1
History of integer factorization

Samuel S. Wagstaff, Jr.

The integer factorization problem is well known as the mathematical foundation of the public-key cryptosystem RSA. When RSA was introduced forty years ago, the largest number one could factor was still tiny. Spurred on by the practical deployment of RSA, more and more factoring algorithms were developed and fine-tuned, from the Elliptic Curve Method to the Number Field Sieve. This chapter treats algorithmic developments in factoring integers and their effect on cryptography, not just on RSA. For a long time, factoring records followed a regular pattern. In this chapter, an overview of factoring progress through the years is given, with a bold prediction of what might come to be.

This chapter covers some of the material of the author's book [68], in abbreviated form. See that work for a fuller treatment, proofs and more examples.

1.1 The Dark Ages: Before RSA

Many years ago, people published tables of primes and of prime factors of integers. The first such table was created by Eratosthenes more than 2500 years ago. Cataldi published a table of factors of numbers up to 800 in 1603. In 1811, Chernac went up to 1,020,000. D. N. Lehmer [31] published the last factor table (to 10,017,000) more than a century ago. He also published a table of all primes up to 10,006,721. No one will publish any more tables of factors or primes up to some limit because one can compute them in seconds using a sieve.

1.1.1 Trial Division

The basic Trial Division Algorithm is old and slow. For hundreds of years, factorers who used Trial Division found ways to omit trial divisors that could not possibly divide the candidate number. The algorithm need not try composite numbers as divisors. A simple way to skip some composites is to alternate adding 2 and 4 to a trial divisor to form the next one. This trick skips multiples of 2 and 3. The method, called a *wheel*, can be extended to skip multiples of 2, 3 and 5 by adding the differences between consecutive residue classes relatively prime to 30.

Another approach is to compute a table of primes first and divide the candidate number only by these numbers. A *sieve* can be used to build a table of primes between two limits quickly. See Section 1.4.1 for sieves.

Quadratic residues can be used to speed Trial Division by skipping some *primes* that cannot be divisors. Euler, Gauss and others used this trick hundreds of years ago. Let us try to factor n and assume we know a non-square quadratic residue r modulo n. Then r must also be quadratic residue modulo any prime factor p of n. If r is not a square, the Law of Quadratic Reciprocity restricts p to only one-half of the possible residue classes modulo $4|r|$. Where does one find small non-square quadratic residues for use in the technique described above? One way to construct them is to compute x^2 mod n where x is slightly larger than an integer multiple of $\sqrt{n}$. This idea led to the Quadratic Sieve Algorithm. Another source of small quadratic residues is the continued fraction expansion of $\sqrt{n}$, as we shall see in Section 1.3.

1.1.2 Fermat's Difference of Squares Method

Fermat's Difference of Squares Method may be used to factor an odd number n by expressing n as a difference of two squares, $x^2 - y^2$, with the pair x, y different from $(n + 1)/2$, $(n - 1)/2$ (which gives $x + y = n$ and $x - y = 1$). Any other representation of n as $x^2 - y^2$ produces a nontrivial factorization $n = (x - y)(x + y)$. Here we have $x \geq \sqrt{n}$.

Fermat's Difference of Squares Factoring Algorithm

```
Input: An odd composite positive integer $n$.
$x := \lfloor \sqrt{n} \rfloor$
$t := 2x + 1$
$r := x^2 - n$
while ($r$ is not a square) {
    $r := r + t$
    $t := t + 2$
    }
$x := (t - 1)/2$
$y := \sqrt{r}$
Output: $x - y$ and $x + y$ are the factors of $n$.
```

When the **while** loop terminates we have $r = x^2 - n$, where $x = (t-1)/2$, and also r is a square: $r = y^2$. Then $n = x^2 - y^2$ and n is factored.

In most cases this algorithm will be much slower than Trial Division. However, the algorithm works quite well when n has a divisor within $O(\sqrt[4]{n})$ of $\sqrt{n}$. It is the reason why one must not choose the two primes for RSA too close together.

1.1.3 Pollard's Methods

In the 1970s, Pollard invented two factoring algorithms: the Rho Method and the $p-1$ method. Both methods are better than Trial Division at finding small factors of a large number.

Let n be composite and let p be an unknown prime factor of n, which we hope to find. Pollard [47] suggested choosing a random function f from the set $\{0, 1, \ldots, n-1\}$ into itself, picking a random starting number s in the set and then iterating f:

$$s, f(s), f(f(s)), f(f(f(s))), \ldots.$$

If we reduce these numbers modulo the unknown prime p, we get a sequence of integers in the smaller set $\{0, 1, \ldots, p-1\}$. Because of the birthday paradox in probability, a number in the smaller set will be repeated after about $\sqrt{p}$ iterations of f. If u, v are iterates of f with $u \equiv v \pmod{p}$, then it is likely that $\gcd(u-v, n) = p$ because p divides $u-v$ and n, and probably no other prime factor of n divides $u-v$. Here is a simple version of the algorithm. The main loop iterates the function in two ways, with one iteration per step for A and two iterations per step for B. Then it computes $\gcd(A-B, n)$ and stops when this gcd first exceeds 1. The Pollard Rho Method is a Monte Carlo probabilistic algorithm.

Pollard Rho Factorization Method

```
Input: A composite number $n$ to factor.
Choose a random $b$ in $1 \le b \le n-3$
Choose a random $s$ in $0 \le s \le n-1$
$A := s$ ; $B := s$
Define a function $f(x) := (x^2 + b) \bmod n$
$g := 1$
while ($g = 1$) {
    $A := f(A)$
    $B := f(f(B))$
    $g := \gcd(A-B, n)$
    }
if ($g < n$) { write $g$ "is a proper factor of" $n$ }
else { either give up or try again with new $s$ and/or $b$ }
```

```
Output: A proper factor $g$ of $n$, or else "give up."
```

The Pollard Rho Method takes about $O(\sqrt{p})$ steps to discover the prime factor p of n. It is the reason why the primes for an RSA public modulus must have at least 30 decimal digits.

In 1974, Pollard [46] proposed the $p - 1$ Method. Fermat's Theorem says that $a^{p-1} \equiv 1 \pmod{p}$ when p is a prime not dividing a. Pollard's $p-1$ method chooses a large integer L with many divisors of the form $p - 1$ to try many potential prime factors p of n at once. Since we can't compute a^L mod p because p is an unknown factor of n, we compute a^L mod n.

If the largest prime factor of $p - 1$ is $\leq B$, then $p - 1$ will divide L when L is the product of all primes $\leq B$, each repeated an appropriate number of times. One simple choice for L is $B!$. Usually, B is large and L is enormous. We need not compute L. At step i, we compute $a := a^i$ mod n. Here is a basic form of the algorithm.

Simple Pollard $p - 1$ Factorization Method

```
Input: A composite positive integer $n$ to factor and a bound $B$.
$a := 2$
for ($i := 1$ to $B$) {
    $a := a^i \bmod n$
    }
$g := \gcd(a - 1, n)$
if ($1 < g < n$) { write "$g$ divides $n$" }
else { give up }
Output: A proper factor $g$ of $n$, or else give up.
```

It is best to compute the GCD operation once every few thousand iterations of the **for** loop rather than just once at the end. When this is done, the **for** loop continues in case $g = 1$. If $g = 1$ at the end, one can either give up or try a second stage.

Baillie found the 16-digit prime divisor $p = 1256132134125569$ of the Fermat number $F_{12} = 2^{4096} + 1$ using Pollard $p - 1$ with $B = 30,000,000$. He succeeded since the largest prime factor of $p - 1$ is less than B:

$$p - 1 = 2^{14} \cdot 7^2 \cdot 53 \cdot 29521841.$$

The algorithm has a second stage which chooses a second bound $B_2 > B$ and looks for a factor p of n for which the largest prime factor of $p - 1$ is $\leq B_2$ and the second largest prime factor of $p - 1$ is $\leq B$.

The Pollard $p - 1$ algorithm shows that if p and q are the factors of an RSA public key, then each of $p - 1$ and $q - 1$ must have a prime factor of at least 20 decimal digits.

1.1.4 Primality Testing

Suppose we desire the prime factorization of a positive integer n. The first step is to learn whether n is prime. If it is prime, no factoring is needed. If n is composite and factorization finds that $n = ab$ with $1 < a \leq b < n$, then we must determine whether a and b are prime to decide whether we are done. Some methods of proving that an integer is prime, like Theorem 1.1 below, require factoring a different integer.

Primality testing was as hard as factoring one hundred years ago. Mathematicians would try for a while to factor a large number; if they could not factor it, they might conjecture it is prime. Later, some other mathematician might factor it or prove it is prime.

About a century ago, mathematicians discovered fast tests that reported whether an integer is definitely composite or "probably prime." At about the same time, theorems were proved that gave a rigorous proof that a probable prime number is prime, but some factoring might be needed to complete the proof. These tests were refined until thirty years ago people found a very fast test for primeness that has never been proved correct, but that has never failed either. The theorems also were improved until twenty years ago one could prove with modest effort that virtually every 1000-digit probable prime really is prime. Some of these primality tests are *probabilistic*, meaning that: the algorithm chooses random numbers as it runs, most choices for these numbers will lead to a short running time, some random choices could make the program run for a long time, and the program always gives a correct answer when it finishes. In 2002, a deterministic general polynomial-time algorithm for primality testing was found. Although it runs in deterministic polynomial time and always gives the correct answer, this test is still slower for large numbers than some probabilistic primality tests. The remainder of this section gives highlights of the development of primality testing during the past century. See Pomerance [51] for a common theme of some of the prime proving methods below.

Theorem 1.1 (Lehmer) *Let m be an odd positive integer. Then m is prime if and only if there exists an integer a such that $a^{m-1} \equiv 1 \pmod{m}$, but for every prime q that divides $m - 1$, $a^{(m-1)/q} \not\equiv 1 \pmod{m}$.*

In order to use Theorem 1.1 to prove that m is prime, we must factor $m - 1$ completely. If we can't factor $m - 1$ but are willing to allow a tiny chance of error, there are fast algorithms to decide whether m is prime or composite.

Call an integer $m > 1$ a *probable prime* to base a if $a^{m-1} \equiv 1 \pmod{m}$. By Fermat's Theorem, every odd prime m is a probable prime. A composite probable prime to base a is called a *pseudoprime* to base a. For integers $a > 1$, most probable primes to base a are prime and very few of them are pseudoprimes.

A better test on large numbers is to use a *strong* probable prime test. A *strong probable prime to base a* is an odd positive integer m with this property: If we write $m-1 = 2^e f$ with f odd, then either $a^f \equiv 1 \pmod{m}$ or $a^{f\cdot 2^c} \equiv -1 \pmod{m}$ for some c in $0 \leq c < e$. A *strong pseudoprime* is a composite strong probable prime. Every prime is a strong probable prime to every base it does not divide. Also, every strong probable prime m is a probable prime to the same base. There are infinitely many strong pseudoprimes to every base.

It is known that if m is prime and $\equiv$ 3 or 7 mod 10, then m divides the Fibonacci number u_{m+1}. Baillie, Pomerance, Selfridge and the author [4], [52] conjectured that no composite integer m, whose last decimal digit is 3 or 7, is a strong probable prime to base 2 and also divides the Fibonacci number u_{m+1}. (They made a similar, but slightly more complicated, conjecture for numbers with last digit 1 or 9. See [4] and [52] for details.) No one has ever proved or disproved this conjecture. It has been used millions of times to construct large primes with not a single failure. The authors of [52] offer US$620 to the first person who either proves that this test is always correct or exhibits a composite number that the test says is probably prime. Heuristic arguments suggest that there are composite numbers that the test says are probably prime but that the least such examples have hundreds or even thousands of decimal digits. This probable prime test takes $O((\log m)^3)$ steps to test m, that is, polynomial time. This primality test has become the (ANSI) standard method of selecting primes for an RSA public key. It is used in the Secure Sockets Layer in computer networks.

In 2002, Agrawal, Kayal and Saxena [1] found the first general deterministic polynomial-time primality algorithm. Their original version had running time $O((\log m)^{12})$, but others [44], [22] soon reduced this to about $O((\log m)^6)$, better, but still much slower than the probabilistic test described above. See Agrawal, Kayal and Saxena [1] and articles that refer to it for details.

1.2 The Enlightenment: RSA

People have concealed messages through cryptography for more than 2000 years. Until recently, the sender and receiver had to meet before the secret communication and decide how to hide their future secret messages. They would choose an algorithm, fixed for a long time, and a secret key, changed often.

Some businesses need secret codes to communicate securely with their offices. Some people want cryptography for their personal secrets. In 1975, the

National Bureau of Standards announced a cipher, the Digital Encryption Standard, DES, which they approved for use by individuals and businesses.

As computer networks were being built in the early 1970s, people began to think about electronic communication like email. Computer scientists began to ponder how one could "sign" a digital document so that the recipient could be certain who wrote it.

Others thought about how to create a system in which people who had never met could communicate securely. Symmetric-key ciphers like DES require users to exchange keys securely by meeting before their secret communication happens. All ciphers known then were of this type.

Whit Diffie and Marty Hellman pondered these matters and found some brilliant solutions in a 1976 paper [17].

First, they defined one-way functions, easy to compute forwards, but nearly impossible to invert. They invented a method of choosing a secret key based on the one-way function $f(x) = b^x \bmod p$. Given large numbers b, p and y, with prime p, it is almost impossible to find an x with $f(x) = y$. Their method allows two users who have never met to choose a common secret key, for DES, say, while eavesdroppers listen to their communication.

This key-exchange method provides secure communication between two users connected on the internet. However, it is subject to the man-in-the-middle attack in which a wiretapper hijacks a computer between the two parties trying to communicate and executes the protocol with each of them separately. After that action, the hijacker can read or change encrypted messages between the two parties as she decrypts a message from one and re-enciphers it to pass on to the other. Diffie and Hellman [17] suggested a second innovation to solve this problem. They split the key into two parts, one public and one private. Bob would reveal his public enciphering key and the enciphering algorithm. But he would tell no one his private deciphering key. It would be impossible to find the deciphering key from the enciphering key. This cipher is called a *public key cipher*. To send Bob a secret message, Alice would fetch his public key and use it to encipher her message. Once it was enciphered, only Bob could decipher it because only Bob knows his private key. If Alice wanted to communicate with Bob by DES, say, she could choose a random DES key, encipher the key with Bob's public key and send it to Bob. When Bob got this message, he would decipher it and then use the DES key to communicate with Alice. No eavesdropper could learn the DES key.

However, a public-key cipher lacks authenticity. Anyone could write a message to Bob, sign it with Alice's name, encipher it using Bob's public key, and send it to Bob. Bob would not know whether it actually came from Alice. This way Eve could send a random DES key to Bob, saying that it came from Alice.

Then she could communicate with Bob privately, pretending to be Alice. The third innovation of Diffie and Hellman in [17]. was the notion of a digital signature. Alice could create her own public and private keys, just as Bob did above. She would "sign" a message by applying the deciphering algorithm with her secret key to the plaintext message. Then she could encipher this signed message with Bob's public key and send it to him. Bob would be certain that the message came from Alice when he deciphered it using his private key, enciphered the result with Alice's public key and obtained a meaningful text. This signature verification shows that the message came from Alice because only someone who knows her private key could have constructed a message with these properties.

Diffie and Hellman did not suggest any enciphering and deciphering algorithms for public-key cryptography in their article. Later, Rivest, Shamir and Adleman (RSA) read their paper and found an algorithm [55] for doing this based on the presumed difficulty of factoring large integers. The RSA cipher gives a simple formula for enciphering a message. The public key is the product of two primes large enough so that their product could not be factored. Their formula for deciphering the ciphertext uses the two large primes. They created a *trap-door one-way function*, that is, a function f that cannot be inverted unless one knows a secret, in which case it is easy to find x given $f(x)$. The secret that unlocks the RSA cipher is the prime factors of the public key. Since an obvious attack on this cipher is to factor the public key, the publication of their paper created tremendous interest in the problem of factoring large integers.

Here is the simplest form of the RSA cipher. In order to use RSA to receive secret messages, Alice chooses two large secret primes p and q and computes $n = pq$. Alice also chooses an exponent e relatively prime to $\phi(n) = (p-1)(q-1)$ and computes a number d so that $ed \equiv 1 \pmod{\phi(n)}$. Alice publishes n and e as her public key and keeps p, q and d secret. Plaintext and ciphertext are positive integers less than n. When Bob wishes to send a secret message M to Alice he finds her public key. He enciphers M as $C = M^e \bmod n$ and sends C to Alice. Alice deciphers the message by computing $M = C^d \bmod n$. The original message M is recovered because of Euler's Theorem, which says that $M^{\phi(n)+1} \equiv M \pmod{n}$ when $\gcd(M, n) = 1$. Only Alice can decipher C because only she knows d. Someone who could factor n to discover p and q could compute d the same way Alice computed it.

Martin Gardner [21] wrote an article describing the papers of Diffie, Hellman, Rivest, Shamir and Adleman. In it, RSA offered a challenge message encoded with a 129-digit product of two secret primes. This article was shocking because it revealed to the general public a powerful cipher that even the government couldn't break. The challenge message in this article was deciphered in 1993–1994 by Derek Atkins, Michael Graff, Arjen Lenstra and Paul Leyland [3] who factored the 129-digit number.

It is clear that if you can factor the RSA modulus n, then you can decipher any ciphertext. But it is less clear, and no one has ever proved, that if you can decipher arbitrary ciphertext in RSA, then you can factor n. Two ciphers were invented that do enjoy this equivalence. Rabin and Williams each devised public key ciphers with the property that one can prove that breaking the cipher is equivalent to factoring a large integer n. In Rabin's [54] system, Alice chooses two large primes p and q with $p \equiv q \equiv 3 \pmod 4$. She publishes the product $n = pq$ and keeps p and q secret. When Bob wants to send a message M in $0 < M < n$ to Alice, he enciphers it as $C = M^2 \bmod n$. When Alice receives C, which is a quadratic residue modulo n, she uses her knowledge of p and q to find all four square roots of C modulo n. If M is ordinary plaintext, then only one square root will make sense and M is that one. If M is a binary string or otherwise indistinguishable from the other three square roots of $M^2 \bmod n$, then Bob must indicate which square root is M.

Rabin [54] proved that if one has an algorithm to decipher any message M enciphered with his cipher in a reasonable time, then there is an algorithm to factor the modulus n in a reasonable time.

Williams [71] constructed a similar cipher using primes $p \equiv 3 \pmod 8$ and $q \equiv 7 \pmod 8$, and $n = pq$. His cipher eliminates the ambiguity in deciphering and also has the property that breaking the cipher is equivalent to factoring n.

1.3 The Renaissance: Continued Fractions

Although the factoring method in this section is slow compared to the fastest known ones, its ideas led to them. The Continued Fraction Algorithm, CFRAC, was the first factoring algorithm with subexponential time complexity.

1.3.1 Basic facts about continued fractions

A *simple continued fraction* is an expression of the form

$$x = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots}}} \,. \tag{1.1}$$

The q_i are required to be integers for all i, and also positive when $i > 0$. A simple continued fraction may be finite,

$$x_k = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots + \cfrac{1}{q_k}}}}\,, \tag{1.2}$$

which we write as $[q_0; q_1, q_2, q_3, \ldots, q_k]$.

If an infinite continued fraction (1.1) is truncated at q_k, then it has the value (1.2), clearly a rational number.

Given a finite continued fraction (1.2) we can find its value $x_k = A_k/B_k$ as a rational number in lowest terms working backwards clearing the denominators starting from $q_{k-1}+1/q_k$. Here is a way of finding this rational number working forwards.

Theorem 1.2 *The rational number $A_k/B_k = [q_0; q_1, q_2, \ldots q_k]$ is determined from q_0, …, q_k by $A_{-1} = 1$, $B_{-1} = 0$, $A_0 = q_0$, $B_0 = 1$ and*

$$\begin{aligned} A_i &= q_i A_{i-1} + A_{i-2} \\ B_i &= q_i B_{i-1} + B_{i-2} \quad \text{for } i = 1, 2, \ldots, k. \end{aligned} \tag{1.3}$$

A continued fraction is finite if and only if it represents a rational number. If n is a positive integer and not a square, then $x = \sqrt{n}$ is irrational and its continued fraction is infinite but periodic.

Theorem 1.3 (Galois) *Let n be a positive integer and not a square. Then the continued fraction for $\sqrt{n}$ has the form*

$$\sqrt{n} = [q_0; \overline{q_1, q_2 \ldots, q_{p-1}, 2q_0}\,],$$

where the overbar marks the period.

Example 1.4

$$\begin{aligned} \sqrt{44} &= [6; \overline{1, 1, 1, 2, 1, 1, 1, 12}] \\ \sqrt{77} &= [8; \overline{1, 3, 2, 3, 1, 16}] \\ \sqrt{85} &= [9; \overline{4, 1, 1, 4, 18}]. \end{aligned}$$

When factoring a large integer n by CFRAC the algorithm will compute only a small portion of the continued fraction for $\sqrt{n}$. There is a simple iteration that computes the q_i in the continued fraction for $\sqrt{n}$ using only integer arithmetic.

Several other integers are computed during the iteration. One of them is an integer Q_i which satisfies $A_{i-1}^2 - nB_{i-1}^2 = (-1)^i Q_i$ and $0 < Q_i < 2\sqrt{n}$. If we consider the equation as a congruence modulo n, we have $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{n}$. In other words, the continued fraction iteration produces a sequence $\{(-1)^i Q_i\}$ of quadratic residues modulo n whose absolute values are $< 2\sqrt{n}$, very small indeed. We want small quadratic residues because they are easier to compute, easier to factor, and more likely to be smooth. (An integer is *B-smooth* if all of its prime factors are smaller than the real number B.)

One can prove that $x_i = (P_i + \sqrt{n})/Q_i$ for $i \geq 0$, where

$$P_i = \begin{cases} 0 & \text{if } i = 0\,, \\ q_0 & \text{if } i = 1\,, \\ q_{i-1}Q_{i-1} - P_{i-1} & \text{if } i \geq 2\,, \end{cases} \tag{1.4}$$

and

$$Q_i = \begin{cases} 1 & \text{if } i = 0\,, \\ n - q_0^2 & \text{if } i = 1\,, \\ Q_{i-2} + (P_{i-1} - P_i)q_{i-1} & \text{if } i \geq 2\,. \end{cases} \tag{1.5}$$

The q_i can be computed using

$$q_i = \lfloor x_i \rfloor = \begin{cases} \left\lfloor \sqrt{n} \right\rfloor & \text{if } i = 0\,, \\ \left\lfloor \dfrac{q_0 + P_i}{Q_i} \right\rfloor = \left\lfloor \dfrac{\sqrt{n} + P_i}{Q_i} \right\rfloor & \text{if } i > 0\,. \end{cases} \tag{1.6}$$

Define A_i and B_i as in Theorem 1.2 so that $[q_0; q_1, q_2, \ldots q_i] = A_i/B_i$ for $i \geq 0$.

Example 1.5 Table 1.1 gives the sequences for the continued fraction of $\sqrt{77} = 8.774964387392123$.

1.3.2 A General Plan for Factoring

The following theorem is ancient.

Theorem 1.6 *If n is a composite positive integer, x and y are integers, and $x^2 \equiv y^2 \pmod{n}$, but $x \not\equiv \pm y \pmod{n}$, then $\gcd(x - y, n)$ and $\gcd(x + y, n)$ are proper factors of n.*

Later we will tell how to find x and y with $x^2 \equiv y^2 \pmod{n}$. However, it is difficult to ensure that $x \not\equiv \pm y \pmod{n}$, so we ignore this condition. The next theorem tells how several modern factoring algorithms finish.

Table 1.1 *Continued Fraction Expansion For* $\sqrt{77}$

i	q_i	P_i	Q_i	A_i	B_i
−1	–	–	–	1	0
0	8	0	1	8	1
1	1	8	13	9	1
2	3	5	4	35	4
3	2	7	7	79	9
4	3	7	4	272	31
5	1	5	13	351	40
6	16	8	1	5888	671
7	1	8	13	6239	711
8	3	5	4	24605	2804
9	2	7	7	55449	6319
10	3	7	4	190952	21761
11	1	5	13	246401	28080
12	16	8	1	4133368	471041

Theorem 1.7 *If n is an odd positive integer having at least two different prime factors, and if integers x and y are chosen randomly subject to* $x^2 \equiv y^2 \pmod{n}$*, then, with probability* ≥ 0.5*,* $\gcd(x - y, n)$ *is a proper factor of n.*

One can compute in probabilistic polynomial time a square root of any quadratic residue r modulo n, provided the factors of n are known. In fact, computing square roots modulo n is polynomial-time equivalent to factoring n.

Corollary *Let n have at least two different odd prime factors. If there is a (probabilistic) polynomial time algorithm* $\mathcal{A}$ *to find a solution x to* $x^2 \equiv r \pmod{n}$ *for any quadratic residue r modulo n, then there is a probabilistic polynomial time algorithm* $\mathcal{B}$ *to find a factor of n.*

The general plan of several factoring algorithms is to generate (some) pairs of integers x, y with $x^2 \equiv y^2 \pmod{n}$, and hope that $\gcd(x - y, n)$ is a proper factor of n. Theorem 1.7 says that we will not be disappointed often. It says that each such pair gives at least a 50% chance to factor n. If n has more than two (different) prime factors, then at least one of the greatest common divisors will be composite and we will have more factoring to do. In the fastest modern factoring algorithms it may take a long time to produce the first pair x, y, but after it is found many more random pairs are produced quickly, and these will likely yield all prime factors of n.

1.3.3 The Continued Fraction Factoring Algorithm

The Continued Fraction Factoring Algorithm, CFRAC, of Morrison and Brillhart [45], uses the fact that the Q_i are more likely to be smooth than numbers near $n/2$ because they are small. The algorithm uses the continued fraction expansion for $\sqrt{n}$ to generate the sequences $\{P_i\}$, $\{Q_i\}$, $\{q_i\}$ and $\{A_i \bmod n\}$ via Equations (1.4), (1.5), (1.6) and (1.3), and tries to factor each Q_i by Trial Division. Morrison and Brillhart restricted the primes in the Trial Division to those below some fixed bound B, called the *factor base*. CFRAC saves the B-smooth Q_i, together with the corresponding A_{i-1}, representing the relation $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{n}$. When enough relations have been collected, Gaussian elimination is used to find linear dependencies (modulo 2) among the exponent vectors of the relations. We have enough relations when there are more of them than primes in the factor base. Each linear dependency produces a congruence $x^2 \equiv y^2 \pmod{n}$ and a chance to factor n by Theorem 1.7.

Assuming two plausible hypotheses, Pomerance [49] proved that the time complexity of CFRAC is $L(n)^{\sqrt{2}}$, where $L(x) = \exp\left(\sqrt{(\ln x)\ln\ln x}\right)$.

Let me say more about the linear algebra step. Suppose there are K primes in the factor base. Call them $p_1, p_2, \ldots, p_K$. (These are the primes $p \le B$ for which n is a quadratic residue modulo p. They comprise about half of the primes $\le B$.) The goal is to find a set S of i for which the product $\prod_{i\in S}(-1)^i Q_i$ is the square of an integer. Since a square must be positive, the "prime" $p_0 = -1$ is added to the factor base. For each i for which $(-1)^i Q_i$ is B-smooth, write $(-1)^i Q_i = \prod_{j=0}^{K} p_j^{e_{ij}}$. When $(-1)^i Q_i$ is B-smooth, define the vector $\vec{v}_i = (e_{i0}, e_{i1}, \ldots, e_{iK})$. Note that when $(-1)^i Q_i$ and $(-1)^k Q_k$ are multiplied, the corresponding vectors $\vec{v}_i$, $\vec{v}_j$ are added. A product like $\prod_{i\in S}(-1)^i Q_i$ is a square if and only if all entries in the vector sum $\sum_{i\in S} \vec{v}_i$ are even numbers. Form a matrix with $K + 1$ columns whose rows are the vectors $\vec{v}_i$ (reduced modulo 2) for which $(-1)^i Q_i$ is B-smooth. If there are more rows than columns in this matrix, Gaussian elimination will find nontrivial dependencies among the rows modulo 2. Let S be the set of i for which the row $\vec{v}_i$ is in the dependency. Each nontrivial dependency, say, $\sum_{i\in S} \vec{v}_i = \vec{0}$, gives a product $\prod_{i\in S}(-1)^i Q_i$ which is a square, say, y^2. Let $x = \prod_{i\in S} A_{i-1} \bmod n$. Then $x^2 \equiv y^2 \pmod{n}$, an instance of Theorem 1.7.

The idea of combining several relations $x_i^2 \equiv r_i \bmod n$ to form a congruence $x^2 \equiv y^2 \pmod{n}$ and get a chance to factor n goes back at least to Kraitchik [29] and Lehmer and Powers [30] around 1930.

Here are the most important new ideas in [45]:

- the fixed factor base,
- using linear algebra modulo 2 to combine relations to form a square, and
- using large primes to augment the factor base.

Here is how the third idea works. When you try to factor Q_i using the primes in the factor base, you often fail because a cofactor > 1 remains. It is easy to test the cofactor for primality. If it is prime, save the relation. If another relation with the same large prime is found, then one can create a useful relation by multiplying them. Suppose $A_{i-1}^2 \equiv (-1)^i Q_i \pmod{n}$ and $A_{j-1}^2 \equiv (-1)^j Q_j \pmod{n}$. Their product, $(A_{i-1}A_{j-1})^2 \equiv (-1)^{i+j} Q_i Q_j \pmod{n}$, will have one prime not in the factor base appearing on the right side, but this prime will be squared, so it will not appear in the matrix, and the product relation can be used to find a congruence $x^2 \equiv y^2 \pmod{n}$. If the number of possible large primes is t, then repeated large primes will begin to appear roughly when the number of relations with large primes reaches $1.18\sqrt{t}$ by the birthday paradox. Using large primes this way does not change the theoretical time complexity of CFRAC, but it does have practical value. If B and the upper limit on the size of large primes are chosen well, then most of the relations will come from pairing repeated large primes.

In the 1980s, Smith and Wagstaff [53], [64] and [67] fabricated an Extended-Precision Operand Computer for factoring large integers by the CFRAC. It had a 128-bit wide main processor to generate the A_i and Q_i, and sixteen remaindering units to divide a Q_i by sixteen different primes p_j in parallel, finding only the remainders. As each Q_i was generated, it was loaded into a wide shift register. Sixteen trial divisors p_j were loaded into registers of simple arithmetic-logic units. While the main processor generated the next Q_i and A_i the current Q_i was shifted out of its register, one bit at a time, and broadcast to the remaindering units, which reported when a remainder was 0. The main processor used the reports to determine whether Q_i was smooth enough for the relation to be saved. A personal computer connected to the main processor stored the smooth relations. The linear algebra was done on a larger computer.

1.4 The Reformation: A Quadratic Sieve

This section adds a new idea to CFRAC: Factor the right hand sides of many relations together with an efficient algorithm called a *sieve* rather than separately with slow Trial Division.

1.4.1 The Sieve of Eratosthenes

The Sieve of Eratosthenes finds the primes below some limit J. It writes the numbers $1, 2, \ldots, J$. Then it crosses out the number 1, which is not prime. After that, let p be the first number not crossed out. Cross out all multiples of

p greater than p. Repeat these steps, replacing p by the next number not yet crossed out, as long as $p \leq \sqrt{J}$. When p reaches $\sqrt{J}$ all numbers crossed out are composite (or 1) and all numbers not crossed out are prime.

A computer program would use an array $P[]$ to represent the numbers 1 to J. Let "1" mean that the number is not crossed out and "0" mean that the number is crossed out. The algorithm starts by marking 1 as "crossed out" and the other numbers as "not crossed out." For each prime p, cross out all multiples of the prime p. Then the first number not yet crossed out is the next prime p. At the end, the value of $P[i]$ is 1 if i is prime and 0 if i is 1 or composite. The Sieve of Eratosthenes computes all primes $\leq J$ in $O(J \log\log J)$ steps.

A variation of this sieve factors the numbers in the range of a polynomial $f(x)$ with integer coefficients, but it only finds the prime factors of each $f(x)$ which lie in a finite set $\mathcal{P}$ of primes. The polynomial $f(x)$ is fixed. This sieve algorithm is the heart of the Quadratic and Number Field Sieve factoring algorithms. For each i, a linked list $L[i]$ holds the distinct prime factors of $f(i)$.

Sieve to Factor the Range of a Polynomial

```
Input: Integers $J > I > 1$ and a finite set $\mathcal{P}$ of primes.
**for** ($i := I$ to $J$) { $L[i] :=$ **empty** }
**for each** $p \in \mathcal{P}$ {
    Find the roots $r_1, \ldots, r_d$ of $f(x) \equiv 0 \pmod{p}$
    **for** ($j := 1$ to $d$) {
        $i :=$ the least integer $\geq I$ and $\equiv r_j \pmod{p}$
        **while** ($i \leq J$) { append $p$ to $L[i]$; $i := i + p$ }
        }
    }
Output: For $I \leq i \leq J$, $L[i]$ lists the factors in $\mathcal{P}$ of $f(i)$.
```

The output $L[i]$ lists only the distinct prime factors of $f(i)$ in $\mathcal{P}$. A modification of this algorithm also finds the number of repetitions of each prime factor of $f(i)$ without ever forming that number, which may be huge.

1.4.2 The Quadratic Sieve Factoring Algorithm

The Quadratic Sieve Factoring Algorithm, QS, and the CFRAC differ only in the method of producing relations $x^2 \equiv q \pmod{n}$ with q factored completely. The CFRAC forms x and q from the continued fraction expansion of $\sqrt{n}$ and factors q by slow Trial Division. The QS produces x and q using a quadratic polynomial $q = f(x)$ and factors the q with a sieve, much faster than Trial Division. The quadratic polynomial $f(x)$ is chosen so that the q will be as small as possible. Most q of them will exceed $2\sqrt{n}$, but not by a large factor, so that they are almost as likely to be smooth as the q in CFRAC.

Let $f(x) = x^2 - n$ and $s = \lceil \sqrt{n} \rceil$. The QS factors some of the numbers

$$f(s), f(s+1), f(s+2), \ldots$$

by the algorithm in Section 1.4.1. If there are K primes in the factor base and we find $R > K$ B-smooth numbers $f(x)$, then there will be R relations involving K primes and linear algebra will produce at least $R - K$ congruences $x^2 \equiv y^2 \pmod{n}$, each of which has probability at least $1/2$ of factoring n, by Theorem 1.7.

Sieve using the fast algorithm in Section 1.4.1 to find the B-smooth numbers among $f(s), f(s+1), f(s+2), \ldots$. The factor base $\mathcal{P}$ consists of the primes $p < B$ for which the Legendre symbol $(n/p) \neq -1$. Write the numbers $f(s+i)$ for i in some interval $a \leq i < b$ of convenient length. The first interval will have $a = s$. Subsequent intervals will begin with a equal to the endpoint b of the previous interval. For each prime $p < B$, remove all factors of p from those $f(s+i)$ which p divides. Since $f(x) = x^2 - n$, p divides $f(x)$ precisely when $x^2 \equiv n \pmod{p}$. The solutions x to this congruence lie in the union of two arithmetic progressions with common difference p. If the roots of $x^2 \equiv n \pmod{p}$ are x_1 and x_2, then the arithmetic progressions begin with the first numbers $\equiv x_1$ and $x_2 \pmod{p}$ which are $\geq a$. The sieve is much faster than Trial Division. Pomerance proved that the time complexity of the QS is $L(n) = \exp\left(\sqrt{(\ln n) \ln \ln n}\right)$.

Several variations speed the practical QS Algorithm, although they do not change its theoretical complexity. They include:

- Using large primes (larger than B), as in CFRAC.
- Multiple polynomials (not just $f(x) = x^2 - n$) [62] and [2].
- Self-initializing polynomials (amortize the root-finding) [2].

The front page of the *New York Times* newspaper of October 12, 1988, had an article titled, “A Most Ferocious Math Problem Tamed.” In it, journalist Malcom W. Browne described how Lenstra and Manasse factored a hard 100-digit number by the QS.

In 1994, the QS was used to factor the 129-digit RSA challenge number mentioned in Section 1.2. Atkins, Graff, Lenstra and Leyland [3] used a variation of the QS with two large primes [34] allowed in each relation. This means that each relation may have zero, one or two primes larger than the greatest one in the factor base. When the relations are harvested at the end of the sieve, after the primes in the factor base have been removed from $f(s)$, if the remaining cofactor is composite, then some effort is made to factor this number, often with the Elliptic Curve Method. If it can be factored easily, then the relation is saved. Using two large primes complicates the linear algebra only slightly.

In the 1990s, George Sassoon organized all the personal computers on the Isle of Mull, Scotland, into a factoring group. They used the multiple polynomial version of the QS, with each machine sieving a different set of polynomials. As these computers were not interconnected, the relations were collected on floppy disks and carried to one machine for the linear algebra. They factored more than a dozen numbers this way, including one with 101-digits.

In another large factoring endeavor, many people used multiple personal computers to distribute the sieving of the Quadratic or Number Field Sieve, and then emailed the relations to one large machine used for the linear algebra. See Lenstra and Manasse [33] for factoring by email.

1.5 The Revolution: A Number Field Sieve

Pollard [48] invented the *Number Field Sieve*, NFS, by suggesting raising the degree of the polynomial in the QS, although only for numbers with special form. He factored the Fermat number $F_7 = 2^{2^7} + 1$ (which had been factored earlier by Morrison and Brillhart with CFRAC) using the cubic polynomial $2x^3 + 2$ on a small computer. Manasse and the Lenstra brothers [37] soon extended Pollard's ideas to higher degree polynomials, but still only for numbers of the form $r^e - s$, for small integers r and $|s|$. Their goal was to factor F_9, the smallest Fermat number with no known prime factor. They hoped to use the special form of $F_9 = 2^{512} - (-1)$ to make the numbers that had to be smooth smaller than those needed for the QS. After they factored F_9 in 1990, they and others extended the NFS to general numbers. See Lenstra and Lenstra [32] for more details of the early history of this factoring algorithm. See Pomerance [50] for a summary of the algorithm. Crandall and Pomerance [14] describe the modern algorithm.

Recall that the QS produces many relations $x_i^2 \equiv q_i \pmod{n}$ with q_i factored. After we have enough relations, we match the prime factors of the q_i and create a subset of the q_i whose product is square. In this way, we find congruences $x^2 \equiv y^2 \pmod{n}$ which may factor n by Theorem 1.7.

Now drop the requirement that the left side of a relation must be square. Instead seek relations $r_i \equiv q_i \pmod{n}$ in which both r_i and q_i have been factored completely. Use linear algebra to match the prime factors of r_i *and* the prime factors of q_i and select a subset of the relations for which both the product of the r_i and the product of the q_i are square. This is a fine idea, but too slow to be practical. The main difficulty is that at least one of $|r_i|$, $|q_i|$ must exceed $n/2$, so it has little chance of being smooth.

The NFS solves this problem by letting the numbers on one side of each relation be algebraic integers from an algebraic number field. The idea is to match the irreducible factors so that each occurs an even number of times and hope the product of the algebraic integers in the selected subset of the relations might be a square in the algebraic number field.

1.5.1 Number Fields

See the books [24], [25], [19], [23] for more about number fields. An *algebraic number* is the zero of a polynomial with integer coefficients. If the polynomial is monic, then the algebraic number is called an *algebraic integer*. An *algebraic number field* is a field which contains only algebraic numbers. The smallest algebraic number field containing the algebraic number α is written $\mathbb{Q}(\alpha)$.

The set of all algebraic integers in $\mathbb{Q}(\alpha)$ is written $\mathbb{Z}(\alpha)$. This set forms a commutative ring with unity. A *unit* in $\mathbb{Z}(\alpha)$ is an element having a multiplicative inverse in $\mathbb{Z}(\alpha)$. A nonzero, nonunit element γ of $\mathbb{Z}(\alpha)$ is *irreducible* if it can be factored in $\mathbb{Z}(\alpha)$ only as $\gamma = u\beta$ where u is a unit. When $\gamma = u\beta$, where u is a unit, β is called an *associate* of γ (and γ is an associate of β). An algebraic integer γ has *unique factorization* (in $\mathbb{Z}(\alpha)$) if any two factorizations of γ into the product of irreducible elements and units are the same except for replacing irreducibles by their associates and using different units.

The polynomial of lowest degree having an algebraic number α as a zero must be irreducible, that is, it doesn't factor into the product of two polynomials of lower degree. If an algebraic number α is a zero of the irreducible polynomial $f(x) \in \mathbb{Z}[x]$, then the *conjugates* of α are all of the zeros of $f(x)$. The *norm* $\mathcal{N}(\alpha)$ of α is the product of all of the conjugates of α including α. The norm of an algebraic integer is a rational integer. The norm function is multiplicative: $\mathcal{N}(\alpha\beta) = \mathcal{N}(\alpha)\mathcal{N}(\beta)$. If $\beta = \gamma^2$ for some $\gamma \in \mathbb{Z}(\alpha)$, then $\mathcal{N}(\beta)$ is the square of the integer $\mathcal{N}(\gamma)$. If the algebraic integer α is a zero of the irreducible polynomial $f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1x + c_0$ and a and b are integers, then the norm of $a - b\alpha$ is $\mathcal{N}(a - b\alpha) = F(a, b)$, where F is the homogeneous polynomial

$$F(x, y) = x^d + c_{d-1}x^{d-1}y + \cdots + c_1xy^{d-1} + c_0y^d = y^d f(x/y). \tag{1.7}$$

1.5.2 The Number Field Sieve

We now return to the NFS. Recall that we are letting the numbers on one side of each relation be algebraic integers from an algebraic number field. Then we proposed to match the irreducible factors so that each occurs an even number of times and thus form two squares of integers congruent modulo the number n to factor.

The first problem is writing a congruence modulo n with an algebraic integer on one side. This problem is solved by using a homomorphism h from the algebraic integers $\mathbb{Z}(\alpha)$ to Z_n, the integers modulo n. Suppose we have many algebraic integers θ_i, each factored into irreducibles, and also every $h(\theta_i)$ factored into the product of primes. Then we may match the irreducibles and match the primes to choose a subset of the θ_i whose product is a square γ^2 in $\mathbb{Z}(\alpha)$ and so that the product of the $h(\theta_i)$ is a square y^2 in the integers. Let $x = h(\gamma)$, a residue class modulo n. We have

$$x^2 = (h(\gamma))^2 = h(\gamma^2) = h\left(\prod_{i\in S}\theta_i\right) = \prod_{i\in S} h(\theta_i) \equiv y^2 \pmod{n},$$

which may factor n by Theorem 1.7.

Now we explain how to choose the algebraic number field and construct the homomorphism. We want to have an irreducible monic polynomial

$$f(x) = x^d + c_{d-1}x^{d-1} + \cdots + c_1x + c_0$$

with integer coefficients. Let α be a zero of f in the complex numbers $\mathbb{C}$. The algebraic number field will be $\mathbb{Q}(\alpha)$. Let $\mathbb{Z}[\alpha]$ be the set of all $\sum_{j=0}^{d-1} a_j\alpha^j$, where the a_j are integers. This is a ring contained in the ring $\mathbb{Z}(\alpha)$ of integers of $\mathbb{Q}(\alpha)$. We also need an integer m for which $f(m) \equiv 0 \pmod{n}$. The homomorphism from $\mathbb{Z}[\alpha]$ to Z_n will be defined by setting $h(\alpha) = m \pmod{n}$, that is,

$$h\left(\sum_{j=0}^{d-1} a_j\alpha^j\right) \equiv \sum_{j=0}^{d-1} a_jm^j \pmod{n}.$$

The numbers θ will all have the form $a - b\alpha$. We seek a set $\mathcal{S}$ of pairs (a, b) of integers such that

$$\prod_{(a,b)\in\mathcal{S}} (a - bm) \text{ is a square in } \mathbb{Z}, \tag{1.8}$$

and

$$\prod_{(a,b)\in\mathcal{S}} (a - b\alpha) \text{ is a square in } \mathbb{Z}[\alpha]. \tag{1.9}$$

Let the integer y be a square root of the first product. Let $\gamma \in \mathbb{Z}[\alpha]$ be a square root of the second product. We have $h(\gamma^2) \equiv y^2 \pmod{n}$, since $h(a - b\alpha) \equiv a - bm \pmod{n}$. Let $x = h(\gamma)$. Then $x^2 \equiv y^2 \pmod{n}$, which will factor n with probability at least $1/2$, by Theorem 1.7.

The degree d of the polynomial $f(x)$ is 4, 5 or 6 for numbers we currently factor. In addition to being irreducible and having a known zero m modulo n, we want $f(x)$ to have "small" coefficients compared to n. The next two sections give ways we might satisfy all these conditions.

1.5.3 The Special Number Field Sieve

The requirements on $f(x)$ are easily met in the *Special Number Field Sieve*, which factors numbers $n = r^e - s$, where r and $|s|$ are small positive integers. Suppose we decide (based on experience) that d is the proper degree for the polynomial. Let k be the least positive integer for which $kd \geq e$. Let $t = sr^{kd-e}$. Let $f(x)$ be the polynomial $x^d - t$. Let $m = r^k$. Then $f(m) = r^{kd} - sr^{kd-e} = r^{kd-e}n \equiv 0 \pmod{n}$.

Example 1.8 Let us factor $n = 7^{346}+1$. Let $d = 6, m = 7^{58}$ and $f(x) = x^6+49$. then $f(m) = (7^{58})^6 + 49 = 7^{348} + 7^2 = 7^2(7^{346} + 1) = 49n \equiv 0 \pmod{n}$.

Kleinjung, Bos and Lenstra [27] factored many large Mersenne numbers $2^p - 1$ with the Special NFS. These factorizations are the current record for factoring integers by the NFS.

1.5.4 The General Number Field Sieve

In the *General Number Field Sieve*, n lacks the special form just considered, so there is no obvious polynomial. One standard approach to finding a good polynomial (of degree 5, say) to factor n is to let m be an integer slightly larger than $n^{1/5}$. Write $n = \sum_{i=0}^{5} d_i m^i$ in base m with digits d_i in the interval $0 \leq d_i < m$, small compared to n. Let the polynomial be $f(x) = \sum_{i=0}^{5} d_i x^i$. See the article [9] by Buhler, Lenstra and Pomerance for the origins of the General Number Field Sieve. Montgomery and Murphy [43] give better ways to choose a polynomial for the General Number Field Sieve. Of course, the GNFS would be used to factor an RSA public key because it would not have the special form $r^e - s$.

1.5.5 The Number Field Sieve Again

We now return to both forms of the NFS. The program will have two sieves, one for $a - bm$ and one for $a - b\alpha$. The sieve on $a - bm$ is simple: For each fixed $0 < b < M$ we try to factor the numbers $a - bm$ for $-M < a < M$ by the sieve algorithm in Section 1.4.1.

The goal of the sieve on the numbers $a - b\alpha$ is to allow us to choose a set $\mathcal{S}$ of pairs (a, b) so that the product in Equation (1.9) is a square. Rather than try to factor the algebraic integers $a - b\alpha$, let us work with their norms. If the product in Equation (1.9) is a square, then its norm is a square, and its norm is the product of all $\mathcal{N}(a - b\alpha)$ with $(a, b) \in \mathcal{S}$. Since the norms are rational integers, rather than algebraic integers, it is easy to match the prime factors of norms to form squares. Furthermore, the norm of $a - b\alpha$ is a polynomial, $F(a, b)$ in (1.7), and therefore is something we can factor with sieve algorithm in Section 1.4.1. For each fixed b between 0 and M, sieve the polynomial $F(x, b) = \mathcal{N}(x - b\alpha)$ for x between $-M$ and M to find smooth values of $\mathcal{N}(a - b\alpha)$.

Whenever both $a - bm$ and $\mathcal{N}(a - b\alpha)$ are smooth, save the pair (a, b) to represent the relation $h(a - b\alpha) \equiv a - bm \pmod{n}$. After we have found many relations, use linear algebra to construct sets $\mathcal{S}$ of pairs (a, b) for which the product of $a - bm$ is a square and the product of the norms of $a - b\alpha$ is a square.

Several problems arise, even in this simple description of the NFS algorithm. For example, the fact that $\mathcal{N}(\theta)$ is square need not imply θ is square. One problem is that the norm function does not distinguish among associates. Another is the lack of unique factorization in most number fields. A third problem is computing the square root of algebraic number. All of these problems can be solved. See Crandall and Pomerance [14] for the details.

One can show that the NFS is faster than the QS when n is large and the degree d is chosen well. A careful analysis shows that the time complexity of the NFS is $\exp\left(c(\ln n)^{1/3}(\ln\ln n)^{2/3}\right)$ for some constant $c > 0$. The constant c is smaller for the Special than for the General NFS because the coefficients can be made smaller. Lenstra et al. [10] factored several RSA challenge numbers using the General NFS.

Shamir [59] (see also Lenstra and Shamir [35]) proposed an optoelectronic device called TWINKLE to execute the Number Field Sieve algorithm. Each prime in the factor base is represented by a light-emitting diode. The LED for prime p glows once every p time units; its intensity is proportional to $\log p$. A sensor views the total intensity of all LEDs and saves the current value of a time unit counter whenever the intensity exceeds a threshold. It locates B-smooth numbers by adding the logarithms of their prime factors.

1.6 An Exquisite Diversion: Elliptic Curves

In 1985, H. W. Lenstra, Jr. [39] invented a factoring method using elliptic curves. Lenstra freed Pollard's $p - 1$ factoring algorithm from its dependency on a single number, namely $p - 1$, being smooth. He replaced the multiplicative group of integers modulo p in that algorithm with an elliptic curve modulo p.

Now the integer that must be smooth is the size of the elliptic curve modulo p. This number is about the same size as p. Soon after this discovery, Miller [42] and Koblitz [28] made the same group replacement in many cryptographic algorithms. This change allows one to use smaller p and construct fast cryptographic algorithms with the same level of security as slower ones. Now elliptic curves are an important tool in cryptography.

1.6.1 Basic Properties of Elliptic Curves

An elliptic curve is the graph of an equation like $y^2 = x^3 + ax^2 + bx + c$ with one extra point ∞. For simplicity we use the Weierstrass form $y^2 = x^3 + ax + b$. For cryptography, a, b, x and y lie in a finite field. For factoring, they are integers modulo p or n. We begin by allowing them to be real numbers so that the elliptic curve is a curve in the plane with an added point ∞.

We will define a way to "add" points of the set

$$E_{a,b} = \{(x, y) : x, y \in K,\ y^2 = x^3 + ax + b\} \cup \{\infty\}.$$

If $P = (x, y)$ lies on the graph of $y^2 = x^3 + ax + b$, define $-P = (x, -y)$, that is, $-P$ is P reflected in the x-axis. Also define $-\infty = \infty$.

Given two points P and Q, on the graph but not on the same vertical line, define $P + Q = R$, where $-R$ is the third point on the straight line through P and Q. If P and Q are distinct points on the graph and on the same vertical line, then they must have the form $(x, \pm y)$, that is, $Q = -P$, and we define $P + Q = P + (-P) = \infty$. Also define $P + \infty = \infty + P = P$ for any element P of the elliptic curve (including $P = \infty$). To add a point $P \neq \infty$ to itself, draw the tangent line to the graph at P. If the tangent line is vertical, then $P = (x, 0)$ and we define $P + P = \infty$. If the tangent line is not vertical, then it intersects the graph in exactly one more point R, and we define $P + P = -R$. (If P is a point of inflection, then $R = P$.)

Theorem 1.9 *An elliptic curve $E_{a,b}$ with the addition operation $+$ forms an abelian group with identity ∞. The inverse of P is $-P$.*

We will need formulas for computing the coordinates of $P + Q$ in terms of those of P and Q. If one of P, Q is ∞, then we have already defined $P + Q$. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be on the graph of $y^2 = x^3 + ax + b$. If $x_1 = x_2$ and $y_1 = -y_2$, then $P = -Q$ and $P + Q = \infty$. Otherwise, let s be the slope defined as follows. When $P \neq Q$, let $s = (y_2 - y_1)/(x_2 - x_1)$ be the slope of the line through P and Q. When $P = Q$, let $s = (3x_1^2 + a)/(2y_1)$ be the slope of the tangent line to the graph at P. Then $P + Q = (x_3, y_3)$, where $x_3 = s^2 - x_1 - x_2$ and $y_3 = s(x_1 - x_3) - y_1$.

Now consider elliptic curves modulo a prime p. The formulas for adding points show that if a and b and the coordinates of points P and Q on the elliptic curve $E_{a,b}$ are all rational numbers, then the coordinates of $P+Q$ will be rational numbers (unless $P+Q = \infty$). Therefore, if a and b and the coordinates of points P and Q on the elliptic curve $E_{a,b}$ are integers modulo p, then the coordinates of $P + Q$ will be integers modulo p, unless $P + Q = \infty$, *provided that any division needed in the slope calculation is by a number relatively prime to* p. Of course, the graph is just a set of pairs of numbers modulo p, not a curve in the plane.

When i is a positive integer and P is a point on an elliptic curve, let iP mean P added to itself i times. It is easy to compute iP when i is large. The algorithm below takes about $\log_2 i$ point additions on the elliptic curve.

Fast Point Multiplication.

```
Input: A point $P$ on $E_{a,b}$ modulo $m$ and an integer $i \geq 0$.
$Q := \infty$
$R := P$
**while** $(i > 0)$ {
    **if** ($i$ is odd) { $Q := (Q + R) \bmod m$ }
    $R := (R + R) \bmod m$
    $i := \lfloor i/2 \rfloor$
    }
Output: $iP$ = the final value of $Q$.
```

Theorem 1.10 *Let p be an odd prime. Let (r/p) denote the Legendre symbol. The number $M_{p,a,b}$ of points on the elliptic curve $y^2 \equiv x^3 + ax + b \pmod p$ satisfies $M_{p,a,b} = p + 1 + \sum_{x=0}^{p-1} \big((x^3 + ax + b)/p\big)$.*

Since the Legendre symbol in Theorem 1.10 has the value $+1$ about as often as the value -1, we expect the number of points on a random elliptic curve modulo p to be close to $p + 1$. Hasse proved that this is so.

Theorem 1.11 (Hasse) *Let the elliptic curve $E_{a,b}$ modulo a prime p have $M_{p,a,b}$ points. Then $p + 1 - 2\sqrt{p} \leq M_{p,a,b} \leq p + 1 + 2\sqrt{p}$.*

The range of possible values for $M_{p,a,b}$ is called the *Hasse interval*.

1.6.2 Factoring with Elliptic Curves

In 1985, H. W. Lenstra, Jr. [39] invented the Elliptic Curve Method or ECM, a factoring algorithm using elliptic curves. Let R_p denote the multiplicative group of integers modulo a prime p. Recall that Pollard's $p - 1$ Factoring Algorithm in Section 1.1.3 performs a calculation in the integers modulo n that hides a calculation in R_p. The factor p of n is discovered when the size $p - 1$ of the group R_p divides $L = B!$, but p is not found when $p - 1$ has a prime divisor larger than B. Lenstra replaced R_p with an elliptic curve group $E_{a,b}$ modulo p. By Hasse's Theorem, the two groups have roughly the same size, namely, approximately p. Lenstra's algorithm discovers p when $M_{p,a,b}$ divides $L = B!$. It fails to find p when $M_{p,a,b}$ has a prime factor larger than B. There is only one group R_p, but lots of elliptic curve groups $E_{a,b}$ modulo p. If the size of R_p has a prime factor $> B$, we are stuck. But if the size of $E_{a,b}$ modulo p has a prime factor $> B$, we just change a and b and try a new elliptic curve. Each curve gives an independent chance to find the prime factor p.

Compare this algorithm with Pollard's $p - 1$ algorithm. The two algorithms work exactly the same way, except that Pollard's $p - 1$ algorithm raises a to the power i while the Elliptic Curve Method multiplies P by i. The former algorithm explicitly computes a greatest common divisor with n while the latter algorithm hides this operation in the slope calculation of the elliptic curve point addition. The quantity $4a^3 + 27b^2$ in the algorithm is the discriminant of the cubic polynomial, which tells whether it has a repeated zero. For technical reasons, the algorithm excludes polynomials with a repeated zero.

Simple Elliptic Curve Factorization Method

```
Input: A composite positive integer $n$ to factor and a bound $B$.
Find the primes $p_1 = 2, p_2, \ldots, p_k \le B$
Choose a random elliptic curve $E_{a,b}$ modulo $n$
    and a random point $P \ne \infty$ on it
$g := \gcd(4a^3 + 27b^2, n)$
**if** $(g = n)$ { choose a new curve and point $P$ }
**if** $(g > 1)$ { report the factor $g$ of $n$ and stop }
**for** $(i := 1$ to $k)$ {
    $P := iP$ or else find a factor $g$ of $n$
    }
Give up or try another random elliptic curve.
Output: A proper factor $p$ of $n$, or else give up.
```

A good way to choose the elliptic curve parameters is to choose a random a modulo n and a random point $P = (x_1, y_1)$ modulo n and then let $b = (y_1^2 - x_1^3 - ax_1) \bmod n$. In other words, b has the correct value modulo n so that the point P is on $E_{a,b}$. Lenstra [39] proved that when many curves and points are chosen this way, the sizes $M_{p,a,b}$ of the curves modulo a prime p are well distributed in the Hasse interval.

Whenever two points are added during the computation of iP the coordinates are reduced modulo n. Imagine that the coordinates are also reduced modulo p, an unknown prime divisor of n. If the size $M_{p,a,b}$ of the elliptic curve modulo p divides $L = i!$, then $LP = \infty$ in the elliptic curve modulo p. Since $P \ne \infty$, somewhere during the calculation we must have $P_1 + P_2 = \infty$ for two points

$P_1, P_2 \neq \infty$, working with coordinates modulo p. This means that P_1 and P_2 will have the same x-coordinate modulo p. But we are computing modulo n, and the x-coordinates of the two points will probably not be the same modulo n. In the slope computation, we will try to invert a number not relatively prime to n, so we will factor n instead.

Theorem 1.12 *Let n be a positive integer with an unknown prime factor p. Let B be the optimal bound for finding p by the elliptic curve algorithm. Assume that a random elliptic curve $E_{a,b}$ modulo p has a B-smooth size with probability u^{-u}, where $u = (\ln p)/\ln B$, as Dickman's theorem [16] would predict. Define $L(x) = \exp\left(\sqrt{(\ln x)\ln\ln x}\right)$. Then $B = L(p)^{\sqrt{2}/2}$. The expected total number of point additions performed when the Elliptic Curve algorithm is used to discover p is $L(p)^{\sqrt{2}}$. The expected total work needed to discover one prime factor of n is at most $L(n)$ point additions.*

The ECM has a second stage, just like the Pollard $p - 1$ algorithm. The second stage finds a factor p of n when the largest prime factor of $M_{p,a,b}$ is less than B_2 and all the other prime factors of $M_{p,a,b}$ are less than B.

Efficient modern versions of the ECM discover prime factors up to about 20 digits in a few seconds and those up to about 40 digits in a few hours. Luck is required to discover factors having more than 60 digits. See Silverman and Wagstaff [63] for some practical aspects of ECM. See Zimmermann and Dodson [72] for more about ECM.

1.6.3 Factoring Helping Elliptic Curves Used in Cryptography

Almost all of this chapter concerns factoring to break the RSA public-key cipher. This section, however, uses factoring integers to build efficient cryptographic protocols. There are many uses of elliptic curves in cryptography. Here we mention two ways factoring helps to construct elliptic curves that are well suited for use in cryptography.

One application is how to choose an elliptic curve with simple formulas for adding points. Elliptic curves used in cryptography are groups whose points are defined by an equation over a finite field. The size of a finite field is a prime power $q = p^k$. The size of q is typically about 128 or 256 bits for cryptography. Some of the most useful elliptic curves have size equal to $q \pm 1$. The points (other than ∞) of an elliptic curve over $\mathbb{F}_q$ are pairs of numbers in $\mathbb{F}_q$. The formulas for adding two points use arithmetic in $\mathbb{F}_q$. When q is prime, the arithmetic is in the integers modulo q and it is slow, especially for devices with limited computing power. But if p is a small prime, then arithmetic in $\mathbb{F}_{p^k}$ can be done in steps with numbers no larger than p, so it is fast on a slow machine.

Elliptic curves often lie over $\mathbb{F}_{p^k}$ with $k > 1$. The size of such a curve might be $p^k + 1$. The factorization of the size is needed to ensure that the elliptic curve discrete logarithm problem (ECDLP) is hard. To make ECDLP hard, $p^k + 1$ should have at least one large prime factor. The ECDLP is to find x so that $Q = xP$, where P and Q are two points on an elliptic curve. Many cryptographic algorithms depend on the ECDLP being intractable.

Another use of factoring is for efficient computation of pairings. Let $E_{a,b}$ be an elliptic curve over $\mathbb{F}_q$. A *pairing* on $E_{a,b}$ is a function that takes pairs of points of $E_{a,b}$ to the n-th roots of unity μ_n in an algebraic closure of $\mathbb{F}_q$ satisfying several mathematical properties. The first use of a pairing in cryptography was in an attack on the ECDLP in 1993. The attack by Menezes, Okamoto and Vanstone [41] changes the problem $Q = xP$ on an elliptic curve over $\mathbb{F}_{p^k}$ to an equivalent problem $a = b^x$ in $\mathbb{F}_{p^k}$; the latter problem is easier to solve. Since 2000, many constructive uses of pairings have been found in cryptographic protocols, including three-way key agreement, identity-based encryption and short signatures. These protocols could be broken if the ECDLP were easy.

The naive computation of a pairing by its definition is too slow. There are faster ways to compute pairings for *supersingular* elliptic curves, whose sizes are numbers $p^k \pm 1$. Unfortunately, supersingular elliptic curves are the ones for which the pairing-based MOV attack on the ECDLP works best. Consequently, the parameter choice for these curves is a delicate balance between ease of computing the pairing and the need to keep the ECDLP hard. This decision requires knowledge of the factorization of the elliptic curve size, which has the form $p^n \pm 1$.

See Washington [69] for definitions of the pairings. Boneh and Franklin [7] designed an identity-based encryption system using pairings, which allows someone to use your email address as your public key. See Trappe and Washington [66] for a simple description of the Weil pairing and identity-based encryption. See Estibals [20] for ways of using factors of $2^n - 1$ and $3^n - 1$ to construct elliptic curves in which pairings are easy to compute, but the ECDLP is intractable.

1.7 The Future: How Hard Can Factoring Be?

In this section we discuss factoring from a theoretical computer science point of view and then present some ways to factor RSA public keys. We end with suggestions for new methods of factoring and make some predictions about the future of factoring.

1.7.1 Theoretical Aspects of Factoring

It would be great if we could find an algorithm for factoring n that we could prove always works and runs in a specified time. Of course, Trial Division described in Section 1.1.1 has this property and factors n in $O(n^{1/2})$ steps. There is an algorithm of Pollard [46] and Strassen [65] (see Section 5.5 of [14]) that uses fast polynomial evaluation to factor n in $O(n^{1/4+\varepsilon})$ steps. These algorithms are rigorous and deterministic, that is, they do not choose random numbers and one can prove before they begin that they will succeed in a specified number of steps. Shanks [60] invented a fast algorithm for factoring n by computing the class number of primitive binary quadratic forms of discriminant n. This complicated algorithm runs in $O(n^{1/4+\varepsilon})$ steps, but can be modified to factor n in $O(n^{1/5+\varepsilon})$ steps, assuming the Extended Riemann Hypothesis. Algorithms like Shanks' $O(n^{1/5+\varepsilon})$ one, CFRAC, QS and NFS are fast and deterministic, but the proofs of their running times depend on heuristic hypotheses, so they are not rigorous.

A probabilistic algorithm[1] for factoring integers uses random numbers and may or may not factor the number, depending on the random choices it makes. For a given input, it may perform different steps each time it is invoked and may produce different factorizations or none at all. For example, the Elliptic Curve Method of Section 1.6.2 chooses a random elliptic curve and a random point on it, and then works deterministically to compute a large multiple of the point. For some choices it will factor the input number, perhaps finding different factors (say, if n has two prime factors of about the same size).

A probabilistic algorithm may or may not be rigorous. The Elliptic Curve Method is probabilistic and not rigorous because it assumes without proof that, with a certain probability, there is an elliptic curve having B-smooth size in the Hasse interval. The ability to make random choices is a powerful tool that can make a probabilistic algorithm faster than a deterministic algorithm for the same job. In order to say that a probabilistic algorithm for factoring is rigorous, there must be a proof without unproved assumptions that it will factor the input number n with positive probability in a specified number of steps. It turns out that there is a rigorous probabilistic algorithm for factoring integers with a subexponential running time. Dixon [18] invented such an algorithm in 1981.

Dixon's algorithm for factoring n has two parameters, u and v, to be specified later. It begins by choosing a set S of u random integers between 1 and n. The remainder of the algorithm is deterministic. Call the algorithm A_S. Let the factor base consist of all primes $< v$. For each $z \in S$, try to factor $w = (z^2 \bmod n)$ using only the primes in the factor base. If you succeed in factoring w com-

[1] Technically, a Las Vegas probabilistic algorithm.

pletely, save the relation $z^2 \equiv$ (the product of the factors of w) (mod n). After the factoring is finished, combine the relations using linear algebra modulo 2 as in the QS. The method used to factor the numbers w doesn't matter; even Trial Division would be fast enough for the theorem.

Let $L(n) = \exp\left(\sqrt{(\ln n)\ln\ln n}\right)$. Dixon [18] proved this theorem about the set of algorithms $\{A_S : S \subseteq [1, n], \text{size}(S) = u\}$.

Theorem 1.13 (Dixon) *Let n be an odd integer with at least two different prime factors. Let $v = L(n)^{\sqrt{2}}$ and $u = \lceil v^2 \rceil$. Then the average number of steps taken in the execution of algorithm A_S is $L(n)^{3\sqrt{2}}$. The probability that algorithm A_S fails to factor n is proportional to $L(n)^{-\sqrt{2}}$, uniformly in n.*

The theorem says that when n is large, almost all of the algorithms A_S with correct parameters u and v will factor n successfully and that all A_S run in subexponential time $L(n)^{3\sqrt{2}}$.

Although they have rigorous proofs of their time complexities, the algorithms mentioned above are much slower than the QS and NFS.

The RSA cryptosystem and several other cryptographic algorithms depend on factoring integers being a hard problem. Can we prove that factoring integers is hard?

One must phrase this question carefully. Suppose n has a million decimal digits, and the low-order digit is 6. Then obviously 2 is a prime factor of n. One can trivially "factor" n as $2 \cdot (n/2)$.

Here is one way to ask the question. As a function of n, what is the minimum, taken over all integer factoring algorithms, of the maximum, taken over all integers M between 2 and n, of the number of steps the algorithm takes to factor M? The integer factoring algorithm has input M and outputs a list of all prime factors of M. Integer factoring would be a polynomial-time problem if the answer were $\mathrm{O}((\log n)^c)$ for some constant c.

The model of computation, that is, what constitutes a "step," probably matters a lot in answering the question. Suppose we decide that a single arithmetic operation ($+$, $-$, $\times$, $\div$) is one step. Assume that integers are stored in binary notation and that each register (memory location) may hold one integer of any size. Shamir [58] proved that, in this model, one can factor n in $\mathrm{O}(\log n)$ steps. Shamir's result shows that we can factor n in a polynomial (in $\log n$) number of steps provided that an arithmetic operation with numbers as large as $n!$ counts as one step. This shows why we need to consider the complexity of arithmetic with large numbers when analyzing number theoretic algorithms.

1.7.2 Factor an RSA public key

In this section and the following we consider various ways to factor a large integer with the help of special information available about it. Some of the special information might include its use as a public key for the RSA cipher.

Theorem 1.14 *There is a polynomial time algorithm for factoring a composite integer n, given a base a to which n is a pseudoprime but not a strong pseudoprime.*

If we could find such a base quickly, then we would have a fast method of factoring integers.

There are ways to factor an RSA public modulus n when some information is known about it or the parameters are chosen poorly. None of these tricks actually threaten the RSA cipher because, when the cipher is used properly, the needed information about the modulus is not leaked.

A. Lenstra et al. [38] collected several million 1024-bit RSA public keys from X.509 certificates and PGP on the Web. Most of these keys appeared valid. They computed the GCD of each pair of these n and were able to factor a few of them because they shared a single secret prime factor. This finding suggests poor seeds for random number generators, that is, two (or more) users used the same seed to generate a prime for their RSA key.

There are other attacks on the RSA cipher besides factoring n. For example, if Alice happens to encipher the same message M via RSA using different enciphering exponents e_1 and e_2, but the same RSA modulus n, then an attacker can recover M from the two ciphertexts without factoring n. Boneh [5] describes all the attacks below and many more.

Theorem 1.15 *If n is the product of two different primes p and q, then one can factor n in polynomial time, given n and $\phi(n)$.*

Proof We have $\phi(n) = (p - 1)(q - 1) = n - (p + q) + 1$ and $n = pq$. Thus, $n + 1 - \phi(n) = p + q = p + n/p$ or $p^2 - (n + 1 - \phi(n))p + n = 0$. This quadratic equation in the unknown p has the two solutions p and q, which may be computed easily by the quadratic formula. □

Theorem 1.15 shows why one must not reveal $\phi(n)$ when n is an RSA public key. One must also not reveal the deciphering exponent d when n is an RSA public key. Not only could one decipher all ciphertext knowing d, but one can factor n, too. Of course, the enciphering exponent e is public. This theorem appears in the original RSA paper [55].

Theorem 1.16 *There is a probabilistic polynomial time algorithm to factor*

n, given an integer n which is the product of two unknown primes and given two integers e, d, between 1 *and n with $ed \equiv 1 \pmod{\phi(n)}$.*

There is a *deterministic* method for factoring n, given e and d, but it is more complicated. As we saw above, we can factor n given n and $\phi(n)$ by solving a quadratic equation. May [40] showed how to find $\phi(n)$ from n, e and d in deterministic polynomial time. His result depends on a theorem of Coppersmith telling how to find small solutions to a polynomial congruence modulo n. Coppersmith's theorem relies in turn on the LLL [36] algorithm for finding a reduced basis for a lattice. As this work leads to several other attacks on RSA, we describe them in the following section.

Here is one more way to factor n when a parameter is chosen badly. One can factor n when d is unknown, but satisfies $d \leq \sqrt[4]{n}/3$, according to Wiener [70].

Theorem 1.17 (Wiener) *There is a polynomial time algorithm which can factor n, given n and e, provided $e < \phi(n)$ and $n = pq$, where p and q are (unknown) primes with $q < p < 2q$, and there is an (unknown) integer $d < n^{1/4}/3$ satisfying $ed \equiv 1 \pmod{\phi(n)}$.*

Finally, we give an example of factoring an RSA modulus n by exploiting a hardware failure during signature generation. We explained in Section 1.2 how Alice can set up RSA as a public key cipher to receive encrypted messages from Bob. Alice can sign a message M to Bob as $S = M^d \bmod n$, where d is her private RSA deciphering exponent. When Bob receives S he can verify Alice's signature by getting her public RSA key n, e, and computing $M = S^e \bmod n$. The fact that this M is meaningful text shows that it came from Alice because only Alice could construct such a signature S. Now she can accelerate this calculation of S by a factor of four using her knowledge of the factors p and q of n. She computes $S_p = S \bmod p$ and $S_q = S \bmod q$ by Fast Exponentiation with smaller numbers and then combines these values with the Chinese Remainder Theorem to obtain S.

Suppose that a hardware error complements one bit of S_p during this calculation, but S_q is computed correctly, so that she sends Bob an incorrect signature S'. Suppose also that an eavesdropper Eve obtains S' and M. Eve can factor Alice's public modulus n, given only S', n, e and M, where e is Alice's public encryption exponent. We have $M = S^e \bmod n$. Since the error occurred in computing S_p, we have $M \equiv (S')^e \pmod{q}$ but $M \not\equiv (S')^e \pmod{p}$. Therefore, $\gcd(M - (S')^e, n) = q$. This attack is due to Lenstra et al. [26]. See Boneh [5] for another similar attack, in which Eve has S and S', but not M.

1.7.3 Factoring With Lattices

In this section we give a brief overview of lattices and applications to factoring an RSA public modulus. The definitions and theorems in this section are somewhat vague. See [36] for the true definitions and theorems.

Lattices have many uses in cryptography. They may be used to define cryptosystems and to attack RSA and other ciphers. Several attacks on the RSA public key cipher with poor parameter choices use lattices and the LLL algorithm. We describe here only the ways lattices can be used to factor an RSA modulus.

Definition 1.18 The lattice generated by linearly independent vectors $\vec{v}_1, \ldots,$ $\vec{v}_r$ with integer coordinates is the set of all linear combinations $a_1\vec{v}_1 + \cdots + a_r\vec{v}_r$ with *integers* a_i.

A *basis* for a lattice L is a set of linearly independent vectors $\vec{v}_1, \ldots, \vec{v}_r$ with integer coordinates such that any vector in L can be written as a linear combination $a_1\vec{v}_1 + \cdots + a_r\vec{v}_r$ of the basis vectors with *integer* coefficients a_i.

Every lattice has a basis. Every basis of a given lattice L has the same size r, called the *rank* of L. The rank r is the same as the dimension of the vector space (a subspace of $\mathbb{R}^n$) spanned by any basis of L. A lattice has *full rank* if $r = n$.

Recall that if $\vec{v}_1, \ldots, \vec{v}_r$ and $\vec{w}_1, \ldots, \vec{w}_r$ are two bases for the same vector space, then each $\vec{w}_i$ can be written as a linear combination of the $\vec{v}_j$.

Likewise, if $\vec{v}_1, \ldots, \vec{v}_r$ and $\vec{w}_1, \ldots, \vec{w}_r$ are two bases for the same lattice, then each $\vec{w}_i$ can be written as a linear combination of the $\vec{v}_j$ with *integer* coefficients. Let B be the $r \times n$ matrix whose rows are $\vec{v}_1, \ldots, \vec{v}_r$. The "size" of a lattice L is a real number $\det(L)$ defined in terms of any basis. If L has full rank, then B is square and $\det(L) = |\det(B)|$, the latter "det" being the usual determinant. This is the only case we will use below.

There exists a vector $\vec{v}_1$ in a lattice L with minimum positive norm, the shortest vector. Let $\lambda_1(L)$ be the norm of this shortest $\vec{v}_1$. Every vector $\vec{w} \in L$ which is linearly dependent on $\vec{v}_1$ must be $\vec{w} = t\vec{v}_1$ for some integer t. At least two vectors have this minimum positive length since the norm of $-\vec{v}$ equals the norm of $\vec{v}$.

For integer $k \geq 1$, let $\lambda_k(L)$ be the smallest positive real number so that there is at least one set of k linearly independent vectors of L, with each vector having length $\leq \lambda_k(L)$. This defines a sequence

$$\lambda_1(L) \leq \lambda_2(L) \leq \lambda_3(L) \leq \cdots .$$

Note that we count lengths, not vectors, in this definition.

A lattice is often presented by giving a basis for it. This basis sometimes consists of very long, nearly parallel vectors. Sometimes it is more useful to have a basis with shorter vectors that are closer to being orthogonal to each other. The process of finding such a basis from a poor one is called *reducing the lattice* or *lattice reduction*.

The nicest basis $\{v_1, v_2, \ldots, v_r\}$ for L would have the length of v_i be $\lambda_i(L)$ for each i. It turns out to be $\mathcal{NP}$-hard to find a nicest basis.

The Gram-Schmidt process does not work for lattices because the $m_{i,j}$ are usually not integers, so the new basis vectors are not *integer* linear combinations of the original vectors. There are several definitions of reduced basis and they are not equivalent. Lenstra, Lenstra and Lovász [36] give one definition. Its vectors approximate the shortest possible vectors for a basis of a given lattice, and the angle between any two reduced basis vectors is not allowed to be too small. They give a polynomial time algorithm, related to the Gram-Schmidt process, but more complicated, that computes a reduced basis from a given one. They prove that their algorithm, called the LLL algorithm runs in polynomial time and constructs a reduced basis.

We will use only the following special application of the LLL algorithm.

Theorem 1.19 (LLL [36]) *Let L be a lattice spanned by $\vec{v}_1, \ldots, \vec{v}_r$. With input $\vec{v}_1, \ldots, \vec{v}_r$, the LLL algorithm finds in polynomial time a vector $\vec{b}_1$ with length*

$$||\vec{b}_1|| \le 2^{(r-1)/4} \det(L)^{1/r}.$$

Now we describe some attacks on RSA using LLL.

Recall the notation of RSA: $n = pq$ is the product of two large primes and is hard to factor. Choose e with $\gcd(e, \phi(n)) = 1$, where $\phi(n) = (p-1)(q-1)$. Via the Extended Euclidean Algorithm, find d with $ed \equiv 1 \pmod{\phi(n)}$. Discard p and q. The public key is n, e and the private key is d. Encipher plaintext M as $C = M^e \bmod n$. Decipher ciphertext C as $M = C^d \bmod n$.

Consider these three problems about the RSA cipher.

1. The RSA problem: Given n, e and C, find M.
2. Compute d: Given n and e, find d.
3. Factor n: Given n, find p and q.

Clearly, if we can solve (3), then we can solve (2); and if we can solve (2), then we can solve (1). In fact, (3) is equivalent to (2). It is not known whether (3) is equivalent to (1). The next theorem was proved by May [40]. Its proof uses the LLL algorithm [36].

Theorem 1.20 (May) *Let $n = pq$, where p and q are two primes of the same bit length. Let positive integers e and d satisfy $ed \equiv 1 \pmod{\phi(n)}$ and $ed \le n^2$.*

There is a deterministic polynomial time algorithm which factors n given the input n, e, d.

Theorem 1.20 is a beautiful theoretical result. It says that knowing the RSA deciphering exponent is deterministically polynomial time equivalent to knowing the factors of n. Of course, if an RSA deciphering exponent were accidentally leaked, one could factor n easily in probabilistic polynomial time by the method of Theorem 1.16.

Now assume we have some partial information about p or q or d because they are limited somehow, perhaps by a poor choice of parameters or a faulty random number generator.

Coppersmith [12] proved that if $f(x)$ is a monic polynomial and b is an unknown factor of a given integer n, then one can find all "small" solutions x to $f(x) \equiv 0 \pmod{b}$ quickly via the LLL lattice reduction algorithm.

Theorem 1.20 is one application of his method. Here is another application. We can factor an RSA modulus given a few more than half of the high-order bits of p (as a number $\tilde{p} \approx p$).

Theorem 1.21 (Coppersmith [12]) *One can factor $n = pq$, where $p > q$, in polynomial time, given n and an integer $\tilde{p}$ with $|p - \tilde{p}| < n^{5/28}/2$.*

Similar theorems allow one to factor $n = pq$ given the high order half of the bits of q or the low order bits of either p or q.

Coppersmith [11], [13] also proved a theorem that lets one find small roots of a polynomial with two variables in polynomial time using the LLL algorithm. This result gives a faster algorithm than Theorem 1.21 for factoring $n = pq$ when the high or low order half of the bits of either p or q are known. The bivariate polynomial theorem also gives polynomial time algorithms for factoring an RSA modulus n when some bits of d are known. Here is an example of these results.

Theorem 1.22 (Boneh and Durfee [6]) *There is a polynomial time algorithm to factor $n = pq$, given n and e, provided that there exists an integer $d < n^{0.292}$ such that $ed \equiv 1 \pmod{\phi(n)}$.*

The lesson of Theorems 1.17 and 1.22 is that the deciphering exponent d in RSA should not be too small, or else an attacker will be able to factor n. If you are choosing RSA keys and notice that $d < n^{2/3}$, say, then you should choose new e and d. So far as I know, there is no risk in letting e be small; even $e = 3$ appears to be safe.

1.7.4 The Future of Factoring

One way of predicting the future is by extrapolating from the past. Consider the problem of completely factoring Fermat numbers, and consider the history of how and when each Fermat number was factored.

The Fermat numbers $F_m = 2^{2^m} + 1$ for $5 \leq m \leq 32$ are composite. These numbers have been completely factored for $5 \leq m \leq 11$. This table, adapted from [68], gives the year and method by which the factorization of these seven numbers was finished. The abbreviation NFac denotes the number of prime factors of F_m.

m	NFac	Year	Method	Who
5	2	1732	Trial Division	Euler
6	2	1855	Trial Division and $p \equiv 1 \pmod{2^{m+2}}$	Clausen and Landry
7	2	1970	CFRAC	Morrison and Brillhart
8	2	1980	Pollard Rho	Brent and Pollard
9	2	1990	NFS	Lenstra, Manasse, et al.
10	4	1995	ECM	Brent
11	5	1988	ECM	Brent

Can one predict from this table when more Fermat numbers will be factored? The year increases with m except for the last one. This is because F_{11} has four factors small enough to discover easily by ECM, while F_9 and F_{10} have penultimate factors too big to find easily in 1989. In what year would you guess F_{12} will be completely factored? In 1996, Richard Guy bet John Conway that at least one more Fermat number would be factored completely by his 100th birthday in 2016. Richard lost that bet. Six small factors of F_{12} are known. The remaining cofactor is composite with 1133 decimal digits. This number is too large to factor by QS or NFS. The only way it will be factored soon by a known method is if it has just one very large prime factor and we can discover its small prime factors by ECM. Six different algorithms were used to factor the seven numbers. Perhaps someone will invent a new algorithm to finish F_{12}.

A new factoring algorithm was invented about every five years from 1970 to 1995, as shown in this table taken from [68].

Year	Method	Who
1970	CFRAC	Morrison and Brillhart
1975	Pollard $p - 1$ and Rho	Pollard
1980	Quadratic Sieve	Pomerance
1985	ECM	H.W. Lenstra, Jr.
1990	SNFS	Pollard and Lenstra
1995	GNFS	Pollard and Lenstra

Each of these algorithms did something that previous algorithms could not do.

Why have we found no new faster factoring algorithms since 1995? Many have tried to find new ones. People have fashioned variations of QS, ECM and NFS that lead to faster programs for these algorithms. The improved algorithms are much faster than the first versions, but they still have the same asymptotic time complexity as the first version. Have we already discovered the fastest integer factoring algorithms? I doubt it.

We need a new factoring algorithm. The time complexities of the fastest known algorithms have the form

$$\exp\left(c(\ln n)^t(\ln\ln n)^{1-t}\right), \tag{1.10}$$

for some constants $c > 0$ and $0 < t < 1$. For QS and ECM, $t = 1/2$; for NFS, $t = 1/3$. The time complexity has this shape because these algorithms have to find one or more smooth numbers, and the density of such numbers is stated in Dickman's [16] theorem. Any new factoring algorithm that works by finding smooth numbers probably would have time complexity of the form (1.10). A polynomial time factoring algorithm likely would not rely on smooth numbers.

We suggest ideas for discovering new, faster ways to factor large numbers.

1. Combine the Number Field Sieve and lattices. Sieve the NFS polynomial on the points of a lattice in the plane. C.-P. Schnorr [56], [57] has taken the first steps in this direction.
2. The fastest general algorithms, the QS and GNFS, solve $x^2 \equiv y^2 \pmod{n}$ and use Theorem 1.7. Find a way to construct congruent squares modulo n faster than by using smooth numbers to build relations.
3. Can you quickly find a $1 < b < n$ so that n is a pseudoprime to base b, but not a strong pseudoprime to base b, and apply Theorem 1.14?
4. Factoring algorithms exist with running time (1.10) with $t = 1/2$ and $1/3$. Find one with $t = 1/4$. Note that this t is the reciprocal of 1 plus the number of parameters that you can adjust to "tune" the algorithm. CFRAC, QS and

NFS have factor base size as one parameter; NFS adds the degree of the polynomial as a second parameter. ECM has the parameter B.

5. Find an algorithm to answer this question quickly. Given integers $1 < B < n$, does n have a factor $p \leq B$? If you had a fast algorithm to answer this yes/no question, then you could use it $\log_2 n$ times in a binary search for the least prime factor of n. Note that when $\lfloor \sqrt{n} \rfloor \leq B < n$, the question just asks whether n is composite, and Agrawal, Kayal and Saxena [1] answered that question in polynomial time in 2002. See Section 1.1.4.
6. Build a quantum computer and use Shor's method. In 1994, Shor [61] gave an algorithm for factoring integers in polynomial time using a quantum computer. Basically, Shor's quantum computer computes a Fourier transform with a very sharp peak at $\phi(n)$. When you observe this transform you will learn $\phi(n)$. Then use Theorem 1.15.
7. Simulate Shor's method on a conventional computer. Simulate each qbit by a real variable between 0 and 1. Use non-linear optimization methods (as in [15]) to locate the peak in the transform to find $\phi(n)$.
8. If none of the above works, then try to prove that factoring is hard. Remember Shamir's $O(\log n)$-step algorithm of Section 1.7.1. Can you even prove that there is a constant $c > 0$ such that when factoring some numbers n on a computer with fixed word size, at least $c \log^2 n$ steps are required?

If quantum computers can be made to work, then factoring integers will become easy. A secure RSA public modulus has more than 1000 bits. A quantum computer using Shor's method to factor such a number would require a few million entangled qbits. At present, we can build quantum computers with no more than a few dozen entangled qbits. One problem with building larger quantum computers is decoherence, in which ambient energies observe the entangled qbits before the computation ends. I cannot guess whether this problem will be solved.

Let me go out on a limb and predict that the number of qbits in quantum computers will increase gradually. I also predict that mathematicians will invent slightly faster, but not polynomial time, factoring algorithms. As these new methods are programmed and as quantum computers gradually improve, there will be a slow increase over the next century in the size of integers that humans can factor. This is what has happened during the past fifty years with the Cunningham Project [8], which factors numbers of the form $b^n \pm 1$. The smallest interesting number of this form that remains to be factored has increased smoothly from 30 decimal digits fifty years ago to about 200 digits now, the result of faster algorithms and faster computers.

References

[1] Agrawal, Manindra, Kayal, Neeraj, and Saxena, Nitin. 2004. PRIMES is in P. *Annals of Mathematics*, **160**(2), 781–793.

[2] Alford, W. R., and Pomerance, C. 1993. Implementing the Self-initializing Quadratic Sieve on a Distributed Network. Pages 163–174 of: van der Poorten, A., Shparlinski, I., and Zimmer, H. G. (eds), *Number Theoretic and Algebraic Methods in Computer Science*.

[3] Atkins, Derek, Graff, Michael, Lenstra, Arjen K., and Leyland, Paul C. 1995. The Magic Words are Squeamish Ossifrage. Pages 263–277 of: Pieprzyk, Josef, and Safavi-Naini, Reihaneh (eds), *ASIACRYPT'94*. LNCS, vol. 917. Wollongong, Australia: Springer, Heidelberg, Germany.

[4] Baillie, R., and Wagstaff, Jr., S. S. 1980. Lucas pseudoprimes. *Math. Comp.*, **35**, 1391–1417.

[5] Boneh, D. 1999. Twenty years of attacks on the RSA cryptosystem. *Notices Amer. Math. Soc.*, **46**, 202–213.

[6] Boneh, D., and Durfee, G. 2000. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Trans. on Info. Theory*, **46(4)**, 1339–1349.

[7] Boneh, D., and Franklin, M. 2001. Identity based encryption from the Weil pairing. Pages 213–229 of: *Advances in Cryptology—Proc. CRYPTO '01*. Lecture Notes in Computer Science, vol. 2139.

[8] Brillhart, John, Lehmer, D. H., Selfridge, J. L., Tuckerman, Bryant, and Wagstaff, Jr., S. S. 2002. *Factorizations of* $b^n \pm 1$, *b = 2, 3, 5, 6, 7, 10, 11, 12 up to high powers*. Third edn. Contemporary Mathematics, vol. 22. Providence, Rhode Island: Amer. Math. Soc.

[9] Buhler, J., Lenstra, Jr., H. W., and Pomerance, C. 1993. Factoring integers with the Number Field Sieve. Pages 50–94 of: Lenstra, A. K., and Lenstra, Jr., H. W. (eds), *The Development of the Number Field Sieve*. Lecture Notes in Mathematics, vol. 1554.

[10] Cavallar, Stefania, Dodson, Bruce, Lenstra, Arjen K., Leyland, Paul C., Lioen, Walter M., Montgomery, Peter L., Murphy, Brian, te Riele, Herman, and Zimmermann, Paul. 1999. Factorization of RSA-140 Using the Number Field Sieve. Pages 195–207 of: Lam, Kwok-Yan, Okamoto, Eiji, and Xing, Chaoping (eds), *ASIACRYPT'99*. LNCS, vol. 1716. Singapore: Springer, Heidelberg, Germany.

[11] Coppersmith, D. 1996a. Finding a small root of a bivariate integer equation; factoring with high bits known. Pages 178–189 of: *Advances in Cryptology—Eurocrypt '96*. Lecture Notes in Computer Science, vol. 1070.

[12] Coppersmith, D. 1996b. Finding a small root of a univariate modular equation. Pages 155–165 of: *Advances in Cryptology—Eurocrypt '96*. Lecture Notes in Computer Science, vol. 1070.

[13] Coppersmith, D. 1997. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. Cryptology*, **10(4)**, 233–260.

[14] Crandall, R., and Pomerance, C. 2005. *Prime Numbers: A Computational Perspective*. Second edn. New York: Springer-Verlag.

[15] Dennis, J. E., and Schnabel, R. B. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Computational Mathematics. Prentice-Hall.

[16] Dickman, K. 1930. On the frequency of numbers containing prime factors of a certain relative magnitude. *Ark. Mat., Astronomi och Fysik*, **22A, 10**, 1–14.

[17] Diffie, Whitfield, and Hellman, Martin E. 1976. New Directions in Cryptography. *IEEE Transactions on Information Theory*, **22**(6), 644–654.

[18] Dixon, J. D. 1981. Asymptotically fast factorization of integers. *Math. Comp.*, **36**, 255–260.

[19] Dummit, D. S., and Foote, R. M. 2004. *Abstract Algebra*. Third edn. New York: John Wiley & Sons.

[20] Estibals, N. 2010. Compact hardware for computing the Tate pairing over 128-bit-security supersingular curves. Pages 397–416 of: *Pairing-based cryptography—Pairing 2010*. Lecture Notes in Computer Science, vol. 6487.

[21] Gardner, Martin. August, 1977. A new kind of cipher that would take millions of years to break. *Scientific American*, 120–124.

[22] Granville, A. 2005. It is easy to determine whether a given integer is prime. *Bull. Amer. Math. Soc. (N.S.)*, **42**, 3–38.

[23] Herstein, I. N. 1999. *Abstract Algebra*. Third edn. New York: John Wiley & Sons.

[24] Ireland, K., and Rosen, M. 1998. *A Classical Introduction to Modern Number Theory*. Berlin, New York: Springer-Verlag.

[25] Janusz, G. 1998. *Algebraic Number Fields*. Second edn. Providence, Rhode Island: Amer. Math. Soc.

[26] Joye, Marc, Lenstra, Arjen K., and Quisquater, Jean-Jacques. 1999. Chinese Remaindering Based Cryptosystems in the Presence of Faults. *Journal of Cryptology*, **12**(4), 241–245.

[27] Kleinjung, Thorsten, Bos, Joppe W., and Lenstra, Arjen K. 2014. *Mersenne factorization factory*. Cryptology ePrint Archive, Report 2014/653. `http://eprint.iacr.org/2014/653`.

[28] Koblitz, N. 1987. Elliptic curve cryptosystems. *Math. Comp.*, **48**, 203–209.

[29] Kraitchik, M. 1929. *Recherches sur la Théorie des Nombres*. Paris, France: Gauthiers-Villars.

[30] Lehmer, D. H., and Powers, R. E. 1931. On factoring large numbers. *Bull. Amer. Math. Soc.*, **37**, 770–776.

[31] Lehmer, D. N. 1909. *Factor table for the first ten millions containing the smallest factor of every number not divisible by 2, 3, 5, or 7 between the limits 0 and 10017000*. Carnegie Institute of Washington.

[32] Lenstra, A. K., and Lenstra, Jr., H. W. (eds). 1993. *The Development of the Number Field Sieve*. Lecture Notes in Mathematics, vol. 1554. New York: Springer-Verlag.

[33] Lenstra, Arjen K., and Manasse, Mark S. 1990. Factoring by Electronic Mail. Pages 355–371 of: Quisquater, Jean-Jacques, and Vandewalle, Joos (eds), *EUROCRYPT'89*. LNCS, vol. 434. Houthalen, Belgium: Springer, Heidelberg, Germany.

[34] Lenstra, Arjen K., and Manasse, Mark S. 1991. Factoring With Two Large Primes. Pages 72–82 of: Damgård, Ivan (ed), *EUROCRYPT'90*. LNCS, vol. 473. Aarhus, Denmark: Springer, Heidelberg, Germany.

[35] Lenstra, Arjen K., and Shamir, Adi. 2000. Analysis and Optimization of the

TWINKLE Factoring Device. Pages 35–52 of: Preneel, Bart (ed), *EUROCRYPT 2000*. LNCS, vol. 1807. Bruges, Belgium: Springer, Heidelberg, Germany.

[36] Lenstra, Arjen K., Lenstra, Hendrik W., and Lovász, László. 1982. Factoring polynomials with rational coefficients. *Mathematische Annalen*, **261**(4), 515–534.

[37] Lenstra, Arjen K., Lenstra Jr., Hendrik W., Manasse, Mark S., and Pollard, John M. 1990. The Number Field Sieve. Pages 564–572 of: *22nd ACM STOC*. Baltimore, MD, USA: ACM Press.

[38] Lenstra, Arjen K., Hughes, James P., Augier, Maxime, Bos, Joppe W., Kleinjung, Thorsten, and Wachter, Christophe. 2012. Public Keys. Pages 626–642 of: Safavi-Naini, Reihaneh, and Canetti, Ran (eds), *CRYPTO 2012*. LNCS, vol. 7417. Santa Barbara, CA, USA: Springer, Heidelberg, Germany.

[39] Lenstra, Jr., H. W. 1987. Factoring integers with elliptic curves. *Ann. of Math.*, **126**, 649–673.

[40] May, A. 2004. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. Pages 213–219 of: *Advances in Cryptology—Proc. CRYPTO 2004*. Lecture Notes in Computer Science, vol. 3152.

[41] Menezes, A. J., Okamoto, T., and Vanstone, S. A. 1993. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Trans. on Info. Theory*, **IT-39(5)**, 1639–1646.

[42] Miller, V. 1987. Use of elliptic curves in cryptography. Pages 417–426 of: *Advances in Cryptology—Proc. CRYPTO '85*. Lecture Notes in Computer Science, vol. 218.

[43] Montgomery, P. L., and Murphy, B. 1999. Improved polynomial selection for the Number Field Sieve. In: *The Mathematics of Public Key Cryptography Conference*. Toronto: Fields Institute.

[44] Morain, F. 2004. La primalité en temps polynomial (d'après Adleman, Huang; Agrawal, Kayal, Saxena). *Astérisque*, **294**, 205–230.

[45] Morrison, M. A., and Brillhart, J. 1975. A method of factoring and the factorization of F_7. *Math. Comp.*, **29**, 183–205.

[46] Pollard, J. M. 1974. Theorems on factorization and primality testing. *Proc. Cambridge Philos. Soc.*, **76**, 521–528.

[47] Pollard, J. M. 1975. A Monte Carlo method for factorization. *Nordisk Tidskr. Informationsbehandling (BIT)*, **15**, 331–335.

[48] Pollard, J. M. 1993. *Factoring with cubic integers*. Lecture Notes in Mathematics, vol. 1554. New York: Springer-Verlag. Pages 4–10.

[49] Pomerance, C. 1982. Analysis and comparison of some integer factoring algorithms. Pages 89–139 of: Lenstra, Jr., H. W., and Tijdeman, R. (eds), *Computational Methods in Number Theory, Part 1*. Math. Centrum Tract, vol. 154.

[50] Pomerance, C. 1994. The number field sieve. Pages 465–480 of: *Mathematics of Computation 1943–1993: a half-century of computational mathematics (Vancouver, BC, 1993)*. Proc. Sympos. Appl. Math., vol. 48.

[51] Pomerance, C. 2010. Primality testing: variations on a theme of Lucas. *Congressus Numerantium*, **201**, 301–312.

[52] Pomerance, C., Selfridge, J. L., and Wagstaff, Jr., S. S. 1980. The pseudoprimes to $25 \cdot 10^9$. *Math. Comp.*, **35**, 1003–1026.

[53] Pomerance, Carl, Smith, J. W., and Wagstaff, S. S. 1983. New Ideas for Factoring Large Integers. Pages 81–85 of: Chaum, David (ed), *CRYPTO'83*. Santa Barbara, CA, USA: Plenum Press, New York, USA.

[54] Rabin, M. 1979. *Digitized signatures and public-key functions as intractable as factoring*. Tech. rept. LCS/TR-212. M.I.T. Lab for Computer Science.

[55] Rivest, Ronald L., Shamir, Adi, and Adleman, Leonard M. 1978. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the Association for Computing Machinery*, **21**(2), 120–126.

[56] Schnorr, Claus-Peter. 1990. Factoring integers and computing discrete logarithms via Diophantine apprximation. Pages 171–182 of: *Advances in Computational Complexity Theory*.

[57] Schnorr, Claus-Peter. 1991. Factoring Integers and Computing Discrete Logarithms via Diophantine Approximations. Pages 281–293 of: Davies, Donald W. (ed), *EUROCRYPT'91*. LNCS, vol. 547. Brighton, UK: Springer, Heidelberg, Germany.

[58] Shamir, A. 1979. Factoring numbers in O($\log n$) arithmetic steps. *Inform. Proc. Lett.*, **8**, 28–31.

[59] Shamir, A. 1999. Factoring large numbers with the TWINKLE device (extended abstract). Pages 2–12 of: Koç, Ç., and Paar, C. (eds), *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES '99, Worcester, MA*. Lecture Notes in Computer Science, vol. 1717.

[60] Shanks, D. 1971. Class number, a theory of factorization, and genera. Pages 415–440 of: *1969 Number Theory Institute, Stony Brook, N.Y.* Proc. Sympos. Pure Math., vol. 20. Amer. Math. Soc.

[61] Shor, P. 1994. Algorithms for quantum computation: discrete logarithms and factoring. Pages 124–134 of: *Proceedings of the Thirty-Fifth Annual Symposium on the Foundations of Computer Science*.

[62] Silverman, R. D. 1987. The Multiple Polynomial Quadratic Sieve. *Math. Comp.*, **48**, 329–339.

[63] Silverman, R. D., and Wagstaff, Jr., S. S. 1993. A practical analysis of the elliptic curve factoring algorithm. *Math. Comp.*, **61**, 445–462.

[64] Smith, J. W., and Wagstaff, Jr., S. S. 1983. An extended precision operand computer. Pages 209–216 of: *Proc. of the Twenty-First Southeast Region ACM Conference*.

[65] Strassen, V. 1976/77. Einige Resultate über Berechnungskomplexität. *Jahresber. Deutsch. Math.-Verein.*, **78**, 1–8.

[66] Trappe, W., and Washington, L. C. 2002. *Introduction to Cryptography with Coding Theory*. Upper Saddle River, New Jersey: Prentice Hall.

[67] Wagstaff, Jr., S. S., and Smith, J. W. 1987. Methods of factoring large integers. Pages 281–303 of: Chudnovsky, D. V., Chudnovsky, G. V., Cohn, H., and Nathanson, M. B. (eds), *Number Theory, New York, 1984-1985*. Lecture Notes in Mathematics, vol. 1240.

[68] Wagstaff, Jr., S. S. 2013. *The Joy of Factoring*. Student Mathematical Library, vol. 68. Providence, Rhode Island: Amer. Math. Soc.

[69] Washington, L. C. 2003. *Elliptic Curves: Number Theory and Cryptography*. Boca Raton, Florida: Chapman & Hall/CRC Press.

[70] Wiener, M. J. 1990. Cryptanalysis of short RSA secret exponents. *IEEE Trans. on Info. Theory*, **36**, 553–558.

[71] Williams, Hugh C. 1980. A Modification of the RSA Public-key Encryption Procedure. *IEEE Transactions on Information Theory*, **26**(6), 726–729.

[72] Zimmermann, P., and Dodson, B. 2006. 20 years of ECM. Pages 525–542 of: *Algorithmic Number Theory, Proceedings ANTS 2006*. Lecture Notes in Computer Science, vol. 4076. Berlin: Springer.

Subject index

Abelian group, 22
Adleman, L. M., 8
Agrawal, M., 6, 36
Algebraic number, 18
Atkins, D., 9, 16

Baillie, R., 4, 6
Birthday paradox, 3, 14
Boneh, D., 26, 29, 30, 33
Bos, J. W., 20
Brent, R. P., 34
Brillhart, J., 13, 17, 34, 35
Browne, M. W., 16
Buhler, J. P., 20

Cataldi, P., 1
Chernac, L., 1
Chinese Remainder Theorem, 30
Clausen, T., 34
Complexity
  of AKS primality test, 6
  of BPSW primality test, 6
  of CFRAC, 13
  of Elliptic Curve Method, 25
  of Number Field Sieve, 21
  of Pollard Rho, 4
  of Quadratic Sieve, 16
Continued fraction, 13
Continued Fraction Factoring Algorithm, 13–15, 34, 35
Conway, J., 34
Coppersmith, D., 30, 33
Crandall, R., 17, 21
Cryptography, public-key, 6

DES, 7
Dickman, K., 25, 35
Difference of squares factoring algorithm, 2
Diffie, W., 7, 8
Digital signature, 8
Dixon, J. D., 27, 28
Dodson, B., 25
Durfee, G., 33

ECM, 23
ECM, second stage of, 25
Elliptic curve
  discrete logarithm problem, 26
  method, 23, 27, 34, 35
  pairing, 26
  supersingular, 26
Eratosthenes, 1
Estibals, N., 26
Euler, L., 2, 34

Factor base, 13
Fast Exponentiation Algorithm, 30
Fast Point Multiplication, 23
Fermat
  Factoring Method, 2
  Little Theorem, 4
  number, 4, 17, 34
Fermat, P., 2
Fibonacci number, 6
Franklin, M., 26
Full rank lattice, 31

Galois, E., 10
Gardner, M., 8
Gauss, C. F., 2
Graff, M., 9, 16
Guy, R., 34

Hasse
  interval, 24, 27
  Theorem, 23, 24
Hasse, H., 23

Hellman, M., 7, 8

Kayal, N., 6, 36
Kleinjung, T., 20
Koblitz, N., 22
Kraitchik, M., 13

Landry, F., 34
Las Vegas algorithm, 27
Lattice, 31
Law of Quadratic Reciprocity, 2
Legendre symbol, 16, 23
Lehmer, D. H., 5, 13
Lehmer, D. N., 1
Lenstra, A. K., 9, 16, 17, 20, 21, 29, 30, 32, 34
Lenstra, H. W., Jr., 17, 20, 21, 23, 24, 32, 35
Leyland, P. C., 9, 16
Lovasz, L., 32
$L(x)$, 13, 16, 25, 28

Manasse, M., 16, 17, 34
May, A., 30, 32
Menezes, A. J., 26
Miller, V., 22
Monte Carlo algorithm, 3
Montgomery, P. L., 20
Morrison, M. A., 13, 17, 34, 35
Murphy, B., 20

Number Field Sieve, 17, 34, 35

Okamoto, T., 26

Pollard Rho Method, 3, 34, 35
Pollard, J. M., 3, 4, 17, 27, 34, 35
Pollard $p - 1$ Method, 4, 21, 23–25, 35
Polynomial
  time, 12, 28, 29, 33, 36
Pomerance, C., 5, 6, 13, 16, 17, 20, 21, 35
Powers, R. E., 13
Primality testing, 5
Prime
  proving, 5
Probable prime, 5
Probable prime, strong, 6
Pseudoprime, 6, 29, 35
Pseudoprime, strong, 6, 29, 35
Public key cipher, 7–9

Quadratic
  form, 27
  polynomial, 15
  residue, 2, 9, 11, 13
  Sieve, 15, 17, 35
Quantum computer, 36

Rabin, M. O., 9
Riemann Hypothesis, Extended, 27
Rivest, R., 8
RSA cipher, 8, 28, 29, 31

Sassoon, G., 17
Saxena, N., 6, 36
Schnorr, C.-P., 35
Selfridge, J. L., 6
Shamir, A., 8, 21, 28, 36
Shanks, D., 27
Shor, P., 36
Sieve of Eratosthenes, 15
Silverman, R. D., 25
Size of an elliptic curve modulo p, 23
Smith, J. W., 14
Smooth integer, 11, 13, 25, 35
Strassen, V., 27

Trappe, W., 26
Trial Division, 1, 3, 13, 15, 27, 28, 34
TWINKLE, 21

Vanstone, S. A., 26

Wagstaff, S. S., Jr., 14, 25
Washington, L. C., 26
Weil pairing, 26
Wheel, 2
Wiener, M. J., 30
Williams, H. C., 9

Zimmermann, P., 25