Chapter 4

# Public-Key Parameters

## Contents in Brief



## 4.1 Introduction

The efficient generation of public-key parameters is a prerequisite in public-key systems. A specific example is the requirement of a prime number $p$ to define a finite field $\mathbb{Z}_p$ for use in the Diffie-Hellman key agreement protocol and its derivatives (§12.6). In this case, an element of high order in $\mathbb{Z}_p^*$ is also required. Another example is the requirement of primes $p$ and $q$ for an RSA modulus $n = pq$ (§8.2). In this case, the prime must be of sufficient size, and be "random" in the sense that the probability of any particular prime being selected must be sufficiently small to preclude an adversary from gaining advantage through optimizing a search strategy based on such probability. Prime numbers may be required to have certain additional properties, in order that they do not make the associated cryptosystems susceptible to specialized attacks. A third example is the requirement of an irreducible polynomial $f(x)$ of degree $m$ over the finite field $\mathbb{Z}_p$ for constructing the finite field $\mathbb{F}_{p^m}$. In this case, an element of high order in $\mathbb{F}_{p^m}^*$ is also required.

### Chapter outline

The remainder of §4.1 introduces basic concepts relevant to prime number generation and summarizes some results on the distribution of prime numbers. Probabilistic primality tests, the most important of which is the Miller-Rabin test, are presented in §4.2. True primality tests by which arbitrary integers can be proven to be prime are the topic of §4.3; since these tests are generally more computationally intensive than probabilistic primality tests, they are not described in detail. §4.4 presents four algorithms for generating prime numbers, strong primes, and provable primes. §4.5 describes techniques for constructing irreducible and primitive polynomials, while §4.6 considers the production of generators and elements of high orders in groups. §4.7 concludes with chapter notes and references.

### 4.1.1 Approaches to generating large prime numbers

To motivate the organization of this chapter and introduce many of the relevant concepts, the problem of generating large prime numbers is first considered. The most natural method is to generate a random number $n$ of appropriate size, and check if it is prime. This can be done by checking whether $n$ is divisible by any of the prime numbers $\leq \sqrt{n}$. While more efficient methods are required in practice, to motivate further discussion consider the following approach:

1. Generate as *candidate* a random odd number $n$ of appropriate size.
2. Test $n$ for primality.
3. If $n$ is composite, return to the first step.

A slight modification is to consider candidates restricted to some *search sequence* starting from $n$; a trivial search sequence which may be used is $n, n+2, n+4, n+6, \ldots$. Using specific search sequences may allow one to increase the expectation that a candidate is prime, and to find primes possessing certain additional desirable properties *a priori*.

In step 2, the test for primality might be either a test which *proves* that the candidate is prime (in which case the outcome of the generator is called a *provable prime*), or a test which establishes a weaker result, such as that $n$ is "probably prime" (in which case the outcome of the generator is called a *probable prime*). In the latter case, careful consideration must be given to the exact meaning of this expression. Most so-called *probabilistic primality tests* are absolutely correct when they declare candidates $n$ to be composite, but do not provide a mathematical proof that $n$ is prime in the case when such a number is declared to be "probably" so. In the latter case, however, when used properly one may often be able to draw conclusions more than adequate for the purpose at hand. For this reason, such tests are more properly called *compositeness tests* than probabilistic primality tests. True primality tests, which allow one to conclude with mathematical certainty that a number is prime, also exist, but generally require considerably greater computational resources.

While (true) primality tests can determine (with mathematical certainty) whether a typically random candidate number is prime, other techniques exist whereby candidates $n$ are specially constructed such that it can be established by mathematical reasoning whether a candidate actually is prime. These are called *constructive prime generation* techniques.

A final distinction between different techniques for prime number generation is the use of randomness. Candidates are typically generated as a function of a random input. The technique used to judge the primality of the candidate, however, may or may not itself use random numbers. If it does not, the technique is *deterministic*, and the result is reproducible; if it does, the technique is said to be *randomized*. Both deterministic and randomized probabilistic primality tests exist.

In some cases, prime numbers are required which have additional properties. For example, to make the extraction of discrete logarithms in $\mathbb{Z}_p^*$ resistant to an algorithm due to Pohlig and Hellman (§3.6.4), it is a requirement that $p-1$ have a large prime divisor. Thus techniques for generating public-key parameters, such as prime numbers, of special form need to be considered.

### 4.1.2 Distribution of prime numbers

Let $\pi(x)$ denote the number of primes in the interval $[2, x]$. The prime number theorem (Fact 2.95) states that $\pi(x) \sim \frac{x}{\ln x}$.[1] In other words, the number of primes in the interval

[1] If $f(x)$ and $g(x)$ are two functions, then $f(x) \sim g(x)$ means that $\lim_{x\to\infty} \frac{f(x)}{g(x)} = 1$.

$[2, x]$ is approximately equal to $\frac{x}{\ln x}$. The prime numbers are quite uniformly distributed, as the following three results illustrate.

**4.1 Fact** (*Dirichlet theorem*) If $\gcd(a, n) = 1$, then there are infinitely many primes congruent to $a$ modulo $n$.

A more explicit version of Dirichlet's theorem is the following.

**4.2 Fact** Let $\pi(x, n, a)$ denote the number of primes in the interval $[2, x]$ which are congruent to $a$ modulo $n$, where $\gcd(a, n) = 1$. Then

$$\pi(x, n, a) \sim \frac{x}{\phi(n) \ln x}.$$

In other words, the prime numbers are roughly uniformly distributed among the $\phi(n)$ congruence classes in $\mathbb{Z}_n^*$, for any value of $n$.

**4.3 Fact** (*approximation for the nth prime number*) Let $p_n$ denote the $n$th prime number. Then $p_n \sim n \ln n$. More explicitly,

$$n \ln n \;<\; p_n \;<\; n(\ln n + \ln \ln n) \quad \text{for } n \geq 6.$$

## 4.2 Probabilistic primality tests

The algorithms in this section are methods by which arbitrary positive integers are tested to provide partial information regarding their primality. More specifically, probabilistic primality tests have the following framework. For each odd positive integer $n$, a set $W(n) \subset \mathbb{Z}_n$ is defined such that the following properties hold:

(i) given $a \in \mathbb{Z}_n$, it can be checked in deterministic polynomial time whether $a \in W(n)$;
(ii) if $n$ is prime, then $W(n) = \emptyset$ (the empty set); and
(iii) if $n$ is composite, then $\#W(n) \geq \frac{n}{2}$.

**4.4 Definition** If $n$ is composite, the elements of $W(n)$ are called *witnesses* to the compositeness of $n$, and the elements of the complementary set $L(n) = \mathbb{Z}_n - W(n)$ are called *liars*.

A probabilistic primality test utilizes these properties of the sets $W(n)$ in the following manner. Suppose that $n$ is an integer whose primality is to be determined. An integer $a \in \mathbb{Z}_n$ is chosen at random, and it is checked if $a \in W(n)$. The test outputs "composite" if $a \in W(n)$, and outputs "prime" if $a \notin W(n)$. If indeed $a \in W(n)$, then $n$ is said to *fail the primality test for the base* $a$; in this case, $n$ is surely composite. If $a \notin W(n)$, then $n$ is said to *pass the primality test for the base* $a$; in this case, no conclusion with absolute certainty can be drawn about the primality of $n$, and the declaration "prime" may be incorrect.[2]

Any single execution of this test which declares "composite" establishes this with certainty. On the other hand, successive independent runs of the test all of which return the answer "prime" allow the confidence that the input is indeed prime to be increased to whatever level is desired — the cumulative probability of error is multiplicative over independent trials. If the test is run $t$ times independently on the composite number $n$, the probability that $n$ is declared "prime" all $t$ times (i.e., the probability of error) is at most $(\frac{1}{2})^t$.

[2]This discussion illustrates why a probabilistic primality test is more properly called a *compositeness* test.

**4.5 Definition** An integer $n$ which is believed to be prime on the basis of a probabilistic primality test is called a *probable prime*.

Two probabilistic primality tests are covered in this section: the Solovay-Strassen test (§4.2.2) and the Miller-Rabin test (§4.2.3). For historical reasons, the Fermat test is first discussed in §4.2.1; this test is not truly a probabilistic primality test since it usually fails to distinguish between prime numbers and special composite integers called Carmichael numbers.

### 4.2.1 Fermat's test

Fermat's theorem (Fact 2.127) asserts that if $n$ is a prime and $a$ is any integer, $1 \le a \le n-1$, then $a^{n-1} \equiv 1 \pmod{n}$. Therefore, given an integer $n$ whose primality is under question, finding any integer $a$ in this interval such that this equivalence is not true suffices to prove that $n$ is composite.

**4.6 Definition** Let $n$ be an odd composite integer. An integer $a$, $1 \le a \le n-1$, such that $a^{n-1} \not\equiv 1 \pmod{n}$ is called a *Fermat witness* (to compositeness) for $n$.

Conversely, finding an integer $a$ between 1 and $n-1$ such that $a^{n-1} \equiv 1 \pmod{n}$ makes $n$ appear to be a prime in the sense that it satisfies Fermat's theorem for the base $a$. This motivates the following definition and Algorithm 4.9.

**4.7 Definition** Let $n$ be an odd composite integer and let $a$ be an integer, $1 \le a \le n-1$. Then $n$ is said to be a *pseudoprime to the base* $a$ if $a^{n-1} \equiv 1 \pmod{n}$. The integer $a$ is called a *Fermat liar* (to primality) for $n$.

**4.8 Example** (*pseudoprime*) The composite integer $n = 341$ $(= 11 \times 31)$ is a pseudoprime to the base 2 since $2^{340} \equiv 1 \pmod{341}$. □

---

**4.9 Algorithm** Fermat primality test

---

FERMAT($n$,$t$)
INPUT: an odd integer $n \ge 3$ and security parameter $t \ge 1$.
OUTPUT: an answer "prime" or "composite" to the question: "Is $n$ prime?"

1. For $i$ from 1 to $t$ do the following:
   1.1 Choose a random integer $a$, $2 \le a \le n-2$.
   1.2 Compute $r = a^{n-1} \bmod n$ using Algorithm 2.143.
   1.3 If $r \ne 1$ then return("composite").
2. Return("prime").

---

If Algorithm 4.9 declares "composite", then $n$ is certainly composite. On the other hand, if the algorithm declares "prime" then no proof is provided that $n$ is indeed prime. Nonetheless, since pseudoprimes for a given base $a$ are known to be rare, Fermat's test provides a correct answer on *most* inputs; this, however, is quite distinct from providing a correct answer most of the time (e.g., if run with different bases) on *every* input. In fact, it does not do the latter because there are (even rarer) composite numbers which are pseudoprimes to *every* base $a$ for which $\gcd(a, n) = 1$.

**4.10 Definition** A *Carmichael number* $n$ is a composite integer such that $a^{n-1} \equiv 1 \pmod{n}$ for all integers $a$ which satisfy $\gcd(a, n) = 1$.

If $n$ is a Carmichael number, then the only Fermat witnesses for $n$ are those integers $a$, $1 \le a \le n-1$, for which $\gcd(a, n) > 1$. Thus, if the prime factors of $n$ are all large, then with high probability the Fermat test declares that $n$ is "prime", even if the number of iterations $t$ is large. This deficiency in the Fermat test is removed in the Solovay-Strassen and Miller-Rabin probabilistic primality tests by relying on criteria which are stronger than Fermat's theorem.

This subsection is concluded with some facts about Carmichael numbers. If the prime factorization of $n$ is known, then Fact 4.11 can be used to easily determine whether $n$ is a Carmichael number.

**4.11 Fact** (*necessary and sufficient conditions for Carmichael numbers*) A composite integer $n$ is a Carmichael number if and only if the following two conditions are satisfied:

(i) $n$ is square-free, i.e., $n$ is not divisible by the square of any prime; and
(ii) $p - 1$ divides $n - 1$ for every prime divisor $p$ of $n$.

A consequence of Fact 4.11 is the following.

**4.12 Fact** Every Carmichael number is the product of at least three distinct primes.

**4.13 Fact** (*bounds for the number of Carmichael numbers*)

(i) There are an infinite number of Carmichael numbers. In fact, there are more than $n^{2/7}$ Carmichael numbers in the interval $[2, n]$, once $n$ is sufficiently large.
(ii) The best upper bound known for $C(n)$, the number of Carmichael numbers $\le n$, is:

$$C(n) \le n^{1-\{1+o(1)\}\ln\ln\ln n/\ln\ln n} \quad \text{for } n \to \infty.$$

The smallest Carmichael number is $n = 561 = 3 \times 11 \times 17$. Carmichael numbers are relatively scarce; there are only 105212 Carmichael numbers $\le 10^{15}$.

## 4.2.2 Solovay-Strassen test

The Solovay-Strassen probabilistic primality test was the first such test popularized by the advent of public-key cryptography, in particular the RSA cryptosystem. There is no longer any reason to use this test, because an alternative is available (the Miller-Rabin test) which is both more efficient and always at least as correct (see Note 4.33). Discussion is nonetheless included for historical completeness and to clarify this exact point, since many people continue to reference this test.

Recall (§2.4.5) that $\left(\frac{a}{n}\right)$ denotes the Jacobi symbol, and is equivalent to the Legendre symbol if $n$ is prime. The Solovay-Strassen test is based on the following fact.

**4.14 Fact** (*Euler's criterion*) Let $n$ be an odd prime. Then $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$ for all integers $a$ which satisfy $\gcd(a, n) = 1$.

Fact 4.14 motivates the following definitions.

**4.15 Definition** Let $n$ be an odd composite integer and let $a$ be an integer, $1 \le a \le n-1$.

(i) If either $\gcd(a, n) > 1$ or $a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}$, then $a$ is called an *Euler witness* (to compositeness) for $n$.

(ii) Otherwise, i.e., if $\gcd(a,n)=1$ and $a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n}$, then $n$ is said to be an *Euler pseudoprime to the base* $a$. (That is, $n$ acts like a prime in that it satisfies Euler's criterion for the particular base $a$.) The integer $a$ is called an *Euler liar* (to primality) for $n$.

**4.16 Example** (*Euler pseudoprime*) The composite integer 91 ($=7\times 13$) is an Euler pseudoprime to the base 9 since $9^{45} \equiv 1 \pmod{91}$ and $\left(\frac{9}{91}\right)=1$. □

Euler's criterion (Fact 4.14) can be used as a basis for a probabilistic primality test because of the following result.

**4.17 Fact** Let $n$ be an odd composite integer. Then at most $\phi(n)/2$ of all the numbers $a$, $1 \le a \le n-1$, are Euler liars for $n$ (Definition 4.15). Here, $\phi$ is the Euler phi function (Definition 2.100).

**4.18 Algorithm** Solovay-Strassen probabilistic primality test

SOLOVAY-STRASSEN($n$,$t$)
INPUT: an odd integer $n \ge 3$ and security parameter $t \ge 1$.
OUTPUT: an answer "prime" or "composite" to the question: "Is $n$ prime?"

1. For $i$ from 1 to $t$ do the following:
   1.1 Choose a random integer $a$, $2 \le a \le n-2$.
   1.2 Compute $r = a^{(n-1)/2} \bmod n$ using Algorithm 2.143.
   1.3 If $r \ne 1$ and $r \ne n-1$ then return("composite").
   1.4 Compute the Jacobi symbol $s = \left(\frac{a}{n}\right)$ using Algorithm 2.149.
   1.5 If $r \not\equiv s \pmod{n}$ then return ("composite").
2. Return("prime").

If $\gcd(a,n)=d$, then $d$ is a divisor of $r = a^{(n-1)/2} \bmod n$. Hence, testing whether $r \ne 1$ is step 1.3, eliminates the necessity of testing whether $\gcd(a,n) \ne 1$. If Algorithm 4.18 declares "composite", then $n$ is certainly composite because prime numbers do not violate Euler's criterion (Fact 4.14). Equivalently, if $n$ is actually prime, then the algorithm always declares "prime". On the other hand, if $n$ is actually composite, then since the bases $a$ in step 1.1 are chosen independently during each iteration of step 1, Fact 4.17 can be used to deduce the following probability of the algorithm erroneously declaring "prime".

**4.19 Fact** (*Solovay-Strassen error-probability bound*) Let $n$ be an odd composite integer. The probability that SOLOVAY-STRASSEN($n$,$t$) declares $n$ to be "prime" is less than $(\frac{1}{2})^t$.

### 4.2.3 Miller-Rabin test

The probabilistic primality test used most in practice is the Miller-Rabin test, also known as the *strong pseudoprime test*. The test is based on the following fact.

**4.20 Fact** Let $n$ be an odd prime, and let $n-1=2^s r$ where $r$ is odd. Let $a$ be any integer such that $\gcd(a,n)=1$. Then either $a^r \equiv 1 \pmod{n}$ or $a^{2^j r} \equiv -1 \pmod{n}$ for some $j$, $0 \le j \le s-1$.

Fact 4.20 motivates the following definitions.

**4.21 Definition** Let $n$ be an odd composite integer and let $n-1=2^s r$ where $r$ is odd. Let $a$ be an integer in the interval $[1,n-1]$.

(i) If $a^r \not\equiv 1 \pmod{n}$ and if $a^{2^j r} \not\equiv -1 \pmod{n}$ for all $j$, $0 \le j \le s-1$, then $a$ is called a *strong witness* (to compositeness) for $n$.

(ii) Otherwise, i.e., if either $a^r \equiv 1 \pmod{n}$ or $a^{2^j r} \equiv -1 \pmod{n}$ for some $j$, $0 \le j \le s-1$, then $n$ is said to be a *strong pseudoprime to the base* $a$. (That is, $n$ acts like a prime in that it satisfies Fact 4.20 for the particular base $a$.) The integer $a$ is called a *strong liar* (to primality) for $n$.

**4.22 Example** (*strong pseudoprime*) Consider the composite integer $n=91$ $(=7\times 13)$. Since $91-1=90=2\times 45$, $s=1$ and $r=45$. Since $9^r=9^{45}\equiv 1 \pmod{91}$, 91 is a strong pseudoprime to the base 9. The set of all strong liars for 91 is:

$$\{1,9,10,12,16,17,22,29,38,53,62,69,74,75,79,81,82,90\}.$$

Notice that the number of strong liars for 91 is $18=\phi(91)/4$, where $\phi$ is the Euler phi function (cf. Fact 4.23). □

Fact 4.20 can be used as a basis for a probabilistic primality test due to the following result.

**4.23 Fact** If $n$ is an odd composite integer, then at most $\frac{1}{4}$ of all the numbers $a$, $1\le a\le n-1$, are strong liars for $n$. In fact, if $n\ne 9$, the number of strong liars for $n$ is at most $\phi(n)/4$, where $\phi$ is the Euler phi function (Definition 2.100).

**4.24 Algorithm** Miller-Rabin probabilistic primality test

MILLER-RABIN($n$,$t$)
INPUT: an odd integer $n\ge 3$ and security parameter $t\ge 1$.
OUTPUT: an answer "prime" or "composite" to the question: "Is $n$ prime?"

1. Write $n-1=2^s r$ such that $r$ is odd.
2. For $i$ from 1 to $t$ do the following:
   - 2.1 Choose a random integer $a$, $2\le a\le n-2$.
   - 2.2 Compute $y=a^r \bmod n$ using Algorithm 2.143.
   - 2.3 If $y\ne 1$ and $y\ne n-1$ then do the following:
     - $j\leftarrow 1$.
     - While $j\le s-1$ and $y\ne n-1$ do the following:
       - Compute $y\leftarrow y^2 \bmod n$.
       - If $y=1$ then return("composite").
       - $j\leftarrow j+1$.
     - If $y\ne n-1$ then return ("composite").
3. Return("prime").

Algorithm 4.24 tests whether each base $a$ satisfies the conditions of Definition 4.21(i). In the fifth line of step 2.3, if $y=1$, then $a^{2^j r}\equiv 1 \pmod{n}$. Since it is also the case that $a^{2^{j-1}r}\not\equiv \pm 1 \pmod{n}$, it follows from Fact 3.18 that $n$ is composite (in fact $\gcd(a^{2^{j-1}r}-1,n)$ is a non-trivial factor of $n$). In the seventh line of step 2.3, if $y\ne n-1$, then $a$ is a strong witness for $n$. If Algorithm 4.24 declares "composite", then $n$ is certainly composite because prime numbers do not violate Fact 4.20. Equivalently, if $n$ is actually prime, then the algorithm always declares "prime". On the other hand, if $n$ is actually composite, then Fact 4.23 can be used to deduce the following probability of the algorithm erroneously declaring "prime".

**4.25 Fact** (*Miller-Rabin error-probability bound*) For any odd composite integer $n$, the probability that MILLER-RABIN($n$,$t$) declares $n$ to be "prime" is less than $(\frac{1}{4})^t$.

**4.26 Remark** (*number of strong liars*) For most composite integers $n$, the number of strong liars for $n$ is actually much smaller than the upper bound of $\phi(n)/4$ given in Fact 4.23. Consequently, the Miller-Rabin error-probability bound is much smaller than $(\frac{1}{4})^t$ for most positive integers $n$.

**4.27 Example** (*some composite integers have very few strong liars*) The only strong liars for the composite integer $n = 105$ $(= 3 \times 5 \times 7)$ are 1 and 104. More generally, if $k \geq 2$ and $n$ is the product of the first $k$ odd primes, there are only 2 strong liars for $n$, namely 1 and $n - 1$. □

**4.28 Remark** (*fixed bases in Miller-Rabin*) If $a_1$ and $a_2$ are strong liars for $n$, their product $a_1a_2$ is very likely, but not certain, to also be a strong liar for $n$. A strategy that is sometimes employed is to fix the bases $a$ in the Miller-Rabin algorithm to be the first few primes (composite bases are ignored because of the preceding statement), instead of choosing them at random.

**4.29 Definition** Let $p_1, p_2, \ldots, p_t$ denote the first $t$ primes. Then $\psi_t$ is defined to be the smallest positive composite integer which is a strong pseudoprime to all the bases $p_1, p_2, \ldots, p_t$.

The numbers $\psi_t$ can be interpreted as follows: to determine the primality of any integer $n < \psi_t$, it is sufficient to apply the Miller-Rabin algorithm to $n$ with the bases $a$ being the first $t$ prime numbers. With this choice of bases, the answer returned by Miller-Rabin is always correct. Table 4.1 gives the value of $\psi_t$ for $1 \leq t \leq 8$.

| $t$ | $\psi_t$ |
|---|---|
| 1 | 2047 |
| 2 | 1373653 |
| 3 | 25326001 |
| 4 | 3215031751 |
| 5 | 2152302898747 |
| 6 | 3474749660383 |
| 7 | 341550071728321 |
| 8 | 341550071728321 |

***Table 4.1:*** *Smallest strong pseudoprimes. The table lists values of $\psi_t$, the smallest positive composite integer that is a strong pseudoprime to each of the first t prime bases, for $1 \leq t \leq 8$.*

### 4.2.4 Comparison: Fermat, Solovay-Strassen, and Miller-Rabin

Fact 4.30 describes the relationships between Fermat liars, Euler liars, and strong liars (see Definitions 4.7, 4.15, and 4.21).

**4.30 Fact** Let $n$ be an odd composite integer.

(i) If $a$ is an Euler liar for $n$, then it is also a Fermat liar for $n$.
(ii) If $a$ is a strong liar for $n$, then it is also an Euler liar for $n$.

**4.31 Example** (*Fermat, Euler, strong liars*) Consider the composite integer $n = 65$ $(= 5 \times 13)$. The Fermat liars for 65 are $\{1, 8, 12, 14, 18, 21, 27, 31, 34, 38, 44, 47, 51, 53, 57, 64\}$. The Euler liars for 65 are $\{1, 8, 14, 18, 47, 51, 57, 64\}$, while the strong liars for 65 are $\{1, 8, 18, 47, 57, 64\}$. □

For a fixed composite candidate $n$, the situation is depicted in Figure 4.1. This set-

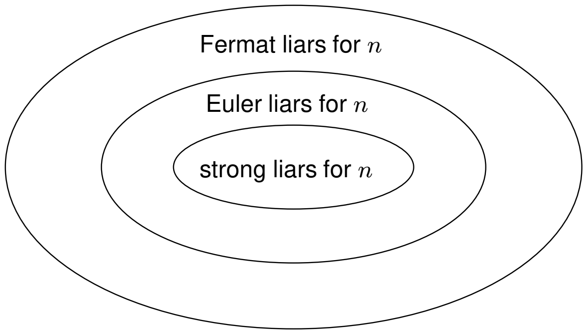


***Figure 4.1:*** *Relationships between Fermat, Euler, and strong liars for a composite integer $n$.*

tles the question of the relative accuracy of the Fermat, Solovay-Strassen, and Miller-Rabin tests, not only in the sense of the relative correctness of each test on a fixed candidate $n$, but also in the sense that given $n$, the specified containments hold for *each* randomly chosen base $a$. Thus, from a correctness point of view, the Miller-Rabin test is never worse than the Solovay-Strassen test, which in turn is never worse than the Fermat test. As the following result shows, there are, however, some composite integers $n$ for which the Solovay-Strassen and Miller-Rabin tests are equally good.

**4.32 Fact** If $n \equiv 3 \pmod 4$, then $a$ is an Euler liar for $n$ if and only if it is a strong liar for $n$.

What remains is a comparison of the computational costs. While the Miller-Rabin test may appear more complex, it actually requires, at worst, the same amount of computation as Fermat's test in terms of modular multiplications; thus the Miller-Rabin test is better than Fermat's test in all regards. At worst, the sequence of computations defined in MILLER-RABIN($n$,1) requires the equivalent of computing $a^{(n-1)/2} \bmod n$. It is also the case that MILLER-RABIN($n$,1) requires less computation than SOLOVAY-STRASSEN($n$,1), the latter requiring the computation of $a^{(n-1)/2} \bmod n$ and possibly a further Jacobi symbol computation. For this reason, the Solovay-Strassen test is both computationally and conceptually more complex.

**4.33 Note** (*Miller-Rabin is better than Solovay-Strassen*) In summary, both the Miller-Rabin and Solovay-Strassen tests are correct in the event that either their input is actually prime, or that they declare their input composite. There is, however, no reason to use the Solovay-Strassen test (nor the Fermat test) over the Miller-Rabin test. The reasons for this are summarized below.

(i) The Solovay-Strassen test is computationally more expensive.
(ii) The Solovay-Strassen test is harder to implement since it also involves Jacobi symbol computations.
(iii) The error probability for Solovay-Strassen is bounded above by $(\frac{1}{2})^t$, while the error probability for Miller-Rabin is bounded above by $(\frac{1}{4})^t$.

(iv) Any strong liar for $n$ is also an Euler liar for $n$. Hence, from a correctness point of view, the Miller-Rabin test is never worse than the Solovay-Strassen test.

## 4.3 (True) Primality tests

The primality tests in this section are methods by which positive integers can be *proven* to be prime, and are often referred to as *primality proving algorithms*. These primality tests are generally more computationally intensive than the probabilistic primality tests of §4.2. Consequently, before applying one of these tests to a candidate prime $n$, the candidate should be subjected to a probabilistic primality test such as Miller-Rabin (Algorithm 4.24).

**4.34 Definition** An integer $n$ which is determined to be prime on the basis of a primality proving algorithm is called a *provable prime*.

### 4.3.1 Testing Mersenne numbers

Efficient algorithms are known for testing primality of some special classes of numbers, such as Mersenne numbers and Fermat numbers. Mersenne primes $n$ are useful because the arithmetic in the field $\mathbb{Z}_n$ for such $n$ can be implemented very efficiently (see §14.3.4). The Lucas-Lehmer test for Mersenne numbers (Algorithm 4.37) is such an algorithm.

**4.35 Definition** Let $s \geq 2$ be an integer. A *Mersenne number* is an integer of the form $2^s - 1$. If $2^s - 1$ is prime, then it is called a *Mersenne prime*.

The following are necessary and sufficient conditions for a Mersenne number to be prime.

**4.36 Fact** Let $s \geq 3$. The Mersenne number $n = 2^s - 1$ is prime if and only if the following two conditions are satisfied:

(i) $s$ is prime; and
(ii) the sequence of integers defined by $u_0 = 4$ and $u_{k+1} = (u_k^2 - 2) \bmod n$ for $k \geq 0$ satisfies $u_{s-2} = 0$.

Fact 4.36 leads to the following deterministic polynomial-time algorithm for determining (with certainty) whether a Mersenne number is prime.

**4.37 Algorithm** Lucas-Lehmer primality test for Mersenne numbers

INPUT: a Mersenne number $n = 2^s - 1$ with $s \geq 3$.
OUTPUT: an answer "prime" or "composite" to the question: "Is $n$ prime?"

1. Use trial division to check if $s$ has any factors between 2 and $\lfloor\sqrt{s}\rfloor$. If it does, then return("composite").
2. Set $u \leftarrow 4$.
3. For $k$ from 1 to $s-2$ do the following: compute $u \leftarrow (u^2 - 2) \bmod n$.
4. If $u = 0$ then return("prime"). Otherwise, return("composite").

It is unknown whether there are infinitely many Mersenne primes. Table 4.2 lists the 33 known Mersenne primes.

| Index $j$ | $M_j$ | decimal digits |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 1 |
| 3 | 5 | 2 |
| 4 | 7 | 3 |
| 5 | 13 | 4 |
| 6 | 17 | 6 |
| 7 | 19 | 6 |
| 8 | 31 | 10 |
| 9 | 61 | 19 |
| 10 | 89 | 27 |
| 11 | 107 | 33 |
| 12 | 127 | 39 |
| 13 | 521 | 157 |
| 14 | 607 | 183 |
| 15 | 1279 | 386 |
| 16 | 2203 | 664 |
| 17 | 2281 | 687 |

| Index $j$ | $M_j$ | decimal digits |
|---|---|---|
| 18 | 3217 | 969 |
| 19 | 4253 | 1281 |
| 20 | 4423 | 1332 |
| 21 | 9689 | 2917 |
| 22 | 9941 | 2993 |
| 23 | 11213 | 3376 |
| 24 | 19937 | 6002 |
| 25 | 21701 | 6533 |
| 26 | 23209 | 6987 |
| 27 | 44497 | 13395 |
| 28 | 86243 | 25962 |
| 29 | 110503 | 33265 |
| 30 | 132049 | 39751 |
| 31 | 216091 | 65050 |
| 32? | 756839 | 227832 |
| 33? | 859433 | 258716 |

***Table 4.2:*** *Known Mersenne primes. The table shows the* $33$ *known exponents* $M_j$*,* $1 \leq j \leq 33$*, for which* $2^{M_j}-1$ *is a Mersenne prime, and also the number of decimal digits in* $2^{M_j}-1$*. The question marks after* $j=32$ *and* $j=33$ *indicate that it is not known whether there are any other exponents* $s$ *between* $M_{31}$ *and these numbers for which* $2^s-1$ *is prime.*

## 4.3.2 Primality testing using the factorization of $n-1$

This section presents results which can be used to prove that an integer $n$ is prime, provided that the factorization or a partial factorization of $n-1$ is known. It may seem odd to consider a technique which requires the factorization of $n-1$ as a subproblem — if integers of this size can be factored, the primality of $n$ itself could be determined by factoring $n$. However, the factorization of $n-1$ may be easier to compute if $n$ has a special form, such as a *Fermat number* $n = 2^{2^k}+1$. Another situation where the factorization of $n-1$ may be easy to compute is when the candidate $n$ is "constructed" by specific methods (see §4.4.4).

**4.38 Fact** Let $n \geq 3$ be an integer. Then $n$ is prime if and only if there exists an integer $a$ satisfying:

(i) $a^{n-1} \equiv 1 \pmod{n}$; and
(ii) $a^{(n-1)/q} \not\equiv 1 \pmod{n}$ for each prime divisor $q$ of $n-1$.

This result follows from the fact that $\mathbb{Z}_n^*$ has an element of order $n-1$ (Definition 2.128) if and only if $n$ is prime; an element $a$ satisfying conditions (i) and (ii) has order $n-1$.

**4.39 Note** (*primality test based on Fact 4.38*) If $n$ is a prime, the number of elements of order $n-1$ is precisely $\phi(n-1)$. Hence, to prove a candidate $n$ prime, one may simply choose an integer $a \in \mathbb{Z}_n$ at random and uses Fact 4.38 to check if $a$ has order $n-1$. If this is the case, then $n$ is certainly prime. Otherwise, another $a \in \mathbb{Z}_n$ is selected and the test is repeated. If $n$ is indeed prime, the expected number of iterations before an element $a$ of order $n-1$ is selected is $O(\ln\ln n)$; this follows since $(n-1)/\phi(n-1) < 6\ln\ln n$ for

$n \geq 5$ (Fact 2.102). Thus, if such an $a$ is not found after a "reasonable" number (for example, $12 \ln \ln n$) of iterations, then $n$ is probably composite and should again be subjected to a probabilistic primality test such as Miller-Rabin (Algorithm 4.24).[3] This method is, in effect, a probabilistic compositeness test.

The next result gives a method for proving primality which requires knowledge of only a *partial* factorization of $n-1$.

**4.40 Fact** (*Pocklington's theorem*) Let $n \geq 3$ be an integer, and let $n = RF + 1$ (i.e. $F$ divides $n-1$) where the prime factorization of $F$ is $F = \prod_{j=1}^{t} q_j^{e_j}$. If there exists an integer $a$ satisfying:

(i) $a^{n-1} \equiv 1 \pmod{n}$; and
(ii) $\gcd(a^{(n-1)/q_j} - 1, n) = 1$ for each $j$, $1 \leq j \leq t$,

then every prime divisor $p$ of $n$ is congruent to 1 modulo $F$. It follows that if $F > \sqrt{n} - 1$, then $n$ is prime.

If $n$ is indeed prime, then the following result establishes that most integers $a$ satisfy conditions (i) and (ii) of Fact 4.40, provided that the prime divisors of $F > \sqrt{n} - 1$ are sufficiently large.

**4.41 Fact** Let $n = RF + 1$ be an odd prime with $F > \sqrt{n} - 1$ and $\gcd(R, F) = 1$. Let the distinct prime factors of $F$ be $q_1, q_2, \ldots, q_t$. Then the probability that a randomly selected base $a$, $1 \leq a \leq n-1$, satisfies both: (i) $a^{n-1} \equiv 1 \pmod{n}$; and (ii) $\gcd(a^{(n-1)/q_j} - 1, n) = 1$ for each $j$, $1 \leq j \leq t$, is $\prod_{j=1}^{t}(1 - 1/q_j) \geq 1 - \sum_{j=1}^{t} 1/q_j$.

Thus, if the factorization of a divisor $F > \sqrt{n} - 1$ of $n-1$ is known then to test $n$ for primality, one may simply choose random integers $a$ in the interval $[2, n-2]$ until one is found satisfying conditions (i) and (ii) of Fact 4.40, implying that $n$ is prime. If such an $a$ is not found after a "reasonable" number of iterations,[4] then $n$ is probably composite and this could be established by subjecting it to a probabilistic primality test (footnote 3 also applies here). This method is, in effect, a probabilistic compositeness test.

The next result gives a method for proving primality which only requires the factorization of a divisor $F$ of $n-1$ that is greater than $\sqrt[3]{n}$. For an example of the use of Fact 4.42, see Note 4.63.

**4.42 Fact** Let $n \geq 3$ be an odd integer. Let $n = 2RF + 1$, and suppose that there exists an integer $a$ satisfying both: (i) $a^{n-1} \equiv 1 \pmod{n}$; and (ii) $\gcd(a^{(n-1)/q} - 1, n) = 1$ for each prime divisor $q$ of $F$. Let $x \geq 0$ and $y$ be defined by $2R = xF + y$ and $0 \leq y < F$. If $F \geq \sqrt[3]{n}$ and if $y^2 - 4x$ is neither 0 nor a perfect square, then $n$ is prime.

### 4.3.3 Jacobi sum test

The *Jacobi sum test* is another true primality test. The basic idea is to test a set of congruences which are analogues of Fermat's theorem (Fact 2.127(i)) in certain *cyclotomic rings*. The running time of the Jacobi sum test for determining the primality of an integer $n$ is $O((\ln n)^{c \ln \ln \ln n})$ bit operations for some constant $c$. This is "almost" a polynomial-time algorithm since the exponent $\ln \ln \ln n$ acts like a constant for the range of values for

[3] Another approach is to run both algorithms in parallel (with an unlimited number of iterations), until one of them stops with a definite conclusion "prime" or "composite".

[4] The number of iterations may be taken to be $T$ where $P^T \leq (\frac{1}{2})^{100}$, and where $P = 1 - \prod_{j=1}^{t}(1 - 1/q_j)$.

$n$ of interest. For example, if $n \leq 2^{512}$, then $\ln\ln\ln n < 1.78$. The version of the Jacobi sum primality test used in practice is a randomized algorithm which terminates within $O(k(\ln n)^{c\ln\ln\ln n})$ steps with probability at least $1 - (\frac{1}{2})^k$ for every $k \geq 1$, and always gives a correct answer. One drawback of the algorithm is that it does not produce a "certificate" which would enable the answer to be verified in much shorter time than running the algorithm itself.

The Jacobi sum test is, indeed, practical in the sense that the primality of numbers that are several hundred decimal digits long can be handled in just a few minutes on a computer. However, the test is not as easy to program as the probabilistic Miller-Rabin test (Algorithm 4.24), and the resulting code is not as compact. The details of the algorithm are complicated and are not given here; pointers to the literature are given in the chapter notes on page 166.

### 4.3.4 Tests using elliptic curves

Elliptic curve primality proving algorithms are based on an elliptic curve analogue of Pocklington's theorem (Fact 4.40). The version of the algorithm used in practice is usually referred to as *Atkin's test* or the *Elliptic Curve Primality Proving algorithm* (ECPP). Under heuristic arguments, the expected running time of this algorithm for proving the primality of an integer $n$ has been shown to be $O((\ln n)^{6+\epsilon})$ bit operations for any $\epsilon > 0$. Atkin's test has the advantage over the Jacobi sum test (§4.3.3) that it produces a short *certificate of primality* which can be used to efficiently verify the primality of the number. Atkin's test has been used to prove the primality of numbers more than 1000 decimal digits long.

The details of the algorithm are complicated and are not presented here; pointers to the literature are given in the chapter notes on page 166.

## 4.4 Prime number generation

This section considers algorithms for the generation of prime numbers for cryptographic purposes. Four algorithms are presented: Algorithm 4.44 for generating *probable* primes (see Definition 4.5), Algorithm 4.53 for generating *strong* primes (see Definition 4.52), Algorithm 4.56 for generating *probable* primes $p$ and $q$ suitable for use in the Digital Signature Algorithm (DSA), and Algorithm 4.62 for generating *provable* primes (see Definition 4.34).

**4.43 Note** (*prime generation vs. primality testing*) Prime number *generation* differs from primality *testing* as described in §4.2 and §4.3, but may and typically does involve the latter. The former allows the construction of candidates of a fixed form which may lead to more efficient testing than possible for random candidates.

### 4.4.1 Random search for probable primes

By the prime number theorem (Fact 2.95), the proportion of (positive) integers $\leq x$ that are prime is approximately $1/\ln x$. Since half of all integers $\leq x$ are even, the proportion of *odd* integers $\leq x$ that are prime is approximately $2/\ln x$. For instance, the proportion of all odd integers $\leq 2^{512}$ that are prime is approximately $2/(512 \cdot \ln(2)) \approx 1/177$. This suggests that a reasonable strategy for selecting a random $k$-bit (probable) prime is to repeatedly pick random $k$-bit odd integers $n$ until one is found that is declared to be "prime" by MILLER-RABIN($n$,$t$) (Algorithm 4.24) for an appropriate value of the security parameter $t$ (discussed below).

If a random $k$-bit odd integer $n$ is divisible by a small prime, it is less computationally expensive to rule out the candidate $n$ by trial division than by using the Miller-Rabin test. Since the probability that a random integer $n$ has a small prime divisor is relatively large, before applying the Miller-Rabin test, the candidate $n$ should be tested for small divisors below a pre-determined bound $B$. This can be done by dividing $n$ by all the primes below $B$, or by computing greatest common divisors of $n$ and (pre-computed) products of several of the primes $\leq B$. The proportion of candidate odd integers $n$ not ruled out by this trial division is $\prod_{3\leq p\leq B}(1-\frac{1}{p})$ which, by Mertens's theorem, is approximately $1.12/\ln B$ (here $p$ ranges over prime values). For example, if $B = 256$, then only 20% of candidate odd integers $n$ pass the trial division stage, i.e., 80% are discarded before the more costly Miller-Rabin test is performed.

**4.44 Algorithm** Random search for a prime using the Miller-Rabin test

RANDOM-SEARCH($k$,$t$)
INPUT: an integer $k$, and a security parameter $t$ (cf. Note 4.49).
OUTPUT: a random $k$-bit probable prime.

1. Generate an odd $k$-bit integer $n$ at random.
2. Use trial division to determine whether $n$ is divisible by any odd prime $\leq B$ (see Note 4.45 for guidance on selecting $B$). If it is then go to step 1.
3. If MILLER-RABIN($n$,$t$) (Algorithm 4.24) outputs "prime" then return($n$). Otherwise, go to step 1.

**4.45 Note** (*optimal trial division bound $B$*) Let $E$ denote the time for a full $k$-bit modular exponentiation, and let $D$ denote the time required for ruling out one small prime as divisor of a $k$-bit integer. (The values $E$ and $D$ depend on the particular implementation of long-integer arithmetic.) Then the trial division bound $B$ that minimizes the expected running time of Algorithm 4.44 for generating a $k$-bit prime is roughly $B = E/D$. A more accurate estimate of the optimum choice for $B$ can be obtained experimentally. The odd primes up to $B$ can be precomputed and stored in a table. If memory is scarce, a value of $B$ that is smaller than the optimum value may be used.

Since the Miller-Rabin test does not provide a mathematical proof that a number is indeed prime, the number $n$ returned by Algorithm 4.44 is a probable prime (Definition 4.5). It is important, therefore, to have an estimate of the probability that $n$ is in fact composite.

**4.46 Definition** The probability that RANDOM-SEARCH($k$,$t$) (Algorithm 4.44) returns a composite number is denoted by $p_{k,t}$.

**4.47 Note** (*remarks on estimating $p_{k,t}$*) It is tempting to conclude directly from Fact 4.25 that $p_{k,t} \leq (\frac{1}{4})^t$. This reasoning is flawed (although typically the conclusion will be correct in practice) since it does not take into account the distribution of the primes. (For example, if all candidates $n$ were chosen from a set $S$ of composite numbers, the probability of error is 1.) The following discussion elaborates on this point. Let $X$ represent the event that $n$ is composite, and let $Y_t$ denote the event than MILLER-RABIN($n$,$t$) declares $n$ to be prime. Then Fact 4.25 states that $P(Y_t|X) \leq (\frac{1}{4})^t$. What is relevant, however, to the estimation of $p_{k,t}$ is the quantity $P(X|Y_t)$. Suppose that candidates $n$ are drawn uniformly and randomly from a set $S$ of odd numbers, and suppose $p$ is the probability that $n$ is prime (this depends on the candidate set $S$). Assume also that $0 < p < 1$. Then by Bayes' theorem (Fact 2.10):

$$P(X|Y_t) \;=\; \frac{P(X)P(Y_t|X)}{P(Y_t)} \;\leq\; \frac{P(Y_t|X)}{P(Y_t)} \;\leq\; \frac{1}{p}\left(\frac{1}{4}\right)^t,$$

since $P(Y_t) \geq p$. Thus the probability $P(X|Y_t)$ may be considerably larger than $(\frac{1}{4})^t$ if $p$ is small. However, the error-probability of Miller-Rabin is usually far smaller than $(\frac{1}{4})^t$ (see Remark 4.26). Using better estimates for $P(Y_t|X)$ and estimates on the number of $k$-bit prime numbers, it has been shown that $p_{k,t}$ is, in fact, smaller than $(\frac{1}{4})^t$ for all sufficiently large $k$. A more concrete result is the following: if candidates $n$ are chosen at random from the set of odd numbers in the interval $[3, x]$, then $P(X|Y_t) \leq (\frac{1}{4})^t$ for all $x \geq 10^{60}$.

Further refinements for $P(Y_t|X)$ allow the following explicit upper bounds on $p_{k,t}$ for various values of $k$ and $t$. [5]

**4.48 Fact** (*some upper bounds on $p_{k,t}$ in Algorithm 4.44*)

(i) $p_{k,1} < k^2 4^{2-\sqrt{k}}$ for $k \geq 2$.
(ii) $p_{k,t} < k^{3/2} 2^t t^{-1/2} 4^{2-\sqrt{tk}}$ for ($t = 2$, $k \geq 88$) or ($3 \leq t \leq k/9$, $k \geq 21$).
(iii) $p_{k,t} < \frac{7}{20} k 2^{-5t} + \frac{1}{7} k^{15/4} 2^{-k/2-2t} + 12k 2^{-k/4-3t}$ for $k/9 \leq t \leq k/4$, $k \geq 21$.
(iv) $p_{k,t} < \frac{1}{7} k^{15/4} 2^{-k/2-2t}$ for $t \geq k/4$, $k \geq 21$.

For example, if $k = 512$ and $t = 6$, then Fact 4.48(ii) gives $p_{512,6} \leq (\frac{1}{2})^{88}$. In other words, the probability that RANDOM-SEARCH(512,6) returns a 512-bit composite integer is less than $(\frac{1}{2})^{88}$. Using more advanced techniques, the upper bounds on $p_{k,t}$ given by Fact 4.48 have been improved. These upper bounds arise from complicated formulae which are not given here. Table 4.3 lists some improved upper bounds on $p_{k,t}$ for some sample values of $k$ and $t$. As an example, the probability that RANDOM-SEARCH(500,6) returns a composite number is $\leq (\frac{1}{2})^{92}$. Notice that the values of $p_{k,t}$ implied by the table are considerably smaller than $(\frac{1}{4})^t = (\frac{1}{2})^{2t}$.

| | $t$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 100 | 5 | 14 | 20 | 25 | 29 | 33 | 36 | 39 | 41 | 44 |
| 150 | 8 | 20 | 28 | 34 | 39 | 43 | 47 | 51 | 54 | 57 |
| 200 | 11 | 25 | 34 | 41 | 47 | 52 | 57 | 61 | 65 | 69 |
| 250 | 14 | 29 | 39 | 47 | 54 | 60 | 65 | 70 | 75 | 79 |
| 300 | 19 | 33 | 44 | 53 | 60 | 67 | 73 | 78 | 83 | 88 |
| 350 | 28 | 38 | 48 | 58 | 66 | 73 | 80 | 86 | 91 | 97 |
| 400 | 37 | 46 | 55 | 63 | 72 | 80 | 87 | 93 | 99 | 105 |
| 450 | 46 | 54 | 62 | 70 | 78 | 85 | 93 | 100 | 106 | 112 |
| 500 | 56 | 63 | 70 | 78 | 85 | 92 | 99 | 106 | 113 | 119 |
| 550 | 65 | 72 | 79 | 86 | 93 | 100 | 107 | 113 | 119 | 126 |
| 600 | 75 | 82 | 88 | 95 | 102 | 108 | 115 | 121 | 127 | 133 |

***Table 4.3:*** *Upper bounds on $p_{k,t}$ for sample values of $k$ and $t$. An entry $j$ corresponding to $k$ and $t$ implies $p_{k,t} \leq (\frac{1}{2})^j$.*

[5] The estimates of $p_{k,t}$ presented in the remainder of this subsection were derived for the situation where Algorithm 4.44 does not use trial division by small primes to rule out some candidates $n$. Since trial division never rules out a prime, it can only give a better chance of rejecting composites. Thus the error probability $p_{k,t}$ might actually be even smaller than the estimates given here.

**4.49 Note** (*controlling the error probability*) In practice, one is usually willing to tolerate an error probability of $(\frac{1}{2})^{80}$ when using Algorithm 4.44 to generate probable primes. For sample values of $k$, Table 4.4 lists the smallest value of $t$ that can be derived from Fact 4.48 for which $p_{k,t} \leq (\frac{1}{2})^{80}$. For example, when generating 1000-bit probable primes, Miller-Rabin with $t = 3$ repetitions suffices. Algorithm 4.44 rules out most candidates $n$ either by trial division (in step 2) or by performing just one iteration of the Miller-Rabin test (in step 3). For this reason, the only effect of selecting a larger security parameter $t$ on the running time of the algorithm will likely be to increase the time required in the final stage when the (probable) prime is chosen.

| $k$ | $t$ | $k$ | $t$ | $k$ | $t$ | $k$ | $t$ | $k$ | $t$ |
|---|---|---|---|---|---|---|---|---|---|
| 100 | 27 | 500 | 6 | 900 | 3 | 1300 | 2 | 1700 | 2 |
| 150 | 18 | 550 | 5 | 950 | 3 | 1350 | 2 | 1750 | 2 |
| 200 | 15 | 600 | 5 | 1000 | 3 | 1400 | 2 | 1800 | 2 |
| 250 | 12 | 650 | 4 | 1050 | 3 | 1450 | 2 | 1850 | 2 |
| 300 | 9 | 700 | 4 | 1100 | 3 | 1500 | 2 | 1900 | 2 |
| 350 | 8 | 750 | 4 | 1150 | 3 | 1550 | 2 | 1950 | 2 |
| 400 | 7 | 800 | 4 | 1200 | 3 | 1600 | 2 | 2000 | 2 |
| 450 | 6 | 850 | 3 | 1250 | 3 | 1650 | 2 | 2050 | 2 |

***Table 4.4:*** *For sample k, the smallest t from Fact 4.48 is given for which* $p_{k,t} \leq (\frac{1}{2})^{80}$.

**4.50 Remark** (*Miller-Rabin test with base* $a = 2$) The Miller-Rabin test involves exponentiating the base $a$; this may be performed using the repeated square-and-multiply algorithm (Algorithm 2.143). If $a = 2$, then multiplication by $a$ is a simple procedure relative to multiplying by $a$ in general. One optimization of Algorithm 4.44 is, therefore, to fix the base $a = 2$ when first performing the Miller-Rabin test in step 3. Since most composite numbers will fail the Miller-Rabin test with base $a = 2$, this modification will lower the expected running time of Algorithm 4.44.

**4.51 Note** (*incremental search*)

(i) An alternative technique to generating candidates $n$ at random in step 1 of Algorithm 4.44 is to first select a random $k$-bit odd number $n_0$, and then test the $s$ numbers $n = n_0, n_0 + 2, n_0 + 4, \ldots, n_0 + 2(s-1)$ for primality. If all these $s$ candidates are found to be composite, the algorithm is said to have *failed*. If $s = c \cdot \ln 2^k$ where $c$ is a constant, the probability $q_{k,t,s}$ that this incremental search variant of Algorithm 4.44 returns a composite number has been shown to be less than $\delta k^3 2^{-\sqrt{k}}$ for some constant $\delta$. Table 4.5 gives some explicit bounds on this error probability for $k = 500$ and $t \leq 10$. Under reasonable number-theoretic assumptions, the probability of the algorithm failing has been shown to be less than $2e^{-2c}$ for large $k$ (here, $e \approx 2.71828$).

(ii) Incremental search has the advantage that fewer random bits are required. Furthermore, the trial division by small primes in step 2 of Algorithm 4.44 can be accomplished very efficiently as follows. First the values $R[p] = n_0 \bmod p$ are computed for each odd prime $p \leq B$. Each time 2 is added to the current candidate, the values in the table $R$ are updated as $R[p] \leftarrow (R[p] + 2) \bmod p$. The candidate passes the trial division stage if and only if none of the $R[p]$ values equal 0.

(iii) If $B$ is large, an alternative method for doing the trial division is to initialize a table $S[i] \leftarrow 0$ for $0 \leq i \leq (s-1)$; the entry $S[i]$ corresponds to the candidate $n_0 + 2i$. For each odd prime $p \leq B$, $n_0 \bmod p$ is computed. Let $j$ be the smallest index for

| $c$ | $t$ | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 1 | 17 | 37 | 51 | 63 | 72 | 81 | 89 | 96 | 103 | 110 |
| 5 | 13 | 32 | 46 | 58 | 68 | 77 | 85 | 92 | 99 | 105 |
| 10 | 11 | 30 | 44 | 56 | 66 | 75 | 83 | 90 | 97 | 103 |

**Table 4.5:** *Upper bounds on the error probability of incremental search (Note 4.51) for $k = 500$ and sample values of $c$ and $t$. An entry $j$ corresponding to $c$ and $t$ implies $q_{500,t,s} \le (\frac{1}{2})^j$, where $s = c \cdot \ln 2^{500}$.*

which $(n_0 + 2j) \equiv 0 \pmod p$. Then $S[j]$ and each $p^{\text{th}}$ entry after it are set to 1. A candidate $n_0 + 2i$ then passes the trial division stage if and only if $S[i] = 0$. Note that the estimate for the optimal trial division bound $B$ given in Note 4.45 does not apply here (nor in (ii)) since the cost of division is amortized over all candidates.

### 4.4.2 Strong primes

The RSA cryptosystem (§8.2) uses a modulus of the form $n = pq$, where $p$ and $q$ are distinct odd primes. The primes $p$ and $q$ must be of sufficient size that factorization of their product is beyond computational reach. Moreover, they should be random primes in the sense that they be chosen as a function of a random input through a process defining a pool of candidates of sufficient cardinality that an exhaustive attack is infeasible. In practice, the resulting primes must also be of a pre-determined bitlength, to meet system specifications. The discovery of the RSA cryptosystem led to the consideration of several additional constraints on the choice of $p$ and $q$ which are necessary to ensure the resulting RSA system safe from cryptanalytic attack, and the notion of a strong prime (Definition 4.52) was defined. These attacks are described at length in Note 8.8(iii); as noted there, it is now believed that strong primes offer little protection beyond that offered by random primes, since randomly selected primes of the sizes typically used in RSA moduli today will satisfy the constraints with high probability. On the other hand, they are no less secure, and require only minimal additional running time to compute; thus, there is little real additional cost in using them.

**4.52 Definition** A prime number $p$ is said to be a *strong prime* if integers $r$, $s$, and $t$ exist such that the following three conditions are satisfied:

(i) $p - 1$ has a large prime factor, denoted $r$;
(ii) $p + 1$ has a large prime factor, denoted $s$; and
(iii) $r - 1$ has a large prime factor, denoted $t$.

In Definition 4.52, a precise qualification of "large" depends on specific attacks that should be guarded against; for further details, see Note 8.8(iii).

**4.53 Algorithm** Gordon's algorithm for generating a strong prime

SUMMARY: a strong prime $p$ is generated.

1. Generate two large random primes $s$ and $t$ of roughly equal bitlength (see Note 4.54).
2. Select an integer $i_0$. Find the first prime in the sequence $2it + 1$, for $i = i_0, i_0 + 1, i_0 + 2, \ldots$ (see Note 4.54). Denote this prime by $r = 2it + 1$.
3. Compute $p_0 = 2(s^{r-2} \bmod r)s - 1$.
4. Select an integer $j_0$. Find the first prime in the sequence $p_0 + 2jrs$, for $j = j_0, j_0 + 1, j_0 + 2, \ldots$ (see Note 4.54). Denote this prime by $p = p_0 + 2jrs$.
5. Return($p$).

*Justification.* To see that the prime $p$ returned by Gordon's algorithm is indeed a strong prime, observe first (assuming $r \neq s$) that $s^{r-1} \equiv 1 \pmod r$; this follows from Fermat's theorem (Fact 2.127). Hence, $p_0 \equiv 1 \pmod r$ and $p_0 \equiv -1 \pmod s$. Finally (cf. Definition 4.52),

(i) $p - 1 = p_0 + 2jrs - 1 \equiv 0 \pmod r$, and hence $p - 1$ has the prime factor $r$;
(ii) $p + 1 = p_0 + 2jrs + 1 \equiv 0 \pmod s$, and hence $p + 1$ has the prime factor $s$; and
(iii) $r - 1 = 2it \equiv 0 \pmod t$, and hence $r - 1$ has the prime factor $t$.

**4.54 Note** (*implementing Gordon's algorithm*)

(i) The primes $s$ and $t$ required in step 1 can be probable primes generated by Algorithm 4.44. The Miller-Rabin test (Algorithm 4.24) can be used to test each candidate for primality in steps 2 and 4, after ruling out candidates that are divisible by a small prime less than some bound $B$. See Note 4.45 for guidance on selecting $B$. Since the Miller-Rabin test is a probabilistic primality test, the output of this implementation of Gordon's algorithm is a probable prime.
(ii) By carefully choosing the sizes of primes $s$, $t$ and parameters $i_0$, $j_0$, one can control the exact bitlength of the resulting prime $p$. Note that the bitlengths of $r$ and $s$ will be about half that of $p$, while the bitlength of $t$ will be slightly less than that of $r$.

**4.55 Fact** (*running time of Gordon's algorithm*) If the Miller-Rabin test is the primality test used in steps 1, 2, and 4, the expected time Gordon's algorithm takes to find a strong prime is only about 19% more than the expected time Algorithm 4.44 takes to find a random prime.

### 4.4.3 NIST method for generating DSA primes

Some public-key schemes require primes satisfying various specific conditions. For example, the NIST Digital Signature Algorithm (DSA of §11.5.1) requires two primes $p$ and $q$ satisfying the following three conditions:

(i) $2^{159} < q < 2^{160}$; that is, $q$ is a 160-bit prime;
(ii) $2^{L-1} < p < 2^L$ for a specified $L$, where $L = 512 + 64l$ for some $0 \leq l \leq 8$; and
(iii) $q$ divides $p - 1$.

This section presents an algorithm for generating such primes $p$ and $q$. In the following, $H$ denotes the SHA-1 hash function (Algorithm 9.53) which maps bitstrings of bitlength $< 2^{64}$ to 160-bit hash-codes. Where required, an integer $x$ in the range $0 \leq x < 2^g$ whose binary representation is $x = x_{g-1}2^{g-1} + x_{g-2}2^{g-2} + \cdots + x_2 2^2 + x_1 2 + x_0$ should be converted to the $g$-bit sequence $(x_{g-1}x_{g-2} \cdots x_2 x_1 x_0)$, and vice versa.

**4.56 Algorithm** NIST method for generating DSA primes

INPUT: an integer $l$, $0 \leq l \leq 8$.
OUTPUT: a 160-bit prime $q$ and an $L$-bit prime $p$, where $L = 512 + 64l$ and $q|(p-1)$.

1. Compute $L = 512 + 64l$. Using long division of $(L-1)$ by 160, find $n$, $b$ such that $L - 1 = 160n + b$, where $0 \leq b < 160$.
2. Repeat the following:
   2.1 Choose a random seed $s$ (not necessarily secret) of bitlength $g \geq 160$.
   2.2 Compute $U = H(s) \oplus H((s+1) \bmod 2^g)$.
   2.3 Form $q$ from $U$ by setting to 1 the most significant and least significant bits of $U$. (Note that $q$ is a 160-bit odd integer.)
   2.4 Test $q$ for primality using MILLER-RABIN($q$,$t$) for $t \geq 18$ (see Note 4.57).
   
   Until $q$ is found to be a (probable) prime.
3. Set $i \leftarrow 0$, $j \leftarrow 2$.
4. While $i < 4096$ do the following:
   4.1 For $k$ from 0 to $n$ do the following: set $V_k \leftarrow H((s + j + k) \bmod 2^g)$.
   4.2 For the integer $W$ defined below, let $X = W + 2^{L-1}$. ($X$ is an $L$-bit integer.)
   $$W = V_0 + V_1 2^{160} + V_2 2^{320} + \cdots + V_{n-1} 2^{160(n-1)} + (V_n \bmod 2^b) 2^{160n}.$$
   4.3 Compute $c = X \bmod 2q$ and set $p = X - (c-1)$. (Note that $p \equiv 1 \pmod{2q}$.)
   4.4 If $p \geq 2^{L-1}$ then do the following:
   Test $p$ for primality using MILLER-RABIN($p$,$t$) for $t \geq 5$ (see Note 4.57).
   If $p$ is a (probable) prime then return($q$,$p$).
   4.5 Set $i \leftarrow i + 1$, $j \leftarrow j + n + 1$.
5. Go to step 2.

**4.57 Note** (*choice of primality test in Algorithm 4.56*)

(i) The FIPS 186 document where Algorithm 4.56 was originally described only specifies that a *robust* primality test be used in steps 2.4 and 4.4, i.e., a primality test where the probability of a composite integer being declared prime is at most $(\frac{1}{2})^{80}$. If the heuristic assumption is made that $q$ is a randomly chosen 160-bit integer then, by Table 4.4, MILLER-RABIN($q$,18) is a robust test for the primality of $q$. If $p$ is assumed to be a randomly chosen $L$-bit integer, then by Table 4.4, MILLER-RABIN($p$,5) is a robust test for the primality of $p$. Since the Miller-Rabin test is a probabilistic primality test, the output of Algorithm 4.56 is a probable prime.

(ii) To improve performance, candidate primes $q$ and $p$ should be subjected to trial division by all odd primes less than some bound $B$ before invoking the Miller-Rabin test. See Note 4.45 for guidance on selecting $B$.

**4.58 Note** (*"weak" primes cannot be intentionally constructed*) Algorithm 4.56 has the feature that the random seed $s$ is not input to the prime number generation portion of the algorithm itself, but rather to an unpredictable and uncontrollable randomization process (steps 2.2 and 4.1), the output of which is used as the actual random seed. This precludes manipulation of the input seed to the prime number generation. If the seed $s$ and counter $i$ are made public, then anyone can verify that $q$ and $p$ were generated using the approved method. This feature prevents a central authority who generates $p$ and $q$ as system-wide parameters for use in the DSA from intentionally constructing "weak" primes $q$ and $p$ which it could subsequently exploit to recover other entities' private keys.

### 4.4.4 Constructive techniques for provable primes

Maurer's algorithm (Algorithm 4.62) generates random *provable* primes that are almost uniformly distributed over the set of all primes of a specified size. The expected time for generating a prime is only slightly greater than that for generating a probable prime of equal size using Algorithm 4.44 with security parameter $t = 1$. (In practice, one may wish to choose $t > 1$ in Algorithm 4.44; cf. Note 4.49.)

The main idea behind Algorithm 4.62 is Fact 4.59, which is a slight modification of Pocklington's theorem (Fact 4.40) and Fact 4.41.

**4.59 Fact** Let $n \geq 3$ be an odd integer, and suppose that $n = 1 + 2Rq$ where $q$ is an odd prime. Suppose further that $q > R$.

(i) If there exists an integer $a$ satisfying $a^{n-1} \equiv 1 \pmod{n}$ and $\gcd(a^{2R} - 1, n) = 1$, then $n$ is prime.

(ii) If $n$ is prime, the probability that a randomly selected base $a$, $1 \leq a \leq n-1$, satisfies $a^{n-1} \equiv 1 \pmod{n}$ and $\gcd(a^{2R} - 1, n) = 1$ is $(1 - 1/q)$.

Algorithm 4.62 recursively generates an odd prime $q$, and then chooses random integers $R$, $R < q$, until $n = 2Rq + 1$ can be proven prime using Fact 4.59(i) for some base $a$. By Fact 4.59(ii) the proportion of such bases is $1 - 1/q$ for prime $n$. On the other hand, if $n$ is composite, then most bases $a$ will fail to satisfy the condition $a^{n-1} \equiv 1 \pmod{n}$.

**4.60 Note** (*description of constants* $c$ *and* $m$ *in Algorithm 4.62*)

(i) The optimal value of the constant $c$ defining the trial division bound $B = ck^2$ in step 2 depends on the implementation of long-integer arithmetic, and is best determined experimentally (cf. Note 4.45).

(ii) The constant $m = 20$ ensures that $I$ is at least 20 bits long and hence the interval from which $R$ is selected, namely $[I + 1, 2I]$, is sufficiently large (for the values of $k$ of practical interest) that it most likely contains at least one value $R$ for which $n = 2Rq + 1$ is prime.

**4.61 Note** (*relative size* $r$ *of* $q$ *with respect to* $n$ *in Algorithm 4.62*) The *relative size* $r$ of $q$ with respect to $n$ is defined to be $r = \lg q / \lg n$. In order to assure that the generated prime $n$ is chosen randomly with essentially uniform distribution from the set of all $k$-bit primes, the size of the prime factor $q$ of $n - 1$ must be chosen according to the probability distribution of the largest prime factor of a randomly selected $k$-bit integer. Since $q$ must be greater than $R$ in order for Fact 4.59 to apply, the relative size $r$ of $q$ is restricted to being in the interval $[\frac{1}{2}, 1]$. It can be deduced from Fact 3.7(i) that the cumulative probability distribution of the relative size $r$ of the largest prime factor of a large random integer, given that $r$ is at least $\frac{1}{2}$, is $(1 + \lg r)$ for $\frac{1}{2} \leq r \leq 1$. In step 4 of Algorithm 4.62, the relative size $r$ is generated according to this distribution by selecting a random number $s \in [0, 1]$ and then setting $r = 2^{s-1}$. If $k \leq 2m$ then $r$ is chosen to be the smallest permissible value, namely $\frac{1}{2}$, in order to ensure that the interval from which $R$ is selected is sufficiently large (cf. Note 4.60(ii)).

**4.62 Algorithm** Maurer's algorithm for generating provable primes

PROVABLE_PRIME($k$)
INPUT: a positive integer $k$.
OUTPUT: a $k$-bit prime number $n$.

1. (*If k is small, then test random integers by trial division. A table of small primes may be precomputed for this purpose.*)
   If $k \le 20$ then repeatedly do the following:
   1.1 Select a random $k$-bit odd integer $n$.
   1.2 Use trial division by all primes less than $\sqrt{n}$ to determine whether $n$ is prime.
   1.3 If $n$ is prime then return($n$).
2. Set $c \leftarrow 0.1$ and $m \leftarrow 20$ (see Note 4.60).
3. (*Trial division bound*) Set $B \leftarrow c \cdot k^2$ (see Note 4.60).
4. (*Generate r, the size of q relative to n — see Note 4.61*) If $k > 2m$ then repeatedly do the following: select a random number $s$ in the interval $[0, 1]$, set $r \leftarrow 2^{s-1}$, until $(k - rk) > m$. Otherwise (i.e. $k \le 2m$), set $r \leftarrow 0.5$.
5. Compute $q \leftarrow$PROVABLE_PRIME($\lfloor r \cdot k \rfloor + 1$).
6. Set $I \leftarrow \lfloor 2^{k-1}/(2q) \rfloor$.
7. success$\leftarrow$0.
8. While (success $= 0$) do the following:
   8.1 (*select a candidate integer n*) Select a random integer $R$ in the interval $[I + 1, 2I]$ and set $n \leftarrow 2Rq + 1$.
   8.2 Use trial division to determine whether $n$ is divisible by any prime number $< B$. If it is not then do the following:
       Select a random integer $a$ in the interval $[2, n - 2]$.
       Compute $b \leftarrow a^{n-1} \bmod n$.
       If $b = 1$ then do the following:
           Compute $b \leftarrow a^{2R} \bmod n$ and $d \leftarrow \gcd(b - 1, n)$.
           If $d = 1$ then success$\leftarrow$1.
9. Return($n$).

**4.63 Note** (*improvements to Algorithm 4.62*)

(i) A speedup can be achieved by using Fact 4.42 instead of Fact 4.59(i) for proving $n = 2Rq + 1$ prime in step 8.2 of Maurer's algorithm — Fact 4.42 only requires that $q$ be greater than $\sqrt[3]{n}$.

(ii) If a candidate $n$ passes the trial division (in step 8.2), then a Miller-Rabin test (Algorithm 4.24) with the single base $a = 2$ should be performed on $n$; only if $n$ passes this test should the attempt to prove its primality (the remainder of step 8.2) be undertaken. This leads to a faster implementation due to the efficiency of the Miller-Rabin test with a single base $a = 2$ (cf. Remark 4.50).

(iii) Step 4 requires the use of real number arithmetic when computing $2^{s-1}$. To avoid these computations, one can precompute and store a list of such values for a selection of random numbers $s \in [0, 1]$.

**4.64 Note** (*provable primes vs. probable primes*) Probable primes are advantageous over provable primes in that Algorithm 4.44 for generating probable primes with $t = 1$ is slightly faster than Maurer's algorithm. Moreover, the latter requires more run-time memory due to its recursive nature. Provable primes are preferable to probable primes in the sense that the former have zero error probability. In any cryptographic application, however, there is always a non-zero error probability of some catastrophic failure, such as the adversary guessing a secret key or hardware failure. Since the error probability of probable primes can be efficiently brought down to acceptably low levels (see Note 4.49 but note the dependence on $t$), there appears to be no reason for mandating the use of provable primes over probable primes.

## 4.5 Irreducible polynomials over $\mathbb{Z}_p$

Recall (Definition 2.190) that a polynomial $f(x) \in \mathbb{Z}_p[x]$ of degree $m \geq 1$ is said to be *irreducible over* $\mathbb{Z}_p$ if it cannot be written as a product of two polynomials in $\mathbb{Z}_p[x]$ each having degree less than $m$. Such a polynomial $f(x)$ can be used to represent the elements of the finite field $\mathbb{F}_{p^m}$ as $\mathbb{F}_{p^m} = \mathbb{Z}_p[x]/(f(x))$, the set of all polynomials in $\mathbb{Z}_p[x]$ of degree less than $m$ where the addition and multiplication of polynomials is performed modulo $f(x)$ (see §2.6.3). This section presents techniques for constructing irreducible polynomials over $\mathbb{Z}_p$, where $p$ is a prime. The characteristic two finite fields $\mathbb{F}_{2^m}$ are of particular interest for cryptographic applications because the arithmetic in these fields can be efficiently performed both in software and in hardware. For this reason, additional attention is given to the special case of irreducible polynomials over $\mathbb{Z}_2$.

The arithmetic in finite fields can usually be implemented more efficiently if the irreducible polynomial chosen has few non-zero terms. Irreducible *trinomials*, i.e., irreducible polynomials having exactly three non-zero terms, are considered in §4.5.2. *Primitive* polynomials, i.e., irreducible polynomials $f(x)$ of degree $m$ in $\mathbb{Z}_p[x]$ for which $x$ is a generator of $\mathbb{F}^*_{p^m}$, the multiplicative group of the finite field $\mathbb{F}_{p^m} = \mathbb{Z}_p[x]/(f(x))$ (Definition 2.228), are the topic of §4.5.3. Primitive polynomials are also used in the generation of linear feedback shift register sequences having the maximum possible period (Fact 6.12).

### 4.5.1 Irreducible polynomials

If $f(x) \in \mathbb{Z}_p[x]$ is irreducible over $\mathbb{Z}_p$ and $a$ is a non-zero element in $\mathbb{Z}_p$, then $a\cdot f(x)$ is also irreducible over $\mathbb{Z}_p$. Hence it suffices to restrict attention to *monic* polynomials in $\mathbb{Z}_p[x]$, i.e., polynomials whose leading coefficient is 1. Observe also that if $f(x)$ is an irreducible polynomial, then its constant term must be non-zero. In particular, if $f(x) \in \mathbb{Z}_2[x]$, then its constant term must be 1.

There is a formula for computing exactly the number of monic irreducible polynomials in $\mathbb{Z}_p[x]$ of a fixed degree. The Möbius function, which is defined next, is used in this formula.

**4.65 Definition** Let $m$ be a positive integer. The *Möbius function* $\mu$ is defined by

$$\mu(m) = \begin{cases} 1, & \text{if } m = 1, \\ 0, & \text{if } m \text{ is divisible by the square of a prime,} \\ (-1)^k, & \text{if } m \text{ is the product of } k \text{ distinct primes.} \end{cases}$$

**4.66 Example** (*Möbius function*) The following table gives the values of the Möbius function $\mu(m)$ for the first 10 values of $m$:

| $m$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $\mu(m)$ | 1 | $-1$ | $-1$ | 0 | $-1$ | 1 | $-1$ | 0 | 0 | 1 |

□

**4.67 Fact** (*number of monic irreducible polynomials*) Let $p$ be a prime and $m$ a positive integer.

(i) The number $N_p(m)$ of monic irreducible polynomials of degree $m$ in $\mathbb{Z}_p[x]$ is given by the following formula:

$$N_p(m) \;=\; \frac{1}{m}\sum_{d|m}\mu(d)p^{m/d},$$

where the summation ranges over all positive divisors $d$ of $m$.

(ii) The probability of a random monic polynomial of degree $m$ in $\mathbb{Z}_p[x]$ being irreducible over $\mathbb{Z}_p$ is roughly $\frac{1}{m}$. More specifically, the number $N_p(m)$ satisfies

$$\frac{1}{2m} \;\le\; \frac{N_p(m)}{p^m} \;\approx\; \frac{1}{m}.$$

Testing irreducibility of polynomials in $\mathbb{Z}_p[x]$ is significantly simpler than testing primality of integers. A polynomial can be tested for irreducibility by verifying that it has no irreducible factors of degree $\le \lfloor \frac{m}{2} \rfloor$. The following result leads to an efficient method (Algorithm 4.69) for accomplishing this.

**4.68 Fact** Let $p$ be a prime and let $k$ be a positive integer.

(i) The product of all monic irreducible polynomials in $\mathbb{Z}_p[x]$ of degree dividing $k$ is equal to $x^{p^k} - x$.

(ii) Let $f(x)$ be a polynomial of degree $m$ in $\mathbb{Z}_p[x]$. Then $f(x)$ is irreducible over $\mathbb{Z}_p$ if and only if $\gcd(f(x), x^{p^i} - x) = 1$ for each $i$, $1 \le i \le \lfloor \frac{m}{2} \rfloor$.

**4.69 Algorithm** Testing a polynomial for irreducibility

INPUT: a prime $p$ and a monic polynomial $f(x)$ of degree $m$ in $\mathbb{Z}_p[x]$.
OUTPUT: an answer to the question: "Is $f(x)$ irreducible over $\mathbb{Z}_p$?"

1. Set $u(x) \leftarrow x$.
2. For $i$ from 1 to $\lfloor \frac{m}{2} \rfloor$ do the following:
   - 2.1 Compute $u(x) \leftarrow u(x)^p \bmod f(x)$ using Algorithm 2.227. (Note that $u(x)$ is a polynomial in $\mathbb{Z}_p[x]$ of degree less than $m$.)
   - 2.2 Compute $d(x) = \gcd(f(x), u(x) - x)$ (using Algorithm 2.218).
   - 2.3 If $d(x) \ne 1$ then return("reducible").
3. Return("irreducible").

Fact 4.67 suggests that one method for finding an irreducible polynomial of degree $m$ in $\mathbb{Z}_p[x]$ is to generate a random monic polynomial of degree $m$ in $\mathbb{Z}_p[x]$, test it for irreducibility, and continue until an irreducible one is found (Algorithm 4.70). The expected number of polynomials to be tried before an irreducible one is found is approximately $m$.

**4.70 Algorithm** Generating a random monic irreducible polynomial over $\mathbb{Z}_p$

INPUT: a prime $p$ and a positive integer $m$.
OUTPUT: a monic irreducible polynomial $f(x)$ of degree $m$ in $\mathbb{Z}_p[x]$.

1. Repeat the following:
   1.1 (*Generate a random monic polynomial of degree $m$ in $\mathbb{Z}_p[x]$*)
   Randomly select integers $a_0, a_1, a_2, \ldots, a_{m-1}$ between 0 and $p-1$ with $a_0 \neq 0$. Let $f(x)$ be the polynomial $f(x) = x^m + a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0$.
   1.2 Use Algorithm 4.69 to test whether $f(x)$ is irreducible over $\mathbb{Z}_p$.

   Until $f(x)$ is irreducible.
2. Return($f(x)$).

It is known that the expected degree of the irreducible factor of least degree of a random polynomial of degree $m$ in $\mathbb{Z}_p[x]$ is $O(\lg m)$. Hence for each choice of $f(x)$, the expected number of times steps 2.1 – 2.3 of Algorithm 4.69 are iterated is $O(\lg m)$. Each iteration takes $O((\lg p)m^2)$ $\mathbb{Z}_p$-operations. These observations, together with Fact 4.67(ii), determine the running time for Algorithm 4.70.

**4.71 Fact** Algorithm 4.70 has an expected running time of $O(m^3(\lg m)(\lg p))$ $\mathbb{Z}_p$-operations.

Given one irreducible polynomial of degree $m$ over $\mathbb{Z}_p$, Note 4.74 describes a method, which is more efficient than Algorithm 4.70, for randomly generating additional such polynomials.

**4.72 Definition** Let $\mathbb{F}_q$ be a finite field of characteristic $p$, and let $\alpha \in \mathbb{F}_q$. A *minimum polynomial* of $\alpha$ over $\mathbb{Z}_p$ is a monic polynomial of least degree in $\mathbb{Z}_p[x]$ having $\alpha$ as a root.

**4.73 Fact** Let $\mathbb{F}_q$ be a finite field of order $q = p^m$, and let $\alpha \in \mathbb{F}_q$.

(i) The minimum polynomial of $\alpha$ over $\mathbb{Z}_p$, denoted $m_\alpha(x)$, is unique.
(ii) $m_\alpha(x)$ is irreducible over $\mathbb{Z}_p$.
(iii) The degree of $m_\alpha(x)$ is a divisor of $m$.
(iv) Let $t$ be the smallest positive integer such that $\alpha^{p^t} = \alpha$. (Note that such a $t$ exists since, by Fact 2.213, $\alpha^{p^m} = \alpha$.) Then

$$m_\alpha(x) = \prod_{i=0}^{t-1}(x - \alpha^{p^i}). \tag{4.1}$$

**4.74 Note** (*generating new irreducible polynomials from a given one*) Suppose that $f(y)$ is a given irreducible polynomial of degree $m$ over $\mathbb{Z}_p$. The finite field $\mathbb{F}_{p^m}$ can then be represented as $\mathbb{F}_{p^m} = \mathbb{Z}_p[y]/(f(y))$. A random monic irreducible polynomial of degree $m$ over $\mathbb{Z}_p$ can be efficiently generated as follows. First generate a random element $\alpha \in \mathbb{F}_{p^m}$ and then, by repeated exponentiation by $p$, determine the smallest positive integer $t$ for which $\alpha^{p^t} = \alpha$. If $t < m$, then generate a new random element $\alpha \in \mathbb{F}_{p^m}$ and repeat; the probability that $t < m$ is known to be at most $(\lg m)/q^{m/2}$. If indeed $t = m$, then compute $m_\alpha(x)$ using the formula (4.1). Then $m_\alpha(x)$ is a random monic irreducible polynomial of degree $m$ in $\mathbb{Z}_p[x]$. This method has an expected running time of $O(m^3(\lg p))$ $\mathbb{Z}_p$-operations (compare with Fact 4.71).

### 4.5.2 Irreducible trinomials

If a polynomial $f(x)$ in $\mathbb{Z}_2[x]$ has an even number of non-zero terms, then $f(1) = 0$, whence $(x + 1)$ is a factor of $f(x)$. Hence, the smallest number of non-zero terms an irreducible polynomial of degree $\geq 2$ in $\mathbb{Z}_2[x]$ can have is three. An irreducible *trinomial* of degree $m$ in $\mathbb{Z}_2[x]$ must be of the form $x^m + x^k + 1$, where $1 \leq k \leq m-1$. Choosing an irreducible trinomial $f(x) \in \mathbb{Z}_2[x]$ of degree $m$ to represent the elements of the finite field $\mathbb{F}_{2^m} = \mathbb{Z}_2[x]/(f(x))$ can lead to a faster implementation of the field arithmetic. The following facts are sometimes of use when searching for irreducible trinomials.

**4.75 Fact** Let $m$ be a positive integer, and let $k$ denote an integer in the interval $[1, m-1]$.

(i) If the trinomial $x^m + x^k + 1$ is irreducible over $\mathbb{Z}_2$ then so is $x^m + x^{m-k} + 1$.
(ii) If $m \equiv 0 \pmod 8$, there is no irreducible trinomial of degree $m$ in $\mathbb{Z}_2[x]$.
(iii) Suppose that either $m \equiv 3 \pmod 8$ or $m \equiv 5 \pmod 8$. Then a necessary condition for $x^m + x^k + 1$ to be irreducible over $\mathbb{Z}_2$ is that either $k$ or $m-k$ must be of the form $2d$ for some positive divisor $d$ of $m$.

Tables 4.6 and 4.7 list an irreducible trinomial of degree $m$ over $\mathbb{Z}_2$ for each $m \leq 1478$ for which such a trinomial exists.

### 4.5.3 Primitive polynomials

Primitive polynomials were introduced at the beginning of §4.5. Let $f(x) \in \mathbb{Z}_p[x]$ be an irreducible polynomial of degree $m$. If the factorization of the integer $p^m - 1$ is known, then Fact 4.76 yields an efficient algorithm (Algorithm 4.77) for testing whether or not $f(x)$ is a primitive polynomial. If the factorization of $p^m - 1$ is unknown, there is no efficient algorithm known for performing this test.

**4.76 Fact** Let $p$ be a prime and let the distinct prime factors of $p^m - 1$ be $r_1, r_2, \ldots, r_t$. Then an irreducible polynomial $f(x) \in \mathbb{Z}_p[x]$ is primitive if and only if for each $i$, $1 \leq i \leq t$:

$$x^{(p^m-1)/r_i} \not\equiv 1 \pmod{f(x)}.$$

(That is, $x$ is an element of order $p^m - 1$ in the field $\mathbb{Z}_p[x]/(f(x))$.)

**4.77 Algorithm** Testing whether an irreducible polynomial is primitive

INPUT: a prime $p$, a positive integer $m$, the distinct prime factors $r_1, r_2, \ldots, r_t$ of $p^m - 1$, and a monic irreducible polynomial $f(x)$ of degree $m$ in $\mathbb{Z}_p[x]$.
OUTPUT: an answer to the question: "Is $f(x)$ a primitive polynomial?"

1. For $i$ from 1 to $t$ do the following:
   1.1 Compute $l(x) = x^{(p^m-1)/r_i} \bmod f(x)$ (using Algorithm 2.227).
   1.2 If $l(x) = 1$ then return("not primitive").
2. Return("primitive").

There are precisely $\phi(p^m - 1)/m$ monic primitive polynomials of degree $m$ in $\mathbb{Z}_p[x]$ (Fact 2.230), where $\phi$ is the Euler phi function (Definition 2.100). Since the number of monic irreducible polynomials of degree $m$ in $\mathbb{Z}_p[x]$ is roughly $p^m/m$ (Fact 4.67(ii)), it follows that the probability of a random monic irreducible polynomial of degree $m$ in $\mathbb{Z}_p[x]$

| $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 93 | 2 | 193 | 15 | 295 | 48 | 402 | 171 | 508 | 9 | 618 | 295 |
| 3 | 1 | 94 | 21 | 194 | 87 | 297 | 5 | 404 | 65 | 510 | 69 | 620 | 9 |
| 4 | 1 | 95 | 11 | 196 | 3 | 300 | 5 | 406 | 141 | 511 | 10 | 622 | 297 |
| 5 | 2 | 97 | 6 | 198 | 9 | 302 | 41 | 407 | 71 | 513 | 26 | 623 | 68 |
| 6 | 1 | 98 | 11 | 199 | 34 | 303 | 1 | 409 | 87 | 514 | 67 | 625 | 133 |
| 7 | 1 | 100 | 15 | 201 | 14 | 305 | 102 | 412 | 147 | 516 | 21 | 626 | 251 |
| 9 | 1 | 102 | 29 | 202 | 55 | 308 | 15 | 414 | 13 | 518 | 33 | 628 | 223 |
| 10 | 3 | 103 | 9 | 204 | 27 | 310 | 93 | 415 | 102 | 519 | 79 | 631 | 307 |
| 11 | 2 | 105 | 4 | 207 | 43 | 313 | 79 | 417 | 107 | 521 | 32 | 633 | 101 |
| 12 | 3 | 106 | 15 | 209 | 6 | 314 | 15 | 418 | 199 | 522 | 39 | 634 | 39 |
| 14 | 5 | 108 | 17 | 210 | 7 | 316 | 63 | 420 | 7 | 524 | 167 | 636 | 217 |
| 15 | 1 | 110 | 33 | 212 | 105 | 318 | 45 | 422 | 149 | 526 | 97 | 639 | 16 |
| 17 | 3 | 111 | 10 | 214 | 73 | 319 | 36 | 423 | 25 | 527 | 47 | 641 | 11 |
| 18 | 3 | 113 | 9 | 215 | 23 | 321 | 31 | 425 | 12 | 529 | 42 | 642 | 119 |
| 20 | 3 | 118 | 33 | 217 | 45 | 322 | 67 | 426 | 63 | 532 | 1 | 646 | 249 |
| 21 | 2 | 119 | 8 | 218 | 11 | 324 | 51 | 428 | 105 | 534 | 161 | 647 | 5 |
| 22 | 1 | 121 | 18 | 220 | 7 | 327 | 34 | 431 | 120 | 537 | 94 | 649 | 37 |
| 23 | 5 | 123 | 2 | 223 | 33 | 329 | 50 | 433 | 33 | 538 | 195 | 650 | 3 |
| 25 | 3 | 124 | 19 | 225 | 32 | 330 | 99 | 436 | 165 | 540 | 9 | 651 | 14 |
| 28 | 1 | 126 | 21 | 228 | 113 | 332 | 89 | 438 | 65 | 543 | 16 | 652 | 93 |
| 29 | 2 | 127 | 1 | 231 | 26 | 333 | 2 | 439 | 49 | 545 | 122 | 654 | 33 |
| 30 | 1 | 129 | 5 | 233 | 74 | 337 | 55 | 441 | 7 | 550 | 193 | 655 | 88 |
| 31 | 3 | 130 | 3 | 234 | 31 | 340 | 45 | 444 | 81 | 551 | 135 | 657 | 38 |
| 33 | 10 | 132 | 17 | 236 | 5 | 342 | 125 | 446 | 105 | 553 | 39 | 658 | 55 |
| 34 | 7 | 134 | 57 | 238 | 73 | 343 | 75 | 447 | 73 | 556 | 153 | 660 | 11 |
| 35 | 2 | 135 | 11 | 239 | 36 | 345 | 22 | 449 | 134 | 558 | 73 | 662 | 21 |
| 36 | 9 | 137 | 21 | 241 | 70 | 346 | 63 | 450 | 47 | 559 | 34 | 663 | 107 |
| 39 | 4 | 140 | 15 | 242 | 95 | 348 | 103 | 455 | 38 | 561 | 71 | 665 | 33 |
| 41 | 3 | 142 | 21 | 244 | 111 | 350 | 53 | 457 | 16 | 564 | 163 | 668 | 147 |
| 42 | 7 | 145 | 52 | 247 | 82 | 351 | 34 | 458 | 203 | 566 | 153 | 670 | 153 |
| 44 | 5 | 146 | 71 | 249 | 35 | 353 | 69 | 460 | 19 | 567 | 28 | 671 | 15 |
| 46 | 1 | 147 | 14 | 250 | 103 | 354 | 99 | 462 | 73 | 569 | 77 | 673 | 28 |
| 47 | 5 | 148 | 27 | 252 | 15 | 358 | 57 | 463 | 93 | 570 | 67 | 676 | 31 |
| 49 | 9 | 150 | 53 | 253 | 46 | 359 | 68 | 465 | 31 | 574 | 13 | 679 | 66 |
| 52 | 3 | 151 | 3 | 255 | 52 | 362 | 63 | 468 | 27 | 575 | 146 | 682 | 171 |
| 54 | 9 | 153 | 1 | 257 | 12 | 364 | 9 | 470 | 9 | 577 | 25 | 684 | 209 |
| 55 | 7 | 154 | 15 | 258 | 71 | 366 | 29 | 471 | 1 | 580 | 237 | 686 | 197 |
| 57 | 4 | 155 | 62 | 260 | 15 | 367 | 21 | 473 | 200 | 582 | 85 | 687 | 13 |
| 58 | 19 | 156 | 9 | 263 | 93 | 369 | 91 | 474 | 191 | 583 | 130 | 689 | 14 |
| 60 | 1 | 159 | 31 | 265 | 42 | 370 | 139 | 476 | 9 | 585 | 88 | 690 | 79 |
| 62 | 29 | 161 | 18 | 266 | 47 | 372 | 111 | 478 | 121 | 588 | 35 | 692 | 299 |
| 63 | 1 | 162 | 27 | 268 | 25 | 375 | 16 | 479 | 104 | 590 | 93 | 694 | 169 |
| 65 | 18 | 166 | 37 | 270 | 53 | 377 | 41 | 481 | 138 | 593 | 86 | 695 | 177 |
| 66 | 3 | 167 | 6 | 271 | 58 | 378 | 43 | 484 | 105 | 594 | 19 | 697 | 267 |
| 68 | 9 | 169 | 34 | 273 | 23 | 380 | 47 | 486 | 81 | 596 | 273 | 698 | 215 |
| 71 | 6 | 170 | 11 | 274 | 67 | 382 | 81 | 487 | 94 | 599 | 30 | 700 | 75 |
| 73 | 25 | 172 | 1 | 276 | 63 | 383 | 90 | 489 | 83 | 601 | 201 | 702 | 37 |
| 74 | 35 | 174 | 13 | 278 | 5 | 385 | 6 | 490 | 219 | 602 | 215 | 705 | 17 |
| 76 | 21 | 175 | 6 | 279 | 5 | 386 | 83 | 492 | 7 | 604 | 105 | 708 | 15 |
| 79 | 9 | 177 | 8 | 281 | 93 | 388 | 159 | 494 | 17 | 606 | 165 | 711 | 92 |
| 81 | 4 | 178 | 31 | 282 | 35 | 390 | 9 | 495 | 76 | 607 | 105 | 713 | 41 |
| 84 | 5 | 180 | 3 | 284 | 53 | 391 | 28 | 497 | 78 | 609 | 31 | 714 | 23 |
| 86 | 21 | 182 | 81 | 286 | 69 | 393 | 7 | 498 | 155 | 610 | 127 | 716 | 183 |
| 87 | 13 | 183 | 56 | 287 | 71 | 394 | 135 | 500 | 27 | 612 | 81 | 718 | 165 |
| 89 | 38 | 185 | 24 | 289 | 21 | 396 | 25 | 503 | 3 | 614 | 45 | 719 | 150 |
| 90 | 27 | 186 | 11 | 292 | 37 | 399 | 26 | 505 | 156 | 615 | 211 | 721 | 9 |
| 92 | 21 | 191 | 9 | 294 | 33 | 401 | 152 | 506 | 23 | 617 | 200 | 722 | 231 |

**Table 4.6:** *Irreducible trinomials $x^m + x^k + 1$ over $\mathbb{Z}_2$. For each $m$, $1 \le m \le 722$, for which an irreducible trinomial of degree $m$ in $\mathbb{Z}_2[x]$ exists, the table lists the smallest $k$ for which $x^m + x^k + 1$ is irreducible over $\mathbb{Z}_2$.*

| $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ | $m$ | $k$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 724 | 207 | 831 | 49 | 937 | 217 | 1050 | 159 | 1159 | 66 | 1265 | 119 | 1374 | 609 |
| 726 | 5 | 833 | 149 | 938 | 207 | 1052 | 291 | 1161 | 365 | 1266 | 7 | 1375 | 52 |
| 727 | 180 | 834 | 15 | 942 | 45 | 1054 | 105 | 1164 | 19 | 1268 | 345 | 1377 | 100 |
| 729 | 58 | 838 | 61 | 943 | 24 | 1055 | 24 | 1166 | 189 | 1270 | 333 | 1380 | 183 |
| 730 | 147 | 839 | 54 | 945 | 77 | 1057 | 198 | 1167 | 133 | 1271 | 17 | 1383 | 130 |
| 732 | 343 | 841 | 144 | 948 | 189 | 1058 | 27 | 1169 | 114 | 1273 | 168 | 1385 | 12 |
| 735 | 44 | 842 | 47 | 951 | 260 | 1060 | 439 | 1170 | 27 | 1276 | 217 | 1386 | 219 |
| 737 | 5 | 844 | 105 | 953 | 168 | 1062 | 49 | 1174 | 133 | 1278 | 189 | 1388 | 11 |
| 738 | 347 | 845 | 2 | 954 | 131 | 1063 | 168 | 1175 | 476 | 1279 | 216 | 1390 | 129 |
| 740 | 135 | 846 | 105 | 956 | 305 | 1065 | 463 | 1177 | 16 | 1281 | 229 | 1391 | 3 |
| 742 | 85 | 847 | 136 | 959 | 143 | 1071 | 7 | 1178 | 375 | 1282 | 231 | 1393 | 300 |
| 743 | 90 | 849 | 253 | 961 | 18 | 1078 | 361 | 1180 | 25 | 1284 | 223 | 1396 | 97 |
| 745 | 258 | 850 | 111 | 964 | 103 | 1079 | 230 | 1182 | 77 | 1286 | 153 | 1398 | 601 |
| 746 | 351 | 852 | 159 | 966 | 201 | 1081 | 24 | 1183 | 87 | 1287 | 470 | 1399 | 55 |
| 748 | 19 | 855 | 29 | 967 | 36 | 1082 | 407 | 1185 | 134 | 1289 | 99 | 1401 | 92 |
| 750 | 309 | 857 | 119 | 969 | 31 | 1084 | 189 | 1186 | 171 | 1294 | 201 | 1402 | 127 |
| 751 | 18 | 858 | 207 | 972 | 7 | 1085 | 62 | 1188 | 75 | 1295 | 38 | 1404 | 81 |
| 753 | 158 | 860 | 35 | 975 | 19 | 1086 | 189 | 1190 | 233 | 1297 | 198 | 1407 | 47 |
| 754 | 19 | 861 | 14 | 977 | 15 | 1087 | 112 | 1191 | 196 | 1298 | 399 | 1409 | 194 |
| 756 | 45 | 862 | 349 | 979 | 178 | 1089 | 91 | 1193 | 173 | 1300 | 75 | 1410 | 383 |
| 758 | 233 | 865 | 1 | 982 | 177 | 1090 | 79 | 1196 | 281 | 1302 | 77 | 1412 | 125 |
| 759 | 98 | 866 | 75 | 983 | 230 | 1092 | 23 | 1198 | 405 | 1305 | 326 | 1414 | 429 |
| 761 | 3 | 868 | 145 | 985 | 222 | 1094 | 57 | 1199 | 114 | 1306 | 39 | 1415 | 282 |
| 762 | 83 | 870 | 301 | 986 | 3 | 1095 | 139 | 1201 | 171 | 1308 | 495 | 1417 | 342 |
| 767 | 168 | 871 | 378 | 988 | 121 | 1097 | 14 | 1202 | 287 | 1310 | 333 | 1420 | 33 |
| 769 | 120 | 873 | 352 | 990 | 161 | 1098 | 83 | 1204 | 43 | 1311 | 476 | 1422 | 49 |
| 772 | 7 | 876 | 149 | 991 | 39 | 1100 | 35 | 1206 | 513 | 1313 | 164 | 1423 | 15 |
| 774 | 185 | 879 | 11 | 993 | 62 | 1102 | 117 | 1207 | 273 | 1314 | 19 | 1425 | 28 |
| 775 | 93 | 881 | 78 | 994 | 223 | 1103 | 65 | 1209 | 118 | 1319 | 129 | 1426 | 103 |
| 777 | 29 | 882 | 99 | 996 | 65 | 1105 | 21 | 1210 | 243 | 1321 | 52 | 1428 | 27 |
| 778 | 375 | 884 | 173 | 998 | 101 | 1106 | 195 | 1212 | 203 | 1324 | 337 | 1430 | 33 |
| 780 | 13 | 887 | 147 | 999 | 59 | 1108 | 327 | 1214 | 257 | 1326 | 397 | 1431 | 17 |
| 782 | 329 | 889 | 127 | 1001 | 17 | 1110 | 417 | 1215 | 302 | 1327 | 277 | 1433 | 387 |
| 783 | 68 | 890 | 183 | 1007 | 75 | 1111 | 13 | 1217 | 393 | 1329 | 73 | 1434 | 363 |
| 785 | 92 | 892 | 31 | 1009 | 55 | 1113 | 107 | 1218 | 91 | 1332 | 95 | 1436 | 83 |
| 791 | 30 | 894 | 173 | 1010 | 99 | 1116 | 59 | 1220 | 413 | 1334 | 617 | 1438 | 357 |
| 793 | 253 | 895 | 12 | 1012 | 115 | 1119 | 283 | 1223 | 255 | 1335 | 392 | 1441 | 322 |
| 794 | 143 | 897 | 113 | 1014 | 385 | 1121 | 62 | 1225 | 234 | 1337 | 75 | 1442 | 395 |
| 798 | 53 | 898 | 207 | 1015 | 186 | 1122 | 427 | 1226 | 167 | 1338 | 315 | 1444 | 595 |
| 799 | 25 | 900 | 1 | 1020 | 135 | 1126 | 105 | 1228 | 27 | 1340 | 125 | 1446 | 421 |
| 801 | 217 | 902 | 21 | 1022 | 317 | 1127 | 27 | 1230 | 433 | 1343 | 348 | 1447 | 195 |
| 804 | 75 | 903 | 35 | 1023 | 7 | 1129 | 103 | 1231 | 105 | 1345 | 553 | 1449 | 13 |
| 806 | 21 | 905 | 117 | 1025 | 294 | 1130 | 551 | 1233 | 151 | 1348 | 553 | 1452 | 315 |
| 807 | 7 | 906 | 123 | 1026 | 35 | 1134 | 129 | 1234 | 427 | 1350 | 237 | 1454 | 297 |
| 809 | 15 | 908 | 143 | 1028 | 119 | 1135 | 9 | 1236 | 49 | 1351 | 39 | 1455 | 52 |
| 810 | 159 | 911 | 204 | 1029 | 98 | 1137 | 277 | 1238 | 153 | 1353 | 371 | 1457 | 314 |
| 812 | 29 | 913 | 91 | 1030 | 93 | 1138 | 31 | 1239 | 4 | 1354 | 255 | 1458 | 243 |
| 814 | 21 | 916 | 183 | 1031 | 68 | 1140 | 141 | 1241 | 54 | 1356 | 131 | 1460 | 185 |
| 815 | 333 | 918 | 77 | 1033 | 108 | 1142 | 357 | 1242 | 203 | 1358 | 117 | 1463 | 575 |
| 817 | 52 | 919 | 36 | 1034 | 75 | 1145 | 227 | 1246 | 25 | 1359 | 98 | 1465 | 39 |
| 818 | 119 | 921 | 221 | 1036 | 411 | 1146 | 131 | 1247 | 14 | 1361 | 56 | 1466 | 311 |
| 820 | 123 | 924 | 31 | 1039 | 21 | 1148 | 23 | 1249 | 187 | 1362 | 655 | 1468 | 181 |
| 822 | 17 | 926 | 365 | 1041 | 412 | 1151 | 90 | 1252 | 97 | 1364 | 239 | 1470 | 49 |
| 823 | 9 | 927 | 403 | 1042 | 439 | 1153 | 241 | 1255 | 589 | 1366 | 1 | 1471 | 25 |
| 825 | 38 | 930 | 31 | 1044 | 41 | 1154 | 75 | 1257 | 289 | 1367 | 134 | 1473 | 77 |
| 826 | 255 | 932 | 177 | 1047 | 10 | 1156 | 307 | 1260 | 21 | 1369 | 88 | 1476 | 21 |
| 828 | 189 | 935 | 417 | 1049 | 141 | 1158 | 245 | 1263 | 77 | 1372 | 181 | 1478 | 69 |

***Table 4.7:*** *Irreducible trinomials $x^m+x^k+1$ over $\mathbb{Z}_2$. For each $m$, $723 \leq m \leq 1478$, for which an irreducible trinomial of degree $m$ in $\mathbb{Z}_2[x]$ exists, the table gives the smallest $k$ for which $x^m+x^k+1$ is irreducible over $\mathbb{Z}_2$.*

being primitive is approximately $\phi(p^m - 1)/p^m$. Using the lower bound for the Euler phi function (Fact 2.102), this probability can be seen to be at least $1/(6 \ln \ln p^m)$. This suggests the following algorithm for generating primitive polynomials.

**4.78 Algorithm** Generating a random monic primitive polynomial over $\mathbb{Z}_p$

INPUT: a prime $p$, integer $m \geq 1$, and the distinct prime factors $r_1, r_2, \ldots, r_t$ of $p^m - 1$.
OUTPUT: a monic primitive polynomial $f(x)$ of degree $m$ in $\mathbb{Z}_p[x]$.

1. Repeat the following:
   - 1.1 Use Algorithm 4.70 to generate a random monic irreducible polynomial $f(x)$ of degree $m$ in $\mathbb{Z}_p[x]$.
   - 1.2 Use Algorithm 4.77 to test whether $f(x)$ is primitive.

   Until $f(x)$ is primitive.
2. Return($f(x)$).

For each $m$, $1 \leq m \leq 229$, Table 4.8 lists a polynomial of degree $m$ that is primitive over $\mathbb{Z}_2$. If there exists a primitive trinomial $f(x) = x^m + x^k + 1$, then the trinomial with the smallest $k$ is listed. If no primitive trinomial exists, then a primitive pentanomial of the form $f(x) = x^m + x^{k_1} + x^{k_2} + x^{k_3} + 1$ is listed.

If $p^m - 1$ is prime, then Fact 4.76 implies that every irreducible polynomial of degree $m$ in $\mathbb{Z}_p[x]$ is also primitive. Table 4.9 gives either a primitive trinomial or a primitive pentanomial of degree $m$ over $\mathbb{Z}_2$ where $m$ is an exponent of one of the first 27 Mersenne primes (Definition 4.35).

## 4.6 Generators and elements of high order

Recall (Definition 2.169) that if $G$ is a (multiplicative) finite group, the *order* of an element $a \in G$ is the least positive integer $t$ such that $a^t = 1$. If there are $n$ elements in $G$, and if $a \in G$ is an element of order $n$, then $G$ is said to be *cyclic* and $a$ is called a *generator* or a *primitive element* of $G$ (Definition 2.167). Of special interest for cryptographic applications are the multiplicative group $\mathbb{Z}_p^*$ of the integers modulo a prime $p$, and the multiplicative group $\mathbb{F}_{2^m}^*$ of the finite field $\mathbb{F}_{2^m}$ of characteristic two; these groups are cyclic (Fact 2.213). Also of interest is the group $\mathbb{Z}_n^*$ (Definition 2.124), where $n$ is the product of two distinct odd primes. This section deals with the problem of finding generators and other elements of high order in $\mathbb{Z}_p^*$, $\mathbb{F}_{2^m}^*$, and $\mathbb{Z}_n^*$. See §2.5.1 for background in group theory and §2.6 for background in finite fields.

Algorithm 4.79 is an efficient method for determining the order of a group element, given the prime factorization of the group order $n$. The correctness of the algorithm follows from the fact that the order of an element must divide $n$ (Fact 2.171).

| $m$ | $k$ or $(k_1,k_2,k_3)$ | $m$ | $k$ or $(k_1,k_2,k_3)$ | $m$ | $k$ or $(k_1,k_2,k_3)$ | $m$ | $k$ or $(k_1,k_2,k_3)$ |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 59 | 22, 21, 1 | 116 | 71, 70, 1 | 173 | 100, 99, 1 |
| 3 | 1 | 60 | 1 | 117 | 20, 18, 2 | 174 | 13 |
| 4 | 1 | 61 | 16, 15, 1 | 118 | 33 | 175 | 6 |
| 5 | 2 | 62 | 57, 56, 1 | 119 | 8 | 176 | 119, 118, 1 |
| 6 | 1 | 63 | 1 | 120 | 118, 111, 7 | 177 | 8 |
| 7 | 1 | 64 | 4, 3, 1 | 121 | 18 | 178 | 87 |
| 8 | 6, 5, 1 | 65 | 18 | 122 | 60, 59, 1 | 179 | 34, 33, 1 |
| 9 | 4 | 66 | 10, 9, 1 | 123 | 2 | 180 | 37, 36, 1 |
| 10 | 3 | 67 | 10, 9, 1 | 124 | 37 | 181 | 7, 6, 1 |
| 11 | 2 | 68 | 9 | 125 | 108, 107, 1 | 182 | 128, 127, 1 |
| 12 | 7, 4, 3 | 69 | 29, 27, 2 | 126 | 37, 36, 1 | 183 | 56 |
| 13 | 4, 3, 1 | 70 | 16, 15, 1 | 127 | 1 | 184 | 102, 101, 1 |
| 14 | 12, 11, 1 | 71 | 6 | 128 | 29, 27, 2 | 185 | 24 |
| 15 | 1 | 72 | 53, 47, 6 | 129 | 5 | 186 | 23, 22, 1 |
| 16 | 5, 3, 2 | 73 | 25 | 130 | 3 | 187 | 58, 57, 1 |
| 17 | 3 | 74 | 16, 15, 1 | 131 | 48, 47, 1 | 188 | 74, 73, 1 |
| 18 | 7 | 75 | 11, 10, 1 | 132 | 29 | 189 | 127, 126, 1 |
| 19 | 6, 5, 1 | 76 | 36, 35, 1 | 133 | 52, 51, 1 | 190 | 18, 17, 1 |
| 20 | 3 | 77 | 31, 30, 1 | 134 | 57 | 191 | 9 |
| 21 | 2 | 78 | 20, 19, 1 | 135 | 11 | 192 | 28, 27, 1 |
| 22 | 1 | 79 | 9 | 136 | 126, 125, 1 | 193 | 15 |
| 23 | 5 | 80 | 38, 37, 1 | 137 | 21 | 194 | 87 |
| 24 | 4, 3, 1 | 81 | 4 | 138 | 8, 7, 1 | 195 | 10, 9, 1 |
| 25 | 3 | 82 | 38, 35, 3 | 139 | 8, 5, 3 | 196 | 66, 65, 1 |
| 26 | 8, 7, 1 | 83 | 46, 45, 1 | 140 | 29 | 197 | 62, 61, 1 |
| 27 | 8, 7, 1 | 84 | 13 | 141 | 32, 31, 1 | 198 | 65 |
| 28 | 3 | 85 | 28, 27, 1 | 142 | 21 | 199 | 34 |
| 29 | 2 | 86 | 13, 12, 1 | 143 | 21, 20, 1 | 200 | 42, 41, 1 |
| 30 | 16, 15, 1 | 87 | 13 | 144 | 70, 69, 1 | 201 | 14 |
| 31 | 3 | 88 | 72, 71, 1 | 145 | 52 | 202 | 55 |
| 32 | 28, 27, 1 | 89 | 38 | 146 | 60, 59, 1 | 203 | 8, 7, 1 |
| 33 | 13 | 90 | 19, 18, 1 | 147 | 38, 37, 1 | 204 | 74, 73, 1 |
| 34 | 15, 14, 1 | 91 | 84, 83, 1 | 148 | 27 | 205 | 30, 29, 1 |
| 35 | 2 | 92 | 13, 12, 1 | 149 | 110, 109, 1 | 206 | 29, 28, 1 |
| 36 | 11 | 93 | 2 | 150 | 53 | 207 | 43 |
| 37 | 12, 10, 2 | 94 | 21 | 151 | 3 | 208 | 62, 59, 3 |
| 38 | 6, 5, 1 | 95 | 11 | 152 | 66, 65, 1 | 209 | 6 |
| 39 | 4 | 96 | 49, 47, 2 | 153 | 1 | 210 | 35, 32, 3 |
| 40 | 21, 19, 2 | 97 | 6 | 154 | 129, 127, 2 | 211 | 46, 45, 1 |
| 41 | 3 | 98 | 11 | 155 | 32, 31, 1 | 212 | 105 |
| 42 | 23, 22, 1 | 99 | 47, 45, 2 | 156 | 116, 115, 1 | 213 | 8, 7, 1 |
| 43 | 6, 5, 1 | 100 | 37 | 157 | 27, 26, 1 | 214 | 49, 48, 1 |
| 44 | 27, 26, 1 | 101 | 7, 6, 1 | 158 | 27, 26, 1 | 215 | 23 |
| 45 | 4, 3, 1 | 102 | 77, 76, 1 | 159 | 31 | 216 | 196, 195, 1 |
| 46 | 21, 20, 1 | 103 | 9 | 160 | 19, 18, 1 | 217 | 45 |
| 47 | 5 | 104 | 11, 10, 1 | 161 | 18 | 218 | 11 |
| 48 | 28, 27, 1 | 105 | 16 | 162 | 88, 87, 1 | 219 | 19, 18, 1 |
| 49 | 9 | 106 | 15 | 163 | 60, 59, 1 | 220 | 15, 14, 1 |
| 50 | 27, 26, 1 | 107 | 65, 63, 2 | 164 | 14, 13, 1 | 221 | 35, 34, 1 |
| 51 | 16, 15, 1 | 108 | 31 | 165 | 31, 30, 1 | 222 | 92, 91, 1 |
| 52 | 3 | 109 | 7, 6, 1 | 166 | 39, 38, 1 | 223 | 33 |
| 53 | 16, 15, 1 | 110 | 13, 12, 1 | 167 | 6 | 224 | 31, 30, 1 |
| 54 | 37, 36, 1 | 111 | 10 | 168 | 17, 15, 2 | 225 | 32 |
| 55 | 24 | 112 | 45, 43, 2 | 169 | 34 | 226 | 58, 57, 1 |
| 56 | 22, 21, 1 | 113 | 9 | 170 | 23 | 227 | 46, 45, 1 |
| 57 | 7 | 114 | 82, 81, 1 | 171 | 19, 18, 1 | 228 | 148, 147, 1 |
| 58 | 19 | 115 | 15, 14, 1 | 172 | 7 | 229 | 64, 63, 1 |

**Table 4.8:** *Primitive polynomials over $\mathbb{Z}_2$. For each $m$, $1 \le m \le 229$, an exponent $k$ is given for which the trinomial $x^m + x^k + 1$ is primitive over $\mathbb{Z}_2$. If no such trinomial exists, a triple of exponents $(k_1, k_2, k_3)$ is given for which the pentanomial $x^m + x^{k_1} + x^{k_2} + x^{k_3} + 1$ is primitive over $\mathbb{Z}_2$.*

| $j$ | $m$ | $k$ $(k_1, k_2, k_3)$ |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 3 | 1 |
| 3 | 5 | 2 |
| 4 | 7 | 1, 3 |
| 5 | 13 | none (4,3,1) |
| 6 | 17 | 3, 5, 6 |
| 7 | 19 | none (5,2,1) |
| 8 | 31 | 3, 6, 7, 13 |
| 9 | 61 | none (43,26,14) |
| 10 | 89 | 38 |
| 11 | 107 | none (82,57,31) |
| 12 | 127 | 1, 7, 15, 30, 63 |
| 13 | 521 | 32, 48, 158, 168 |
| 14 | 607 | 105, 147, 273 |
| 15 | 1279 | 216, 418 |
| 16 | 2203 | none (1656,1197,585) |
| 17 | 2281 | 715, 915, 1029 |
| 18 | 3217 | 67, 576 |
| 19 | 4253 | none (3297,2254,1093) |
| 20 | 4423 | 271, 369, 370, 649, 1393, 1419, 2098 |
| 21 | 9689 | 84, 471, 1836, 2444, 4187 |
| 22 | 9941 | none (7449,4964,2475) |
| 23 | 11213 | none (8218,6181,2304) |
| 24 | 19937 | 881, 7083, 9842 |
| 25 | 21701 | none (15986,11393,5073) |
| 26 | 23209 | 1530, 6619, 9739 |
| 27 | 44497 | 8575, 21034 |

**Table 4.9:** *Primitive polynomials of degree $m$ over $\mathbb{Z}_2$, $2^m-1$ a Mersenne prime. For each exponent $m = M_j$ of the first 27 Mersenne primes, the table lists all values of $k$, $1 \le k \le m/2$, for which the trinomial $x^m + x^k + 1$ is irreducible over $\mathbb{Z}_2$. If no such trinomial exists, a triple of exponents $(k_1, k_2, k_3)$ is listed such that the pentanomial $x^m + x^{k_1} + x^{k_2} + x^{k_3} + 1$ is irreducible over $\mathbb{Z}_2$.*

**4.79 Algorithm** Determining the order of a group element

INPUT: a (multiplicative) finite group $G$ of order $n$, an element $a \in G$, and the prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.
OUTPUT: the order $t$ of $a$.

1. Set $t \leftarrow n$.
2. For $i$ from 1 to $k$ do the following:
   - 2.1 Set $t \leftarrow t/p_i^{e_i}$.
   - 2.2 Compute $a_1 \leftarrow a^t$.
   - 2.3 While $a_1 \neq 1$ do the following: compute $a_1 \leftarrow a_1^{p_i}$ and set $t \leftarrow t \cdot p_i$.
3. Return($t$).

Suppose now that $G$ is a cyclic group of order $n$. Then for any divisor $d$ of $n$ the number of elements of order $d$ in $G$ is exactly $\phi(d)$ (Fact 2.173(ii)), where $\phi$ is the Euler phi function (Definition 2.100). In particular, $G$ has exactly $\phi(n)$ generators, and hence the probability of a random element in $G$ being a generator is $\phi(n)/n$. Using the lower bound for the Euler phi function (Fact 2.102), this probability can be seen to be at least $1/(6 \ln \ln n)$. This suggests the following efficient randomized algorithm for finding a generator of a cyclic group.

**4.80 Algorithm** Finding a generator of a cyclic group

INPUT: a cyclic group $G$ of order $n$, and the prime factorization $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$.
OUTPUT: a generator $\alpha$ of $G$.

1. Choose a random element $\alpha$ in $G$.
2. For $i$ from 1 to $k$ do the following:
   - 2.1 Compute $b \leftarrow \alpha^{n/p_i}$.
   - 2.2 If $b = 1$ then go to step 1.
3. Return($\alpha$).

**4.81 Note** (*group elements of high order*) In some situations it may be desirable to have an element of high order, and not a generator. Given a generator $\alpha$ in a cyclic group $G$ of order $n$, and given a divisor $d$ of $n$, an element $\beta$ of order $d$ in $G$ can be efficiently obtained as follows: $\beta = \alpha^{n/d}$. If $q$ is a prime divisor of the order $n$ of a cyclic group $G$, then the following method finds an element $\beta \in G$ of order $q$ without first having to find a generator of $G$: select a random element $g \in G$ and compute $\beta = g^{n/q}$; repeat until $\beta \neq 1$.

**4.82 Note** (*generators of* $\mathbb{F}_{2^m}^*$) There are two basic approaches to finding a generator of $\mathbb{F}_{2^m}^*$. Both techniques require the factorization of the order of $\mathbb{F}_{2^m}^*$, namely $2^m - 1$.

(i) Generate a monic primitive polynomial $f(x)$ of degree $m$ over $\mathbb{Z}_2$ (Algorithm 4.78). The finite field $\mathbb{F}_{2^m}$ can then be represented as $\mathbb{Z}_2[x]/(f(x))$, the set of all polynomials over $\mathbb{Z}_2$ modulo $f(x)$, and the element $\alpha = x$ is a generator.
(ii) Select the method for representing elements of $\mathbb{F}_{2^m}$ first. Then use Algorithm 4.80 with $G = \mathbb{F}_{2^m}^*$ and $n = 2^m - 1$ to find a generator $\alpha$ of $\mathbb{F}_{2^m}^*$.

If $n = pq$, where $p$ and $q$ are distinct odd primes, then $\mathbb{Z}_n^*$ is a non-cyclic group of order $\phi(n) = (p-1)(q-1)$. The maximum order of an element in $\mathbb{Z}_n^*$ is $\text{lcm}(p-1, q-1)$. Algorithm 4.83 is a method for generating such an element which requires the factorizations of $p - 1$ and $q - 1$.

**4.83 Algorithm** Selecting an element of maximum order in $\mathbb{Z}_n^*$, where $n = pq$

INPUT: two distinct odd primes, $p$, $q$, and the factorizations of $p - 1$ and $q - 1$.
OUTPUT: an element $\alpha$ of maximum order $\text{lcm}(p-1, q-1)$ in $\mathbb{Z}_n^*$, where $n = pq$.

1. Use Algorithm 4.80 with $G = \mathbb{Z}_p^*$ and $n = p - 1$ to find a generator $a$ of $\mathbb{Z}_p^*$.
2. Use Algorithm 4.80 with $G = \mathbb{Z}_q^*$ and $n = q - 1$ to find a generator $b$ of $\mathbb{Z}_q^*$.
3. Use Gauss's algorithm (Algorithm 2.121) to find an integer $\alpha$, $1 \leq \alpha \leq n - 1$, satisfying $\alpha \equiv a \pmod{p}$ and $\alpha \equiv b \pmod{q}$.
4. Return($\alpha$).

### 4.6.1 Selecting a prime $p$ and generator of $\mathbb{Z}_p^*$

In cryptographic applications for which a generator of $\mathbb{Z}_p^*$ is required, one usually has the flexibility of selecting the prime $p$. To guard against the Pohlig-Hellman algorithm for computing discrete logarithms (Algorithm 3.63), a security requirement is that $p-1$ should contain a "large" prime factor $q$. In this context, "large" means that the quantity $\sqrt{q}$ represents an infeasible amount of computation; for example, $q \geq 2^{160}$. This suggests the following algorithm for selecting appropriate parameters $(p, \alpha)$.

**4.84 Algorithm** Selecting a $k$-bit prime $p$ and a generator $\alpha$ of $\mathbb{Z}_p^*$

INPUT: the required bitlength $k$ of the prime and a security parameter $t$.
OUTPUT: a $k$-bit prime $p$ such that $p-1$ has a prime factor $\geq t$, and a generator $\alpha$ of $\mathbb{Z}_p^*$.

1. Repeat the following:
   - 1.1 Select a random $k$-bit prime $p$ (for example, using Algorithm 4.44).
   - 1.2 Factor $p-1$.

   Until $p-1$ has a prime factor $\geq t$.
2. Use Algorithm 4.80 with $G = \mathbb{Z}_p^*$ and $n = p-1$ to find a generator $\alpha$ of $\mathbb{Z}_p^*$.
3. Return($p$,$\alpha$).

Algorithm 4.84 is relatively inefficient as it requires the use of an integer factorization algorithm in step 1.2. An alternative approach is to generate the prime $p$ by first choosing a large prime $q$ and then selecting relatively small integers $R$ at random until $p = 2Rq + 1$ is prime. Since $p - 1 = 2Rq$, the factorization of $p-1$ can be obtained by factoring $R$. A particularly convenient situation occurs by imposing the condition $R = 1$. In this case the factorization of $p-1$ is simply $2q$. Furthermore, since $\phi(p-1) = \phi(2q) = \phi(2)\phi(q) = q-1$, the probability that a randomly selected element $\alpha \in \mathbb{Z}_p^*$ is a generator is $\frac{q-1}{2q} \approx \frac{1}{2}$.

**4.85 Definition** A *safe prime* $p$ is a prime of the form $p = 2q + 1$ where $q$ is prime.

Algorithm 4.86 generates a safe (probable) prime $p$ and a generator of $\mathbb{Z}_p^*$.

**4.86 Algorithm** Selecting a $k$-bit safe prime $p$ and a generator $\alpha$ of $\mathbb{Z}_p^*$

INPUT: the required bitlength $k$ of the prime.
OUTPUT: a $k$-bit safe prime $p$ and a generator $\alpha$ of $\mathbb{Z}_p^*$.

1. Do the following:
   - 1.1 Select a random $(k-1)$-bit prime $q$ (for example, using Algorithm 4.44).
   - 1.2 Compute $p \leftarrow 2q + 1$, and test whether $p$ is prime (for example, using trial division by small primes and Algorithm 4.24).

   Until $p$ is prime.
2. Use Algorithm 4.80 to find a generator $\alpha$ of $\mathbb{Z}_p^*$.
3. Return($p$,$\alpha$).

## 4.7 Notes and further references

§4.1

Several books provide extensive treatments of primality testing including those by Bressoud [198], Bach and Shallit [70], and Koblitz [697]. The book by Kranakis [710] offers a more theoretical approach. Cohen [263] gives a comprehensive treatment of modern primality tests. See also the survey articles by A. Lenstra [747] and A. Lenstra and H. Lenstra [748]. Facts 4.1 and 4.2 were proven in 1837 by Dirichlet. For proofs of these results, see Chapter 16 of Ireland and Rosen [572]. Fact 4.3 is due to Rosser and Schoenfeld [1070]. Bach and Shallit [70] have further results on the distribution of prime numbers.

§4.2

Fact 4.13(i) was proven by Alford, Granville, and Pomerance [24]; see also Granville [521]. Fact 4.13(ii) is due to Pomerance, Selfridge, and Wagstaff [996]. Pinch [974] showed that there are 105212 Carmichael numbers up to $10^{15}$.

The Solovay-Strassen probabilistic primality test (Algorithm 4.18) is due to Solovay and Strassen [1163], as modified by Atkin and Larson [57].

Fact 4.23 was proven independently by Monier [892] and Rabin [1024]. The Miller-Rabin test (Algorithm 4.24) originated in the work of Miller [876] who presented it as a non-probabilistic polynomial-time algorithm assuming the correctness of the Extended Riemann Hypothesis (ERH). Rabin [1021, 1024] rephrased Miller's algorithm as a probabilistic primality test. Rabin's algorithm required a small number of gcd computations. The Miller-Rabin test (Algorithm 4.24) is a simplification of Rabin's algorithm which does not require any gcd computations, and is due to Knuth [692, p.379]. Arazi [55], making use of Montgomery modular multiplication (§14.3.2), showed how the Miller-Rabin test can be implemented by "divisionless modular exponentiations" only, yielding a probabilistic primality test which does not use any division operations.

Miller [876], appealing to the work of Ankeny [32], proved under assumption of the Extended Riemann Hypothesis that, if $n$ is an odd composite integer, then its least strong witness is less than $c(\ln n)^2$, where $c$ is some constant. Bach [63] proved that this constant may be taken to be $c = 2$; see also Bach [64]. As a consequence, one can test $n$ for primality in $O((\lg n)^5)$ bit operations by executing the Miller-Rabin algorithm for all bases $a \leq 2(\ln n)^2$. This gives a deterministic polynomial-time algorithm for primality testing, under the assumption that the ERH is true.

Table 4.1 is from Jaeschke [630], building on earlier work of Pomerance, Selfridge, and Wagstaff [996]. Arnault [56] found the following 46-digit composite integer

$$n = 1195068768795265792518361315725116351898245581$$

that is a strong pseudoprime to all the 11 prime bases up to 31. Arnault also found a 337-digit composite integer which is a strong pseudoprime to all 46 prime bases up to 199.

The Miller-Rabin test (Algorithm 4.24) randomly generates $t$ independent bases $a$ and tests to see if each is a strong witness for $n$. Let $n$ be an odd composite integer and let $t = \lceil \frac{1}{2} \lg n \rceil$. In situations where random bits are scarce, one may choose instead to generate a single random base $a$ and use the bases $a, a+1, \ldots, a+t-1$. Bach [66] proved that for a randomly chosen integer $a$, the probability that $a, a+1, \ldots, a+t-1$ are all strong liars for $n$ is bounded above by $n^{-1/4+o(1)}$; in other words, the probability that the Miller-Rabin algorithm using these bases mistakenly declares an odd composite integer "prime" is at most $n^{-1/4+o(1)}$. Peralta and Shoup [969] later improved this bound to $n^{-1/2+o(1)}$.

Monier [892] gave exact formulas for the number of Fermat liars, Euler liars, and strong liars for composite integers. One consequence of Monier's formulas is the following improvement (in the case where $n$ is not a prime power) of Fact 4.17 (see Kranakis [710, p.68]). If $n \geq 3$ is an odd composite integer having $r$ distinct prime factors, and if $n \equiv 3 \pmod 4$, then there are at most $\phi(n)/2^{r-1}$ Euler liars for $n$. Another consequence is the following improvement (in the case where $n$ has at least three distinct prime factors) of Fact 4.23. If $n \geq 3$ is an odd composite integer having $r$ distinct prime factors, then there are at most $\phi(n)/2^{r-1}$ strong liars for $n$. Erdös and Pomerance [373] estimated the average number of Fermat liars, Euler liars, and strong liars for composite integers. Fact 4.30(ii) was proven independently by Atkin and Larson [57], Monier [892], and Pomerance, Selfridge, and Wagstaff [996].

Pinch [975] reviewed the probabilistic primality tests used in the *Mathematica*, *Maple V*, *Axiom*, and *Pari/GP* computer algebra systems. Some of these systems use a probabilistic primality test known as the *Lucas test*; a description of this test is provided by Pomerance, Selfridge, and Wagstaff [996].

§4.3

If a number $n$ is composite, providing a non-trivial divisor of $n$ is evidence of its compositeness that can be verified in polynomial time (by long division). In other words, the decision problem "is $n$ composite?" belongs to the complexity class **NP** (cf. Example 2.65). Pratt [1000] used Fact 4.38 to show that this decision problem is also in **co-NP**. That is, if $n$ is prime there exists some evidence of this (called a *certificate of primality*) that can be verified in polynomial time. Note that the issue here is not in finding such evidence, but rather in determining whether such evidence exists which, if found, allows efficient verification. Pomerance [992] improved Pratt's results and showed that every prime $n$ has a certificate of primality which requires $O(\ln n)$ multiplications modulo $n$ for its verification.

Primality of the *Fermat number* $F_k = 2^{2^k} + 1$ can be determined in deterministic polynomial time by *Pepin's test*: for $k \geq 2$, $F_k$ is prime if and only if $5^{(F_k-1)/2} \equiv -1 \pmod{F_k}$. For the history behind Pepin's test and the Lucas-Lehmer test (Algorithm 4.37), see Bach and Shallit [70].

In Fact 4.38, the integer $a$ does not have to be the same for all $q$. More precisely, Brillhart and Selfridge [212] showed that Fact 4.38 can be refined as follows: an integer $n \geq 3$ is prime if and only if for each prime divisor $q$ of $n-1$, there exists an integer $a_q$ such that $a_q^{n-1} \equiv 1 \pmod n$ and $a_q^{(n-1)/q} \not\equiv 1 \pmod n$. The same is true of Fact 4.40, which is due to Pocklington [981]. For a proof of Fact 4.41, see Maurer [818]. Fact 4.42 is due to Brillhart, Lehmer, and Selfridge [210]; a simplified proof is given by Maurer [818].

The original Jacobi sum test was discovered by Adleman, Pomerance, and Rumely [16]. The algorithm was simplified, both theoretically and algorithmically, by Cohen and H. Lenstra [265]. Cohen and A. Lenstra [264] give an implementation report of the Cohen-Lenstra Jacobi sum test; see also Chapter 9 of Cohen [263]. Further improvements of the Jacobi sum test are reported by Bosma and van der Hulst [174].

Elliptic curves were first used for primality proving by Goldwasser and Kilian [477], who presented a randomized algorithm which has an expected running time of $O((\ln n)^{11})$ bit operations for *most* inputs $n$. Subsequently, Adleman and Huang [13] designed a primality proving algorithm using hyperelliptic curves of genus two whose expected running time is polynomial for *all* inputs $n$. This established that the decision problem "is $n$ prime?" is in the complexity class **RP** (Definition 2.77(ii)). The Goldwasser-Kilian and Adleman-Huang algorithms are inefficient in practice. Atkin's test, and an implementation of it, is extensively described by Atkin and Morain [58]; see also Chapter 9 of Cohen [263]. The largest number proven prime as of 1996 by a general purpose primality proving algorithm is a 1505-decimal digit number, accomplished by Morain [903] using Atkin's test. The total time for the computation was estimated to be 4 years of CPU time distributed among 21 SUN 3/60 workstations. See also Morain [902] for an implementation report on Atkin's test which was used to prove the primality of the 1065-decimal digit number $(2^{3539}+1)/3$.

§4.4

A proof of Mertens's theorem can be found in Hardy and Wright [540]. The optimal trial division bound (Note 4.45) was derived by Maurer [818]. The discussion (Note 4.47) on the probability $P(X|Y_t)$ is from Beauchemin et al. [81]; the result mentioned in the last sentence of this note is due to Kim and Pomerance [673]. Fact 4.48 was derived by Damgård, Landrock, and Pomerance [300], building on earlier work of Erdös and Pomerance [373], Kim and Pomerance [673], and Damgård and Landrock [299]. Table 4.3 is Table 2 of Damgård, Landrock, and Pomerance [300]. The suggestions to first do a Miller-Rabin test with base $a = 2$ (Remark 4.50) and to do an incremental search (Note 4.51) in Algorithm 4.44 were made by Brandt, Damgård, and Landrock [187]. The error and failure probabilities for incremental search (Note 4.51(i)) were obtained by Brandt and Damgård [186]; consult this paper for more concrete estimates of these probabilities.

Algorithm 4.53 for generating strong primes is due to Gordon [514, 513]. Gordon originally proposed computing $p_0 = (s^{r-1} - r^{s-1}) \bmod rs$ in step 3. Kaliski (personal communication, April 1996) proposed the modified formula $p_0 = (2s^{r-2} \bmod r)s - 1$ which can be computed more efficiently. Williams and Schmid [1249] proposed an algorithm for generating strong primes $p$ with the additional constraint that $p - 1 = 2q$ where $q$ is prime; this algorithm is not as efficient as Gordon's algorithm. Hellman and Bach [550] recommended an additional constraint on strong primes, specifying that $s - 1$ (where $s$ is a large prime factor of $p+1$) must have a large prime factor (see §15.2.3(v)); this thwarts cycling attacks based on Lucas sequences.

The NIST method for prime generation (Algorithm 4.56) is that recommended by the NIST Federal Information Processing Standards Publication (FIPS) 186 [406].

Fact 4.59 and Algorithm 4.62 for provable prime generation are derived from Maurer [818]. Algorithm 4.62 is based on that of Shawe-Taylor [1123]. Maurer notes that the total diversity of reachable primes using the original version of his algorithm is roughly 10% of all primes. Maurer also presents a more complicated algorithm for generating provable primes with a better diversity than Algorithm 4.62, and provides extensive implementation details and analysis of the expected running time. Maurer [812] provides heuristic justification that Algorithm 4.62 generates primes with virtually uniform distribution. Mihailescu [870] observed that Maurer's algorithm can be improved by using the Eratosthenes sieve method for trial division (in step 8.2 of Algorithm 4.62) and by searching for a prime $n$ in an appropriate interval of the arithmetic progression $2q+1, 4q+1, 6q+1, \ldots$ instead of generating $R$'s at random until $n = 2Rq+1$ is prime. The second improvement comes at the expense of a reduction of the set of primes which may be produced by the algorithm. Mihailescu's paper includes extensive analysis and an implementation report.

§4.5

Lidl and Niederreiter [764] provide a comprehensive treatment of irreducible polynomials; proofs of Facts 4.67 and 4.68 can be found there.

Algorithm 4.69 for testing a polynomial for irreducibility is due to Ben-Or [109]. The fastest algorithm known for generating irreducible polynomials is due to Shoup [1131] and has an expected running time of $O(m^3 \lg m + m^2 \lg p)$ $\mathbb{Z}_p$-operations. There is no *deterministic* polynomial-time algorithm known for finding an irreducible polynomial of a specified degree $m$ in $\mathbb{Z}_p[x]$. Adleman and Lenstra [14] give a deterministic algorithm that runs in polynomial time under the assumption that the ERH is true. The best deterministic algorithm known is due to Shoup [1129] and takes $O(m^4\sqrt{p})$ $\mathbb{Z}_p$-operations, ignoring powers of $\log m$ and $\log p$. Gordon [512] presents an improved method for computing minimum polynomials of elements in $\mathbb{F}_{2^m}$.

Zierler and Brillhart [1271] provide a table of all irreducible trinomials of degree $\leq 1000$ in $\mathbb{Z}_2[x]$. Blake, Gao, and Lambert [146] extended this list to all irreducible trinomials of degree $\leq 2000$ in $\mathbb{Z}_2[x]$. Fact 4.75 is from their paper.

Table 4.8 extends a similar table by Stahnke [1168]. The primitive pentanomials $x^m + x^{k_1} + x^{k_2} + x^{k_3} + 1$ listed in Table 4.8 have the following properties: (i) $k_1 = k_2 + k_3$; (ii) $k_2 > k_3$; and (iii) $k_3$ is as small as possible, and for this particular value of $k_3$, $k_2$ is as small as possible. The rational behind this form is explained in Stahnke's paper. For each $m < 5000$ for which the factorization of $2^m - 1$ is known, Živković [1275, 1276] gives a primitive trinomial in $\mathbb{Z}_2[x]$, one primitive polynomial in $\mathbb{Z}_2[x]$ having five non-zero terms, and one primitive polynomial in $\mathbb{Z}_2[x]$ having seven non-zero terms, provided that such polynomials exist. The factorizations of $2^m - 1$ are known for all $m \leq 510$ and for some additional $m \leq 5000$. A list of such factorizations can be found in Brillhart et al. [211] and updates of the list are available by anonymous ftp from `sable.ox.ac.uk` in the `/pub/math/cunningham/` directory. Hansen and Mullen [538] describe some improvements to Algorithm 4.78 for generating primitive polynomials. They also give tables of primitive polynomials of degree $m$ in $\mathbb{Z}_p[x]$ for each prime power $p^m \leq 10^{50}$ with $p \leq 97$. Moreover, for each such $p$ and $m$, the primitive polynomial of degree $m$ over $\mathbb{Z}_p$ listed has the smallest number of non-zero coefficients among all such polynomials.

The entries of Table 4.9 were obtained from Zierler [1270] for Mersenne exponents $M_j$, $1 \leq j \leq 23$, and from Kurita and Matsumoto [719] for Mersenne exponents $M_j$, $24 \leq j \leq 27$.

Let $f(x) \in \mathbb{Z}_p[x]$ be an irreducible polynomial of degree $m$, and consider the finite field $\mathbb{F}_{p^m} = \mathbb{Z}_p[x]/(f(x))$. Then $f(x)$ is called a *normal polynomial* if the set $\{x, x^p, x^{p^2}, \ldots, x^{p^{m-1}}\}$ forms a basis for $\mathbb{F}_{p^m}$ over $\mathbb{Z}_p$; such a basis is called a *normal basis*. Mullin et al. [911] introduced the concept of an *optimal normal basis* in order to reduce the hardware complexity of multiplying field elements in the finite field $\mathbb{F}_{2^m}$. A VLSI implementation of the arithmetic in $\mathbb{F}_{2^m}$ which uses optimal normal bases is described by Agnew et al. [18]. A normal polynomial which is also primitive is called a *primitive normal polynomial*. Davenport [301] proved that for any prime $p$ and positive integer $m$ there exists a primitive normal polynomial of degree $m$ in $\mathbb{Z}_p[x]$. See also Lenstra and Schoof [760] who generalized this result from prime fields $\mathbb{Z}_p$ to prime power fields $\mathbb{F}_q$. Morgan and Mullen [905] give a primitive normal polynomial of degree $m$ over $\mathbb{Z}_p$ for each prime power $p^m \leq 10^{50}$ with $p \leq 97$. Moreover, each polynomial has the smallest number of non-zero coefficients among all primitive normal polynomials of degree $m$ over $\mathbb{Z}_p$; in fact, each polynomial has at most five non-zero terms.

§4.6

No polynomial-time algorithm is known for finding generators, or even for testing whether an element is a generator, of a finite field $\mathbb{F}_q$ if the factorization of $q-1$ is unknown. Shoup [1130] considered the problem of deterministically generating in polynomial time a subset of $\mathbb{F}_q$ that contains a generator, and presented a solution to the problem for the case where the characteristic $p$ of $\mathbb{F}_q$ is small (e.g. $p = 2$). Maurer [818] discusses how his algorithm (Algorithm 4.62) can be used to generate the parameters $(p, \alpha)$, where $p$ is a *provable* prime and $\alpha$ is a generator of $\mathbb{Z}_p^*$.