Evaluation Report on the **Factoring** Problem

Jacques Stern

1 Introduction

This document is an evaluation of the factoring problem, as a basis for designing cryptographic schemes. It relies on the analysis of numerous research papers on the subject.

The present report is organized as follows: firstly, we review the factoring problem and several related problems such as the problem of extracting e-th roots, which forms the basis of the RSA cryptosystem. Next, we analyze the various algorithms that are currently known to solve the problem. For each algorithm, we study its asymptotic behaviour, as well as its practical running time, based on experiments reported in the literature. Finally, we derive consequences in terms of key sizes for cryptosystems whose security depend on the hardness of factoring. We conclude by making some predictions on how the key sizes might evolve. This is as requested by IPA.

2 Factoring and related problems

In this section, we review the problem of factoring integers and several related problems, and we investigate their security in terms of complexity-theoretic reductions. Besides the problem of extracting e-th roots, which forms the basis of the RSA cryptosystem, we cover the strong RSA problem and the approximate e-th root problem AERP, which have been recently considered in relation with various cryptographic schemes.

2.1 Factoring as a cryptographic primitive

It is well known that any integer n can be obtained, in a unique manner, as the product of non necessarily distinct prime numbers. Factoring n consists in finding the list of such primes, also called the prime decomposition of n. Of particular interest, is the case of integers n, which are the product of two prime factors p and q.

The basic security assumption on which numerous cryptosystems rely is the *hardness* of factoring: an attacker cannot find the prime decomposition of n. In a more precise complexity-theoretic framework, this means that the success probability $\mathsf{Succ}^{\mathsf{fact}}(\mathcal{A})$

of any polynomial time algorithm $\mathcal{A}$ attempting to factor n is negligible, i.e. asymptotically smaller than the inverse of any polynomial function of the security parameter. Probabilities are taken over n and the random coins of $\mathcal{A}$. The security parameter k is directly related to the bit-size of n. For example, one can simply set $k = |n|$. In most cryptographic applications, especially in the case of integers of the form $n = pq$, n is not chosen at random, but is, implicitly or explicitly, produced by a key generation algorithm $\mathcal{K}(1^k)$. In the case of the RSA cryptosystem, the key generation algorithm, chooses, on input k, two large primes p, q of equal size and issues the so-called modulus $n = pq$. The sizes of p, q are set in such a way that the binary length $|n|$ of n equals $2k$. Further tests, related to the factors of $p-1$ and $q-1$ are usually included that avoid weak moduli. This restricts the family of integers that $\mathcal{A}$ attempts to factor to those output by the key generation algorithm $\mathcal{K}(1^k)$. In this case probabilities also range over the random coins that $\mathcal{K}$ uses.

To go from asymptotic to exact estimates, we can define $\mathsf{Succ}^{\mathsf{fact}}(\tau, k)$ as the probability for an adversary to factor a given integer within time τ. We turn the above into symbols, in the case of integers n of the form p, q, $p < q$:

$$\mathsf{Succ}^{\mathsf{fact}}(\mathcal{A}) = \Pr[n \leftarrow \mathcal{K}(1^k), n = pq : \mathcal{A}(n) = p].$$

The hardness of the factoring is the statement that, for large enough k, this probability is extremely small.

Many cryptographic schemes rely on the assumption that factoring is hard, and, at this point, we will simply mention the celebrated RSA algorithm proposed by Rivest, Shamir and Adleman [51].

2.2 RSA and related schemes

The modern approach to cryptographic design relies on the notion of security proof. Such proofs are *reductions* in the sense of complexity theory. Given an attacker $\mathcal{A}$ that breaks the cryptographic scheme, one designs another machine $\mathcal{B}$, solving the underlying hard problem. The relation between running times and success probabilities of $\mathcal{A}$ and $\mathcal{B}$ is further made explicit, so that, if the security loss is not too large, concrete estimates on key sizes may be derived from the proof.

Ideally, one would like to establish the security of a scheme based on the sole assumption that the underlying problem is hard. Unfortunately, very few schemes allow such a proof. For the others, the best one can hope for is a proof carried in a non-standard computational model, as proposed by Bellare and Rogaway [4], following an earlier suggestion by Fiat and Shamir [27]. In this model, called the random oracle model, concrete objects such that hash functions are treated as random objects. This allows to carry through the usual reduction arguments to the context of relativized computations, where the hash function is treated as an oracle returning a random answer for each new query. A reduction still uses an adversary as a subroutine of a program that contradicts a mathematical assumption, such as the hardness of factoring However, probabilities are taken not only over coin tosses but also over the random oracle.

In the case of factoring, whatever the computational model is, it is often the case that security is not related to the problem itself, but rather to the question of extracting e-th root modulo n. The case $e = 2$ is the problem of extracting modular square roots, which is easily shown equivalent to factoring. The general case is related to the RSA cryptosystem. At key generation, an exponent e, relatively prime to $\varphi(n) = (p-1)(q-1)$ is chosen, and the corresponding RSA function is defined by

$$x :\longrightarrow (x^e) \bmod n$$

The main conjecture related to this function is its *one-wayness* (OW): using only public data, an attacker cannot invert the function. More precisely, denote by $\mathsf{Succ}^{\mathsf{rsa}}(\tau, k)$ the probability for an adversary to find the preimage of a given element within time τ, in symbols:

$$\mathsf{Succ}^{\mathsf{rsa}}(\tau, k) = \Pr[(n, e) \leftarrow \mathcal{K}(1^k), y \leftarrow \mathbb{Z}_N, x \leftarrow \mathcal{A}(n, e, y) : y = x^e \bmod n],$$

then, for large enough moduli, this probability is extremely small. The asymptotic version states that, when $\mathcal{A}$ has running time bounded by a polynomial function τ of the security parameter, $\mathsf{Succ}^{\mathsf{rsa}}(\tau, k)$ eventually becomes smaller than the inverse of any polynomial in k.

It is well known that, if d is the inverse of e modulo $\varphi(n)$, $y^d \bmod n$ computes the inverse of the RSA function. Thus, the factorization of n allows to invert the RSA function, since d can be computed from p and q. It is unknown whether the converse is true, i.e. whether factoring and inverting RSA are computationally equivalent. There are indications that it might not be true (see [8]). Thus, the assumption that RSA is one-way might be a stronger assumption than the hardness of factoring. Still, it is a widely believed assumption, and the only method to assess the strength of RSA is to check whether the size of the modulus n outreaches the current performances of the various factoring algorithms.

The *strong RSA problem*, attempts to find x and $e > 1$, such that $x^e = y \bmod n$, given an element y in $\mathbb{Z}_n^\star$. The assumption that the problem is hard is stronger than the RSA assumption, that was just considered above. This hypothesis has been recently used as a basis for the security of several cryptographic schemes. Of particular interest is the Cramer-Shoup signature scheme [11], where the security proof is carried in the standard model of computation, without any random oracle.

Another problem that has been considered in [45] is the *approximate e-th root problem* (AERP: given n and y in $\mathbb{Z}_n^\star$, find x such that $x^e \bmod n$ lies in a prescribed interval centered at y. Of particular interest is the case when the length of the interval is of the order of $n^{2/3}$, which forms the basis of the ESIGN [25, 45] signature scheme. In this case, the key generation algorithms produces moduli of but-size $3k$, where k is the security parameter and the exact format of the problem is to find x such that $y \leq x^e \bmod n < y + 2^{2k-1}$. Again, computing the inverse of the RSA function clearly allows to solve the AERP problem. It is unknown whether the converse is true, i.e. whether AERP and inverting RSA are computationally equivalent. This is conjectured in [26], as soon as $e \geq 4$.

It should be mentioned that the ESIGN signature scheme uses moduli of the form p^2q, where p and q are primes of size $k = \frac{|n|}{3}$. This is related to an algebraic property of exponentiation modulo such integers, which offers an extremely efficient way to find an element of $\mathbb{Z}_n^\star$ whose e-th power lies in a prescribed interval of length at least pq. Moduli with three or more distinct factors are also being considered in the setting of *multiprime* RSA (see [54]).

In conclusion, the only method known to attack the RSA problem, as well as its variants, including AERP, is is to factor the modulus. Therefore, in order to estimate whether the parameters of a cryptographic scheme, relying on the hardness of these problems, offer a wide security margin, one needs to refer to the performances of the various algorithms known for factoring.

3 Algorithms for factoring

As was just observed, the security of many cryptographic schemes can only be measured in terms of the hardness of factoring integers. We now review the known factoring algorithms. Before discussing the two main general purpose factoring algorithms, the quadratic sive (QS) and the number field sive (NFS), we decribe two specific families of algorithms, respectively covering the case where the number n to factor has relatively small factor, and the case where there are repeated factors.

3.1 Factoring techniques sensitive to the size of the smaller factor

3.1.1 Pollard's ρ-method.

Assume that the modulus to factor is denoted by n and its smallest factor by p. The idea behind the method is to iterate a polynomial P with integer coefficients, that is to say computing $x_1 = P(x_0) \bmod n$, $x_2 = P(P(x_0)) \bmod n$, etc. We try to find a collision modulo p, i.e. two values x and x' such that $x = x' \bmod p$. Since it is extremely unlikely that the same equality holds modulo n, we then factor n by computing the g.c.d. of $x - x'$ and n. P is usually taken to be $x^2 - 1$.

Collison search is an extremely well understood problem whose time complexity is

$O(\sqrt{p})$ and which can be programmed efficiently. The problem here is that the collison test is a gcd computation, which slows down the whole algorithm. Although there are several optimizations, the ρ-method can only be used to cast out small factors of an integer (say 30-digit factors). As far as we know, it has not been used to find significantly larger factors. Although large scale collision search seems possible, following the ideas in [57], experiments along these lines have not been reported in the literature.

3.1.2 The $p-1$ method.

Let B be a positive integer. A number is B-smooth if it is of a product of prime numbers all $< B$. B-smooth numbers are usually used through a table of primes $< B$. The $p-1$ method relies on the use of Fermat's little theorem: if $p-1$ happens to be B-smooth, then the gcd of n and $a^{\ell(B)}-1$ factors n, where $\ell(B)$ is the product of all prime factors $< B$.

Of course, the $p-1$ method only works if $p-1$ is smooth. Therefore, it is quite irrelevant to cryptography, since key generation algorithms usually eliminate such primes. We have included the method in the present report as a step towards more efficient techniques.

3.1.3 The elliptic curve method.

The ECM is a generalization of the $p-1$ method, for which the above simple countermeasure is not sufficient. Consider an elliptic curve mod$n$ with equation

$$y^2 = x^3 + ax + 1$$

If the number of points of this curve modulo p is B-smooth, then a factor of n can be discovered along the computation of the scalar multiplication of $M_0 = (0,1)$ by $\ell(B)$, according to the group law of the elliptic curve.

As for many factoring algorithms the success probability of the algorithm, and the resulting complexity estimates, are naturally expressed in terms of the L-function:

$$L_q[s;c] = \exp(c(\ln q)^s(\ln\ln q)^{1-s}).$$

The reason why this function is involved is its relation with the asymptotic probability that a random element can be factored into elements of a factor base (see [29, 46]). For instance, it is known that the probability that a random integer $< L_x[\nu;\lambda]$ has all its prime factors $< L_x[w;\mu]$ is asymptotically

$$L_x[\nu-w; -\lambda(\nu-w)/\mu + o(1)].$$

Using such estimates, it can be shown that the elliptic curve is $L_p[1/2;\alpha]$-smooth with probability $L_p[1/2;-1/(2\alpha)+o(1)]$. This is minimal for $\alpha = 1/\sqrt{2}$ and gives an expected running time of $L_p[1/2;\sqrt{2}+o(1)]$ group operations on the curve.

There have been several improvements of ECM factoring, notably the FFT extension of P. Montgomery. Furthermore, several implementations of ECM are available. The current ECM factoring record was established in december 1999, when a prime factor with 54 digits of a 127-digit composite number n was found with GMP-ECM, a free implementation of the Elliptic Curve Method (see [39]). The limit used was $B = 15,000,000$.

In a recent paper [9], Richard Brent extrapolates the ECM record to be of D digits at year about

$$Y = 9.3 * \sqrt{D} + 1932.3$$

this would give records of $D = 60$ digits at year $Y = 2004$ and $D = 70$ at year 2010. Such record would need $B \simeq 2,900,000,000$ and require testing something like $340,000$ curves. It can be noted that, if Brent's prediction is correct, moduli with factors of 256 bits will become insecure at year $Y = 2014$. It should be noted that the ECM method decribed in this section is superseded by the algorithms that appear further on in the present report. However, since its computing time is related to the size of the smallest factor p of n, ECM gives an indication on the minimum size of p. Accordingly, it limits the number of distinct prime factors of equal size that can be allowed for multiprime schemes, when the size of the modulus is given.

3.2 Specific factoring techniques for numbers of the form $p^r q$

In recent work (see [7]), a new factoring method that applies to integers of the form $p^r q$ has been found. The method is based on an earlier result of Coppersmith (see[18]), showing that an RSA modulus $n = pq$, with p, q of the same size, can be factored given half the most significant bits of p. It turns out that, for numbers of the form $n = p^r q$, with p, q of the same size, fewer bits are needed. In the sequel we let c be a real constant such that $q \simeq p^c$. In many cases of interest for cryptography, c is set to 1.

Note that disclosing the leading bits of p provides a rough approximation P of p. What remains to be found is the difference $x_0 = p - P$. Note that x_0 is a root of $f(x) = (x + P)^r$, modulo p^r. The new method is based on finding polynomials with short enough integer coefficients, which vanish at x_0 modulo some power p^{rm} of p. It relies on techniques that find small solutions to modular polynomial equations, investigated by Coppersmith [17, 19], Howgrave-Graham [30] and Jutla [32]. The key result is the following simple lemma:

Lemma 3.1 *Let $h(x)$ be a polynomial of degree d with integer coefficients and let X be a positive integer. Assume $||h(xX)|| < M/\sqrt{d}$. If $h(x_0) = 0\,(mod\,M)$ and $|x_0| < X$, then $h(x_0) = 0$ holds over the integers.*

Proof. From the Cauchy inequality, we get that $|h(x_0)|$ is bounded by $\frac{M}{\sqrt{d}}\sqrt{d} = M$. Hence the result.

The lemma suggests to look for polynomials $h(x)$, such that $h(x)$ has x_0 as a root modulo $M = p^{rm}$, where m is an appropriate integer, and $h(xX)$ has norm less than roughly p^{rm}. We let

$$g_{i,k}(x) = n^{m-k}x^i f(x)^j.$$

Observe that x_0 is a root of $g_{i,k}(x)$ modulo M, for all i and all $k = 0, \dots, m$. Thus, in order to apply the lemma, it is enough to find a polynomial of small enough norm in the lattice L generated by various $g_{i,k}(xX)$ of bounded degree. This is a standard lattice reduction problem.

We use d such polynomials, where $d > mr$, will be determined later.

1. all $g_{i,k}(x)$, with $i = 0, \dots, r-1$ and $k = 0, \dots, m-1$,
2. all $g_{j,m}(x)$, with $j = 0, \dots, d - mr - 1$.

The determinant of the lattice L is easily seen to be the product of the leading coefficients

$$\Delta = X^{d^2/2} n^{rm(m+1)/2}$$

Based on LLL heuristics [31], short vectors of this lattice should therefore be around $X^{d/2} n^{\frac{rm(m+1)}{2d}}$. When using the LLL algorithm, there is a further multiplicative factor $2^{d/2}$ (see [34]), but we do not take this into account. Thus the method is heuristically valid when

$$X^{d/2} n^{\frac{rm(m+1)}{2d}} < p^{rm}/\sqrt{d}$$

Again, we forget the $\sqrt{d}$ term, as we did for the $2^{d/2}$ multiplicative factor and obtain:

$$X^{d/2} < p^{-\frac{rm(m+1)(r+c)}{2d}} p^{rm}$$

The optimal value of m is attained at $\lfloor \frac{d}{r+c} - \frac{1}{2} \rfloor$ and we may choose d so that $\frac{d}{r+c}$ is within $\frac{1}{2(r+c)}$ of an integer. Working through arithmetics yields the bound

$$X < p^{1 - \frac{c}{r+1} - \frac{r}{d}(1+\delta)}$$

with

$$\delta = \frac{1}{r+c} - \frac{r+c}{4d}$$

Since δ is < 1, we obtain the simpler bound

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

To conclude, lattice reduction will find a polynomial that vanishes at x_0 over the integers if the bound X for the difference $p - P$ is bounded by

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

At this point, any standard root-finding algorithm will disclose x_0, and therefore p.

Note that, when p, q are of almost equal size, $c = 1$, we can take d of the order of r^2 and ignore the term $2\frac{r}{d}$. This means that factoring is performed by lattice reduction as soon as an approximation P of p is known, such that

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

In other words, it is enough to guess a fraction $\frac{1}{r+1}$ of the leading bits of p, to be able to factor n.

Comparing the above estimate with the running time for ECM, one can see that the new method beats ECM for r larger than, approximately $\sqrt{\ln p}$. Several experiments are reported in [7]. For example, the authors were able to factor a 768 bit modulus, with p, q of equal bit-size 96 and $r = 7$, given the 22 leading bits of p. The running time for LLL was 10 hours. No experiments appears possible when $r = 1$, 2 or 3, which are the cases of interest for cryptography. Thus, for very small values of r, the algorithm is certainly impractical.

We close this section by mentioning that it is also possible to slightly speed up ECM for numbers of the form p^2q (see [47], even though this does not apply to $n = p^dq$, with odd d. Despite, its academic value, the improvement does not change the order of magnitude of the running time for ECM.

3.3 The Quadratic Sieve.

The quadratic sieve method (QS) factors n by gathering many congruences of the form

$$x^2 = (-1)^{e_0} p_1^{e_1} \cdots p_m^{e_m}$$

where $p_1, \cdots, p_m$ is a list of prime numbers $< B$, called the factor base. This is done by finding B-smooth numbers of the form $Q(a) = (\sqrt{n} + a)^2 - n$. It turns out that there is a very efficient sieving process that performs the job without division, hence the name QS. Once enough congruences have been gathered, one obtains another congruence of the same type with all exponents e_i even: this is done by Gaussian elimination mod2. Thus one gets a relation $x^2 = y^2 \bmod n$ and, with significant probability, computing $\gcd(x-y, n)$ factors n. The time complexity of QS is $L[1/2; 1+o(1)]$ but, as it uses very simple operations, it is usually more efficient than ECM for numbers whose smallest prime factor exceeds $n^{1/3}$.

Many improvements of the basic method have been found, notably the multiple polynomial variation (MPQS) and the large prime variation. This has led to very efficient implementation and, until the mid-nineties, was used to set up factoring records. The largest number factored by MPQS is the 129-digit number from the "RSA Challenge" (see [53]). It was factored in april 1994 and took approximately 5000 mips-years (see [3]).

3.4 The Number Field Sieve.

The number field sieve (NFS) is somehow similar to the QS but it searches for congruences in some number field (algebraic extension of the rational numbers). The method was first introduced in 1988 by John Pollard [49] to factor numbers of the form x^3+k. It was quickly extended to handle numbers of the form r^e+s for small positive r and $|s|$: this was successfully applied to the Fermat number $F_9 = 2^{512}+1$ (see [37]). This version of the algorithm is now called the special number field sieve (SNFS) [36], in contrast with the general number field sieve (GNFS) [10], which can handle arbitrary integers. GNFS factors integers n in heuristic time $L_n[1/3; c+o(1)]$, with $c = (64/9)^{1/3} \approx 1.9$. This is asymptotically considerably better than QS.

3.4.1 Overview

The number field sieve (NFS) first selects two low-degree irreducible polynomials $f_1(X)$ and $f_2(X)$ in $\mathbb{Z}[X]$ with small coefficients, such that $f_1(X)$ and $f_2(X)$ have a common root m modulo n. These polynomials define two number fields $\mathbb{Q}(\alpha_1)$ and $\mathbb{Q}(\alpha_2)$. Because m is a root, there is a natural ring homomorphism φ_j from $\mathbb{Z}[\alpha_j]$ to $\mathbb{Z}_n$ ($j \in \{1,2\}$), induced by $\varphi_j(\alpha_j) = m$. We extend φ_j to the field $\mathbb{Q}(\alpha_j)$, ignoring potential divisibility problems, which only appear if n gets factored.

The ring of integers of a number field is not necessarily a unique factorization domain, but the ring of fractional ideals always is. Thus, the NFS selects two ideal factor bases $\mathcal{B}_1$ and $\mathcal{B}_2$ (corresponding to $\mathbb{Q}(\alpha_1)$ and $\mathbb{Q}(\alpha_2)$) consisting of prime ideals of norm only divisible by small primes. We define smoothness as follows: a fractional ideal $\mathcal{I}$ of $\mathbb{Q}(\alpha_j)$ is *smooth* if it can be factored completely over the factor base $\mathcal{B}_j$, and an algebraic number x is *smooth* if the fractional ideal $\langle x \rangle$ it spans is smooth.

By sieving, the NFS finds a huge collection of pairs (a_i, b_i) of small integers such that each $a_i - b_i\alpha_j$ is smooth: as a result, the factorization of $\langle a_i - b_i\alpha_j \rangle$ is known. Linear algebra modulo 2 produces many sets S of indices such that, for each $j = 1, 2$, the fractional ideal $\prod_{i \in S} \langle a_i - b_i\alpha_j \rangle$ is a square. This does not necessarily mean that the algebraic number $\prod_{i \in S}(a_i - b_i\alpha_j)$ is a square in $\mathbb{Q}(\alpha_j)$. However, using specific characters defined from $\mathbb{Q}(\alpha_j)$ to $\mathbb{Z}_2$, Adleman [1] was able to add a few linear equations to ensure that $\prod_i (a_i - b_i\alpha_j)$ is in fact a square in $\mathbb{Q}(\alpha_j)$. If both algebraic numbers are squares, the following congruence

$$\varphi_1\left(\prod_{i\in S}(a_i - b_i\alpha_1)\right) \equiv \varphi_2\left(\prod_{i\in S}(a_i - b_i\alpha_2)\right) \pmod{n},$$

can be rewritten as a congruence of squares:

$$\varphi_1\left(\sqrt{\prod_{i\in S}(a_i - b_i\alpha_1)}\right)^2 \equiv \varphi_2\left(\sqrt{\prod_{i\in S}(a_i - b_i\alpha_2)}\right)^2 \pmod{n}.$$

If it is possible to extract square roots of gigantic algebraic numbers, this yields a non-trivial factor of n with significant probability. We now comment on the various steps of the algorithm.

3.4.2 Polynomial selection

This is where the GNFS differs from the SNFS: in the SNFS, the polynomials are derived from the special form of the number n to factor. Few polynomial selection methods for the GNFS have been proposed. Most of them are based on the principle described in [10]: first select m around $n^{1/(d+1)}$, where d is a small integer chosen according to the size of n, and expand n in base m. This produces a first polynomial f_1 of degree d, while the second polynomial is set to $f_2(x) = x - m$. Several such pairs are chosen and a bit of sieving is performed to see which pair gives the best results. Recently, Montgomery and Murphy (see [43, 13, 14]) refined this construction: they suggested additional tests for selecting the polynomials based on the number of roots modulo small primes. Polynomial which have many roots modulo small primes, behave much better. Trial and error is still used, once several pairs have been selected. In [14], the authors estimated that the new polynomial selection was four times better than what was expected from the former method. In [22, 23], Montgomery proposed another construction outputting a pair of quadratic polynomials rather than a high-degree polynomial and a linear one. However this construction does not appear optimal.

3.4.3 Sieving

In this stage, one tries to find many pairs of small integers a_i and b_i such that each $a_i - b_i\alpha_j$ is smooth. There are basically two techniques: lattice sieving and line sieving. Assume that the second polynomial f_2 is linear, as is the case in practice. In lattice sieving [50], one fixes a prime q and finds pairs (a_i, b_i) such that $(a_i - b_i\alpha_1)/q$ and $a_i - b_i\alpha_2$ are smooth. This is repeated for many special q's. In line sieving (see [22, 23]), one fixes a value of b_i, and finds values of a_i in a given interval, for which both $a_i - b_i\alpha_1$ and $a_i - b_i\alpha_2$ are smooth. Recents experiments [13, 14] used both lattice sieving and line sieving, although this is somehow redundant. This follows from the fact that the code for lattice sieving does not currently allow large factor bases.

3.4.4 Linear algebra

The linear algebra has a first technical step, which we do not describe in detail, since it depends on many details of the sieving procedure, which we omitted. It consists in deleting equartions and minimizing their number, as well as the number of unknowns. The filtering procedure used in recent experiments [13, 14] is due to Cavallar [12].

The time-consuming part of the linear algebra is devoted to solving a huge sparse linear system modulo 2. Applying structured Gaussian elimination is not recommended when the system is huge, due to memory constraints. Instead, one prefers to use either the Lanczos (see [33]) or Wiedemann [58] algorithms, or one of their variants. These are iterative algorithms: instead of modifying the input matrix, they repeatedly apply the input matrix to a vector. Thus, they only store, besides the original matrix, a few temporary vectors. The Lanczos methods uses a symmetric matrix, but it follows from elementary algebra that this is no restriction. Although it was designed for real-valued matrices, it applies to other fields, unless it encounters a vector orthogonal to itself. To avoid self-orthogonal vectors, it is better to operate in an extension field rather than $GF(2)$ itself. Block variants [15, 16, 42] of both algorithms have been proposed to take advantage of word operations. The method currently in use in NFS factorization experiments is Montgomery's block Lanczos algorithm [42]: it requires $n/(N-0.76)$ iterations where N is word size in bits. Each iteration applies the original matrix, as well as its transpose, to an $n \times N$ matrix and performs a few additional operations, which we do not describe in detail. It should be pointed out that, at the moment, none of the above algorithms can be massively distributed. This is a bottleneck for future NFS factorizations, since the factor base will grow in size, thus leading to larger matrices to be handled at the linear algebra stage.

3.4.5 Square root

Once the linear algebra stage has been performed, one needs to extract square roots of huge algebraic numbers. Let γ be such an algebraic number in the number field $\mathbb{K} = \mathbb{Q}(\alpha)$: γ is actually given as a large product of small algebraic numbers. The first efficient method was found by Couveignes [20]: Couveignes noticed that, when the degree d of the number field $\mathbb{K}$ is odd, there is a way to distinguish between the two square roots of a given number. This allows to compute $\sqrt{\gamma}$ using Chinese remaindering. However, Couveignes's algorithm does not scale up to currently used sizes. A major improvement was later found by Montgomery (see [40, 41, 22, 23, 44]) and is the method used in recent NFS experiments. Computations show that the running time is negligible compared to other stages of the NFS, such as sieving and linear algebra, even if no one currently knows how to prove the fact.

We sketch the basic principle of Montgomery's algorithm and refer to [41, 22, 44] for details. The trick is to look for $\sqrt{\gamma}$, as a large product of small algebraic numbers. *A priori*, it is not clear that such a decomposition exists, and this is one of the reasons why the complexity of Montgomery's method is still unknown. The algorithm tries to find a decomposition through an iterative process. Notice that the ideal factorization of $\langle\gamma\rangle$ is completely known. Since γ is a square, we can thus obtain the ideal factorization of $\sqrt{\langle\gamma\rangle}$. At each step, the algorithm selects a reasonably small integral ideal in either the numerator or denominator of $\sqrt{\langle\gamma\rangle}$, and uses lattice reduction to find an appropriate algebraic integer within this ideal. Even though the small ideal is not in general principal, it turns out that, in practice, it is possible to find an algebraic integer which is almost a generator, and which improves on the iterative approximation of $\sqrt{\gamma}$. After sufficiently many steps, one obtains a reasonably good approximation of $\sqrt{\gamma}$, as a product of appropriate small algebraic integers obtained by lattice reduction. The exact value of the error in the approximation can be found by Chinese remaindering. Finally, a square root $\sqrt{\gamma}$ has been be computed.

3.5 Practical experiments

In practical terms, NFS beats QS for numbers of more than about 110 digits (see [22]). The number field sieve was used to factor the 130-digit RSA challenge number in april 1996, with an amount of computer time which was only a fraction of what was spent on the old 129-digit QS-record. It was later used to factor RSA-140 in february 1999 with an amount of computer time about 2000 Mips-years. In august 1999, the factorization of RSA-155 from the RSA list was obtained ([14]). The complete project took seven calendar months. The polynomial selection took about one month on several fast workstations. The sieving, was done on about 300 fast PCs and workstations during 3.7 months. Filtering the relations and reducing the matrix corresponding to these relations took one calendar month on an SGI Origin 2000 computer. The block Lanczos step to find dependencies in this matrix took 224 CPU hours and 2 Gbytes of central memory on a Cray C916 supercomputer. The final square root step was performed in about two days on an SGI Origin 2000 computer. The amount of computer time spent on this new factoring world record is equivalent to 8000 mips-years, whereas extrapolation based on RSA-140 and the asymptotic complexity formula for NFS predicted approximately 14000 mips-years. As observed in section 3.4.2, the gain was caused by the improved polynomial search method.

bit-size of the modulus	complexity of NFS in $\log_2$
512	63
1024	85
4096	155
6144	182
8192	206

Figure 1: Complexity of factoring

4 Consequences in terms of key sizes

4.1 The current status of factoring

The current status of factoring is well understood. The current record is still at 512 bits. However, factoring technology has spread out: another number of the same bit-size has been factored in October 2000, by a swedish team [2], in answer to a challenge published in a book by Singh [55]. The experiment was carried through by a group which did not consist of experts in the area, and, despite the fact that the running time that they used was not optimal, they were successful. It is thus expected that larger numbers will be soon factored, the main obstacle to a fully scalable implementation of NFS being actually the linear algebra.

4.2 Selecting key sizes

In practical situations, the sizes of cryptographic keys based on the hardness of factoring have to be chosen so that they outreach the expected performances of the factoring algorithms, during the entire lifetime of the system. This involves making predictions. The theoretical complexity of NFS $L_n[1/3; c + o(1)]$, $c \approx 1.92$, is a rough estimate, which is not of much use, for many reasons:

- Due to memory constraints, the size of the factor base is always much less than the optimal choice that theory would dictate.
- Methods used in the implementations differ from those used in the complexity analysis: the polynomial selection appears better in practical terms, while the linear algebra is difficult to handle.

Still, a table of time estimates for the best known factoring method NFS, as shown on figure 1, can provide some indication. To derive an estimate of the key size corresponding to a given workload 2^ℓ, one searches for a modulus for which the righthand side is ℓ. As a typical case, one can select a modulus of 1024 bits to provide a security level over 2^{80}.

In [14], the authors derive the following formula

$$Y = 13.24D^{1/3} + 1928.6$$

for predicting the calendar year for factoring D-digit number by NFS. The same formula appears in [9] and produces $Y = 2018$ for $D = 309$, i.e. for a 1024 bit modulus. Such estimates should lead to immediately ban 768 bit moduli and to adopt 2048 bit moduli for long-term security.

For moduli which are not of the form $n = pq$, for example for multiprime RSA moduli, or for ESIGN moduli of the form p^2q, one should further take into account the results in sections 3.1.3 and 3.2. Our findings is that lattice-based factoring cannot be a threat, if the repeated factor appears with exponent 2 , 3 or 4. Similarly, ECM remains inefficient for moduli with three prime factors. For moduli with 4 prime factors, it is likely that they will be threatened by ECM before they are at hand for NFS, at least for the minimal size 1024 bits. Accordingly, we do not recommend their use.

References

[1] L. M. Adleman. Factoring numbers using singular integers, *Proc. 23rd ACM Symp. on Theory of Computation*, 64–71, 1991.

[2] F. Almgren, G. Andersson, T. Granlund, L. Ivansson and S. Ulfberg. Singh challenge 10.
`http://wwwhome.cs.utwente.nl/ crypto/challenge-10.html`

[3] D. Atkins, M. Graff, A. K. Lenstra and P. Leyland. The magic words are squeeming ossifrage, *Proc. Asiacrypt'94*, *LNCS* 917, (1995), 263–277.

[4] M. Bellare and P. Rogaway. Random Oracles Are Practical: a Paradigm for Designing Efficient Protocols, *Proc. 1st CCS*, 62–73. ACM Press, New York, 1993.

[5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among Notions of Security for Public-Key Encryption Schemes. *Proc. Crypto '98*, *LNCS* 1462, 26–45. Springer-Verlag, Berlin, 1998.

[6] M. Bellare and P. Rogaway, Optimal asymmetric encryption - How to encrypt with RSA, *Proc. Eurocrypt'94*, *LNCS* 950, 1995, 92–111.

[7] D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $N = p^rq$ for large r, *Proc. Crypto'99*, *LNCS* 1666, 1999, 326–337.

[8] D. Boneh and R. Venkatesan. Breaking RSA may not be equivalent to factoring, *Proc. Eurocrypt '98*, LNCS 1402, 59–71, Springer-Verlag, Berlin, 1998.

[9] R. P. Brent. Some Parallel Algorithms for Integer Factorisation, Euro-Par 99, *LNCS* 1685, (1999), 1–22.

[10] J. P. Buhler, H. W. Lenstra, and Carl Pomerance. Factoring integers with the number field sieve, in [35], 50-94.

[11] R. Cramer and V. Shoup. Signature Schemes Based on the Strong RSA Assumption, *6th ACM Conference on Computer and Communication Security*. ACM Press, (1999), 46–51.

[12] S. Cavallar. Strategies for filtering in the Number Field Sieve, *Algorithmic Number Theory – Proc. ANTS-IV*, *LNCS* 1838, Springer-Verlag, 2000.

[13] S. Cavallar, B. Dodson, A. K. Lenstra, P. Leyland, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, and P. Zimmermann. Factorization of RSA-140 using the number field sieve, *Proc. Asiacrypt '99*, *LNCS* 1716, Springer-Verlag, 1999.

[14] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. Putnam, C. Putnam, and P. Zimmermann. Factorization of 512-bit RSA modulus, *Proc. Eurocrypt '00*, *LNCS* 1807, 1–18, Springer-Verlag, 2000.

[15] D. Coppersmith. Solving linear equations over GF(2): block Lanczos algorithm, *Linear Algebra and its Applications*, 192:33–60, 1993.

[16] D. Coppersmith. Solving linear equations over GF(2) via block Wiedemann algorithm, *Math. Comp.*, 62:333–350, 1994.

[17] D. Coppersmith, Finding a Small Root of a Univariate Modular Equation, *Proc. Eurocrypt'96*, *LNCS* 1070, 1996, 178–189.

[18] D. Coppersmith, Finding a Small Root of a Bivariate Integer Equation; Factoring with High Bits Known, *Proc. Eurocrypt'96*, *LNCS* 1070, 1996, 178–189.

[19] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities, *J. Cryptology*, 10, 1997, 233–260.

[20] J.-M. Couveignes. Computing a square root for the number field sieve, in [35], 95–102 .

[21] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography, *Proc. 23rd STOC.* ACM Press, New York, 1991.

[22] R.-M. Elkenbracht-Huizing. An implementation of the number field sieve. it Exp. Math. 5, 1996, 231–253.

[23] M. Elkenbracht-Huizing. A multiple polynomial general number field sieve. *Algorithmic Number Theory – Proc. ANTS-II*, *LNCS* 1122, 101–116, Springer-Verlag, 1996.

[24] R.-M. Elkenbracht-Huizing. An implementation of the number field sieve, it Exp. Math. 5, 1996, 231-253.

[25] NTT Corporation. ESIGN Specification, submission to CRYPTREC, September 2001.

[26] NTT Corporation. Self evaluation of ESIGN. Submission to CRYPTREC, September 2001.

[27] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions of Identification and Signature Problems, *Proc. Crypto '86*, *LNCS* 263, 186–194. Springer-Verlag, Berlin, 1987.

[28] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA–OAEP is Still Alive, Cryptology ePrint Archive 2000/061, `http://eprint.iacr.org/`.

[29] A. Hildebrand and G. Tenenbaum. Integers without large prime factors, *J. Théor. Nombres Bordeaux*, (5):411–484, 1993

[30] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited, Cryptography and Coding, *LNCS* 1355, 1997, 131–142.

[31] A. Joux and J. Stern. Lattice Reduction: a Toolbox for the Cryptanalyst, *J. Cryptology* 11, 1998, 161–186

[32] C. S. Jutla. On finding small solutions of modular multivariate polynomial equations, *Proc. Eurocrypt '98*, *LNCS* 1233, 158–170.

[33] B. A. Lamacchia and A. M. Odlyzko. Solving large sparse systems over finite fields, *Proc. Crypto '90*, *LNCS* 537, Springer-Verlag, 1990.

[34] A. K. Lenstra, H. W. Lenstra and L. Lovász. Factoring polynomials with rational coefficients, *Mathematische Ann.*, 261, (1982), 513–534.

[35] A. K. Lenstra and H. W. Lenstra, Jr. *The Development of the Number Field Sieve*, Lecture Notes in Mathematics 1554, Springer-Verlag, 1993.

[36] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve, 11-42 in [35].

[37] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The factorization of the ninth Fermat number, *Math. Comp.*, 61:319–349, 1993.

[38] A.K. Lenstra and E. Verheul. Selecting cryptographic key sizes, *Proc. PKC'2000*, *LNCS* 1751, 446–465, 2000.

[39] N. Lygeros, M. Mizony, P. Zimmermann, A new ECM record with 54 digits, `http://www.desargues.univ-lyon1.fr/home/lygeros/Mensa/ecm54.html`

[40] P. L. Montgomery. Square roots of products of algebraic numbers. Walter Gautschi, editor, *Mathematics of Computation 1943-1993: a Half-Century of Computational Mathematics*, Proceedings of Symposia in Applied Mathematics, 567–571, American Mathematical Society, 1994.

[41] P. L. Montgomery. Square roots of products of algebraic numbers, Draft of June, 1995. Available at `ftp://ftp.cwi.nl/pub/pmontgom/sqrt.ps.gz`.

[42] P. L. Montgomery. A block Lanczos algorithm for finding dependencies over GF(2), *Proc. Eurocrypt'95*, *LNCS* 921, 1995, 106–120.

[43] B. Murphy. Modelling the yield of number field sieve polynomials, *Algorithmic Number Theory – Proc. ANTS-III*, *LNCS* 1423, Springer-Verlag, 1998.

[44] P. Nguyen. A Montgomery-like square root for the number field sieve, *Algorithmic Number Theory – Proc. ANTS-III*, *LNCS* 1423, Springer-Verlag, 1998.

[45] T. Okamoto, E. Fujisaki and H. Morita. TSH-ESIGN: Efficient Digital Signature Scheme Using Trisection Size Hash. Submission to P1363a, 1998.

[46] D. Panario, X. Gourdon, and P. Flajolet. An analytic approach to smooth polynomials over finite fields, *Algorithmic Number Theory – Proc. ANTS-III*, LNCS 1423. Springer-Verlag, 1998.

[47] R. Peralta and E. Okamoto. Faster Factoring of Integers of a Special Form , IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences, v. E79-A, n.4 (1996), 489–493.

[48] J. Pollard. Monte Carlo methods for index computation mod p, *Math. Comp.*, 32, 918–924, 1978.

[49] J. M. Pollard. Factoring with cubic integers, in [35], 4–11.

[50] J. M. Pollard. The lattice sieve, in [35], 43–49.

[51] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems, *Communications of the ACM*, 21(2):120–126, February 1978.

[52] R. L. Rivest, A. Shamir, L. M. Adleman. Cryptographic Communications System and Method, US patent 4 405 829, September 20, 1983 (filed 14/12/1977).

[53] RSA Laboratories. Information on the RSA challenge, `http://www.rsa.com/rsalabs/html/challenges/html`

[54] RSA Laboratories. PKCS # 1 Version 2.1: RSA Cryptography Standard. Draft, January 2001, `http://www.rsa.com/rsalabs/pubs/PKCS/`.

[55] S. Singh. The Code Book, 1999.

[56] T. Takagi. Fast RSA-Type Cryptosystem Modulo $p^k q$, *Proc. Crypto'98*, *LNCS* 1462, (1998), 318-326.

[57] P.C. van Oorschot and M. J. Wiener. Parallel collision search with cryptanalytic applications, *J. Cryptology*, 12, (1999), 1–28.

[58] D. H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inform. Theory*, 32:54–62, 1986.