



HAL
open science

Factoring $N = p^r q^s$ for Large r and s

Jean-Sébastien Coron, Jean-Charles Faugère, Guénaél Renault, Rina Zeitoun

► **To cite this version:**

Jean-Sébastien Coron, Jean-Charles Faugère, Guénaél Renault, Rina Zeitoun. Factoring $N = p^r q^s$ for Large r and s . RSA Conference Cryptographers' Track , Feb 2016, San Francisco, United States. 10.1007/978-3-319-29485-8_26 . hal-01250302

HAL Id: hal-01250302

<https://inria.hal.science/hal-01250302v1>

Submitted on 11 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Factoring $N = p^r q^s$ for Large r and s

Jean-Sébastien Coron¹, Jean-Charles Faugère^{2,3,4}, Guénaél Renault^{3,2,4}, and Rina Zeitoun⁵

¹ University of Luxembourg

jean-sebastien.coron@uni.lu

² INRIA, POLSYS, Centre Paris-Rocquencourt, F-78153, Le Chesnay, France

³ Sorbonne Universités, UPMC Univ Paris 06, Équipe POLSYS, LIP6 UPMC, F-75005, Paris, France

⁴ CNRS, UMR 7606, LIP6 UPMC, F-75005, Paris, France

jean-charles.faugere@inria.fr

guenael.renault@lip6.fr

⁵ Oberthur Technologies, 420 rue d'Estienne d'Orves, CS 40008, 92705 Colombes, France

r.zeitoun@oberthur.com

Abstract. Boneh *et al.* showed at Crypto 99 that moduli of the form $N = p^r q$ can be factored in polynomial time when $r \simeq \log p$. Their algorithm is based on Coppersmith's technique for finding small roots of polynomial equations. In this paper we show that $N = p^r q^s$ can also be factored in polynomial time when r or s is at least $(\log p)^3$; therefore we identify a new class of integers that can be efficiently factored.

We also generalize our algorithm to moduli with k prime factors $N = \prod_{i=1}^k p_i^{r_i}$; we show that a non-trivial factor of N can be extracted in polynomial-time if one of the exponents r_i is large enough.

1 Introduction

At Crypto 98, Takagi [Tak98] showed that RSA decryption can be performed significantly faster with a modulus of the form $N = p^r q$, by using a p -adic expansion technique [Tak97]. However, at Crypto 99, Boneh, Durfee and Howgrave-Graham (BDH) showed that $N = p^r q$ can be factored in polynomial time for large r , when $r \simeq \log p$ [BDHG99]. Their algorithm is based on Coppersmith's technique for finding small roots of polynomial equations [Cop97], based on lattice reduction. This implies that Takagi's cryptosystem should not be used with a large r .

In light of the BDH attack, Takagi's cryptosystem was later extended by Lim *et al.* in [LKYL00] to moduli of the form $N = p^r q^s$. Namely the authors describe a public-key cryptosystem with modulus $N = p^r q^s$, and obtain even faster decryption than in Takagi's cryptosystem. In particular, for a 8192-bit RSA modulus of the form $N = p^2 q^3$, decryption becomes 15 times faster than for a standard RSA modulus of the same size.

In the BDH paper, the generalization of factoring moduli of the form $N = p^r q^s$ was explicitly left as an open problem. Therefore one could be tempted to use the Lim *et al.* cryptosystem [LKYL00], since no attack is known and it offers a significant speed-up compared to standard RSA. In this paper we show that moduli of the form $N = p^r q^s$ can also be factored in polynomial time for large r and/or s ; this gives a new class of integers that can be factored efficiently. Our result implies that the Lim *et al.* cryptosystem should not be used for large r or s .

Factoring $N = p^r q$ with Coppersmith. Coppersmith's technique for finding small roots of polynomial equations [Cop97] has found numerous applications in cryptography, for exam-

ple cryptanalysis of RSA with $d < N^{0.29}$ [BD00] (see also [DN00] for an extension), cryptanalysis of RSA with small secret CRT-exponents [JM07], and deterministic equivalence between recovering the private exponent d and factoring N [May04].

Coppersmith also showed that $N = pq$ can be factored in polynomial time when half of the bits of p are known [Cop97]. The BDH paper is actually an extension of this result for moduli $N = p^r q$, using a simplification by Howgrave-Graham [HG97]; namely the authors showed that knowing a fraction $1/(r+1)$ of the bits of p is enough for polynomial-time factorization of $N = p^r q$. Therefore when $r \simeq \log p$ only a constant number of bits of p must be known, hence those bits can be recovered by exhaustive search, and factoring $N = p^r q$ becomes polynomial-time [BDHG99].

As mentioned previously, in the BDH paper the generalization to moduli of the form $N = p^r q^s$ (where r and s can have the same size), is explicitly left as an open problem. To factor such N one could let $Q := q^s$ and try to apply BDH on $N = p^r Q$; however the condition for polynomial-time factorization becomes $r \simeq \log Q \simeq s \log q$; therefore this can only work if r is much larger than s . Alternatively a natural approach to factor $N = p^r q^s$ would be to write $N = (P+x)^r (Q+y)^s$ and apply Coppersmith's second theorem for finding small roots of bivariate polynomials over \mathbb{Z} ; however from Coppersmith's bound this does not seem to give a polynomial-time factorization (see Appendix A).

Factoring $N = p^r q^s$. In this paper we solve this open problem and describe a new algorithm to factor $N = p^r q^s$ in deterministic polynomial time when r and/or s is greater than $(\log p)^3$.

We first illustrate our technique with a particular case. Let consider a modulus of the form $N = p^{r+1} q^r$. As observed in [LKYL00], we can rewrite $N = (pq)^r p = P^r Q$ with $P := pq$ and $Q := p$ and apply BDH to $N = P^r Q$ to recover P and Q , which gives p and q . In that case the condition for polynomial-time factorization becomes $r = \Omega(\log Q) = \Omega(\log p)$, the same condition as BDH. This shows that $N = p^{r+1} q^r$ can also be factored in polynomial time for large r . We note that in [LKYL00] only moduli of the form $N = p^{r+1} q^r$ were considered for lattice-based factorisation.

However it is easy to generalize the previous observation to any modulus $N = p^{\alpha \cdot r + a} q^{\beta \cdot r + b}$ for small integers α, β, a and b . Namely as previously one can let $P := p^\alpha q^\beta$ and $Q := p^a q^b$ and apply BDH on $N = P^r Q$ to recover P and Q , which gives p and q . The condition for polynomial-time factorization is again $r = \Omega(\log Q)$, which for small a, b gives the same condition $r = \Omega(\log p)$ as previously (assuming that p and q have similar bitsize).

Now it is natural to ask whether we can generalize the above method to any modulus $N = p^r q^s$. More precisely, we should determine which class of integers (r, s) can be written as:

$$\begin{cases} r = u \cdot \alpha + a \\ s = u \cdot \beta + b \end{cases} \quad (1)$$

with large enough integer u , and small enough integers α, β, a, b , so that we can apply the above method; namely rewrite $N = p^r q^s$ as $N = P^u Q$ where $P := p^\alpha q^\beta$ and $Q := p^a q^b$, and apply BDH on $N = P^u Q$ to recover P and Q and eventually p and q . In this paper we show that it is enough that the max of r and s is $\Omega(\log^3 \max(p, q))$; namely in that case we are guaranteed to find a “good” decomposition of r and s according to (1), leading to a polynomial-time factorization of $N = p^r q^s$. Hence we identify a new class of integers that can be efficiently factored, namely $N = p^r q^s$ for large enough r or s (or both).

Extension to $N = \prod_{i=1}^k p_i^{r_i}$. We extend the above technique to moduli with k prime factors $N = \prod_{i=1}^k p_i^{r_i}$. Note that with 3 prime factors or more (instead of only 2) we cannot hope to obtain a complete factorization of N . Namely starting from an RSA modulus $N_1 = pq$ one could artificially embed N_1 into a larger modulus $N = (pq)^r q'$ for some known prime q' , and hope to recover the factorization of N_1 by factoring N ; clearly this cannot work. For the same reason we cannot hope to extract even a single prime factor of N ; namely given two RSA moduli $N_1 = p_1 q_1$ and $N_2 = p_2 q_2$ and using $N = (N_1)^r N_2$, extracting a prime factor of N would factor either N_1 or N_2 . Instead we show that we can always extract a non-trivial factor of N , if one of the exponents r_i is large enough. More precisely we can extract a non-trivial (not necessarily prime) factor of N in polynomial-time if one of the k exponents r_i is at least $(\log p)^{\theta_k}$, with $\theta_3 = 17$, $\theta_4 = 61$, $\theta_5 = 257$ and $\theta_k \sim 4e \cdot (k-1)!$ for large k . Note that the exponent θ_k grows exponentially with the number of prime factors k ; however for a fixed value of k extracting a non-trivial factor of N is always polynomial-time in $\log N$.

Practical Experiments. It is well known that the BDH algorithm for factoring $N = p^r q$ is unpractical. Namely the experiments from [BDHG99] show that the BDH algorithm is practical only for relatively small primes p and q , namely 96 bits in [BDHG99], but for such small primes factors the ECM method [Len87] performs much better. However ECM is subexponential whereas BDH is polynomial-time, so at some point the BDH algorithm must beat ECM; the authors conjecture that BDH should become faster than ECM in practice when p and q are roughly 400 bits.

Needless to say, our algorithm for factoring $N = p^r q^s$ should be even less practical, since for $N = p^r q^s$ we need much larger exponents r or s than in BDH for $N = p^r q$. However we have performed some practical experiments, in order to estimate the running time of our algorithm for factoring a modulus of the form $N = p^r q^s$. We describe the results in Section 5; unsurprisingly we observed that for relatively small primes p and q , namely 128 bits, our algorithm performs much worse than ECM. However as for BDH our algorithm scales polynomially whereas ECM scales exponentially, so our algorithm must also beat ECM for large enough p and q .

2 Background

We first recall the following Landau notations: we write $f(n) = \mathcal{O}(g(n))$ if there exists constants n_0 and $c > 0$ such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$. We write $f(n) = \Omega(g(n))$ if $g(n) = \mathcal{O}(f(n))$. Therefore $f(n) = \Omega(g(n))$ if and only if there exists constants n_0 and $c > 0$ such that $|f(n)| \geq c|g(n)|$ for all $n \geq n_0$.

2.1 LLL and Simultaneous Diophantine Approximation

Let $b_1, \dots, b_d \in \mathbb{Z}^n$ be linearly independent vectors with $d \leq n$. A lattice L spanned by $\langle b_1, \dots, b_d \rangle$ is the set of all integer linear combinations of b_1, \dots, b_d . Here we consider full-rank lattices, *i.e.* $d = n$. The $d \times d$ matrix $M = (b_1, \dots, b_d)$ is called a *basis* of L . The algorithms described in this paper require the ability to find short vectors in a lattice. This can be achieved by the celebrated LLL algorithm [LLL82].

Theorem 1 (LLL). *Let L be a lattice spanned by $\langle b_1, \dots, b_d \rangle \in \mathbb{Z}^n$. The LLL algorithm, given $\langle b_1, \dots, b_d \rangle$, finds in time polynomial in the size of the entries, a vector v such that:*

$$\|v\| \leq 2^{(d-1)/4} \det(L)^{1/d}.$$

In this paper we also use an application of LLL for simultaneous Diophantine approximation; we recall the theorem from [LLL82].

Theorem 2. *There exists a polynomial time algorithm that, given a positive integer n and rational numbers $e_1, e_2, \dots, e_n, \varepsilon$ satisfying $0 < \varepsilon < 1$, finds integers p_1, p_2, \dots, p_n, q for which*

$$|p_i - qe_i| \leq \varepsilon \text{ for } 1 \leq i \leq n, \text{ and } 1 \leq q \leq 2^{\frac{n(n+1)}{4}} \varepsilon^{-n}.$$

2.2 Coppersmith's Algorithm

We recall Coppersmith's first theorem [Cop97] for finding small roots of univariate modular polynomial equations.

Theorem 3 (Coppersmith). *Let $f(x)$ be a monic polynomial of degree r in one variable, modulo an integer N of unknown factorization. Let X be such that $X < N^{1/r}$. One can find all integers x_0 with $f(x_0) \equiv 0 \pmod{N}$ and $|x_0| < X$ in time polynomial in $(\log N, r)$.*

In the original Coppersmith paper the complexity is stated as polynomial in $(\log N, 2^r)$ where r is the degree of the polynomial equation, but it is well known that the 2^r is a typo and the complexity is polynomial in r only; see for example [BM05, Theorem 11]. We recall the main steps of Coppersmith's algorithm in Appendix B.

The following variant of Coppersmith's first theorem was obtained by Blömer and May [BM05], using Coppersmith's technique for finding small roots of bivariate integer equations.

Theorem 4 ([BM05, Corollary 14]). *Let N be a composite integer of unknown factorization with divisor $b \geq N^\beta$. Let $f(x) = \sum_i f_i x^i \in \mathbb{Z}[x]$ be a polynomial of degree δ with $\gcd(f_1, \dots, f_\delta, N) = 1$. Then we can find all points $x_0 \in \mathbb{Z}$ satisfying $f(x_0) = b$ in time polynomial in $\log N$ and δ provided that $|x_0| \leq N^{\beta^2/\delta}$.*

2.3 The Boneh-Durfee-Howgrave-Graham Algorithm

At Crypto 99, Boneh, Durfee and Howgrave-Graham [BDHG99] showed that moduli of the form $N = p^r q$ can be factored in polynomial time for large r , when $r \simeq \log p$. We recall their main theorem.

Theorem 5 (BDH). *Let $N = p^r q$ where $q < p^c$ for some c . The factor p can be recovered from N , r , and c by an algorithm with a running time of:*

$$\exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot \mathcal{O}(\gamma),$$

where γ is the time it takes to run LLL on a lattice of dimension $\mathcal{O}(r^2)$ with entries of size $\mathcal{O}(r \log N)$. The algorithm is deterministic, and runs in polynomial space.

Their algorithm is based on Coppersmith's technique for finding small roots of polynomial equations. We recall the main steps of the proof in Appendix C. When p and q have similar bitsize we can take $c = 1$; in that case we have $(c + 1)/(r + c) = \mathcal{O}(1/r)$ and therefore the algorithm is polynomial time when $r = \Omega(\log p)$. More generally one can take $c = \log q / \log p$, which gives:

$$\frac{c + 1}{r + c} \cdot \log p \leq \frac{c + 1}{r} \cdot \log p \leq \frac{\frac{\log q}{\log p} + 1}{r} \cdot \log p \leq \frac{\log q + \log p}{r}$$

Therefore a sufficient condition for polynomial-time factorization is $r = \Omega(\log q + \log p)$.

Actually by simple inspection of the proof of Theorem 5 in [BDHG99] one can obtain the slightly simpler condition $r = \Omega(\log q)$. We use the following theorem for the rest of the paper.

Theorem 6 (BDH). *Let p and q be two integers with $p \geq 2$ and $q \geq 2$, and let $N = p^r q$. The factors p and q can be recovered in polynomial time in $\log N$ if $r = \Omega(\log q)$.*

We provide the proof of Theorem 6 in Appendix D, based on Lemma 3.3 from [BDHG99]. Note that p and q can be any integers, not necessarily primes.

We can also obtain a proof of Theorem 6 directly from [BM05, Corollary 14], as recalled in Theorem 4. Namely given $N = p^r q$ we let the divisor $b := p^r$ and:

$$f(x) = (V + x)^r$$

where V is an integer such that $p = V + x_0$ and the high-order bits of V are the same as the high-order bits of p . One must then solve $f(x_0) = b$, and applying [BM05, Corollary 14] this can be done in time polynomial in $\log N$ and r provided that $|x_0| < N^{\beta^2/r}$. We can take β such that $b = p^r = N^\beta$. This gives the condition:

$$|x_0| < p^\beta \tag{2}$$

From $p^r = N^\beta = (p^r q)^\beta$ we get:

$$\beta = \frac{r \log p}{r \log p + \log q} = \frac{1}{1 + \frac{\log q}{r \log p}} \geq 1 - \frac{\log q}{r \log p}$$

Therefore from (2) a sufficient condition for applying [BM05, Corollary 14] is:

$$|x_0| < p \cdot q^{-1/r}$$

Therefore one can perform exhaustive search on the high-order bits of p under the previous condition $r = \Omega(\log q)$, and eventually recover the factors p and q , still in time polynomial in $\log N$.

3 Factoring $N = p^r q^s$ for Large r

We prove the following theorem; this is the main theorem of our paper.

Theorem 7. *Let $N = p^r q^s$ be an integer of unknown factorization with $r > s$ and $\gcd(r, s) = 1$. Given N as input one can recover the prime factors p and q in polynomial time in $\log N$ under the condition $r = \Omega(\log^3 \max(p, q))$.*

We first provide a proof intuition. Note that given $N = p^r q^s$ as input we can assume that the exponents r and s are known, since otherwise they can be recovered by exhaustive search in time $\mathcal{O}(\log^2 N)$.

As explained in introduction, given as input $N = p^r q^s$ and assuming that r and s are known, our technique consists in representing r and s as:

$$\begin{cases} r = u \cdot \alpha + a \\ s = u \cdot \beta + b \end{cases} \quad (3)$$

with large enough integer u , and small enough integers α, β, a, b , so that N can be rewritten as:

$$N = p^r q^s = p^{u \cdot \alpha + a} \cdot q^{u \cdot \beta + b} = (p^\alpha q^\beta)^u \cdot p^a q^b = P^u \cdot Q$$

where $P := p^\alpha q^\beta$ and $Q := p^a q^b$. One can then apply BDH on $N = P^u Q$ to recover P and Q and eventually p and q .

Observe that from (3) we obtain:

$$r \cdot \beta - s \cdot \alpha = \gamma \quad (4)$$

where $\gamma := a \cdot \beta - b \cdot \alpha$ must be a small integer, since α, β, a and b must be small. This gives:

$$\alpha \cdot s \equiv -\gamma \pmod{r} \quad (5)$$

Using LLL in dimension 2 (or equivalently the Gauss-Lagrange algorithm), we can find two small integers α and γ satisfying (5) with $|\alpha| \cdot |\gamma| \simeq r$. We then recover β from (4); for integers r and s of similar bitsize, we get $|\beta| \simeq |\alpha|$. The integer u is then defined as $u := \lfloor r/\alpha \rfloor$, and we let a be the remainder of the division of r by α . We obtain $|a|, |\beta| \simeq |\alpha|$ and $|b| \simeq |\gamma|/|\alpha| \simeq r/|\alpha|^2$.

Recall that the condition for BDH factorization is $u = \Omega(\log Q)$; assuming for simplicity that p and q have similar bitsize, from $Q = p^a q^b$ we get the condition:

$$u \simeq \left(|\alpha| + \frac{r}{|\alpha|^2} \right) \log p$$

It is therefore optimal to take $|\alpha| \simeq r^{1/3}$, which gives $u \simeq r^{1/3} \log p$, and with $u := \lfloor r/\alpha \rfloor \simeq r^{2/3}$ we obtain $r^{1/3} \simeq \log p$. This gives the condition $r \simeq \log^3 p$; therefore we recover the condition from Theorem 7 for prime factors p and q of similar bitsize.

We now provide a rigorous analysis. The proof of Theorem 7 is based on the following lemma.

Lemma 1. *Let r and s be two integers such that $r > s > 0$. One can compute in polynomial time integers u, α, β, a, b such that*

$$\begin{cases} r = u \cdot \alpha + a \\ s = u \cdot \beta + b \end{cases} \quad (6)$$

with $0 < \alpha \leq 2r^{1/3}$, $0 \leq \beta \leq \alpha$, $|a| < \alpha$, $|b| \leq 6r^{2/3}/\alpha$, $u > r/\alpha - 1$, where the integers a and b are either both ≥ 0 (Case 1), or both ≤ 0 (Case 2).

Proof. We first generate two small integers $\alpha > 0$ and β such that:

$$r \cdot \beta - s \cdot \alpha = \gamma, \quad (7)$$

for some small integer γ . For this we apply LLL on the following matrix M of row vectors:

$$M = \begin{pmatrix} \lfloor r^{1/3} \rfloor & -s \\ 0 & r \end{pmatrix}.$$

We obtain a short non-zero vector $\mathbf{v} = (\lfloor r^{1/3} \rfloor \cdot \alpha, \gamma)$, where $\gamma = -s \cdot \alpha + r \cdot \beta$ for some $\beta \in \mathbb{Z}$; hence we obtain integers α , β and γ satisfying (7). From Theorem 1 we must have

$$\|\mathbf{v}\| \leq 2^{1/4} \cdot (\det M)^{1/2} \leq 2^{1/4} \cdot (\lfloor r^{1/3} \rfloor \cdot r)^{1/2} \leq 2^{1/4} \cdot r^{2/3} \quad (8)$$

Note that by applying the Gauss-Lagrange algorithm instead of LLL one can obtain a slightly better bound for $\|\mathbf{v}\|$, corresponding to Minkowski bound.

From (8) we obtain $|\alpha| \leq 2r^{1/3}$ and $|\gamma| \leq 2r^{2/3}$. We can take $\alpha \geq 0$. Moreover we must have $\alpha \neq 0$ since otherwise we would have $\mathbf{v} = (0, \beta r)$ for some integer $\beta \neq 0$, which would give $\|\mathbf{v}\| \geq r$, which would contradict the previous bound. Therefore we must have $0 < \alpha \leq 2r^{1/3}$.

From (7) we have $\beta = (\gamma + \alpha \cdot s)/r$ and moreover using $-1 < \gamma/r < 1$ and $0 < s < r$ we obtain:

$$-1 < \frac{\gamma}{r} < \frac{\gamma + \alpha \cdot s}{r} < \frac{\gamma}{r} + \alpha < 1 + \alpha$$

Since α and β are integers this implies $0 \leq \beta \leq \alpha$. We now show how to generate the integers u , a and b . We distinguish two cases.

Case 1: $\beta = 0$ or ($\beta \neq 0$ and $\lfloor r/\alpha \rfloor \leq s/\beta$). In that case we let:

$$u := \left\lfloor \frac{r}{\alpha} \right\rfloor$$

and we let $a := r - u \cdot \alpha$ and $b := s - u \cdot \beta$; this gives (6) as required. Since a is the remainder of the division of r by α we must have $0 \leq a < \alpha$. If $\beta = 0$ we then have $b = s > 0$. If $\beta \neq 0$ we have using $\lfloor r/\alpha \rfloor \leq s/\beta$:

$$b = s - u \cdot \beta = s - \left\lfloor \frac{r}{\alpha} \right\rfloor \cdot \beta \geq s - \frac{s}{\beta} \cdot \beta = 0$$

so in both cases $b \geq 0$. Therefore in Case 1 we have that the integers a and b are both ≥ 0 . Moreover combining (6) and (7) we obtain $a \cdot \beta - b \cdot \alpha = \gamma$, which gives using $0 \leq \beta \leq \alpha$ and $0 \leq a < \alpha$:

$$0 \leq b = \frac{a \cdot \beta - \gamma}{\alpha} < \alpha + \frac{2r^{2/3}}{\alpha}$$

Since $0 < \alpha \leq 2r^{1/3}$ we have $4r^{2/3}/\alpha \geq 2r^{1/3} \geq \alpha$, therefore we obtain as required:

$$0 \leq b < \frac{6r^{2/3}}{\alpha}$$

Case 2: $\beta \neq 0$ and $\lfloor r/\alpha \rfloor > s/\beta$. In that case we let:

$$u := \left\lceil \frac{r}{\alpha} \right\rceil$$

As previously we let $a := r - u \cdot \alpha$ and $b := s - u \cdot \beta$, which gives again (6); moreover we have $-\alpha < a \leq 0$. As previously using $\lceil r/\alpha \rceil \geq \lfloor r/\alpha \rfloor > s/\beta$ we obtain:

$$b = s - u \cdot \beta = s - \left\lceil \frac{r}{\alpha} \right\rceil \cdot \beta < s - \frac{s}{\beta} \cdot \beta = 0$$

Therefore in Case 2 we have that the integers a and b are both ≤ 0 . As previously using $0 \leq \beta \leq \alpha$, $-\alpha < a \leq 0$ and $\alpha \leq 4r^{2/3}/\alpha$ we obtain as required:

$$|b| \leq \left| \frac{a \cdot \beta - \gamma}{\alpha} \right| < \alpha + \frac{2r^{2/3}}{\alpha} \leq \frac{6r^{2/3}}{\alpha}$$

This terminates the proof of Lemma 1. □

3.1 Proof of Theorem 7

We now proceed with the proof of Theorem 7. We are given as input $N = p^r q^s$ with $r > s > 0$ and $\gcd(r, s) = 1$. We can assume that the exponents r and s are known, otherwise they can be recovered by exhaustive search in time $\mathcal{O}(\log^2 N)$. We apply Lemma 1 with r, s and obtain u, α, β, a and b such that:

$$\begin{cases} r = u \cdot \alpha + a \\ s = u \cdot \beta + b \end{cases}$$

We first consider Case 1 of Lemma 1 with $a \geq 0$ and $b \geq 0$. In that case the modulus $N = p^r q^s$ can be rewritten as follows:

$$N = p^r q^s = p^{u \cdot \alpha + a} q^{u \cdot \beta + b} = (p^\alpha q^\beta)^u p^a q^b = P^u Q ,$$

where $P := p^\alpha q^\beta$ and $Q := p^a q^b$. One can then apply Theorem 6 on $N = P^u Q$ to recover P and Q in polynomial time in $\log N$ under the condition $u = \Omega(\log Q)$. Since $u > r/\alpha - 1$, we get the sufficient condition $r = \Omega(\alpha \cdot \log Q)$. We have from the bounds of Lemma 1:

$$\begin{aligned} \alpha \cdot \log Q &= \alpha \cdot (a \log p + b \log q) \leq \alpha \cdot \left(\alpha \cdot \log p + \frac{6r^{2/3}}{\alpha} \cdot \log q \right) \\ &\leq \alpha^2 \cdot \log p + 6r^{2/3} \cdot \log q \leq 10 \cdot r^{2/3} \cdot \log \max(p, q) \end{aligned}$$

which gives the sufficient condition $r = \Omega(r^{2/3} \cdot \log \max(p, q))$. Therefore one can recover P and Q in polynomial time under the condition:

$$r = \Omega(\log^3 \max(p, q))$$

Alternatively the factors P and Q can be recovered by applying the variant of Copper-smith's theorem from [BM05, Corollary 14], *i.e.* Theorem 4. Namely as explained in Section 2.3, given $N = P^u q$ we can let $b := P^u$ and let:

$$f(x) := (V + x)^u$$

where V is an integer such that $P = V + x_0$ and the high-order bits of V are the same as the high-order bits of p . One must then solve $f(x_0) = b$, and applying [BM05, Corollary 14] this

can be done in time polynomial in $\log N$ and u provided that $|x_0| < N^{\beta^2/u}$. As in Section 2.3, we can take $b = P^u = N^\beta$, and we obtain the sufficient condition:

$$|x_0| < P \cdot Q^{-1/u}$$

Therefore one can perform exhaustive search on the high-order bits of P in polynomial time in $\log N$ under the same condition as previously, namely $u = \Omega(\log Q)$. As previously one recovers P and Q in polynomial time under the condition $r = \Omega(\log^3 \max(p, q))$.

Finally the prime factors p and q can easily be recovered from $P = p^\alpha q^\beta$ and $Q = p^a q^b$. Namely the matrix $\begin{pmatrix} a & b \\ \alpha & \beta \end{pmatrix}$ whose determinant is $a\beta - b\alpha = \gamma$, is invertible with inverse $\begin{pmatrix} \beta/\gamma & -b/\gamma \\ -\alpha/\gamma & a/\gamma \end{pmatrix}$. Namely we must have $\gamma \neq 0$, since otherwise we would have $\beta \cdot r = \alpha \cdot s$; since we have $\gcd(r, s) = 1$, the integer α would be a non-zero multiple of r , which would contradict the bound from Lemma 1. Therefore one can retrieve p and q by computing:

$$\begin{cases} Q^{\frac{\beta}{\gamma}} \cdot P^{-\frac{b}{\gamma}} = (p^a q^b)^{\frac{\beta}{\gamma}} \cdot (p^\alpha q^\beta)^{-\frac{b}{\gamma}} = p^{\frac{a\beta - b\alpha}{\gamma}} \cdot q^{\frac{b\beta - b\beta}{\gamma}} = p^1 \cdot q^0 = p \\ Q^{-\frac{\alpha}{\gamma}} \cdot P^{\frac{a}{\gamma}} = (p^a q^b)^{-\frac{\alpha}{\gamma}} \cdot (p^\alpha q^\beta)^{\frac{a}{\gamma}} = p^{\frac{a\alpha - a\alpha}{\gamma}} \cdot q^{\frac{a\beta - b\alpha}{\gamma}} = p^0 \cdot q^1 = q \end{cases}.$$

We now consider Case 2 from Lemma 1, that is $a \leq 0$ and $b \leq 0$. In that case we can write:

$$N = p^r q^s = p^{u\alpha+a} q^{u\beta+b} = (p^\alpha q^\beta)^u p^a q^b = P^u / Q$$

for $P := p^\alpha q^\beta$ and $Q := p^{-a} q^{-b}$. Note that Q is an integer because $a \leq 0$ and $b \leq 0$. We obtain $P^u = Q \cdot N$ which implies:

$$P^u \equiv 0 \pmod{N}$$

Therefore P is a small root of a univariate polynomial equation of degree u modulo N ; hence we can apply Coppersmith's first theorem; the condition from Theorem 3 is $P \leq N^{1/u} = P/Q^{1/u}$. Although the condition is not directly satisfied, it can be met by doing exhaustive search on the high-order $(\log Q)/u$ bits of P , which is still polynomial time under the condition $u = \Omega(\log Q)$; this is the same condition as in Case 1 for BDH.

More precisely, we write $P = X \cdot t + x_0$ where $X = \lfloor N^{1/u} \rfloor$ and $|x_0| \leq X$. We obtain the polynomial equation:

$$(X \cdot t + x_0)^u \equiv 0 \pmod{N}$$

For a fixed t this is a univariate modular polynomial equation of degree u and small unknown x_0 . We have $X < N^{1/u}$; therefore we can apply Theorem 3 and recover x_0 in polynomial time in $\log N$, since the degree u satisfies $u \leq r \leq \log N$. We do exhaustive search on t , where:

$$0 \leq t \leq P/X \leq 2P/N^{1/u} = 2Q^{1/u}$$

Therefore the algorithm is still polynomial time under the same condition as in Case 1, namely $u = \Omega(\log Q)$. Since in Lemma 1 the bounds on u , a and b are the same in both Case 1 and Case 2, we obtain that in Case 2 recovering P and Q is polynomial-time under the same condition $r = \Omega(\log^3 \max(p, q))$. As previously given P and Q one can easily recover the prime factors p and q . This terminates the proof of Theorem 7.

4 Generalization to $N = \prod_{i=1}^k p_i^{r_i}$ for Large r_i 's

We prove the following theorem, which is a generalization of Theorem 7 to moduli $N = \prod_{i=1}^k p_i^{r_i}$ with more than two prime factors. As explained in introduction, in that case we cannot hope to obtain a complete factorization of N ; however we show that we can always recover a non-trivial factor of N in polynomial time if the largest r_i is at least $\Omega(\log^{\theta_k} \max p_i)$, for some sequence θ_k with $\theta_3 = 17$, $\theta_4 = 61$, $\theta_5 = 257$ and $\theta_k \sim 4e \cdot (k-1)!$ for large k .

Theorem 8. *Let $k \geq 2$ be fixed and let $N = \prod_{i=1}^k p_i^{r_i}$ where $r_1 = \max(r_i)$. Let $p := \max\{p_i, 1 \leq i \leq k\}$. Given N as input one can recover a non-trivial factor of N in time polynomial in $\log N$ if $r_1 = \Omega(\log^{\theta_k} p)$, where $\theta_2 = 5$ and:*

$$\theta_k = 4(k-1) \left(1 + \sum_{i=1}^{k-2} \prod_{j=i}^{k-2} j \right) + 1,$$

with $\theta_k = 4e \cdot (k-1)! - 3 - o(1)$ for large k .

4.1 A Preliminary Lemma

We first provide a generalization of Lemma 1 to ℓ integers.

Lemma 2. *Let $\ell \geq 1$, let $r_1 \geq r_2 \geq \dots \geq r_\ell > 0$ be integers and let ε with $0 < \varepsilon < 1$. One can compute in polynomial time integers u , a_i and b_i such that for all $1 \leq i \leq \ell$, $r_i = u \cdot a_i + b_i$, with $a_1 \neq 0$, $u > (1 - \varepsilon) \cdot r_1/a_1 - 1$, and for all $1 \leq i \leq \ell$, $0 \leq a_i \leq 2^{\ell^2/4} \cdot \varepsilon^{-(\ell-1)}$ and:*

$$0 \leq b_i \leq a_1 + 2 \cdot \frac{r_1 \cdot \varepsilon}{a_1} \cdot \frac{r_1}{r_\ell} \quad (9)$$

Proof. If $\ell = 1$ we take $u = r_1$, $a_1 = 1$ and $b_1 = 0$. We now consider the case $\ell \geq 2$. We start by finding ℓ small integers a_1, \dots, a_ℓ and $\ell - 1$ small integers c_2, \dots, c_ℓ such that:

$$2 \leq i \leq \ell, \quad r_1 \cdot a_i - r_i \cdot a_1 = c_i \quad (10)$$

For this we apply Theorem 2 with $n := \ell - 1$ and $e_{i-1} := r_i/r_1$ for $2 \leq i \leq \ell$. This gives integers a_1, a_2, \dots, a_ℓ such that $|a_i - a_1 \cdot r_i/r_1| \leq \varepsilon$ for all $2 \leq i \leq \ell$. Therefore we obtain (10) with

$$2 \leq i \leq \ell, \quad |c_i| \leq r_1 \cdot \varepsilon, \quad \text{and} \quad 1 \leq a_1 \leq 2^{\ell^2/4} \cdot \varepsilon^{-(\ell-1)}$$

From (10), we have $a_i = (c_i + r_i \cdot a_1)/r_1$, which gives using $r_i \leq r_1$ and $0 < \varepsilon < 1$:

$$-1 < -\varepsilon < -\varepsilon + \frac{r_i \cdot a_1}{r_1} \leq a_i = \frac{c_i + r_i \cdot a_1}{r_1} \leq \varepsilon + \frac{r_i \cdot a_1}{r_1} < 1 + a_1, \quad ,$$

and since a_1 and a_i are integers, as required we must have $0 \leq a_i \leq a_1 \leq 2^{\ell^2/4} \cdot \varepsilon^{-(\ell-1)}$ for all $2 \leq i \leq \ell$.

We now show how to generate the integers u and b_i . We let:

$$u := \min \left\{ \left\lfloor \frac{r_i}{a_i} \right\rfloor \text{ for } 1 \leq i \leq \ell, \text{ with } a_i \neq 0 \right\}.$$

We know that such u exists because $a_1 \neq 0$. We take the smallest index j such that $u = \lfloor r_j/a_j \rfloor$. Using $r_1 \cdot a_j - r_j \cdot a_1 = c_j$ with $|c_j| \leq r_1 \cdot \varepsilon$ we obtain as required:

$$u = \left\lfloor \frac{r_j}{a_j} \right\rfloor > \frac{r_j}{a_j} - 1 = \frac{r_1}{a_1} - \frac{c_j}{a_1 \cdot a_j} - 1 \geq \frac{r_1}{a_1} - \frac{r_1 \cdot \varepsilon}{a_1 \cdot a_j} - 1 \geq \frac{r_1}{a_1} \cdot (1 - \varepsilon) - 1$$

We let $b_i := r_i - u \cdot a_i$ for all $1 \leq i \leq \ell$, which gives as required:

$$r_i = u \cdot a_i + b_i \tag{11}$$

and by definition of u we must have $b_i \geq 0$ for all $1 \leq i \leq \ell$. Combining (10) and (11) we obtain:

$$b_1 \cdot a_i - b_i \cdot a_1 = c_i \tag{12}$$

From $0 \leq a_i \leq a_1$ for all $1 \leq i \leq \ell$, we obtain for all $1 \leq i \leq \ell$:

$$|b_i| = \left| \frac{b_1 \cdot a_i - c_i}{a_1} \right| \leq \frac{|b_1| \cdot a_i + |c_i|}{a_1} \leq |b_1| + \frac{r_1 \cdot \varepsilon}{a_1} . \tag{13}$$

Moreover for index j by definition of u the integer b_j is the remainder of the division of r_j by a_j , therefore $0 \leq b_j < a_j$. Using $b_1 = (b_j \cdot a_1 + c_j)/a_j$ from (12), we obtain using (13) and $|c_j| \leq r_1 \cdot \varepsilon$, for all $1 \leq i \leq \ell$:

$$|b_i| \leq |b_1| + \frac{r_1 \cdot \varepsilon}{a_1} \leq \left| \frac{b_j \cdot a_1 + c_j}{a_j} \right| + \frac{r_1 \cdot \varepsilon}{a_1} \leq a_1 + \frac{r_1 \cdot \varepsilon}{a_j} + \frac{r_1 \cdot \varepsilon}{a_1}$$

From the definition of j we have $r_j/a_j \leq r_1/a_1$, which gives using $r_1 \geq r_j$:

$$|b_i| \leq a_1 + \frac{r_1 \cdot \varepsilon}{a_1} \cdot \frac{r_1}{r_j} + \frac{r_1 \cdot \varepsilon}{a_1} \leq a_1 + 2 \cdot \frac{r_1 \cdot \varepsilon}{a_1} \cdot \frac{r_1}{r_j} \tag{14}$$

Eventually from $r_j \geq r_\ell$ we obtain (9); this proves Lemma 2. \square

4.2 Factoring with Gaps

Using the previous lemma we show that $N = \prod_{i=1}^k p_i^{r_i}$ can be factored in polynomial time under certain conditions, namely r_1 should be large enough, and moreover there should be a gap between r_ℓ and $r_{\ell+1}$ for some $\ell < k$, or all the r_i 's should be large enough. We later show how to remove this additional condition, in order to get a condition on r_1 only, as required in Theorem 8.

Lemma 3. *Let $k \geq 2$ be fixed and let $N = \prod_{i=1}^k p_i^{r_i}$ with $r_1 \geq r_2 \geq \dots \geq r_k$, and let $p := \max\{p_i, 1 \leq i \leq k\}$. Let $\ell \in \mathbb{Z}$ with $1 \leq \ell \leq k$ and let $\rho \geq 0$ be such that $r_1/r_\ell \leq \log^\rho p$ and $r_1/r_{\ell+1} > \log^{(\ell-1)(\rho+1)+1} p$ if $\ell < k$. One can recover a non-trivial factor of N in polynomial time in $\log N$ if $r_1 = \Omega(\log^{2(\ell-1)(\rho+1)+1} p)$.*

Proof. As previously we can assume that the exponents r_i 's are known; otherwise we can recover them by exhaustive search in time $\mathcal{O}(\log^k N)$; for a fixed k this is still polynomial in $\log N$.

We let $\varepsilon := 1/\log^{\rho+1} p$. From Lemma 2 we compute in polynomial time integers u , a_i and b_i such that for all $1 \leq i \leq \ell$:

$$r_i = u \cdot a_i + b_i$$

In Lemma 2 the integers a_i 's and b_i 's are all non-negative. Therefore we can write:

$$N = \prod_{i=1}^k p_i^{r_i} = \left(\prod_{i=1}^{\ell} p_i^{a_i} \right)^u \left(\prod_{i=1}^{\ell} p_i^{b_i} \prod_{i=\ell+1}^k p_i^{r_i} \right) = P^u Q,$$

where

$$P := \prod_{i=1}^{\ell} p_i^{a_i}, \quad Q := \left(\prod_{i=1}^{\ell} p_i^{b_i} \right) \left(\prod_{i=\ell+1}^k p_i^{r_i} \right)$$

According to Theorem 6, one can therefore apply the BDH factorization method on $N = P^u Q$ to recover P and Q in polynomial time in $\log N$ if $u = \Omega(\log Q)$. Using $u > (1 - \varepsilon) \cdot r_1/a_1 - 1$, we get the sufficient condition $r_1 = \Omega(a_1 \log Q)$. When $\ell < k$, we have:

$$a_1 \log Q = a_1 \cdot \left(\sum_{i=1}^{\ell} b_i \log p_i + \sum_{i=\ell+1}^k r_i \log p_i \right)$$

Using (9) from Lemma 2, and $r_i \leq r_{\ell+1}$ for all $\ell + 1 \leq i \leq k$, we obtain:

$$\begin{aligned} a_1 \log Q &\leq a_1 \cdot \left(\ell \cdot \left(a_1 + 2 \cdot \frac{r_1 \cdot \varepsilon}{a_1} \cdot \frac{r_1}{r_{\ell}} \right) + (k - \ell) \cdot r_{\ell+1} \right) \cdot \log p \\ &\leq \left(a_1^2 \cdot \ell + 2\ell \cdot r_1 \cdot \varepsilon \cdot \frac{r_1}{r_{\ell}} + (k - \ell) \cdot a_1 \cdot r_{\ell+1} \right) \cdot \log p \end{aligned}$$

Under the conditions of Lemma 3 we have $r_1/r_{\ell} \leq \log^{\rho} p$ and $r_{\ell+1} < r_1 \cdot \log^{-(\ell-1)(\rho+1)-1} p$, which gives:

$$a_1 \log Q \leq a_1^2 \cdot k \cdot \log p + 2k \cdot r_1 \cdot \varepsilon \cdot \log^{\rho+1} p + (k - \ell) \cdot a_1 \cdot r_1 \cdot \log^{-(\ell-1)(\rho+1)} p$$

From Lemma 2 we have:

$$0 < a_1 \leq 2^{\ell^2/4} \varepsilon^{-(\ell-1)} \leq 2^{k^2/4} \cdot \log^{(\ell-1)(\rho+1)} p \quad (15)$$

and using $\varepsilon = 1/\log^{\rho+1} p$ we obtain:

$$a_1 \log Q \leq k \cdot a_1^2 \cdot \log p + 2k \cdot r_1 + (k - \ell) \cdot 2^{k^2/4} \cdot r_1 \quad (16)$$

Similarly when $\ell = k$ we have:

$$a_1 \log Q = a_1 \cdot \sum_{i=1}^k b_i \log p_i \leq k \cdot a_1^2 \cdot \log p + 2k \cdot r_1$$

Therefore (16) holds for any $1 \leq \ell \leq k$.

Recall that to recover P and Q in polynomial time we must ensure $r_1 = \Omega(a_1 \log Q)$. Since k is fixed, from (16) it suffices to have $r_1 = \Omega(a_1^2 \log p)$. From (15) we have:

$$a_1^2 \cdot \log p \leq 2^{k^2/2} \cdot \log^{2 \cdot (\ell-1) \cdot (\rho+1) + 1} p$$

which as required gives the sufficient condition:

$$r_1 = \Omega\left(\log^{2 \cdot (\ell-1) \cdot (\rho+1)+1} p\right) \quad (17)$$

Finally since $r_1 = \Omega(a_1^2 \log p)$ we must have $r_1 > a_1$ for large enough $\log p$. This gives $0 < a_1 < r_1$ and therefore $1 < P < N$; therefore P is a non-trivial factor of N . We can therefore obtain a non-trivial factor of N in polynomial time under condition (17); this proves Lemma 3. \square

4.3 Proof of Theorem 8

The previous lemma is not completely satisfactory since to recover a non-trivial factor of N we need an additional condition on the exponents r_i , namely there must be a gap between r_ℓ and $r_{\ell+1}$ for some $\ell < k$, or all the r_i 's should be large enough. In this section we show how to remove this condition and prove Theorem 8 with a condition on r_1 only. Namely if r_1 is large enough, we show that either all the r_i 's are large enough, or there must be a gap between r_ℓ and $r_{\ell+1}$ for some $\ell < k$.

We are given a modulus $N = \prod_{i=1}^k p_i^{r_i}$ with $r_1 \geq r_2 \geq \dots \geq r_k$. As previously we can assume that the exponents r_i 's are known; otherwise we can recover them by exhaustive search in time $\mathcal{O}(\log^k N)$; for a fixed k this is still polynomial in $\log N$. We define $\rho_1 = 0$ and for all $1 \leq \ell \leq k-1$:

$$\rho_{\ell+1} = (\ell-1)(\rho_\ell + 1) + 1 \quad (18)$$

We consider the following possible cases on the exponents r_i :

$$1 \leq \ell \leq k-1, \quad \text{Case } \ell : \begin{cases} r_1/r_\ell \leq \log^{\rho_\ell} p \\ r_1/r_{\ell+1} > \log^{(\ell-1)(\rho_\ell+1)+1} p \end{cases}$$

$$\text{Case } k : r_1/r_k \leq \log^{\rho_k} p$$

It is easy to check that Case 1 to Case k cover all possible cases. Namely if the second inequality in Case ℓ is not satisfied, we obtain:

$$r_1/r_{\ell+1} \leq \log^{(\ell-1)(\rho_\ell+1)+1} p$$

which implies using (18) that the first inequality $r_1/r_{\ell+1} \leq \log^{\rho_{\ell+1}} p$ in Case $\ell+1$ must be satisfied. Since the first inequality in Case 1 is automatically satisfied, this implies that one of Case ℓ must apply, for some $1 \leq \ell \leq k$.

For any Case ℓ we can apply Lemma 3 with $\rho = \rho_\ell$; the sufficient condition for extracting a non-trivial factor of N is then:

$$r_1 = \Omega(\log^{2 \cdot (\ell-1) \cdot (\rho_\ell+1)+1} p)$$

Since $\ell \leq k$ and from (18) we have $\rho_\ell \leq \rho_k$, to handle all possible Case ℓ we obtain the sufficient condition $r_1 = \Omega(\log^{\theta_k} p)$ where:

$$\theta_k = 2(k-1)(\rho_k + 1) + 1 \quad (19)$$

Therefore we can recover a non-trivial factor of N under the condition $r_1 = \Omega(\log^{\theta_k} p)$.

Eventually we give a formula for θ_k . From (18) we have:

$$\begin{aligned}
\rho_\ell &= (\ell - 2)(\rho_{\ell-1} + 1) + 1 = (\ell - 2)((\ell - 3)(\rho_{\ell-2} + 1) + 2) + 1 \\
&= (\ell - 2)(\ell - 3)(\rho_{\ell-2} + 1) + 2(\ell - 2) + 1 \\
&= (\ell - 2)(\ell - 3)((\ell - 4)(\rho_{\ell-3} + 1) + 2) + 2(\ell - 2) + 1 \\
&= (\ell - 2)(\ell - 3)(\ell - 4)(\rho_{\ell-3} + 1) + 2(\ell - 2)(\ell - 3) + 2(\ell - 2) + 1
\end{aligned}$$

More generally, using $\rho_2 = 1$ we obtain for all $\ell \geq 3$:

$$\rho_\ell = 1 + 2 \sum_{i=1}^{\ell-2} \prod_{j=i}^{\ell-2} j$$

From (19) we have $\theta_2 = 5$ and for all $k \geq 3$:

$$\theta_k = 2(k-1)(\rho_k + 1) + 1 = 2(k-1) \left(2 + 2 \sum_{i=1}^{k-2} \prod_{j=i}^{k-2} j \right) + 1$$

which gives eventually:

$$\theta_k = 4(k-1) \left(1 + \sum_{i=1}^{k-2} \prod_{j=i}^{k-2} j \right) + 1 \tag{20}$$

Finally we obtain from (20):

$$\begin{aligned}
\theta_k &= 1 + 4 \cdot (k-1) \cdot \left(1 + \sum_{i=1}^{k-2} \frac{(k-2)!}{(i-1)!} \right) = 1 + 4 \cdot (k-1) \cdot \sum_{i=1}^{k-1} \frac{(k-2)!}{(i-1)!} \\
&= 1 + 4 \cdot \sum_{i=1}^{k-1} \frac{(k-1)!}{(i-1)!} = 1 + 4 \cdot \sum_{i=0}^{k-2} \frac{(k-1)!}{i!} = 4 \cdot \sum_{i=0}^{k-1} \frac{(k-1)!}{i!} - 3
\end{aligned}$$

Using $\sum_{i=0}^{\infty} 1/i! = e$ we obtain:

$$\theta_k = 4 \cdot (k-1)! \cdot \left(e - \sum_{i=k}^{\infty} \frac{1}{i!} \right) - 3$$

which gives eventually for large k :

$$\theta_k = 4e \cdot (k-1)! - 3 - o(1)$$

This terminates the proof of Theorem 8.

Remark 1. For two prime factors with $\theta_2 = 5$ we obtain the condition $r_1 = \Omega(\log^5 p)$; this is because in Lemma 3 only the BDH method is used. To get the better bound $r_1 = \Omega(\log^3 p)$ of Theorem 7 one must use both BDH and Coppersmith.

Remark 2. We provide the first values of ρ_k and θ_k in Table 1.

k	2	3	4	5	6
ρ_k	1	3	9	31	129
θ_k	5	17	61	257	1301

Table 1. Values of ρ_k and θ_k for a modulus $N = \prod_{i=1}^k p_i^{r_i}$ with k prime factors. The condition on the largest exponent r_1 is $r_1 = \Omega(\log^{\theta_k} \max_i p_i)$.

5 Experiments

We have implemented our algorithm using Magma Software V2.19-5. We considered four moduli $N = p^r q^s$ with $r = 8$, and $s = 1, 3, 5, 7$, with 128-bit primes p and q . Since in Section 3 a fraction $1/u$ of the bits of Q is guessed by exhaustive search, for each modulus N we have determined the values of α , β , a and b that minimize the quantity $\log(Q)/u$; such minimum is reached either by the BDH method (Case 1), or by the Coppersmith method (Case 2). To speed up the LLL computation we have implemented the Rounding and Chaining methods from [BCF⁺14]. This consists in applying LLL on a matrix with truncated coefficients (Rounding), and using partially LLL-reduced matrices when doing the exhaustive search (Chaining); the first LLL reduction is then costlier than the subsequent ones.

In Table 2 we give the optimal decomposition of N , using either the BDH method (Case 1) or the Coppersmith method (Case 2), with number of bits given, lattice dimension, running time LLL_f of the first LLL reduction, and running time LLL_c of subsequent LLL reductions; finally we also estimate the total running time of the factorization, by multiplying LLL_c by 2^n where n is the number of bits given.

	Method	$(p^\alpha q^\beta)^u p^a q^b$	bits given	dim.	LLL_f	LLL_c	Est. time
$N = \mathbf{p}^8 \mathbf{q}$	BDH	$p^8 q$	29	68	142 s	8.6 s	146 years
$N = \mathbf{p}^8 \mathbf{q}^3$	Copp.	$(p^2 q)^4 q^{-1}$	51	61	86 s	4.2 s	$3 \cdot 10^8$ years
$N = \mathbf{p}^8 \mathbf{q}^5$	BDH	$(p^2 q)^4 q$	55	105	115 s	1.3 s	$2 \cdot 10^9$ years
$N = \mathbf{p}^8 \mathbf{q}^7$	Copp.	$(pq)^8 q^{-1}$	38	81	676 s	26 s	$2 \cdot 10^5$ years

Table 2. Number of bits given, lattice dimension, running time LLL_f of the first LLL, running time LLL_c of subsequent LLLs, and estimated total running time.

As observed in [BDHG99] the BDH algorithm is unpractical compared to the ECM factorization algorithm [Len87]. Namely for 128-bit primes p and q and $N = p^{10} q$ the predicted runtime of ECM from [BDHG99] is only 7000 hours [BDHG99], instead of 146 years for BDH for $N = p^8 q$. As illustrated in Table 2 for integers $N = p^r q^s$ our algorithm performs even worse. However the ECM scales exponentially¹, whereas our algorithm scales polynomially. Hence for large enough primes p and q our algorithm (like BDH) must outpace ECM.

¹ The complexity of the ECM for extracting a prime factor p is $\exp((\sqrt{2} + o(1))\sqrt{\log p \log \log p})$

References

- [BCF⁺14] Jingguo Bi, Jean-Sébastien Coron, Jean-Charles Faugère, PhongQ. Nguyen, Guénaél Renault, and Rina Zeitoun. Rounding and chaining III: Finding faster small roots of univariate polynomial congruences. *IACR Cryptology ePrint Archive*, 2014, 2014.
- [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.
- [BDHG99] Dan Boneh, Glenn Durfee, and Nick Howgrave-Graham. Factoring $n = p^r q$ for large r . In *Advances in Cryptology - Proc. CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1999.
- [BM05] Johannes Blömer and Alexander May. A tool kit for finding small roots of bivariate polynomials over the integers. In *Advances in Cryptology - Proc. EUROCRYPT '05*, volume 3494 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2005.
- [Cop96a] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.
- [Cop96b] Don Coppersmith. Finding a small root of a univariate modular equation. In *Advances in Cryptology - Proc. EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165. Springer, 1996.
- [Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, 10(4):233–260, 1997. Journal version of [Cop96b,Cop96a].
- [DN00] Glenn Durfee and Phong Q. Nguyen. Cryptanalysis of the RSA schemes with short secret exponent from asiacrypt '99. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, pages 14–29, 2000.
- [HG97] Nick Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding - Proc. IMA '97*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997.
- [JM07] Ellen Jochemsz and Alexander May. A polynomial time attack on RSA with private crt-exponents smaller than $N^{0.073}$. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 395–411, 2007.
- [Len87] H.W. Lenstra. Factoring integers with elliptic curves. *Ann. Math.*, 126:649–673, 1987.
- [LKYL00] Seongan Lim, Seungjoo Kim, Ikkwon Yie, and Hongsub Lee. A generalized takagi-cryptosystem with a modulus of the form $p^r q^s$. In *Progress in Cryptology - INDOCRYPT 2000, First International Conference in Cryptology in India, Calcutta, India, December 10-13, 2000, Proceedings*, pages 283–294, 2000.
- [LLL82] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.
- [May04] Alexander May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 213–219, 2004.
- [NS09] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. of Computing*, 39(3):874–903, 2009.
- [NSV11] Andrew Novocin, Damien Stehlé, and Gilles Villard. An LLL-reduction algorithm with quasi-linear time complexity: extended abstract. In *Proc. STOC '11*, pages 403–412. ACM, 2011.
- [Tak97] Tsuyoshi Takagi. Fast rsa-type cryptosystems using n-adic expansion. In *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, pages 372–384, 1997.
- [Tak98] Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, pages 318–326, 1998.

A Coppersmith's Second Theorem for Factoring $N = p^r q^s$

A natural approach to factor $N = p^r q^s$ would be to write $N = (P + x)^r (Q + y)^s$ and apply Coppersmith's second theorem for finding small roots of bivariate polynomials over \mathbb{Z} . Here we show that this approach does not work. We first recall Coppersmith's second theorem.

Theorem 9 (Coppersmith [Cop97]). *Let $f(x, y)$ be an irreducible polynomial in two variables over \mathbb{Z} , of maximum degree δ in each variable separately. Let X and Y be upper bounds on the desired integer solution (x_0, y_0) , and let $W = \max_{i,j} |f_{ij}| X^i Y^j$. If $XY < W^{2/(3\delta)}$, then in time polynomial in $(\log W, 2^\delta)$, one can find all integer pairs (x_0, y_0) such that $f(x_0, y_0) = 0$, $|x_0| \leq X$, and $|y_0| \leq Y$.*

For $N = p^r q^s$ we write $p = P + x_0$ and $q = Q + y_0$ where $|x_0| \leq X$ and $|y_0| \leq Y$ for some y , and we assume that P and Q are given. Therefore (x_0, y_0) is a small root over \mathbb{Z} of the bivariate polynomial:

$$f(x, y) = (P + x)^r (Q + y)^s$$

Assuming that $r > s$, the degree of $f(x, y)$ is at most r separately in x and y . Therefore we must have:

$$XY < W^{2/(3r)}$$

where $W = P^r Q^s \simeq N$. Assuming $r \simeq s$, we have:

$$W^{2/(3r)} \simeq N^{2/(3r)} = p^{2/3} q^{2s/(3r)} \simeq (pq)^{2/3}$$

Therefore one should take the bounds $X \simeq p^{2/3}$ and $Y \simeq q^{2/3}$. This implies that to recover p and q in polynomial time we must know at least $1/3$ of the high-order bits of p and $1/3$ of the high-order bits of q . Since this is a constant fraction of the bits of p and q , Coppersmith's second theorem does not enable to factor $N = p^r q^s$ in polynomial-time.

We stress that the above reasoning does not prove that Coppersmith's bivariate technique will not work. Namely as shown in [BM05] to obtain the optimal bound one must use the right construction corresponding to $f(x, y)$'s Newton polygon. However for $r \simeq s$ the polynomial $f(x, y)$ has (almost) the same degree in x and y separately, so it is natural to use the bounds from Coppersmith's original bivariate theorem (Theorem 9) as above; this corresponds to the Rectangle construction in [BM05].

B Coppersmith's First Theorem

In this section we recall the main steps of Coppersmith's algorithm for finding small roots of univariate modular equations modulo N , corresponding to Theorem 3. We follow the classical approach by Howgrave-Graham [HG97].

Let $f(x)$ be a polynomial of degree r , with small unknown x_0 such that

$$f(x_0) \equiv 0 \pmod{N}.$$

One considers the following polynomials $g_{i,j}(x)$, where $m \geq 1$ is a given parameter:

$$g_{i,j}(x) = x^j \cdot N^{m-i} f^i(x)$$

for all i and j such that $0 \leq i < m$ and $0 \leq j < r$, and $j = 0$ for $i = m$. We have:

$$g_{i,j}(x_0) \equiv 0 \pmod{N^m}$$

Let $h(x)$ be a linear combination of the $g_{i,j}(x)$; therefore we must have

$$h(x_0) \equiv 0 \pmod{N^m} \tag{21}$$

Let X be such that $|x_0| < X$. If the coefficients of $h(x)$ are sufficiently small, since x_0 is small we will have $|h(x_0)| < N^m$ and therefore Equation (21) will hold over \mathbb{Z} . The root x_0 of $h(x_0) = 0$ can then be recovered using a classical root-finding algorithm. The condition is formalized by the following lemma due to Howgrave-Graham [HG97]. Given a polynomial $h(x) = \sum_i h_i x^i$ we define $\|h(x)\|^2 = \sum_i |h_i|^2$.

Lemma 4 (Howgrave-Graham). *Let $h(x) \in \mathbb{Z}[x]$ be the sum of at most d monomials. Assume that $h(x_0) \equiv 0 \pmod{N^m}$ where $|x_0| \leq X$ and $\|h(xX)\| < N^m/\sqrt{d}$. Then $h(x_0) = 0$ over the integers.*

Proof. We have:

$$\begin{aligned} |h(x_0)| &= \left| \sum h_i x_0^i \right| = \left| \sum h_i X^i \left(\frac{x_0}{X}\right)^i \right| \leq \sum \left| h_i X^i \left(\frac{x_0}{X}\right)^i \right| \\ &\leq \sum |h_i X^i| \leq \sqrt{d} \|h(xX)\| < N^m. \end{aligned}$$

Since $h(x_0) \equiv 0 \pmod{N^m}$, this gives $h(x_0) = 0$. □

It remains to show how to obtain $h(x)$ such that $\|h(xX)\| < N^m/\sqrt{d}$. We consider the matrix M of dimension $d = rm + 1$ whose row vectors are the coefficients of the polynomials $g_{i,j}(xX)$. This matrix is reduced using the well-known LLL algorithm [LLL82] or an analogous algorithm with improved complexity [NS09,NSV11]. Since the matrix M is triangular, the determinant of M is the product of its diagonal elements:

$$\det M = N^{(m+1)(d-1)/2} X^{d(d-1)/2}.$$

From Theorem 1, the first resulting polynomial $v(xX)$ of the reduced matrix is such that $\|v(xX)\| \leq 2^{(d-1)/4} (\det M)^{1/d}$. As a consequence, we get:

$$\|v(xX)\| \leq 2^{(d-1)/4} N^{(m+1)(d-1)/2d} X^{(d-1)/2}.$$

In order to fulfill the condition $\|v(xX)\| < N^m/\sqrt{d}$, we get the following condition on the upper-bound X , under which the solution $|x_0| < X$ can be retrieved:

$$X < \frac{1}{4} \cdot N^{\frac{1}{r} - \frac{1}{rd}}.$$

Eventually by using a dimension $d = \mathcal{O}(\log N)$ and performing exhaustive search on a constant number of high-order bits of x_0 , one obtains the sufficient condition $X < N^{1/r}$; this proves Theorem 3.

C The BDH Method for Factoring $N = p^r q$

In this section we recall the main steps of the BDH method from Theorem 6; we refer to [BDHG99] for more details. Let $N = p^r q$. Assume that we are also given an integer V such that $p = V + x_0$ where the high-order bits of V are the same as the high-order bits of p , and x_0 is a small unknown. One considers the polynomial $f(x) = (V + x)^r$ which satisfies:

$$f(x_0) \equiv (V + x_0)^r \equiv 0 \pmod{p^r}$$

Moreover we also have:

$$N \equiv 0 \pmod{p^r}$$

Therefore for a given integer m one considers the polynomials

$$g_{ik}(x) = N^{m-k} x^i f^k(x)$$

for $0 \leq k \leq m$ and $i \geq 0$, and we have for all k, i :

$$g_{ik}(x_0) \equiv N^{m-k} \cdot x_0^i \cdot f^k(x_0) \equiv 0 \pmod{p^{rm}}$$

Let X be a bound on x_0 . One considers the lattice L spanned by the coefficient vectors of $g_{ik}(xX)$ for $0 \leq k \leq m-1$ and $0 \leq i \leq r-1$, and also $g_{ik}(xX)$ for $k = m$ and $0 \leq i \leq d - mr - 1$, where d is a parameter which is actually the lattice dimension. Since the matrix basis of the lattice is triangular, the determinant of the lattice is the product of the diagonal entries, which gives:

$$\det L = \left(\prod_{k=0}^{m-1} \prod_{i=0}^{r-1} N^{m-k} \right) \left(\prod_{j=0}^{d-1} X^j \right) < N^{rm(m+1)/2} X^{d^2/2}$$

By applying the LLL algorithm on the previous matrix, we obtain a short vector $v(xX)$ such that:

$$\|v(xX)\|^d \leq 2^{d^2/2} \det L \leq N^{rm(m+1)/2} (2X)^{d^2/2}$$

From Lemma 4 and omitting the \sqrt{d} factor, we must have $\|v(xX)\| \leq p^{rm}$, which gives the condition:

$$(2X)^{d^2/2} < p^{rmd} N^{-rm(m+1)/2}$$

We assume that $q < p^c$ for some $c > 0$. This gives $N < p^{r+c}$, which gives the condition:

$$(2X)^{d^2/2} < p^{rmd - r(r+c)m(m+1)/2} .$$

We wish to maximize the value $md - (r+c)m(m+1)/2$. Working through tedious arithmetic allows to find that the maximum is well approximated by $\frac{d^2}{2(r+c)} \left(1 - \frac{r+c}{d}\right)$, which is reached for $m = \frac{d}{r+c} - \frac{1}{2}$ (we assume that $m \in \mathbb{N}$ by an appropriate choice of r and d). Therefore, this results in the following condition on X :

$$X < p^{1 - \frac{c}{r+c} - 2\frac{r}{d}},$$

which proves Lemma 3.3 from [BDHG99].

Lemma 5 ([BDHG99]). *Let $N = p^r q$ be given, and assume $q < p^c$ for some c . Furthermore assume that P is an integer satisfying*

$$|P - p| < p^{1 - \frac{c}{r+c} - 2\frac{r}{d}}$$

Then the factor p may be computed from N , r , c and P by an algorithm whose running time is dominated by the time it takes to run LLL on a lattice of dimension d .

In [BDHG99] the authors take $d = 2r(r + c)$, which gives:

$$|P - p| < p^{1 - \frac{c+1}{r+c}}$$

and therefore to factor $N = p^r q$ it suffices to perform exhaustive search on a fraction $(c + 1)/(r + c)$ of the bits of p , and the running time becomes:

$$\exp\left(\frac{c+1}{r+c} \cdot \log p\right) \cdot \text{poly}(\log N)$$

which proves Theorem 5.

D Proof of Theorem 6

We start from Lemma 5 from [BDHG99] whose proof is briefly recalled in the previous section. Note that in Lemma 5 the integers p and q can be any integers ≥ 2 , not necessarily primes; namely the proof of Lemma 5 does not depend on p and q being primes.

Instead of taking $d = 2r(r + c)$ as in the previous section, we now take $d = 2\lceil r \cdot \log p \rceil$, which gives:

$$|P - p| < p^{1 - \frac{c}{r+c} - \frac{1}{\log p}}$$

and therefore to factor $N = p^r q$ it suffices to perform exhaustive search on a fraction $c/(r+c) < c/r$ of the bits of p , which gives a running time:

$$\exp\left(\frac{c}{r} \cdot \log p\right) \cdot \text{poly}(\log N)$$

Moreover we can take c such that $(p^c)/2 < q < p^c$, which gives $p^c < 2q$, which gives $c \log p < \log q + \log 2$. Therefore the running time is:

$$\exp\left(\frac{\log q}{r}\right) \cdot \text{poly}(\log N)$$

and therefore a sufficient condition for polynomial-time factorization of $N = p^r q$ is $r = \Omega(\log q)$; this proves Theorem 6.