INTERNATIONAL STANDARD

ISO/IEC 18032

First edition
2005-01-15

Information technology — Security techniques — Prime number generation

Technologies de l'information — Techniques de sécurité — Génération de nombres premiers

Reference number
ISO/IEC 18032:2005(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 18032 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

Information technology — Security techniques — Prime number generation

1 Scope

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

Firstly, this International Standard specifies methods for testing whether a given number is prime. The testing methods included in this International Standard can be divided into two groups:

- Probabilistic primality tests, which have a small error probability. All probabilistic tests described here may declare a composite to be a prime. One test described here may declare a prime to be composite.
- Deterministic methods, which are guaranteed to give the right verdict. These methods use so-called primality certificates.

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

NOTE Readers with a background in algorithm theory may have had previous encounters with probabilistic and deterministic algorithms. We stress that the deterministic methods in this International Standard internally still make use of random bits, and "deterministic" only refers to the fact that the output is correct with probability one.

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

The methods for generating, proving and verifying primality defined by this International Standard are applicable to cryptographic systems based on the properties of the primes.

NOTE The specifications of the tests given in this International Standard define the properties to be tested in the simplest possible form. Following these specifications directly will not necessarily produce the most efficient implementations. This is especially the case for the Frobenius-Grantham test.

2 Normative references

The following referenced documents are indispensable for the application 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 9796-2:2002, *Information technology — Security techniques — Digital signature schemes giving message recovery — Part 2: Integer factorization based mechanisms*

ISO/IEC 15946-1:2002, *Information technology — Security techniques — Cryptographic techniques based on elliptic curves — Part 1: General*

3 Terms and definitions

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

3.1
composite number
composite
Integer $N > 1$ is composite if it is not prime, i.e. there exist divisors of N that are not trivial divisors.

3.2
entropy
Total amount of information yielded by a set of bits. It is representative of the work effort required for an adversary to be able to reproduce the same set of bits.

3.3
Jacobi symbol
Jacobi symbol of a with respect to an odd number n is the product of the Legendre symbols of a with respect to the prime factors of n (repeating the Legendre symbols for repeated prime factors).

NOTE Defined in Annex A of ISO/IEC 9796-2.

3.4
Legendre symbol
Let p be an odd prime, and let a be a positive integer. The Legendre symbol of a with respect to p is defined as $a^{(p-1)/2} \bmod p$.

NOTE Defined in Annex A of ISO/IEC 9796-2.

3.5
primality certificate
Mathematical proof that a given number is indeed a prime. For small numbers primality is most efficiently proven by trial division. In these cases, the primality certificate may therefore be empty.

3.6
prime number
prime
Integer $N > 1$ is prime if the only divisors of N are trivial divisors.

3.7
pseudo-random bit generator
Deterministic algorithm which when given some form of a bit sequence of length k outputs a sequence of bits of length $l > k$, computationally infeasible to distinguish from true random bits.

3.8
trial division
Trial division of a number N means checking all prime numbers smaller than or equal to $\sqrt{N}$ to see if they divide N.

3.9
trivial divisor
Any integer N is always divisible by 1, -1, N and $-N$. These numbers are the trivial divisors of N.

4 Symbols

a mod n – for integers a and n, (a mod n) denotes the (non-negative) remainder obtained when a is divided by n.

C – a primality certificate

$C(N)$ – primality certificate of the number N.

$C_0(N)$ – the empty primality certificate. It indicates that trial division should be used to verify that N is a prime.

gcd – greatest common divisor

$g(x) \bmod (N,f(x))$ – the remainder when the polynomial $g(x)$ is divided by the polynomial $f(x)$, calculating coefficients modulo the integer N

k – number of bits in N

L – limit below which primality is verified by trial division

$\log_b(a)$ – the logarithm of a with respect to base b

$\ln()$ – the natural logarithm (i.e., with respect to the number e)

M – a lower bound on the size of an interval in which a prime number has to be found

$\min\{a,b\}$ – the minimum of the numbers a and b

N – candidate number to be tested for primality, where N is always a positive, odd number.

n_0 – a starting point in an incremental search for a prime

n_{max} – an end point in an incremental search for a prime

T – a (probabilistic) test for primality

Z_N – the set of the integer numbers 0, 1, 2, ..., N-1, representing the ring of integers modulo N.

Z_N^* – the subset of Z_N containing the numbers that have a multiplicative inverse modulo N (i.e., if N is prime: the integer numbers 1, 2, ..., N-1)

$Z_N[x]/f(x)$ – the ring of polynomials modulo $f(x)$ with coefficients in Z_N

$\lfloor x \rfloor$ – the largest integer smaller than or equal to x.

β – a parameter which determines the lower bound of the entropy of the output of a prime generation algorithm

μ – the maximal number of steps in an incremental search for a prime

5 Trial division

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

1. For all primes $p \leq \sqrt{N}$
    a. If $N \bmod p = 0$ then return “reject”
2. Return “accept”

For small integers N, trial division is less computationally expensive than other primality tests. Implementations of any primality test described in this standard may define a trial division bound L, below which trial division is used in order to prove the primality of integers. This International Standard sets no value for L.

NOTE It is assumed that the set of prime numbers below a certain size are already known. One practical way to implement the test may be 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 The size of integers for which trial division is less computationally expensive than another primality test, depends on the test and its implementation. A typical value for L might be $L = 10^3$.

6 Probabilistic primality tests

A probabilistic primality test takes a positive, odd number N as input and returns "accept" or "reject". If N is a composite number, the tests described in this clause output "reject", except with some error probability depending on the test being used. The probability that a composite number is accepted is not negligible. If N is a prime number, the Miller-Rabin test and the Frobenius-Grantham test always output "accept". The Lehmann test rejects a prime number with a probability that is not negligible.

In order to reduce the probability of errors, one usually makes several iterations with the same input (and different choices for the random values).

6.1 Miller-Rabin primality test

On input of a candidate number, N, the Miller-Rabin test starts with the following initialisation step:

1. Determine positive integers t and s such that $N\text{-}1 = 2^t s$, where s is odd.

Subsequently, the Miller-Rabin test proceeds with one or more iterations of the following algorithm, which takes inputs t, s, N and outputs "accept" or "reject". For each iteration, a different base b must be selected, with $2 \le b \le N\text{-}2$.

NOTE One iteration of the algorithm is called testing N for primality with respect to the base b.

1. Choose a random integer b such that $2 \le b \le N\text{-}2$.
2. Let $y = b^s \bmod N$
3. if $y \ne 1$ and $y \ne N\text{-}1$ then do
    a. $i = 1$
    b. while $i < t$ and $y \ne N\text{-}1$ do
        i. $y = y^2 \bmod N$
        ii. if $y = 1$ then return "reject"
        iii. $i = i + 1$
    c. if $y \ne N\text{-}1$ then return "reject"
4. return "accept"

A test for primality does not need to use exactly the same algorithm to be compliant with this International Standard. A test for primality is compliant with this International Standard if one iteration outputs "accept" if and only if one of the following requirements is satisfied for the chosen base:

- $b^s \bmod N$ equals 1
- $b^s \bmod N$ equals N-1
- For some i $(0<i<t)$: $(b^s)^{2^i} \bmod N$ equals N–1

The test only accepts the candidate if all iterations accept. Therefore the test may be stopped as soon as a base not leading to acceptance has been found.

NOTE Informative Annex A contains estimates of the error probability of this test depending on the number of iterations.

6.2 Frobenius-Grantham primality test

This test uses arithmetic in the ring $Z_N[x]/(f(x))$, where $f(x)$ is some polynomial of degree 2. The test is as follows.

On input of an odd number N, the Frobenius-Grantham primality test starts with the following initialisation steps:

1. Test N for divisibility by primes less than or equal to min{50000, $\sqrt{N}$} (trial division).
2. Test if N is a square. If yes, reject N and stop.
3. Determine positive integers r and s such that $2^r s = N^2-1$ and s is odd.

Subsequently, the test proceeds with one or more iterations of the following algorithm, which takes inputs r, s, N and outputs "accept" or "reject". In each iteration, different values for b and c must be selected.

1. Choose random $b,c \in Z_N$ until one of the following is true:
   - Either $\gcd(b^2+4c,N)$ or $\gcd(c,N)$ is a non-trivial divisor of N.
   - The Jacobi symbol of b^2+4c with respect to N is -1 mod N and the Jacobi symbol of $-c$ with respect to N is 1 mod N.

   In the first case, reject N and stop. In the second case continue with the next step.
2. Test if the polynomial $x^{(N+1)/2} \bmod (N, x^2-bx-c)$ has degree 0, or equivalently, is in Z_N. If not, reject N and stop.
3. Test if $x^{N+1} \bmod (N, x^2-bx-c) = -c$. If not, reject N and stop.
4. Test if either $x^s \bmod (N, x^2-bx-c) = 1$ or j $(0 \le j \le r-2)$ exists such that $x^{2^j s} \bmod (N, x^2-bx-c) = -1$. If not, reject N and stop, otherwise accept N.

NOTE The Jacobi symbol with respect to N may be efficiently computed without knowledge of the prime factors of N [16].

NOTE A more detailed description of the test can be found in [9]. In the same document, it is proven that an iteration of the test can be implemented in such a way that its running time is approximately equal to 3 times the running time of one iteration of the Miller-Rabin test.

NOTE Information regarding the error probability of this test can be found in Annex A.

6.3 Lehmann primality test

NOTE The Lehmann test is based on the following fact: An odd integer N > 1 is prime if and only if $\forall\ a \in Z_N^*: a^{(N-1)/2} = \pm 1 \bmod N$ and $\exists\ a \in Z_N^*: a^{(N-1)/2} = -1 \bmod N$.

The Lehmann primality test takes as input a candidate number N and a parameter t indicating the maximum number of iterations to be executed. The test executes the following algorithm.

1. Set f = "false".
2. Do t times

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

b. Let $y = b^{(N-1)/2} \bmod N$.

c. If $y \neq 1$ and $y \neq N-1$ then return "reject".

d. If $y = N-1$, then set f = "true".

3. If f = "true" then return "accept", else return "reject".

NOTE Informative Annex A contains information on the error probability of this test depending on the number of iterations.

7 Deterministic primality verification methods

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 Maurer's algorithm (see Clause 8.3.1)

A primality certificate contains information that enables efficient verification that a given number is a prime. For both types of certificates described in this International Standard, small numbers (i.e. numbers smaller than the trial division bound L) are most efficiently 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 method for computing this certificate can in effect also be used to verify primality. The primality certificate obtained through Maurer's algorithm is generated as part of generating a prime, and it cannot in general be efficiently computed for an arbitrary prime (after the prime has been generated).

7.1 Elliptic curve primality certificate

Information regarding elliptic curves can be found in ISO/IEC 15946-1.

NOTE The method is based on the following fact: Let $\gcd(N,6) = 1$ and let r be a prime such that $r > (N^{1/4}+1)^2$. If there exists a (non-singular) elliptic curve $y^2 = x^3+ax+b$ modulo N and a point $P \neq 0$ on this curve with order r, then N is prime.

To prove that a number is prime the method is used recursively. The recursion parameter is denoted by r. The total collection of data generated by the method is organised in a primality certificate. Checking the certificate, and thereby proving that N is prime, is considerably faster than generating the certificate.

NOTE If the number N is not prime, then the method will search for an elliptic curve that doesn't exist. Hence, the method will run indefinitely, unless precautions are taken. If the method is stopped before it produces a primality certificate (e.g. after a predetermined amount of time), the number N cannot be judged to be prime or composite.

7.1.1 Elliptic curve primality certificate generation

On input of an integer number N with $\gcd(N,6) = 1$, the elliptic curve primality certificate generation starts with the following initialisations:

1. Let $C = \{\}$
2. Let $j = 0$
3. Let $r = N$

Subsequently the following steps are executed for decreasing values of r, until $r \le L$:

1. $j = j + 1$.
2. $t_{min} = (\sqrt[4]{r}+1)^2$.
3. Determine integers t, a, b and a point P such that the following conditions are satisfied:
   a. t is a probable prime and $t \ge t_{min}$
   b. The order of the elliptic curve E: $y^2 = x^3 + ax + b \bmod r$ is a multiple of t.
   c. P is a point on E with order t.
4. Let $C_j = (r,t,a,b,P)$.
5. Append C_j to C.
6. $r = t$.

When $r \le L$, the primality of r is proven by means of trial division. The output of the algorithm is the sequence C.

NOTE Algorithms to implement step 3.b. efficiently are described e.g. in [2].

7.1.2 Elliptic curve primality certificate verification

For a number N with $\gcd(N,6) = 1$, an elliptic curve primality certificate is a sequence $C = \{C_1, C_2, \ldots, C_k\}$, where:

$$C_i = (p_i, r_i, a_i, b_i, P_i)$$

such that:

- $p_1 = N$.
- r_k is a prime less than L.

and for all $i = 1, 2, \ldots, k$:

1. $\gcd(p_i,6) = 1$.
2. P_i is a point on the elliptic curve E: $y^2 = x^3 + a_i x + b_i \bmod p_i$,
3. P_i has order r_i.
4. $p_{i+1} = r_i$ for $1 \le i < k$.
5. $r_i > (p_i^{1/4}+1)^2$.

The number N is prime if it has such a certificate.

7.2 Primality certificate based on Maurer's algorithm

This type of primality certificate can be computed only during the generation process of the prime number, which is described in Clause 8.3.1.

NOTE Maurer's algorithm is based on the following fact. Let $N = 1 + 2Rq$, where q is an odd prime and $q > R$. If an integer a exists satisfying $a^{N-1} = 1 \bmod N$ and $gcd(a^{2R}-1,N) = 1$, then N is prime.

The proof that an odd number, $N \geq L$, is prime has the following structure:

$$\text{Proof}(N) = (N, q, a)$$

This tuple is a proof of primality if the following conditions are fulfilled:

- q is an odd prime such that:
  - q is a divisor of $N-1$.
  - $q^2 > (N-1)/2$.
- a is an integer such that $1 < a < N$ and
  - $a^{N-1} \pmod N = 1$.
  - $\gcd(a^{(N-1)/q}-1, N) = 1$.

The primality certificate, $C(N)$, for the integer N is as follows:

- If $N \leq L$ then $C(N) = C_0(N)$.
- If $N > L$ then $C(N) =$ "Proof(N), $C(q)$", where Proof(N) = (N, q, a).

8 Prime number generation

This clause specifies two different ways of generating a prime. The first type of method is based on probabilistic primality tests. The primes generated by these methods may optionally be certified by subsequently generating an elliptic curve certificate. The second type of method is based on deterministic generating methods. It includes Maurer's algorithm, which generates a prime number together with a primality certificate for that number.

8.1 Requirements

In order for a prime number generation algorithm to be compliant with this International Standard, the algorithm must be capable of generating output numbers of a given bit length k. Secondly, it must ensure that the error probability is at most 2^{-100}. Thirdly, the entropy of the output number should be at least B bits and also at least $\beta k^{1/3}$ bits. The absolute bound B and the relative bound β. can be set depending on the security requirements of the application.

NOTE For applications where the primes generated need to be kept secret, it is recommended to set B=100 and β =12. This ensures that it will be infeasible for an adversary to predict the output of the algorithm with any significant probability. The condition is true, for instance, if the output is chosen at random from a set of at least $2^{\beta k^{1/3}}$ k-bit primes. Furthermore, as primes are often generated using a pseudo random number generator based on a seed of a certain length, the uncertainty of the primes cannot be larger than that of the seed. In particular it is not harder to guess the primes (e.g., in the case of RSA) than to guess the seed. The bound on the entropy increases asymptotically in the same way as the expected complexity for factoring a modulus of size $2k$ by means of the number field sieve algorithm increases [13]. Using β = 12, the fixed bound meets the variable bound for $k \approx 512$ (i.e., for primes of 512 bits).

NOTE Annex A contains more information on how the properties of the algorithms can be established. With current mathematical state of the art, it is only possible to rigorously prove some of the properties. The rest can be shown based on well-established conjectures that have been extensively verified heuristically.

8.2 Using probabilistic tests

Let T denote any of the probabilistic primality tests given in Clause 6. The error probability of T depends on the number of iterations of the test.

NOTE Recommended values for the number of iterations for each of the probabilistic tests described in this International Standard are listed in Annex A.

The following subclauses describe two algorithms that meet the demands of Clause 8.1. It is not necessary to follow the example algorithms exactly in order to be in compliance with this International Standard; it is only necessary to implement an algorithm that is functionally equivalent to those algorithms given here.

8.2.1 Random Choice of Candidates

The following algorithm can be used assuming the test T is chosen appropriately:

1. Choose at random an odd number N such that $2^{k-1} < N < 2^k$.
2. If N passes T, output N and stop. Otherwise, go to step 1.

To estimate the probability that this algorithm outputs a composite, one must analyse how the test T behaves on randomly chosen inputs. This may not be the same as the maximal probability that T accepts a composite.

NOTE For instance, this worst-case probability is as mentioned in Annex A, ¼ for one iteration of the Miller-Rabin test, but in fact the probability is typically much smaller, i.e., a typical composite is accepted with much smaller probability than ¼. Thus the number of iterations depends on such an average-case analysis. Annex A contains information on how to choose the number of iterations.

NOTE The algorithm is guaranteed to produce uniformly chosen k-bit primes and therefore satisfies the general requirement on the output distribution.

8.2.2 Incremental search

The following algorithm may be used. Let μ denote a parameter limiting the number of increments when searching for a prime. The procedure is based on a different way of choosing the candidate primes and has some efficiency advantages in practice compared to the method based on random choices of candidates (Clause 8.2.1), when combined with test division:

1. Choose at random odd n_0 such that $2^{k-1} < n_0 < 2^k$. Set $N = n_0$ and $n_{max} = \min(2^k, n_0+2\mu)$. Execute the following procedure:
2. If N passes the test T, output N and stop.
3. Set $N = N+2$. If $N > n_{max}$ go to Step 1 else go to Step 2.

This algorithm chooses at random n_0 and then tests n_0, $n_0+2,\ldots,n_0+2\mu$ for primality. If no prime is found, a new random starting point n_0 is chosen and the procedure is repeated.

NOTE The error probability of this algorithm can be estimated using the same methods as for the algorithm choosing random candidates. The estimates will not be identical - they will depend both on the choice of T and μ. It is worth mentioning that due to the fact that the candidates in incremental search are not independent, error estimates tend to be weaker for this method. Note that a trade-off is involved in the choice of μ: a larger value of μ will decrease the probability that an execution of the inner loop outputs “fail”, and this tends to make the algorithm faster. On the other hand, the error estimates tend to be weaker for larger μ.

It is recommended to use the parameter $\mu = 10\ln(2^k)$. This results in a high probability of finding a prime in the first interval and, still, the distribution of the found prime is almost uniform.

NOTE This algorithm does not choose primes with exactly the uniform distribution among k-bits numbers. Due to the fact that primes are not regularly distributed; some primes will have larger probability of being chosen than others.

However, the output distribution will be almost uniform for proper choices of the parameter μ (see Annex A). In other words, the uncertainty about the prime produced by this method will be close to maximum, and therefore the incremental search method can also be considered to satisfy the demand on output distribution.

8.2.3 Primes with an elliptic curve primality certificate

A certified prime can be generated by first generating a prime using the methods described in Clause 8.2.1 and Clause 8.2.2, and then computing an elliptic curve primality certificate for that number, as described in Clause 7.1.1.

8.3 Using deterministic methods

8.3.1 Maurer's algorithm

Maurer's algorithm generates a random prime number N from scratch together with a proof that N is prime.

NOTE This method may be set up in such a way that the output number is an almost uniformly chosen prime from a given interval. Therefore the distribution of the resulting primes satisfies the requirements of Clause 8.1.

A prime, together with a primality certificate, is generated using the algorithm described below. This algorithm takes as input an integer k, the number of bits in the required prime. It returns a number, N, and a primality certificate, $C(N)$. The algorithms uses the following two parameters:

- L, the trial division bound.
- M is a parameter ensuring that primes of correct form exist.

The value M=20 is recommended.

The algorithm proceeds as follows:

1. If $2^k < L$ then generate a random odd k-bit integer, N, and test N for primality by trial division. Repeat until N is a prime. Output N and the certificate $C_0(N)$.
2. If $k \leq 2M$, set $r = ½$. Otherwise, repeat the following until $k - rk > M$.
    a. Select at random a real number s, $0 \leq s \leq 1$, and let $r = 2^{s-1}$.
3. Let $k_1 = \lfloor rk \rfloor + 1$. Call this algorithm recursively with input k_1, in order to determine a k_1-bits prime q and a certificate $C(q)$.
4. Set $t = 2^{k-1}/(2q)$.
5. Select a random integer R such that $t < R \leq 2t$ and let $N = 2Rq + 1$.
6. Select an integer a with $1 < a < N-1$. If
    - $a^{N-1} \pmod N = 1$ and
    - $\gcd(a^{2R}-1, N) = 1$

    then output N and the certificate (N, q, a), $C(q)$.
7. Otherwise repeat the steps 5-7.

NOTE The number of elements R, $t < R \leq 2t$, is lower bounded by 2^M. M must be chosen such that the algorithm has a reasonable chance of finding a prime of suitable form among these elements. The choice of t ensures that N is a k-bits integer.

NOTE The test on k in Step 2 can be avoided by choosing $L > 2^{2M}$.

NOTE The performance of Step 5 can be improved by performing trial division with primes less than a certain limit (the precise limit depends on the performance of the division) and possibly one execution of the Miller-Rabin test with base 2 and selecting a new R if the trial division or the Miller-Rabin test decide that N is composite.

NOTE More information on Maurer's algorithm can be found in [3].

8.3.2 Shawe-Taylor's algorithm

Shawe-Taylor's algorithm generates a random prime number N from scratch. It can be implemented using the algorithm described below. This algorithm takes as input an integer k, the number of bits in the required prime. It returns a number, N. The algorithm uses one parameter: L, the trial division bound.

The algorithm proceeds as follows:

1. If $2^k < L$ then generate a random odd k-bit integer, N, and test N for primality by trial division. Repeat until N is a prime. Output N and stop.
2. If k is odd, let $k_1 = (k+3)/2$. If k is even, let $k_1 = k/2 + 1$. Call this algorithm recursively with input k_1, in order to determine a k_1-bits prime q.
3. Select a random integer number x, $2^{k-1} \leq x < 2^k$.
4. Let t be the least integer greater than $x/(2q)$.
5. If $2tq + 1 \geq 2^k$, then let t be the least integer greater than $2^{k-1}/(2q)$.
6. Let $N = 2tq + 1$.
7. Select a random integer a such that $1 < a < N-1$ and let $x = a^{2t} \bmod N$. If
   - $x \neq 1$
   - $gcd(x-1,N) = 1$
   - $x^q = 1 \bmod N$

   then output N.
8. Otherwise let $t = t + 1$ and repeat steps 5-8.

NOTE More information on Shawe-Taylor's algorithm is given in [14]. The algorithm listed in this International Standard has been taken from [15].

9 Candidate prime testing

A given number can be tested for primality by different means:

- If a primality certificate is given (such as those described in Clause 7.1 and 7.2) then the primality certificate must be verified; or
- If no primality certificate is given, then there are two possibilities: first one could construct a certificate for the candidate number, and second one could verify the candidate number using a probabilistic test.

If one wishes to verify the primality of a number using a primality certificate when one is not given, then one should try to construct an elliptic curve primality certificate. The given number is a prime if and only if the primality certificate can be constructed.

NOTE The primality certificate in Clause 7.2 (based on Maurer's algorithm) cannot in general be used here, as this certificate can only be efficiently constructed at the same time as the prime is generated.

If one uses a probabilistic primality test to verify a candidate number, then it is important to apply the right error estimates for the test. When generating a prime one has control over the distribution of the numbers. This allows for improved error estimates. When testing a given number for primality one has no information on how that number was chosen. To be on the safe side it is therefore necessary to anticipate that the number was chosen in order to make the test fail, and therefore worst case probabilities must be applied.

Annex A
(informative)

Error probabilities

For any probabilistic test, the number of iterations that needs 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 may have been chosen to have special properties.

For the Miller-Rabin test and the Frobenius-Grantham test, worst-case error estimates are given for the probability that the test accepts a given composite input number. For the Lehmann test, estimates are given only for the probability of accepting a composite number, although there is also a non-zero probability of rejecting a prime.

For each of the tests, it is reasonable to assume that when using it to test a number that is large and randomly chosen, say a random odd k bit number, the error probability would typically be much smaller. For the Miller-Rabin test, this intuition can be quantified, and good average-case error estimates can be given. This implies directly estimates of the error probabilities of the recommended algorithms. Tables of such estimates are given below. For the Frobenius-Grantham test and the Lehmann test, only a worst-case error estimate is known. Therefore, we first give a general formula that allows translating worst-case estimates into average-case estimates for any primality test.

In general, the estimates depend on two parameters:

- k, the bit length of the numbers generated and
- t, the maximal number of times the test is iterated on each candidate.

Recall that two algorithms were recommended for prime number generation: the random choice of candidates (see Clause 8.2.1) algorithm and the incremental search for candidates (see Clause 8.2.2) algorithm. Error estimates for both are given below. The incremental search algorithm actually depends on an additional parameter μ and the error estimates are given for the recommended value $\mu = 10\ln(2^k)$.

A.1 General estimates

Take any test T that accepts any prime and accepts a composite with a maximum probability of ε in a single iteration. Even if this is all we know about the test, we can still give estimates for its average case behaviour.

The probability that the random choice algorithm using T as primality test outputs a composite is at most (cf. e.g. [13]):

$$\varepsilon^t/(\varepsilon^t + 2.8/(k\text{-}2.8))$$

For the incremental search algorithm, let $q_{k,t,\mu}$ be the probability of outputting a composite starting from a single starting point, when we:

- Use t iterations of the test T,
- Choose the starting point as a random odd k-bit number, and
- Test a maximum of μ candidates.

Then the error probability for the overall algorithm, where we keep choosing a new starting point until we succeed, is at most [3],[6]:

$$k^2 q_{k,t,\mu} + (1\text{-}2.8/k)^{k^2}$$

Considering search in an interval, where μ candidates are tested, the worst-case scenario is when all candidates are composite. Considering this case, one can see that $q_{k,t,\mu} \leq 1\text{-}(1\text{-}\varepsilon^t)^\mu$.

A.2 Worst-case error estimate for the Miller-Rabin primality test

The probability that a composite number is accepted is at most $(¼)^t$ [7].

A.3 Average-case error estimates for the Miller-Rabin primality test

Table A.1 contains estimates of the base-2 logarithm of the average-case error probability of the Miller-Rabin test. For instance, in the first sub-table, the entry for k = 256 and t = 11 is −84, showing that the error probability for 11 Miller-Rabin tests of a 256-bit number is 2^{-84}. The second sub-table shows that 4 Miller-Rabin tests of a 1024-bit number limit the error probability to 2^{-109}. The entries are derived from the work in [4], [5].

Table A.1 — Average-case error estimates for the Miller-Rabin primality test.

k\t	**10**	**11**	**12**	**13**	**14**	**15**	**16**
256	-81	-84	-88	-92	-95	-98	-101

k\t	**2**	**3**	**4**	**5**	**6**	**7**
512	-44	-61	-72	-82	-91	-100
1024	-72	-93	-109	-124		
2048	-102	-139	-162	-182		

Usually, implementations of the Miller-Rabin test are fastest when using base 2. For performance reasons it may therefore be desirable to start with base 2. However, the average case error estimates for t iterations of the test assume that every iteration uses a random base. Therefore, if one first uses base 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.

A.4 Worst-case error estimate for the Frobenius-Grantham primality test

The probability that a composite number is accepted is at most $(1/7710)^t$ [9].

A.5 Average-case error estimates for the Frobenius-Grantham primality test

There are no results on the average-case error estimates known, but the general formula given in Annex A.1 can be used to translate the worst-case estimates into average-case estimates using ε = 1/7710. Table A.2 contains estimates of the base-2 logarithm of the error probabilities.

Table A.2 — Average-case error estimates for the Frobenius-Grantham primality test.

k\t	9	12	17
256	-109.7	-148.5	-213.0
512	-108.7	-147.4	-212.0
1024	-107.7	-146.4	-211.0
2048	-106.7	-145.4	-210.0

A.6 Worst-case error estimate for the Lehmann primality test

The probability that a composite number is accepted, is at most 2^{-t}. Additionally, there is a non-zero probability that a prime number is rejected.

A.7 Average-case error estimate for the Lehmann primality test

There are no results on the average-case error estimates known, but the general formula given in Annex A.1 can be used to translate the worst-case estimates into average-case estimates using $\varepsilon = 1/2$.