Algorithms in number theory

A.K. Lenstra and H.W. Lenstra, Jr.
The University of Chicago

Technical Report 87-008, May 1987

UNIVERSITY OF CHICAGO
DEPARTMENT OF COMPUTER SCIENCE

Algorithms in number theory

A.K. Lenstra
Department of Computer Science
The University of Chicago
Chicago, IL 60637

H.W. Lenstra, Jr.
Department of Mathematics
University of California
Berkeley, CA 94720

Abstract. In this paper we discuss three closely related computational problems in number theory: computing discrete logarithms, factoring integers, and proving primality.

version 19880304.
Key words: discrete logarithm, integer factorization, primality testing, elliptic curves, class groups.
1980 Mathematics subject classification (1985): 11Axx, 11-01, 68Q25.
CR Categories: F.2.1.

Acknowledgment. The second author is supported by the National Science Foundation under Grant No. DMS-8706176.

To appear in: J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, D. Perrin (eds), *Handbook of theoretical computer science*, North-Holland, Amsterdam.

Introduction

Number theory, once believed to be the purest of sciences, has appeared to be widely applicable in computer science. Ironically, some of these applications are not based on achievements number theory may be proud of. Instead, they rely on the fact that certain number theoretical problems have not been solved satisfactorily. This would not be something to be dissatisfied with, if one could at least *prove* that those problems are intractable, but in this direction nothing is known.

A famous example is the RSA signature scheme [55], whose security depends on our inability to factor a composite integer efficiently. On the other hand, the RSA-scheme would not have been applicable without the existing fast methods to find large prime numbers.

In this chapter we will concentrate on what *can* be done in our attempt to solve three closely related computational problems in number theory. After a preparatory first section, we discuss in Section 2 the problem of computing discrete logarithms, a subject which can conveniently be phrased in terms of algorithms for finite abelian groups. This is also the case for many of the algorithms in Sections 3 and 4, which are devoted to the problem of determining the prime factorization of integers: algorithms for factoring integers in Section 3, and methods for proving primality in Section 4. Of these problems, only the problem in Section 4 is considered to be reasonably well solved. We will not present applications of the possible intractability of the other two problems; for this we refer to the contribution of R. Rivest to this volume.

We have restricted ourselves to the asymptotically fastest algorithms, which often represent the most recent developments. For those who are interested in a more historical treatment, we have attempted to give sufficiently many references to the extensive literature on these subjects.

Throughout this chapter *time* will mean *number of bit operations*.

1. Preliminaries

(1.1) Smoothness

In many of the algorithms that we will present the notion of *smoothness* will play an important role. We say that an integer is *smooth with respect to* y, or y*-smooth*, if all its prime factors are $\leq y$. In what follows, we will often be interested in the probability that a random integer between 1 and x is smooth with respect to some y.

To derive an expression for this probability, we define $\psi(x, y)$ as the number of positive integers $\leq x$ that are smooth with respect to y. Lower and upper bounds for $\psi(x, y)$ are known from [12] and [21]. Combination of these results yields the following. For a fixed arbitrary $\varepsilon > 0$, we have that for $x \geq 10$ and $u \leq (\log x)^{1-\varepsilon}$,

$$\psi(x, x^{1/u}) = x \cdot u^{-u+f(x, u)},$$

for a function f that satisfies $f(x, u)/u \to 0$ for $u \to \infty$ uniformly in x. For fixed $\alpha, \beta \in \mathbb{R}_{>0}$ we find that for $n \to \infty$

$$\psi(n^{\alpha}, n^{\beta\sqrt{(\log\log n)/\log n}}) = n^{\alpha} \cdot ((\alpha/\beta)\sqrt{\log n/\log\log n}\,)^{-(1+o(1))(\alpha/\beta)\sqrt{\log n/\log\log n}},$$

which can conveniently be written as

$$\psi(n^{\alpha}, L(n)^{\beta}) = n^{\alpha} \cdot L(n)^{-\alpha/(2\beta)+o(1)}$$

where $L(n) = e^{\sqrt{\log n \log\log n}}$. It follows that a random positive integer $\leq n^{\alpha}$ is smooth with respect to $L(n)^{\beta}$ with probability $L(n)^{-\alpha/(2\beta)+o(1)}$, for $n \to \infty$.

For $\beta \in \mathbb{R}$ we will often write $L_n[\beta]$ for $L(n)^{\beta}$, and we will abbreviate $L_n[\beta+o(1)]$ to $L_n[\beta]$, for $n \to \infty$. Notice that in this notation $L_n[\alpha]+L_n[\beta] = L_n[\max(\alpha, \beta)]$, and that the prime counting function π satisfies $\pi(L_n[\beta]) = L_n[\beta]$.

(1.2) Elliptic curves

We give an introduction to elliptic curves. For details and proofs we refer to [37,62]. Our presentation is by no means conventional, but reflects the way in which we apply elliptic curves.

Let p be a prime number and let $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$. The projective plane $\mathbb{P}^2(\mathbb{F}_p)$ over $\mathbb{F}_p$ consists of the equivalence classes of triples $(x, y, z) \in \mathbb{F}_p \times \mathbb{F}_p \times \mathbb{F}_p$, $(x, y, z) \neq 0$, where two triples (x, y, z) and (x', y', z') are equivalent if $cx = x'$, $cy = y'$, and $cz = z'$ for some $c \in \mathbb{F}_p^*$; the equivalence class containing (x, y, z) is denoted by $(x{:}y{:}z)$.

Now assume that p is unequal to 2 or 3. An *elliptic curve* over $\mathbb{F}_p$ is a pair $a, b \in \mathbb{F}_p$ for which $4a^3+27b^2 \neq 0$. These elements are to be thought of as the coefficients in the Weierstrass equation

(1.3) $\qquad y^2 = x^3+ax+b.$

An elliptic curve a, b is denoted by $E_{a,b}$, or simply by E.

(1.4) *Set of points of an elliptic curve.* Let E be an elliptic curve over $\mathbb{F}_p$. The *set of points* $E(\mathbb{F}_p)$ of E over $\mathbb{F}_p$ is defined by

$$E(\mathbb{F}_p) = \{(x{:}y{:}z) \in \mathbb{P}^2(\mathbb{F}_p)\colon y^2z = x^3+axz^2+bz^3\}.$$

There is one point $(x{:}y{:}z) \in E(\mathbb{F}_p)$ for which $z = 0$, namely the *zero point* (0:1:0), denoted by O. The other points of $E(\mathbb{F}_p)$ are the points $(x{:}y{:}1)$, where $x, y \in \mathbb{F}_p$ satisfy (1.3). The set $E(\mathbb{F}_p)$ has the structure of an *abelian group.* The group law, which we will write additively, is defined as follows.

(1.5) *The group law.* For any $P \in E(\mathbb{F}_p)$ we define $P+O = O+P = P$. For non-zero $P = (x_1{:}y_1{:}1)$, $Q = (x_2{:}y_2{:}1) \in E(\mathbb{F}_p)$ we define $P+Q = O$ if $x_1 = x_2$ and $y_1 = -y_2$. Otherwise, the sum $P+Q$ is defined as the point $(x{:}-y{:}1) \in E(\mathbb{F}_p)$ for which (x, y) satisfies (1.3) and lies on the line through (x_1, y_1) and (x_2, y_2); if $x_1 = x_2$ we take the tangent line to the curve in (x_1, y_1) instead. With $\lambda = (y_1-y_2)/(x_1-x_2)$ if $x_1 \neq x_2$, and $\lambda = (3x_1^2+a)/(2y_1)$ otherwise, we find that $x = \lambda^2-x_1-x_2$ and $y = \lambda(x-x_1)+y_1$. The proof that $E(\mathbb{F}_p)$ becomes an abelian group with this group law can be found in [62, Chapter 3].

(1.6) *The order of* $E(\mathbb{F}_p)$. The order $\#E(\mathbb{F}_p)$ of the abelian group $E(\mathbb{F}_p)$ equals $p+1-t$ for some integer t with $|t| \le 2\sqrt{p}$, a theorem due to Hasse (1934). Conversely, a result of Deuring [22] can be used to obtain an expression for the number of times a given integer of the above form $p+1-t$ occurs as $\#E(\mathbb{F}_p)$, for a fixed p, where E ranges over all elliptic curves over $\mathbb{F}_p$. This result implies that for any integer t with $|t| < 2\sqrt{p}$ there is an elliptic curve E over $\mathbb{F}_p$ for which $\#E(\mathbb{F}_p) = p+1-t$. A consequence of this result that will prove to be important for our purposes, is that $\#E(\mathbb{F}_p)$ is approximately uniformly distributed over the numbers near $p+1$ if E is uniformly distributed over all elliptic curves over $\mathbb{F}_p$.

(1.7) Proposition *(cf. [37, Proposition (1.16)]). There are positive effectively computable constants c_1 and c_2 such that for any prime number $p \ge 5$ and any set S of integers s for which $|s-(p+1)| < \sqrt{p}$ one has*

$$\frac{\#S-2}{2\lfloor\sqrt{p}\rfloor+1}\cdot c_1(\log p)^{-1} \le \frac{N}{p^2} \le \frac{\#S}{2\lfloor\sqrt{p}\rfloor+1}\cdot c_2(\log p)\cdot(\log\log p)^2,$$

where N denotes the number of pairs $a, b \in \mathbb{F}_p$ that define an elliptic curve $E = E_{a,b}$ over $\mathbb{F}_p$ with $\#E(\mathbb{F}_p) \in S$.

Because N/p^2 is the probability that a random pair a, b defines an elliptic curve E over $\mathbb{F}_p$ for which $\#E(\mathbb{F}_p) \in S$, this proposition asserts that this probability is essentially equal to the probability that a random integer near p is in S.

(1.8) *Computing the order of* $E(\mathbb{F}_p)$. For an elliptic curve E over $\mathbb{F}_p$ the number $\#E(\mathbb{F}_p)$ can be computed by means of the *division points method*, due to Schoof [59]. This method works by investigating the action of the Frobenius endomorphism on the l-division points of the curve, for various small prime numbers l. An l-division point is a point P over an extension of $\mathbb{F}_p$ for which $l \cdot P = O$, and the Frobenius endomorphism is the map sending $(x{:}y{:}z)$ to $(x^p{:}y^p{:}z^p)$. The division points method is completely deterministic, guaranteed to work if p is prime, and runs in $O((\log p)^8)$ bit operations (cf. [38]); with fast multiplication techniques this becomes $(\log p)^{5+o(1)}$. Its practical value is questionable, however.

Another method makes use of the *complex multiplication field*. The complex multiplication field L of an elliptic curve E with $\#E(\mathbb{F}_p) = p+1-t$ is defined as the imaginary quadratic field $\mathbb{Q}((t^2-4p)^{1/2})$ (cf. (1.6)). For certain special curves L is known; for instance for the curve $y^2 = x^3+4x$ and $p \equiv 1 \bmod 4$ we have $L = \mathbb{Q}(i)$, a fact that was already known to Gauss. Knowing L gives a fast way of computing $\#E(\mathbb{F}_p)$. Namely, suppose that L is known for some elliptic curve E, then the ring of integers A of L contains the zeros ρ, $\bar{\rho}$ of the polynomial X^2-tX+p, and $\#E(\mathbb{F}_p) = (\rho-1)(\bar{\rho}-1)$. Although this polynomial is not known, a zero can be determined by looking for an element π in A for which $\pi\bar{\pi} = p$ (see (4.12)). This π can be shown to be unique up to complex conjugation and units in A. For a suitable unit u in A we then have that $\rho \doteq u\pi$, so that $\#E(\mathbb{F}_p) = (u\pi-1)(\overline{u\pi}-1)$. In most cases A will have only two units, namely 1 and -1; only if $L = \mathbb{Q}(i)$ (or $L = \mathbb{Q}(\sqrt{-3})$) we have four (or six) units in A. In the case that A has only the units 1 and -1, an immediate method to decide whether $\#E(\mathbb{F}_p)$ equals $(\pi-1)(\bar{\pi}-1) = m'$ or $(-\pi-1)(-\bar{\pi}-1) = m''$ does not yet exist, as far as we know; in practice one could select a random point $P \in E(\mathbb{F}_p)$ such that not both $m' \cdot P$ and $m'' \cdot P$ are equal to O, so that $\#E(\mathbb{F}_p) = m$ for the unique $m \in \{m', m''\}$ for which $m \cdot P = O$. If A contains four or six units there exists a more direct method [28, Chapter 18].

In Section 4 we will use this method in the situation where L, A, and p are known; the elliptic curve E will then be *constructed* as a function of L and p.

(1.9) *Elliptic curves modulo n.* To motivate what follows, we briefly discuss elliptic curves modulo n, for a positive integer n. First we define what we mean by the projective plane $\mathbb{P}^2(\mathbb{Z}/n\mathbb{Z})$ over the ring $\mathbb{Z}/n\mathbb{Z}$. Consider the set of all triples $(x, y, z) \in (\mathbb{Z}/n\mathbb{Z})^3$ for which x, y, z generate the unit ideal of $\mathbb{Z}/n\mathbb{Z}$. The group of units $(\mathbb{Z}/n\mathbb{Z})^*$ acts on this set by $u(x, y, z) = (ux, uy, uz)$. The orbit of (x, y, z) under this action is denoted by $(x:y:z)$, and $\mathbb{P}^2(\mathbb{Z}/n\mathbb{Z})$ is the set of all orbits.

An elliptic curve $E = E_{a,b}$ modulo n is a pair $a, b \in \mathbb{Z}/n\mathbb{Z}$ for which $6(4a^3+27b^2) \in (\mathbb{Z}/n\mathbb{Z})^*$. It follows from (1.2) that for any prime p dividing n, the pair $\overline{a} = a \bmod p$, $\overline{b} = b \bmod p$ defines an elliptic curve $E_{\overline{a},\overline{b}}$ over $\mathbb{F}_p$. The set of points of this latter curve will be denoted by $E(\mathbb{F}_p)$.

The set of points $E(\mathbb{Z}/n\mathbb{Z})$ of E modulo n is defined by

$$E(\mathbb{Z}/n\mathbb{Z}) = \{(x:y:z) \in \mathbb{P}^2(\mathbb{Z}/n\mathbb{Z}): y^2z = x^3+axz^2+bz^3\}.$$

Clearly, for any $(x:y:z) \in E(\mathbb{Z}/n\mathbb{Z})$ and for any prime p dividing n, we have that $(x \bmod p : y \bmod p : z \bmod p) \in E(\mathbb{F}_p)$. It is possible to define a group law so that $E(\mathbb{Z}/n\mathbb{Z})$ becomes an abelian group, but we do not need this group structure for our purposes. Instead it suffices to define the following "pseudo-addition" on a subset of $E(\mathbb{Z}/n\mathbb{Z})$.

(1.10) *Partial addition algorithm.* Let $V_n \subset \mathbb{P}^2(\mathbb{Z}/n\mathbb{Z})$ consist of the elements $(x:y:1)$ of $\mathbb{P}^2(\mathbb{Z}/n\mathbb{Z})$ together with the zero element $(0:1:0)$, which will be denoted by O. For any $P \in V_n$ we define $P+O = O+P = P$. For non-zero $P = (x_1:y_1:1)$, $Q = (x_2:y_2:1) \in V_n$, and any $a \in \mathbb{Z}/n\mathbb{Z}$ we describe an addition algorithm that *either* finds a divisor d of n with $1 < d < n$, *or* determines an element of $R \in V_n$ that will be called the *sum* of P and Q:

(1) If $x_1 = x_2$ and $y_1 = -y_2$ put $R = O$ and stop.

(2) If $x_1 \neq x_2$, perform step (2a), otherwise perform step (2b).

(2a) Use the Euclidean algorithm to compute $s, t \in \mathbb{Z}/n\mathbb{Z}$ such that $s(x_1-x_2)+tn = \gcd(x_1-x_2, n)$. If this gcd is not equal to 1, call it d and stop. Otherwise put $\lambda = s(y_1-y_2)$, and proceed to step (3).

(2b) Use the Euclidean algorithm to compute $s, t \in \mathbb{Z}/n\mathbb{Z}$ such that $s(y_1+y_2)+tn = \gcd(y_1+y_2, n)$. If this gcd is not equal to 1, call it d and stop. Otherwise put $\lambda = s(3x_1^2+a)$, and proceed to step (3).

(3) Put $x = \lambda^2-x_1-x_2$, $y = \lambda(x-x_1)+y_1$, $R = (x:-y:1)$ and stop.

This finishes the description of the addition algorithm. Clearly the algorithm requires $O((\log n)^2)$ bit operations. Notice that this algorithm can be applied to any $P, Q \in V_n$, for any $a \in \mathbb{Z}/n\mathbb{Z}$, irrespective as to whether there exists $b \in \mathbb{Z}/n\mathbb{Z}$ such that a, b defines an elliptic curve modulo n.

(1.11) *Partial addition when taken modulo p.* Let p be any prime dividing n, and let P_p denote the point of $\mathbb{P}^2(\mathbb{F}_p)$ obtained from $P \in V_n$ by reducing its coordinates modulo p.

Assume that, for some $a \in \mathbb{Z}/n\mathbb{Z}$, and $P, Q \in V_n$ the algorithm in (1.10) has been successful in computing the sum $R = P+Q \in V_n$. Let $\overline{a}$ denote $a \bmod p$, and suppose that there exists an element $b \in \mathbb{F}_p$ such that $6(4\overline{a}^3+27b^2) \neq 0$ and such that $P_p, Q_p \in E_{\overline{a},b}(\mathbb{F}_p)$. It then follows from (1.5) and (1.10) that $R_p = P_p+Q_p$ in the group $E_{\overline{a},b}(\mathbb{F}_p)$.

Notice also that $P = O$ if and only if $P_p = O_p$, for $P \in V_n$.

(1.12) *Multiplication by a constant.* The algorithm in (1.10) allows us to multiply an element $P \in V_n$ by an integer $k \in \mathbb{Z}_{>0}$ in the following way. By repeated application of the addition algorithm in (1.10) for some $a \in \mathbb{Z}/n\mathbb{Z}$, we either find a divisor d of n with $1 < d < n$, or determine an element $R = k \cdot P \in V_n$ such that according to (1.11) the following holds: for any prime p dividing n for which there exists an element $b \in \mathbb{F}_p$ such that $6(4\overline{a}^3+27b^2) \neq 0$ and

$P_p \in E_{\bar{a},b}(\mathbb{F}_p)$, we have $R_p = k \cdot P_p$ in $E_{\bar{a},b}(\mathbb{F}_p)$, where $\bar{a} = a \bmod p$.

Notice that in the latter case $R_p = O_p$ if and only if the order of $P_p \in E_{\bar{a},b}(\mathbb{F}_p)$ divides k. But $R_p = O_p$ if and only if $R = O$, as we noted in (1.11), which is equivalent to $R_q = O_q$ for any prime q dividing n. We conclude that, if $k \cdot P$ has been computed successfully, and if q is another prime satisfying the same conditions as p above, then k is a multiple of the order of P_p if and only if k is a multiple of the order of P_q.

By repeated duplications and additions, multiplication by k can be done in $O(\log k)$ applications of algorithm (1.10), and therefore in $O((\log k)(\log n)^2)$ bit operations.

(1.13) *Randomly selecting curves and points.* In Section 2 we will be in the situation where we suspect that n is prime and have to select elliptic curves E modulo n and points in $E(\mathbb{Z}/n\mathbb{Z})$ at random. This can be accomplished as follows. Assume that $\gcd(n, 6) = 1$. Randomly select $a, b \in \mathbb{Z}/n\mathbb{Z}$ until $4a^3+27b^2 \neq 0$, and verify that $\gcd(n, 4a^3+27b^2) = 1$, as should be the case for prime n; per trial the probability of success is $(n-1)/n$, for n prime. The pair a, b now defines an elliptic curve modulo n, according to (1.9).

Given an elliptic curve $E = E_{a,b}$ modulo n, we randomly construct a point in $E(\mathbb{Z}/n\mathbb{Z})$. First, we randomly select an $x \in \mathbb{Z}/n\mathbb{Z}$ until x^3+ax+b is a square in $\mathbb{Z}/n\mathbb{Z}$. Because we suspect that n is prime, this can be done by checking whether $(x^3+ax+b)^{(n-1)/2} = 1$. Next, we determine y as a zero of the polynomial $X^2-(x^3+ax+b) \in (\mathbb{Z}/n\mathbb{Z})[X]$ using for instance the probabilistic method for finding roots of polynomials over finite fields described in [31, Section 4.6.2]. The resulting point $(x:y:1)$ is in $E(\mathbb{Z}/n\mathbb{Z})$.

For these algorithms to work, we do not need a proof that n is prime, but if n is prime, they run in expected time polynomial in $\log n$.

(1.14) Class groups

We review some results about class groups. For details and proofs we refer to [8,58]. A polynomial $aX^2+bXY+cY^2 \in \mathbb{Z}[X, Y]$ is called a *binary quadratic form*, and $\Delta = b^2-4ac$ is called its *discriminant*. We denote a binary quadratic form $aX^2+bXY+cY^2$ by (a, b, c). A form for which $a > 0$ and $\Delta < 0$ is called *positive*, and a form is *primitive* if $\gcd(a, b, c) = 1$. Two forms (a, b, c) and (a', b', c') are *equivalent* if there exist $\alpha, \beta, \gamma, \delta \in \mathbb{Z}$ with $\alpha\delta-\beta\gamma = 1$ such that $a'U^2+b'UV+c'V^2 = aX^2+bXY+cY^2$, where $U = \alpha X+\gamma Y$, and $V = \beta X+\delta Y$. Notice that two equivalent forms have the same discriminant.

Now fix some negative integer Δ with $\Delta \equiv 0$ or $1 \bmod 4$. We will often denote a form (a, b, c) of discriminant Δ by (a, b), since c is determined by $\Delta = b^2-4ac$. The set of equivalence classes of positive, primitive, binary quadratic forms of discriminant Δ is denoted by C_Δ. The existence of the form $(1, \Delta)$ shows that C_Δ is non-empty.

(1.15) *Reduction algorithm.* It has been proved by Gauss that each equivalence class in C_Δ contains precisely one *reduced form*, where a form (a, b, c) is reduced if

$$\begin{cases} |b| \le a \le c \\ b \ge 0 \text{ if } |b| = a \text{ or if } a = c. \end{cases}$$

These inequalities imply that $a \le \sqrt{|\Delta|/3}$; it follows that C_Δ is finite. For any form (a, b, c) of discriminant Δ we can easily find the reduced form equivalent to it by means of the following reduction algorithm:

(1) Replace (a, b) by $(a, b-2ka)$, where $k \in \mathbb{Z}$ is such that $-a < b-2ka \le a$.

(2) If (a, b, c) is reduced, then stop; otherwise, replace (a, b, c) by $(c, -b, a)$ and go back to step (1).

It is easily verified that this is a polynomial-time algorithm. Including the observation made in [31, exercise 4.5.2.30] in the analysis from [33], the reduction algorithm can be shown to take $O((\log a)^2)$ bit operations, where we assume that the initial b is already $O(a)$. It is not unlikely that with fast multiplication techniques one gets $O((\log a)^{1+\epsilon})$ by means of a method analogous to [57].

If the reduction algorithm applied to a form (a', b', c') yields the reduced form (a, b, c), then for any value $ax^2+bxy+cy^2$ a pair $u = \alpha x+\gamma y$, $v = \beta x+\delta y$ with $a'u^2+b'uv+c'v^2 = ax^2+bxy+cy^2$ can be computed if we keep track of a 2×2-transformation matrix in the algorithm. This does not affect the asymptotic running time of the reduction algorithm.

(1.16) *Composition algorithm.* The set C_Δ, which can now be identified with the set of reduced forms of discriminant Δ, is a finite abelian group, the *class group*. The group law, which we will write multiplicatively, is defined as follows. The inverse of (a, b) follows from an application of the reduction algorithm to $(a, -b)$, and the unit element 1_Δ is $(1, 1)$ for Δ odd, and $(1, 0)$ for Δ even. To compute $(a_1, b_1)\cdot(a_2, b_2)$, we use the Euclidean algorithm to determine $d = \gcd(a_1, a_2, (b_1+b_2)/2)$, and $r, s, t \in \mathbb{Z}$ such that $d = ra_1+sa_2+t(b_1+b_2)/2$. The product then follows from an application of the reduction algorithm to $(a_1a_2/d^2, b_2+2a_2(s(b_1-b_2)/2-tc_2)/d)$, where $c_2 = (b_2^2-\Delta)/(4a_2)$. It is again an easy matter to verify that this is a polynomial-time algorithm.

(1.17) *Ambiguous forms.* A reduced form is *ambiguous* if its square equals 1_Δ; for an ambiguous form we have $b = 0$, or $a = b$, or $a = c$. From now on we assume that $\Delta \equiv 1 \bmod 4$. It was already known to Gauss that for these Δ's there is a bijective correspondence between ambiguous forms and factorizations of $|\Delta|$ into two relatively prime factors. For relatively prime p and q the factorization $|\Delta| = pq$ corresponds to the ambiguous form (p, p) for $3p \leq q$, and to $((p+q)/4, (q-p)/2)$ for $p < q \leq 3p$. Notice that the ambiguous form $(1, 1)$ corresponds to the factorization $|\Delta| = 1\cdot|\Delta|$.

(1.18) *The class number.* The *class number* h_Δ of Δ is defined as the cardinality of the class group C_Δ. Efficient algorithms to compute the class number are not known. In [58] an algorithm is given that takes time $|\Delta|^{1/5+o(1)}$, for $\Delta \rightarrow -\infty$; both its running time and correctness depend on the assumption of the generalized Riemann hypothesis (GRH). It follows from the Brauer-Siegel theorem (cf. [35, Ch. XVI]) that $h_\Delta = |\Delta|^{1/2+o(1)}$ for $\Delta \rightarrow -\infty$. Furthermore, $h_\Delta < (\sqrt{|\Delta|}\log|\Delta|)/2$ for $\Delta < -3$. It follows from (1.17) that h_Δ is even if and only if $|\Delta|$ is not a prime power.

(1.19) *Finding ambiguous forms.* The ambiguous forms are obtained from forms whose order is a power of two. Namely, if (a, b) has order 2^k with $k > 0$, then $(a, b)^{2^{k-1}}$ is an ambiguous form. Because of the bound on h_Δ, we see that an ambiguous form can be computed in $O(\log|\Delta|)$ squarings, if a form (a, b) of 2-power order is given.

Such forms can be determined if we have an odd multiple u of the largest odd divisor of h_Δ, because for any form (c, d), the form $(c, d)^u$ is of 2-power order. Forms of 2-power order can therefore be determined by computing $(c, d)^u$ for randomly selected forms (c, d), or by letting (c, d) run through a set of generators for C_Δ; if in the latter case no (c, d) is found with $(c, d)^u \neq 1_\Delta$, then h_Δ is odd, so that Δ is a prime power according to (1.18).

(1.20) *Prime forms.* For a prime number p we define the Kronecker symbol $(\frac{\Delta}{p})$ by

$$(\frac{\Delta}{p}) = \begin{cases} 1 & \text{if } \Delta \text{ is a quadratic residue modulo } 4p \text{ and } \gcd(\Delta, p) = 1 \\ 0 & \text{if } \gcd(\Delta, p) \neq 1 \\ -1 & \text{otherwise.} \end{cases}$$

For a prime p for which $(\frac{\Delta}{p}) = 1$, we define the *prime form* I_p as the reduced form equivalent to (p, b_p), where $b_p = \min\{b \in \mathbb{Z}_{>0} : b^2 \equiv \Delta \bmod 4p\}$. It follows from a result in [34] that, if the generalized Riemann hypothesis holds, then there is an effectively computable constant

c, such that C_Δ is generated by the prime forms I_p with $p \le c \cdot (\log|\Delta|)^2$, where we only consider primes p for which $(\frac{\Delta}{p}) = 1$ (cf. [58, Cor. 6.2]); apparently no explicit value for the constant c has been published.

(1.21) *Smoothness of forms.* A form (a, b, c) of discriminant Δ, with $\gcd(a, \Delta) = 1$, for which the prime factorization of a is known, can be factored into prime forms in the following way. If $a = \prod_{p \text{ prime}} p^{e_p}$ is the prime factorization of a, then $(a, b) = \prod_{p \text{ prime}} I_p^{s_p e_p}$, where $s_p \in \{-1, +1\}$ satisfies $b \equiv s_p b_p \bmod 2p$, with b_p as in (1.20). Notice that the prime forms I_p are well-defined because the primes p divide a, $\gcd(a, \Delta) = 1$, and $b^2 \equiv \Delta \bmod 4a$.

We say that a form (a, b) is y-smooth if a is y-smooth. In [61] it has been proved that, under the assumption of the GRH, a random reduced form $(a, b) \in C_\Delta$ is $L_{|\Delta|}[\beta]$-smooth with probability at least $L_{|\Delta|}[-1/(4\beta)]$, for any $\beta \in \mathbb{R}_{>0}$. Since $a \le \sqrt{|\Delta|/3}$ this is what can be expected on the basis of (1.1); the GRH is needed because we only have the primes for which $(\frac{\Delta}{p}) = 1$.

(1.22) Solving systems of linear equations

Let A be an $n \times n$-matrix over a finite field, for some positive integer n, and let b be an n-dimensional vector over the same field. Suppose we want to solve the system $Ax = b$ over the field. It is well-known that this can be done by means of Gaussian elimination in $O(n^3)$ field operations. This number of operations can be improved to $O(n^{2.376})$ using the techniques from [19].

A more important improvement can be obtained if the matrix A is *sparse*, i.e., if the number of non-zero entries in A is very small. This will be the case in the applications below. There are several methods that take advantage of sparseness. For two of those algorithms, we refer to [18] and [44]. There it is shown that both the *conjugate gradient method* and the *Lanczos method*, methods that are known to be efficient for sparse systems over the real numbers, can be adapted to finite fields. These algorithms, which are due to D. Coppersmith, N. Karmarkar, and A.M. Odlyzko, achieve, for sparse systems, essentially the same running time as the method that we are going to present here.

(1.23) *The coordinate recurrence method.* This method is due to Wiedemann [67]. Assume that A is non-singular. Let F be the minimal polynomial of A on the vector space spanned by $b, Ab, A^2b, \ldots$. Because F has degree $\le n$ we have

$$F(A)b = \sum_{i=0}^{n} f_i A^i b = 0,$$

and for any $t \ge 0$,

$$\sum_{i=0}^{n} f_i A^{i+t} b = 0.$$

Let $v_{i\,j}$ be the jth coordinate of the vector $A^i b$, then

$$\sum_{i=0}^{n} f_i v_{i+t\,j} = 0 \tag{1.24}$$

for every $t \ge 0$ and $1 \le j \le n$. Fixing j, $1 \le j \le n$, we see that the sequence $(v_{i\,j})_{i=0}^{\infty}$ satisfies the linear recurrence relation (1.24) in the yet unknown coefficients f_i of F. Suppose we have computed $v_{i\,j}$ for $i = 0, 1, \ldots, 2n$ as the jth coordinate of $A^i b$. Given the first $2n+1$ terms $v_{0\,j}, v_{1\,j}, \ldots, v_{2n\,j}$ of the sequence satisfying a recurrence relation like (1.24), the minimal polynomial of the recurrence can be computed in $O(n^2)$ field operations by means of the Berlekamp-Massey algorithm [40]; denote by F_j this minimal polynomial. Clearly F_j divides F.

If we compute F_j for several values of j, it is not unlikely that F is the least common multiple of the F_j's. We expect that a small number of F_j's, say 20, suffice for this purpose (cf. [44,67]). Suppose we have computed F in this way. Because of the non-singularity of A we have $f_0 \neq 0$, so that

$$(1.25) \qquad x = -f_0^{-1} \sum_{i=1}^{n} f_i A^{i-1} b$$

satisfies $Ax = b$.

To analyse the running time of this algorithm for a sparse matrix A, let $w(A)$ denote the number of field operations needed to multiply A by a vector. The vectors $A^i b$ for $i = 0, 1, \ldots, 2n$ can then be computed in $O(nw(A))$ field operations. The same estimate holds for the computation of x. Because we expect that we need only a few F_j's to compute F, the applications of the Berlekamp-Massey algorithm take $O(n^2)$ field operations. The method requires storage for $O(n^2)$ field elements. At the cost of recomputing the $A^i b$ in (1.25), this can be improved to $O(n)+w(A)$ field elements, if we store only those coordinates of the $A^i b$ that we need to compute the F_j's. For a rigorous proof of these timings and a deterministic version of this probabilistic algorithm we refer to [67]. How the singular case should be handled, can be found in [67] and [44].

(1.26) *Solving equations over the ring* $\mathbb{Z}/m\mathbb{Z}$. In the sequel we often have to solve a system of equations over the ring $\mathbb{Z}/m\mathbb{Z}$, where m is not necessarily prime. We briefly sketch how this can be done using Wiedemann's coordinate recurrence method. Instead of solving the system over $\mathbb{Z}/m\mathbb{Z}$, we solve the system over the fields $\mathbb{Z}/p\mathbb{Z}$ for the primes $p \mid m$, lift the solutions to the rings $\mathbb{Z}/p^k\mathbb{Z}$ for the prime powers $p^k \mid m$, and finally combine these solutions to the solution over $\mathbb{Z}/m\mathbb{Z}$ by means of the Chinese remainder algorithm. In practice we will not try to obtain a complete factorization of m, but we just start solving the system modulo m, and continue until we try to divide by a zero-divisor, in which case a factor of m is found.

Lifting a solution $Ax_0 = b$ modulo p to a solution modulo p^k can be done by writing $Ax_0-b = py$ for some integer vector y, and solving $Ax_1 = y$ modulo p. It follows that $A(x_0-px_1) = b$ modulo p^2. This process is repeated until the solution modulo p^k is determined. We conclude that a system over $\mathbb{Z}/m\mathbb{Z}$ can be solved by $O(\log m)$ applications of algorithm (1.23).

2. Algorithms for finite abelian groups

(2.1) Introduction

Let G be a finite abelian group whose elements can be represented in such a way that the group operations can be performed efficiently. In this section we are interested in two computational problems concerning G: finding the order of G or of one of its elements, and computing discrete logarithms in G. For the latter problem we will often assume that the order n of G, or a small multiple of n, is known.

By computing discrete logarithms we mean the following. Let H be the subgroup of G generated by an element $h \in G$. For an element y of G, the problem of computing the *discrete logarithm* $\log_h y$ *of* y *with respect to* h, is the problem to decide whether $y \in H$, and if so, to compute an integer m such that $h^m = y$; in the latter case we write $\log_h y = m$. Evidently, $\log_h y$ is only defined modulo the order of H. Because the order of H is an unknown divisor of n, we will regard $\log_h y$ as a not necessarily well-defined integer modulo n, and represent it by an integer in $\{0, 1, \ldots, n-1\}$. Although $\log_h y$ is often referred to as the *index of* y *with respect to* h, we will only refer to it as the discrete logarithm, or logarithm, of y.

Examples of groups we are interested in are: multiplicative groups of finite fields, sets of points of elliptic curves modulo primes (cf. (1.2)), class groups (cf. (1.14)), and multiplicative groups modulo composite integers. Notice that in the first example n is known, and that for the second example two methods to compute n were mentioned in (1.8).

In all examples above, the group elements can be represented in a unique way. Equality of two elements can therefore be tested efficiently, and membership of a sorted list of cardinality k can be decided in $\log k$ comparisons. Examples where unique representations do *not* exist, are for instance multiplicative groups modulo an *unspecified* prime divisor of an integer n, or sets of points of an elliptic curve modulo n, when taken modulo an *unspecified* prime divisor of n (cf. (1.9)). In these examples *inequality* can be tested by means of a gcd-computation. If two non-identically represented elements are equal, the gcd will be a non-trivial divisor of n. In Section 3 we will see how this can be exploited.

In Section (2.2) we present some algorithms for both our problems that can be applied to any group G as above. By their general nature they are quite slow; the number of group operations required is an exponential function of $\log n$. Algorithms for groups with *smooth order* are given in Section (2.6). For groups containing many *smooth elements* subexponential discrete logarithm algorithms are given in Section (2.8). Almost all of the algorithms in Section (2.8) are only applicable to the case where G is the multiplicative group of a finite field, with the added restriction that h is a primitive root of the same field. In that case $G = H$, so that the decision problem becomes trivial. An application of these techniques to class groups is presented in Remark (2.15).

For practical consequences of the algorithms in this section we refer to the original papers and to [44].

(2.2) Exponential algorithms

Let G be a finite abelian group as in (2.1), let $h \in G$ be a generator of a subgroup H of G, and let $y \in G$. In this section we discuss three algorithms to compute $\log_h y$. The algorithms have in common that, with the proper choice for y, they can easily be adapted to compute the order n_h of h, or a small multiple of n_h.

Of course, $\log_h y$ can be computed deterministically in at most n_h multiplications and comparisons in G, by computing h^i for $i = 1, 2, \ldots$ until $h^i = y$ or $h^i = 1$; here 1 denotes the unit element in G. Then $y \in H$ if and only if $h^i = y$ for some i, and if $y \notin H$ the algorithm terminates after $O(n_h)$ operations in G; in the latter case (and if $y = 1$) the order of h has been computed. The method requires storage for only a constant number of group elements.

(2.3) *Shanks's baby-step-giant-step algorithm (cf. [32, exercise 5.17]).* We can improve on the number of operations of the above algorithm if we allow for more storage being used, and if a unique representation of the group elements exists. The algorithm is based on the following observation. If $y \in H$ and $\log_h y < s^2$ for some $s \in \mathbb{Z}_{>0}$, then there exist integers i and j with $0 \le i, j < s$ such that $y = h^{is+j}$. In this situation $\log_h y$ can be computed as follows. First, make a sorted list of the values h^j for $0 \le j < s$ in $O(s \log s)$ operations in G. Next, compute yh^{-is} for $i = 0, 1, \ldots, s-1$ until yh^{-is} equals one of the values in the list; this search can be done in $O(\log s)$ comparisons per i because the list is sorted. If yh^{-is} is found to be equal to h^j, then $\log_h y = is+j$. Otherwise, if yh^{-is} is not found in the list for any of the values of i, then either $y \notin H$ or $\log_h y \ge s^2$.

This method can be turned into a method that can be guaranteed to use $O(\sqrt{n_h} \log n_h)$ operations in G, both to compute discrete logarithms and to compute n_h. For the latter problem, we put $y = 1$, and apply the above method with $s = 2^k$ for $k = 1, 2, \ldots$ in succession, excluding the case where both i and j are zero. After $O(\sum_{k=1}^{\lceil \log_2 n_h \rceil /2} 2^k \log 2^k) = O(\sqrt{n_h} \log n_h)$ operations in G, we find i and j such that $h^{i2^k+j} = 1$, and therefore a small multiple of n_h. To compute $\log_h y$ we proceed similarly, but to guarantee a timely termination of the algorithm in case $y \notin H$, we look for h^{-is} in the list as well; if some h^{-is} is in the list, but none of the yh^{-is} is, then $y \notin H$. We could also first determine n_h, and put $s = \lceil \sqrt{n_h} \rceil$.

We conclude that both the order of h and discrete logarithms with respect to h can be computed deterministically in $n_h^{1/2+o(1)}$ multiplications and comparisons in G, for $n_h \to \infty$. The method requires storage for $O(\sqrt{n_h})$ group elements. In practice it can be recommended to use hashing (cf. [32, Section 6.4]) instead of sorting.

(2.4) *Pollard's Rho method (cf. [48]).* The following randomized method needs only a constant amount of storage. It is randomized in the sense that we cannot give a worst-case upper bound for its running time. We can only say that the *expected* number of group operations to be performed is $O(\sqrt{n})$ to compute discrete logarithms, and $O(\sqrt{n_h})$ to compute the order n_h of h; here n is the order of G. Let us concentrate on computing discrete logarithms first.

Assume that a number n is known that equals the order of G, or a small multiple thereof. We randomly partition G into three subsets G_1, G_2, and G_3, of approximately the same size. By an *operation* in G we mean either a group operation, or a membership test $x \overset{?}{\in} G_i$. For $y \in G$ we define the sequence $y_0, y_1, y_2, \ldots$ in G by $y_0 = y$, and

$$(2.5) \qquad y_i = \begin{cases} h \cdot y_{i-1} & \text{if } y_{i-1} \in G_1 \\ y_{i-1}^2 & \text{if } y_{i-1} \in G_2 \\ y \cdot y_{i-1} & \text{if } y_{i-1} \in G_3, \end{cases}$$

for $i > 0$. If this sequence behaves as a random mapping from G to G, its expected cycle length is $O(\sqrt{n})$ [31, exercise 4.5.4.4]. Therefore, when comparing y_i and y_{2i} for $i = 1, 2, \ldots$, we expect to find $y_k = y_{2k}$ for $k = O(\sqrt{n})$. The sequence has been defined in such a way that $y_k = y_{2k}$ easily yields $y^{e_k} = h^{m_k}$ for certain $e_k, m_k \in \{0, 1, \ldots, n-1\}$. Using the extended Euclidean algorithm we compute s and t such that $s \cdot e_k + t \cdot n = d$ where $d = \gcd(e_k, n)$; if $d = 1$, we find $\log_h y = s \cdot m_k \bmod n$.

Otherwise, if $d > 1$, we can proceed as follows. We introduce a number $l > 0$, to be thought of as the smallest known multiple of n_h. Initially we put $l = n$. Every time that l is changed, we check that $y^l = 1$ (if $y^l \neq 1$ then clearly $y \notin H$), and we compute new s, t, and d with $d = \gcd(e_k, l) = s \cdot e_k + t \cdot l$. Note that $h^{lm_k/d} = y^{le_k/d} = 1$, so that $n_h \mid lm_k/d$. If d does not divide m_k, then change l to $\gcd(l, lm_k/d)$. Ultimately, d divides m_k. We have that $y^d = h^{sm_k}$, so we may stop if $d = 1$. Otherwise, we determine the order d' of $h^{l/d}$ by means of any of the methods described in sections (2.2) and (2.6). If this is difficult to do then d is large (which is unlikely), and it is probably best to generate another relation of the sort $y^{e_k} = h^{m_k}$. If $d' < d$ then change l to ld'/d. Finally, suppose that $d' = d$. Let $y' = yh^{-sm_k/d}$, then $y \in H$ if and only if $y' \in H$, and since $(y')^d = 1$, this is the case if and only if y' belongs to the subgroup generated by $h' = h^{l/d}$. The problem with y and h is now reduced to the same problem with y' and h', with the added knowledge that the order of h' equals d. The new problem can be solved by means of any of the methods described in sections (2.2) and (2.6).

Of course, we could define the recurrence relation (2.5) in various other ways, as long as the resulting sequence satisfies our requirements.

Notice that, if $y \in H$, the recurrence relation (2.5) is defined over H. If also the $G_i \cap H$ are such that the sequence behaves as a random mapping from H to H, then we expect the discrete logarithm algorithm to run in $O(\sqrt{n_h})$ operations in G. A multiple of n_h can be computed in a similar way in about $O(\sqrt{n_h})$ operations in G. To do this one partitions G into a somewhat larger number of subsets G_j, say 20, and one defines $y_0 = 1$, and $y_i = h^{t_j} \cdot y_{i-1}$ if $y_{i-1} \in G_j$; here the numbers t_j are randomly chosen from $\{2, 3, \ldots, B-1\}$, where B is an estimate for n_h (cf. [56]).

We conclude this section by mentioning another randomized algorithm for computing discrete logarithms, the so-called *Lambda method for catching kangaroos*, also due to Pollard [48]. It can only be used when $\log_h y$ is *known* to exist, and lies in a specified interval of width w. The method requires $O(\sqrt{w})$ operations in G, and a small amount of storage (depending on the implementation), but cannot be guaranteed to have success; the failure probability ε, however, can be made arbitrarily small, at the cost of increasing the running time which depends linearly on $\sqrt{\log(1/\varepsilon)}$. We will not pursue this approach further, but refer the interested reader to [48]. Notice that, with $w = n$, this method can be used instead of the Rho method described above, if at least $y \in H$.

(2.6) Groups with smooth order

In some cases one might suspect that the order of G, or of h, has only small prime factors, i.e., is s-smooth for some small $s \in \mathbb{Z}_{>0}$. If one also knows an upper bound B on the order, this smoothness can easily be tested. Namely, in these circumstances the order should divide $k = \prod p^{t_p}$, where p ranges over the primes $\le s$, and where $t_p \in \mathbb{Z}_{\ge 0}$ is maximal such that $p^{t_p} \le B$. Raising h to the kth power should yield the unit element in G; this takes $O(s \log_s B)$ multiplications in G to verify. If h^k indeed equals the unit element, the order of h can be deduced after some additional computations.

(2.7) *The Chinese remainder theorem method (cf. [46]).* Also for the discrete logarithm problem a smooth order is helpful, as was first noticed by R. Silver, and later by Pohlig and Hellman. Let $n_h = \prod_{p \mid n_h} p^{e_p}$ be the prime factorization of n_h. If $y \in H$, then it suffices to determine $\log_h y = m$ modulo each of the p^{e_p}, followed by an application of the Chinese remainder algorithm.

To compute m modulo p^e, where p is one of the primes dividing n_h and $e = e_p$, we proceed as follows. Write $m \equiv \sum_{i=0}^{e-1} m_i p^i$ modulo p^e, with $m_i \in \{0, 1, ..., p-1\}$, and notice that

$$(m-(m \bmod p^i))n_h/p^{i+1} \equiv (n_h/p)m_i \text{ modulo } n_h$$

for $i = 0, 1, ..., e-1$. This implies that, if $y \in H$, then

$$(y \cdot h^{-(m \bmod p^i)})^{n_h/p^{i+1}} = (h^{n_h/p})^{m_i}.$$

Because $\bar{h} = h^{n_h/p}$ generates a cyclic subgroup $\bar{H}$ of G of order p, we can compute $m_0, m_1, ..., m_{e-1}$ in succession by computing the discrete logarithms of $\bar{y}_i = (y \cdot h^{-(m \bmod p^i)})^{n_h/p^{i+1}}$ with respect to $\bar{h}$, for $i = 0, 1, ..., e-1$. This can be done by means of any of the methods mentioned in Section (2.2). If $\bar{y}_i \notin \bar{H}$ for some i, then $y \notin H$, and the algorithm terminates. We leave the analysis of running time and storage requirements to the reader.

(2.8) Subexponential algorithms

In this section we will concentrate on algorithms to compute discrete logarithms with respect to a primitive root g of the multiplicative group G of a finite field. In this case the order of G is known. In principle the methods to be presented here can be applied to any group for which the concept of smoothness makes sense, and that contains sufficiently many smooth elements. This is the case for instance for class groups, as is shown in Remark (2.15).

We do not address the problem of finding a primitive root of G, or deciding whether a given element is a primitive root. Notice however that the latter can easily be accomplished if the factorization of the order of G is known. It would be interesting to analyse how the algorithms in this section behave in the case where it is not known whether g is a primitive root or not.

A rigorous analysis of the expected running time has only been given for a slightly different version of the first algorithm below [51]. The timings of the other algorithms in this section are heuristic estimates.

(2.9) Remark. Any algorithm that computes discrete logarithms with respect to a primitive root of a finite field, can be used to compute logarithms with respect to any non-zero element of the field. Let g be a primitive root of a finite field, G the multiplicative group of order n of the field, and h and y any two elements of G. To decide whether $y \in \langle h \rangle = H$ and, if so, to compute $\log_h y$, we proceed as follows. Compute $\log_g h = m_h$, $\log_g y = m_y$, and $\text{ind}(h) = \gcd(n, m_h)$. Then $y \in H$ if and only if $\text{ind}(h)$ divides m_y, and if $y \in H$ then $\log_h y = (m_y/\text{ind}(h))(m_h/\text{ind}(h))^{-1} \bmod n_h$, where $n_h = n/\text{ind}(h)$ is the order of h.

(2.10) *Smoothness in* $(\mathbb{Z}/p\mathbb{Z})^*$. If $G = (\mathbb{Z}/p\mathbb{Z})^*$, for some prime p, we identify G with the set $\{1, 2, ..., p-1\}$ of least positive residues modulo p; the order n of G equals $p-1$. It follows from (1.1) that a randomly selected element of G that is $\leq n^\alpha$ is $L_n[\beta]$-smooth with probability $L_n[-\alpha/(2\beta)]$, for $\alpha, \beta \in \mathbb{R}_{>0}$ fixed with $\alpha \leq 1$, and $n \to \infty$. The number of primes $\leq L_n[\beta]$ is $\pi(L_n[\beta]) = L_n[\beta]$. In Section 3 we will see that an element of G can be tested for $L_n[\beta]$-smoothness in expected time $L_n[0]$; in case of smoothness the complete factorization is computed at the same time (cf. (3.5)).

(2.11) *Smoothness in* $\mathbb{F}_{2^m}^*$. If $G = \mathbb{F}_{2^m}^*$, for some positive integer m, we select an irreducible polynomial $f \in \mathbb{F}_2[X]$ of degree m, so that $\mathbb{F}_{2^m} \cong (\mathbb{F}_2[X])/(f)$. The elements of G are then identified with non-zero polynomials in $\mathbb{F}_2[X]$ of degree $< m$. We define the *norm* $\text{N}(h)$ of an element $h \in G$ as $\text{N}(h) = 2^{\text{degree}(h)}$. Remark that $\text{N}(f) = \#\mathbb{F}_{2^m}$, and that the order n of G equals 2^m-1.

A polynomial in $\mathbb{F}_2[X]$ is *smooth with respect to* x, for some $x \in \mathbb{R}_{>0}$, if it factors as a product of irreducible polynomials of norm $\leq x$. It follows from a theorem of Odlyzko [44] that a random element of G of norm $\leq n^\alpha$ is $L_n[\beta]$-smooth with probability $L_n[-\alpha/(2\beta)]$, for $\alpha, \beta \in \mathbb{R}_{>0}$ fixed with $\alpha < 1$, and $n \to \infty$. Furthermore, an element of G of degree k can be factored in time polynomial in k (cf. [31]). The number of irreducible polynomials of norm $\leq L_n[\beta]$ is about $L_n[\beta]/\log_2(L_n[\beta]) = L_n[\beta]$.

These results can all easily be generalized to finite fields of arbitrary, but fixed, characteristic.

(2.12) *The index-calculus algorithm.* Let g be a generator of a group G of order n as in (2.10) or (2.11); 'prime element' will mean 'prime number' (2.10) or 'irreducible polynomial' (2.11), and for $G = (\mathbb{Z}/p\mathbb{Z})^*$ the 'norm' of $x \in G$ will be x itself. Let $y \in G$, and let S be the set of prime elements of norm $\leq L_n[\beta]$ for some $\beta \in \mathbb{R}_{>0}$. We abbreviate $L_n[\beta]$ to $L[\beta]$. The algorithms to compute $\log_g y$ that we present in this section consist of two stages (cf. [66]):

1 - Precomputation: compute $\log_g s$ for all $s \in S$;

2 - Computation of $\log_g y$: find a multiplicative relation between y and the elements of S, and derive $\log_g y$ using the result from the precomputation stage.

This gives rise to an algorithm whose expected running time is bounded by a polynomial function of $L(n)$; notice that this is better than $O(n^\varepsilon)$ for every $\varepsilon > 0$ (cf. [1]).

First, we will describe the second stage in more detail, and analyse its expected running time. Suppose that the discrete logarithms of the prime elements of norm $\leq L[\beta]$ all have been computed in the first stage. We determine an integer e such that $y \cdot g^e$ factors as a product of elements of S, by randomly selecting integers $e \in \{0, 1, ..., n-1\}$ until $y \cdot g^e \in G$ is smooth with respect to $L[\beta]$. For the resulting e we have

$$y \cdot g^e = \prod_{s \in S} s^{e_s},$$

so that

$$\log_g y = ((\sum_{s \in S} e_s \log_g s) - e) \bmod n,$$

where the $\log_g s$ are known from the precomputation stage. By the results cited in (2.10) and (2.11) we expect that $L[1/(2\beta)]$ trials suffice to find e. Because the time per trial is bounded by $L[0]$ for both types of groups, we expect to spend time $L[1/(2\beta)]$ for each discrete logarithm.

Now consider the precomputation stage, the computation of $\log_g s$ for all $s \in S$. We collect multiplicative relations between the elements of S, i.e., linear equations in the $\log_g s$. Once we have sufficiently many relations, we can compute the $\log_g s$ by solving a system of linear equations.

Collecting multiplicative relations can be done by randomly selecting integers $e \in \{0, 1, ..., n-1\}$ until $g^e \in G$ is smooth with respect to $L[\beta]$. For a successful e we have

$$g^e = \prod_{s \in S} s^{e_s}$$

which yields the linear equation

(2.13) $$e = \Big(\sum_{s \in S} e_s \log_g s\Big) \bmod n.$$

We need about $|S| \approx L[\beta]$ equations of the form (2.13) to be able to solve the resulting system of linear equations, so we repeat this step about $L[\beta]$ times.

It follows from the analysis of the second stage that collecting equations can be done in expected time $L[\beta+1/(2\beta)]$. Because the system can be solved in time $L[3\beta]$ by ordinary Gaussian elimination (cf. (1.22) and (1.26)), the precomputation stage takes expected time $L[\max(\beta+1/(2\beta), 3\beta)]$, which is $L[3/2]$ for the optimal choice $\beta = 1/2$. This dominates the cost of the second stage which takes, for $\beta = 1/2$, time $L[1]$ per logarithm. The storage requirements are $L[1]$ for the precomputation (to store the system of equations), and $L[1/2]$ for the second stage (to store the $\log_g s$ for $s \in S$).

An important improvement can be obtained by noticing that in the equations of the form (2.13) at most $\log_2 n$ of the $|S| \approx L[\beta]$ coefficients e_s can be nonzero. This implies that we can use the coordinate recurrence method described in (1.23), which has, combined with (1.26), the following consequence. Multiplying the matrix defining the system by a vector can be done in time $(\log_2 n)L[\beta]$, which is $L[\beta]$. The system can therefore be solved in time $L[2\beta]$, so that the expected time for the precomputation stage becomes $L[\max(\beta+1/(2\beta), 2\beta)]$. For $\beta = \sqrt{1/2}$, we get $L[\sqrt{2}]$ arithmetic operations in G or $\mathbb{Z}/n\mathbb{Z}$ for the precomputation, and $L[\sqrt{1/2}]$ operations per logarithm. The method requires storage for $L[\sqrt{1/2}]$ group elements both in the precomputation and in the second stage. We refer to [51] for a rigorous proof that a slightly modified version of the index-calculus algorithm runs in time $L[\sqrt{1/2}]$, for both our choices of G.

(2.14) Remark. As suggested at the end of (2.11), the algorithm in (2.12), and the modifications presented below, can be adapted to finite fields of arbitrary, but fixed, characteristic. For $\mathbb{F}_{p^2}$ a modified version of the index calculus algorithm is presented in [24]; according to Odlyzko [44] this method applies to $\mathbb{F}_{p^m}$, for fixed m, as well. It is an as yet unanswered and interesting question how to compute discrete logarithms when *both p and m* tend to infinity.

(2.15) Remark. The ideas from the index calculus algorithm can be applied to other groups as well. Consider for instance the case that G is a class group as in (1.14), of unknown order n. Suppose we want to compute the discrete logarithm of y with respect to h, for $h, y \in G$. Let S be a set of prime forms that generates G (cf. (1.20)). The mapping ϕ from $\mathbb{Z}^S$ to G that maps $(e_s)_{s \in S} \in \mathbb{Z}^S$ to $\prod_{s \in S} s^{e_s} \in G$ is a surjection. The kernel of ϕ is a sublattice of the lattice $\mathbb{Z}^S$, and $\mathbb{Z}^S/\ker(\phi) \cong G$. In particular the determinant of $\ker(\phi)$ equals n.

To calculate $\ker(\phi)$, we introduce a subgroup Λ of $\mathbb{Z}^S$, to be thought of as the largest subgroup of $\ker(\phi)$ that is known. Initially one puts $\Lambda = \{0\}$. To enlarge Λ, one looks for relations between the elements of S. Such relations can be found in a way similar to the precomputation stage of (2.12), as described in (3.14); the primitive root g is replaced by a product of random powers of elements of S, thus producing a random group element. Every relation gives rise to an element $r \in \ker(\phi)$. One tests whether $r \in \Lambda$, and if not one replaces Λ by $\Lambda+\mathbb{Z}r$; if Λ is given by a basis in Hermite form, this can be done by means of the algorithm of [30]. Repeating this a number of times, one may expect to find a lattice Λ containing $|S|$ independent vectors. The determinant of Λ is then a non-zero multiple of n. After some additional steps it will happen that Λ does not change anymore, so that one may hope that $\Lambda = \ker(\phi)$. In that case $\det(\Lambda) = n$, and $\mathbb{Z}^S/\Lambda \cong G$.

Supposing that $\Lambda = \ker(\phi)$, we can write G as a direct sum of cyclic groups, by bringing the matrix defining Λ to diagonal form [30]. This may change the set of generators of G. To solve the discrete logarithm problem one expresses both h and y as products of powers of the new generators, and applies (2.9) repeatedly. Notice that if the assumption $\Lambda = \ker(\phi)$ is wrong (i.e., we did not find sufficiently many relations), we may incorrectly decide that $y \notin <h>$.

(2.16) *A method based on the residue-list sieve from [18].* We now discuss a variant of the index-calculus algorithm that yields a better heuristic running time. Instead of looking for random smooth group elements that yield equations like (2.13), we look for smooth elements of much smaller norm, that still produce the necessary equations. Because elements of smaller norm have a higher probability of being smooth, we expect that this will give a faster algorithm.

For ease of exposition we take $G = (\mathbb{Z}/p\,\mathbb{Z})^*$, as in (2.10), so that $n = p-1$. Let the notation be as in (2.12). Linear equations in the $\log_g s$ for $s \in S$ are collected as follows. Let $\alpha \in \mathbb{R}_{>0}$ and let u and v be two integers in $\{\lfloor\sqrt{p}\rfloor+1, ..., \lfloor\sqrt{p}+L[\alpha]\rfloor\}$, both smooth with respect to $L[\beta]$. If $uv-p$ is also smooth with respect to $L[\beta]$, then we have found an equation of the type we were looking for, because $\log_g u + \log_g v = \log_g(uv-p)$.

We analyse how much time it takes to collect $L[\beta]$ equations in this way. The probability of $uv-p = O(L[\alpha]\sqrt{p})$ being smooth with respect to $L[\beta]$ is $L[-1/(4\beta)]$, so we have to consider $L[\beta+1/(4\beta)]$ smooth pairs (u, v), and test the corresponding $uv-p$ for smoothness. This takes time $L[\beta+1/(4\beta)]$. It follows that we need $L[\beta/2+1/(8\beta)]$ integers $u \in \{\lfloor\sqrt{p}\rfloor+1, ..., \lfloor\sqrt{p}+L[\alpha]\rfloor\}$ that are smooth with respect to $L[\beta]$. For that purpose we take $L[\beta/2+1/(8\beta)+1/(4\beta)]$ integers in $\{\lfloor\sqrt{p}\rfloor+1, ..., \lfloor\sqrt{p}+L[\alpha]\rfloor\}$ and test them for smoothness, because the probability of smoothness is $L[-1/(4\beta)]$. Generating the u's therefore takes time $L[\beta/2+3/(8\beta)]$. Notice that we can take $\alpha = \beta/2+3/(8\beta)$. Notice also that u, v, and $uv-p$ are not generated randomly, but instead are selected in a deterministic way. Although we cannot justify it theoretically, we assume that these numbers have the same probability of smoothness as random numbers of about the same size. The running times we get are therefore only heuristic estimates.

Combined with the coordinate recurrence method (cf. (1.23), (1.26)), we find that the precomputation takes time $L[\max(\beta+1/(4\beta), \beta/2+3/(8\beta), 2\beta)]$. This is minimized for $\beta = 1/2$, so that the precomputation can be done in expected time $L[1]$ and storage $L[1/2]$. Notice that for $\beta = 1/2$ we have $\alpha = 1$.

The second stage as described in (2.12) also takes time $L[1]$. If we keep the $L[1/2]$ smooth u's from the precomputation stage, then the second stage can be modified as follows. We find e such that $y \cdot g^e \bmod p$ is smooth with respect to $L[2]$ in time $L[1/4]$. To calculate $\log_g y$, it suffices to calculate $\log_g x$ for each prime factor $x \leq L[2]$ of $y \cdot g^e \bmod p$. For fixed x this is done as follows. Find v in an interval of size $L[1/2]$ around $\sqrt{p}/x$ that is smooth with respect to $L[1/2]$ in time $L[1/2]$. Finally, find one of the $L[1/2]$ smooth u's such that

$uvx-p = O(L[5/2]\sqrt{p})$ is smooth with respect to $L[1/2]$ in time $L[1/2]$. The value of $\log_g x$ now follows. Individual logarithms can therefore be computed in expected time and storage $L[1/2]$.

Generalization of this idea to $G = \mathbb{F}_{2^m}^*$, as in (2.11), follows immediately, if we select some polynomial $g \in \mathbb{F}_2[X]$ of norm about $2^{m/2}$ (for instance $g = X^{\lfloor m/2 \rfloor}$), and compute $q, r \in \mathbb{F}_2[X]$ such that $f = qg+r$ (cf. (2.11)) with degree$(r) <$ degree(g). In the precomputation we consider $u = g+\bar{u}$, $v = q+\bar{v}$ for polynomials $\bar{u}, \bar{v} \in \mathbb{F}_2[X]$ of norm $\leq L[\alpha]$, so that $\mathrm{N}(uv-f)$ is close to $L[\alpha]2^{m/2}$; here $L[\alpha] = L_{2^m-1}[\alpha]$. In the second stage we write $q = hx+\bar{x}$ for $h, \bar{x} \in \mathbb{F}_2[X]$ with degree$(\bar{x}) <$ degree(x), where x is as above, choose $v = h+\bar{v}$ with $\mathrm{N}(\bar{v}) < L[1/2]$, and consider $uvx-f$. The running time analysis remains unchanged. Instead of finding g, q, r as above, we could also choose f in (2.11) such that $f = X^m+f_1$ with degree$(f_1) < m/2$, so that we can take $g = q = X^{\lfloor (m+1)/2 \rfloor}$.

(2.17) *A method based on the linear sieve algorithm from [18].* Again we consider $G = (\mathbb{Z}/p\mathbb{Z})^*$. An improvement of (2.16) that is of practical importance, although it does not affect the timings when expressed in $L(n)$, can be obtained by including the numbers $u \in \{\lfloor \sqrt{p} \rfloor+1, \ldots, \lfloor \sqrt{p}+L[\alpha] \rfloor\}$ in the set S as well. For such u and v we again have $uv-p = O(L[\alpha]\sqrt{p})$, but now we only require that $uv-p$ is smooth with respect to $L[\beta]$, without requiring smoothness for u or v. It follows in a similar way as above that the $L[\beta]+L[\alpha]$ equations can be collected in time $L[1]$ and storage $L[1/2]$ for $\beta = 1/2$ and $\alpha = \beta/2+1/(8\beta) = 1/2$. The reason that this version will run faster than the algorithm from (2.16), is that $uv-p$ is now only $O(L[1/2]\sqrt{p})$, whereas it is $O(L[1]\sqrt{p})$ in (2.16). In practice this will make a considerable difference in the probability of smoothness. The second stage can be adapted in a straightforward way. The running times we get are again only heuristic estimates.

In the methods for $G = (\mathbb{Z}/p\mathbb{Z})^*$ described in (2.16) and (2.17), the use of the smoothness test referred to in (2.10) can be replaced by sieving techniques. This does not change the asymptotic running times, but the resulting algorithms will probably be faster in practice [18].

(2.18) *A more general L function.* For the description of the last algorithm in this section, the bimodal polynomials method, it will be convenient to extend the definition of the function L from (1.1) slightly. For $\alpha, r \in \mathbb{R}$ with $0 \leq r \leq 1$, we denote by $L_x[r;\alpha]$ any function of x that equals $e^{(\alpha+o(1))(\log x)^r(\log\log x)^{1-r}}$, for $x \to \infty$. Notice that this is $(\log x)^\alpha$ for $r = 0$, and x^α for $r = 1$, up to the $o(1)$ in the exponent. For $r = 1/2$ we get the L from (1.1).

The smoothness probabilities from (1.1) and (2.11) can now be formulated as follows. Let $\alpha, \beta, r, s \in \mathbb{R}$ be fixed with $\alpha, \beta > 0$, $0 < r \leq 1$, and $0 < s < r$. From (1.1) we find that a random positive integer $\leq L_x[r;\alpha]$ is $L_x[s;\beta]$-smooth with probability $L_x[r-s; -\alpha(r-s)/\beta]$, for $x \to \infty$. From the same theorem of Odlyzko referred to in (2.11) we have that, for $r/100 < s < 99r/100$, a random polynomial in $\mathbb{F}_2[X]$ of norm $\leq L_x[r;\alpha]$ is smooth with respect to $L_x[s;\beta]$ with probability $L_x[r-s; -\alpha(r-s)/\beta]$, for $x \to \infty$.

(2.19) *Coppersmith's bimodal polynomials method (cf. [17]).* We conclude this section with an algorithm that was especially designed for $G = \mathbb{F}_{2^m}^*$, as in (2.11). This algorithm does not apply to fields with a large characteristic. It is again a variant of the index-calculus algorithm (2.12). We assume that f can be written as X^m+f_1, for $f_1 \in \mathbb{F}_2[X]$ of degree $< m^{2/3}$. Because about one out of every m polynomials in $\mathbb{F}_2[X]$ of degree m is irreducible, we expect that such an f can be found.

We use the function L from (2.18), and we abbreviate $L_{2^m-1}[r;\alpha]$ to $L[r;\alpha]$. Notice that with this notation $L[r;\alpha] = 2^{\alpha(1+o(1))m^r(\log_2 m)^{1-r}}$, for $\alpha > 0$, and $m \to \infty$.

Let S be the set of irreducible polynomials in $\mathbb{F}_2[X]$ of norm $\leq L[1/3;\beta]$, for some $\beta \neq 0$. Furthermore, let k be a power of 2 such that $\mathrm{N}(X^{\lfloor m/k \rfloor})$ is as close as possible to $\mathrm{N}(v^k)$, for a polynomial $v \in \mathbb{F}_2[X]$ of norm $L[1/3;\beta]$; this is achieved for a power of 2 close to $\beta^{-1/2}m^{1/3}(\log_2 m)^{-1/3}$. We find that $t = k/(\beta^{-1/2}m^{1/3}(\log_2 m)^{-1/3})$ satisfies $\sqrt{1/2} < t \le \sqrt{2}$ and that $\mathrm{N}(X^{\lfloor m/k \rfloor}) \le L[2/3;\sqrt{\beta}/t]$ and $\mathrm{N}(v^k) \le L[2/3;t\sqrt{\beta}]$. For polynomials $v_1, v_2 \in \mathbb{F}_2[X]$ of norm $\le L[1/3;\beta]$, we take $u_1 = X^{\lfloor m/k \rfloor+1}v_1+v_2$, and $u_2 = u_1^k \bmod f$. Remark that the polynomial u_1 can be considered as a string of bits with two peaks; this explains the name of the method. Since $\log_g u_2 = (k \cdot \log_g u_1) \bmod (2^m-1)$, we find a linear equation in the $\log_g s$ for $s \in S$, if both u_i's are smooth with respect to $L[1/3;\beta]$. Because the equations generated in this way are homogeneous, we assume that g is smooth with respect to $L[1/3;\beta]$ as well. To analyse the probability that both u_i's are smooth, we compute their norms. By the choice of k we have that $\mathrm{N}(u_1) \le L[2/3;\sqrt{\beta}/t]$. Because k is a power of 2, we have

$$\begin{aligned} u_2 &= (X^{(\lfloor m/k \rfloor+1)k} v_1^k + v_2^k) \bmod f \\ &= X^{(\lfloor m/k \rfloor+1)k-m} f_1 v_1^k + v_2^k, \end{aligned}$$

so that $\mathrm{N}(u_2) \le L[2/3;t\sqrt{\beta}]$. The probability that both are smooth with respect to $L[1/3;\beta]$ therefore is assumed to be $L[1/3;-1/(3t\sqrt{\beta})] \cdot L[1/3;-t/(3\sqrt{\beta})] = L[1/3;-(t+t^{-1})/(3\sqrt{\beta})]$. The $L[1/3;\beta]^2$ pairs (v_1, v_2) must suffice to generate the $\approx L[1/3;\beta]$ equations that we need (where we only consider polynomials v_1, v_2 that are relatively prime because the pairs (v_1, v_2) and $(w \cdot v_1, w \cdot v_2)$ yield the same equation). It follows that β must satisfy

$$L[1/3;2\beta] \ge L[1/3;\beta+(t+t^{-1})/(3\sqrt{\beta})].$$

The optimal choice for β is $((t+t^{-1})/3)^{2/3}$, and the value for t then follows by taking t with $\sqrt{1/2} < t \le \sqrt{2}$ such that $t((t+t^{-1})/3)^{-1/3}m^{1/3}(\log_2 m)^{-1/3}$ is a power of 2. In the worst case $t = \sqrt{2}$ we find $\beta = (1/2)^{1/3} = 0.794$, so that the precomputation can be done in time $2^{(1.588+o(1))m^{1/3}(\log_2 m)^{2/3}}$ (cf. (1.23), (1.26)). If we are so lucky that t can be chosen as 1, we find $\beta = (4/9)^{1/3} = 0.764$, which makes the precomputation slightly faster.

To compute $\log_g y$ for $y \in \mathbb{F}_{2^m}^*$ we proceed as follows. We find e such that $y \cdot g^e \bmod f$ of norm $\le L[1;1]$ is smooth with respect to $L[2/3;1]$ in time $L[1/3;1/3]$. Let $\bar{y}$ be one of the irreducible factors of $y \cdot g^e \bmod f$ with $\mathrm{N}(\bar{y}) \le L[2/3;1]$. Let k be a power of 2 such that $\mathrm{N}(X^{\lfloor m/k \rfloor}) \approx \mathrm{N}(v^k)$, for a polynomial $v \in \mathbb{F}_2[X]$ of norm $L[2/3;1]$; in the worst case we get $\mathrm{N}(X^{\lfloor m/k \rfloor}) = L[5/6;\sqrt{1/2}]$ and $\mathrm{N}(v^k) = L[5/6;\sqrt{2}]$. Find polynomials $v_1, v_2 \in \mathbb{F}_2[X]$ of norm $\le L[2/3;1]$, such that $\bar{y}$ divides $u_1 = X^{\lfloor m/k \rfloor+1}v_1+v_2$, and such that both $u_1/\bar{y}$ and $u_2 = u_1^k \bmod f$ are smooth with respect to $L[1/2;1]$. It follows from the choice for k that $u_1/\bar{y}$ and u_2 have norms bounded by $L[5/6;\sqrt{1/2}]$ and $L[5/6;\sqrt{2}]$, respectively, so that the probability that both are smooth with respect to $L[1/2;1]$ is assumed to be $L[1/3;-\sqrt{2}/6] \cdot L[1/3;-\sqrt{2}/3] = L[1/3;-\sqrt{1/2}]$. Because $L[2/3;1]^2/L[2/3;1]$ of the pairs (v_1, v_2) satisfy the condition that $\bar{y}$ divides u_1, we must have that

$$L[2/3;1] \ge L[1/3;\sqrt{1/2}].$$

This condition is satisfied, and we find that the computation of the u_i's can be done in time $L[1/3;\sqrt{1/2}] = 2^{(\sqrt{1/2}+o(1))m^{1/3}(\log_2 m)^{2/3}}$.

Because $\log_g u_2 = (k \cdot (\log_g(u_1/\bar{y}) + \log_g \bar{y})) \bmod (2^m-1)$, we have reduced the problem of computing the discrete logarithm of a polynomial of norm $L[2/3;1]$ (the factor $\bar{y}$ of $y \cdot g^e \bmod f$), to the problem of computing the discrete logarithms of polynomials of norm $\le L[1/2;1]$ (the irreducible factors of $u_1/\bar{y}$ and u_2). To express $\log_g y$ in terms of $\log_g s$ for $s \in S$, we apply the above method recursively to each of the irreducible factors of $u_1/\bar{y}$ and u_2, thus creating a sequence of norms $L[1/3+1/3;1]$, $L[1/3+1/6;1]$, $L[1/3+1/12;1]$, ... that converges to $L[1/3;1]$. The recursion is always applied to $< m$ polynomials per recursion step, and at recursion depth $O(\log m)$ all factors have norm $\le L[1/3;1]$, so that the total time to express $\log_g y$ in terms of $\log_g s$ for $s \in S$ is bounded by $m^{O(\log m)} L[1/3;\sqrt{1/2}] = 2^{(\sqrt{1/2}+o(1))m^{1/3}(\log_2 m)^{2/3}}$.

The analysis of the storage needed for this algorithm is left to the reader. We refer to [17] for some useful remarks concerning the implementation of this algorithm.

3. Factoring integers

(3.1) Introduction

A well-known method to factor a composite number n, *Pollard's* $p-1$ *method*, is based on the following observation. For a prime p and any multiple k of the order $p-1$ of $(\mathbb{Z}/p\mathbb{Z})^*$, we have $a^k \equiv 1 \bmod p$, for any integer a that is not divisible by p. Therefore, if p divides n, then p divides $\gcd(a^k-1, n)$, and it is not unlikely that a non-trivial divisor of n is found by computing this gcd. This implies that prime factors p of n for which $p-1$ is s-smooth, for some $s \in \mathbb{Z}_{>0}$, can often be detected in $O(s \log_s n)$ operations in $\mathbb{Z}/n\mathbb{Z}$, if we take k as in (2.6), with $B = n$. Notice that, in this method, we consider a multiplicative group modulo an unspecified prime divisor of n, and that we *hope* that the order of this group is smooth (cf. (2.1)).

Unfortunately, this method is only useful for composite numbers that have prime factors p for which $p-1$ is s-smooth for some *small* s. Among the generalizations of this method [6,47,69] one method, the *elliptic curve method* [37], stands out: instead of relying on fixed properties of a factor p, it depends on properties that can be randomized, independently of p. To be more precise, the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ of fixed order $p-1$, is replaced by the set of points of an elliptic curve modulo p. This set of points is a group whose order is *close to p*; varying the curve will vary the order of the group and trying sufficiently many curves will almost certainly produce a group with a smooth order.

Another way of randomizing the group is by using class groups. For a small positive integer t with $t \equiv -n \bmod 4$, we have that $\Delta = -tn$ satisfies $\Delta \equiv 1 \bmod 4$ if n is odd. According to (1.19) a factorization of Δ can be obtained if we are able to compute an odd multiple of the largest odd divisor of the class number h_Δ. If h_Δ is s-smooth, such a multiple is given by k as in (2.6), where p ranges over the primes $2 < p \le s$, and $B = |\Delta|^{1/2+o(1)}$ (cf. (1.18)). By varying t, we expect to find a smooth class number after a while: with $s = L_n[1/2]$, we expect $L_n[1/2]$ trials (cf. (1.1), (1.18)), so that, with (2.6) and (1.19), it takes expected time $L_n[1]$ to factor n. For details of this method, the *class group method*, we refer to [56].

In this section we will discuss the elliptic curve method (Section 3.2), its consequences for other methods (Section (3.9)), and a very practical factoring algorithm that does not depend on the use of elliptic curves, the multiple polynomial variation of the quadratic sieve algorithm (Section (3.19)).

Other methods and extensions of the ideas presented here can be found in [31,39,54]. The running times we derive are only informal upper bounds. For rigorous proofs of some of the results below, and for lower bounds, we refer to [49] and [51].

(3.2) Factoring integers with elliptic curves

Let n be a composite integer that we wish to factor. In this section we present an algorithm to factor n that is based on the theory of elliptic curves. The running time analysis of this factoring algorithm depends upon an as yet unproved hypothesis, for which we refer to Remark (3.6).

(3.3) *The elliptic curve method (cf. [37]).* We assume that $n > 1$, that $\gcd(n, 6) = 1$, and that n is not a power with exponent > 1; these conditions can easily be checked. To factor n we proceed as follows:

> Randomly draw $a, x, y \in \mathbb{Z}/n\mathbb{Z}$, put $P = (x:y:1) \in V_n$ (cf. (1.10)), and select an integer k as in (2.6) (with s and B to be specified below). Attempt to compute $k \cdot P$ by means of the algorithm described in (1.12). If the attempt fails, a divisor d of n with

$1 < d < n$ is found, and we are done; otherwise, if we have computed $k \cdot P$, we start all over again.

This finishes the description of the algorithm.

(3.4) *Explanation of the elliptic curve method.* We expect this algorithm to work, for a suitable choice of k, for the following reason. Let p and q be primes dividing n with $p < q$. In most iterations of the algorithm it will be the case that the pair a, y^2-x^3-ax when taken modulo p (q) defines an elliptic curve over $\mathbb{F}_p$ $(\mathbb{F}_q)$. Now suppose that k is a multiple of the order of P_p; the value for k will be chosen such that a certain amount of luck is needed for this to happen. *If* it happens, it is unlikely that we are so lucky for q as well, so that k is *not* a multiple of the order of P_q. Then $k \cdot P$ cannot have been computed successfully (see (1.12)), but a factorization of n has been found instead.

(3.5) *Running time analysis.* Let p be the smallest prime divisor of n, and let $\beta \in \mathbb{R}_{>0}$. We assume that the probability that the order of P_p is smooth with respect to $L_p[\beta]$ is approximately $L_p[-1/(2\beta)]$ (cf. (1.1) and (1.6), and see Remark (3.6)). Therefore, if we take k as in (2.6) with $s = L_p[\beta]$ and $B = p+2\sqrt{p}+1$ (cf. (1.6)), then about one out of every $L_p[1/(2\beta)]$ iterations will be successful in factoring n. According to (2.6) and (1.12) each iteration takes $O(L_p[\beta] \cdot \log p)$ additions in V_n, which amount to $O(L_p[\beta](\log p)(\log n)^2)$ bit operations. The total expected running time therefore is $O((\log p)(\log n)^2 L_p[\beta+1/(2\beta)])$ which becomes $O((\log n)^2 L_p[\sqrt{2}])$ for the optimal choice $\beta = \sqrt{1/2}$.

Of course the above choice for k depends on the divisor p of n that we do not know yet. This can be remedied by replacing p by a tentative upper bound v in the above analysis. If one starts with a small v that is suitably increased in the course of the algorithm, one finds that a non-trivial factor of n can be found in expected time $O((\log n)^2 L_p[\sqrt{2}])$, under the assumption made in (3.6). In the worst case $v = \sqrt{n}$ this becomes $L_n[1]$. The storage required is $O(\log n)$.

Another consequence is that for any fixed $\alpha \in \mathbb{R}_{>0}$, an integer n can be tested for smoothness with respect to $v = L_n[\alpha]$ in time $L_n[0]$; in case of smoothness the complete factorization of n can be computed in time $L_n[0]$ as well.

For useful remarks concerning the implementation of the elliptic curve method we refer to [10] and [42].

(3.6) **Remark.** A point that needs some further explanation is our assumption in (3.5) that the order of P_p is $L_p[\beta]$-smooth with probability approximately $L_p[-1/(2\beta)]$. Let $E_{\bar{a},\bar{b}}(\mathbb{F}_p)$ be the group under consideration. Regarding $\bar{a}$ and $\bar{b}$ as random integers modulo p, Proposition (1.7) asserts that the probability that $\#E_{\bar{a},\bar{b}}(\mathbb{F}_p)$ is smooth with respect to $L_p[\beta]$ and contained in the interval $(p-\sqrt{p}+1, p+\sqrt{p}+1)$, is essentially the same as the probability that a random integer in $(p-\sqrt{p}+1, p+\sqrt{p}+1)$ is $L_p[\beta]$-smooth.

From (1.1) we know that a random integer $\leq p$ is $L_p[\beta]$-smooth with probability $L_p[-1/(2\beta)]$, and we *assume* here that the same holds for random integers in $(p-\sqrt{p}+1, p+\sqrt{p}+1)$. Because this has not been proved yet, the running times in (3.5) are conjectural.

Of course, if $\#E_{\bar{a},\bar{b}}(\mathbb{F}_p)$ is $L_p[\beta]$-smooth, then the order of P_p is $L_p[\beta]$-smooth as well.

(3.7) *A rigorous smoothness test.* As explained in (3.6), the running times in (3.5) are conjectural. The result concerning the elliptic curve smoothness test can, however, be rigorously proved, in a slightly weaker and average sense. Briefly, the following has been shown in [51].

Let for some real number y the set $S(y)$ be as in [51] the set of primes p, $3 < p \leq y$, for which the number of $e^{(\log p)^{6/7}}$-smooth integers in the interval $(p-\sqrt{p}, p+\sqrt{p})$ is more than $\sqrt{p} \cdot e^{-((\log y)^{1/7} \log\log y)/6}$. As shown in [51] it follows from a result in [25] that $\#S(y)$ is reasonably close to $\pi(y)$, the number of *all* primes $\leq y$. Define $\psi_1(x, y, z)$ as the number of integers $\leq x$ that are built up from primes p such that $p \leq z$ or $p \in S(y)$. For the proper y and z, this function $\psi_1(x, y, z)$ behaves as $\psi(x, y)$ (cf. (1.1)):

(3.8) $\psi_1(x, L_x[\beta], e^{64(\mathrm{loglog}x)^6}) = x \cdot L_x[-1/(2\beta)]$,

for $\beta \in \mathbb{R}_{>0}$ fixed (cf. [51, Lemma 3.1]). We say that an integer $\leq x$ is (x, y)-smooth if it is built up from primes p such that $p \leq e^{64(\mathrm{loglog}x)^6}$ or $p \in S(y)$.

It follows from (3.8) that $(n, L_n[\beta])$-smooth numbers occur asymptotically about as frequently as ordinary $L_n[\beta]$-smooth numbers. Furthermore, it can be proved that an $(n, L_n[\beta])$-smooth number $\leq n$ can be recognized with high probability in time $L_n[0]$. This is done as follows. First, the prime factors $\leq e^{64(\mathrm{loglog}n)^6}$ are removed by trial division. If the resulting quotient a is not equal to 1, apply the elliptic curve method to find the factors $\leq L_n[\beta]$ of a. If a is $(n, L_n[\beta])$-smooth, then all factors in the second stage are actually in $S(L_n[\beta])$. Because of the way $S(y)$ is defined, it can be proved that, in case of smoothness, the complete factorization will be found with probability at least $1-(\log a)/a$ and in time $L_n[0]$ (cf. [51, Theorem 2.1]).

(3.9) Applications of the elliptic curve method to older factorization methods

The elliptic curve smoothness test that we have seen at the end of (3.5) appears to be very useful as an auxiliary tool in various other subexponential integer factoring algorithms. In this section we will illustrate this with three examples. We abbreviate $L_n[\beta]$ to $L[\beta]$.

(3.10) *Dixon's random squares algorithm (cf. [23,49]).* Let n be a composite integer that we wish to factor, and let $\beta \in \mathbb{R}_{>0}$. In this algorithm one attempts to find integers x and y such that $x^2 \equiv y^2 \bmod n$ in the following way:

(1) randomly select integers m until sufficiently many are found for which the least positive residue $r(m)$ of $m^2 \bmod n$ is $L[\beta]$-smooth;

(2) find a subset of the m's such that the product of the corresponding $r(m)$'s is a square, say x^2;

(3) put y equal to the product of the m's in this subset, then $x^2 \equiv y^2 \bmod n$.

Dixon has shown that, if n is composite, not a prime power, and free of factors $\leq L[\beta]$, then with probability at least 1/2, a factor of n will be found by computing $\gcd(x+y, n)$, for x and y as above (cf. [23]). Therefore, we expect to factor n if we repeat the second and third step a small number of times.

Before analysing the running time of this algorithm, let us briefly explain how the second step can be done. First notice that $\pi(L[\beta]) = L[\beta]$ (cf. (1.1)). Therefore, each $r(m)$ can be represented by an $L[\beta]$-dimensional integer vector whose ith coordinate is the number of times the ith prime occurs in $r(m)$. A linear dependency modulo 2 among those vectors then yields a product of $r(m)$'s where all primes occur an even number of times, and therefore the desired x^2. This idea was first described in [43].

To analyse the running time of the random squares algorithm, notice that we need about $L[\beta]$ smooth m's in the first step to be able to find a linear dependency in the second step. According to (1.1) a random integer $\leq n$ is $L[\beta]$-smooth with probability $L[-1/(2\beta)]$, and according to (3.5) such an integer can be tested for smoothness with respect to $L[\beta]$ in time $L[0]$. One $L[\beta]$-smooth $r(m)$ can therefore be found in expected time $L[1/(2\beta)]$, and $L[\beta]$ of them will take time $L[\beta+1/(2\beta)]$. It is on this point that the random squares algorithm distinguishes itself from the other factoring algorithms in this section. Namely, it can be *proved* that, for random m's, the $r(m)$'s behave with respect to smoothness properties as random integers $< n$ (cf. [23]). This makes it possible to give a *rigorous* analysis of the expected running time of the random squares algorithm. For practical purposes, however, the algorithm cannot be recommended.

The linear dependencies in the second step can be found by means of Gaussian elimination in time $L[3\beta]$. The whole algorithm therefore runs in expected time $L[\max(\beta+1/(2\beta), 3\beta)]$.

This is minimized for $\beta = 1/2$, so that we find that the random squares algorithm takes time $L\,[3/2]$ and storage $L\,[1]$.

As in algorithm (2.12), however, we notice that at most $\log_2 n$ of the $L\,[\beta]$ coordinates of each vector can be nonzero. To multiply the matrix consisting of the vectors representing $r(m)$ by another vector takes therefore time at most $(\log_2 n)L\,[\beta] = L\,[\beta]$. Applying the coordinate recurrence method (cf. (1.23)) we conclude that dependencies can be found in expected time $L\,[2\beta]$, so that the random squares algorithm takes expected time $L\,[\max(\beta+1/(2\beta), 2\beta)]$, which is $L\,[\sqrt{2}]$ for $\beta = \sqrt{1/2}$. The storage needed is $L\,[\sqrt{1/2}]$. The random squares algorithm is the fastest, fully proved factoring algorithm; for a rigorous proof using the smoothness test from (3.7) we refer to [51]. Notice that the random squares algorithm is in a way very similar to the index calculus algorithm (2.12).

(3.11) *The continued fraction algorithm (cf. [43]).* If we could generate the m's in step (1) of the random squares algorithm in such a way that the $r(m)$'s are small, say $\leq \sqrt{n}$, then the $r(m)$'s would have a higher probability of being smooth, and that would probably speed up the factoring algorithm. This is precisely what is done in the continued fraction algorithm. Suppose that n is not a square, let a_i/b_i denote the ith continued fraction convergent to $\sqrt{n}$, and let $r(a_i) = a_i^2 - nb_i^2$. It follows from the theory of continued fractions (cf. [27, theorem 164]) that $|r(a_i)| < 2\sqrt{n}$. Therefore we replace the first step of the random squares algorithm by the following:

> Compute a_i mod n and $r(a_i)$ for $i = 1, 2, \ldots$ until sufficiently many $L\,[\beta]$-smooth $r(a_i)$'s are found.

The computation of the a_i mod n and $r(a_i)$ can be done in $O((\log n)^2)$ bit operations (given the previous values) by means of an iteration that is given in [43]. The second step of the random squares algorithm can be adapted by including an extra coordinate in the vector representing $r(a_i)$ for the factor -1. The smoothness test is again done by means of the elliptic curve method. Assuming that the $|r(a_i)|$ behave like random numbers $< 2\sqrt{n}$ the probability of smoothness is $L\,[1/(4\beta)]$, so that the total running time of the algorithm becomes $L\,[\max(\beta+1/(4\beta), 2\beta)]$. With the optimal choice $\beta = 1/2$ we find that time and storage are bounded by $L\,[1]$ and $L\,[1/2]$, respectively.

We have assumed that the $|r(a_i)|$ have the same probability of smoothness as random numbers $< 2\sqrt{n}$. The fact that all primes p dividing $r(a_i)$ and not dividing n satisfy $(\frac{n}{p}) = 1$, is not a serious objection against this assumption; this follows from (3.17) under the assumption of the generalized Riemann hypothesis. More serious is that the $r(a_i)$ are generated in a deterministic way, and that the period of the continued fraction expansion for $\sqrt{n}$ might be short. In that case one may replace n by a small multiple.

The algorithm has proved to be quite practical, where we should note that in the implementations the smoothness of the $r(a_i)$ is usually tested by other methods. For a further discussion on the theoretical justification of this method we refer to [49].

(3.12) *Seysen's class group algorithm (cf. [61]).* Let n be the composite integer to be factored. We assume that n is odd, and that $-n \equiv 1 \bmod 4$, which can be achieved by replacing n by $3n$ if necessary. Put $\Delta = -n$, and consider the class group C_Δ (cf. (1.14)). We introduce some concepts that we need in order to describe the factorization algorithm.

(3.13) *Randomly generating reduced forms with known factorization.* Consider the prime forms I_p, with $p \leq c \cdot (\log|\Delta|)^2$, that generate C_Δ (cf. (1.20)). Let $e_p \in \{0, 1, \ldots, |\Delta|-1\}$ be randomly and independently selected, for every I_p. It follows from the bound on the class number h_Δ (cf. (1.18)) and from the fact that the I_p generate C_Δ, that the reduced form $\prod I_p^{e_p}$ behaves approximately as a random reduced form in C_Δ, i.e.: for any reduced form $f \in C_\Delta$ we have that $f = \prod I_p^{e_p}$ with probability $(1+o(1))/h_\Delta$, for $n \to \infty$ (cf. [61, Lemma 8.2]).

(3.14) *Finding an ambiguous form.* Let $\beta \in \mathbb{R}_{>0}$; notice that $L[\beta] > c\cdot(\log|\Delta|)^2$. We attempt to find an ambiguous form (cf. (1.17)) in a way that is more or less similar to the random squares algorithm.

A randomly selected reduced form $(a, b) \in C_\Delta$ can be written as $\prod_{p \le L[\beta]} I_p^{t_p}$ with probability $L[-1/(4\beta)]$ (cf. (1.21)), where at most $O(\log|\Delta|)$ of the exponents t_p are non-zero. According to (3.13) we get the same probability of smoothness if we generate the forms (a, b) as is done in (3.13). Therefore, if we use (3.13) to generate the random reduced forms, we find with probability $L[-1/(4\beta)]$ a relation

$$\prod_{\substack{p \le c\cdot(\log|\Delta|)^2 \\ p \text{ prime}}} I_p^{e_p} = \prod_{\substack{p \le L[\beta] \\ p \text{ prime}}} I_p^{t_p}.$$

With $r_p = e_p - t_p$, where $e_p = 0$ for $p > c\cdot(\log|\Delta|)^2$, we get

$$(3.15) \qquad \prod_{\substack{p \le L[\beta] \\ p \text{ prime}}} I_p^{r_p} = 1_\Delta.$$

Notice that at most $c\cdot(\log|\Delta|)^2 + \log|\Delta|$ of the exponents r_p are non-zero. If all exponents are even, then the left hand side of (3.15) with r_p replaced by $r_p/2$ is an ambiguous form. Therefore, if we have many equations like (3.15), and combine them in the proper way, we might be able to find an ambiguous form; as in the random squares algorithm this is done by looking for a linear dependency modulo 2 among the vectors consisting of the exponents r_p.

There is no guarantee, however, that the thus constructed ambiguous form leads to a non-trivial factorization of $|\Delta|$. Fortunately, the probability that this happens is large enough, as shown in [61, Proposition 8.6] or [36, Section (4.6)]: if $L[\beta]$ equations as in (3.15) have been determined in the way described above, then a random linear dependency modulo 2 among the exponent-vectors leads to a non-trivial factorization with probability at least $1/2 - o(1)$.

(3.16) *Running time analysis.* The $L[\beta] \times L[\beta]$-matrix containing the exponent-vectors is sparse, as reasoned above, so that a linear dependency modulo 2 can be found in expected time $L[2\beta]$ by means of the coordinate recurrence method (cf. (1.23)). For a randomly selected reduced form (a, b), we assume that a can be tested for $L[\beta]$-smoothness in time $L[0]$ (cf. (3.5)). Generation of the $L[\beta]$ equations like (3.15) then takes time $L[\beta + 1/(4\beta)]$, under the assumption of the GRH. The whole algorithm therefore takes expected time $L[\max(\beta + 1/(4\beta), 2\beta)]$, which is $L[1]$ for $\beta = 1/2$, under the assumption of the generalized Riemann hypothesis.

We can prove this expected running time rigorously under the assumption of the GRH, if we adapt the smoothness test from (3.7) to this situation. Let $\psi_\Delta(x, y)$ be the number of positive integers $\le x$ that are built up from primes $p \le y$ for which $(\frac{\Delta}{p}) = 1$. Under the assumption of the generalized Riemann hypothesis we have that

$$(3.17) \qquad \psi_\Delta(x, L[\beta]) = x \cdot L[-1/(4\beta)],$$

where $x = \sqrt{|\Delta|}/2$ and with $\beta \in \mathbb{R}_{>0}$ fixed (cf. [61, Theorem 5.2]). Let $S(y)$ be as in (3.7), and let $\psi_{\Delta,1}(x, y, z)$ be the number of positive integers $\le x$ that are built up from primes p for which $p \le z$ or $p \in S(y)$, and for which $(\frac{\Delta}{p}) = 1$.

As in (3.7) we have that for the proper x, y, and z, the function $\psi_{\Delta,1}(x, y, z)$ behaves as $\psi_\Delta(x, y)$:

$$(3.18) \qquad \psi_{\Delta,1}(x, L[\beta], e^{64(\log\log x)^6}) = x \cdot L[-1/(4\beta)],$$

where again $x = \sqrt{|\Delta|}/2$ and $\beta \in \mathbb{R}_{>0}$ fixed. This follows from (3.17) and the proof of [51, Lemma 3.1], under the assumption of the GRH. We say that an integer $\le x$ is (x, y, Δ)-smooth if it is built up from primes p with $(\frac{\Delta}{p}) = 1$ and such that $p \le e^{64(\log\log x)^6}$ or

$p \in S(y)$; define (x, y, Δ)-smoothness for forms correspondingly.

From [61, Lemma 5.1] we have that the number of $L[\beta]$-smooth reduced forms (a, b) with $a < \sqrt{|\Delta|/2}$ is at least $\psi_\Delta(\sqrt{|\Delta|/2}, L[\beta])$, and it follows then easily from (3.17) and the bound on h_Δ from (1.18) that a random reduced form is $L[\beta]$-smooth with probability at least $L[-1/(4\beta)]$. Similarly one proves, using (3.18), that a random reduced form is $(\sqrt{|\Delta|/2}, L[\beta], \Delta)$-smooth with probability at least $L[-1/(4\beta)]$. If we therefore use this somewhat stronger notion of smoothness in (3.14), we still have a probability $L[-1/(4\beta)]$ to find a relation as in (3.15). The smoothness test is done as explained in the last paragraph of (3.7). It follows from (3.7) that, in case of smoothness, the complete factorization will be found with probability at least $1-(\log a)/a$, where a is the least prime in $S(L[\beta])$ that is $> e^{64(\log\log n)^6}$. The time needed for the smoothness test is $L[0]$.

From these observations it follows that a random reduced form that is $(\sqrt{|\Delta|/2}, L[\beta], \Delta)$-smooth, can be found in expected time $L[1/(4\beta)]$. We conclude that, under the assumption of the generalized Riemann hypothesis, this version of Seysen's class group algorithm can be rigorously proved to run in expected time $L[1]$. This improves Seysen's result. We refer to [36] for a more detailed proof.

(3.19) The quadratic sieve algorithm

In this final section about factorization algorithms, we briefly describe a practical factoring algorithm that runs in expected time $L_n[1]$, and that existed before the elliptic curve method. As the methods from the previous section, but unlike the elliptic curve method, the running time of the algorithm to be presented here does not depend on the size of the factors. Nevertheless, the method has proved to be very useful, especially in cases where the elliptic curve method performs poorly, i.e., if the number n to be factored is the product of two primes of about the same size. We abbreviate $L_n[\beta]$ to $L[\beta]$.

(3.20) *Pomerance's quadratic sieve algorithm (cf. [49]).* The quadratic sieve algorithms only differ from the algorithms in (3.10) and (3.11) in the way the $L[\beta]$-smooth quadratic residues modulo n are determined, for some $\beta \in \mathbb{R}_{>0}$. In the ordinary quadratic sieve algorithm that is done as follows. Let $r(X) = ([\sqrt{n}]+X)^2-n$ be a quadratic polynomial in X. For any $m \in \mathbb{Z}$ we have that $r(m) \equiv ([\sqrt{n}]+m)^2 \bmod n$ is a square modulo n, so in order to solve $x^2 \equiv y^2 \bmod n$ we look for $\approx L[\beta]$ integers m such that $r(m)$ is $L[\beta]$-smooth.

Let $\alpha \in \mathbb{R}_{>0}$, and let $|m| \leq L[\alpha]$. Then $|r(m)| = O(L[\alpha]\sqrt{n})$, so that $|r(m)|$ is $L[\beta]$-smooth with probability $L[-1/(4\beta)]$ according to (1.1), if $|r(m)|$ behaves as a random integer $\leq L[\alpha]\sqrt{n}$. Under this assumption we find that we must take $\alpha \geq \beta+1/(4\beta)$, in order to obtain sufficiently many smooth $r(m)$'s for $|m| \leq L[\alpha]$.

We have that $(\frac{n}{p}) = 1$ for primes $p \neq 2$ not dividing n, because if $p \mid r(m)$, then $([\sqrt{n}]+m)^2 \equiv n \bmod p$. As in (3.11), this is not a serious objection against our assumption that the $r(m)$ have the same probability of smoothness as random numbers of order $L[\alpha]\sqrt{n}$ (cf. (3.17) under the GRH). The problem is to prove that at least a certain fraction of the $r(m)$'s with $|m| \leq L[\alpha]$ behave with respect to smoothness properties as random numbers of order $L[\alpha]\sqrt{n}$. For a further discussion of this point see [49].

Now consider how to test the $L[\alpha]$ numbers $r(m)$ for smoothness with respect to $L[\beta]$. Of course, this can be done by means of the elliptic curve smoothness test in time $L[\alpha]$ (cf. (3.5)), thus giving a method that runs in time $L[\max(\beta+1/(4\beta), 2\beta)] = L[1]$ for $\beta = 1/2$ (cf. (1.23)). The same time can, however, also be achieved without the elliptic curve method. Let p be a prime $\leq L[\beta]$ not dividing n such that $p \neq 2$ and $(\frac{n}{p}) = 1$. The equation $r(X) \equiv 0 \bmod p$ then has two solutions $m_1(p)$ and $m_2(p)$, which can be found by means of a probabilistic method in time polynomial in $\log p$ (cf. [31, Section 4.6.2]). But then $r(m_i(p)+kp) \equiv 0 \bmod p$ for any $k \in \mathbb{Z}$. Therefore, if we have a list of values of $r(m)$ for all consecutive values of m under consideration, we easily find the multiples of p among them at locations $m_i(p)+kp$ for any $k \in \mathbb{Z}$ such that $|m_i(p)+kp| \leq L[\alpha]$, and $i = 1, 2$. For every p this takes twice time $L[\alpha]/p$, so that for all $p \leq L[\beta]$ with $(\frac{n}{p}) = 1$ together, this so-called *sieving* can be done in time $\sum_p L[\alpha]/p = L[\alpha]$. A similar procedure takes care of the powers of p and $p = 2$. We conclude that we indeed get the same time $L[1]$ as with the elliptic curve smoothness test, but now we need to store all $L[1]$ values $r(m)$. We should note, however, that sieving is in practice much faster than applying the elliptic curve smoothness test, and that the sieving interval can easily be divided into smaller consecutive intervals, to reduce the storage requirements. (Actually, not the $r(m)$ but their logarithms are stored, and the $r(m)$ are not divided by p but $\log p$ is subtracted from $\log r(m)$ during the sieving.) For other practical considerations we refer to [49].

(3.21) *The multiple polynomial variation (cf. [50,63]).* Because there is only one polynomial in (3.20) that generates all smooth numbers that are needed, the size of the sieving interval must be quite large. Also, the quadratic residues $r(m)$ grow linearly with the size of the interval, which reduces the smoothness probability. If we could use *many* polynomials as in (3.20) and use a smaller interval for each of them, we might get a faster algorithm. This idea is due to Davis (cf. [20]); we follow the approach that was independently suggested by Montgomery (cf. [50,63]).

Let $r(X) = a^2X^2+bX+c$, for $a, b, c \in \mathbb{Z}$. In order for $r(m)$ to be a quadratic residue modulo n, we require that the discriminant $D = b^2-4a^2c$ is divisible by n, because then $r(m) \equiv (am+b/(2a))^2 \bmod n$. We show how to select a, b and c so that $|r(m)| = O(L[\alpha]\sqrt{n})$ for $|m| \leq L[\alpha]$. Let $D \equiv 1 \bmod 4$ be a small multiple of n, and let $a \equiv 3 \bmod 4$ be free of primes $\leq L[\beta]$ (if p divides a then $r(X)$ has at most one root modulo p), such that $a^2 \approx \sqrt{D}/L[\alpha]$ and the Jacobi symbol $(\frac{a}{D})$ equals 1. For a we take a probable prime satisfying these conditions (cf. (4.3)). We need an integer b such that $b^2 \equiv D \bmod 4a^2$; the value for c then follows. We put $b_1 = D^{(a+1)/4} \bmod a$, so that $b_1^2 \equiv D \bmod a$ because a is a quadratic residue modulo D and $D \equiv 1 \bmod 4$. Hensel's lemma now gives us $b = b_1+a\cdot((2b_1)^{-1}((D-b_1^2)/a) \bmod a)$; if b is even, we replace b by $b-a^2$, so that the result satisfies $b^2 \equiv D \bmod 4a^2$.

It follows from $a^2 \approx \sqrt{D}/L[\alpha]$ that $b = O(\sqrt{D}/L[\alpha])$, so that $c = O(L[\alpha]\sqrt{D})$. We find that $r(m) = O(L[\alpha]\sqrt{D})$ for $|m| \leq L[\alpha]$. For any a as above, we can now generate a quadratic polynomial satisfying our needs. Doing this for many a's, we can sieve over many shorter intervals, with a higher probability of success. Remark that this can be done in parallel and independently on any number of machines, each machine working on its own sequence of a's.

4. Primality testing

(4.1) Introduction

As we will see in Section (4.2) it is usually easy to prove the compositeness of a composite number, without finding any of its factors. Given the fact that a number is composite it is in general quite hard to find its factorization, but once a factorization is found it is an easy matter to verify its correctness. For prime numbers it is just the other way around. There it is easy to find the answer, i.e., prime or composite, but in case of primality it is not at all straightforward to verify the correctness of the answer. The latter problem, namely ***proving*** primality, is the subject of this section. By *primality test* we will mean an algorithm to prove primality.

In Section 3 we have seen that replacing the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ in Pollard's $p-1$ method by the group $E(\mathbb{Z}/p\mathbb{Z})$, for an elliptic curve E modulo p, resulted in a more general factoring algorithm. In this section we will see that a similar change in an older primality test that is based on the properties of $(\mathbb{Z}/p\mathbb{Z})^*$, leads to new primality tests.

This older algorithm is reviewed in Section (4.2), together with some well-known results concerning probabilistic compositeness algorithms. The primality tests that depend on the use of elliptic curves are described in Section (4.6).

More about primality tests and their implementations can be found in [68] and [39].

(4.2) Some classical methods

Let n be a positive integer to be tested for primality. In this section we review a method, based on a variant of *Fermat's theorem*, by which compositeness of n can easily be proved. If n has not been proved to be composite by this method, then it is considered to be very likely that n is a prime; actually, such numbers are called *probable primes.* It remains to *prove* that such a number is prime. For this purpose, we will present a method that is based on a theorem of Pocklington, and that makes use of the factorization of $n-1$.

(4.3) *A probabilistic compositeness test.* Fermat's theorem states that, if n is prime, then $a^n \equiv a \bmod n$ for all integers a. Therefore, to prove that n is composite, it suffices to find an integer a for which $a^n \not\equiv a \bmod n$; such an a is called a *witness* to the compositeness of n. Unfortunately, there exist composite numbers, the so-called *Carmichael numbers*, for which no witnesses exist, so that a compositeness test based on Fermat's theorem cannot be guaranteed to work.

The following variant of Fermat's theorem does not have this disadvantage: if n is prime, then $a^u \equiv \pm 1 \bmod n$ or $a^{u \cdot 2^i} \equiv -1 \bmod n$ for an integer $i \in \{1, 2, ..., k-1\}$, where $0 < a < n$ and $n-1 = u \cdot 2^k$ with u odd. Any a for which no such i exists is again called a witness to the compositeness of n; if a is not a witness we say that n passes the test for this a. It has been proved [53] that for an odd composite n, there are at least $3(n-1)/4$ witnesses among $\{1, 2, ..., n-1\}$. Therefore, if we randomly select some a's from this interval, and subject n to the test using these a's, it is rather unlikely that a composite n passes all tests. A number passing several tests, say 10, is called a *probable prime.*

Miller has shown that, if the generalized Riemann hypothesis holds, then there is for each composite n a witness in $\{2, 3, ..., c \cdot (\log n)^2\}$ for some effectively computable constant c [41]; according to [5] the value $c = 2$ suffices. Notice that a proof of the generalized Riemann hypothesis therefore would lead to a primality test that runs in time polynomial in $\log n$. For a probabilistic compositeness test based on Jacobi symbols see [64]; this test is weaker than the above test, in the sense that for composite n there is a smaller number of witnesses to the compositeness of n.

Now that we can recognize composite numbers, let us consider how to prove the primality of a probable prime.

(4.4) Pocklington's theorem *(cf. [45]). Let n be an integer > 1, and let s be a positive divisor of $n-1$. Suppose there is an integer a satisfying*

$$a^{n-1} \equiv 1 \bmod n,$$
$$\gcd(a^{(n-1)/q}-1,\ n) = 1 \textit{ for each prime } q \textit{ dividing } s.$$

Then every prime p dividing n is $1 \bmod s$, *and if* $s > \sqrt{n}-1$ *then n is prime.*

We omit the proof of this theorem, as it can easily be deduced from the proof of a similar theorem below (cf. (4.7)), by replacing the role that is played by $E(\mathbb{Z}/p\mathbb{Z})$ in that proof by $(\mathbb{Z}/p\mathbb{Z})^*$ here. Instead, let us consider how this theorem can be employed to prove the primality of a probable prime n.

Apparently, to prove the primality of n by means of this theorem, we need a factor s of $n-1$, such that $s > \sqrt{n}-1$, and such that the complete factorization of s is known. Given such an s, we simply select non-zero integers $a \in \mathbb{Z}/n\mathbb{Z}$ at random until both conditions are satisfied. For such a, the first condition must be satisfied, unless n is composite. The second condition might cause more problems, but if n is prime then $q-1$ out of q choices for a will satisfy it, for a fixed q dividing s. Therefore, if an a satisfying both conditions is not found after a reasonable number of trials, we begin to suspect that n is probably not prime, and we subject n to some probabilistic compositeness tests as in (4.3).

The main disadvantage of this method is that an s as above is not easy to find, because factoring $n-1$ is usually hard. If n is prime, then $n-1$ is the order of $(\mathbb{Z}/p\mathbb{Z})^*$ for the only prime p dividing n; in the next section we will randomize this order by replacing $(\mathbb{Z}/p\mathbb{Z})^*$ by $E(\mathbb{Z}/p\mathbb{Z})$. For other generalizations of this method we refer to the extensive literature on this subject [11,39,54,60,68].

(4.5) *The Jacobi sum test (cf. [3,16]).* The first primality test that could routinely handle numbers of a few hundred decimal digits was the Cohen-Lenstra version [16] of the primality test by Adleman, Pomerance, and Rumely [3]. It runs in time $(\log n)^{O(\log\log\log n)}$, which makes it the fastest deterministic primality test. Details concerning the implementation of this algorithm can be found in [15].

(4.6) Primality testing using elliptic curves

We assume that the reader is familiar with the material and the notation introduced in Section (1.2). In this section we discuss the consequences of the following analogue of Theorem (4.4).

(4.7) Theorem. *Let $n > 1$ be an integer with* $\gcd(n, 6) = 1$. *Let $E = E_{a,b}$ be an elliptic curve modulo n (cf. (1.9)), and let m and s be positive integers with s dividing m. Suppose there is a point $P \in (V_n - \{O\}) \cap E(\mathbb{Z}/n\mathbb{Z})$ (cf. (1.10)) satisfying*

$m \cdot P = O$ (cf. (1.12)),

$(m/q) \cdot P$ is defined and different from O, for each prime q dividing s,

where in (1.12) we choose the a that is used in the definition of the elliptic curve $E_{a,b}$. Then $\#E(\mathbb{F}_p) \equiv 0 \bmod s$ *for every prime p dividing n (cf. (1.4), (1.9)), and if* $s > (n^{1/4}+1)^2$ *then n is prime.*

Proof. Let p be a prime dividing n, and let $Q = (m/s) \cdot P$. Because $P_p \in E(\mathbb{F}_p)$, we have that $Q_p \in E(\mathbb{F}_p)$, according to (1.9) and (1.11). From $m \cdot P = O$ it follows that $s \cdot Q = O$, so that, according to (1.11), the order of Q_p divides s. But for a prime q dividing s we have that $(s/q) \cdot Q_p = (m/q) \cdot P_p \neq O_p$, because $(m/q) \cdot P \neq O$ (cf. (1.11)). The order of Q_p is therefore not a divisor of s/q, for any prime q dividing s, so this order equals s, and we find that $\#E(\mathbb{F}_p) \equiv 0 \bmod s$.

In (1.6) we have seen that $\#E(\mathbb{F}_p) = p+1-t$, for some integer t with $|t| \leq 2\sqrt{p}$ (Hasse's inequality). It follows that $(p^{1/2}+1)^2 \geq \#E(\mathbb{F}_p)$. With $s > (n^{1/4}+1)^2$ and $\#E(\mathbb{F}_p) \equiv 0 \bmod s$ this implies that $p > \sqrt{n}$, for any prime p dividing n, so that n must be prime. This proves the theorem.

(4.8) Remark. The proof of theorem (4.4) follows the same lines, with $p-1$ replacing m.

Theorem (4.7) can be used to prove the primality of a probable prime n in the following way, an idea that is due to Goldwasser and Kilian (cf. [26]); for earlier applications of elliptic curves to primality tests see [9] and [14].

(4.9) *Outline of the primality test.* First, select an elliptic curve E over $\mathbb{Z}/n\mathbb{Z}$ and an integer m, such that $m = \#E(\mathbb{Z}/n\mathbb{Z})$ if n is prime, and such that m can be written as kq for a small integer $k > 1$ and probable prime $q > (n^{1/4}+1)^2$. Next, find a point $P \in E(\mathbb{Z}/n\mathbb{Z})$ satisfying the requirements in Theorem (4.7) with $s = q$, on the assumption that q is prime. This is done as follows. First, use (1.13) to find a random point $P \in E(\mathbb{Z}/n\mathbb{Z})$. Next, compute $(m/q) \cdot P = k \cdot P$; if $k \cdot P$ is undefined, we find a non-trivial divisor of n, which is exceedingly unlikely. If $k \cdot P = O$, something that happens with probability $< 1/2$ if n is prime, select a new P and try again. Otherwise, verify that $q \cdot (k \cdot P) = m \cdot P = O$, which must be the case if n is prime, because in that case $\#E(\mathbb{Z}/n\mathbb{Z}) = m$. The existence of P now proves that n is prime if q is prime, by (4.7). Finally, the primality of q is proved recursively.

We will discuss two methods to select the pair E, m.

(4.10) *The random curve test (cf. [26]).* Select a random elliptic curve E modulo n, and attempt to apply the division points method mentioned in (1.8) to E. If this algorithm works, then it produces an integer m that is equal to $\#E(\mathbb{Z}/n\mathbb{Z})$ if n is prime. If the algorithm does not work, then n is not prime, because it is guaranteed to work for prime n.

This must be repeated until m satisfies the requirements in (4.9).

(4.11) *The running time of the random curve test.* First remark that the recursion depth is $O(\log n)$, because $k > 1$ so that $q \le (\sqrt{n}+1)^2/2$ (cf. (1.6)). Now consider how often a random elliptic curve E modulo n has to be selected before a pair E, m as in (4.9) is found. Assuming that n is prime, $\#E(\mathbb{Z}/n\mathbb{Z})$ behaves approximately like a random integer near n, according to Proposition (1.7). Therefore, the probability that $m = kq$ with k and q as in (4.9) should be of the order $(\log n)^{-1}$, so that $O(\log n)$ random choices for E should suffice to find a pair E, m.

The problem is to *prove* that this probability is indeed of the order $(\log n)^{-c}$, for a positive constant c. This can be shown to be the case if we suppose that there is a positive constant c such that for all $x \in \mathbb{R}_{\ge 2}$ the number of primes between x and $x+\sqrt{2x}$ (cf. (1.6)) is of the order $\sqrt{x}(\log x)^{-c}$. Under this assumption, the random curve test proves the primality of n in expected time $O((\log n)^{9+c})$ (cf. [26]).

By a theorem of Heath-Brown, the assumption is *on the average* correct. In [26] it is shown that this implies that the fraction of primes n for which the algorithm runs in expected time polynomial in $\log n$, is at least $1-O(2^{-l^{1/\log\log l}})$, where $l = \lfloor \log_2 n \rfloor$. In their original algorithm, however, Goldwasser and Kilian only allow $k = 2$, i.e., they wait for an elliptic curve E such that $\#E(\mathbb{Z}/n\mathbb{Z}) = 2q$. By allowing more values for k, the fraction of primes for which the algorithm runs in polynomial time can be shown to be much higher [52] (cf. [2]). For a primality test that runs in expected polynomial time for all n see (4.14) below.

Because the random curve test makes use of the division points method, it is not considered to be of much practical value. A practical version of (4.9) is the following test, due to Atkin [4].

(4.12) *The complex multiplication test [4].* Here one does not start by selecting E, but by selecting the complex multiplication field L of E (cf. (1.8)). The field L can be used to calculate m, and only if m is of the required form kq (cf. (4.9)), one determines the pair a, b defining E.

This is done as follows. Let Δ be a negative fundamental discriminant ≤ -7, i.e., $\Delta \equiv 0$ or 1 mod 4 and there is no $s \in \mathbb{Z}_{>1}$ such that Δ/s^2 is a discriminant. Denote by L the imaginary quadratic field $\mathbb{Q}(\sqrt{\Delta})$ and by $A = \mathbb{Z}[(\Delta+\sqrt{\Delta})/2]$ its ring of integers (cf. (1.8)). We try to find ν with $\nu\bar{\nu} = n$ in A. It is known that $(\frac{\Delta}{n}) = 1$ and $(\frac{n}{p}) = 1$ for the odd prime divisors p of Δ are necessary conditions for the existence of ν, where we assume that $\gcd(n, 2\Delta) = 1$. If these conditions are not satisfied, select another Δ and try again. Otherwise, compute an integer $b \in \mathbb{Z}$ with $b^2 \equiv \Delta \bmod n$. This can for instance be done using a probabilistic method for finding the roots of a polynomial over a finite field [31, Section 4.6.2], where we assume that n is prime; for this algorithm to work, we do not need a proof that n is prime. If necessary add n to b to achieve that b and Δ have the same parity. We then have that $b^2 \equiv \Delta \bmod 4n$, and that $\underline{n} = \mathbb{Z}n+\mathbb{Z}((b+\sqrt{\Delta})/2)$ is a prime ideal in A with $\underline{n}\cdot\overline{\underline{n}} = A\cdot n$. Attempt to solve $\underline{n} = A\cdot\nu$ by looking for a shortest non-zero vector μ in the lattice $\underline{n}$. If $\mu\bar{\mu} = n$ then take $\nu = \mu$; otherwise $\nu\bar{\nu} = n$ is unsolvable.

Finding μ, and ν if it exists, can for example be done by means of the reduction algorithm (1.15). With b as above, consider the form (a, b, c) with $a = n$ and $c = (b^2-\Delta)/(4n)$. For any two integers x and y the value $ax^2+bxy+cy^2$ of the form at x, y equals $|xn+y((b+\sqrt{\Delta})/2)|^2/n$, the square of the absolute value of the corresponding element of $\underline{n}$

divided by n. It follows that μ can be determined by computing integers x and y for which $ax^2+bxy+cy^2$ is minimal. More in particular, it follows that ν with $\nu\bar{\nu}=n$ exists if and only if there exist integers x and y for which the form assumes the value 1.

Because $\gcd(n, 2\Delta)=1$, we have that $\gcd(n, b)=1$, so that the form (a, b, c) is primitive, which makes the theory of (1.14) applicable. Apply the reduction algorithm (1.15) to (a, b, c); obviously the set $\{ax^2+bxy+cy^2 : x, y \in \mathbb{Z}\}$ does not change in the course of the algorithm. Because a reduced form assumes its minimal value for $x = 1$ and $y = 0$, the x and y for which the original form (a, b, c) is minimized now follow, as mentioned in the last paragraph of (1.15). The shortest non-zero vector $\mu \in \underline{n}$ is then given by $xn+y((b+\sqrt{\Delta})/2)$. Now remark that $ax^2+bxy+cy^2=1$ if and only if the reduced form equivalent to (a, b, c) is the unit element 1_Δ. Therefore, if the reduced form equals 1_Δ, put $\nu=\mu$; otherwise select another Δ and try again because ν with $\nu\bar{\nu}=n$ does not exist.

Assuming that ν has been computed, consider $m=(\nu-1)(\bar{\nu}-1)$, and $m'=(-\nu-1)(-\bar{\nu}-1)$. If neither m nor m' is of the required form, select another Δ and try again. (For $n \equiv 1 \bmod 3$ ($n \equiv 1 \bmod 4$) we should also include $\Delta=-3$ ($\Delta=-4$), as it gives rise to six (four) pairs E, m (cf. [38]).) Supposing that $m = kq$, an elliptic curve E such that $\#E(\mathbb{Z}/n\mathbb{Z})=m$ if n is prime can be constructed as a function of a zero in $\mathbb{Z}/n\mathbb{Z}$ of a certain polynomial $F_\Delta \in \mathbb{Z}[X]$. To determine this polynomial F_Δ define, for a complex number z with $\mathrm{Im}\ z > 0$,

$$j(z)=\frac{(1+240\sum_{k=1}^{\infty}\frac{k^3q^k}{1-q^k})^3}{q\cdot\prod_{k=1}^{\infty}(1-q^k)^{24}}$$

where $q=e^{2\pi iz}$. Then

$$F_\Delta=\prod_{(a,\,b)}(X-j(\frac{b+\sqrt{\Delta}}{2a}))$$

with (a, b) ranging over the set of reduced forms of discriminant Δ, see (1.15). The degree of F_Δ equals the class number of L, and is therefore $\approx \sqrt{|\Delta|}$. As these polynomials depend only on Δ, they should be tabulated. More about the computation of these polynomials can be found in [65, Sections 125-133].

Compute a zero $j \in \mathbb{Z}/n\mathbb{Z}$ of F_Δ over $\mathbb{Z}/n\mathbb{Z}$, and let c be a quadratic non-residue modulo n (assuming that n is prime). Put $k=j/(1728-j)$, then k is well-defined and non-zero because $\Delta \le -7$. Finally, choose E as the elliptic curve $E_{3k,2k}$ or $E_{3kc^2,2kc^3}$ in such a way that $\#E(\mathbb{Z}/n\mathbb{Z})=m$ if n is prime; the right choice can be made as described at the end of (1.8).

The computation of ν is dominated by the computation of $\sqrt{\Delta} \bmod n$ and therefore takes expected time $O((\log n)^3)$ [31, Section 4.6.2]; with fast multiplication techniques this can be reduced to $O((\log n)^{2+\epsilon})$. It is reasonable to expect that one has to try $O((\log n)^{2+\epsilon})$ values of Δ before m (or m') has the required form, so that we may assume that the final Δ is $O((\log n)^{2+\epsilon})$. For a reduced form (a, b) and $z=(b+\sqrt{\Delta})/(2a)$, $q=e^{2\pi iz}$, one can show that $|j(z)-q^{-1}|<2100$; and if, with the same notation, the summation in the definition of $j(z)$ is terminated after K terms and the product after K factors, then the error is $O(K^3q^K)$. To bound the coefficients of F_Δ we notice that $j(z)$ can only be large for small a. Since the number of reduced forms (a, b) with a fixed a is bounded by the number of divisors of a, there cannot be too many large $j(z)$'s. It follows that one polynomial F_Δ can be computed in time $|\Delta|^{2+o(1)}=O((\log n)^{4+\epsilon})$; it is likely that it can be done in time $|\Delta|^{1+o(1)}=O((\log n)^{2+\epsilon})$ using fast multiplication techniques. Assuming that n is prime, a zero of F_Δ can be computed in time $O((\deg F_\Delta)^2(\log n)^3)=O((\log n)^{5+\epsilon})$ (ordinary), or $O((\deg F_\Delta)(\log n)^{2+\epsilon})=O((\log n)^{3+\epsilon})$ (fast). Heuristically, it follows that the whole primality proof takes time $O((\log n)^{6+\epsilon})$, which includes the $O(\log n)$ factor for the recursion. The method has proved to be quite practical.

With fast multiplication techniques one gets $O((\log n)^{5+\epsilon})$. As J.O. Shallit observed the latter result can be improved to $O((\log n)^{4+\epsilon})$, if we only use Δ's that can be written as the product of some small primes; to compute the square roots modulo n of the Δ's, it then suffices to compute the square roots of those small primes, which can be done at the beginning of the computation.

(4.13) Remark. It should be noted that both algorithms based on (4.9), if successful, yield a certificate of primality that can be checked in polynomial time.

(4.14) *The abelian variety test (cf. [2]).* A primality test that runs in expected polynomial time for all n can be obtained by using abelian varieties of higher dimensions, as claimed by Adleman and Huang in [2]. We give the idea of their algorithm; our description is not complete and serves only as an illustration of the method.

Abelian varieties are higher dimensional analogues of elliptic curves. By definition, an abelian variety over a field K is a projective group variety A over K. The set of points $A(K)$ of an abelian variety over a field K has the structure of an abelian group. Moreover, if $K = \mathbb{F}_p$ then $\#A(\mathbb{F}_p) = p^g + O((4p)^{g-1/2})$, where g is the dimension of A. One-dimensional abelian varieties are the same as elliptic curves.

Examples of abelian varieties over $\mathbb{F}_p$, for an odd prime p, can be obtained as follows. Let f be a monic square-free polynomial of odd degree $2g+1$ over $\mathbb{F}_p$, and consider the *hyperelliptic curve* $y^2 = f(x)$ over $\mathbb{F}_p$. Then the Jacobian A of this curve is an abelian variety of dimension g over $\mathbb{F}_p$. The elements of $A(\mathbb{F}_p)$ can in this case be regarded as pairs (a, b) with $a, b \in \mathbb{F}_p[T]$, a monic, $b^2 \equiv f \bmod a$ and degree$(b) <$ degree$(a) \leq g$. Note the analogy with the definition of reduced forms in (1.14) and (1.15), with f playing the role of Δ. The composition in the abelian group $A(\mathbb{F}_p)$ can be done as in (1.16) (cf. [13]).

The abelian variety test proceeds in a similar way as the random curve test, but with $g = 1$ replaced by $g = 2$. The order of $A(\mathbb{F}_p)$ is then in an interval of length $O(x^{3/4})$ around $x = p^2$. The main difference with the random curve test is that it can be *proved* that this interval contains sufficiently many primes [29]. The problem of proving the primality of a probable prime n is then reduced, in expected polynomial time, to proving the primality of a number of order of magnitude n^2. Although the recursion obviously goes in the wrong direction, it has been claimed in [2] that, after a few iterations, we may expect to hit upon a number whose primality can be proved in polynomial time by means of the random curve test (4.10).

References

1. L.M. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications*, Proc. 20th IEEE Found. Comp. Sci. Symp. (1979), 55-60.

2. L.M. Adleman, M.A. Huang, *Recognizing primes in random polynomial time*, Proc. 19th Annual ACM Symp. on Theory of Computing (1987), 462-469.

3. L.M. Adleman, C. Pomerance, R.S. Rumely, *On distinguishing prime numbers from composite numbers*, Ann. of Math. **117** (1983), 173-206.

4. A.O.L. Atkin, personal communication.

5. E. Bach, *Analytic methods in the analysis and design of number-theoretic algorithms*, MIT Press (1985).

6. E. Bach, J. Shallit, *Factoring with cyclotomic polynomials*, Proceedings 26th FOCS (1985), 443-450.

7. T. Beth, N. Cot, I. Ingemarsson (eds), *Advances in cryptology*, Springer Lecture Notes in Computer Science, Vol. 209 (1985).

8. Z.I. Borevič, I.R. Šafarevič, *Teorija čisel*, Moscow 1964. Translated into German, English and French.

9. W. Bosma, *Primality testing using elliptic curves*, report 85-12, Mathematisch Instituut, Universiteit van Amsterdam 1985.

10. R.P. Brent, *Some integer factorization algorithms using elliptic curves*, research report CMA-R32-85, The Australian National University, Canberra 1985.

11. J.Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, S.S. Wagstaff, Jr., *Factorizations of $b^n \pm 1$, $b = 2,3,5,6,7,10,11,12$ up to high powers*, Contemporary Mathematics **22**, Providence: A.M.S., 1983.

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

13. D.G. Cantor, *Computing in the Jacobian of a hyperelliptic curve*, Math. Comp., **48** (1987), 95-101.

14. D.V. Chudnovsky, G.V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*, Advances in Appl. Math. **7** (1986), 187-237.

15. H. Cohen, A.K. Lenstra, *Implementation of a new primality test*, Math. Comp. **48** (1987), 103-121.

16. H. Cohen, H.W. Lenstra, Jr., *Primality testing and Jacobi sums*, Math. Comp. **42** (1984), 297-330.

17. D. Coppersmith, *Fast evaluation of logarithms in fields of characteristic two*, IEEE Trans. Inform. Theory, IT-30 (1984), 587-594.

18. D. Coppersmith, A.M. Odlyzko, R. Schroeppel, *Discrete logarithms in GF(p)*, Algorithmica **1** (1986), 1-15.

19. D. Coppersmith, S. Winograd, *Matrix multiplication via Behrend's Theorem*, IBM technical report, 1986.

20. J.A. Davis, D.B. Holdridge, *Factorization using the quadratic sieve algorithm*, Sandia National Laboratories Tech Rpt. SAND 83-1346 (Dec. 1983).

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

22. M. Deuring, *Die Typen der Multiplikatorenringe elliptischer Funktionenkörper*, Abh. Math. Sem. Hansischen Univ. **14** (1941), 197-272.

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

24. T. ElGamal, *A subexponential-time algorithm for computing discrete logarithms over $GF(p^2)$*, IEEE Trans. Inform. Theory, IT-31 (1985), 473-481.

25. J.B. Friedlander, J.C. Lagarias, *On the distribution in short intervals of integers having no large prime factor*, J. Number Theory **25** (1987), 249-273.

26. S. Goldwasser, J. Kilian, *Almost all primes can be quickly certified*, Proc. 18th Annual ACM Symp. on Theory of Computing (1986), 316-329.

27. G.H. Hardy, E.M. Wright, *An introduction to the theory of numbers*, fifth edition, Oxford Univ. Press, Oxford, 1979.

28. K. Ireland, M. Rosen, *A classical introduction to modern number theory*, Graduate Texts in Math. **84**, Springer-Verlag, New York 1982.

29. H. Iwaniec, M. Jutila, *Primes in short intervals*, Ark. Mat. **17** (1979), 167-176.

30. R. Kannan, A. Bachem, *Polynomial algorithms for computing the Smith and Hermite normal forms of an integer matrix*, SIAM J. Comput. **8** (1979), 499-507.

31. D.E. Knuth, *The art of computer programming*, vol. 2, *Seminumerical algorithms*, second edition. Reading: Addison-Wesley (1981).

32. D.E. Knuth, *The art of computer programming*, vol. 3, *Sorting and searching*. Reading: Addison-Wesley (1973).

33. J.C. Lagarias, *Worst-case complexity bounds for algorithms in the theory of integral quadratic forms*, J. of Algorithms **1** (1980), 142-186.

34. J.C. Lagarias, H.L. Montgomery, A.M. Odlyzko, *A bound for the least prime ideal in the Chebotarev density theorem*, Inventiones Math. **54** (1975), 137-144.

35. S. Lang, *Algebraic number theory*, Addison-Wesley, Reading, 1970.

36. A.K. Lenstra, *Fast and rigorous factorization under the generalized Riemann hypothesis*, technical report 87-007, Department of Computer Science, The University of Chicago, 1987.

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

38. H.W. Lenstra, Jr., *Elliptic curves and number-theoretic algorithms*, Universiteit van Amsterdam, report 86-19; to appear in Proceedings International Congress of Mathematicians 1986, Berkeley.

39. H.W. Lenstra, Jr., R. Tijdeman (eds), *Computational methods in number theory*, Math. Centre Tracts **154/155**, Mathematisch Centrum, Amsterdam 1982.

40. J.L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Trans. Inform. Theory **15** (1969), 122-127.

41. G.L. Miller, *Riemann's hypothesis and tests for primality*, J. Comput. System Sci. **13** (1976), 300-317.

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

43. M.A. Morrison, J. Brillhart, *A method of factoring and the factorization of F_7*, Math. Comp. **29** (1975), 183-205.

44. A.M. Odlyzko, *Discrete logarithms and their cryptographic significance*, pp 224-314 in: [7]

45. H.C. Pocklington, *The determination of the prime and composite nature of large numbers by Fermat's theorem*, Proc. Cambridge Philos. Soc **18** (1914-16), 29-30.

46. S.C. Pohlig, M.E. Hellman, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Trans. Inform. Theory, IT-24 (1978), 106-110.

47. J.M. Pollard, *Theorems on factorization and primality testing*, Proc. Cambridge Philos. Soc. **76** (1974), 521-528.

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

49. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, pp 89-139 in: [39]

50. C. Pomerance, *The quadratic sieve factoring algorithm*, pp 169-182 in: [7].

51. C. Pomerance, *Fast, rigorous factorization and discrete logarithm algorithms*, in: T. Nishizeki, H. Wilf (eds), *Discrete algorithms and complexity*, Proceedings of the Japan-US joint seminar on discrete algorithms and complexity theory, Academic Press, to appear.

52. C. Pomerance, personal communication.

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

54. H. Riesel, *Prime numbers and computer methods for factorization*, Progr. Math. 57, Birkhäuser, Boston 1985.

55. R.L. Rivest, A. Shamir, L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Comm. ACM **21** (1978), 120-126.

56. C.P. Schnorr, H.W. Lenstra, Jr., *A Monte Carlo factoring algorithm with linear storage*, Math. Comp. **43** (1984), 289-311.

57. A. Schönhage, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Informatica **1** (1971), 139-144.

58. R.J. Schoof, *Quadratic fields and factorization*, pp 235-286 in: [39].

59. R.J. Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Math. Comp. **44** (1985), 483-494.

60. J.L. Selfridge, M.C. Wunderlich, *An efficient algorithm for testing large numbers for primality*, pp. 109-120 in: *Proc. Fourth Manitoba Conf. Numerical Math.*, University of Manitoba, Congressus Numerantium XII, Utilitas Math. Winnipeg 1975.

61. M. Seysen, *A probabilistic factorization algorithm with quadratic forms of negative discriminant*, Math. Comp. **48** (1987), 757-780.

62. J.H. Silverman, *The arithmetic of elliptic curves*, Graduate Texts in Math. **106**, Springer-Verlag, New York 1986.

63. R.D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987), 329-339.

64. R. Solovay, V. Strassen, *A fast Monte-Carlo test for primality*, SIAM J. Comput. **6** (1977), 84-85; erratum, ibid., **7** (1978), 118.

65. H. Weber, *Lehrbuch der Algebra, Bd. 3*, Vieweg, Braunschweig, 1908.

66. A.E. Western, J.C.P. Miller, *Tables of indices and primitive roots*, Royal Society Mathematical Tables, vol. 9, Cambridge Univ. Press, 1968.

67. D.H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory, IT-32 (1986), 54-62.

68. H.C. Williams, *Primality testing on a computer*, Ars Combin. **5** (1978), 127-185.

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