When e-th Roots Become Easier Than Factoring

Antoine Joux[1], David Naccache[2], and Emmanuel Thomé[3]

[1] DGA and Université de Versailles, UVSQ PRISM 45 avenue des États-Unis, F-78035 Versailles CEDEX, France
antoine.joux@m4x.org

[2] École normale supérieure, Équipe de cryptographie, 45 rue d'Ulm, F-75230 Paris CEDEX 05, France
david.naccache@ens.fr

[3] INRIA Lorraine, LORIA, CACAO – bâtiment A, 615 rue du Jardin botanique, F-54602 Villiers-lès-Nancy CEDEX, France
emmanuel.thome@normalesup.org

Abstract. We show that computing e-th roots modulo n is easier than factoring n with currently known methods, given subexponential access to an oracle outputting the roots of numbers of the form $x_i + c$.

Here c is fixed and x_i denotes small integers of the attacker's choosing.

The attack comes in two flavors:

– A first version is illustrated here by producing selective roots of the form $x_i + c$ in $L_n(\frac{1}{3}, \sqrt[3]{\frac{32}{9}})$. This matches the *special* number field sieve's (SNFS) complexity.
– A second variant computes *arbitrary* e-th roots in $L_n(\frac{1}{3}, \gamma)$ after a subexponential number of oracle queries. The constant γ depends on the type of oracle used.

  This addresses in particular the One More RSA Inversion problem, where the e-th root oracle is not restricted to numbers of a special form. The aforementioned constant γ is then $\sqrt[3]{\frac{32}{9}}$.

  If the oracle is constrained to roots of the form $\sqrt[e]{x_i + c} \bmod n$ then $\gamma = \sqrt[3]{6}$.

Both methods are faster than factoring n using the GNFS ($L_n(\frac{1}{3}, \sqrt[3]{\frac{64}{9}})$).

This sheds additional light on RSA's malleability in general and on RSA's resistance to affine forgeries in particular – a problem known to be polynomial for $x_i > \sqrt[3]{n}$, but for which no algorithm faster than factoring was known before this work.

Keywords: RSA, factoring, NFS, roots.

Work partially supported by DGA research grant 05.34.058

1 Introduction

The RSA cryptosystem [17] is commonly used for providing privacy and authenticity of digital data. A very common *historical* practice for signing with RSA was to first hash the message, add a padding pattern c and then raise the result to the power of the decryption exponent. This paradigm is the basis of numerous standards such as PKCS#1 v1.5 [18].

Let n and e denote usual RSA public parameters (with $\lceil \log_2 n \rceil = N$)[4].

In this paper we explore RSA signatures with a fixed c but without the hash function, *i.e.* modular roots of the form:

$$\sqrt[e]{c + x} \bmod n$$

We call such numbers *affine modular roots* (AMRs).

A thread of publications [15, 7, 10, 14, 4, 13] stretching over a decade progressively established that given access to an AMR-oracle, new AMRs could be forged in polynomial complexity for $x > \sqrt[3]{n}$.

No strategies faster than factoring n are known for $x < \sqrt[3]{n}$ – a case tackled here at the cost of subexponential complexity. The main novelty in this paper is that, while subexponential, our method forges new AMRs for *arbitrarily small* x (down to $x < \sqrt[\epsilon]{n}$, $\forall \epsilon > 0$) *faster than factoring* n.

Moreover, we show that access to an e-th root oracle (in particular, an AMR-oracle) even allows to compute *arbitrary* e-th roots faster than factoring n. Here, the arbitrary e-th root to be computed is not known before all oracle queries have been completed.

We achieve this by tweaking the quadratic sieve (QS) and the number field sieve (NFS) factoring algorithms.

The Results

Denoting $L_n(\alpha, c) = \exp\left(c\,(1 + o(1))\,(\log n)^\alpha\,(\log\log n)^{1-\alpha}\right)$, we show that:

- Using a QS-like algorithm, new AMRs can be computed in $L_{\frac{n}{x}}(\frac{1}{2}, 1)$ instead of the $L_n(\frac{1}{2}, 1)$ required for QS-factoring n.
- Using an NFS-like approach, we selectively produce new AMRs in $L_n(\frac{1}{3}, \sqrt[3]{\frac{32}{9}})$. This matches the *special* number field sieve's (SNFS) complexity which is substantially lower than the $L_n(\frac{1}{3}, \sqrt[3]{\frac{64}{9}})$ required to GNFS-factor n.

  Our experimental results for $N = 512$ and a recent SNFS-factoring record[5], clearly underline the insecurity of affine-padding RSA.

[4] throughout this paper, we will frequently denote by $|x|$ the bitlength of x.
[5] [1], factoring a 1039-bit number using $\simeq$ 95 Pentium-D-years at 3 GHz.

– We present a procedure for computing *arbitrary* e-th roots in $L_n(\frac{1}{3}, \sqrt[3]{\frac{32}{9}} \simeq 1.53)$, requiring a general (not only AMR) e-th root oracle.

  A more practical variant with a slightly higher complexity $L_n(\frac{1}{3}, 1.58)$ was used in the experiments reported in this paper.

– Finally, a last variant allows the computation of *arbitrary* e-th roots given access to an AMR-oracle with complexity $L_n(\frac{1}{3}, \sqrt[3]{6})$. To date, we could not make this variant practical.

Our algorithms rely on an extension of Montgomery's square root algorithm for the number field sieve [16]. If one avoids this algorithm, alternative variants exist which claim a higher complexity ($L_n(\frac{1}{3}, \sqrt[3]{\frac{9}{2}})$).

2 The Strategy – A General Outline

For the sake of simplicity assume that $|x| = \frac{N}{4}$ (generalization to smaller x sizes is straightforward). We start by writing c, as a modular ratio[6]:

$$c = \frac{a}{b} \bmod n \quad \text{where} \quad |a| = (1-s)N \quad \text{and} \quad |b| = sN$$

for some $0 < s < 1$ that will be determined later.

Noting that $c + x = \frac{a+xb}{b} \bmod n$, it is easy to derive an index calculus attack[7] as in [6][8] on numbers of the form $a + xb$, that we expect to be smooth with respect to some factor base $\mathcal{B}$. We can ascertain that $a + xb$ is partially smooth by applying a special-q strategy. Two options are possible: Either choose different partial products of size $\frac{N}{4}$ of primes belonging to $\mathcal{B}$ (denote these partial products u_i) and sieve on x_i values such that $x_i = -c \bmod u_i$ or, select as special-q primes of size $\frac{N}{4}$ and use them as the u_i in the first option. From an asymptotic standpoint, the two approaches are equivalent. In practice, the first approach can produce any given equation more than once and thus require extra bookkeeping. As for the second approach, each special-q requires one extra equation to cancel out, thereby resulting in a larger system of equations.

It remains to optimize s. To maximize the smoothness odds of $a + xb$ we require that $|a| = |xb|$ hence:

$$(1-s)N = |a| = |xb| = |x| + |b| = \frac{N}{4} + |b| = \frac{N}{4} + sN \quad \Rightarrow \quad s = \frac{3}{8}$$

In other words, we need to find multiplicative relations between numbers of size $\frac{5N}{8}$ divisible, *by construction*, by smooth factors of size $\frac{N}{4}$. All in all this

[6] *e.g.* using a continued fraction algorithm.

[7] Treat b as an extra element of the factor base, together with the primes in the basis to account for the denominator in the equations.

[8] In [6] the signing oracle is used to compute e-th roots whose combination allows to compute new e-th roots of factor-base elements.

amounts to chasing smooth numbers of size $\frac{3N}{8}$ which is easier than QS-factoring n (identical task for numbers of size $\frac{N}{2}=\frac{4N}{8}$).

More generally, when x is an $\frac{N}{t}$ bit number, the job boils down to finding smooth numbers of size $\frac{N(t-1)}{2t}$ *i.e.* QS-factoring $\frac{N(t-1)}{t}$ bit RSA moduli.

Hence, the presented strategy approaches the QS's complexity as t grows, while remaining below the QS's complexity curve[9].

Given that the quadratic-sieve is *not* the fastest factoring strategy for usual-size RSA moduli, the extension of the above strategy to the NFS is a natural question (that this paper answers positively).

NFS algorithms work by exhibiting relations between objects in two different "worlds". In some cases, we have a single number field and consider relations between integers and elements in that field. In other cases, there are two number fields. Nonetheless, with both approaches, there are two sides to consider. In this paper, the AMR-oracle is going to replace one of the two sides. Consequently, our setting is that of a single-sided NFS. This turns out to greatly improve the smoothness probability and hence the algorithm's efficiency.

We start by selecting a parameter d and finding a polynomial f of degree d having sufficiently small coefficients such that $f(c) \equiv 0 \bmod n$. Without loss of generality, we may assume that f is irreducible over $\mathbb{Q}$. Indeed, if $f = f_1 \times f_2$, either $\gcd(f_1(c), n)$ is a non-trivial factor of n, or we can use the (smaller) polynomial f_1 instead of f.

Once f is chosen, we construct the number field $K = \mathbb{Q}[\alpha]$ where α is a root of f over $\mathbb{Q}$. We now proceed as in the NFS and given integers x, we construct elements $\alpha + x \in \mathbb{Q}[\alpha]$ with smooth norm over some factor base $\mathcal{B}$. We recall that the norm of $\alpha + x$ is the absolute value of $f(-x)$. Note a major difference with NFS-factoring: indeed, we only need to smooth a *single* $\alpha + x$ for each candidate x as there is no second (or rational) side to smooth in addition. Instead, the second side is given for free by the AMR-oracle for the number corresponding to $\alpha + x$, *i.e.* for $c + x$. When the norm is smooth, we can decompose $\alpha + x$ into a product of ideals of small norm in the ring of integers $\mathcal{O}_K$ of $K = \mathbb{Q}[\alpha]$.

Once enough smooth elements are found, we write them down as rows in a matrix where each row contains the valuation of the corresponding $\alpha + x$ at each prime ideal occurring in its decomposition. We also add to each row enough character maps in order to account for the existence of units in the number field.

Then, using a sparse linear algebra algorithm, we find a linear combination of rows equal to zero modulo e. This allows us to write an e-th power in $\mathbb{Q}[\alpha]$ as a product of $\alpha + x_i$.

The final step computes the actual e-th root of this e-th power. This yields a multiplicative relation between AMRs corresponding to the $\alpha + x_i$ used in the relation. Thus, querying all these values but one yields a new AMR for the missing

[9] To sieve, it suffices to set $x_i = -c \bmod u_i$ and consider successively $a(x_i + ju_i) + b$ for $j = 1, 2, \ldots$ (note that this will pollute a logarithmic number of bits in c).

value. The e-th root can be computed in a way very similar to the NFS' square root computation phase.

Alternatively, the final step can be replaced by a more involved strategy. Namely, combining the e-th root computation with a descent procedure very similar to the individual logarithm step of discrete logarithm computations with the NFS. This enables the calculation of e-th roots of *arbitrary* values, *i.e.* not restricted to the form $c + x$, by making a small number of additional queries of the restricted form $c + x$. This option is presented in Section 4.

3 A Detailed Step-By-Step Description

3.1 Polynomial Construction:

Given a target degree d we first need to construct a polynomial f irreducible over $\mathbb{Q}$. f will then be used to define the number field $K = \mathbb{Q}[\alpha]$. The two important constraints on f are that its coefficients should be small and that we must have $f(c) = 0 \bmod n$. Since we want to minimize the average norm $f(-x)$ of numbers $\alpha + x$, it is a good idea to use a skewed polynomial. More precisely, assume that B bounds the absolute value of x, then we want to choose a bound C such that the coefficient of degree i in f has absolute value smaller than $\frac{C}{B^i}$. Assuming[10] that $B^{d(d+1)} < n$, we choose $C = \sqrt[d+1]{n}B^{d/2}$ and the polynomial f can be constructed by reducing the lattice generated by the columns of the $(d+1) \times (d+1)$ matrix

$$L = \begin{pmatrix} \Lambda \cdots & \Lambda c^d & \Lambda n \\ 1 & 0 & 0 \\ \ddots & & \vdots \\ 0 & B^d & 0 \end{pmatrix}$$

where Λ is a sufficiently large constant to guarantee that any short vector in the lattice has zero in its first coordinate. Such a short vector can be easily interpreted as a polynomial by reading the coefficient of x^i in row $i + 2$ (the coefficient should be re-normalized by a division by B^i). This polynomial clearly has c as a root modulo n. Moreover, when evaluating the polynomial at x smaller than B (in absolute value) we see that each term is bounded by the corresponding value in the initial short vector.

Since the determinant of L is $nB^{d(d+1)/2}$, we expect short-vector coefficients to be of size

$$C = \sqrt[d+1]{n}B^{\frac{d}{2}}2^{\frac{d}{4}}$$

[10] This is necessary to avoid finding zero for high degree coefficients; of course, where necessary, we can always lower B in this construction and sieve over a smaller x range (as long as enough equations are found.)

3.2 Sieving

From a sieving standpoint, there is an essential difference between our algorithm and the NFS. Indeed, our sieving has a single degree of freedom instead of two. More precisely, instead of scanning numbers of the form $a\alpha + b$ for a fixed α and arbitrary pairs of small $\{a, b\}$, we need to examine numbers of the form $\alpha + x$.

Luckily, the bound on x is large enough to compensate the absence of the second degree of freedom but this restricts our sieving technique options. Indeed, we cannot use a lattice sieve strategy and have to rely instead on a straightforward sieve-by-line algorithm. To avoid using large numbers while sieving over x, we used a special-q approach: for each special-q prime ideal $\langle q, \alpha - r\rangle$, we considered the algebraic integers $\alpha + (qx - r)$, with $x \in [-\frac{S}{q}, +\frac{S}{q}]$.

3.3 Linear Algebra and Characters

Depending on the size of e, one may either use Lanczos/Wiedeman or block Lanczos/Wiedeman approach. If e is large enough, no self-orthogonal vector appears (unless we are extremely unlucky) and the simple approach succeeds. For smaller e, a block approach is required.

When linear algebra is performed directly on the sieving phase's output, the method yields a multiplicative relation between ideals of the form:

$$\prod_i \langle \alpha + x_i \rangle^{\mu_i} = \left(\prod_j \mathfrak{p}_j^{\nu_j} \right)^e$$

Such a relation, however, is insufficient to ensure that the product $\prod_i (\alpha + x_i)^{\mu_i}$ is an e-th power in K. Obstructions may arise from the e-part of the ideal class group of $\mathcal{O}_K$, as well as from the quotient of the unit group $\mathcal{O}_K^*/(\mathcal{O}_K^*)^e$. To annihilate these obstructions we have to add characters. We require that:

$$\chi \left(\prod_i (\alpha + x_i)^{\mu_i} \right) = \sum_i \mu_i \chi(\alpha + x_i)$$

vanishes, for several (additive) character maps $\chi : K^* \to \mathbb{F}_e$. We have the following choices for character maps:

- In [19], an approximation of the e-adic logarithm is used. Such characters are easy to compute but might fail to account for the full obstruction, as they cover at most the obstruction stemming from the unit group. Should e ramify in $\mathcal{O}_K$, or $e\mathcal{O}_K$ be divisible by a prime ideal belonging to the factor base, technicalities occur but do not prevent from using these characters.
- It is also possible to follow the classical approach used for NFS-factoring [3] *i.e.* test for powers modulo primes congruent to $1 \bmod e$. The number of characters accessible thereby is infinite. To map these multiplicative characters to additive ones, a discrete logarithm modulo e must be solved, which

is trivial for small e. For larger e values (where this might be a problem) heuristic arguments indicate that characters of the first kind would suffice anyway [19].

A typical drawback of characters is that they add a dense part to the relation matrix, which might cause a slight performance penalty. In the particular case we are interested in (just as in NFS-factoring) it is possible to perform the linear algebra *without* the character columns, produce several row dependencies and do a second reduction to recombine these dependencies into dependencies with vanishing characters.

If we elect to adopt the latter idea it becomes particularly advisable to use block algorithms for the linear algebra, since these algorithms output several vectors of the null-space simultaneously.

The linear algebra step also gives us the opportunity to check that K's class number is co-prime to e (to avoid possible technical problems *infra*). We do so by checking that the rank of the relation matrix is not abnormally low modulo e. This extra check is achieved in the same complexity and is therefore ignored hereafter. Moreover, when e is a large prime, we do not need to test anything, since the probability that e divides the class number is negligible.

3.4 Root Extraction

The linear algebra stage yields a product of algebraic integers $\pi = \prod(\alpha + x_i)^{\mu_i}$ which is known to be an e-th power in K since $\chi(\pi) = 0$ for satisfyingly many characters χ. This allows us to compute an e-th root in $\mathbb{Z}_n$ for any $c + x_{i'}$, as long as the corresponding exponent $\mu_{i'} \neq 0 \bmod e$. To do so, we first have to raise π to the power of $\mu_{i'}^{-1} \bmod e$. In other words, we can assume without loss of generality that $\mu_{i'} = 1$.

When e is small, the computation of the e-th root of π can be done using a straightforward generalization of Montgomery's square root algorithm [16].

Once the e-th root $R(\alpha)$ is computed, we have:

$$(c + x_{i'}) \prod_{i \neq i'} (c + x_i)^{\mu_i} = R(c)^e \bmod n,$$

$$\textit{i.e.:} \qquad (c + x_{i'})^d = R(c) \prod_{i \neq i'} (c + x_i)^{-\mu_i} \bmod n.$$

One might question the applicability of Montgomery's algorithm to very large values of e. Our computations in appendix A indicate that $e = 65,537$ is achievable with no difficulty and tests up to $e \simeq 10^{15}$ were conducted successfully. These results lead us to infer that this approach is practical at least for our range of interest.

However, should this strategy become difficult for larger e, a different (more expensive) approach might be used: replace the sparse linear algebra modulo e by exact Gaussian elimination or Hermite normal form and find relations expressing each ideal as a product (quotient) of smooth elements. This associates to each ideal a projection[11] in $\mathbb{Z}_n$ and also its e-th root. The drawbacks are higher memory requirements and a higher exponent in the linear algebra's complexity.

3.5 Complexity Analysis

Our complexity analysis closely follows the NFS's one. Let w denote the linear algebra's exponent. We write the degree d, the sieving range $[-S,+S]$ and the factor base bound B as:

$$d = \delta \times \sqrt[3]{\frac{\log n}{\log\log n}}, \quad S = L_n(\frac{1}{3}, w\beta) \text{ and } B = L_n(\frac{1}{3}, \beta).$$

This particular choice of S and B ensures that the sieving step (which costs S) and the linear algebra step (which costs B^w) are balanced.

Using the lattice-based construction, the coefficients of f have average size $A = \sqrt[d]{n} = L_n(\frac{2}{3}, \frac{1}{\delta})$. By choosing a skewed f, we find that the size of $f(x)$ for $x \in [-S,+S]$ is:

$$A \times S^{d/2} = L_n(\frac{2}{3}, \frac{1}{\delta} + \frac{w}{2}\delta\beta)$$

The probability that $f(x)$ is B-smooth is $L_n(\frac{1}{3}, -\pi)$ with $\pi = \frac{1}{3}(\frac{1}{\delta\beta} + \frac{w}{2}\delta)$.

To get enough smooth relations, we need to ensure that $w\beta - \pi = \beta$.

For $w = 2$, these equations lead to the choice $\{\delta = \sqrt[3]{\frac{3}{2}}, \beta = \sqrt[3]{\frac{4}{9}}\}$. As a consequence, the complexity of the sieving and linear algebra steps put together is $L_n(\frac{1}{3}, 2\beta) = L_n(\frac{1}{3}, \sqrt[3]{\frac{32}{9}})$. This is equal to the complexity the SNFS factoring algorithm which applies to a restricted class of numbers [12].

Another very important parameter is the number of AMR-oracle queries, which is subexponential but significatively smaller than the algorithm's runtime. This number of queries is $L_n(\frac{1}{3}, \beta) = L_n(\frac{1}{3}, \sqrt[3]{\frac{4}{9}})$.

The alternative using integer linear algebra mentioned above yields a complexity of:

$$L_n(\frac{1}{3}, \sqrt[3]{\frac{2w^4}{9(w-1)^2}})$$

The case $w = 3$ gives $L_n(\frac{1}{3}, \sqrt[3]{\frac{9}{2}} \simeq 1.65)$. Note that according to [8, 9], the integer linear algebra can be done with exponent $w = 2.5$, which yields $L_n(\frac{1}{3}, \sqrt[3]{\frac{625}{162}} \simeq 1.57)$. However this approach requires asymptotically fast matrix

[11] In theory, such a projection can be defined rigorously using the Hilbert class field of the number field used. Indeed, in the Hilbert class field, all ideals are principal and sending a generator to $\mathbb{Z}_n$ is easy; however, since the degree of the Hilbert class field is extremely large, it cannot be used in practice.

multiplication techniques which might prove too cumbersome for cryptographic applications.

As our algorithms are subexponential, the assessment of their *experimental* behavior is essential. We hence implemented them and actually forged a 512-bit AMR. Details are given in Appendix A.

Open Problem – Potential Improvements: When the number of fixed pad bits is small enough, the possible sieving range of x when sieving over $c + x$ (or $\alpha + x$) may be too large[12].

Under such circumstances, we get some additional freedom when constructing f. Indeed, we may replace c by some $c' \simeq c$, thereby reducing the sieving range. Clearly, amongst all possible c' values some yield f'-s whose coefficients are smaller than average.

We could not find any efficient way of taking advantage of this extra freedom to build better polynomials and further reduce the attack's complexity.

4 Attacking the One More RSA Inversion Problem

Up to now, we have obtained either an AMR-forgery or an adaptive chosen ciphertext attack (CCA2) on plain RSA. In this section, we extend the attack to obtain a non adaptive chosen ciphertext attack (CCA1) on plain RSA. Equivalently, we attack the One More RSA Inversion Problem, proposed by Bellare *et alii* in [2]. Again, while subexponential, this attack is faster than GNFS-factoring n. In the context of the One More RSA Problem it is not really meaningful to assume that the initial RSA queries have a special form, thus we grant the attacker access to an unlimited e-th root oracle during the first phase of the attack.

Once the restriction on oracle queries is lifted, we are no longer constrained to use polynomials with a prescribed root P. Moreover, we are no longer limited to a single dimensional sieve, but can use a classical NFS sieve with two degrees of freedom, using a *lattice sieving* technique. This does not change the asymptotic complexity but allows us to reuse existing fast sieving code more easily. Not being restricted to a prescribed root, we may use any polynomial of our choice. Despite this clear gain, to solve the One More RSA Inversion Problem and become non-adaptive, we need to devise an algorithm allowing us to compute the e-th root of an arbitrary number without any additional oracle queries. This requires a new descent procedure since the technique sketched at the end of Section 2 requires additional oracle queries. Looking at similar problems arising in the individual discrete logarithm phase of discrete logarithms computations, we see that such a non adaptive descent can be done by alternating between two NFS sides. Thus, we need to introduce a second side into our algorithm. While, at a first glance, this seems to void our single-sided NFS complexity improvement, it turns out

[12] *cf.* to the related footnote in section 3.1

that this intuitive perception is false since we can initially do the single sided NFS separately for both sides.

The addition of a second side entails a complication for the descent, however. To achieve the announced complexity, the initial factor base bound is set to $L_n(\frac{1}{3}, \sqrt[3]{\frac{4}{9}})$. This is well below the $L_n(\frac{1}{3}, \sqrt[3]{\frac{8}{9}})$ encountered when computing discrete logarithms. This implies that the descent procedure has to descend *below* what is done for computing discrete logarithms. While the impact on the overall complexity is not visible, this is a clear practical concern. To compensate for this fact, we add an intermediate phase in our algorithm in order to enlarge the factor base from $L_n(\frac{1}{3}, \sqrt[3]{\frac{4}{9}})$ to $L_n(\frac{1}{3}, \sqrt[3]{\frac{8}{9}})$.

4.1 The Inversion Algorithm

Step 0 – Setup. We first set up on the *algebraic side* a number field $K = \mathbb{Q}(\alpha)$ defined by a polynomial equation $f(\alpha) = 0$. The easiest (though not unique) choice for the second side is a *rational side* given by a polynomial g such that f and g share a common root P modulo n. The classical *base-m* technique can be used for this purpose.

We denote by ρ the rational root of g (we have $\rho = m$ if g is monic).

Step 1 – Precomputation. The factor base $\mathcal{F}$ on the algebraic side consists of ideals of norm bounded by $B \simeq L_n(\frac{1}{3}, \sqrt[3]{\frac{4}{9}})$. By sieving, we obtain coefficient pairs $\{x, y\}$ yielding relations of the form:

$$(x - y\alpha) = \prod_{\mathfrak{p} \in \mathcal{F}} \mathfrak{p}^{m_\mathfrak{p}}, \quad \text{and} \quad \chi(x - y\alpha) = (\lambda_k)_{k=1,\ldots,c}$$

where χ is a character map onto $\mathbb{F}_e^c$, for some arbitrary dimension c. We concatenate the coefficients $(m_\mathfrak{p})$ and λ_k to form the rows of a matrix M.

Step 2 – Factor Base Extension. The extended factor base $\mathcal{F}'$ consists of ideals of norm bounded by $B' \simeq L_n(\frac{1}{3}, \sqrt[3]{\frac{8}{9}})$. We sieve on the algebraic side only, using each additional prime ideal that we want to add as a special-q. We ask for a single relation between this prime ideal and the smaller ones.

Step 3 – Oracle Queries. We query the oracle for the e-th root of the numbers $x - yP$ for each integers pair $\{x, y\}$ encountered in steps 1 and 2. We also query for the e-th root of all prime numbers below B'.

Step 4 – Descent Initialization. In our game, it is only at this point that the attacker learns the challenge number t whose e-th root he must compute.

The descent mimics individual discrete logarithm computations. The descent is initialized by picking a random mask m and two integers u and v such that $\frac{u}{v} \equiv m^e t \bmod n$, and which factor simultaneous into primes bounded by $L_n(\frac{2}{3}, \bullet)$.

Step 5 – Descent. We maintain a set $\{(\sigma, \epsilon)\}$ of polynomials σ and exponents ϵ such that $S = \prod \sigma^\epsilon$ satisfies:

$$(S(\alpha)) = \prod_{\mathfrak{p} \in \mathcal{F}'} \mathfrak{p}^{\mu_\mathfrak{p}} \cdot I_1 \quad \text{(algebraic side)},$$

$$\text{and} \qquad \frac{u}{v} S(\rho) = \prod_{p < B'} p^{\nu_p} \cdot I_2 \quad \text{(rational side)}.$$

Initially $S = 1$, and the exponents ν_p mark the prime numbers appearing in the factorization of u and v.

The remaining terms I_1 and I_2 factor into ideals (or primes) outside the factor base. The descent procedure aims at eliminating these ideals. For this purpose, we iteratively use special-q sieving to trade these ideals for ideals of smaller norm.

Using the relations obtained from the factor base extension step, we form another rational fraction T such that the ideal $(S(\alpha)T(\alpha))$ factors into ideals belonging to the *smaller* factor base $\mathcal{F}$.

Step 6 – Linear Algebra. Once we have reached the point where $I_1 = (1)$ and $I_2 = (1)$, we seek a linear combination of the rows of the matrix M which equals the valuations and character values corresponding to the algebraic number $S(\alpha)T(\alpha)$.

This inhomogeneous linear system amounts to exhibiting an algebraic number $U(\alpha)$ obtained as a combination of the numbers $x - y\alpha$ found in step 1, and such that $S(\alpha)T(\alpha)U(\alpha)$ is an e-th power in K.

Step 7 – End. We use Montgomery's e-th root algorithm to write the previous number explicitly as an e-th power of an algebraic number $r(\alpha)^e$.

By construction, the e-th roots of $T(P)$ and $U(P)$ are known by the oracle queries. Using the rational side product form and the corresponding oracle queries, the e-th root of $\frac{u}{v}S(P)$ is known as well. We infer:

$$\sqrt[e]{t} = \frac{1}{m} \cdot \frac{\sqrt[e]{\frac{u}{v}S(P)}}{r(P)} \sqrt[e]{T(P)} \sqrt[e]{U(P)}.$$

4.2 Complexity Analysis

Using the same parameters as in Section 3 ($\delta = \sqrt[3]{\frac{3}{2}}, \beta = \sqrt[3]{\frac{4}{9}}$), all steps except steps 2 to 5 are achieved with complexity $L_n(\frac{1}{3}, \sqrt[3]{\frac{32}{9}})$.

The complexity of step 2 depends of course on the choice of β'. Let us consider one step of the factor base extension phase, where an ideal I of norm bounded by $L_n(\frac{1}{3}, x)$ is being considered. Using lattice reduction, we form a reduced basis of the lattice of algebraic integers $a - b\alpha$ such that I divides the ideal $(a - b\alpha)$. An expected bound for the coordinates of this basis is $L_n(\frac{1}{3}, x/2)$. Assume that we form linear combinations of the basis vectors with coefficients bounded by $L_n(\frac{1}{3}, f(x))$. We obtain algebraic integers $a - b\alpha$ with coefficients a, b bounded by $L_n(\frac{1}{3}, f(x) + x/2)$. Their algebraic norm is bounded by $L_n(\frac{2}{3}, \frac{1}{\delta} + (f(x) + x/2)\delta)$.

The probability that such a norm is $L_n(\frac{1}{3}, x)$-smooth is $L_n(\frac{1}{3}, -\pi)$, with:

$$\pi = \frac{1}{3}(\frac{1}{\delta x} + f(x)\frac{\delta}{x} + \frac{\delta}{2})$$

If $f(x)$ is suitably chosen, we expect exactly one relation in the sieve area, which rewrites as:

$$\begin{aligned} 2f(x) = \pi &= \frac{1}{3}\left(\frac{1}{\delta x} + f(x)\frac{\delta}{x} + \frac{\delta}{2}\right), \\ \left(6 - \frac{\delta}{x}\right) f(x) &= \frac{1}{\delta x} + \frac{\delta}{2}, \\ f(x) &= \frac{\frac{1}{\delta x} + \frac{\delta}{2}}{6 - \frac{\delta}{x}} = \frac{2 + \delta^2 x}{2\delta(6x - \delta)} \end{aligned}$$

The complexity is obtained by summing the work factor $L_n(\frac{1}{3}, 2f(x))$ over the extended factor base range: from $L_n(\frac{1}{3}, \beta)$ to $L_n(\frac{1}{3}, \beta')$. Since the number of ideals of norm bounded by $L_n(\frac{1}{3}, x)$ is approximated by $L_n(\frac{1}{3}, x)$, we are interested in the maximum value of $L_n(\frac{1}{3}, x + 2f(x))$. The function $x + 2f(x)$ increases over the range $[\beta, \beta']$. Therefore with the suggested extended bound $\beta' = \sqrt[3]{\frac{8}{9}}$, the complexity of the factor base extension is $L_n(\frac{1}{3}, 1.577)$.

The number of oracle queries (step 3) is $L_n(\frac{1}{3}, \beta' = \sqrt[3]{\frac{8}{9}})$.

The descent (steps 4 and 5) is analyzed in [5], and found to have complexity $L_n(\frac{1}{3}, \sqrt[3]{3})$.

We highlight, however, the complexity of the last descent steps, where ideals of norm just above $B' = L_n(\frac{1}{3}, \beta')$ have to be canceled. For each such ideal, one relation is sought. Using special-q sieving, we can form $L_n(\frac{1}{3}, 2\alpha)$ candidates whose algebraic (respectively rational) norm is bounded by $L_n\left(\frac{2}{3}, \frac{1}{\delta} + \delta\left(\alpha + \frac{\beta'}{2}\right)\right)$ (respectively $L_n\left(\frac{2}{3}, \frac{1}{\delta}\right)$). One relation is expected when α satisfies:

$$2\alpha - \frac{1}{3}\left(\frac{1}{\delta\beta'} + \frac{\delta}{\beta'}\left(\alpha + \frac{\beta'}{2}\right) + \frac{1}{\delta\beta'}\right) = 0.$$

Substituting $\beta' = \sqrt[3]{\frac{8}{9}}$ above, we obtain that the last descent steps are achieved in complexity $L_n(\frac{1}{3}, 0.99)$, which is not dominating. Using $\beta' = \beta$ (thereby skipping the factor base extension), this cost would be $L_n(\frac{1}{3}, 1.27)$ which is not dominating either.

This implies that we have some flexibility in the tuning of the factor base extension. In order to match previously completed discrete logarithm computations, we chose to extend to $\beta' = \sqrt[3]{\frac{8}{9}}$, but this choice should be regarded as unconstrained.

We conclude that the asymptotic complexity of the arbitrary e-th root computation is either $L_n(\frac{1}{3}, \sqrt[3]{\frac{32}{9}})$ or $L_n(\frac{1}{3}, 1.58)$. We believe the latter to be more practical, as is illustrated by our experiments (Appendix B).

4.3 Computing e-th Roots With an AMR-Oracle

While we have presented and implemented the arbitrary e-th root computation algorithm using access to a general e-th root oracle, the same can also be achieved using an AMR-oracle only. In this case, the common root P *is* prescribed, and it is not possible to use a rational side. Nonetheless, the above approach works using *two* algebraic sides; steps 1, 2, 6, and 7 have to be done separately on both sides.

Step 4, however, turns out to be more difficult when ony an AMR-oracle is accessible. As described in Subsection 4.1, Step 4 smoothes integers bounded by $\sqrt{n} = L_n(1, \frac{1}{2})$. Lacking a rational side, an adaptation of this step would require writing a target $m^e t$ as the image of an algebraic number under the constructed homomorphism from $\mathbb{Q}[\alpha]$ to $\mathbb{Z}_n$. Using lattice reduction, it is possible to obtain such an algebraic number as a quotient as in [11]. Unfortunately, the norms obtained in this manner are rather large. Assuming that algebraic integers of degree

$$\alpha \left(\frac{\log n}{\log \log n} \right)^a$$

are considered, the best possible upper bound for the norm is $L(1, \frac{\alpha}{\delta} + \frac{\delta}{2\alpha})$ (with $a = \frac{1}{3}$). This bound on the numerator and denominator is asymptotically $L_n(1, \sqrt{2})$ with $\alpha\sqrt{2} = \delta$. The following lemma details the analysis found in [5], and shows that this strategy of using a quotient fails in our case.

Lemma 1. *Assume that a black box process outputs sets of integers whose product is bounded by n^y. The goal is to isolate an output of the process in completely factored form, with factors bounded by the subexponential quantity $L_n(\chi, x)$, where $\chi < 1$. Using the Elliptic Curve Method (*ECM*), the complexity is minimized by setting $\chi = \frac{2}{3}$, $x = \sqrt[3]{\frac{y^2}{3}}$. The complexity is then $L_n(\frac{1}{3}, \sqrt[3]{3y})$.*

Proof. The outline of the factoring strategy is as follows. We determine a bound on the ECM factoring effort which is necessary to find factors bounded by $L_n(\chi, x)$. A process' output is successfully factored if it meets the $L_n(\chi, x)$-smoothness property. The algorithm's complexity is therefore given by multiplying the ECM effort by the inverse of the smoothness probability.

An output of the process is $L_n(\chi, x)$-smooth with probability $L_n(1 - \chi, \bullet)$. A factor bounded by X is found by ECM in time $L_X(\frac{1}{2}, \sqrt{2})$, which rewrites as

$L_n(\frac{\chi}{2}, \bullet)$ if X is substituted with $L_n(\chi, x)$. We therefore optimize the complexity by setting $\chi = \frac{2}{3}$.

We make the analysis more precise by fixing $\chi = \frac{2}{3}$. The smoothness probability is $L_n(\frac{1}{3}, -\frac{y}{3x})$. The ECM complexity is $L_{L_n(\frac{2}{3},x)}(\frac{1}{2}, \sqrt{2}) = L_n(\frac{1}{3}, \sqrt{\frac{4x}{3}})$ per factorization attempt. The overall complexity $L_n(\frac{1}{3}, \frac{y}{3x} + \sqrt{\frac{4x}{3}})$ reaches a minimum value $L_n(\frac{1}{3}, \sqrt[3]{3y})$ when one sets $x = \sqrt[3]{\frac{y^2}{3}}$, as claimed.

Applying Lemma 1 to $y = 2\sqrt{2}$ (numerator and denominator bounded by $L_n(1, \sqrt{2})$) gives a complexity of $L_n(\frac{1}{3}, \sqrt[3]{6\sqrt{2}})$. This complexity exceeds the GNFS's complexity, and therefore the strategy does not pay-off.

A seemingly cruder approach, though, consists in writing the target number as the image of *one* algebraic integer under the homomorphism from $\mathbb{Q}[\alpha]$ to $\mathbb{Z}_n$. This is done using the following steps.

- Find a random integer m such that $m^e t \bmod n = t_1 \times t_2 \ldots \times t_k$ is $L_n(\frac{2}{3}, x_0)$-smooth.
- For each factor t_i, pick an algebraic integer $u(\alpha)$ of degree $d-1$ such that $u(\alpha)$ maps to zero in $\mathbb{Z}_n$. Repeat until the norm of $t_i \pm u(\alpha)$ is $L_n(\frac{2}{3}, x_1)$-smooth.

The first step above is completed with complexity $L_n(\frac{1}{3}, \sqrt[3]{3})$, with $x_0 = \frac{1}{\sqrt[3]{3}}$. For the second step, we use the same lattice as constructed during the polynomial selection step. The norm of $u(\alpha)$ is bounded by $L_n(1, 2)$. The bound is not affected if $t_i \pm u(\alpha)$ is considered instead, since the t_i are assumed to be bounded by $L_n(\frac{2}{3}, x_0)$, which is below the bound $L_n(\frac{2}{3}, \frac{1}{\delta})$ on the coefficients of u. We apply Lemma 1 to $y = 2$, which gives a complexity of $L_n(\frac{1}{3}, \sqrt[3]{6})$ for Step 4 of the descent.

Step 5 of the descent is adapted in a straightforward way. The analysis done in Section 4.2 carries over *mutatis mutandis* with few modifications. However, the analysis of the last descent steps shows that the factor base extension becomes necessary, since setting $\beta' = \beta$ would imply that the last descent steps are more costly than the GNFS. The factor base extension to *e.g.* $\beta' = \sqrt[3]{\frac{8}{9}}$ lowers this complexity to $L_n(\frac{1}{3}, 1.10)$.

The complexity of computing arbitrary e-th roots with access to an AMR-oracle only is therefore dominated by the complexity of Step 4 of the descent, *i.e.* $L_n(\frac{1}{3}, \sqrt[3]{6})$.

Acknowledgements

The authors would like to thank P. Zimmermann who provided early experimental input for the quadratic sieve version.

References

1. K. Aoki, J. Franke, T. Kleinjung, A. Lenstra and D. Osvik, *Electronic newsgroup posting announcing the factorization of the 1039-th Mersenne number by the* SNFS, `http://www.loria.fr/ zimmerma/records/21039-`, May 21, 2007.
2. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, *The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme*, Journal of Cryptology, vol. 16(3), pp. 185–215, 2003.
3. J. P. Buhler, A. K. Lenstra, and J. M. Pollard, *Factoring integers with the number field sieve*, In A. K. Lenstra and H. W. Lenstra, Jr., Eds., *The development of the number field sieve*, LNM vol. 1554, pp. 50–94. Springer–Verlag, 1993.
4. É. Brier, C. Clavier, J.-S. Coron, and D. Naccache, *Cryptanalysis of* RSA *signatures with fixed-pattern padding*, Proceedings of CRYPTO'01, LNCS vol. 2139, pp. 433–439, Springer–Verlag, 2001.
5. A. Commeine, and I. Semaev *An algorithm to solve the discrete logarithm problem with the number field sieve*, Proceedings of the 9-th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography, LNCS vol. 3958, pp. 174–190, 2006.
6. J.-S. Coron, D. Naccache, and J. P. Stern *On the Security of* RSA *padding*, Proceedings of CRYPTO'99, LNCS vol. 1666, pp. 1–18, Springer–Verlag, 1999.
7. W. De Jonge and D. Chaum, *Attacks on some* RSA *signatures*. Proceedings of CRYPTO'85, LNCS vol. 218, pp. 18–27, Springer-Verlag, 1986.
8. W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard, *Solving sparse rational linear systems*, In B. M. Trager Ed. ISSAC 2006, pp. 63–70, ACM Press, 2006.
9. W. Eberly, M. Giesbrecht, P. Giorgi, A. Storjohann, and G. Villard. *Faster inversion and other black box matrix computations using efficient block projections*, In C. W. Brown, Ed., ISSAC 2007, pp. 143–150, ACM Press, 2007.
10. M. Girault and J.-F. Misarksy, *Selective forgery of RSA signatures using redundancy*, Proceedings of EUROCRYPT'97, LNCS vol. 1233, Springer-Verlag, pp. 495–507, 1997.
11. A. Joux and R. Lercier, *Improvements to the general number field sieve for discrete logarithms in prime fields. A comparison with the gaussian integer method*, Mathematics of Computation, vol. 242(72), pp. 953–967, 2003.
12. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. *The number field sieve*, In A. K. Lenstra and H. W. Lenstra, Jr., Eds., *The development of the number field sieve*, LNM vol. 1554, pp. 11–42. Springer–Verlag, 1993.
13. A. K. Lenstra, and I. Shparlinski, Selective forgery of RSA signatures with fixed-pattern padding, Proceedings of the 5-th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography, LNCS vol. 2274, pp. 228–236, 2002.
14. J.-F. Misarsky, *A multiplicative attack using* LLL *algorithm on* RSA *signatures with redundancy*, Proceedings of CRYPTO'97, LNCS vol. 1294, Springer-Verlag, pp. 221–234, 1997.
15. J.-F. Misarsky, *How (not) to design* RSA *signature schemes*, Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography, Springer-Verlag, LNCS vol. 1431, pp. 14–28, 1998.
16. P. L. Montgomery, *Square roots of products of algebraic numbers*, In W. Gautschi, Ed., Mathematics of Computation 1943–1993: A Half-Century of Computational Mathematics, vol. 48 of Proc. Sympos. Appl. Math., pp. 567–571. AMS, 1994.

17. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, CACM 21, 1978.
18. RSA Laboratories, PKCS #1 : RSA *cryptography specifications*, version 2.0, September 1998.
19. O. Schirokauer, *Discrete logarithms and local units*, Philos. Trans. Roy. Soc. London Ser. A, 345(1676), pp. 409–423, 1993.

A Implementation Details

As our algorithms are subexponential, the assessment of their *experimental* behavior is essential. We hence implemented them and actually computed a 512-bit AMR.

We wrote our software chain in C and C++, relying upon the computer algebra systems PARI-GP and MAGMA for a handful of specific tasks. The attacked instance was $c = 10^{154}$, $e = 65,537$ and $n =$ RSA-155 (RSA Laboratories 512-bit challenge modulus).

The polynomial selection (section 3.1) was implemented in MAGMA. To obtain a satisfactory relation yield, we have set $B = 2^{22}$ (*i.e* a factor base comprising *circa* 300,000 prime ideals). For $S = 2^{50}$, the polynomial selection program returned the quartic candidate[13] $f(x) = \sum_{i=0}^{4} a_i x^i$ where:

$$
\begin{aligned}
a_4 &= 8\\
a_3 &= 5451802006688119\\
a_2 &= -7344893341388732622814165470437\\
a_1 &= 833050630351576525584507524542841090670386803\\
a_0 &= -80690902433251999116158516330020702292190401223994350445959
\end{aligned}
$$

We worked in $K = \mathbb{Q}[x]/f$ and counted 295,842 prime ideals of degree one (or dividing the leading coefficient) in K's integer ring.

The sieving process was run on a heterogeneous set of CPUs: AMD Opteron 250 at 2.4 GHz and Intel Core-2 at various clock speeds.

For each special-q ideal written as $\langle q, \alpha - r\rangle$, we isolated the integers $x \in [-2^{28}, 2^{28}]$ such that the added contribution of factor base ideals to the norm of the ideal $(r + qx - \alpha)$ exceeded 2^{145} (out of an order of magnitude just below 2^{200}). This selection process isolated instantaneously[14] *circa* 100 candidates of which around nineteen yielded relations. Considering the largest 20,000 ideals in the factor base as special-q ideals, we obtained 380,000 relations. The sieving step was distributed over twenty CPUs and claimed a couple of hours. We stress that we did not use any "large prime" variation.

After pruning the columns corresponding to ideals never encountered in the factorizations, we were left with a row dependency to be obtained on a 283,355×283,355 matrix. We included four readily computed character columns in the matrix, to ensure that the computed dependency corresponds to an e-th

[13] Best amongst a set of 1,000 candidates.
[14] 2.667 GHz Intel Core-2 CPU

power. The dependency was obtained using the block Wiedemann algorithm, with a "blocking factor" of $m = n = 8$. This took four CPU[15] hours distributed on four machines to produce one row dependency.

The e-th root computation was done in MAGMA.

We started with a product formula π whose numerator and denominator had a norm $\simeq e^{7.6\times 10^5}$ and with a moderate unit contribution, since the logarithms of the complex embeddings were approximately:

$$(\lambda + 45, \lambda + 45, \lambda - 155, \lambda + 65) \quad \text{where} \quad \lambda = \frac{1}{d} \log \mathrm{Norm}(\pi) \simeq 6710$$

Here λ is the normalizing term. This is quite small since a unit with logarithms of complex embeddings equal to $(45, 45, -155, 65)$ would correspond to an algebraic integer with coefficients of about twenty decimal digits. The first four reduction steps sufficed to eliminate this unit contribution (*i.e.* equalling the logarithms of the complex embeddings with their average). After $2,000$ reduction steps, we obtained a complete product formula for the root, the remaining e-th power being -1. It took five minutes to compute this e-th root.

The corresponding final multiplicative dependency involved $242,700$ integers of the form $c + x_i \bmod n$.

B An e-th Root Computation Example

As an experimental illustration of the arbitrary e-th root computation, we used once again $n =$ RSA-155. For a public exponent $e = 65,537$, we detail the computation of an arbitrary e-th root given access to a preliminary e-th root oracle (the attacker is given the challenge only once all oracle queries have been performed).

We chose a setup resembling a typical NFS factoring experiment or a computation of discrete logarithms. The polynomials $f_1 = \sum a_i x^i$ and $f_2 = \sum b_i x^i$ are given by the following coefficients, the polynomial f_2 corresponding to a rational side:

$a_5 =$ 28200000
$a_4 =$ - 7229989539851
$a_3 =$ - 24220733860168568962
$a_2 =$ - 6401736489600175386662132
$a_1 =$ 4117850270472750057831223534880
$a_0 =$ 747474581145576370776244346990929200

$b_1 =$ 14507315380338583
$b_0 =$ - 2078583364878181938242401502 87

These two polynomials are easily seen to share a common root P modulo n.

The sieving stage has been performed only on the number field side. We chose as a *small* factor base the set of prime ideals of norm below $B = 4 \times 10^6$

[15] 2.667 GHz Intel Core-2

(*i.e.* 283,042 ideals). For the sieving, we have used the lattice sieving program `lasieve4` of J. Franke and T. Kleinjung included in the `ggnfs` software suite. The program was modified to sieve only on one side. Using a double large prime variation, the sieving step has been completed in two CPU hours on a 2.4GHz AMD Opteron.

We then extended the factor base to the larger bound $B' = 2^{32}$. After 44 CPU hours, we were able to relate 37% of the ideals of this larger factor base to ideals of the smaller factor base (the larger factor base comprises approximately 2×10^8 ideals).

Counting oracle queries related to both sides, we need to perform 4×10^8 queries before being able to compute arbitrary e-th roots.

We implemented the descent procedure using MAGMA, as well as the `lasieve4` program, modified in order to account for very large *special q's* as used in the descent process. The factorization of the numerous sieve residues produced was handled by the GMP-ECM program.

The descent was initialized on the rational side. We obtained integers u and v which factored into primes with at most 35 decimal digits. Each descent procedure step involved a `lasieve4` call, in order to select several candidate polynomials. Amongst the possible polynomials, our strategy selected the one leading to the fewest ideals outside the factor base (taking into account the large ideals coming from the factor base extension). After 42 descent steps, we obtained a product formula involving 594 prime numbers and ideals below $B' = 2^{32}$. Some (19) ideals in this product formula belonged to the set of "missed" ideals from the larger factor base. With 21 extra descent steps, these ideals were eliminated. The descent procedure took roughly one hour.

The schedule time for solving the resulting inhomogeneous linear system and computing the algebraic e-th root compares in every respect to the data given for the previous example (Appendix A).