}_{i_1,\ldots,i_n} \mid \begin{matrix} 1 \le k = i_1 + \ldots + i_{n-1} \\ i_1 + \ldots + i_n < te \end{matrix} \right\}.$$

We assume that the set of polynomials we get by applying Coppersmiths method with the polynomial set $\mathscr{P}$ define an algebraic variety of dimension 0.

Theorem 7. *Under Heuristic 2, given as inputs an RSA modulus N, the linear congruential generator parameters a and b, two messages m_0 and m_1 and a* PKCS*#1 v1.5 ciphertext $c = \mu(m_b, r)$ for some bit $b \in \{0,1\}$ such that the randomizer r is the concatenation of $x_1, \ldots, x_n$ (where x_1 is a random seed of size M and $x_{i+1} = ax_i + b \mod M$ for $i \in \mathbb{N}$), we can recover the seed x_1 (and thus the bit b) in polynomial time in $\log(N)$ as soon as $M < N^{1/e}$.*

Bounds for the Polynomials Modulo N. We consider the set

$$\begin{aligned}\mathscr{P}_f = \{\tilde{f}_{i_1,\ldots,i_n,j} = x_1^{i_1} \cdots x_n^{i_n} \cdot f^j \mod N^j \mid 1 \le j < t,\, 0 \le i_1 < e \\ \text{and } \deg(\tilde{f}_{i_1,\ldots,i_n,j}) = i_1 + \ldots + i_n + je < te\}\end{aligned}$$

as a combinatorial class, with the size function $S_f(\tilde{f}_{i_1,\ldots,i_n,j}) = i_1 + \ldots + i_n + je$ and the parameter function $\chi_f(\tilde{f}_{i_1,\ldots,i_n,j}) = j$. The degree of each variable x_k is 1, whereas the degree of f is e. For the sake of simplicity, we can consider $0 \leqslant j < t$ since the parameter function is equal to 0 on the elements such that $j = 0$. We thus consider $\mathscr{P}_f$ as

$$\underbrace{(\varepsilon + \mathcal{Z} + \ldots + \mathcal{Z}^{e-1})}_{x_1} \times \prod_{k=2}^{n} \underbrace{\textsc{Seq}(Z)}_{x_k} \times \underbrace{\textsc{Seq}(uZ^e)}_{f} \times \underbrace{\textsc{Seq}(Z)}_{\text{dummy var.}}$$

which leads to the following generating function:

$$F(u,z) = (1 + z + \ldots + z^{e-1}) \frac{1}{(1-z)^{n-1}} \frac{1}{1-uz^e} \frac{1}{1-z}.$$

We take the partial derivative in u and then let $u = 1$:

$$\left.\frac{\partial F}{\partial u}(u,z)\right|_{u=1} = (1 + z + \ldots + z^{e-1}) \frac{1}{(1-z)^n} \frac{z^e}{(1-z^e)^2}.$$

We take the equivalent when $z \to 1$, using the formula $1 - z^e \sim e(1-z)$:

$$\left.\frac{\partial F}{\partial u}(u,z)\right|_{u=1} \underset{z \to 1}{\sim} \frac{1}{e(1-z)^{n+2}}.$$

Applying Theorem 3, since $te \sim te - 1$, one finally gets

$$\chi_{f,<te}(\mathscr{P}_f) \sim \frac{1}{e(n+1)!}(te)^{n+1}.$$

Bounds for the Polynomials Modulo M. We consider the set

$$\begin{aligned}\mathscr{P}_g = \{\tilde{g}_{i_1,\ldots,i_n} = g_1^{i_1} \cdots g_{n-1}^{i_{n-1}} \cdot x_n^{i_n} \mod M^{i_1 + \cdots + i_{n-1}} \mid 1 \le k = i_1 + \cdots + i_{n-1} \\ \text{and } \deg(\tilde{g}_{i_1,\ldots,i_n}) = i_1 + \ldots + i_n < te\}\end{aligned}$$

as a combinatorial class, with the size function $S_{_g}(\tilde{g}_{i_1,\ldots,i_n}) = i_1+\ldots+i_n$ and the parameter function $\chi_{_g}(\tilde{g}_{i_1,\ldots,i_n}) = i_1+\ldots+i_{n-1}$. The degree of each polynomial g_k is 1, as well as the degree of x_n. For the sake of simplicity, we can consider $0 \leqslant k$ since the parameter function is equal to 0 on the elements such that $k = 0$. We thus consider $\mathscr{P}_{_g}$ as

$$\prod_{k=1}^{n-1} \underbrace{\textsc{Seq}(uZ)}_{g_k} \times \underbrace{\textsc{Seq}(Z)}_{x_n} \times \underbrace{\textsc{Seq}(Z)}_{\text{dummy var,}}$$

which leads to the following generating function:

$$G(u,z) = \frac{1}{(1-uz)^{n-1}} \frac{1}{1-z} \frac{1}{1-z}.$$

We take the partial derivative in u, then let $u = 1$, and finally take the equivalent when $z \to 1$:

$$\left.\frac{\partial G}{\partial u}(u,z)\right|_{u=1} = \frac{(n-1)z}{(1-z)^{n+2}} \underset{z\to 1}{\sim} \frac{n-1}{(1-z)^{n+2}}.$$

Applying Theorem 3, since $te \sim te-1$, one finally gets

$$\chi_{_{g,<te}}(\mathscr{P}_{_g}) \sim \frac{n-1}{(n+1)!}(te)^{n+1}.$$

Bounds for the Monomials Modulo $\boldsymbol{M}$. We consider the set

$$\mathscr{M} = \{{x_1}^{i_1} \ldots {x_n}^{i_n} \mod M^{i_1+\ldots+i_n} \mid 0 \leqslant i_1+\ldots+i_n < te\}.$$

as a combinatorial class, with the size function $S_{_x}({x_1}^{i_1} \ldots {x_n}^{i_n}) = i_1+\ldots+i_n$ and the parameter function $\chi_{_x}({x_1}^{i_1} \ldots {x_n}^{i_n}) = i_1+\ldots+i_n$. The degree of each variable x_k is 1. We thus consider $\mathscr{M}$ as

$$\prod_{k=1}^{n} \underbrace{\textsc{Seq}(uZ)}_{x_k} \times \underbrace{\textsc{Seq}(Z)}_{\text{dummy var,}}$$

which leads to the following generating function:

$$H(u,z) = \frac{1}{(1-uz)^{n}} \frac{1}{1-z}.$$

We first take the partial derivative in u, then let $u = 1$, and finally take the equivalent when $z \to 1$:

$$\left.\frac{\partial H}{\partial u}(u,z)\right|_{u=1} = \frac{nz}{(1-z)^{n+2}} \underset{z\to 1}{\sim} \frac{n}{(1-z)^{n+2}}.$$

Applying Theorem 3, since $te \sim te-1$, one finally gets

$$\chi_{_{x,<te}}(\mathscr{M}) \sim \frac{n}{(n+1)!}(te)^{n+1}.$$

Condition. If we denote by $\nu = \chi_{f,<te}(\mathscr{P}_f)$, $\mu = \chi_{g,<te}(\mathscr{P}_g)$ and $\xi = \chi_{x,<te}(\mathscr{M})$, the condition for Coppersmith's method is $N^\nu \cdot M^\mu > M^\xi$, where

$$\frac{\nu}{\xi-\mu} = \frac{\chi_{f,<te}(\mathscr{P}_f)}{\chi_{x,<te}(\mathscr{M}) - \chi_{g,<te}(\mathscr{P}_g)} \underset{z\to 1}{\sim} \frac{\frac{1}{e(n+1)!}(te)^{n+1}}{\frac{n}{(n+1)!}(te)^{n+1} - \frac{n-1}{(n+1)!}(te)^{n+1}} \underset{z\to 1}{\sim} \frac{1}{e}$$

which leads to the expected bound $M < N^{1/e}$.

Scenario 2: Truncated Linear Congruential Generator. In 1997, Bellare et al. [4] broke the Digital Signature Algorithm (DSA) when the random nonces used in signature generation are computed using a linear congruential generator. They also broke the DSA signature scheme if the nonces are computed by a *truncated* linear congruential generator. In order to pursue the parallel with their work, in the second attack scenario, we the previous analysis to the case where the randomize in PKCS#1 v1.5 padding is constructed by concatenating any consecutive bits of the outputs of a linear congruential generator (with a seed picked uniformly at random).

More precisely, the seed of the linear congruential generator is now denoted $v_1 = y_1 + x_1 \cdot 2^{\gamma_y \log M} + z_1 \cdot 2^{\gamma_x \log M + \gamma_y \log M}$, where y_1 has $\gamma_y \log M$ bits, x_1 has $\gamma_x \log M$ bits, z_1 has $\gamma_z \log M$ bits and $\gamma_x + \gamma_y + \gamma_z = 1$. We define the (weak)pseudorandom sequence by $v_{i+1} = av_i + b \mod M$ for $i \in \mathbb{N}$ (with public a, b and M). We denote $v_i = y_i + x_i \cdot 2^{\gamma_y \log M} + z_i \cdot 2^{\gamma_x \log M + \gamma_y \log M}$, where y_i has $\gamma_y \log M$ bits, x_i has $\gamma_x \log M$ bits and z_i has $\gamma_z \log M$ bits.

As above, the challenge ciphertext will be $c = \mu(m_b, r)^e \mod N$ where b is a bit picked uniformly at random by the challenger and the randomizer r is the concatenation of $x_1, \ldots, x_n$ for $n = (k - 3 - \ell)/(8\gamma_x \log M)$. We have

$$\begin{aligned} \mu(m_b, r) &= \mathtt{0002}_{16}||r||\mathtt{00}_{16}||m_b \\ &= (\tilde{\alpha_1} x_1 + \tilde{\alpha_2} x_2 + \ldots + \tilde{\alpha_n} x_n + \tilde{\beta}). \end{aligned}$$

Description of the attack. The adversary is looking for the solutions of the following multivariate modular polynomial system: of *monic* polynomial equations:

$$\begin{cases} f = (x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n + \beta)^e \mod N \\ g_1 = x_1 + a' y_1 + a'' z_1 + b x_2 + b' y_2 + b'' z_2 + c \mod M \\ \quad \vdots \\ g_{n-1} = x_{n-1} + a' y_{n-1} + a'' y_{n-1} + b x_n + b' y_n + b'' z_n + c \mod M \end{cases}$$

where β can be derived easily from the value m_b and the constants $\alpha_2, \ldots, \alpha_N$, a', a'', b, b' and b'' are public. As in the previous scenario, the attack consists in applying Coppersmith Method for multivariate polynomials with two moduli (see Sect. 2) to the two systems derived from the two possible values for m_b.

For the choice of the polynomials collection, we choose in this scenario the following polynomial family (parameterized by some integer $t \in \mathbb{N}$):

$$\begin{aligned} \tilde{f}_{\boldsymbol{i},\boldsymbol{i}',\boldsymbol{i}'',j} &= x_1^{i_1} \dots x_n^{i_n} \cdot y_1^{i'_1} \dots y_n^{i'_n} \cdot z_1^{i''_1} \dots z_n^{i''_n} \cdot f^j \mod N^j \\ &\text{with } 1 \le j < t,\ 0 \le i_1 < e \text{ and } \deg(\tilde{f}_{\dots}) < te \\ \tilde{g}_{\boldsymbol{i},\boldsymbol{i}',\boldsymbol{i}''} &= g_1^{i_1} \dots g_{n-1}^{i_{n-1}} \cdot x_n^{i_n} \cdot y_1^{i'_1} \dots y_n^{i'_n} \cdot z_1^{i''_1} \dots z_n^{i''_n} \mod M^k \\ &\text{with } 1 \le k = i_1 + \dots + i_{n-1} \text{ and } \deg(\tilde{g}_{\dots}) < te. \end{aligned}$$

As above, the moduli N and M are *coprime* and the polynomials $\tilde{f}_{i_1,\dots,i_n,j}$ on one hand and the polynomials $\tilde{g}_{i_0,\dots,i_n}$ on the other hand are linearly independent.

Again the Coppersmith method does not necessarily imply that we can solve the system of equations and we have to make the following heuristic:

Heuristic 3. *Let $\mathscr{P}$ denote the polynomial set*

$$\mathscr{P} = \left\{ \tilde{f}_{\boldsymbol{i},\boldsymbol{i}',\boldsymbol{i}'',j} \mid {\scriptstyle \begin{array}{c} 1 \le j < t,\ 0 \le i_1 < e \\ i_1 + \dots + i_n + +i'_1 + \dots + i'_{n-1} + i''_1 + \dots + i''_{n-1} + je < te \end{array}} \right\}$$
$$\bigcup \left\{ \tilde{g}_{\boldsymbol{i},\boldsymbol{i}',\boldsymbol{i}''} \mid {\scriptstyle \begin{array}{c} 1 \le k = i_1 + \dots + i_{n-1} \\ i_1 + \dots + i_n + +i'_1 + \dots + i'_{n-1} + i''_1 + \dots + i'_{n-1} < te \end{array}} \right\}.$$

We assume that the set of polynomials we get by applying Coppersmiths method with the polynomial set $\mathscr{P}$ define an algebraic variety of dimension 0.

Theorem 8. *Under Heuristic 3, given as inputs an RSA modulus N, the truncated linear congruential generator parameters a and b, two messages m_0 and m_1 and a* PKCS*#1 v1.5 ciphertext $c = \mu(m_b, r)$ for some bit $b \in \{0, 1\}$ such that the randomizer r is the concatenation of truncations of $v_1, \dots, v_n$ (where v_1 is a random seed of size M and $v_{i+1} = av_i + b \mod M$ for $i \in \mathbb{N}$), we can recover the seed v_1 (and thus the bit b) in polynomial time in* $\log(N)$ *as soon as $M < N^{1/e}$.*

Due to lack of space, the details of the computation are provided in the full version.

Scenario 3: Truncated Linear Congruential Generator and Multiple Ciphertexts. We can also extend the first *chosen distribution attack* by letting the adversary control m pair of messages (as in the semantic security for multiple ciphertexts, see e.g. [19]) and that the randomizer used in all the PKCS#1 v1.5 paddings are constructed by concatenating the successive outputs of a linear congruential generator (with a unique seed picked uniformly at random). We also apply our toolbox to this scenario and for an RSA modulus N with a public exponent e and a linear congruential generator with modulus M, our heuristic attacks are polynomial-time in $\log(N)$ for the $M < N^{m/e}$ (see details in the full version).

Acknowledgments. The authors are supported in part by the French ANR JCJC ROMAnTIC project (ANR-12-JS02-0004). The authors thank Aurélie Bauer for her participation and contributions in the early stage of this work.

References

1. Bauer, A., Coron, J.-S., Naccache, D., Tibouchi, M., Vergnaud, D.: On the broadcast and validity-checking security of PKCS#1 v1.5 encryption. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 1–18. Springer, Heidelberg (2010)
2. Bauer, A., Vergnaud, D., Zapalowicz, J.-C.: Inferring sequences produced by nonlinear pseudorandom number generators using Coppersmith's methods. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 609–626. Springer, Heidelberg (2012)
3. Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: how to protect against bad randomness. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 232–249. Springer, Heidelberg (2009)
4. Bellare, M., Goldwasser, S., Micciancio, D.: "Pseudo-random" number generation within cryptographic algorithms: the DSS case. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 277–291. Springer, Heidelberg (1997)
5. Blackburn, S.R., Gómez-Pérez, D., Gutierrez, J., Shparlinski, I.: Predicting nonlinear pseudorandom number generators. Math. Comput. **74**(251), 1471–1494 (2005)
6. Blackburn, S.R., Gómez-Pérez, D., Gutierrez, J., Shparlinski, I.: Reconstructing noisy polynomial evaluation in residue rings. J. Algorithms **61**(2), 47–59 (2006)
7. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 1–12. Springer, Heidelberg (1998)
8. Blömer, J., May, A.: A tool kit for finding small roots of bivariate polynomials over the integers. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 251–267. Springer, Heidelberg (2005)
9. Boyar, J.: Inferring sequences produced by a linear congruential generator missing low-order bits. J. Cryptology **1**(3), 177–184 (1989)
10. Boyar, J.: Inferring sequences produced by pseudo-random number generators. J. ACM **36**(1), 129–141 (1989)
11. Coppersmith, D.: Finding a small root of a bivariate integer equation; factoring with high bits known. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 178–189. Springer, Heidelberg (1996)
12. Coppersmith, D.: Finding a small root of a univariate modular equation. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 155–165. Springer, Heidelberg (1996)
13. Coppersmith, D.: Small solutions to polynomial equations, and low exponent RSA vulnerabilities. J. Cryptology **10**(4), 233–260 (1997)
14. Coron, J.-S., Joye, M., Naccache, D., Paillier, P.: New attacks on PKCS#1 v1.5 encryption. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 369–381. Springer, Heidelberg (2000)
15. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press, Cambridge (2009)
16. Fouque, P.-A., Tibouchi, M., Zapalowicz, J.-C.: Recovering private keys generated with weak PRNGs. In: Stam, M. (ed.) IMACC 2013. LNCS, vol. 8308, pp. 158–172. Springer, Heidelberg (2013)
17. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
18. Herrmann, M., May, A.: Attacking power generators using unravelled linearization: when do we output too much? In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 487–504. Springer, Heidelberg (2009)

19. Hofheinz, D., Jager, T.: Tightly secure signatures and public-key encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 590–607. Springer, Heidelberg (2012)
20. Howgrave-Graham, N.: Finding small roots of univariate modular equations revisited. In: Darnell, M. (ed.) Cryptography and Coding 1997. LNCS, vol. 1355, pp. 131–142. Springer, Heidelberg (1997)
21. Jager, T., Schinzel, S., Somorovsky, J.: Bleichenbacher's attack strikes again: breaking PKCS#1 v1.5 in XML encryption. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 752–769. Springer, Heidelberg (2012)
22. Jochemsz, E., May, A.: A strategy for finding roots of multivariate polynomials with new applications in attacking RSA variants. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 267–282. Springer, Heidelberg (2006)
23. Joux, A., Stern, J.: Lattice reduction: a toolbox for the cryptanalyst. J. Cryptology **11**(3), 161–185 (1998)
24. Kaliski, B.: PKCS #1: RSA Encryption Version 1.5. RFC 2313, Internet Engineering Task Force, March 1998. http://www.rfc-editor.org/rfc/rfc2313.txt
25. Koshiba, T.: On sufficient randomness for secure public-key cryptosystems. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 34–47. Springer, Heidelberg (2002)
26. Lipton, R.J., Regan, K.W.: People, Problems, and Proofs - Essays from Gödel's Lost Letter: 2010. Springer, Berlin (2013)
27. May, A.: Using lll-reduction for solving RSA and factorization problems. In: Nguyen, P.Q., Vallée, B. (eds.) The LLL Algorithm - Survey and Applications. Information Security and Cryptography, pp. 315–348, Springer, Heidelberg (2010). http://dx.org/10.1007/978-3-642-02295-1
28. May, A., Ritzenhofen, M.: Solving systems of modular equations in one variable: how many RSA-encrypted messages does eve need to know? In: Cramer, R. (ed.) PKC 2008. LNCS, vol. 4939, pp. 37–46. Springer, Heidelberg (2008)
29. May, A., Ritzenhofen, M.: Implicit factoring: on polynomial time factoring given only an implicit hint. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 1–14. Springer, Heidelberg (2009)
30. Ritzenhofen, M.: On efficiently calculating small solutions of systems of polynomial equations: lattice-based methods and applications to cryptography. Ph.D. thesis, Ruhr University Bochum (2010). http://www-brs.ub.ruhr-uni-bochum.de/netahtml/HSS/Diss/RitzenhofenMaike/
31. Stern, J.: Secret linear congruential generators are not cryptographically secure. In: 28th FOCS, pp. 421–426. IEEE Computer Society Press, October 1987