Solving Linear Equations Modulo Divisors: On Factoring Given Any Bits

Mathias Herrmann, Alexander May *

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

Abstract. We study the problem of finding solutions to linear equations modulo an unknown divisor p of a known composite integer N. An important application of this problem is factorization of N with given bits of p. It is well-known that this problem is polynomial-time solvable if at most half of the bits of p are unknown and if the unknown bits are located in one *consecutive* block. We introduce an heuristic algorithm that extends *factoring with known bits* to an arbitrary number n of blocks. Surprisingly, we are able to show that $\ln(2) \approx 70\%$ of the bits are sufficient for any n in order to find the factorization. The algorithm's running time is however exponential in the parameter n. Thus, our algorithm is polynomial time only for $n = \mathcal{O}(\log \log N)$ blocks.

Keywords: Lattices, small roots, factoring with known bits

1 Introduction

Finding solutions to polynomial modular equations is a central mathematical problem and lies at the heart of almost any cryptanalytic approach. For instance, most symmetric encryption functions can be interpreted as polynomial transformations from plaintexts to ciphertexts. Solving the corresponding polynomial equations yields the secret key.

Among all polynomial equations the linear equations $f(x_1, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n$ play a special role, since they are often easier to solve. Many problems already admit a linear structure. For instance, the subset sum problem for finding a subset of $s_1, \ldots, s_n$ that sums to t asks for a 0,1-solution $(y_1, \ldots, y_n)$ of the linear equation $s_1x_1 + \cdots + s_nx_n - t = 0$. Special instances of this problem can be solved by lattice techniques [CJL+92].

Although many problems are inherently of non-linear type, solution strategies for these problems commonly involve some linearization step. In this work, we address the problem of solving *modular* linear equations $f(x_1, \ldots, x_n) = 0 \bmod N$ for some N with unknown factorization. Note that modular equations usually

* This research was supported by the German Research Foundation (DFG) as part of the project MA 2536/3-1

have many solutions $(y_1, \ldots, y_n) \in \mathbb{Z}_N^n$. An easy counting argument however shows that one can expect a unique solution whenever the product of the unknowns is smaller than the modulus - provided the coefficients a_i are uniformly distributed in $\mathbb{Z}_N$. More precisely, let X_i be upper bounds such that $|y_i| \leq X_i$ for $i = 1 \ldots n$. Then one can roughly expect a unique solution whenever the condition $\prod_i X_i \leq N$ holds.

It is folklore knowledge that under the same condition $\prod_i X_i \leq N$ the unique solution $(y_1, \ldots, y_n)$ can heuristically be recovered by computing a shortest vector in an n-dimensional lattice. In fact, this approach lies at the heart of many cryptanalytic results (see e.g. [GM97,NS01,Ngu04,BM06]). If in turn we have $\prod_i X_i \geq N^{1+\epsilon}$ then the linear equation usually has N^ϵ many solutions, which is exponential in the bit-size of N. So there is no hope to find efficient algorithms that in general improve on this bound, since one cannot even output all roots in polynomial time.

In the late 80's, Hastad [Has88] and Toffin, Girault, Vallée [GTV88] extended the lattice-based approach for linear equations to modular univariate monic polynomials $f(x) = a_0 + a_1 x + \cdots + a_{\delta-1}x^{\delta-1} + x^\delta$. In 1996, Coppersmith [Cop96b] further improved the bounds of [Has88,GTV88] to $|x_0| \leq N^{\frac{1}{\delta}}$ for lattice-based solutions that find small roots of $f(x)$. For modular univariate polynomials $f(x)$ there are again counting arguments that show that this bound cannot be improved in general. Even more astonishing than the improved bound is the fact that Coppersmith's method does neither rely on a heuristic nor on the computation of a shortest vector, but provably provides all roots smaller than this bound and runs in polynomial time using the L^3 algorithm [LLL82].

In the same year, Coppersmith [Cop96a] formulated another rigorous method for bivariate polynomials $f(x, y)$, see also [Cor07]. This method has several nice applications, most notably the problem of *factoring with high bits known* and also an algorithm that shows the deterministic polynomial time equivalence of factoring and computing the RSA secret key [May04,CM07]. In the *factoring with high bits known* problem, one is given an RSA modulus $N = pq$ and an approximation $\tilde{p}$ of p. This enables to compute an approximation $\tilde{q}$ of q, which leads to the bivariate polynomial equation $f(x, y) = (\tilde{p} + x)(\tilde{q} + y) - N$. Finding the unique solution in turn enables to factor. Coppersmith showed that this can be done in polynomial time given 50% of the bits of p and thereby improved upon a result from Rivest and Shamir [RS85], who required 60% of the bits of p. Using an oracle that answers arbitrary questions instead of returning bits of the prime factor, Maurer [Mau95] presented a probabilistic algorithm based on elliptic curves, that factors an integer N in polynomial time making at most $\epsilon \log N$ oracle queries for any $\epsilon > 0$.

In 2001, Howgrave-Graham [HG01] gave a reformulation of the *factoring with high bits known* problem, showing that the remaining bits of p can be recovered if $\gcd(\tilde{p} + x, N)$ is sufficiently large. This can also be stated as finding the root of the linear monic polynomial $f(x) = \tilde{p} + x \bmod p$ where $p \geq N^\beta$ for some $0 < \beta \leq 1$. Later, this was generalized by May [May03] to arbitrary monic

modular polynomials of degree δ which results in the bound $|x_0| \leq N^{\frac{\beta^2}{\delta}}$. The result for *factoring with high bits known* follows for the choice $\beta = \frac{1}{2}$, $\delta = 1$.

Notice that in the *factoring with high bits known* problem, the unknown bits have to be in one consecutive block of bits. This variant of the factorization problem is strongly motivated by side-channel attacks that in most cases enable an attacker to recover some of the bits of the secret key. The attacker is then left with the problem of reconstructing the whole secret out of the obtained partial information. Unfortunately, the unknown part is in general not located in one consecutive bit block but widely spread over the whole bit string. This raises the question whether we can sharpen our tools to this general scenario.

Our contribution: We study the problem of finding small roots of linear modular polynomials $f(x_1, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n + a_{n+1} \bmod p$ for some unknown $p \geq N^\beta$ that divides the known modulus N. This enables us to model the problem of *factoring with high bits known* to an arbitrary number n of unknown blocks. Namely, if the k-th unknown block starts in the ℓ-th bit position we choose $a_k = 2^\ell$.

We are able to show an explicit bound for the product $\prod_i X_i = N^\gamma$, where γ is a function in β and n. For the special case in which $p = N$, i.e. $\beta = 1$ and the modulus p is in fact known, we obtain the previously mentioned folklore bound $\prod_i X_i \leq N$. Naturally, the larger the number n of blocks, the smaller is the bound for $\prod_i X_i$ and the larger is the running time of our algorithm. In other words, the larger the number of blocks, the more bits of p we do have to know in the *factoring with known bits* problem. What is really surprising about our lattice-based method is that even for an arbitrary number n of blocks, our algorithm still requires only a constant fraction of the bits of p. More precisely, a fraction of $\ln(2) \approx 70\%$ of p is always sufficient to recover p.

Unfortunately, the running time for our algorithm heavily depends on n. Namely, the dimension of the lattice basis that we have to L^3-reduce grows exponentially in n. Thus, our algorithm is polynomial time only if $n = \mathcal{O}(\log \log N)$. For larger values of n, our algorithm gets super-polynomial. To the best of our knowledge state-of-the-art general purpose factorization algorithms like the GNFS cannot take advantage of extra information like given bits of one of the prime factors. Thus, our algorithm still outperforms the GNFS for the *factoring with known bits* problem provided that $n = o(\log^{\frac{1}{3}} N \log \log^{\frac{2}{3}} N)$.

We would like to notice that our analysis for arbitrary n yields a bound $\prod_i X_i \leq N^\gamma$ that holds no matter how the size of the unknowns are distributed among the X_i. In case the X_i are of strongly different sizes, one might even improve on the bound N^γ. For our starting point $n = 2$, we sketch such a general analysis for arbitrary sizes of X_1, X_2. The analysis shows that the bound for the product X_1X_2 is minimal when $X_1 = X_2$ and that it converges to the known Coppersmith result $N^{\frac{1}{4}}$ in the extreme case, where one of the X_i is set to $X_i = 1$.

Notice that if one of the upper bounds is set to $X_i = 1$ then the bivariate linear equation essentially collapses to a univariate equation. In this case, we

also obtain the bound $N^{\frac{1}{4}}$ for the *factoring with known bits* problem. Thus, our algorithm does not only include the folklore bound as a special case but also the Coppersmith bound for univariate linear modular equations.

As our lattice-based algorithm eventually outputs multivariate polynomials over the integers, we are using a well-established heuristic [Cop97,BD00] for extracting the roots. We show experimentally that this heuristic works well in practice and always yielded the desired factorization. In addition to previous papers that proposed to use resultant or Gröbner basis computations, we use the multidimensional Newton method from numerical mathematics to efficiently extract the roots.

The paper is organized as follows. Section 2 recalls basic lattice theory. In Section 3 we give the analysis of bivariate linear equations modulo an unknown divisor. As noticed before, we prove a general bound that holds for all distributions of X_1, X_2 as well as sketch an optimized analysis for strongly unbalanced X_1, X_2. Section 4 generalizes the analysis to an arbitrary number n of variables. Here, we also establish the $\ln(2) \approx 70\%$ result for *factoring with known bits*. We experimentally verify the underlying heuristic in Section 5.

2 Preliminaries

Let $b_1, \ldots, b_k$ be linearly independent vectors in $\mathbb{R}^n$. Then the *lattice* spanned by $b_1, \ldots, b_k$ is the set of all integer linear combinations of $b_1, \ldots, b_k$. We call $b_1, \ldots, b_k$ a basis of L. The integer k is called the *dimension* or *rank* of the lattice and we say that the lattice has *full rank* if $k = n$.

Every nontrivial lattice in $\mathbb{R}^n$ has infinitely many bases, therefore we seek for *good* ones. The most important quality measure is the length of the basis vectors which corresponds to the basis vectors' orthogonality. A famous theorem of Minkowski [Min10] relates the length of the shortest vector in a lattice to the determinant:

Theorem 1 (Minkowski) *In an ω-dimensional lattice, there exists a non-zero vector v with*

$$\|v\| \leq \sqrt{\omega} \det(L)^{\frac{1}{\omega}}. \tag{1}$$

In lattices with fixed dimension we can efficiently find a shortest vector, but for arbitrary dimensions, the problem of computing a shortest vector is known to be NP-hard under randomized reductions [Ajt98]. The L^3 algorithm, however, computes in polynomial time an approximation of the shortest vector, which is sufficient for many applications. The basis vectors of an L^3-reduced basis fulfill the following property (for a proof see e.g. [May03]).

Theorem 2 (L^3) *Let L be an integer lattice of dimension ω. The L^3 algorithm outputs a reduced basis spanned by $\{v_1 \ldots, v_\omega\}$ with*

$$\|v_1\| \leq \|v_2\| \leq \ldots \leq \|v_i\| \leq 2^{\frac{\omega(\omega-i)}{4(\omega+1-i)}} \det(L)^{\frac{1}{\omega+1-i}}, \quad i = 1, \ldots, \omega \tag{2}$$

in polynomial time.

The underlying idea of Coppersmith's method for finding small roots of polynomial equations is to reduce the problem of finding roots of $f(x_1, \ldots, x_n) \bmod p$ to finding roots over the integers. Therefore, one constructs a collection of polynomials that share a common root modulo p^m for some well-chosen integer m. Then one finds an integer linear combination which has a sufficiently small norm. The search for such a small norm linear combination is done by defining a lattice basis via the polynomials' coefficient vectors. An application of L^3 yields a small norm coefficient vector that corresponds to a small norm polynomial.

The following lemma due to Howgrave-Graham [HG97] gives a sufficient condition under which modular roots are also roots over $\mathbb{Z}$ and quantifies the term *sufficiently small*.

Lemma 1 *Let $g(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ be an integer polynomial with at most ω monomials. Suppose that*

1. $g(y_1, \ldots, y_n) = 0 \bmod p^m$ *for* $|y_1| \leq X_1, \ldots, |y_n| \leq X_n$ *and*
2. $\|g(x_1X_1, \ldots, x_nX_n)\| < \frac{p^m}{\sqrt{\omega}}$

Then $g(y_1, \ldots, y_n) = 0$ holds over the integers.

Our approach relies on heuristic assumptions for computations with multivariate polynomials.

Assumption 1 *Our lattice-based construction yields algebraically independent polynomials. The common roots of these polynomials can be efficiently computed using numerical methods.*

The first part of Assumption 1 assures that the constructed polynomials allow for extracting the common roots, while the second part assures that we are able to compute these common roots efficiently. We would like to point out that our subsequent complexity considerations solely refer to our lattice-based construction, that turns a linear polynomial $f(x_1, \ldots, x_n) \bmod p$ into n polynomials over the integers. We assume that the running time for extracting the desired root out of these n polynomials is negligible compared to the time complexity of the lattice construction. We verify this experimentally in Section 5. Usually, our method yields more than n polynomials, so one can make use of additional polynomials as well.

3 Bivariate Linear Equations

The starting point of our analysis are bivariate linear modular equations $f(x_1, x_2) = a_1x_1 + a_2x_2 + a_3 \bmod p$. The parameter p is unknown, we only know a multiple N of p, and the parameter β that quantifies the size relation $p \geq N^\beta$. Let X_1, X_2 be upper bounds on the desired solution y_1, y_2, respectively. Moreover, we require that our linear polynomial is monic with respect to one of the variables, i.e. either $a_1 = 1$ or $a_2 = 1$. This is usually not a restriction, since we could e.g. multiply $f(x_1, x_2)$ by $a_1^{-1} \bmod N$. If this inverse does not exist, we can factorize N.

In the following theorem, we give an explicit bound on X_1X_2 under which we can find two polynomials $g_1(x_1, x_2)$ and $g_2(x_1, x_2)$ that evaluate to zero at all small points (y_1, y_2) with $|y_1y_2| \leq X_1X_2$. Under the heuristic that g_1 and g_2 are *algebraically independent*, all roots smaller than X_1X_2 can be recovered by standard methods over the integers.

Theorem 3 *Let $\epsilon > 0$ and let N be a sufficiently large composite integer with a divisor $p \geq N^\beta$. Furthermore, let $f(x_1, x_2) \in \mathbb{Z}[x_1, x_2]$ be a linear polynomial in two variables. Under Assumption 1, we can find all solutions (y_1, y_2) of the equation $f(x_1, x_2) = 0 \bmod p$ with $|y_1| \leq N^\gamma$ and $|y_2| \leq N^\delta$ if*

$$\gamma + \delta \leq 3\beta - 2 + 2(1-\beta)^{\frac{3}{2}} - \epsilon \tag{3}$$

The algorithm's time and space complexity is polynomial in $\log N$ and ϵ^{-1}.

Before we provide a proof for Theorem 3, we would like to interpret its implications. Notice that Theorem 3 yields in the special case $\beta = 1$ the bound $X_1X_2 \leq N^{1-\epsilon}$ that corresponds to the folklore bound for linear equations. Since we are unaware of a good reference for the folklore method in the cryptographic literature, we briefly sketch the derivation of this bound in Appendix A. Thus, our result generalizes the folklore method to more general moduli.

On the other hand, we would like to compare our result with the one of Coppersmith for *factoring with high bits known* when p, q are of equal bit-size, i.e. $\beta = \frac{1}{2}$. Coppersmith's result allows a maximal size of $N^{0.25}$ for one unknown block. Our result states a bound of $N^{0.207}$ for the product of two blocks. The best that we could hope for was to obtain a total of $N^{0.25}$ for two blocks as well. However, it seems quite natural that the bound decreases with the number n of blocks. On the other hand, we are able to show that if the unknown blocks are significantly unbalanced in size, then one can improve on the bound $N^{0.207}$. It turns out that the more unbalanced X_1, X_2 are, the better. In the extreme case, we obtain $X_1 = N^{0.25}, X_2 = 1$. Notice that in this case, the variable x_2 vanishes and we indeed obtain the univariate result $N^{0.25}$ of Coppersmith. Hence, our method contains the Coppersmith-bound as a special case as well. We give more details after the following proof of Theorem 3.

Proof. Define $X_1X_2 := N^{3\beta-2+2(1-\beta)^{\frac{3}{2}}-\epsilon}$ and fix $m = \left\lceil \frac{3\beta\left(1+\sqrt{1-\beta}\right)}{\epsilon} \right\rceil$.

We define a collection of polynomials which share a common root modulo p^t by

$$g_{k,i}(x_1, x_2) := x_2^i f^k(x_1, x_2) N^{\max\{t-k,0\}} \tag{4}$$

for $k = 0, ..., m; i = 0, ..., m-k$ and some $t = \tau m$, that will be optimized later.

We can define the following polynomial ordering for our collection. Let $g_{k,i}, g_{l,j}$ be two polynomials. If $k < l$ then $g_{k,i} < g_{l,j}$, if $k = l$ then $g_{k,i} < g_{l,j} \Leftrightarrow i < j$. If we sort the polynomials according to that ordering, every subsequent polynomial in the ordering introduces exactly one new monomial. Thus, the corresponding coefficient vectors define a lower triangular lattice basis, like in Figure 1.

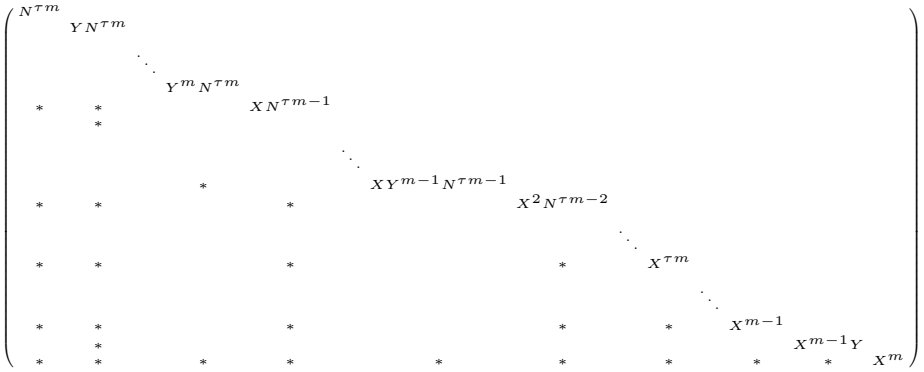

Fig. 1. Basis Matrix in Triangular Form

From the basis matrix we can easily compute the determinant as the product of the entries on the diagonal as $\det(L) = X^{s_x} Y^{s_y} N^{s_N}$, where

$$s_x = s_y = \frac{1}{6}(m^3 + 3m^2 + 2m), \quad s_N = \sum_{i=0}^{\tau m} (m + 1 - i)(\tau m - i) \tag{5}$$

Now we apply L^3 basis reduction to the lattice basis. Our goal is to find two coefficient vectors whose corresponding polynomials contain all small roots over the integer. Theorem 2 gives us an upper bound on the norm of a second-to-shortest vector in the L^3-reduced basis. If this bound is in turn smaller than the bound in Howgrave-Graham's lemma (Lemma 1), we obtain the desired two polynomials. I.e., we have to satisfy the condition

$$2^{\frac{d(d-1)}{4(d-1)}} \det(L)^{\frac{1}{d-1}} < d^{-\frac{1}{2}} N^{\beta\tau m}, \tag{6}$$

where d is the dimension of the lattice L, which in our case is $d = \frac{1}{2}(m^2+3m+2)$. If we plug in the value for the determinant and use the fact that $s_x = \frac{md}{3}$, we obtain the condition

$$X_1 X_2 < 2^{-\frac{3(d-1)}{4m}} d^{-\frac{3(d-1)}{2md}} N^{\frac{3\beta\tau m(d-1)}{md} - \frac{3s_N}{md}}. \tag{7}$$

Setting $\tau = 1 - \sqrt{1-\beta}$, the exponent of N can be lower bounded by

$$3\beta - 2 + 2(1-\beta)^{\frac{3}{2}} - \frac{3\beta\left(1 + \sqrt{1-\beta}\right)}{m}. \tag{8}$$

[Details can be found in Appendix B.]

Comparing this with the value of $X_1 X_2$, which we defined in the beginning, we can express how m depends on the error term ϵ:

$$m \geq \frac{3\beta\left(1 + \sqrt{1-\beta}\right)}{\epsilon}. \tag{9}$$

which holds for our choice of m. Therefore, the required condition is fulfilled.

It remains to show that the algorithm's complexity is polynomial in $\log(N)$ and ϵ^{-1}. The running time is dominated by L^3 reduction, which is polynomial in the dimension of the lattice and in the bitsize of the entries. Recall that our lattice's dimension is $\mathcal{O}(m^2)$ and therefore polynomial in ϵ^{-1}. For the matrix entries we notice that the power f^k in the $g_{k,i}$'s can be reduced modulo N^k, since we are looking for roots modulo N^k. Thus, the coefficients of $f^k N^{max(\tau m-k,0)}$ have bitsize $\mathcal{O}(m \log(N))$. Powers of X_2 appear only with exponents up to m and therefore their bitsize can also be upper bounded by $\mathcal{O}(m \log(N))$. Thus, the coefficients' bitsize is $\mathcal{O}(\epsilon^{-1} \log(N))$.

Remark: We also analyzed the bivariate modular instance as a trivariate equation over the integers, which is modelled by

$$(a_1 x_1 + a_2 x_2 + a_3) y - N = 0, \tag{10}$$

where y stands for $\frac{N}{p}$. It turns out that we obtain the same bounds as in the modular case.

Theorem 3 holds for any bounds X_1, X_2 within the proven bound for the product $X_1 X_2$. As pointed out before, the analysis can be improved if one of the bounds is significantly smaller than the other one, say $X_1 \ll X_2$. Then one should employ additional extra shifts in the smaller variable, which intuitively means that the smaller variable gets stronger weight since it causes smaller costs.

We do not give the exact formulas for this optimization process. Instead, we show in Figure 2 the resulting graph that demonstrates how the result converges to the known bound $N^{0.25}$ for unbalanced block-sizes.

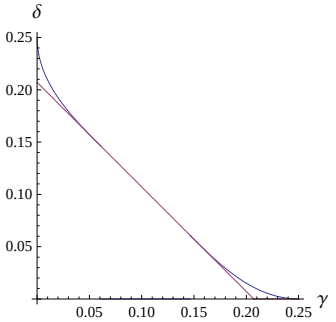

Fig. 2. Optimized Result

Notice that the result from Theorem 3 is indeed optimal not only for equal block-sizes $X_1 = X_2$ but for most of the possible splittings of block-sizes. Only in extreme cases a better result can be achieved. In the subsequent chapter, we generalize Theorem 3 to an arbitrary number n of blocks. In the generalization however, we will not consider the improvement that can be achieved for strongly unbalanced block-sizes.

Naturally, the bounds $N^{0.25}$ for $n = 1$ and $N^{0.207}$ for $n = 2$ get worse for arbitrary n. But surprisingly, we will show that for $n \to \infty$ the bound does not converge to N^0 as one might expect, but instead to $N^{0.153}$. To illustrate this result: If N is a 1000-bit modulus and p, q are 500 bit each. Then 153 bit can be recovered given the remaining 347 bits, or 69.4% of p, in any *known* positions. However as we will see in the next section, the complexity heavily depends on the number of unknown blocks.

4 Extension to More Variables

In this section, we generalize the result of Section 3 from bivariate linear equations with $n = 2$ to an arbitrary number n of variables $x_1, \ldots, x_n$.

Let $X_1, X_2, \ldots, X_n$ be upper bounds for the variables $x_1, x_2, \ldots, x_n$. As in Theorem 3, we will focus on proving a general upper bound for the product $X_1 X_2 \ldots X_n$ that is valid for any $X_1, X_2, \ldots, X_n$. Similar to the reasoning in Section 3 it is possible to achieve better results for strongly unbalanced X_i by giving more weight to variables x_i with small upper bounds. Although we did not analyze it, we strongly expect that in the case $X_1 = N^{0.25}$, $X_2 = \cdots = X_n = 1$ everything boils down to the univariate case analyzed by Coppersmith/Howgrave-Graham – except that we obtain an unnecessarily large lattice dimension.

Naturally, we achieve an inferior bound than $N^{0.25}$. But in contrast, our bound holds no matter how the sizes of the unknowns are distributed among the upper bounds X_i. Let us state our main theorem.

Theorem 4 *Let $\epsilon > 0$ and let N be a sufficiently large composite integer with a divisor $p \geq N^\beta$. Furthermore, let $f(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ be a monic linear polynomial in n variables. Under Assumption 1, we can find all solutions $(y_1, \ldots, y_n)$ of the equation $f(x_1, \ldots, x_n) = 0 \bmod p$ with $|y_1| \leq N^{\gamma_1}, \ldots, |y_n| \leq N^{\gamma_n}$ if*

$$\sum_i^n \gamma_i \leq 1 - (1-\beta)^{\frac{n+1}{n}} - (n+1)(1 - \sqrt[n]{1-\beta})(1-\beta) - \epsilon \tag{11}$$

The time and space complexity of the algorithm is polynomial in $\log N$ and $(\frac{e}{\epsilon})^n$, where e is Euler's constant.

We will prove Theorem 4 at the end of this section. Let us first discuss the implications of the result and the consequences for the *factoring with known bits problem*. First of all, the algorithm's running time is exponential in the number n of blocks. Thus in order to obtain a polynomial complexity one has to restrict

$$n = \mathcal{O}\left(\frac{\log\log N}{1 + \log(\frac{1}{\epsilon})}\right).$$

This implies that for any constant error term ϵ, our algorithm is polynomial time whenever $n = \mathcal{O}(\log\log N)$.

The proof of the following theorem shows that the bound for $X_1 \ldots X_n$ in Theorem 4 converges for $n \to \infty$ to $N^{\beta + (1-\beta)\ln(1-\beta)}$. For the *factoring with known bits* problem with $\beta = \frac{1}{2}$ this yields the bound $N^{\frac{1}{2}(1-\ln(2))} \approx N^{0.153}$. This means that we can recover a $(1 - \ln(2)) \approx 0.306$-fraction of the bits of p, or in other words an $\ln(2) \approx 0.694$-fraction of the bits of p has to be known.

Theorem 5 *Let $\epsilon > 0$. Suppose N is a sufficiently large composite integer with a divisor $p \geq N^\beta$. Further, suppose we are given an*

$$\left(1 - \frac{1}{\beta}\right) \cdot \ln(1-\beta) + \epsilon \quad \textit{fraction} \tag{12}$$

of the bits of p. Then, under Assumption 1, we can compute the unknown bits of p in time polynomial in $\log N$ and $(\frac{e}{\epsilon})^n$, where e is Euler's constant.

Proof. From Theorem 4 we know, that we can compute a solution to the equation

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_{n+1} = 0 \bmod p$$

as long as the product of the unknowns is smaller than N^γ, where $\gamma = \sum_i^n \gamma_i$ is upper-bounded as in Inequality (11). As noticed already, the bound for γ actually converges for $n \to \infty$ to a value different from zero. Namely,

$$\lim_{n\to\infty} \left(1 - (1-\beta)^{\frac{n+1}{n}} - (n+1)(1 - \sqrt[n]{1-\beta})(1-\beta)\right) = \beta + (1-\beta)\ln(1-\beta) \tag{13}$$

Hence, this is the portion of p we can at least compute, no matter how many unknowns we have.

Conversely, once we have $((\beta-1)\ln(1-\beta)+\epsilon)\log(N)$ bits of p given together with their positions, we are able to compute the missing ones. Since $\log N \leq \frac{\log p}{\beta}$, we need at most an $((1-\frac{1}{\beta})\ln(1-\beta)+\epsilon)$-fraction of the bits of p.

Theorem 5 implies a polynomial-time algorithm for the *factoring with known bits* problem whenever the number of unknown bit-blocks is $n = \mathcal{O}(\log\log N)$. However, the algorithm can be applied for larger n as well. As long as n is sub-polynomial in the bit-size of N, the resulting complexity will be sub-exponential in the bit-size of N.
It remains to prove our main theorem.

Proof of Theorem 4:
Define $\prod_{i=1}^n X_i := N^{1-(1-\beta)^{\frac{n+1}{n}} - (n+1)(1-\sqrt[n]{1-\beta})(1-\beta)-\epsilon}$. Let us fix

$$m = \left\lceil \frac{n(\frac{1}{\pi}(1-\beta)^{-0.278465} - \beta\ln(1-\beta))}{\epsilon} \right\rceil \tag{14}$$

We define the following collection of polynomials which share a common root modulo p^t

$$g_{i_2,\ldots,i_n,k} = x_2^{i_2}\ldots x_n^{i_n} f^k N^{\max\{t-k,0\}} \tag{15}$$

where $i_j \in \{0,\ldots,m\}$ such that $\sum_{j=2}^n i_j \leq m-k$. The parameter $t = \tau m$ has to be optimized. Notice that the set of monomials of $g_{i_2,\ldots,i_n,k}$ defines an n-dimensional simplex.

It is not hard to see that there is an ordering of the polynomials in such a way that each new polynomial introduces exactly one new monomial. Therefore the lattice basis constructed from the coefficient vectors of the $g_{i_2,\ldots,i_n,k}$'s has triangular form, if they are sorted according to the order. The determinant $\det(L)$ of the corresponding lattice L is then simply the product of the entries on the diagonal:

$$\det(L) = \prod_{i=1}^n X_i^{s_{x_i}} N^{s_N}, \tag{16}$$

with $s_{x_i} = \binom{m+n}{m-1}$ and $s_N = md\tau - \binom{m+n}{m-1} + \binom{m(1-\tau)+n}{m(1-\tau)-1}$, where $d = \binom{m+n}{m}$ is the dimension of the lattice.

Now we ensure that the vectors from L^3 are sufficiently small, so that we can apply the Lemma of Howgrave-Graham (Lemma 1) to obtain a solution over $\mathbb{Z}$. We have to satisfy the condition

$$2^{\frac{d(d-1)}{4(d-n+1)}} \det(L)^{\frac{1}{d-n+1}} < d^{-\frac{1}{2}} N^{\beta\tau m}$$

Using the value of the determinant in (16) and the fact that $s_{x_i} = \frac{md}{n+1}$ we obtain

$$\prod_{i=1}^{n} X_i \leq 2^{-\frac{(d-1)(n+1)}{4m}} d^{-\frac{(n+1)(d-n+1)}{2md}} N^{(\beta m\tau(d-n+1) - dm\tau + \binom{m+n}{m-1} - \binom{m(1-\tau)+n}{m(1-\tau)-1})\frac{n+1}{md}}.$$

In Appendix C we show how to derive a lower bound on the right-hand side for the optimal value $\tau = 1 - (1-\beta)^{\frac{1}{n}}$. Using $X_i = N^{\gamma_i}$ the condition reduces to

$$\sum_{i=1}^{n} \gamma_i \leq 1 - (1-\beta)^{\frac{n+1}{n}} - (n+1)(1 - \sqrt[n]{1-\beta})(1-\beta) - \frac{n\frac{1}{\pi}(1-\beta)^{-0.278465}}{m} + \beta \ln(1-\beta)\frac{n}{m}.$$

Comparing this to the initial definition of $\prod_{i=1}^{n} X_i$, we obtain for the error term ϵ

$$\begin{aligned} -\frac{n\frac{1}{\pi}(1-\beta)^{-0.278465}}{m} + \beta\ln(1-\beta)\frac{n}{m} &\geq -\epsilon \\ \Leftrightarrow m \geq \frac{n(\frac{1}{\pi}(1-\beta)^{-0.278465} - \beta\ln(1-\beta))}{\epsilon} &= \mathcal{O}(\frac{n}{\epsilon}) \end{aligned}$$

which holds for our choice of m.

To conclude the proof, we notice that the dimension of the lattice is $d = \mathcal{O}(\frac{m^n}{n!}) = \mathcal{O}(\frac{n^n e^n}{\epsilon^n n^n}) = \mathcal{O}(\frac{e^n}{\epsilon^n})$. For the bitsize of the entries in the basis matrix we observe that we can reduce the coefficients of f^i in g modulo N^i. Thus the product $f^k N^{\max\{\tau m - k, 0\}}$ is upper bounded by $B = m\log(N)$. Further notice that the bitsize of $X_2^{i_2} \ldots X_n^{i_2}$ is also upper bounded by $m\log(N)$ since $\sum_{i=2}^{n} i_j \leq m$ and $X_i \leq N$.

The running time is dominated by the time to run L^3-lattice reduction on a basis matrix of dimension d and bit-size B. Thus, the time and space complexity of our algorithm is polynomial in $\log N$ and $(\frac{e}{\epsilon})^n$. □

5 Experimental Results

We implemented our lattice-based algorithm using the L^2-algorithm from Nguyen, Stehlé [NS05]. We tested the algorithm for instances of the *factoring with known bits* problem with $n = 2, 3$ and 4 blocks of unknown bits. Table 5 shows the experimental results for an 512-bit RSA modulus N with divisor p of size $p \geq N^{\frac{1}{2}}$.

For given parameters m, t we computed the number of bits that one should theoretically be able to recover from p (column *pred* of Table 5). For each bound

we made two experiments (column *exp*). The first experiment splits the bound into n equally sized pieces, whereas the second experiment unbalancedly splits the bound in one large piece and $n-1$ small ones. In the unbalanced case, we were able to recover a larger number of bits than theoretically predicted. This is consistent with the reasoning in Section 3 and 4.

n	m	t	$\dim(L)$	pred (bit)	exp (bit)	time (min)
2	15	4	136	90	45/45	25
2	15	4	136	90	87/5	15
3	7	1	120	56	19/19/19	0.3
3	7	1	120	56	52/5/5	0.3
3	10	2	286	69	23/23/23	450
3	10	2	286	69	57/6/6	580
4	5	1	126	22	7/6/6/6	3
4	5	1	126	22	22/2/2/2	4.5

Table 1. Experimental Results

In all of our experiments, we successfully recovered the desired small root, thereby deriving the factorization of N. We were able to extract the root both by Gröbner basis reduction as well as by numerical methods in a fraction of a second.

For Gröbner basis computations, it turns out to be useful that our algorithm actually outputs more sufficiently small norm polynomials than predicted by the L^3-bounds. This in turn helps to speed up the computation a lot.

As a numerical method, we used multidimensional Newton iteration on the starting point $\frac{1}{2}(X_1, \ldots, X_n)$. Usually this did already work. If not, we were successful with the vector of upper-bounds $(X_1, \ldots, X_n)$ as a starting point. Although this approach worked well and highly efficient in practice, we are unaware of a starting point that provably lets the Newton method converge to the desired root.

Though Assumption 1 worked perfectly for the described experiments, we also considered two pathological cases, where one has to take special care.

First, a problem arises when we have a prediction of k bits that can be recovered, but we use a much smaller sum of bits in our n blocks. In this case, the smallest vector lies in a sublattice of small dimension. As a consequence, we discovered that then usually all of our small norm polynomials shared $f(x)$ as a common divisor. When we removed the gcd, the polynomials were again algebraically independent and we were able to retrieve the root. Notice that removing $f(x)$ does not eliminate the desired root, since $f(x)$ does not contain the root over the integers (but $\bmod p$).

A second problem may arise in the case of two closely adjacent unknown blocks, e.g. two blocks that are separated by one known bit only. Since in comparison with the n-block case the case of $n-1$ blocks gives a superior bound,

it turns out to be better in some cases to merge two closely adjacent blocks into one variable. That is what implicitly seems to happen in our approach. The computations then yield the desired root only in those variables which are sufficiently separated. The others have to be merged before re-running the algorithm in order to obtain all the unknown bits. Alternatively, we confirmed experimentally that merging the nearby blocks from the beginning immediately yields the desired root.

Both pathological cases are no failure of Assumption 1, since one can still easily extract the desired root. All that one has to do is to either remove a gcd or to merge variables.

6 Conclusion and Open Problems

We proposed a heuristic lattice-based algorithm for finding small solutions of linear equations $a_1x_1 + \cdots + a_nx_n + a_{n+1} = 0 \bmod p$, where p is an unknown divisor of some known N. Our algorithm gives a solution for the *factoring with known bits* problem given $\ln(2) \approx 70\%$ of the bits of p in any locations.

Since the time and space complexity of our algorithm is polynomial in $\log N$ but exponential in the number n of variables, we obtain a polynomial time algorithm for $n = \mathcal{O}(\log \log N)$ and a subexponential time algorithm for $n = o(\log N)$. This naturally raises the question whether there exists some algorithm with the same bound having complexity polynomial in n. This would immediately yield a polynomial time algorithm for *factoring with* 70% *bits given*, independently of the given bit locations *and* the number of consecutive bit blocks. We do not know whether such an algorithm can be achieved for polynomial equations with *unknown divisor*. On the other hand, we feel that the complexity gap between the folklore method for *known divisors* with complexity linear in n and our method is quite large, even though the folklore method relies on much stronger assumptions.

Notice that in the *factoring with known bits* problem, an attacker is given the location of the given bits of p and he has to fill in the missing bits. Let us give a crude analogy for this from coding theory, where one is given the codeword p with erasures in some locations. Notice that our algorithm is able to correct the erasures with the help of the redundancy given by N. Now a challenging question is whether there exist similar algorithms for *error-correction* of codewords p. I.e., one is given p with a certain percentage of the bits flipped. Having an algorithm for this problem would be highly interesting in situations with error-prone side-channels.

We would like to thank the anonymous reviewers and especially Robert Israel for helpful comments and ideas.

References

[Ajt98] Miklós Ajtai. The Shortest Vector Problem in L_2 is *NP*-hard for Randomized Reductions (Extended Abstract). In *STOC*, pages 10–19, 1998.

[BD00] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):1339, 2000.

[BM06] Daniel Bleichenbacher and Alexander May. New Attacks on RSA with Small Secret CRT-Exponents. In *Public Key Cryptography*, pages 1–13, 2006.

[CJL+92] Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved Low-Density Subset Sum Algorithms. *Computational Complexity*, 2:111–128, 1992.

[CM07] Jean-Sébastien Coron and Alexander May. Deterministic Polynomial-Time Equivalence of Computing the RSA Secret Key and Factoring. *J. Cryptology*, 20(1):39–50, 2007.

[Cop96a] Don Coppersmith. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In *EUROCRYPT*, pages 178–189, 1996.

[Cop96b] Don Coppersmith. Finding a Small Root of a Univariate Modular Equation. In *EUROCRYPT*, pages 155–165, 1996.

[Cop97] Don Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997.

[Cor07] Jean-Sébastien Coron. Finding Small Roots of Bivariate Integer Polynomial Equations: A Direct Approach. In *CRYPTO*, pages 379–394, 2007.

[GM97] Marc Girault and Jean-François Misarsky. Selective Forgery of RSA Signatures Using Redundancy. In *EUROCRYPT*, pages 495–507, 1997.

[GTV88] Marc Girault, Philippe Toffin, and Brigitte Vallée. Computation of Approximate L-th Roots Modulo n and Application to Cryptography. In *CRYPTO*, pages 100–117, 1988.

[Has88] Johan Hastad. Solving Simultaneous Modular Equations of Low Degree. *SIAM Journal on Computing*, 17(2):336–341, 1988.

[HG97] Nick Howgrave-Graham. Finding Small Roots of Univariate Modular Equations Revisited. *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, pages 131–142, 1997.

[HG01] Nick Howgrave-Graham. Approximate Integer Common Divisors. In *CaLC*, pages 51–66, 2001.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and László Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[Mau95] Ueli M. Maurer. On the Oracle Complexity of Factoring Integers. *Computational Complexity*, 5(3/4):237–247, 1995.

[May03] Alexander May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD thesis, University of Paderborn, 2003.

[May04] Alexander May. Computing the RSA Secret Key Is Deterministic Polynomial Time Equivalent to Factoring. In *CRYPTO*, pages 213–219, 2004.

[Min10] Herrmann Minkowski. *Geometrie der Zahlen*. Teubner, 1910.

[Ngu04] Phong Q. Nguyen. Can We Trust Cryptographic Software? Cryptographic Flaws in GNU Privacy Guard v1.2.3. In *EUROCRYPT*, pages 555–570, 2004.

[NS01] Phong Q. Nguyen and Jacques Stern. The Two Faces of Lattices in Cryptology. In *CaLC*, pages 146–180, 2001.

[NS05] Phong Q. Nguyen and Damien Stehlé. Floating-Point LLL Revisited. In *EUROCRYPT*, pages 215–233, 2005.

[RS85] Ronald L. Rivest and Adi Shamir. Efficient Factoring Based on Partial Information. In *EUROCRYPT*, pages 31–34, 1985.

A Linear Equations with Known Modulus

We briefly sketch the folklore method for finding small roots of linear modular equations $a_1x_1 + \cdots + a_nx_n = 0 \bmod N$ with *known* modulus N. Further, we assume that $\gcd(a_i, N) = 1$ for some i, wlog $\gcd(a_n, N) = 1$. Let X_i be upper bounds on $|y_i|$. We can handle inhomogeneous modular equations by introducing a term $a_{n+1}x_{n+1}$, where $|y_{n+1}| \leq X_{n+1} = 1$.

We would like to point out that the heuristic for the folklore method is quite different compared to the one taken in our approach. First of all, the method requires to solve a shortest vector problem in a certain lattice. This problem is known to be NP-hard for general lattices. Second, one assumes that there is only *one* linear independent vector that fulfills the Minkowski bound (Theorem 1) for the shortest vector.

We will show under this heuristic assumption that the shortest vector yields the unique solution $(y_1, \ldots, y_n)$ whenever

$$\prod_{i=1}^{n} X_i \leq N.$$

We multiply our linear equation with $-a_n^{-1}$ and obtain

$$b_1x_1 + b_2x_2 + \ldots + b_{n-1}x_{n-1} = x_n \bmod N \quad \text{,where } b_i = a_n^{-1}a_i \tag{17}$$

For a solution $(y_1, \ldots, y_n)$ of (17) we know $\sum_{i=1}^{n-1} b_iy_i = y_n - yN$ for some $y \in \mathbb{Z}$. Consider the lattice L generated by the row vectors of the following matrix

$$B = \begin{pmatrix} Y_1 & 0 & 0 & \ldots & 0 & Y_nb_1 \\ 0 & Y_2 & 0 & & 0 & Y_nb_2 \\ \vdots & & \ddots & & \vdots & \vdots \\ \vdots & & & & Y_{n-1} & Y_nb_{n-1} \\ 0 & 0 & 0 & \ldots & 0 & Y_nN \end{pmatrix}$$

with $Y_i = \frac{N}{X_i}$. By construction,

$$v = (y_1, \ldots, y_{n-1}, y) \cdot B = (Y_1y_1, \ldots, Y_ny_n)$$

is a vector of L. We show, that this is a short vector which fulfills the Minkowski bound from Theorem 1. If we assume that v is actually the shortest vector, then we can solve an SVP instance.

Since $Y_iy_i = \frac{y_i}{X_i}N \leq N$ we have $\|v\| \leq \sqrt{n}N$. Further, the determinant of the lattice L is

$$\det(L) = N\prod_{i=1}^{n} Y_i = N\prod_{i=1}^{n} \frac{N}{X_i} = N^{n+1}\prod_{i=1}^{n} \frac{1}{X_i}.$$

The vector v thus fulfills the Minkowski bound, if

$$\sqrt{n}N \leq \sqrt{n}\det(L)^{\frac{1}{n}} \quad \Leftrightarrow \quad \prod_{i=1}^{n} X_i \leq N.$$

B Lower Bound in Theorem 2

Starting with

$$X_1 X_2 < 2^{-\frac{3(d-1)}{4m}} d^{-\frac{3(d-1)}{2md}} N^{\frac{3\beta\tau m(d-1)}{md} - \frac{3s_N}{md}}$$

we wish to derive a lower bound of the right-hand side. First we notice that for sufficiently large N the powers of 2 and d are negligible. Thus, we only examine the exponent of N. We use the values $d = \frac{1}{2}(m^2+3m+2)$ and $s_N = \sum_{i=0}^{\tau m}(m+1-i)(\tau m - i)$ and get

$$\tau\left(3\beta - 3\tau + \tau^2\right) + \frac{-\tau - 6\beta\tau + \tau^3}{1+m} - \frac{2\left(\tau - 3\beta\tau - 3\tau^2 + 2\tau^3\right)}{2+m}.$$

For τ we choose $1-\sqrt{(1-\beta)}$, resulting in

$$\begin{aligned}&-2 + 2\sqrt{1-\beta} + 3\beta - 2\beta\sqrt{1-\beta}\\ &\quad - \frac{3\sqrt{1-\beta}}{1+m} + \frac{6\sqrt{1-\beta}}{2+m} + \frac{7\beta\sqrt{1-\beta}}{1+m} - \frac{10\beta\sqrt{1-\beta}}{2+m} + \frac{6(-1+2\beta)}{2+m} - \frac{3(-1+3\beta)}{1+m}.\end{aligned}$$

Now we combine the terms that change their sign in the possible β-range, such that we obtain a term which is either positive or negative for all $\beta \in (0,1)$

$$\begin{aligned}-\frac{3\sqrt{1-\beta}}{1+m} - \frac{3(-1+3\beta)}{1+m} + \frac{7\beta\sqrt{1-\beta}}{1+m} &= \frac{3 - 3\sqrt{1-\beta} - 9\beta + 7\beta\sqrt{1-\beta}}{1+m} < 0\\ \frac{6(-1+2\beta)}{2+m} + \frac{6\sqrt{1-\beta}}{2+m} &= \frac{6\left(-1+\sqrt{1-\beta}+2\beta\right)}{2+m} > 0 \text{ for all } \beta \in (0,1).\end{aligned}$$

Finally, we approximate the positive terms by $\frac{*}{2m}$ and the negative ones by $\frac{*}{m}$ and obtain

$$2^{-\frac{3(d-1)}{4m}} d^{-\frac{3(d-1)}{2md}} N^{\frac{3\beta\tau m(d-1)}{md} - \frac{3s_N}{md}} \geq N^{-2+2\sqrt{1-\beta}+3\beta-2\beta\sqrt{1-\beta} - \frac{3\beta\left(1+\sqrt{1-\beta}\right)}{m}}. \quad (18)$$

C Lower Bound in Theorem 3

We derive a lower bound of

$$2^{-\frac{(d-1)(n+1)}{4m}} d^{-\frac{(n+1)(d-n+1)}{2md}} N^{\left(\beta m\tau(d-n+1) - dm\tau + \binom{m+n}{m-1} - \binom{m(1-\tau)+n}{m(1-\tau)-1}\right)\frac{n+1}{md}}.$$

For sufficiently large N, the powers of 2 and d are negligible and thus we consider in the following only the exponent of N

$$\begin{aligned}&\left(\beta m\tau(d-n+1) - dm\tau + \binom{m+n}{m-1} - \binom{m(1-\tau)+n}{m(1-\tau)-1}\right)\frac{n+1}{md}\\ &= \beta\tau(n+1) - \frac{\beta\tau(n-1)(n+1)}{d} - \tau(n+1) + 1 - \frac{\prod_{k=0}^{n}(m(1-\tau)+k)}{n!md}.\end{aligned}$$

With $d = \binom{m+n}{m} = \frac{(m+n)!}{m!n!} = \frac{\prod_{k=1}^{n}(m+k)}{n!}$ we have

$$\beta\tau(n+1) - \tau(n+1) + 1 - \frac{\beta\tau(n-1)(n+1)!}{\prod_{k=1}^{n}(m+k)} - \frac{\prod_{k=0}^{n}(m(1-\tau)+k)}{\prod_{k=0}^{n}(m+k)}.$$

We now analyze the last two terms separately. For the first one, if we choose $\tau = 1 - \sqrt[n]{(1-\beta)}$ we obtain

$$\frac{\beta\tau(n-1)(n+1)!}{\prod_{k=1}^{n}(m+k)} \leq \frac{\beta(1-\sqrt[n]{1-\beta})(n-1)(n+1)!}{(m+1)\prod_{k=2}^{n}k} \leq \frac{\beta(1-\sqrt[n]{1-\beta})n^2}{m}.$$

Fact 1

$$n(1-\sqrt[n]{1-\beta}) \leq -\ln(1-\beta) \tag{19}$$

Using this approximation, we obtain

$$\frac{\beta\tau(n-1)(n+1)!}{\prod_{k=1}^{n}(m+k)} \leq -\ln(1-\beta)\beta\frac{n}{m}.$$

The analysis of the second term $\frac{\prod_{k=0}^{n}(m(1-\tau)+k)}{\prod_{k=0}^{n}(m+k)}$ is a bit more involved. We use its partial fraction expansion to show an upper bound.

Lemma 2 *For $\tau = 1-(1-\beta)^{\frac{1}{n}}$ we have*

$$\frac{\prod_{k=0}^{n}(m(1-\tau)+k)}{\prod_{k=0}^{n}(m+k)} \leq (1-\beta)^{\frac{n+1}{n}} + \frac{1}{\pi}(1-\beta)^{-0.278465}\frac{n}{m}. \tag{20}$$

Proof. First notice that

$$\frac{\prod_{k=0}^{n}(m(1-\tau)+k)}{\prod_{k=0}^{n}(m+k)} = (1-\tau)^{n+1} + \frac{\prod_{k=0}^{n}(m(1-\tau)+k) - (1-\tau)^{n+1}\prod_{k=0}^{n}(m+k)}{\prod_{k=0}^{n}(m+k)}.$$

We analyze the second part of this sum. Its partial fraction expansion is

$$\frac{\prod_{k=0}^{n}(m(1-\tau)+k) - (1-\tau)^{n+1}\prod_{k=0}^{n}(m+k)}{\prod_{k=0}^{n}(m+k)} = \frac{c_0}{m} + \frac{c_1}{m+1} + \ldots + \frac{c_n}{m+n}. \tag{21}$$

Our goal is to determine the values c_i. Start by multiplying with $\prod_{k=0}^{n}(m+k)$:

$$\prod_{k=0}^{n}(m(1-\tau)+k) - (1-\tau)^{n+1}\prod_{k=0}^{n}(m+k) = \sum_{i=0}^{n}c_i\prod_{\substack{k=0\\k\neq i}}^{n}(m+k).$$

Now we successively set m equal to the roots of the denominator and solve for c_i. For the i-th root $m=-i$ we obtain

$$\prod_{k=0}^{n}(-i(1-\tau)+k) = c_i\prod_{\substack{k=0\\k\neq i}}^{n}(k-i)$$

$$c_i = \frac{\prod_{k=0}^{n}(-i(1-\tau)+k)}{\prod_{\substack{k=0\\k\neq i}}^{n}(k-i)}.$$

We can rewrite this in terms of the Gamma function as

$$c_i = (-1)^i\frac{\Gamma(-i(1-\tau)+n+1)}{\Gamma(i+1)\Gamma(n-i+1)\Gamma(-i(1-\tau))}.$$

Using the identity $\Gamma(-z) = -\frac{\pi}{\sin(\pi z)\Gamma(z+1)}$, we obtain

$$c_i = (-1)^{i+1} \frac{\Gamma(-i(1-\tau)+n+1)\Gamma(i(1-\tau)+1)}{\Gamma(i+1)\Gamma(n-i+1)} \frac{\sin(\pi i(1-\tau))}{\pi}.$$

In the following we use $Q := \frac{\Gamma(-i(1-\tau)+n+1)\Gamma(i(1-\tau)+1)}{\Gamma(i+1)\Gamma(n-i+1)}$.

We now give an upper bound on the absolute value of c_i. Start by using the value $\tau = 1 - \sqrt[n]{1-\beta}$ and let $1-\beta = e^{-c}$ for some $c > 0$. Consider

$$\ln \frac{\Gamma(ie^{-\frac{c}{n}}+1)}{\Gamma(i+1)} = \ln(\Gamma(ie^{-\frac{c}{n}}+1)) - \ln(\Gamma(i+1)) = -\int_0^{i-ie^{-\frac{c}{n}}} \Psi(1+i-t)dt \quad \text{and}$$

$$\ln \frac{\Gamma(-ie^{-\frac{c}{n}}+n+1)}{\Gamma(n-i+1)} = \ln(\Gamma(-ie^{-\frac{c}{n}}+n+1)) - \ln(\Gamma(n-i+1)) = \int_0^{i-ie^{-\frac{c}{n}}} \Psi(n-i+1+t)dt.$$

Therefore

$$\ln Q = \int_0^{i-ie^{-\frac{c}{n}}} \Psi(n-i+1+t) - \Psi(1+i-t)dt.$$

The Digamma function Ψ is increasing and thus the integrand is increasing and we get the approximation

$$\ln Q \leq (i - ie^{-\frac{c}{n}})(\Psi(n+1-ie^{-\frac{c}{n}}) - \Psi(1+ie^{-\frac{c}{n}})).$$

Let $i = tn$. Then for fixed t the expression on the right-hand side converges for $n \to \infty$ to

$$\lim_{n\to\infty} (i - ie^{-\frac{c}{n}})(\Psi(n+1-ie^{-\frac{c}{n}}) - \Psi(1+ie^{-\frac{c}{n}})) = ct\ln(\frac{1}{t}-1).$$

By numeric computation, the maximum of $t\ln(\frac{1}{t}-1)$ in the range $0 < t < 1$ is 0.278465. Thus,

$$\begin{aligned} \ln Q &\leq 0.278465c \\ Q &\leq (1-\beta)^{-0.278465}. \end{aligned}$$

Putting things together, we have

$$c_i \leq (-1)^{i+1}(1-\beta)^{-0.278465} \frac{\sin(\pi i(1-\tau))}{\pi} \leq \frac{1}{\pi}(1-\beta)^{-0.278465}.$$

The initial problem of estimating the partial fraction expansion from equation (21) now states

$$\begin{aligned} \frac{\prod_{k=0}^n (m(1-\tau)+k) - (1-\tau)^{n+1}\prod_{k=0}^n (m+k)}{\prod_{k=0}^n (m+k)} &= \frac{c_0}{m} + \frac{c_1}{m+1} + \ldots + \frac{c_n}{m+n} \\ &\leq \frac{\sum c_i}{m} \\ &\leq \frac{n\frac{1}{\pi}(1-\beta)^{-0.278465}}{m}. \end{aligned}$$

Now that we have bounds on the individual terms, we can give a bound on the complete expression

$$\begin{aligned} &2^{-\frac{(d-1)(n+1)}{4m}} d^{-\frac{(n+1)(d-n+1)}{2md}} N^{(\beta m\tau(d-n+1) - dm\tau + \binom{m+n}{m-1} - \binom{m(1-\tau)+n}{m(1-\tau)-1})\frac{n+1}{md}} \\ &\geq N^{\beta\tau(n+1)-\tau(n+1)+1-(1-\tau)^{n+1} - \frac{n\frac{1}{\pi}(1-\beta)^{-0.278465}}{m} + \ln(1-\beta)\beta\frac{n}{m}}. \end{aligned}$$