On Factoring Arbitrary Integers with Known Bits

Mathias Herrmann, Alexander May

Faculty of Computer Science, TU Darmstadt, 64289 Darmstadt, Germany
herrmann@rbg.informatik.tu-darmstadt.de, may@informatik.tu-darmstadt.de

Abstract: We study the *factoring with known bits problem*, where we are given a composite integer $N = p_1 p_2 \dots p_r$ and oracle access to the bits of the prime factors p_i, $i = 1, \dots, r$. Our goal is to find the full factorization of N in polynomial time with a minimal number of calls to the oracle. We present a rigorous algorithm that efficiently factors N given $(1 - \frac{1}{r} H_r) \log N$ bits, where H_r denotes the r^{th} harmonic number.

1 Introduction

One of the most challenging tasks in algorithmic number theory is to efficiently find the factorization of a composite number N. The security of the most popular public key cryptosystem RSA is based on the difficulty of the factorization problem. Thus, it is no surprise that considerable efforts have been made to lower the computational complexity of factorization algorithms. In the Turing machine model, the currently best algorithms – the Elliptic Curve Method and the Number Field Sieve – require subexponential time.

On the other hand, another interesting line of research established in the last two decades that deals with relaxations of the factorization problem which are solvable in polynomial time. A natural relaxation is to provide additional limited oracle access to the prime factors' bits. This relaxation is motivated by cryptographic practice, where several side-channels are known that leak bits of the factors.

Rivest and Shamir [RS86] showed in 1985, that for an RSA-modulus $N = pq$ an amount of $\frac{1}{3} \log N$ of the bits of p is sufficient to factor N. This result was improved by Coppersmith [Cop96] in 1996 to $\frac{3}{10} \log N$ bits, and in 1997 again by Coppersmith [Cop97] to $\frac{1}{4} \log N$ bits.

In 1999, Boneh, Durfee and Howgrave-Graham [BDHG99] generalized the Coppersmith result to moduli of the form $N = p^k q$. They showed that $\frac{k}{(k+1)^2} \log N$ bits are sufficient to find the factorization of N in polynomial time. One should notice that this result coincides with the one of Coppersmith for the RSA case, where $k = 1$.

Recently, Santoso, Kunihiro, Kanayama and Ohta [SKKO06] generalized the factorization with known bits approach to square-free moduli $N = p_1 \dots p_r$, where all prime factors $p_i, i = 1, \dots, r$ have the same bit-size. Santoso et al showed that the full factorization of N can be found given $\left(1 - \frac{2}{r+2}\right) \log N$ bits.

We would like to remark that this result does *not* coincide with Coppersmith's bound for the RSA case, in which $r = 2$. Moreover, as opposed to the results of Coppersmith and Boneh, Durfee, Howgrave-Graham the approach in [SKKO06] is not rigorous. Santoso et al. model the factorization problem as a lattice-based root finding problem for an r-variate polynomial. The authors use a heuristic algorithm of Coron [Cor04] for finding a root of a multivariate polynomial equation. The root in turn yields all the prime factors.

Our contribution: We present a *rigorous* algorithm for factoring square-free integers $N = p_1 p_2 \dots p_r$. As opposed to [SKKO06], we solve the factorization problem iteratively by finding one prime factor of N in each iteration. This allows us to model the factorization as a root finding problem for modular univariate polynomials. Therefore, we can use Coppersmith's rigorous algorithm for finding the roots of univariate polynomial equations.

Our factorization algorithm requires only a total of $(1 - \frac{1}{r}H_r) \log N$ bits, where $H_r = \sum_{i=1}^{r} \frac{1}{i}$ is the r^{th} harmonic number. This improves upon the bound of Santoso et al. for all r. Moreover, for the RSA-case $r = 2$ the bound coincides with the Coppersmith bound of $\frac{1}{4} \log N$ bits. The complexity of our factorization algorithm is polynomial in $(\log N, r)$.

We also consider the case where the prime factors p_i are not of the same bit-size. We show that in our iterative process of factoring, it is best to recover in each iteration the smallest prime factor p_i. The smaller p_i is relative to the other factors, the less bits we need to discover it. Consequently, the case where all prime factors are of the same bit-size turns out to be the worst case for our algorithm in terms of the number of oracle calls. Thus, $(1 - \frac{1}{r}H_r) \log N$ bits is an upper bound for general integers of the form $N = p_1 p_2 \dots p_r$.

Furthermore, our algorithm easily extends to integers that contain arbitrary prime powers, i.e. the r primes factors p_i must not necessarily be distinct. Once again, we can show that our upper bound holds, and the more multiplicities N has, the less oracle calls are required in our algorithm.

We would like to point out that our results have implications for fast variants of RSA, which make use of multiprime RSA moduli. Application for such moduli have been proposed by Boneh, Shacham [BS02] in order to speed up the RSA decryption/signing process.

2 The Factorization Algorithm

At Eurocrypt 96, Coppersmith presented a method for finding small roots of univariate modular polynomials [Cop96]. We will use the result in the reformulation of May [May07]:

Theorem 1. *Let N be an integer of unknown factorization with a divisor $b \geq N^{\beta}$. Let $f_b(x)$ be a univariate, monic polynomial of degree δ. Furthermore, let c_N be a function that is upper-bounded by a polynomial in* $\log N$.*Then we can find all solutions x_0 for the equation $f_b(x) = 0 \bmod b$ with*

$$|x_0| \leq c_N N^{\frac{\beta^2}{\delta}} \tag{1}$$

in polynomial time in $(\log N, \delta)$.

Now we present a method to factor an integer of the form $N = p_1 p_2 \cdot \ldots \cdot p_r$ given access to an oracle which delivers bits of $r-1$ of the primes, applying the method from Coppersmith. We compute the prime factors in an iterative manner. I.e. we start by looking at a polynomial with a small root modulo one of the prime factors of N, w.l.o.g. p_1, and continue with a polynomial which has a small root modulo a prime factor of $N_2 = \frac{N}{p_1}$ and so on.

Algorithm 1 Factorization Algorithm

```
Input: $r, N = p_1 \cdots p_r$ w.l.o.g. $p_1 \le p_2 \le \ldots \le p_r$
  $N' \leftarrow N$
  for $i = 1$ to $r-1$ do
    $\tilde{p_i} \leftarrow$ Call Oracle for $\frac{r-i}{r(r-i+1)} \log N$ most significant bits of prime factor $p_i$ of $N'$
    $p_i \leftarrow$ Apply Theorem 1 with the polynomial $f_{p_i} = \tilde{p_i} + x$
    $N' \leftarrow \frac{N'}{p_i}$
  end for
Output: $p_1, \ldots, p_r$
```

Theorem 2. *Let N be a composite square-free integer with prime factors $p_1, \ldots, p_r$ of the same bit-size. If we have approximations of $p_1, \ldots, p_{r-1}$ with*

$$|p_i - \tilde{p_i}| \le N^{\frac{r-i}{r(r-i+1)}} \tag{2}$$

then we can factor N in polynomial time in $(\log N, r)$.

The construction of $\tilde{p}_1, \ldots \tilde{p}_{r-1}$ requires $\left(1 - \frac{1}{r} H_r\right) \log N$ calls to the oracle.

Proof. The polynomial $f_1 = \tilde{p_1} + x$ has the small root $p_1 - \tilde{p_1}$ modulo p_1. Our goal is to find this root, which yields the prime factor p_1. To apply Theorem 1, we need to bound the size of the divisor used. For at least one of the prime factors we have $p_j > N^{\frac{1}{r}}$. From the assumption we made, we know $p_i \ge \frac{1}{2} N^{\frac{1}{r}}$ for all i, since all factors have the same bit-size.

To obtain a bound β, we rewrite $\frac{1}{2} N^{\frac{1}{r}} = N^{\frac{1}{r} - \frac{1}{\log_2 N}}$. Hence we define $\beta = \frac{1}{r} - \frac{1}{\log_2 N}$. The degree of f_1 equals 1 and with the parameter $c_N = 4^{\frac{1}{r}}$ we obtain the following upper bound on the size of the roots

$$4^{\frac{1}{r}} N^{\frac{\beta^2}{\delta}} = 4^{\frac{1}{r}} N^{\frac{1}{r^2} - \frac{2}{r \log_2 N} + \frac{1}{\log_2^2 N}} \ge 4^{\frac{1}{r}} \left(\frac{1}{4}\right)^{\frac{1}{r}} N^{\frac{1}{r^2}} = N^{\frac{1}{r^2}} \tag{3}$$

In order to recover p_1 we need to know $\frac{1}{r} - \frac{1}{r^2} = \frac{r-1}{r^2} \log N$ most significant bits of p_1. Given p_1 we can simplify N to $N_2 = \frac{N}{p_1} = p_2 \ldots p_r$. Now consider the polynomial $f_2 = \tilde{p}_2 + x_2$. We know that f_2 has a small root modulo a divisor of N_2, namely p_2. Along the lines of f_1 we establish a lower bound on the size of the divisor. But in this case, the size is different, since we don't consider N but N_2. Analog to the above, we obtain a bound on the size of the divisor by

$$p_2 = \frac{N_2}{p_3 \ldots p_r} \ge \frac{1}{2} N_2^{\frac{1}{r-1}} = N_2^{\frac{1}{r-1} - \frac{1}{\log_2 N_2}} \tag{4}$$

With $\beta_2 = \frac{1}{r-1} - \frac{1}{\log_2 N_2}$, $\delta_2 = 1$ we compute

$$N_2^{\frac{\beta_2^2}{\delta_2}} = N_2^{\frac{1}{(r-1)^2} - \frac{2}{(r-1)\log_2 N_2} + \frac{1}{\log_2^2 N_2}} \geq \left(\frac{1}{4}\right)^{\frac{1}{r-1}} N_2^{\frac{1}{(r-1)^2}} = \left(\frac{1}{4}\right)^{\frac{1}{r-1}} N_2^{\frac{1}{(r-1)^2}} \tag{5}$$

To express N_2 in terms of N, we use a similar argumentation as before. Since $p_j \leq N^{\frac{1}{r}}$ for at least one j, we can upper bound $p_i \leq 2N^{\frac{1}{r}}$ for all i as they have all the same bit-size. Then $N_2 = \frac{N}{p_1} \geq \frac{1}{2}N^{1-\frac{1}{r}}$. Continuing equation (5) we obtain

$$\left(\frac{1}{4}\right)^{\frac{1}{r-1}} N_2^{\frac{1}{(r-1)^2}} \geq \left(\frac{1}{4}\right)^{\frac{1}{r-1}} \left(\frac{1}{2}N^{\frac{r-1}{r}}\right)^{\frac{1}{(r-1)^2}} \geq \left(\frac{1}{4}\right)^{\frac{1}{r-2}} N^{\frac{1}{r(r-1)}}$$

Hence we may apply Theorem 1 with $c_{N_2} = 4^{\frac{1}{r-2}}$ and obtain

$$4^{\frac{1}{r-2}} N_2^{\frac{\beta_2^2}{\delta}} \geq N^{\frac{1}{r(r-1)}} \tag{6}$$

Thus we require $\frac{1}{r} - \frac{1}{r(r-1)} = \frac{r-2}{r(r-1)} \log N$ most significant bits of p_2.

For the i-th prime we have $p_i = \frac{N_i}{p_{i+1} \cdot \ldots \cdot p_r} \geq \frac{1}{2} N_i^{\frac{1}{r-i+1}} = N_i^{\frac{1}{r-i+1} - \frac{1}{\log_2 N_i}}$ and

$$N_i = \frac{N}{p_1 \cdot \ldots \cdot p_{i-1}} \geq \begin{cases} (\frac{1}{2})^{i-1} N^{\frac{r-i+1}{r}} & \text{if } i-1 < \frac{r}{2} \\ (\frac{1}{2})^{r-(i-1)} N^{\frac{r-i+1}{r}} & \text{if } i-1 \geq \frac{r}{2} \end{cases} \tag{7}$$

Then

$$N_i^{\frac{\beta^2}{\delta}} \geq \begin{cases} (\frac{1}{4})^{\frac{2(r-i+1)+(i-1)}{2(r-i+1)^2}} N^{\frac{1}{r(r-i+1)}} & \text{if } i-1 < \frac{r}{2} \\ (\frac{1}{4})^{\frac{3}{2(r-i+1)}} N^{\frac{1}{r(r-i+1)}} & \text{if } i-1 \geq \frac{r}{2} \end{cases} \tag{8}$$

By using the case differentiation, we prevent c_{N_i} from being exponential in r. It is based on the fact, that at most $\frac{r}{2}$ of the primes can be of size $\frac{1}{2}N^{\frac{1}{r}}$. We then have to choose c_{N_i} as $4^{\frac{2(r-i+1)+(i-1)}{2(r-i+1)^2}}$ respectively $4^{\frac{3}{2(r-i+1)}}$. The maximum of c_{N_i} in the interval $1 \leq i \leq r-1$ is $2\sqrt{2}$. Therefore the requirement of Theorem 1 for c_N to be polynomial in $\log N$ is fulfilled.

In this fashion we formulate bounds on the required approximations for $r-1$ of the prime factors of N (we get the last one for free). Eventually we are interested in the total number of bits required to factor the composite number N. Summing up the values for the $\tilde{p_i}$, we need the following number of oracle calls:

$$\frac{1}{r} \sum_{i=1}^{r} \frac{r-i}{r-i+1} \log N = \left(1 - \frac{1}{r} H_r\right) \log N \tag{9}$$

□

Our algorithm improves on a recent result from Santoso, Kunihiro, Kanayama, Ohta [SKKO06]. They require $\left(\frac{1}{r+2} + \frac{\epsilon}{r(r-1)}\right) \log N$ high bits for each of the prime factors, which sums up to $\frac{r}{r+2} \log N$ bits in total. Additionally our algorithm is rigorous, i.e. it does not depend on a heuristic assumption like the one in [SKKO06].

3 Unbalanced Prime Factors

In the previous section we assumed the prime factors of a squarefree integer to be balanced. In this section we will show that this is actually the worst case, i.e. the number of required bits gets smaller if the prime factors are unbalanced. First we notice that for unbalanced prime factors the order in which they are processed does make a difference.

We will argue that it is a better choice to use the small prime factors first. Suppose we have an integer $N = p_1 p_2 \dots p_r$ and $p_1 = N^{b_1}, p_2 = N^{b_2}$. We will examine in which order the first two prime factors should be processed.

Taking first p_1 and then p_2, the number of required oracle calls is

$$b_1 - b_1^2 + b_2 - \frac{b_2^2}{1-b_1} + Remaining \tag{10}$$

Switching the order of the first two primes, we require

$$b_2 - b_2^2 + b_1 - \frac{b_1^2}{1-b_2} + Remaining \tag{11}$$

oracle calls. The *Remaining* part is equal for both cases. Now we are interested under what condition we get a smaller number of required bits using first p_1, then p_2. Thus,

$$b_1 - b_1^2 + b_2 - \frac{b_2^2}{1-b_1} \leq b_2 - b_2^2 + b_1 - \frac{b_1^2}{1-b_2}$$

This can be reduced to the condition

$$b_1 \leq b_2 \tag{12}$$

Hence we require less oracle calls if we start with the smaller factor. Using the same argumentation on other pairs of primes we eventually obtain that the number of oracle calls is minimal if the prime factors are processed from smallest to largest. This reflects in the precondition $p_1 \leq p_2 \leq \dots \leq p_r$ of Algorithm 1.

Now we can state the main theorem of this section.

Theorem 3. *The result*

$$\left(1 - \frac{1}{r} H_r\right) \log N \tag{13}$$

from equation (9) *is an upper bound for the number of required bits to factor an arbitrary square-free composite integer.*

Proof. Proof by induction over the number of prime factors:

Base Case: Let $r = 2$ and $N = p_1 p_2$. Suppose $p_1 = N^{b_1}$.

From Theorem 1 we obtain that we can recover $b_1^2 \log N$ bits of p_1 and therefore we require $b_1 - b_1^2 \log N$ bits. The maximum of $b_1 - b_1^2$ is reached for $b_1 = \frac{1}{2}$.

Hypothesis:

For a composite number with $r-1$ distinct prime factors the worst case of oracle calls is in the balanced case.

Inductive Step:

Let $N = p_1 p_2 \dots p_r$. Assume $p_1 = N^{b_1}$ is the smallest factor, then $b_1 \le \frac{1}{r}$.

We need to find the maximum number of required bits. I.e. we need to maximize $b_1 - b_1^2 + bits\ required\ for\ the\ rest$ with the constraint $0 \le b_1 \le \frac{1}{r}$. From the induction hypothesis we derive that the number of bits for the remaining part of N is in the worst case $\frac{1}{r-1}\sum_{i=1}^{r-1}\frac{r-1-i}{r-i}\log N_2$, where $N_2 = \frac{N}{p_1}$. Expressing N_2 in terms of N, we obtain the following function to optimize

$$b_1 - b_1^2 + (1-b_1)\frac{1}{r-1}\sum_{i=1}^{r-1}\frac{r-1-i}{r-i} \tag{14}$$

We achieve the maximum value in the intervall $0 \le b_1 \le \frac{1}{r}$ for $b_1 = \frac{1}{r}$. N_2 is then of size $N^{\frac{r-1}{r}}$ and has by the hypothesis $r-1$ balanced prime factors. Thus, each prime factor is of size N^b with $b = \frac{r-1}{r}\frac{1}{r-1} = \frac{1}{r}$.

This completes the proof. □

4 Prime Powers

Theorem 4. *The result*

$$\left(1 - \frac{1}{r}H_r\right)\log N \tag{15}$$

from equation (9) *is an upper bound for the number of required bits to factor an arbitrary composite integer.*

As in the proof for the unbalanced case we will need to begin with the smallest of the prime powers. In this case however it is not that easy to see what smaller means. Therefore we will define the following:

Definition 1. *A prime power p^k is said to be* smaller *than a prime power q^l if the number of oracle calls to recover p is less.*

We will now show by induction that balanced prime factors with exponents $k_i = 1$ are the worst case in the sense of required oracle calls.

Proof. **Base Case:** $r = 2$

Let $N = p_1^k p_2^l$ and $p_1 = N^x, p_2 = N^y$. Suppose p_1^k is the smallest prime power, then $x - kx^2 \le y - ly^2$, which reduces in this case to $l \le k$, since we have $kx + ly = 1$. The number of oracle calls our algorithm requires equals $x - kx^2$. The maximum of $x - kx^2$ is obtained for $x = \frac{1}{2} = \frac{1}{r}$ and $k = 1$. Since $l \le k$ it follows $l = 1$ and $y = \frac{1}{2}$.

Hypothesis:

For a composite number with $r-1$ prime powers the worst case of oracle calls is for exponents $k_i = 1$ and balanced prime factors.

Inductive Step:

Let $N = p_1^{k_1} p_2^{k_2} \dots p_r^{k_r}$. Assume $p_1^{k_1}$ is the *smallest* factor.

As in the proof for unbalanced prime factors, we seek for the maximum of required oracle calls. The number of oracle calls is $f(x) = x - k_1 x^2 + remaining$. By induction hypothesis, the number of oracle calls of the remaining, is in the worst case of exponents $k_i = 1$ equal to $\frac{1}{n-k_1} \sum_{i=1}^{n-k_1-1} \frac{n-k_1-i-1}{n-k_1-i} \log N_2$, with $N_2 = \frac{N}{p_1^{k_1}} = N^{1-k_1 x}$. Expressing N_2 in terms of N gives the final function to be maximized

$$f(x) = x - k_1 x^2 + (1-x) \frac{1}{n-k_1} \sum_{i=1}^{n-k_1-1} \frac{n-k_1-i-1}{n-k_1-i}$$

To obtain the correct bound, we need to formulate the side condition of $p_1^{k_1}$ being the smallest factor. I.e. the number of required oracle calls is less than for the smallest factor of the rest. By the hypothesis the remaining primes are balanced and have exponents $k_i = 1$. The size of $p_i (i \geq 2)$ is therefore equal to $N_2^{\frac{1}{n-k_1}} = N^{\frac{1-k_1 x}{n-k_1}}$ and the number of oracle calls for the first prime of the remaining is $\frac{1-k_1 x}{n-k_1} - \left(\frac{1-k_1 x}{n-k_1}\right)^2 \log N$. The required side condition is therefore:

$$\begin{aligned} x - k_1 x^2 &\leq \frac{1-k_1 x}{n-k_1} - \left(\frac{1-k_1 x}{n-k_1}\right)^2 \\ &\dots \\ 0 &\leq \frac{n-k_1-1}{(n-k_1)^2 - k_1} - x \end{aligned} \tag{16}$$

Using an appropriate method for solving optimization problems with inequalities as side conditions (e.g. Karush-Kuhn-Tucker), the computation shows that the maximum is attained for $x = \frac{1}{n}$ and $k_1 = 1$.

Dividing N by p_1 leaves us with $N_2 = N^{\frac{r-1}{r}}$. By the hypothesis the worst case for N_2 is the balanced, therefore $p_i = N^b$ with $b = \frac{r-1}{r} \cdot \frac{1}{r-1} = \frac{1}{r}$. Further by the hypothesis $k_i = 1$.

This completes the proof. □

References

[BDHG99] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring N=p^{k}q for Large r, Advances in Cryptology, CRYPTO'99. *Springer, LNCS*, 1666:326–337, 1999.

[BS02] D. Boneh and H. Shacham. Fast variants of RSA, 2002.

[Cop96] Don Coppersmith. Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known. In *EUROCRYPT*, pages 178–189, 1996.

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

[Cor04] Jean-Sébastien Coron. Finding Small Roots of Bivariate Integer Polynomial Equations Revisited. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 492–505. Springer, 2004.

[May07] Alexander May. Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey. *LLL+25 Conference in honour of the 25th birthday of the LLL algorithm*, 2007.

[RS86] RL Rivest and A. Shamir. Efficient factoring based on partial information. *EUROCRYPT'85*, pages 31–34, 1986.

[SKKO06] Bagus Santoso, Noboru Kunihiro, Naoki Kanayama, and Kazuo Ohta. Factorization of Square-Free Integers with High Bits Known. In Phong Q. Nguyen, editor, *VIETCRYPT*, volume 4341 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2006.