Encyclopedia of Information Security **Entries** related to integer factoring

Arjen K. Lenstra
arjen.lenstra@citigroup.com

Version: 0.1

August 30, 2007

Birthday paradox

The birthday paradox refers to the fact that there is a probability of more than 50% that among a group of at least 23 randomly selected people at least two have the same birthday. It follows from

$$\frac{365}{365} \cdot \frac{365-1}{365} \cdot \cdots \cdot \frac{365-22}{365} \approx 0.49 < 0.5;$$

it is called a paradox because the 23 is felt to be unreasonably small compared to 365. More in general, it follows from

$$\prod_{0 \le i \le 1.18\sqrt{p}} \frac{p-i}{p} \approx 0.5$$

that it is not unreasonable to expect a duplicate after about $\sqrt{p}$ elements have been picked at random (and with replacement) from a set of cardinality p.

Continued fraction method

See **Integer factoring**.

Factor base

See **relation collection** in **Integer factoring**.

Factoring circuits

In [1] Daniel Bernstein proposed a new approach to the relation combination step of the number field sieve (see **Integer factoring**), based on sorting in a large mesh of small processors. A variant based on routing in a similar mesh was later described in [2]. A hardware approach to relation collection based on more traditional methods was also proposed in [1]; a more detailed hardware design is described in [3]. The practical implications of these new hardware-based approaches for the security of, for instance, 1024-bit **RSA moduli** remain to be seen.

An alternative way to measure the cost of integer factoring was presented in [1] as well, namely as the product of the runtime and the cost of the hardware required, as opposed to just the runtime (see also [4]). Refer to [1] and [2] for a discussion of the effect this has on the parameter choices and asymptotic cost of the **Number field sieve**.

References

[1] D.J. Bernstein, *Circuits for integer factorization: a proposal*, manuscript, November 2001; available at cr.yp.to/papers.html#nfscircuit.

[2] A.K. Lenstra, A. Shamir. J. Tomlinson, E. Tromer, *Analysis of Bernstein's factorization circuit*, Proceedings Asiacrypt 2002, LNCS 2501, Springer-Verlag 2003, 1-26.

[3] A. Shamir, E. Tromer, *Title?: Hardware design for relation collection*, Asiacrypt 2002, rump session, December 2002.

[4] M.J. Wiener, *The full cost of cryptanalytic attacks*, accepted for publication in J. of Cryptology.

Integer factoring

Definition. Integer factoring is the following problem: given a positive composite integer n, find positive integers v and w, both greater than 1, such that $n = v \cdot w$.

Relation to information security. Integer factoring is widely assumed to be a hard problem. Obviously, it is not hard for all composites, but composites for which it is believed to be difficult can easily be generated. This belief underlies the security of the **RSA public key cryptosystem**. To the present day, no proof of the difficulty of factoring has been published. This is quite unlike the **Discrete logarithm problem**, where the difficulty is provable for a generic group [19, 27]. However, this result does not have much practical relevance. In particular it does not say anything about the hardness of computing discrete logarithms in multiplicative groups of finite fields, a problem that is widely regarded as being as hard (or as easy) as integer factoring. On a quantum computer both problems are easy in the sense that they allow polynomial-time solutions. Given the current state of the art in quantum computer manufacturing, this is not yet considered to be a threat affecting factoring or discrete logarithm based cryptosystems. Quantum computer factoring is not discussed here.

Methods for integer factorization. RSA cryptosystems are faster when smaller composites are used, but believed to be more secure for larger ones. Finding the right middle-ground between efficiency and security requirements requires study of theoretical and practical aspects of integer factorization methods. Often, two types of integer factoring methods are distinguished: general purpose and special purpose methods. For general purpose methods the factoring effort depends solely on the size of the composite n to be factored. For special purpose methods properties of n (mostly but not always of one of the factors of n) come into play as well. RSA composites are generally chosen in such a way that special purpose methods would be less efficient than general purpose ones. Special purpose methods are therefore hardly relevant for RSA composites. For randomly selected composites, however, special purpose methods are on average very effective. For example, almost 92% of all positive integers have a factor < 1000; if such a factor exists it will be found very quickly using trial division, the simplest of the special purpose methods (see below).

Here the following factoring methods are sketched:
special purpose: trial division; Pollard's rho method; Pollard's $p-1$ method and generalizations thereof; **Elliptic curve method**.
general purpose: Fermat's method and congruence of squares; Dixon's random squares method; continued fraction method (CFRAC); linear sieve; **Quadratic sieve**; **Number field sieve**.
For a more complete survey refer to [6] and the references therein.
Establishing compositeness. Fermat's little theorem says that $a^{p-1} \equiv 1 \bmod p$ if p is prime and a is a positive integer $< p$. Thus, an $a \in \{1, 2, \ldots, n-1\}$ for which $a^{n-1} \not\equiv 1 \bmod n$ would establish the compositeness of n at the cost of a single exponentiation modulo n. The proof of compositeness does not provide any information that may be useful to find a non-trivial factor of n. Also, this type of compositeness proof does not work for all composites, because for some composites $a^{n-1} \equiv 1 \bmod n$ for all a that are coprime to n. There are infinitely many of such composites, the so-called Carmichael numbers [1].

Fermat's little theorem allows an alternative formulation for which the converse is always useful for compositeness testing. Let $n - 1 = 2^t \cdot u$ for integers t and u with u odd. If $n > 2$ were prime, then any integer $a \in \{2, 3, \ldots, n-1\}$ satisfies the condition that either $a^u \equiv 1 \bmod n$ or $a^{2^i u} \equiv -1 \bmod n$ for some $i \in \{0, 1, \ldots, t-1\}$. An integer $a \in \{2, 3, \ldots, n-1\}$ for which this condition does not hold is called a 'witness to the compositeness of n.' For odd composite n at least 75% of the numbers in $\{2, 3, \ldots, n-1\}$ are witnesses to their compositeness [24]. Therefore, it can in general be expected that n's compositeness can be proved at the cost of at most a few exponentiations modulo n, simply be trying elements of $\{2, 3, \ldots, n-1\}$ at random until a witness has been found. This probabilistic compositeness test is often referred to as the Rabin/Miller primality test. If n itself is randomly selected too (as may happen during the search for a prime number), it is usually faster to establish its compositeness using trial division (see below).
Distinct factors. Let a be a witness to the compositeness of n, as above. If $\gcd(a^n - a, n) = 1$, then n is not a prime power. Consequently, n has at least two distinct prime factors. Because $(a^{2^{t-1}u})^2 = a^{n-1}$, with t and u as above, this fact can often be established at negligible additional cost.
Repeated factors. If n is an odd composite and not a prime power, it may still be a proper power of a composite (i.e., $n = m^\ell$ for $m, \ell \in \mathbf{Z}_{>1}$ with m

composite) or it may properly contain a square (i.e., $n = m^\ell \cdot w$ for $m, \ell, w \in \mathbf{Z}_{>1}$ with $\gcd(m, w) = 1$). Proper powers can be recognized by approximating ℓ-th roots of n for $1 \le \ell \le [\frac{\log n}{\log 3}]$ using a numerical method. At present there is in general no better way to find out if n properly contains a square than factoring n.

Trial division up to bound B is the process of checking for all primes $\le B$ in succession if they divide n, until the smallest prime factor p of n is found or until it is shown that $p > B$. This takes time proportional to $\log n \cdot \min(p, B)$. For randomly selected n trial division can be expected to be very effective. It cannot be recommended to use B larger than, say, 10^6 (with the precise value depending on the relative speeds of implementations) because larger p can be found more efficiently using one of the methods described below.

Pollard's rho method [21] is based on the **Birthday paradox**: if $x_0, x_1, x_2, \ldots$ is a random walk on $\mathbf{Z}/p\mathbf{Z}$, then for any p there is a fair probability that $x_i = x_j$ for some indices $i \ne j$ up to about $\sqrt{p}$. Similarly, if $x_0, x_1, x_2, \ldots$ is a random walk on $\mathbf{Z}/n\mathbf{Z}$, then for any $p < n$ there is a fair probability for a collision $x_i \equiv x_j \bmod p$ for $i \ne j$ up to about $\sqrt{p}$; if p is an unknown divisor of n, such a collision can be recognized because it implies that p divides $\gcd(n, x_i - x_j)$.

In Pollard's rho method a walk on $\mathbf{Z}/n\mathbf{Z}$ is defined by selecting $x_0 \in \mathbf{Z}/n\mathbf{Z}$ at random and by defining $x_{i+1} = x_i^2 + 1 \bmod n$. There is no a priori reason why this would define a random walk on $\mathbf{Z}/n\mathbf{Z}$, but if it does it may reveal the smallest factor p of n after only about $\sqrt{p}$ iterations. At the i-th iteration, this would require $i - 1$ gcd-computations $\gcd(n, x_i - x_j)$ for $j < i$, making the method slower than trial division. This problem is overcome by means of Floyd's cycle-finding method: at the i-th iteration compute just $\gcd(n, x_i - x_{2i})$ (thus requiring computation of not only x_i but x_{2i} as well). As a result, and under the assumption that the walk is random, the expected time to find p becomes proportional to $(\log n)^2 \cdot \sqrt{p}$; this closely matches practical observations. The name of the method is based on the shape of the Greek character rho ('ρ') depicting a sequence that bites in its own tails. The method is related to **Pollard's rho method for discrete logarithms**.

In practice the gcd-computation per iteration is replaced by a single gcd-computation of n and the product modulo n of, say, 100 consecutive $(x_i - x_{2i})$'s. In the unlikely event that the gcd turns out to be equal to n, one backs up and computes the gcd's more frequently. See also [16].

Pollard's $p - 1$ method [20]. It follows from Fermat's little theorem that if a is coprime to a prime p and k is an integer multiple of $p - 1$, then $a^k \equiv 1 \bmod p$. Thus, if p is a prime factor of n, then p divides either $\gcd(a, n)$ or $\gcd(a^k - 1, n)$ where a is randomly selected from $\{2, 3, \ldots, n-2\}$. This means that primes p dividing n for which $p - 1$ is B-smooth (**Smoothness**), may be found by selecting an integer $a \in \{2, 3, \ldots, n-2\}$ at random, checking that $\gcd(a, n) = 1$, and computing $\gcd(a^k - 1, n)$ where k is the product of the primes $\le B$ and appropriately chosen small powers thereof. This takes time proportional to $(\log n)^2 \cdot B$. In a 'second stage' one may successively try $k \cdot q$ as well for the primes q between B and B', thereby finding p for which $p - 1$ is the product of a B-smooth part and a single larger prime factor up to B'; the additional cost is proportional to $B' - B$.

For n with unknown factorization, the best values B and B' are unknown too and in general too large to make the method practical. However, one may try values B, B' depending on the amount of computing time one finds reasonable and turn out to be lucky; if not one gives up as far as Pollard's $p-1$ method is concerned. Despite its low probability of success, the method is quite popular, and has led to some surprising factorizations.

Generalizations of Pollard's $p-1$ method. Pollard's $p-1$ method is the special case $d = 1$ of a more general method that finds a prime factor p of n for which the order $p^d - 1$ of the multiplicative group $\mathbf{F}_{p^d}^*$ of $\mathbf{F}_{p^d}$ is smooth. Because

$$X^d - 1 = \prod_{t \text{ dividing } d} \Phi_t(X)$$

where $\Phi_t(X)$ is the t-th cyclotomic polynomial (thus, $\Phi_1(X) = X - 1$, $\Phi_2(X) = \frac{X^2-1}{X-1} = X + 1$, $\Phi_3(X) = \frac{X^3-1}{X-1} = X^2 + X + 1$, etc.), the order of $\mathbf{F}_{p^d}^*$ is smooth if and only if $\Phi_t(p)$ is smooth for all integers t dividing d. For each t the smoothness test (possibly leading to a factorization of n) consists of an exponentiation in a ring modulo n that contains the order $\Phi_t(p)$ subgroup of the multiplicative group of the subfield $\mathbf{F}_{p^t}$ of $\mathbf{F}_{p^d}$. For $t = d = 2$ the method is known as Williams' $p + 1$ method [31]; for general d it is due to Bach and Shallit [3].

Usage of 'strong primes' in RSA. It is not uncommon, and even prescribed in some standards, to use so-called **strong primes** as factors of RSA moduli. These are primes for which both $p-1$ and $p+1$ have a very large prime factor, rendering ineffective a $p-1$ or $p+1$ attack against the modulus. This approach overlooks other $\Phi_t(p)$ attacks (which, for random moduli, have an even smaller probability of success). More importantly it overlooks the fact that the resulting RSA modulus is just as likely to be vulnerable to a single elliptic curve when using the **Elliptic curve factoring method**. It follows that usage of strong primes does in general not make RSA moduli more resistant against factoring attacks. See also [26].

Cycling attacks against RSA. These attacks, also called 'superencryption attacks' work by repeatedly re-encrypting an RSA ciphertext, in the hope that after k re-encryptions (for some reasonable k) the original ciphertext appears. They are used as an additional reason why **strong primes** should be used in RSA. However, it is shown in [26] that a generalized and more efficient version of cycling attacks can be regarded as a special purpose factoring method that is successful only if all prime factors of $p - 1$ are contained in $e^k - 1$ for one of the primes p dividing n, where e is the RSA public exponent. The success probability of this attack is therefore small, even when compared to the success probability of Pollard's $p - 1$ method.

Elliptic curve method [13]. The success of Pollard's $p - 1$ method (or its generalizations) depends on the smoothness of the order of one of the groups $\mathbf{F}_p^*$ (or $\mathbf{F}_{p^d}^*$) with p ranging over the prime factors of n. Given n, the group orders are fixed, so the method works efficiently for some n but for most n it would require too much computing time. In the elliptic curve method each fixed group $\mathbf{F}_p^*$ of fixed order (given n) is replaced by the group E_p of points modulo p of an elliptic curve modulo n. For randomly selected elliptic curves modulo n, the order $\#E_p$ of E_p behaves as a random number close to p. If $\#E_p$ is smooth, then p can efficiently be found using arithmetic in the group of points of the elliptic curve modulo n. It is conjectured that the smoothness behavior of $\#E_p$ is similar to that of ordinary integers of that size (**Smoothness probability**), which implies that the method works efficiently for all n. It also implies that the method can be expected to find smaller factors faster than larger ones. In the worst case where n is the product of two primes of about the same size the heuristic expected runtime is $L_n[1/2, 1]$, with L_n as in ***L* function**; this is subexponential in $\log(n)$. See **Elliptic curve method** for a more complete description and more detailed expected runtimes.

Fermat's method and congruence of squares. Fermat's method attempts to factor n by writing it as the difference of two integer squares. Let $n = p \cdot q$ for odd p and q with $p < q$, so that $q - p = 2y$ for an integer y. With $x = p + y$ it follows that $n = (x - y)(x + y) = x^2 - y^2$. Thus, if one tries $x = [\sqrt{n}] + 1$, $[\sqrt{n}]+2$, $[\sqrt{n}]+3$, ... in succession until $x^2 - n$ is a perfect square, one ultimately finds $x^2 - n = y^2$. This is efficient only if the resulting y, the difference between the factors, is small; if it is large the method is inferior even to trial division.

Integers x and y that satisfy the similar but weaker condition

$$x^2 \equiv y^2 \bmod n$$

may also lead to a factorization of n: from the fact that n divides $x^2 - y^2 = (x - y)(x + y)$ it follows that

$$n = \gcd(n, x - y)\gcd(n, x + y).$$

If x and y are random solutions to $x^2 \equiv y^2 \bmod n$, then there is a probability of at least 50% that this yields a non-trivial factorization of n. All general purpose factoring methods described below work by finding 'random' solutions to this equation.

The Morrison-Brillhart approach. To construct solutions to $x^2 \equiv y^2 \bmod n$ that may be assumed to be sufficiently random, Kraïtchik in the 1920s proposed to piece together solutions to $x^2 \equiv a \bmod n$. In the Morrison-Brillhart approach this is achieved using the following two steps [18]:

Relation collection. Fix a set P of primes (often called the 'factor base'), and collect a set V of more than $\#P$ integers v such that

$$v^2 \equiv \left(\prod_{p \in P} p^{e_{v,p}}\right) \bmod n$$

with $e_{v,p} \in \mathbf{Z}$. These identities are often called 'relations' modulo n. If P is the set of primes $\leq B$, then v's such that $v^2 \bmod n$ is B-smooth lead to relations. For each v the exponents $e_{v,p}$ are regarded as a $\#P$-dimensional vector, denoted $(e_{v,p})_{p \in P}$.

Relation combination. Because $\#V > \#P$, the $\#P$-dimensional vectors $(e_{v,p})_{p \in P}$ are linearly dependent and there exist at least $\#V - \#P$ linearly independent subsets S of V such that

$$\sum_{v \in S} e_{v,p} = 2(s_p)_{p \in P} \text{ with } (s_p)_{p \in P} \in \mathbf{Z}^{\#P}.$$

These subsets S with corresponding vectors $(s_p)_{p \in P}$ give rise to at least $\#V - \#P$ independent solutions to $x^2 \equiv y^2 \bmod n$, namely

$$x = \left(\prod_{v \in S} v\right) \bmod n, \quad y = \left(\prod_{p \in P} p^{s_p}\right) \bmod n,$$

and thereby at least $\#V - \#P$ independent chances to factor n.

All current general purpose factoring methods are based on the Morrison-Brillhart approach. They differ in the way the relations are collected, but they are all based on, more or less, the same relation combination step.

Matrix step. Because S and $(s_p)_{p \in P}$ as above can be found by looking for linear dependencies modulo 2 among the rows of the $(\#V \times \#P)$-matrix $(e_{v,p})_{v \in V, p \in P}$, the relation combination step is often referred to as the 'matrix step.' With Gaussian elimination the matrix step can be done in $(\#P)^3$ steps (since $\#V \approx \#P$). Faster methods, such as conjugate gradient, Lanczos, or Wiedemann's coordinate recurrence method, require $O(w \cdot \#P)$ steps, where w is the number of non-zero entries of the matrix $(e_{v,p} \bmod 2)_{v \in V, p \in P}$. See [5, 11, 17, 23, 29, 30] for details.

In the various runtime analyses below, $\#P$ is measured using the **L function** and w turns out to be $c \cdot \#P$ for a c that disappears in the $o(1)$ of the L function, so that the runtime $O(w \cdot \#P)$ simplifies to $(\#P)^2$.

Dixon's random squares method [8]. The simplest relation collection method is to define P as the set of primes $\leq B$ for some bound B and to select different v's at random from $\mathbf{Z}/n\mathbf{Z}$ until more than $\pi(B)$ ones have been found for which $v^2 \bmod n$ is B-smooth. The choice of B, and the resulting expected runtime, depends on the way the values $v^2 \bmod n$ are tested for B-smoothness. If smoothness is tested using trial division, then $B = L_n[1/2, 1/2]$ (with L_n as in **L function**). For each candidate v, the number $v^2 \bmod n$ is assumed to behave as a random number $\leq n = L_n[1, 1]$, and therefore, according to **Smoothness probability**, B-smooth with probability $L_n[1/2, -1]$. Testing each candidate for B-smoothness using trial division takes time $\#P = \pi(B) = L_n[1/2, 1/2]$ (using the properties of L_n as set forth in **L function**), so collecting somewhat more than $\#P$ relations can be expected to take time

$$\overbrace{L_n[1/2, 1/2]}^{\substack{\text{number}\\\text{of relations}\\\text{to be collected}}} \cdot \overbrace{L_n[1/2, 1/2]}^{\substack{\text{trial}\\\text{division}}} \cdot \overbrace{(L_n[1/2, -1])^{-1}}^{\substack{\text{inverse of}\\\text{smoothness}\\\text{probability}}} = L_n[1/2, 2].$$

Gaussian elimination on the $\#V \times \#P$ matrix takes time

$$L_n[1/2, 1/2]^3 = L_n[1/2, 3/2].$$

Because at most $\log_2(n)$ entries are non-zero for each vector $(e_{v,p})_{p\in P}$, the total number of non-zero entries of the matrix is $\#V \cdot \log_2(n) = L_n[1/2, 1/2]$ and the matrix step can be done in

$$L_n[1/2, 1/2]^2 = L_n[1/2, 1]$$

steps using Lanczos or Wiedemann algorithms. In either case the runtime is dominated by relation collection and the total expected time required for Dixon's method with trial division is $L_n[1/2, 2]$. Unlike most methods described below, the expected runtime of the trial division variant of Dixon's method can rigorously be proved, i.e., it does not depend on any heuristic arguments or conjectures.

If B-smoothness is tested using the elliptic curve method, the time to test each $v^2 \bmod n$ is reduced to $L_n[1/2, 0]$: the entire cost of the smoothness tests disappears in the $o(1)$. As a result the two stages can be seen to require time $L_n[1/2, 3/2]$ each when Gaussian elimination is used for the matrix step. In this case, i.e., when using the elliptic curve method for smoothness testing, the runtime can be further reduced by using Lanczos or Wiedemann methods and a different value for B. Redefine B as $L_n[1/2, \sqrt{1/2}]$ so that relation collection takes time

$$L_n[1/2, \sqrt{1/2}] \cdot L_n[1/2, 0] \cdot (L_n[1/2, -\sqrt{1/2}])^{-1} = L_n[1/2, \sqrt{2}]$$

and the matrix step requires $L_n[1/2, \sqrt{1/2}]^2 = L_n[1/2, \sqrt{2}]$ steps. The overall runtime of Dixon's method becomes

$$L_n[1/2, \sqrt{2}] + L_n[1/2, \sqrt{2}] = L_n[1/2, \sqrt{2}] :$$

asymptotically relation collection and combination are equally expensive. As described here, the expected runtime of this elliptic curve based variant of Dixon's method depends on the conjecture involved in the expected runtime of the elliptic curve method. It is shown in [22], however, that the expected runtime of a variant of the elliptic curve smoothness test can rigorously be proved. That leads to a rigorous $L_n[1/2, \sqrt{2}]$ expected runtime for Dixon's method.

Continued fraction method (CFRAC) [18]. The quadratic residues $v^2 \bmod n$ in Dixon's method provably behave with respect to smoothness probabilities as random non-negative integers less than n. That allows the rigorous proof of the expected runtime of Dixon's method. However, this theoretical advantage is not a practical concern. It would be preferable to generate smaller quadratic residues, thereby improving the smoothness chances and thus speeding up relation collection, even though it may no longer be possible to rigorously prove the expected runtime of the resulting method. The earliest relation collection method where quadratic residues were generated that are substantially smaller than n was due to Morrison and Brillhart and is based on the use of continued fractions; actually, this method (dubbed 'CFRAC') predates Dixon's method.

If a_i/b_i is the ith continued fraction convergent to $\sqrt{n}$, then $|a_i^2 - nb_i^2| < 2\sqrt{n}$. Thus, if v is chosen as a_i for $i = 1, 2, \dots$ in succession, then $v^2 \bmod n = a_i^2 - nb_i^2$ is a quadratic residue modulo n that is $< 2\sqrt{n}$ and thus much smaller than n. In practice this leads to a substantially larger smoothness probability than in Dixon's method, despite the fact that if prime p divides $v^2 \bmod n$, then $(a_i/b_i)^2 \equiv n \bmod p$ so that n is a quadratic residue modulo p. With $B = L_n[1/2, 1/2]$, P the set of primes $p \leq B$ with $(\frac{n}{p}) = 1$, and elliptic curve smoothness testing, the heuristic expected runtime becomes $L_n[1/2, 1]$. The heuristic is based on the assumption that the residues $v^2 \bmod n$ behave, with respect to smoothness properties, as ordinary random integers $\leq n$ and that the set of primes $p \leq B$ for which $(\frac{n}{p}) \neq 1$ does not behave unexpectedly. In that case, when the L function is used to express smoothness probabilities, the difference with truly random integers disappears in the $o(1)$.

In [14] it is shown how this same expected runtime can be achieved rigorously (by a method that is based on the use of class groups). If elliptic curve smoothness testing is replaced by trial division, $B = L_n[1/2, \sqrt{1/8}]$ is optimal and the heuristic expected runtime becomes $L_n[1/2, \sqrt{2}]$.

Note on the size of RSA moduli. In the mid 1970s CFRAC (with trial division based smoothness testing) was the factoring method of choice. Strangely, at that time, noone seemed to be aware of its (heuristic) subexponential expected runtime $L_n[1/2, \sqrt{2}]$. Had this been known by the time the RSA challenge [9] was posed, Ron Rivest may have based his runtime estimates on CFRAC instead of Pollard's rho (with its exponential expected runtime) [25], come up with more realistic estimates for the difficulty of factoring a 129-digit modulus, and could have decided that 129 digits were too close for comfort (as shown in [2]). As a result, 512-bit RSA moduli may have become less popular.

Linear sieve. It was quickly realized that the practical performance of CFRAC was marred by the trial division based smoothness test. In the late 1970s Richard Schroeppel therefore developed a new way to generate relatively small residues modulo n that can be tested for smoothness very quickly: look for small integers i, j such that

$$f(i,j) = (i + [\sqrt{n}])(j + [\sqrt{n}]) - n \approx (i + j)\sqrt{n}$$

is smooth. Compared to CFRAC the residues are somewhat bigger, namely $(i + j)\sqrt{n}$ as opposed to $2\sqrt{n}$. But the advantage is that smoothness can be tested for many i, j simultaneously using a sieve: if p divides $f(i,j)$ then p divides $f(i + kp, j + \ell p)$ for any $k, \ell \in \mathbf{Z}$. This means that if $f(i,j)$ is tested for B-smoothness for $0 \leq i < I$ and $0 \leq j < J$, the smoothness tests no longer take time $I \cdot J \cdot \pi(B) \approx I \cdot J \cdot B / \log B$, but

$$\sum_{p \leq B} \sum_{0 \leq i < I} \sum_{0 \leq j < J} \frac{1}{p} = O(I \cdot J \cdot \log\log(B)).$$

This leads to a heuristic expected runtime $L_n[1/2, 1]$. Inconveniently, $(i +$

$[\sqrt{n}])(j + [\sqrt{n}])$ is not automatically a square, which means that for all values $i + [\sqrt{n}]$ and $j + [\sqrt{n}]$ that occur in smooth $f(i,j)$'s columns have to be included in the matrix. The effect this has on the expected runtime disappears in the $o(1)$ in L_n.

This method, dubbed 'linear sieve,' was the first factoring method that was heuristically shown (by Schroeppel) to have subexponential expected runtime. (That the earlier CFRAC also had subexponential expected runtime was realized only later; see also [10].) Its main historical significance is, however, that it led to the **Quadratic sieve**, for many years the world's most practical factoring method.

Quadratic sieve. The first crude version of the quadratic sieve was due to Carl Pomerance who realized that it may be profitable to take $i = j$ in Schroeppel's linear sieve. Although smoothness could still be tested quickly using a sieve and the heuristic expected runtime (with sieving) turned out to be a low $L_n[1/2, 1]$, in practice the method suffered from deteriorating smoothness probabilities (due to the linear growth of the quadratic residue $f(i,i)$). This problem was, however, quickly overcome by Jim Davis and Diane Holdridge which led to the first factorization of a number of more than 70 decimal digits [7]. Since then the method has been embellished in various ways (most importantly by Peter Montgomery's multiple polynomial version, as described in [28]) to make it even more practical. See **Quadratic sieve** for details. At this point the largest factorization obtained using quadratic sieve is the 135-digit factorization reported in [15].

Number field sieve. Until the late 1980s the best factoring methods, including the most practical one (quadratic sieve), shared the same expected runtime $L_n[1/2, 1]$ despite the fact that the underlying mathematics varied considerably: heuristically for quadratic and linear sieve, CFRAC, and the worst case of the elliptic curve method, and rigorously for the class group method from [14]. This remarkable coincidence fostered the hope among users of the RSA cryptosystem that $L_n[1/2, 1]$, halfway between linear time $\log n$ and exponential time n (***L* function**), is the 'true' complexity of factoring.

The situation changed, slowly, when in late 1988 John Pollard distributed a letter to a handful of colleagues. In it he described a novel method, still based on the Morrison-Brillhart approach, to factor integers close to a cube and expressed his hope that, one day, the method may be used to factor the ninth Fermat number $F_9 = 2^{2^9} + 1$, back then the world's 'most wanted' composite. It was quickly established that for certain 'nice' n Pollard's new method should work in heuristic expected runtime $L_n[1/3, (\frac{32}{9})^{1/3}] \approx L_n[1/3, 1.526]$. This was the first indication that, conceivably, the complexity of factoring would not be stuck at $L_n[1/2, \ldots]$. The initial work was soon followed by the factorization of several large 'nice' integers, culminating in 1990 in the factorization of F_9 [12]. Further theoretical work removed the 'niceness' restriction and led to the method that is now referred to as the 'number field sieve:' a general purpose factoring method with heuristic expected runtime $L_n[1/3, (\frac{64}{9})^{1/3}] \approx L_n[1/3, 1.923]$. The method as it applies to 'nice' numbers is now called the 'special number field sieve.' See **Number field sieve** for details. The first time a 512-bit RSA modulus was factored, using the number field sieve, was in 1999 [4].

With hindsight the property that all $L_n[1/2, 1]$ factoring methods have in common is their dependence, in one way or another, on smoothness of numbers of order $n^{O(1)}$. The number field sieve breaks through the $n^{O(1)}$ barrier and depends on smoothness of numbers of order $n^{o(1)}$.

References

[1] W.R. Alford, A. Granville, C. Pomerance, *There are infinitely many Carmichael numbers*, Ann. of Math. **140** (1994) 703-722.

[2] D. Atkins, M. Graff, A.K. Lenstra, P.C. Leyland, *THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE*, Proceedings Asiacrypt'94, LNCS 917, Springer-Verlag 1995, 265-277.

[3] E. Bach, J. Shallit, *Cyclotomic polynomials and factoring*, Math. Comp. **52** (1989) 201-219.

[4] S. Cavallar, B. Dodson, A.K. Lenstra, W. Lioen, P.L. Montgomery, B. Murphy, H. te Riele, et al., *Factorization of a 512-bit RSA modulus*, Proceedings Eurocrypt 2000, LNCS 2807, Springer-Verlag 2000, 1-18.

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

[6] R.E. Crandall, C. Pomerance, *Prime numbers: a computational perspective*, Springer-Verlag 2001.

[7] J.A. Davis, D.B. Holdridge, *Factorization using the quadratic sieve algorithm*, Tech Report SAND 8301346, Sandia National Laboratories, Albuquerque, NM, 1983.

[8] J.D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. **36** (1981) 255-260.

[9] M. Gardner, *Mathematical games, A new kind of cipher that would take millions of years to break*, Scientific American (August 1977) 120-124.

[10] D.E. Knuth, *The art of computer programming, Volume 2, Seminumerical algorithms*, third edition, Addison-Wesley, 1998.

[11] B.A. LaMacchia, A.M. Odlyzko, *Solving large sparse linear systems over finite fields*, Proceedings Crypto'90, LNCS 537, Springer-Verlag 1990, 109-133.

[12] A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M. Pollard, *The factorization of the ninth Fermat number*, Math. Comp. **61** (1993) 319-349.

[13] H.W. Lenstra, Jr, *Factoring integers with elliptic curves*, Ann.of Math. **126** (1987) 649-673.

[14] H.W. Lenstra, Jr, C. Pomerance, *A rigorous time bound for factoring integers*, J. Amer. Math. Soc. **5** (1982) 483-516.

[15] P. Leyland, A.K. Lenstra, B. Dodson, A. Muffett, S. Wagstaff, *MPQS with three large primes*, Proceedings ANTS V, LNCS 2369 (2002) 448-462.

[16] P.L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Math. Comp. **48** (1987) 243-264.

[17] P.L. Montgomery, *A block Lanczos algorithm for finding dependencies over GF(2)*, Proceedings Eurocrypt'95, LNCS 925, Springer-Verlag 1995, 106-120.

[18] M.A. Morrison, J. Brillhart, *A method of factorization and the factorization of F_7*, Math. comp. **29** (1975) 183-205.

[19] V.I. Nechaev, *Complexity of a determinate algorithm for the discrete logarithm*, Mathematical Notes, 55(2) (1994) 155-172. Translated from Matematicheskie Zametki, 55(2) (1994) 91-101. This result dates from 1968.

[20] J.M. Pollard, *Theorems on factorization and primality testing*, Proceedings of the Cambridge philosophical society, **76** (1974) 521-528.

[21] J.M. Pollard, *Monte Carlo methods for index computation (mod p)*, Math. Comp. **32** (1978) 918-924.

[22] C. Pomerance, *Fast, rigorous factorization and discrete logarithm algorithms*, in: D.S. Johnson, T. Hishizeki, A. Nozaki and H.S. Wilf, eds., *Discrete algorithms and complexity* (Academic press, Orlando, FL, 1987) 119-143.

[23] C. Pomerance, J.W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experimental Math. **1** (1992) 89-94.

[24] M.O. Rabin, *Probabilistic algorithms for primality testing*, J. Number Theory **12** (1980) 128-138.

[25] R.L. Rivest, letter to M. Gardner containing an estimate of the difficulty of factoring a 129-digit modulus using Pollard's rho method, date?

[26] R.L. Rivest, R.D. Silverman, *Are 'strong' primes needed for RSA?*, manuscript, April 1997, available from www.iacr.org.

[27] V. Shoup, *Lower bounds for discrete logarithms and related problems*, Proceedings Eurocrypt'97, LNCS 1233, Springer-Verlag 1997, 256-266.

[28] R.D. Silverman,*The multiple polynomial quadratic sieve*, Math. Comp. **46** (1987) 327-339.

[29] G. Villard, *Further analysis of Coppersmith's block Wiedemann algorithm for the solution of sparse linear systems (extended abstract)*, Proceedings 1997 International Symposium on Symbolic and Algebraic Computation, ACM Press (1997) 32–39.

[30] D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Transactions on Information Theory **IT-32** (1986) 54–62.

[31] H.C. Williams, *A $p+1$ method of factoring*, Math. Comp. **39** (1982) 225-234.

L function

For $t, \gamma \in \mathbf{R}$ with $0 \leq t \leq 1$ the notation $L_x[t, \gamma]$ is used for any function of x that equals

$$e^{(\gamma+o(1))(\log x)^t(\log\log x)^{1-t}}, \text{ for } x \to \infty,$$

where logarithms are natural. This function has the following properties:

- $L_x[t, \gamma] + L_x[t, \delta] = L_x[t, \max(\gamma, \delta)]$,
- $L_x[t, \gamma]L_x[t, \delta] = L_x[t, \gamma + \delta]$,
- $L_x[t, \gamma]L_x[s, \delta] = L_x[t, \gamma]$ if $t > s$,
- for any fixed k:
  - $L_x[t, \gamma]^k = L_x[t, k\gamma]$,
  - if $\gamma > 0$ then $(\log x)^k L_x[t, \gamma] = L_x[t, \gamma]$.
- $\pi(L_x[t, \gamma]) = L_x[t, \gamma]$ where $\pi(y)$ is the number of primes $\leq y$.

When used to indicate runtimes and for γ fixed, $L_x[t, \gamma]$ for t ranging from 0 to 1 ranges from polynomial-time to exponential-time in $\log(x)$:

- runtime
  $$L_x[0, \gamma] = e^{(\gamma+o(1))\log\log x} = (\log x)^{\gamma+o(1)}$$
  is polynomial in $\log(x)$,
- runtimes $L_x[t, \gamma]$ with $0 < t < 1$ are examples of runtimes that are subexponential in $\log(x)$, i.e., asymptotically greater than polynomial and less than exponential,
- runtime
  $$L_x[1, \gamma] = e^{(\gamma+o(1))\log x} = x^{\gamma+o(1)}$$
  is exponential in $\log(x)$.

Pollard's $p-1$ method
See **Integer factoring**.

Pollard's rho method
See **Integer factoring**.

Random squares method
See **Dixon's random squares method** in **Integer factoring**.

Smoothness
A integer is B-smooth if all its prime factors are at most B.

Smoothness probability
Let $\alpha, \beta, r, s \in \mathbf{R}_{>0}$ with $s < r \leq 1$. With L_x as in **L function**, it follows from [1, 2] that a random positive integer $\leq L_x[r, \alpha]$ is $L_x[s, \beta]$-smooth with probability

$$L_x[r-s, -\alpha(r-s)/\beta], \text{ for } x \to \infty.$$

References

[1] E.R. Canfield, P. Erdös, C. Pomerance, *On a problem of Oppenheim concerning "Factorisatio Numerorum"*, J. Number Theory **17** (1983) 1-28.

[2] N.G. De Bruijn, *On the number of positive integers $\leq x$ and free of prime factors $> y$, II*, Indag. Math. **38** (1966) 239-247.