INTERNATIONAL STANDARD

ISO/IEC 18032

Second edition
2020-12

Information security — Prime number generation

Sécurité de l'information — Génération de nombres premiers

Reference number
ISO/IEC 18032:2020(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Information security, cybersecurity and privacy protection*.

This second edition cancels and replaces the first edition (ISO/IEC 18032:2005), which has been technically revised.

The main changes compared to the previous edition are as follows:

— the Frobenius-Grantham primality test in 6.2, the Lehmann primality test in 6.3 and Maurer's algorithm in 8.3.1, have been removed;

– the Elliptic curve primality proving algorithm, The Shawe-Taylor algorithm and the algorithm to generate primes with side conditions, have been added or substantially revised.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Information security — Prime number generation

1 Scope

This document specifies methods for generating and testing prime numbers as required in cryptographic protocols and algorithms.

Firstly, this document specifies methods for testing whether a given number is prime. The testing methods included in this document are divided into two groups:

— probabilistic primality tests, which have a small error probability. All probabilistic tests described here can declare a composite to be a prime;

— deterministic methods, which are guaranteed to give the right verdict. These methods use so-called primality certificates.

Secondly, this document specifies methods to generate prime numbers. Again, both probabilistic and deterministic methods are presented.

NOTE It is possible that readers with a background in algorithm theory have already had previous encounters with probabilistic and deterministic algorithms. The deterministic methods in this document internally still make use of random bits (to be generated via methods described in ISO/IEC 18031), and “deterministic” only refers to the fact that the output is correct with probability one.

Annex A provides error probabilities that are utilized by the Miller-Rabin primality test.

Annex B describes variants of the methods for generating primes so that particular cryptographic requirements can be met.

Annex C defines primitives utilized by the prime generation and verification methods.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 18031, *Information technology — Security techniques — Random bit generation*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
composite number
composite
integer for which divisors exist that are not *trivial divisors* (3.8)

3.2
deterministic random bit generator
DRBG
random bit generator that produces a random-appearing sequence of bits by applying a deterministic algorithm to a suitably random initial value called a seed and, possibly, some secondary inputs on which the security of the random bit generator does not depend

3.3
entropy
measure of the disorder, randomness or variability in a closed system

[SOURCE: ISO/IEC 18031:2011, 3.11]

3.4
Jacobi symbol
Jacobi symbol of a positive integer *a* with respect to an odd integer *n*
product of the Legendre symbols of a with respect to the prime factors of n, including *multiplicity* (3.5)

Note 1 to entry: If the prime factor p occurs with multiplicity $m \geq 1$ in the factorization of n, then the Legendre symbol of a with respect to p occurs with multiplicity m in the product that yields the Jacobi symbol of a with respect to n.

Note 2 to entry: The Legendre symbol of a positive integer a with respect to a prime number p is the value

$a^{(p-1)/2} \bmod p$

3.5
multiplicity
multiplicity of a prime divisor *p* of *n*
largest positive integer e with p^e dividing n

3.6
primality certificate
mathematical proof that a given integer is indeed a prime

Note 1 to entry: For a small integer, primality is most efficiently proven by trial division. In this case, the primality certificate can therefore be empty.

3.7
prime number
prime
positive integer for which there exist only *trivial divisors* (3.8)

3.8
trivial divisors
trivial divisors of a nonzero integer *N*
1, -1, N and $-N$

Note 1 to entry: Any nonzero integer N is divisible by (at least) 1, -1, N and $-N$.

4 Symbols and abbreviated terms

a div n	for integers a and n, with $n \neq 0$, a div n is the unique integer s satisfying $a = s \cdot n + r$ where $0 \leq r < n$.
a mod n	for integers a and n, with $n \neq 0$, a mod n is the unique non-negative integer r satisfying $a = (a \text{ div } n) \cdot n + r$.
C	primality certificate

$C(N)$	primality certificate for the number N
$C_0(N)$	empty primality certificate, indicating that trial division should be used to verify that N is a prime
$\exp(c)$	natural exponential function evaluated at c, i.e. e^c where $e \approx 2{,}718\ 28$
gcd	greatest common divisor
Jacobi(a,N)	Jacobi symbol for an integer a with respect to a nonzero odd integer N
k	number of bits in N
L	limit below which primality is verified by trial division
Lucas(D, K, N)	K^{th} element of the Lucas sequence modulo N with discriminant D
$\ln(a)$	natural logarithm of a with respect to the base $e \approx 2{,}718\ 28$
$\log_b(a)$	logarithm of a with respect to base b
$\min\{a,b\}$	minimum of the numbers a and b
N	candidate number to be tested for primality, where N is always a positive, odd number
sgn(D)	sign of a number, i.e. sgn(D) = 1 if $D \geq 0$ and -1 otherwise.
T	(probabilistic) test for primality
Z_N	the set of the integers 0, 1, 2, ..., N-1, representing the ring of integers modulo N
$Z_N{}^*$	subset of Z_N containing the numbers that have a multiplicative inverse modulo N (e.g., if N is prime, $Z_N{}^*$ consists of the integers 1, 2, ..., N-1)
β	parameter that determines the lower bound of the entropy of the output of a prime generation algorithm
μ	maximal number of steps in an incremental search for a prime
$\lfloor x \rfloor$	largest integer smaller than or equal to x
$\lceil x \rceil$	smallest integer greater than or equal to x
$\sqrt{n}$	principal (i.e. non-negative) square root of a non-negative number n

5 Trial division

The primality of an integer N can be proven by means of trial division. This shall be done in the following way:

a) For all primes $p \leq \sqrt{N}$:

   1) if N mod p = 0 then return “N composite” and stop;

b) return “N prime” and stop.

For small integers N, trial division is less computationally expensive than other primality tests. Implementations of any primality test described in this document may define a trial division bound

L, below which trial division is used in order to prove the primality of integers. This document sets no value for L except for a lower bound, i.e. $L > 6$.

NOTE 1 It is assumed that the set of prime numbers below a certain size are already known. One practical way to implement the test is to have a pre-computed table of the first few primes, do trial division by these, and then simply trial divide by all odd integers up to the square root.

NOTE 2 The size of integers for which trial division is less computationally expensive than another primality test depends on the test and its implementation. A possible value for L can be $L = 10^{10}$.

NOTE 3 A binned gcd method, as described in ANSI X9.80-2010,[1] can be more efficient than trial division for certain implementations.

6 Probabilistic primality test

6.1 General

A probabilistic primality test takes a positive, odd integer N as input and returns "N accepted" or "N composite". The Miller-Rabin primality test described in 6.3 will always output "N accepted" when N is a prime number. However, if N is a composite number, then an instance of the test can erroneously return "N accepted". In order to reduce the probability of such errors, one usually performs several iterations of testing on N, using different choices for the random values employed.

The probabilistic tests in this clause shall only be applied to odd integers that are greater or equal to the trial division bound L. If $N < L$, trial division shall be applied to determine the primality of N.

6.2 Requirements

In order for a number to be accepted as a (probable) prime, this document requires the error probability, i.e. the probability the number is composite, to be at most 2^{-100}. This provides significant confidence that any candidate prime being tested for primality that meets this threshold is indeed prime. The 2^{-100} probability bound is achieved by requiring a sufficient number of Miller-Rabin tests, depending on how the number was generated (see Annex A).

6.3 Miller-Rabin primality test

The Miller-Rabin primality test is based on the following observation. Suppose that N actually is an odd prime number and that r and s are the unique positive integers such that $N - 1 = 2^r s$, with s odd. For each positive integer $b < N$, exactly one of the following three conditions will be satisfied:

— $b^s \bmod N = 1$;

— $b^s \bmod N = N - 1$; or

— $\left(b^s\right)^{2^i} \bmod N = N - 1$, for some i with $0 < i < r$.

Equivalently, if $N \geq 3$ is an odd integer and there exists a positive integer $b < N$ that does not satisfy any of the conditions above, then N is a composite number. A probabilistic primality test based on this observation shall be applied to any odd integer $N \geq L$, as follows.

Initialization

a) Determine positive integers r and s such that $N - 1 = 2^r s$, where s is odd.

b) Set rounds = 0.

Perform t iterations (rounds) of Miller-Rabin testing (for integer $t \geq 1$).

c) Choose a random integer b such that $2 \leq b \leq N - 2$.

```
d)  Set $y = b^s \bmod N$.
e)  If $y = 1$ or $y = N - 1$,
    1)  set rounds = rounds + 1;
    2)  if rounds $< t$;
            go to step c)
        else
            return “$N$ probably prime” and stop testing.
f)  For $i = 1$ to $r - 1$, do:
    1)  set $y = y^2 \bmod N$;
    2)  if $y = N - 1$;
        i)   set rounds = rounds + 1;
        ii)  if rounds $< t$, go to step c);
               otherwise return “$N$ probably prime” and stop testing.
g)  Return “$N$ composite” and stop testing.
```

The candidate N is accepted as being (probably) prime at the conclusion of the iterated testing process if and only if N passes all t rounds of Miller-Rabin primality testing (with each choice of b satisfying one of the conditions associated with primality). The testing process returns “N composite” and is immediately halted if, for some choice of b, none of the tested conditions are satisfied (see A.2 and A.3 to determine the number of iterations required by this document).

The integer b generated in step c) shall be generated using a random bit generator that meets the specifications of ISO/IEC 18031 and converted to a number using the conversion methods in Annex C or ISO/IEC 18031. For each iteration, the process of selecting a value for b in step c) is performed anew.

NOTE 1 The rationale for the base b being randomly generated is two-fold. Firstly, the average case error estimates adopted from Reference [9] and used in A.3 assume b is random. Secondly, for a known base b, it is not difficult to construct composite integers that will pass a round of Miller-Rabin with respect to that base.

NOTE 2 Step f) is only applicable when $r > 1$, i.e. if $N \bmod 4 = 1$.

7 Deterministic primality verification methods

7.1 General

Deterministic primality verification methods use primality certificates in order to verify the primality of a given number. This clause specifies the content of two types of primality certificates:

— primality certificates based on elliptic curves;

— primality certificates for primes generated by The Shawe-Taylor algorithm (see 8.4.2).

A primality certificate contains information that enables efficient verification that a given number is a prime. For both types of certificates described in this document, small numbers (i.e. numbers smaller than the trial division bound L) shall be verified to be primes by trial division. Let C_0 denote the empty primality certificate for such numbers.

An elliptic curve primality certificate can be computed given any prime. Hence, the methods for computing this certificate may be used to verify primality. The primality certificate obtained in The

Shawe-Taylor algorithm is generated as part of the process of generating a prime, and cannot be efficiently computed for an arbitrary prime (after the prime has been generated).

7.2 Elliptic curve primality proving algorithm

7.2.1 General

Information regarding elliptic curves can be obtained from ISO/IEC 15946-1.

7.2.2 Elliptic curve primality certificate generation

To generate a certificate of primality for an odd integer $N \geq L$, the method described below is used recursively. If the method succeeds, the total collection of data generated by the method is organized in a primality certificate, $C(N)$. Verifying $C(N)$, and thereby proving that N is prime, is considerably faster than generating the certificate.

On input of an integer $N \geq L$, with gcd(N,6) = 1, the elliptic curve primality proving algorithm shall start with the following initializations.

a) Set $S = \{ N \}$ (the set of integers that require a primality certificate), and set $Certs = \{ \}$ (a contentless certificate).

   In the following steps, various primality tests are applied to an integer r selected from set S. If a test (provisionally) accepts the primality of r, it returns a certificate $C(r)$, and, possibly, a set of probable primes $\{ q_i \}$, in which case the certificate shall not be used to prove that r is prime, unless each q_i has been proven prime. If necessary, the algorithm shall proceed recursively, attempting to generate a certificate of primality for each of those probable primes. In the course of generating those certificates, additional probable primes may be added to the set of integers that require certificates. The algorithm shall continue until either there are no more probable primes to process or the processing is aborted because some integer requiring a certificate is determined to be a composite number.

b) Select a value for r from S and set $S = S - \{ r \}$ (i.e. remove the element r from S).

   1) Apply either:

      — Pocklington's primality test to r (see D.2) and, if that test is inconclusive, also apply the Deterministic Lucas primality test to r (see D.4.2); or

      — the Brillhart-Lehmer-Selfridge test to r (see D.5).

      In either case, when the testing is performed, allow probable primes $\geq L$ to occur (with multiplicity) in the partial factorizations of $r - 1$ and/or $r + 1$.

   2) If the testing indicates r is composite, return "r composite" (along with the value of r) and stop.

   3) If the testing is inconclusive, proceed to step c).

   4) If the testing indicates that r is prime, then adjoin $C(r)$ to $Certs$, where $C(r)$ is the certificate for r returned by the successful test, and set $S = S \cup \{ q_i \}$, where $\{ q_i \}$ is the set of probable primes (if any) appearing in the factorizations of $r - 1$ and/or $r + 1$ (whichever were used in the testing); if S is not empty, repeat step b). Otherwise, return "N prime" along with $C(N) = Certs$, and stop.

c) Generate an elliptic curve E given by $y^2 = x^3 + ax + b$ for some integers a, b.

   1) Apply the elliptic curve primality test to r (see D.6). When the test is performed, allow probable primes $\geq L$ to occur (with multiplicity) in the partial factorization of the order of E_r (the curve E reduced modulo r).

   2) If the test indicates that r is composite, return "r composite" along with the value of r and stop.

3) If the test is inconclusive, repeat step c) with a new choice of elliptic curve.

4) If the test indicates that r is prime, then adjoin $C(r)$ to *Certs*, where $C(r)$ is the certificate for r returned by the successful test, and set $S = S \cup \{ q_i \}$, where $\{ q_i \}$ is the set of probable primes (if any) appearing in the factorization of the order of E_r; if S is not empty, go to step b). Otherwise, return "N is prime" along with $C(N) = Certs$, and stop.

The probable prime factors (if any) appearing in steps b) and c) are required to be greater than or equal to the trial division bound. The primality of smaller factors shall be ascertained using trial division.

During the recursion, the tests in step b) can be skipped. It is included in the algorithm's description for efficiency purposes.

If the primality of N is inconclusive, the test may be executed again. The test may not be rerun in full if partial results of the previous execution are retained (e.g., the value of *Certs* when testing process was stopped and the composite r value that caused the early termination). It can be sufficient to back the recursion step one level prior to the point of failure. Varying choices of curves in step c) and/or optionally skipping step b) is likely to result in different sets of probable primes to test.

In case r is composite, the elliptic curve test in step c) is likely to be inconclusive and so the algorithm can fail to terminate. As a precaution, a limit on the number of iterations of step c) may be enforced.

NOTE The elliptic curve E in step c) can be generated using the CM method described in D.8.

7.2.3 Elliptic curve primality certificate verification

An elliptic curve primality certificate for an integer $N \geq L$, with $\gcd(N, 6) = 1$, is a set of (interdependent) certificates, $C(N) = \{ C_i \}$, where each C_i asserts the primality of some integer $r_i \geq L$, and exactly one of the r_i is equal to N. Each C_i is a certificate resulting from the execution of either Pocklington's test (D.2.2), the Deterministic Lucas test (D.4.2), the Brillhart-Selfridge-Lehmer test (D.5) or the elliptic curve test (D.6).

If C_i asserts the primality of r_i based (in part) on the assumed primality of one or more other integers $q_j \geq L$, then the certificate C_i (and hence the primality of r_i) shall only be successfully verified after the primality of each of the q_j is accepted via the verification of their certificates, which shall also be included in $C(N)$. If C_i asserts the primality of r_i based (in part) on the assumed primality of any integers less than L, then trial division shall be used to verify the primality of those integers as part of the verification of C_i.

The elliptic curve primality certificate $C(N) = \{ C_i \}$ is accepted only if every C_i is accepted (and exactly one of the r_i is equal to N). If the verification of any C_i fails, then the elliptic curve primality certificate for N shall be rejected.

Verifying each of the individual certificates C_i shall be done as specified in Annex D.

7.3 Primality certificate based on The Shawe-Taylor algorithm

This type of primality certificate C is a collection of certificates $\{ C_i \}$ which shall be computed (only) during the generation process of the prime number using the process described in 8.4.2 (The Shawe-Taylor algorithm). The certificates C_i are the result of the Pocklington test (see D.2.2) whose proofs of primality have the following structure:

$$\mathrm{Proof}(r_i) = (r_i, q_i, a_i)$$

where r_i, q_i, and a_i are integers. The certificate C_i, which asserts that r_i is prime, is verified once it passes the checks in C.2 and q_i is proven to be (an odd) prime (e.g., by trial division or by verifying its own certificate, which has also been included in C). The last certificate to be verified, say $C_1 = (N, q_1, a_1)$, contains the value N. If all of the C_i are verified (and are confirmed to chain up to N), then C itself is verified and N is accepted as being prime.

8 Prime number generation

8.1 General

This clause specifies two approaches to prime number generation. The first approach is to employ the probabilistic Miller-Rabin primality test on randomly chosen candidates (8.3). [Optionally, a probable prime, N, generated by such methods may subsequently be proven prime by generating an elliptic curve primality certificate (7.2) for N.] The second approach to prime number generation is to employ the deterministic Shawe-Taylor method (8.4) which yield integers known to be prime. This method can also produce primality certificates as part of the generation process.

The methods in this clause generate primes in the interval $(2^{k-1}, 2^k)$ for some k. To generate primes with additional constraints, such as generating primes in a more restrictive interval and/or primes that satisfy certain congruence conditions, the methods in B.2 shall be used. Examples of primes generated with additional constraints are given in Annex E.

These techniques shall only be applied to generate primes greater or equal to the trial division bound L. Generating primes less than L can be simply done by selecting integers less than L and testing for primality using trial division.

8.2 Requirements

In order for a prime number generation algorithm to conform to this document, the algorithm shall:

— generate random bits using a deterministic random bit generator that meets the specifications of ISO/IEC 18031;

— generate random numbers from sequences of random bits using the conversion methods specified in Annex C or ISO/IEC 18031;

— ensure, in the case of non-provable primes (8.3), that the error probability that a composite passes as prime is at most 2^{-100}.

Annex A contains more information on how the algorithms in 8.3 shall satisfy the 2^{-100} error properties.

For applications where the primes generated need to be secret, the following also applies:

— the (secret) entropy used in the random bit generator to produce the output number shall satisfy the requirements of ISO/IEC 18031 and shall be at least B bits where B is the security level of that application;

— if additional constraints are placed on the prime being generated (B.2), the constraints shall be limited so as not to affect the security of the system and the requirements in Annex B shall apply.

In the case of generating primes for use in RSA, B should be at least 112 for 1 024-bit primes, 128 for 1 536-bit primes, 192 for 3 840-bit primes, and 256 for 7 680-bit primes. ISO/IEC 18031 requires a minimum entropy of 120 bits to avoid collisions. In the case of RSA, collisions can lead to primes repeating for different RSA moduli. Such primes can be recovered by computing a gcd among the RSA moduli.

NOTE Limiting the total number of constraints is necessary to avoid Coppersmith-style attacks,[7][14] e.g. these attacks can factor RSA moduli if roughly half the bits of a prime factor are known. If the constraints (such as the number of known bits) are limited, then these attacks do not apply. For example, requiring an integer $n < 2^k$ to lie in the interval $(2^{k-2} + 2^{k-1}, 2^k)$ with $n \bmod 4 = 3$, and $\gcd(n\text{-}1, e) = 1$ for some odd encryption exponent gives less than 5 bits of constraint. If the interval is replaced with $(2^{k-1}\sqrt{2}, 2^k)$, the total constraint is less than 4,5 bits. Generating a k–bit number n with $n \bmod q = 1$ for some randomly generated (and secret) integer q of size m bits places roughly two bits of constraint on n (since n can be expressed as $n = 1 + u\, q$ where u, q are both k-m, m-bit numbers respectively). Further requiring q to be prime adds a few additional bits of constraint, approximately $\log_2(m) - 0{,}528$ bits by the prime number theorem.

When regenerating (secret) primes from a fixed seed, care should be taken so as to avoid side-channel attacks.

8.3 Using the Miller-Rabin primality test

8.3.1 General

The error probability that a composite passes as being prime depends on the number of iterations used in the Miller-Rabin primality test. Annex A shall be followed to determine the requisite number of iterations.

Subclauses 8.3.2 and 8.3.3 describe two algorithms that meet the requirements of 8.2.

8.3.2 Random search

The following algorithm may be used to generate a k-bit prime.

a) Generate a random number N such that $2^{k-1} < N < 2^k$. If N is even, increment N by 1.

b) Apply the Miller-Rabin primality test. If it is accepted, return N and stop. Otherwise, go to step a).

8.3.3 Incremental search

The following algorithm may be used to generate a k-bit prime. It differs from the algorithm given in 8.3.2 by the way it selects candidates for primality testing. If a randomly chosen (odd) candidate N is not accepted by the Miller-Rabin primality test, several of the following consecutive odd integers (N+2, N+4, etc.) are tested before randomly choosing another value for N. As a result, this algorithm can have some practical advantages (in terms of the efficiency of an implementation) over 8.3.2. A parameter denoted μ limits the number of consecutive odd integers that are tested between random choices of N.

a) Generate a random number N such that $2^{k-1} < N < 2^k$. If N is even, increment N by 1.

b) Set $Max = \min\{ 2^k -1, N + 2\mu \}$ for some appropriately chosen value of μ, e.g. $\mu = 10 \ln(2^k)$.

c) If N is accepted by the Miller-Rabin primality test, return N and stop.

d) Set $N = N+2$.

e) If $N > Max$, go to step a). Otherwise, go to step c).

Rather than increment the candidate prime N in step d) by 2, one may apply the Miller-Rabin test to elements in the sequence N, N+2, N+4, ..., Max which survive the sieving procedure described in C.1. The sieve process eliminates candidate primes that are divisible by the small primes used in the sieve.

NOTE The suggested value of $10 \ln(2^k)$ for μ is based on the Prime Number theorem which gives an approximation of $2^k / \ln(2^k)$ for the number of primes less than 2^k. The extra factor of 10 is given to increase the likelihood that a prime is contained in the interval $[N, N+2\mu]$. In particular, Lemma 4 of Reference [5] conjectures such a prime exists with probability about $1 - \exp(-2 \cdot c)$, $c = 10$.

8.3.4 Primes with an elliptic curve primality certificate

A certified prime may be generated by first generating a prime using the methods described in 8.3.2 and 8.3.3, and then computing an elliptic curve primality certificate for that number, as described in 7.2.

8.4 Using deterministic methods

8.4.1 General

The deterministic method in this subclause constructs a provable prime of the desired size by recursively constructing smaller provable primes. The prime generation algorithm calls itself repeatedly to generate a third-size (or half-size) prime and this recursion only stops when the prime size is such that a random candidate prime can be proven prime by trial division, after which the recursion unwinds by constructing triple-size (or doubled-size) primes that are provably prime by construction.

8.4.2 The Shawe-Taylor algorithm

The Shawe-Taylor algorithm generates a random prime number N from scratch. It shall be implemented using the algorithm described. This algorithm takes as input an integer k, the number of bits in the required prime. It returns a number N and a certificate of primality if requested.

This algorithm is called recursively, such that for a given bit-length $j \geq \log_2(L)$, there is a:

a) j'-bit prime q where $j' = \lceil j/3 \rceil + 1$ (or $\lceil j/2 \rceil + 1$) and a primality certificate $C(q)$ (which includes the certificates of any smaller primes used to construct q) if requested.

from which a j-bit prime p is constructed as follows:

b) Select an integer x at random from the interval $(2^{j-1}, 2^j - 2q]$.

c) Set $p = x + ((1-x) \bmod 2q)$, $t = (p - 1) \operatorname{div} q$ (note q divides $p - 1$).

d) Apply the Pocklington's primality test to p, with p-1 = $F\,R$ where $F = q$, $R = t$.

```
If test returns "p prime";
    return p along with certificate C(p) (if a certificate is requested) and stop.
else if $p < 2^j - 2q$;
    set $p = p + 2q$, $t = t + 2$, and go to step d).
else go to step b).
```

When $j < \log_2(L)$, a j-bit prime q_0 shall be constructed using trial division.

The certificate C for the k-bit prime N is the collection of certificates generated in step d) and the trial division certificate $C_0(q_0)$.

The algorithm may be implemented by first setting $j_n = k$ (for some n), and recursively computing bit-lengths $j_{i-1} = \lceil j_i/3 \rceil + 1$ (or $\lceil j_i/2 \rceil + 1$) until $j_0 < \log_2(L)$, upon which the j_0-bit prime q_0 is constructed using trial division. The prime q_0 is then used to construct the j_1-bit prime q_1 via steps b) to d). The prime q_1 is then used to construct a j_2-bit prime q_2, and so forth until a k-bit prime $N = q_n$ is generated.

Rather than increment the candidate prime p in step c) by $2q$ [in step d)], for some J in $(\,(2^{j-1} - p)\,/\,2q, (2^j - p)\,/\,2q\,)$ the Pocklington test may be applied to elements in the sequence $p, p+2q, p+4q, \ldots, p+2Jq$ which survive the sieving procedure described in C.1. The sieve process eliminates candidate primes that are divisible by small primes used in the sieve.

NOTE Use of the alternative value $j' = \lceil j/2 \rceil + 1$ in step a) is equivalent to the Shawe-Taylor algorithm in ISO/IEC 18032:2005 (and adapted from Reference [18]). The choice $j' = \lceil j/3 \rceil + 1$ makes use of the more general version of Pocklington's theorem given in D.2, and results in a more efficient algorithm due to fewer recursive steps.

Annex A
(normative)

Error probabilities

A.1 General

For any probabilistic test, the number of iterations that need to be performed depends on the error probability of one iteration of the test. The error probability of one iteration varies depending on whether the number to be tested has been selected at random, or whether it was chosen to have special properties.

For the Miller-Rabin test, worst-case error estimates (A.2) are given for the probability that an iterated application of the test accepts a given composite input number. The worst-case error estimates shall be applied unless stated otherwise within this document.

For large and randomly chosen integers, good average-case error estimates can be given, which significantly reduce the number of iterations of Miller-Rabin to be performed. The error estimates in A.3 apply only to the random search method of 8.3.2. In the case of the incremental search method of 8.3.3, different error probabilities apply.[5] To ensure the incremental search method achieves the requisite 2^{-100} error probability, the number of iterations of Miller-Rabin to be performed shall be increased by 1. Thus, for example, a 1 024-bit candidate prime requires at least 5 iterations. Further, if the number of iterations of Miller-Rabin is reduced for either search method, the candidate prime should also pass a single iteration of the probabilistic Lucas test (D.3) before being accepted as a (probable) prime. As, to date, there are no known examples of composites which pass a single iteration of Miller-Rabin (with base/witness $a = 2$) and a single iteration of the Lucas test, i.e. Baillie-PSW pseudo-primes.[15]

In general, the estimates depend on two parameters:

— k, the bit length of the numbers generated; and

— t, the number of times the test is iterated on each candidate.

A.2 Worst-case error estimate for *t* Miller-Rabin primality tests

The probability that a composite number is accepted by t (independent) Miller-Rabin tests is at most $(¼)^t$ (see Reference [9]). Thus, to ensure an error probability of at most 2^{-100}, t shall satisfy $t \geq 50$.

A.3 Average-case error estimates for *t* Miller-Rabin primality tests

Table A.1 and Table A.2 contain estimates of the base-2 logarithm of the average-case error probability for t Miller-Rabin tests. The underlined entries correspond to the minimum number of independent Miller-Rabin tests required to reach the 2^{-100} threshold. For instance, the entry for $k = 256$ and $t = 16$ is −101, showing that the error probability for 16 Miller-Rabin tests of a 256-bit number is 2^{-101}. The second sub-table shows that 4 Miller-Rabin tests of a 1 024-bit number limit the error probability to 2^{-109}. The entries are derived from the formula provided in Appendix F of Reference [10] based on results in References [5] and [9].

Table A.1 — Average-case error estimates for *t* Miller-Rabin primality tests (*k* = 256)

k\t	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
256	-80	-84	-88	-91	-95	-98	<u>-101</u>	-104

Table A.2 — Average-case error estimates for *t* Miller-Rabin primality tests ($k \geq 512$)

k\t	**1**	**2**	**3**	**4**	**5**	**6**	**7**
512	-26	-47	-61	-73	-83	-92	-100
1 024	-42	-72	-93	-109	-124	-137	-148
1 536	-56	-92	-117	-137	-155	-171	-186
2 048	-67	-109	-137	-161	-182	-200	-217
3 072	-86	-137	-172	-201	-227	-249	-270
4 096	-103	-160	-201	-235	-264	-291	-315
6 144	-130	--200	-250	-292	-328	-361	-391

The average case error estimates for *t* iterations of the test assume that every iteration uses a random base. Therefore, if one first uses a fixed base, e.g. $a = 2$, and then t-1 random bases, then the error probabilities listed here should be used as if only t-1 iterations have been made.

For bit-lengths 4 096 or greater, $t = 1$ satisfies the error probability bound of 2^{-100}. However, implementations can want to consider using an additional round as a precaution that the necessary conditions required for use of these tables are not strictly met.

In general, applying additional rounds of Miller-Rabin primality tests are permitted. For example, certain applications may want the average-case error estimates to match the intended security level for which the prime(s) are to be used. Further, for a fixed security level the size of the primes can differ depending on the public key algorithm. For example, a 3 072-bit prime *p* used in Diffie-Hellman (over a prime field) offers roughly 128 bits of security, whereas the prime factors for a 3 072-bit RSA modulus are 1 536-bit numbers. Thus, in the case of Diffie-Hellman, 3 rounds of Miller-Rabin primality testing would be required to match the 128 bit security level, and 4 rounds would be required for each of the prime factors of a RSA modulus.

NOTE The average-case probabilities decrease as the bit-length of the prime increases.

Annex B
(normative)

Generating primes with side conditions

B.1 General

In some cases, it is necessary to generate a prime that satisfies certain extra conditions. For example:

— in certain cryptographic applications it can be necessary that the generated prime be congruent to 3 modulo 4. For example, in the Feige-Fiat-Shamir identification protocol, the modulus is a product of two primes subject to this restriction. Further, in any RSA application using (public) exponent e, only primes p with $\gcd(p\text{-}1,e) = 1$ shall be used as factors of the RSA modulus (as e should be invertible in Z_N);

— the prime should be in a certain interval (e.g., for RSA it may not be sufficient that the prime consists of k bits, but its product with a second prime should result in a $2k$-bit number).

In the constructions described below, an integer x is generated subject to certain congruence conditions and/or size conditions. If x is randomly generated under these constraints and tested for primality using the Miller-Rabin primality test, this document allows the reduced number of iterations given in Table A.2 plus one. The probable prime should also pass a single iteration of the probabilistic Lucas test, D.3. That is, the same number of iterations allowed for the incremental search method of 8.3.3 applies. It follows from Dirichlet's theorem[13] that primes are evenly distributed across congruence classes (which are coprime to the modulus), and so these additional constraints should not affect the average-case analysis leveraged in A.3.

Care should be taken that the congruence modulus is not too large and/or the size of the interval too small. If the primes are being generated for use in RSA, the total number of bits of constraints resulting from the interval size restriction and/or congruence conditions shall not exceed 20 so as to avoid Coppersmith style attacks (see NOTE in 8.2). For example, requiring a number $n < 2^k$ to lie in the interval $(2^{k-2} + 2^{k-1}, 2^k)$ and satisfy $n \bmod 4 = 3$ yields 4 bits of constraint: the interval size requires the top two bits of n to be set and the congruence condition requires the least two significant bits to be set.

Random numbers shall be constructed using a deterministic random bit generator that satisfies the requirements of ISO/IEC 18031 and a conversion method (from sequences of bits to numbers) in Annex C or ISO/IEC 18031.

B.2 Congruence restrictions on primes

B.2.1 General

This document describes the generation of primes based on congruence conditions.

B.2.2 Congruence restrictions and incremental or random search

Let m be a modulus, e a positive odd integer, and r an integer satisfying $0 \le r < m$. Given a primality test T, a k-bit prime N satisfying $N \bmod m = r$ and $\gcd(N\text{-}1,e)=1$ shall be generated as follows.

a) If m is odd:

If r is even, set $r = r + m$ and set $m = 2m$.

b) Select an integer x at random from the interval $(2^{k-1}, 2^k - m]$.

```
c) Set $p = x + ((r-x) \bmod m)$.
d) If $\gcd(p-1, e) \neq 1$;
     go to step b).
e) Apply primality test $T$ to $p$.
   If test returns "$p$ prime";
     return $p$ and stop.
   If $p \geq 2^k - m$;
     go to step b);
   else
     set $p = p + m$ and go to step d) (incremental search);
   or
     go to step b) (random search).
```

Rather than increment the candidate prime p in step e) by m, for some $J < (2^k - p) / 2m$, T may be applied to elements in the sequence p, $p+2m$, $p + 4m$, ..., $p + 2Jm$ which survive the sieving procedure described in C.1. The sieve process eliminates candidate primes that are divisible by small primes used in the sieve.

In case only the constraint $N \bmod m = r$ is needed, then step d) may be omitted. If only the constraint $\gcd(N-1,e)=1$ is needed, then step a) may be admitted upon setting $m = 2$ and $r = 1$.

NOTE 1 Step e) allows two different search methods, incrementing by m or generating a new candidate at random (subject to the fixed congruence condition).

NOTE 2 To generate a prime p such that $p \bmod m$ lies in some subset S of $\{1, 2, ..., m-1\}$, one can first select a random r in S and then apply the above algorithm to construct a prime p satisfying $p \bmod m = r$.

NOTE 3 For an odd modulus m, step a) is necessary to ensure the integer p in step c) is odd.

NOTE 4 Care needs to be taken in consideration of the choices of e, r, and m. For example, if m and r are even, the candidate prime p in step c) is always even and the algorithm fails to return a prime. Similarly, the values $e = 3$, $r = 1$, $m = 3$ result in no primes being returned as the condition $N \bmod 3 = 1$ forces $\gcd(N-1,e) = 3$.

Common applications of the above algorithm follow.

a) If T equals the Miller-Rabin primality test, then the algorithm returns a random probable prime N subject to the condition that $N \bmod m = r$. The trivial case of $m = 2$ and $r = 1$, ensures the candidate prime value p in step c) is odd.

b) The case T equals the Pocklington test (so m = a prime of the necessary size and $r = 1$) is used in the Shawe-Taylor algorithm (see 8.4.2). If requested, the algorithm may also return a (Pocklington) certificate of primality.

c) Given two auxiliary primes p_1 and p_2, the above algorithm can be used to return a prime N with p_1 dividing N-1 and p_2 dividing $N + 1$ by setting $m = p_1\, p_2$ and $r = ((p_2^{-1} \bmod p_1) \cdot p_2 - (p_1^{-1} \bmod p_2) \cdot p_1) \bmod m$. For example, if T = Brillhart-Lehmer-Selfridge test and the necessary conditions are satisfied, the algorithm returns a provable prime N (and a certificate of primality if requested). If T equals the Miller-Rabin primality test, a probabilistic prime is returned.

B.2.3 Congruence restrictions and The Shawe-Taylor algorithm

In the final recursion of The Shawe-Taylor algorithm (8.4.2), the returned k-bit prime N satisfies $N \bmod 2q = 1$ for some prime q. This may be modified to ensure the additional congruence condition $N \bmod m = r$ (with $m \neq q$) as follows. If m is prime to $2q$, set $m_0 = 2qm$, $r_0 = (((2q)^{-1} \bmod m) \cdot 2q + (m^{-1} \bmod 2q) \cdot m)$

mod m_0. If 2 divides m (note that r should be odd to ensure p is odd) set $m_0 = qm$, $r_0 = ((q^{-1} \bmod m) \cdot q + (m^{-1} \bmod q) \cdot m) \bmod m_0$. In either case, the integer value $p = x + (r_0 - x) \bmod m_0$ satisfies $p \bmod m = r$ and $p \bmod 2q = 1$. Thus, for the final recursion, steps b) through d) may now be changed to the following.

b) Select an integer x at random from the interval $(2^{k-1}, 2^k - m_0]$.

c) Set $p = x + (r_0 - x) \bmod m_0$, $t = (p - 1) \operatorname{div} q$.

d) Apply the Pocklington test to p, with p-1 = $F\,R$ where $F = q$, $R = t$.

If test returns "p prime", return p along with certificate $C(p)$ (if a certificate is requested) and stop.

If $p < 2^k - m_0$, set $p = p + m_0$, $t = t + (m_0 \operatorname{div} q)$, and go to step d).

Otherwise, go to step b).

If N should satisfy the constraint gcd(N-1, e) = 1, for some integer e (as the case with RSA where e = (public) exponent), the p in step d) should be checked to satisfy gcd(p-1,e) = 1 before applying the more computationally expensive Pocklington test.

Rather than increment the candidate prime p in step c) by m_0 [in step d)], for some J in ($(2^{j-1} - p) / m_0$, $(2^j - p) / m_0$), the Pocklington test may be applied to elements in the sequence p, $p+m_0$, $p+2\,m_0$, ..., $p+J\,m_0$ which survive the sieving procedure described in C.1. The sieve process eliminates candidate primes that are divisible by small primes used in the sieve.

B.2.4 Generating primes in an interval

The text in this document deals with generating a random k-bit prime, i.e. a prime p such that $2^{k-1} \leq p < 2^k$. To generate a prime between some arbitrary lower bound A and upper bound B, one may apply any of the recommended algorithms described earlier, but replace 2^{k-1} and 2^k by A and B, respectively. Depending on the interval, this may also be accomplished by simply setting a certain number of high bits of the prime. Some care should be taken in selecting the values for A and B, e.g. if $B - A$ is small, certain algorithms can fail. This is certainly the case for generating primes based on congruence conditions (B.2) when $B - A < m$.

A common application is to generate $2k$-bit RSA modulus $n = pq$ where p and q are k-bit primes. In this case, taking $A = 2^{k-1}\sqrt{2}$ (or $A = 2^{k-1} + 2^{k-2}$, i.e. generate primes with the high two bits set) and $B = 2^k$ suffices.

For the case of the Shawe-Taylor algorithm, the interval restriction needs to only be applied to step b) of the final recursion step (which returns a k-bit prime). Further, the check of $p < 2^j - 2q$ in step d) is replaced (in the final recursion step) with $p < B - 2q$. Similarly, when employing the sieving method (in any of these algorithms), the interval being sieved should be contained in $[A, B]$.

Annex C
(normative)

Additional random number generation methods

C.1 General

Methods of generating numbers from a sequence of random bits are provided in ISO/IEC 18031. This document describes three additional methods, two of which are akin to the simple discard method and the simple modular method of ISO/IEC 18031 with the exception that the sequence of bits are used in reverse order to generate a random number. With respect to this ordering, the first (random) output bit becomes the most significant bit of the generated number.

The discard and modular methods generate a random number in the interval $[0, r)$ for some bound r. If the random number is to be bounded below by a value A, the random number generation method may be repeated until the returned value satisfies the lower bound A. Alternatively, these algorithms may be used to generate a random number, say a, in $[0, r - A)$ and instead return the number $a + A$ [which lies in $[A, r)$].

C.2 Simple conversion method

To construct a random m-bit integer:

a) use the DRBG to generate a sequence of m random bits, $(b_0, b_1, ..., b_{m-1})$;

b) return $c = b_{m-1} + 2\, b_{m-2} + ... + 2^{m-2}\, b_1 + 2^{m-1}\, b_0$.

In applications where random numbers are to be generated with certain bits fixed, the corresponding bits in step a) may be appropriately set.

C.3 Simple discard method (reverse order)

Let m be the unique positive integer satisfying $2^{m-1} \leq r \leq 2^m - 1$.

a) Use the DRBG to generate a sequence of m random bits, $(b_0, b_1, ..., b_{m-1})$.

b) Let $c = b_{m-1} + 2\, b_{m-2} + ... + 2^{m-2}\, b_1 + 2^{m-1}\, b_0$.

c) If $c < r$, then return c, else discard c and go to step a).

C.4 Simple modular method (reverse order)

Let m be the unique positive integer satisfying $2^{m-1} \leq r \leq 2^m - 1$, and let l be a security parameter.

a) Use the DRBG to generate a sequence of $m+l$ random bits, $(b_0, b_1, ..., b_{m+l-1})$.

b) Let $c = b_{m+l-1} + 2\, b_{m+l-2} + ... + 2^{m+l-2}\, b_1 + 2^{m+l-1}\, b_0$.

c) Return $c \bmod r$.

As the mod operator is not uniform, a value of $l = 64$ is often recommended to make the effects negligible.

Annex D
(normative)

Auxiliary methods

D.1 Sieving procedure

Given a sequence of integers $S = Y_0, Y_0+h, \ldots, Y_0 + J \cdot h$ for some non-negative integers h and J, this procedure identifies which integers in the sequence are divisible by prime factors up to some limit K. A typical value for K is between 10^3 and 10^5. Proceed as follows.

a) Select a factor base of all primes from 2 up to K, i.e. $B = \{2, 3, 5, \ldots, p_k\}$, where $p_k \leq K$.

b) Initialize each element of an array A (with indices in $[0,J]$) to zero:

for $i = 0,1, \ldots, J$, set $A[i] = 0$.

c) For each p_j in B, do:

```
1) set $i = (-h^{-1}Y_0) \bmod p_j$.
2) while ($i \leq J$):
    i) set $A[i] = 1$;
    ii) set $i = i + p_j$.
```

Upon the completion of step c), the following assertions are true for each integer $i \in [0, J\,]$:

— $Y_0 + i \cdot h$ is divisible by some prime in B if and only if $A[\,i\,] = 1$.

— Equivalently $Y_0 + i \cdot h$ is divisible by none of the primes in B if and only if $A[\,i\,] = 0$.

D.2 Primality tests based on Pocklington's theorem

D.2.1 General

NOTE 1 The algorithm given here produces a primality certificate for verification purposes. If an integer passes this test, it is guaranteed to be prime. However, if aborted early, the integer in question can be composite or its primality undetermined.

The primality testing method provided in D.2 is based on the following fact (a variant of Pocklington's theorem).

Given an odd integer $N > 1$, suppose that there exists a factorization $N - 1 = F \cdot R$, where F and R are positive integers, and the prime factorization of F is known. Suppose further that the unique non-negative integers s and r satisfying $R = s \cdot F + r$ and $0 \leq r < F$ also satisfy $s < F + r$.

If, for each prime factor q of F, there exists an integer $a \in [2, N - 1]$ such that both:

1) $a^{N-1} \bmod N = 1$; and

2) $\gcd(a^{(N-1)/q} - 1, N) = 1$;

then N is prime if and only if either $s = 0$ or $r^2 - 4s$ is not a perfect square.

NOTE 2 The value of a can be different for each prime q dividing F.

D.2.2 Pocklington's primality test

To test whether an integer $N \geq L$ (the trial division bound) is prime given the prime factorization $F = q_1^{e_1} q_2^{e_2} \ldots q_n^{e_n}$, where $N - 1 = F{\cdot}R$, Pocklington's theorem shall be applied as follows.

a) Let $R = sF + r$, where $0 \leq r < F$ and $0 \leq s$.

If $s \geq F + r$, return "test inconclusive" and stop (because the conditions needed to apply this test are not met).

If $s > 0$ and $r^2 - 4s$ is a perfect square, return "N composite" and stop.

b) For i from 1 to n, do the following:

1) set $j = 0$;

2) select an appropriate value of T (e.g., from Table D.1), based on the value q_i;

3) while ($j < T$):

i) select an integer a in [2, N-1];

If $a^{N-1} \bmod N \neq 1$, return "N composite" and stop.

If gcd($(a^{(N-1)/q_i} - 1) \bmod N, N) = 1$, set $a_i = a$ and exit while loop.

If gcd($(a^{(N-1)/q_i} - 1) \bmod N, N) \neq N$, return "$N$ composite" and stop.

ii) set $j = j + 1$.

4) If $j = T$, return "test inconclusive" and stop.

c) Return "N prime" and stop. If a certificate is requested, return certificate C containing the values $\{(N,q_1,a_1), (N,q_2,a_2), \ldots,(N,q_n,a_n)\}$.

Table D.1 — Number of *a* to test for a given *q*

q	T = number of a to test
2	7
3	5
5 or 7	3
$11 \leq q \leq 97$	2
$97 < q$	1

If N is prime then for a given q, the probability that a randomly chosen a satisfies both conditions 1 and 2 of D.2.1 is $1 - 1/q$. The value of T in Table D.1 is set so that the probability an a fails step b) 3) is less than 0,01, provided N is prime.

NOTE 1 Testing if $r^2 - 4s$ is a perfect square in step a) can be done using one of the methods listed in D.9.

NOTE 2 The failure rate of the test, that is, the rate the test is inconclusive, can be reduced by increasing the value of T selected in step b) 2).

NOTE 3 The a selected in step 3) can be selected non-randomly, e.g. implementations can start with $a = 2$, and increment within the while loop to the next non-perfect, integer power. For example, if $a = 2$ has been selected in step 3), the values $a = 4, 8, 16, \ldots$ can be skipped as check 3) i) would be superfluous, i.e. $2^{N-1} \bmod N = 1$ implies $4^{N-1} \bmod N = 1$, $8^{N-1} \bmod N = 1$, etc.

NOTE 4 The q_i appearing in the factorization of F can themselves be accompanied by certificates of primality. Assuming the accompanying certificates are correct, they are also included in the certificate C returned in step c).

Verifying the above certificate C for N shall be done as follows.

a) Set $F = 1$.

b) Set $R = N$-1.

c) For $i = 1$ to n:

   1) Verify that q_i is (deterministically) prime by any method given in this document.

      If q_i is not prime,

        return “certificate invalid” and stop.

   2) If $a_i^{N\text{-}1} \bmod N \neq 1$,

        return “certificate invalid, N composite” and stop.

   3) If gcd($(a_i^{(N\text{-}1)/q_i}$ - 1) mod N, N) ≠ 1, N,

        return “certificate invalid, N composite” and stop;

      else, if gcd($(a^{(N\text{-}1)/q_i}$ - 1) mod N, N) = N,

        return “certificate invalid” and stop.

   4) while (R mod q_i) = 0:

      i) set $R = R$ div q_i;

      ii) set $F = F \cdot q_i$.

d) Compute the ordered pair of integers (r,s) where $R = sF + r$ with $s \geq 0$ and $0 \leq r < F$.

e) If $s \geq F + r$, return “certificate invalid” and stop.

   If $s = 0$ or $r^2 - 4s$ is a non-square, accept the certificate, return “N prime” and stop.

   Otherwise, return “certificate invalid, N composite” and stop.

NOTE 5 The value of F used in the attempt to verify the certificate $\{(N,q_1,a_1), (N,q_2,a_2), ...,(N,q_n,a_n)\}$ can involve higher powers of those primes than did the value of F used to generate that certificate, since the while loop in step c) 4) constructs the largest F dividing N-1 with prime factors in the list $\{q_1, q_2, ..., q_n\}$. It follows that the “recovered” values of R, s, and r can also be different from those used to originally construct the certificate for N.

NOTE 6 The generation and verification algorithms as listed in this document is adapted from theorem 5 of Reference [6]. The condition $s < F + r$ is equivalent to the condition $N < F^3 + r\cdot F^2 + r\cdot F + 1 = (F + 1)(F^2 + (r\text{-}1)\cdot F + 1)$.

D.2.3 Partial Pocklington’s primality test

If the condition $s < F + r$ is not satisfied, primality of the number being tested can still be established by combining results with the partial deterministic Lucas primality test (D.4) as specified in the Brillhart-Lehmer-Selfridge test (D.5). The algorithm still returns a “partial” certificate – the full certificate is the partial certificate combined with the partial deterministic Lucas certificate.

This weakened version is referred to as the partial Pocklington’s primality test and modifies the Pocklington’s primality test as follows.

— The condition $s \geq F + r$ is removed in step a).

— Step c) returns “N passes” with the partial certificate C containing the values $\{(N,q_1,a_1), (N,q_2,a_2), ..., (N,q_n ,a_n)\}$.

Verification of the partial certificate C is modified similarly.

— Step e) checking $s \geq F + r$ is removed.

— Step e) returns “N passes” and the value F computed in step c) if the conditions on s and $r^2 - 4s$ are met, and returns “certificate invalid, N composite” otherwise.

D.3 Probabilistic Lucas primality test

The primality testing method provided here is based on the following fact (see Reference [3]):

If N is prime and D is integer with $D \bmod 4 = 1$ such that:

1) Jacobi(D, N) = –1;

2) gcd(Q, N) = 1 ($Q = (1 - D)/4$);

then Lucas(D, N + 1, N) = 0.

To test whether an odd integer $N \geq L$ (the trial division bound) is prime, the probabilistic Lucas test shall be applied as follows.

a) If N is a perfect square, return “N composite” and stop.

b) Set $D = 5$ and $Q = (1 - D)/4$.

c) While (Jacobi(D, N) ≠ -1 or gcd(N, Q) ≠ 1):

   1) If Jacobi(D,N) = 0 and $D \bmod N \neq 0$, return “N composite” and stop.

   2) If gcd(N, Q) ≠ 1 and $Q \bmod N \neq 0$, return “N composite” and stop.

   3) Set $D = -\operatorname{sgn}(D) \cdot (\operatorname{sgn}(D) \cdot D + 2)$ and $Q = (1 - D)/4$.

d) If Lucas(D, N+1, N) = 0, return “N accepted” and stop.

   Otherwise, return “N composite” and stop.

NOTE 1 Methods for testing for a perfect square are given in D.9, for computing the Jacobi symbol in D.10, and Lucas(D,N+1,N) in D.11.

NOTE 2 The test in step a) is performed to avoid an infinite loop in step c) in case N is a perfect square.

NOTE 3 The value D starts with $D = 5$ in step b) as Jacobi(1,N) is always 1 and ranges over the set {5, -7, 9, -11, 13, -15, 17, ...}.

D.4 Lucas’ deterministic primality tests

D.4.1 General

NOTE 1 The algorithm given here produces a primality certificate for verification purposes. If an integer passes this test, it is guaranteed to be prime. However, if aborted early, the integer in question can be composite or its primality undetermined.

The primality testing method provided in D.4 is based on the following fact:

Given an odd integer $N > 1$, suppose that there exists a factorization $N + 1 = F \cdot R$, where F and R are positive integers, and the prime factorization of F is known. Suppose further that the unique non-negative integers s and r satisfying $R = s \cdot F + r$ and $0 \leq r < F$ also satisfy $s + r < F$.

If for each prime factor q of F there exists an integer D with D mod 4 = 1 such that:

1) Jacobi(D, N) = –1;

2) Lucas(D, N+1, N) = 0; and

3) gcd(Lucas(D, (N+1)/q, N), N) = 1;

then N is prime if and only if either $s = 0$ or $r^2 + 4s$ is not a perfect square.

The choice of D can be different for each of the q_i dividing F. Furthermore, if for a particular D, condition 1 holds true but condition 2 fails, then N is composite and the test may be stopped.

NOTE 2 Methods for testing for a perfect square are given in D.9, for computing the Jacobi symbol in D.10, and Lucas(D,N+1,N) in D.11.

D.4.2 Deterministic Lucas primality test

To test whether an integer $N \geq L$ (the trial division bound) is prime given the prime factorization $F = q_1^{e_1} q_2^{e_2} \ldots q_n^{e_n}$, where $N + 1 = F \cdot R$, the Lucas test shall be applied as follows.

```
a) Let $R = sF + r$, where $0 \leq r < F$ and $0 \leq s$.

   If $s + r \geq F$, return "test inconclusive" and stop (because the conditions needed to apply the test have not been satisfied).

   If $s \neq 0$ and $r^2 + 4s$ is a perfect square, return "$N$ composite" and stop.

   If "$N$ passes" is returned with a partial certificate.

b) For $i$ from 1 to $n$:

   1) Select an appropriate value of $T$ (e.g., from Table D.2), based on the value of $q_i$.

   2) Set $j$=0, set $D$=5.

   3) While (Jacobi($D$, $N$) ≠ -1 and $j < T$):

      i) If Jacobi($D$, $N$) = 0:

           If $D$ mod $N \neq 0$, return "$N$ composite" and stop.

           Otherwise, set $j = j + 1$.

      ii) Set $D = -\,\mathrm{sgn}(D) \cdot (\mathrm{sgn}(D) \cdot D + 2)$, i.e. select the next value for $D$.

   4) If $j = T$, return "test inconclusive" and stop.

   5) If Lucas($D$, $N + 1$, $N$) ≠ 0:

        If $Q$ mod $N \neq 0$ where $Q = (1 - D)$ div 4, return "$N$ composite" and stop.

        Otherwise, set $D = -\,\mathrm{sgn}(D) \cdot (\mathrm{sgn}(D) \cdot D + 2)$ and go to step b) 3).

   6) If gcd(Lucas($D$, $(N + 1)/q_i$, $N$), $N$) = 1, set $D_i = D$.

        Otherwise, if Lucas($D$, $(N + 1)/q_i$, $N$) ≠ 0, return "$N$ composite" and stop.

        Otherwise, set $D = -\,\mathrm{sgn}(D) \cdot (\mathrm{sgn}(D) \cdot D + 2)$ and go to step b) 3).

c) Return "$N$ prime." If certificate is requested, return certificate with values {($N$,$q_1$,$D_1$), ($N$,$q_2$,$D_2$), ..., ($N$,$q_n$,$D_n$)}. Stop.
```

Table D.2 — Number of D to test for a given q

q	T = number of D to test
2	17
3	12
5,7	10
$11 \leq q \leq 23$	8
$23 < q$	7

If N is prime then for a given q, the probability that a random D satisfies conditions 1 through 3 of D.4.1 is about $(1/2)\cdot(1 - 1/q)$. The value of T in Table D.2 is set so that the probability a D fails in step b) 3) is less than 0,01 if N is prime.

NOTE 1 The q_i appearing in the factorization of F can themselves be accompanied by certificates of primality. Assuming the accompanying certificates are correct, they are also included in the certificate C returned in step c).

NOTE 2 Testing $r^2 + 4s$ is a perfect square in step a) can be done using the deterministic methods in D.9.

NOTE 3 The condition in step b) 5) is a result of the following facts:

1) for prime N with Jacobi(D, N) = -1 and gcd(Q, N) = 1, then Lucas(D, N + 1, N) = 0; that is, if Lucas(D, N + 1, N) ≠ 0, Jacobi(D, N) = -1, and gcd(Q, N) = 1, then N cannot be prime;

2) if 1 < gcd(Q, N) < N, then N is not prime. Thus if Lucas(D, N + 1, N) ≠ 0, Jacobi(D, N) = -1 and gcd(Q, N) < N, or equivalently Q mod N ≠ 0, then N is not prime.

Verifying the above certificate C for N shall be done as follows.

```
a)  Set $F = 1$.
b)  Set $R = N + 1$.
c)  For $i = 1$ to $n$:
    1)  If $q_i$ is not prime, return “certificate invalid” and stop.
    2)  If Jacobi($D$, $N$) = 0:
            If $D$ mod $N \neq 0$, return “certificate invalid, $N$ composite” and stop.
            Otherwise, return “certificate invalid” and stop.
    3)  If Jacobi($D$,$N$) = 1, return “certificate invalid” and stop.
    4)  If Lucas($D$, $N + 1$, $N$) ≠ 0:
            If $Q$ mod $N \neq 0$ where $Q = (1 - D)$ div 4, return “certificate invalid, $N$ composite” and stop.
            Otherwise, return “certificate invalid” and stop.
    5)  If gcd(Lucas($D_i$, $(N + 1)/q_i$, $N$), $N$) ≠ 1:
            If Lucas($D_i$, $(N + 1)/q_i$, $N$) ≠ 0, return “certificate invalid, $N$ composite” and stop.
            Otherwise, return “certificate invalid” and stop.
    6)  While ($R$ mod $q_i$) = 0:
        i)  Set $R$ = R div $q_i$.
        ii) Set $F = F \cdot q_i$.
```

d) Compute (r,s) where $R = sF + r$ with $s \geq 0$ and $0 \leq r < F$.

e) If $s + r \geq F$, return “certificate invalid” and stop.

f) If either $s = 0$ or $r^2 + 4s$ is not a perfect square, accept the certificate, return “N prime” and stop.

Otherwise, return “certificate invalid, N composite” and stop.

Verification of the primality of q_i in step c) 1) shall be done using a deterministic method in this document.

NOTE 4 The value of F used in the attempt to verify the certificate $\{(N,q_1,D_1), (N,q_2,D_2), ...,(N,q_n,D_n)\}$ can involve higher powers of those primes than did the value of F used to generate that certificate, since the while loop in step c) 6) constructs the largest F dividing N+1 with prime factors in the list $\{q_1, q_2, ..., q_n\}$. It follows that the “recovered” values of R, s, and r can also be different than those used to originally construct the certificate for N.

NOTE 5 The generation and verification algorithms as listed in this document is adapted from theorem 17 of Reference [6] The condition $s + r < F$ is equivalent to the condition $N < F^3 - r \cdot F^2 + r \cdot F - 1 = (F - 1) (F^2 + (1 - r) \cdot F + 1)$.

D.4.3 The partial deterministic Lucas primality test

If the condition $s + r < F$ is not satisfied, primality of the number being tested may still be established by combining results with the partial Pocklington’s primality test (D.2) as specified in the Brillhart-Lehmer-Selfridge test (D.5). The algorithm still returns a “partial” certificate – the full certificate is the partial certificate combined with the partial Pocklington certificate.

This weakened version is referred to as the partial deterministic Lucas primality test and modifies the deterministic Lucas primality test as follows.

— The condition $s + r \geq F$ is removed in step a).

— Step c) returns “N passes” with the partial certificate C containing the values $\{(N,q_1,a_1), (N,q_2,a_2), ...,(N,q_n, a_n)\}$.

Verification of the partial certificate C is modified similarly:

— Step e) which checks $s + r \geq F$ is removed.

— Step f) returns “N passes” and the value F computed in step c) if the conditions on s and $r^2 + 4s$ are met, and returns “certificate invalid, N composite” otherwise.

D.5 Brillhart-Lehmer-Selfridge primality test

NOTE 1 The algorithm given here produces a primality certificate for verification purposes. If an integer passes this test, it is guaranteed to be prime. However, if aborted early, the integer in question can be composite or its primality undetermined.

NOTE 2 This method can be used for generating strong primes, i.e. primes p where p+1 and p-1 have a large prime divisor.

The primality testing method provided in D.5 is based on the following.

Given an odd integer $N > 1$, suppose that there exist factorizations $N - 1 = F_1 \cdot R_1$ and $N + 1 = F_2 \cdot R_2$ and the prime factorizations of F_1 and F_2 are both known.

If, for each prime factor q of F_1, there exists an integer $a \in [2, N - 1]$ such that both:

1) $a^{N-1} \equiv 1 \pmod N$; and

2) $\gcd(a^{(N-1)/q} - 1, N) = 1$,

and for each prime factor r of F_2 there exists a (discriminant) D in {5, -7, 9, -11, 13, -15, 17, ...} such that:

1) Jacobi(D, N) = –1;

2) Lucas(D, N+1, N) = 0; and

3) gcd(Lucas(D, (N+1)/r, N), N) = 1;

then any positive divisor d of N shall satisfy: d mod lcm(F_1, F_2) = 1 or d mod lcm(F_1, F_2) = N mod lcm(F_1, F_2).

Thus, if all five conditions 1)-2) and 1)-3) hold, it follows N is prime if:

a) lcm(F_1, F_2) > N;

b) $\sqrt{N}$ -1 < lcm(F_1, F_2) < N and N mod lcm(F_1, F_2) is not a divisor of N; or

c) there does not exist a proper divisor d satisfying d mod lcm(F_1, F_2) = 1.

NOTE 3 If either a) or b) holds, then N is prime as all proper divisors of N would be larger than $\sqrt{N}$. If c) holds, N is also prime. Otherwise, N mod lcm(F_1, F_2) = N^2 mod lcm(F_1, F_2) (as N is a product of two proper divisors), i.e. N mod lcm(F_1, F_2) = 1 which contradicts c).

NOTE 4 If lcm(F_1, F_2) > $N^{1/3}$, then c) can be tested using Lenstra's algorithm (see D.12).

NOTE 5 The value of a can be different for each q dividing F_1 and the value of D can be different for each r dividing F_2. Table D.1 gives an indication of the number of values to test for a given q. Table D.2 gives an indication of the number of D values to test for a given r.

To test whether an integer $N \geq L$ (the trial division bound) is prime given the prime factorization $F_1 = q_1^{e_1} q_2^{e_2} \dots q_n^{e_n}$ and $F_2 = r_1^{f_1} r_2^{f_2} \dots r_m^{f_m}$ where N -1 = $F_1 \cdot R_1$ and N+1 = $F_2 \cdot R_2$, the result above shall be applied as follows.

a) If lcm(F_1, F_2) ≤ $N^{1/3}$, stop; the necessary conditions for this test are not met.

b) Apply the partial Pocklington's primality test to N using the partial factorization N – 1 = $F_1 \cdot R_1$.

   If "test inconclusive" is returned, return "test inconclusive" and stop.

   If "N composite" is returned, return "N composite" and stop.

   If "N passes" is returned with a partial certificate with values $\{(N,a_1,q_1), (N,a_2,q_2), \dots, (N,a_n,q_n)\}$, proceed to the next step.

c) Apply the partial deterministic Lucas primality test to N using the partial factorization N+1 = $F_2 \cdot R_2$.

   If "test inconclusive" is returned, return "test inconclusive" and stop.

   If "N composite" is returned, return "N composite" and stop.

   If "N passes" is returned with a partial certificate with values $\{(N,r_1,D_1), (N,r_2,D_2), \dots,(N,r_m ,D_m)\}$, proceed to the next step.

d) If lcm(F_1, F_2) > N, return "N prime." If a certificate is requested, return certificate with values $\{(N,a_1,q_1), (N,a_2,q_2), \dots, (N,a_n,q_n), (N,r_1,D_1), (N,r_2,D_2), \dots,(N,r_m ,D_m)\}$. Stop.

e) If $\sqrt{N}$ -1 < lcm(F_1, F_2) < N:

   If N mod (N mod lcm(F_1, F_2)) = 0, return "N composite" and stop.

   Otherwise, return "N prime." If a certificate is requested, return certificate with values $\{(N,a_1,q_1), (N,a_2,q_2), \dots, (N,a_n,q_n), (N,r_1,D_1), (N,r_2,D_2), \dots,(N,r_m ,D_m)\}$. Stop.

f) Apply Lenstra's residue test D.12 with inputs s =lcm(F_1,F_2) and $n = N$.

If test returns false, i.e. there does not exist a proper divisor d of N with d mod lcm(F_1, F_2) = 1, return "N prime." If a certificate is requested, return certificate with values {(N,a_1,q_1), (N,a_2,q_2), ..., (N,a_n,q_n), (N,r_1,D_1), (N,r_2,D_2), ...,(N,r_m ,D_m)}. Stop.

Otherwise, return "N composite" and stop.

The use of Lenstra's algorithm D.12, can be avoided by requiring only partial factorizations satisfying $\sqrt{N}$ -1 < lcm(F_1, F_2).

Verifying the above certificate C for N shall be done as follows.

a) Verify the partial Pocklington certificate {(N,a_1,q_1), (N,a_2,q_2), ..., (N,a_n,q_n)}.

If returns "certificate invalid, N composite," return "certificate invalid, N composite" and stop.

If returns "certificate invalid," return "certificate invalid" and stop.

If returns "N passes" with a value F, set $F_1 = F$ and proceed to the next step.

b) Verify the partial deterministic Lucas certificate {(N,r_1,D_1), (N,r_2,D_2), ..., (N,r_m ,D_m)}.

If returns "certificate invalid, N composite," return "certificate invalid, N composite" and stop.

If returns "certificate invalid," return "certificate invalid" and stop.

If returns "N passes" with a value F, set $F_2 = F$ and proceed to the next step.

c) If lcm(F_1, F_2) ≤ $N^{1/3}$, return "certificate invalid" and stop.

d) If lcm(F_1, F_2) > N, accept the certificate, return "N prime" and stop.

e) If $\sqrt{N}$ -1 < lcm(F_1, F_2) < N:

If N mod (N mod lcm(F_1, F_2)) = 0, return "certificate invalid, N composite" and stop.

Otherwise, accept the certificate, return "N prime" and stop.

f) Apply Lenstra's residue test, D.12, with inputs s =lcm(F_1, F_2) and $n = N$.

8.5 If test returns false, i.e. there does not exist a proper divisor *d* of *N* with *d* mod lcm(F_1, F_2) = 1, accept the certificate, return "*N* prime" and stop.

Otherwise, return "certificate invalid, N composite" and stop.

NOTE 6 This algorithm is adapted from theorem 20 of Reference [6] and combines the results of the "N–1" test given by the Pocklington test and the "N+1" test given by the deterministic Lucas test. Moreover, conditions 1-2 of D.2.1 and 1-3 of D.4.1 imply all divisors d of N satisfy d mod F_1 = 1 and d mod F_2 = ±1 (theorems 4 and 16 of Reference [6]) from which it follows d mod lcm(F_1, F_2) = 1 or N mod lcm(F_1, F_2).

D.6 Elliptic curve primality test

The point 0 on an elliptic curve denotes the identity element. Additional information regarding elliptic curves can be obtained from ISO/IEC 15946-1 and References [2], [4] and [19].

NOTE 1 The algorithm given here produces a primality certificate for verification purposes. If an integer passes this test, it is guaranteed to be prime. However, if aborted early, the integer in question can be composite or its primality undetermined.

The methods in D.6 are based on the following fact for an integer N with gcd(N,6) = 1. If there exists an elliptic curve E given by $y^2 = x^3+ax+b$ such that:

a) gcd(N,$4a^3+27b^2$) = 1;

b) there is a point $P = (x,y) \neq 0$ on E_N, the curve E reduced modulo N, of order r where:

   1) r is prime;

   2) $r > (N^{1/4}+1)^2$;

then N is prime.

See Reference [19] for properties of elliptic curves modulo composite numbers.

To test whether an integer $N \geq L$ (the trial division bound) is prime given an elliptic curve E with defining equation $y^2 = x^3+ax+b$, the elliptic curve test shall be applied as follows.

a) If gcd($4a^3+ 27b^2$, N) =1 and gcd(N, 6) = 1, proceed to step b).

   If either gcd is greater than 1 and less than N, return "N composite" and stop.

   Otherwise, return "test inconclusive" and stop.

b) Assuming N is prime (so that Z_N is a field), determine t = the order of the putative elliptic curve E_N (the curve E reduced modulo N) and a prime divisor r of t satisfying $r > (N^{1/4}+1)^2$. If no such r exists, return "test inconclusive" and stop. Otherwise, proceed to the next step.

c) Find a point $P = (x,y)$ on E_N of order r, i.e. $P \neq 0$ and $r \cdot P = 0$. If such a point is found, return "N prime" and the certificate containing values (N, r, t, a, b, P). Otherwise, return "test inconclusive" and stop.

A method for finding a point P of order r, is to take an arbitrary nonzero point $Q = (x_0, y_0)$ on E_N and check if $P = (t/r) \cdot Q$ is of order r. If $P \neq 0$ and $r \cdot P \neq 0$, then N is not prime and step c) may return "N composite." If $P = 0$, then another arbitrary element Q can be chosen.

In case N is not prime, the computations in steps b) and c) can fail due to trying to invert a non-invertible element of Z_N; that is, an integer s is found such that gcd(s, N) is a proper divisor of N. In such cases, the algorithm may return "N composite" and stop. However, if N is only divisible by large factors, finding a divisor s of N this way is unlikely, i.e. the test is likely inconclusive. As a precaution, an implementation may perform additional testing, such as several Miller-Rabin primality tests, on N *before* applying the elliptic curve test.

NOTE 2 Care can be needed in step b), since it can otherwise become a time-consuming bottleneck. Computation of t can be done using several algorithms such as Schoof's algorithm[16] or Elkies' and Atkins' improvement.[11] However, it is more efficient to avoid this computational step by constructing curves of prescribed orders using the CM method.[1]

To test whether an integer $N \geq L$ is prime given a certificate with values (N, r, t, a, b, P), proceed as follows:

a) Verify r is prime, e.g., by trial division or by verifying its certificate. If r not prime, return "certificate invalid" and stop.

b) If gcd($4a^3+27b^2$,N) =1 and gcd(N, 6) = 1, proceed to step c).

   If either gcd is greater than 1 and less than N, return "certificate invalid, N composite" and stop.

   Otherwise, return "certificate invalid" and stop.

c) Verify $r > (N^{1/4}+1)^2$. If not, return "certificate invalid" and stop.

d) Verify $P = (x,y)$ lies on E_N, i.e. $y^2 \bmod N = (x^3 + a\,x + b) \bmod N$. If not, return "certificate invalid" and stop.