RSNI3

Rancangan Standar Nasional Indonesia 3

RSNI3 ISO/IEC 18032:2020
(Ditetapkan oleh BSN tahun 2024)

Keamanan informasi — Pembangkitan bilangan prima

Information security — Prime number generation

(ISO/IEC 18032:2020, IDT)

Pengguna dari RSNI ini diminta untuk menginformasikan adanya hak paten dalam dokumen ini, bila diketahui, serta memberikan informasi pendukung lainnya (pemilik paten, bagian yang terkena paten, alamat pemberi paten dan lain-lain)

ICS 35.030

BADAN STANDARDISASI NASIONAL

© ISO/IEC 2020 – All rights reserved
© BSN 2024 untuk kepentingan adopsi standar © ISO/IEC menjadi SNI – Semua hak dilindungi

Hak cipta dilindungi undang-undang. Dilarang mengumumkan dan memperbanyak sebagian atau seluruh isi dokumen ini dengan cara dan dalam bentuk apapun serta dilarang mendistribusikan dokumen ini baik secara elektronik maupun tercetak tanpa izin tertulis BSN

BSN
Email: dokinfo@bsn.go.id
www.bsn.go.id

Diterbitkan di Jakarta

Daftar isi

Prakata

SNI ISO/IEC 18032:2020, *Keamanan informasi — Pembangkitan bilangan prima*, merupakan standar yang disusun dengan jalur adopsi tingkat keselarasan identik dari ISO/IEC 18032:2020 *Information security — Prime number generation*, dengan metode adopsi terjemahan dua bahasa dan ditetapkan oleh BSN Tahun 2024.

Standar ini disusun oleh Komite Teknis 35-04, Keamanan Informasi, Keamanan Siber, dan Perlindungan Privasi. Standar ini telah dibahas melalui rapat teknis dan disepakati dalam rapat konsensus pada tanggal 21 Juni 2024 di Depok, yang dihadiri oleh para pemangku kepentingan (*stakeholder*) terkait yaitu perwakilan dari pemerintah, pelaku usaha, konsumen, dan pakar. Standar ini telah melalui tahap jajak pendapat pada tanggal 18 Juli 2024 sampai dengan 1 Agustus 2024 dengan hasil akhir disetujui menjadi SNI.

Kosakata yang digunakan dalam Standar ini mengikuti bentuk baku yang dicantumkan dalam Kamus Besar Bahasa Indonesia (KBBI), tetapi ada beberapa kosakata yang belum ada di dalam KBBI.

Kata/istilah "*seed", "trial division"*, "*binned", "side-channel", "sieve",* dan *"sieving*" tidak diterjemahkan dalam Standar ini karena Komite Teknis 35-04 belum menemukan padanan kata/istilah yang sesuai dengan konteks dalam Bahasa Indonesia.

Pada dokumen ini terdapat kesalahan penomoran pada bagian D.5 butir f) dimana terdapat nomor 8.5 yang seharusnya adalah satu bagian dengan butir f).

Apabila pengguna menemukan keraguan dalam Standar ini, maka disarankan untuk melihat standar aslinya yaitu ISO/IEC 18032:2020, dan/atau dokumen terkait lain yang menyertainya.

Perlu diperhatikan bahwa kemungkinan beberapa unsur dari Standar ini dapat berupa hak kekayaan intelektual (HAKI). Namun selama proses perumusan SNI, Badan Standardisasi Nasional telah memperhatikan penyelesaian terhadap kemungkinan adanya HAKI terkait substansi SNI. Apabila setelah penetapan SNI masih terdapat permasalahan terkait HAKI, Badan Standardisasi Nasional tidak bertanggung jawab mengenai bukti, validitas, dan ruang lingkup dari HAKI tersebut.

Keamanan informasi — Pembangkitan bilangan prima

1 Ruang lingkup

Dokumen ini menspesifikasikan metode untuk pembangkitan dan pengetesan bilangan prima seperti yang disyaratkan dalam protokol dan algoritma kriptografis.

Pertama, dokumen ini menspesifikasikan metode untuk pengetesan apakah suatu bilangan yang diberikan adalah prima. Metode pengetesan yang disertakan dalam dokumen ini terbagi menjadi dua kelompok:

— tes primalitas probabilistik, yang memiliki probabilitas galat kecil. Semua tes probabilistik yang dijelaskan di sini dapat mendeklarasikan suatu komposit sebagai bilangan prima;

— metode deterministik, yang dijamin memberikan keputusan yang tepat. Metode ini menggunakan apa yang disebut sertifikat primalitas.

Kedua, dokumen ini menspesifikasikan metode untuk membangkitkan bilangan prima. Sekali lagi, metode probabilistik dan deterministik disajikan.

CATATAN Ada kemungkinan bahwa pembaca dengan latar belakang teori algoritma sudah pernah mengenal algoritma probabilistik dan deterministik sebelumnya. Metode deterministik dalam dokumen ini secara internal masih menggunakan bit acak (yang dibangkitkan melalui metode yang dijelaskan dalam ISO/IEC 18031), dan “deterministik” hanya mengacu pada fakta bahwa keluarannya benar dengan probabilitas satu.

Lampiran A memberikan probabilitas galat yang digunakan oleh tes primalitas Miller-Rabin.

Lampiran B menjelaskan varian metode untuk membangkitkan bilangan prima sehingga persyaratan kriptografi tertentu dapat dipenuhi.

Lampiran C mendefinisikan primitif yang digunakan oleh metode pembangkitan dan verifikasi bilangan prima .

2 Acuan normatif

Dokumen-dokumen berikut diacu dalam teks sedemikian rupa sehingga beberapa atau semua isinya merupakan persyaratan dokumen ini. Untuk acuan bertanggal, hanya edisi yang dikutip yang berlaku. Untuk acuan yang tidak bertanggal, berlaku edisi terakhir dari dokumen acuan (termasuk setiap amandemen).

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

3 Istilah dan definisi

Untuk tujuan dokumen ini, istilah dan definisi berikut berlaku.

ISO dan IEC memelihara basis data terminologis untuk penggunaan dalam standardisasi di alamat berikut:

— ISO *Online Browsing* : tersedia di https://www.iso.org/obp

— IEC Electropedia: tersedia di https://www.electropedia.org/

3.1
bilangan komposit
komposit
integer yang memiliki pembagi yang bukan merupakan pembagi trivial (3.8)

3.2
pembangkit bit acak deterministik
DRBG
pembangkit bit acak yang menghasilkan sekuens bit yang muncul tampak acak dengan menerapkan algoritma deterministik ke nilai awal acak yang sesuai yang disebut *seed* dan, mungkin, beberapa input sekunder yang tidak bergantung pada keamanan pembangkit bit acak

3.3
entropi
ukuran ketidakteraturan, keacakan atau variabilitas dalam sistem tertutup

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

3.4
simbol Jacobi
simbol Jacobi dari integer positif a terhadap integer ganjil n
hasil kali simbol Legendre a terhadap faktor prima dari n, termasuk multiplisitasnya (3.5)

Catatan 1 untuk entri: Jika faktor prima p terjadi dengan multiplisitas $m \geq 1$ pada faktorisasi n, maka simbol Legendre dari a terhadap p terjadi dengan multiplisitas m pada hasil kali yang menghasilkan simbol Jacobi dari a terhadap n .

Catatan 2 untuk entri: Simbol Legendre dari integer positif a terhadap bilangan prima *p* adalah nilai $a^{(p-1)/2} mod\ p$

3.5
multiplicity
multiplisitasnya pembagi prima *p* dari *n*
integer positif terbesar e dengan p^e membagi *n*

3.6
sertifikat primalitas
bukti matematis bahwa suatu integer yang diberikan memang suatu bilangan prima

Catatan 1 untuk entri: Untuk integer kecil, primalitas paling efisien dibuktikan dengan *trial division*. Dalam kasus ini, sertifikat primalitas karenanya bisa jadi kosong.

3.7
bilangan prima
prima
integer positif yang hanya mempunyai pembagi trivial (3.8)

3.8
pembagi trivial
pembagi trivial suatu integer nonzero *N*
1, -1, N, dan $-N$

Catatan 1 untuk entri: Integer nonzero N apa pun dapat dibagi (setidaknya) 1, -1, N, dan $-N$.

4 Simbol dan singkatan istilah

$a \operatorname{div} n$	*for integers a and n, with $n \neq 0$, $a \operatorname{div} n$ is the unique integer s satisfying $a = s \cdot n + r$ where $0 \leq r < n$.* (untuk integer a dan n, dengan $n \neq 0$, $a \operatorname{div} n$ adalah integer unik s yang memenuhi $a = s \cdot n + r$ dimana $0 \leq r < n$.)
$a \bmod n$	*for integers a and n, with $n \neq 0$, a* mod n *is the unique non-negative integer $r$satisfying $a = (a \operatorname{div} n) \cdot n + r$.* (untuk integer a dan n, dengan $n \neq 0$, $a \bmod n$ adalah integer non-negatif unik r yang memenuhi $a = (a \operatorname{div} n) \cdot n + r$)
C	*primality certificate* (sertifikat primalitas)
$C(N)$	*primality certificate for the number N.* (sertifikat primalitas untuk bilangan N)
$C_0(N)$	*empty primality certificate, indicating that trial division should be used to verify that N is a prime* (sertifikat primalitas kosong, menunjukkan bahwa *trial division* sebaiknya digunakan untuk memverifikasi bahwa N adalah bilangan prima)
$\exp(c)$	*natural exponential function evaluated at c , i.e. $e^c$where $e \approx 2{,}71828$* (fungsi eksponensial natural dievaluasi pada c, yaitu e^cdimana $e \approx 2{,}71828$)
gcd	*greatest common divisor* (pembagi persekutuan terbesar)
Jacobi(a, N)	*Jacobi symbol for an integer a with respect to a nonzero odd integer N* (Simbol Jacobi untuk integer a terhadap integer ganjil nonzero N)
k	jumlah bit which dalam N
L	batas di bawah mana primalitas diverifikasi oleh *trial division*
Lucas(D, K, N)	K^{th} element of the Lucas sequence modulo N with discriminant D (elemen ke-K dari sekuens Lucas modulo N dengan diskriminan D)
$\ln(a)$	*natural logarithm of a with respect to the base e ≈ 2,718 28* (logaritma natural a terhadap basis e ≈ 2,718 28
$\log_b(a)$	logarithm of a with respect to base b (logaritma a terhadap basis b)
$\min\{a, b\}$	*minimum of the numbers a and b* (minimum dari bilangan a dan b)
N	*candidate number to be tested for primality, where N is always a positive, odd number* (kandidat bilangan yang akan dites primalitasnya, dimana N selalu bilangan ganjil positif

$\mathrm{sgn}(D)$ *sign of a number, i.e.* $\mathrm{sgn}(D) = 1$ *if* $D \geq 0$ *and -1 otherwise* (tanda suatu bilangan, yaitu $\mathrm{sgn}(D) = 1$ jika $D \geq 0$ dan -1 sebaliknya.)

T *(probabilistic) test for primality* (tes (probabilistik) untuk primalitas)

Z_N *the set of the integers* $0, 1, 2, \ldots, N-1$, *representing the ring of integers modulo N* (himpunan integer $0, 1, 2, \ldots, N-1$, merepresentasikan ring integer 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)* (himpunan bagian dari Z_N yang memuat bilangan-bilangan yang mempunyai modulo invers perkalian N (misalnya, jika N bilangan prima, $Z_N{}^*$ terdiri dari integer 1, 2, ..., N-1)

β *parameter that determines the lower bound of the entropy of the output of a prime generation algorithm* (parameter yang menentukan batas bawah entropi keluaran algoritma pembangkitan bilangan prima)

M *maximal number of steps in an incremental search for a prime* (jumlah langkah maksimal dalam pencarian inkremental bilangan prima)

$\lfloor x \rfloor$ *largest integer smaller than or equal to* x *(*integer terbesar yang lebih kecil atau sama dengan x)

$\lceil x \rceil$ *smallest integer greater than or equal to* x*. (*integer terkecil yang lebih besar atau sama dengan x)

$\sqrt{n}$ *principal (i.e. non-negative) square root of a non-negative number* n *(*akar kuadrat utama (yaitu non-negatif) dari bilangan non-negatif n

5 *Trial division*

Primalitas suatu integer N dapat dibuktikan melalui *trial division*. Hal ini harus dilakukan dengan cara berikut:

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

   1) *if* $N \bmod p = 0$ *then return* "N *composite*" *and stop*;

b) *return* "N prime" *and stop*.

Untuk integer kecil N, *trial division* lebih murah secara komputasi dibandingkan tes primalitas lainnya. Implementasi dari tes primalitas apa pun yang dijelaskan dalam dokumen ini dapat mendefinisikan batasan *trial division* L, yang terikat di bawah *trial division* yang digunakan untuk membuktikan primalitas bilangan bulat. Dokumen ini tidak mengeset nilai untuk L kecuali batasan bawah, yaitu $L > 6$.

CATATAN 1 Diasumsikan bahwa himpunan bilangan prima di bawah ukuran tertentu telah diketahui. Salah satu cara praktis untuk mengimplementasikan pengetesan tersebut adalah dengan memiliki tabel beberapa bilangan prima pertama yang telah dihitung sebelumnya, melakukan *trial division* dengan

bilangan prima tersebut, dan kemudian coba membagi hanya dengan semua integer ganjil hingga akar kuadrat.

CATATAN 2 Ukuran integer yang *trial division*nya lebih murah secara komputasi dibandingkan tes primalitas lainnya bergantung pada pengetesan dan implementasinya. Nilai yang mungkin untuk L adalah $L = 10^{10}$.

CATATAN 3 Metode gcd *binned*, sebagaimana dijelaskan dalam ANSI X9.80-2010,[1] dapat lebih efisien daripada *trial division* untuk implementasi tertentu.

6. Tes primalitas probabilistik

6.1 Umum

Tes primalitas probabilistik mengambil integer ganjil positif N sebagai input dan mengembalikan "N diterima" atau "N komposit". Tes primalitas Miller-Rabin yang dijelaskan pada 6.3 akan selalu menghasilkan "N diterima" jika N adalah bilangan prima. Namun, jika N adalah bilangan komposit, maka sebuah contoh pengetesan dapat secara keliru mengembalikan "N diterima". Untuk mengurangi kemungkinan galat tersebut, seseorang biasanya melakukan beberapa iterasi pengetesan pada N, menggunakan pilihan berbeda untuk nilai acak yang digunakan.

Tes probabilistik dalam pasal ini harus hanya berlaku pada integer ganjil yang lebih besar atau sama dengan batasan *trial division* L. Jika $N < L$, maka *trial division* harus diterapkan untuk menentukan primalitas N.

6.2 Persyaratan

Agar suatu bilangan diterima sebagai (kemungkinan) prima, dokumen ini mensyaratkan probabilitas galat, yaitu peluang bilangan tersebut komposit, paling banyak 2^{-100} Hal ini memberikan keyakinan yang signifikan bahwa kandidat bilangan prima manapun yang dites primalitasnya yang memenuhi ambang batas ini memang merupakan bilangan prima. Batasan probabilitas 2^{-100} dicapai dengan mensyaratkan tes Miller-Rabin dalam jumlah yang memadai, bergantung pada cara bilangan tersebut dibangkitkan (lihat Lampiran A).

6.3 Tes primalitas Miller-Rabin

Tes primalitas Miller-Rabin berbasiskan observasi berikut. Andaikan N sebenarnya adalah bilangan prima ganjil dan r serta *s* adalah integer positif unik sedemikian hingga $N - 1 = 2^r s$, dengan s ganjil. Untuk setiap integer positif $b < N$, tepat satu dari tiga kondisi berikut akan terpenuhi:

— $b^s \bmod N = 1$

— $b^s \bmod N = N - 1$; atau

— $(b^s)^{2^i} \bmod N = N - 1$, untuk beberapa i dengan $0 < i < r$.

Secara ekuivalen, jika $N \geq 3$ adalah integer ganjil dan terdapat integer positif $b < N$ yang tidak memenuhi salah satu kondisi di atas, maka N adalah bilangan komposit. Tes primalitas probabilistik berbasis observasi ini harus diterapkan pada integer ganjil $N \geq L$, sebagai berikut.

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* (untuk 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*.

Kandidat N diterima sebagai (mungkin) prima pada konklusi proses pengetesan berulang jika dan hanya jika N lulus semua t *round* pengetesan primalitas Miller-Rabin (dengan setiap pilihan b memenuhi salah satu kondisi yang terasosiasi dengan primalitas). Proses pengetesan mengembalikan "N komposit" dan segera dihentikan jika, untuk beberapa pilihan b, tidak ada satupun kondisi pengetesan yang terpenuhi (lihat A.2 dan A.3 untuk menentukan jumlah iterasi yang disyaratkan oleh dokumen ini).

Integer b yang dibangkitkan pada langkah c) harus dibangkitkan menggunakan pembangkit bit acak yang memenuhi spesifikasi ISO/IEC 18031 dan dikonversi menjadi bilangan menggunakan metode konversi dalam Lampiran C atau ISO/IEC 18031. Untuk setiap iterasi, proses seleksi suatu nilai untuk b pada langkah c) dilakukan lagi.

CATATAN 1 Alasan mengapa basis b dibangkitkan secara acak ada dua. Pertama, estimasi rata-rata kasus galat diadopsi dari Acuan [9] dan digunakan dalam A.3 dengan asumsi b adalah acak. Kedua, untuk basis b yang diketahui, tidak sulit untuk membangun integer komposit yang akan lulus *round* Miller-Rabin sehubungan dengan basis itu.

CATATAN 2 Langkah f) hanya berlaku bila $r > 1$, yaitu jika $N \bmod 4 = 1$.

7 Metode verifikasi primalitas deterministik

7.1 Umum

Metode-metode verifikasi primalitas deterministik menggunakan sertifikat primalitas dalam rangka memverifikasi primalitas suatu bilangan yang diberikan. Pasal ini menspesifikasikan konten dari dua tipe sertifikat primalitas:

— sertifikat primalitas berbasis kurva eliptis

— sertifikat primalitas untuk bilangan prima yang dibangkitkan menurut algoritma Shawe-Taylor (lihat 8.4.2).

Sertifikat primalitas berisi informasi yang memungkinkan verifikasi efisien bahwa suatu bilangan yang diberikan adalah prima. Untuk kedua tipe sertifikat yang dijelaskan dalam dokumen ini, bilangan kecil (yaitu bilangan yang lebih kecil dari batas *trial division* L) harus diverifikasi menjadi bilangan prima dengan *trial division*. Misalkan C_0 melambangkan sertifikat primalitas kosong untuk bilangan tersebut.

Suatu sertifikat primalitas kurva eliptis dapat dilakukan komputasi dengan sebarang bilangan prima yang diberikan. Oleh karena itu, metode komputasi sertifikat ini dapat digunakan untuk memverifikasi primalitas. Sertifikat primalitas yang diperoleh pada algoritma Shawe-Taylor dibangkitkan sebagai bagian dari proses membangkitkan bilangan prima, dan tidak dapat dihitung secara efisien untuk bilangan prima sebarang (setelah bilangan prima dibangkitkan).

7.2 Algoritma pembuktian primalitas kurva eliptis

7.2.1 Umum

Informasi mengenai kurva eliptis dapat diperoleh dari ISO/IEC 15946-1.

7.2.2 Pembangkitan sertifikat primalitas kurva eliptis

Untuk membangkitkan sertifikat primalitas integer ganjil $N \geq L$, metode yang dijelaskan di bawah ini digunakan secara rekursif. Jika metode tersebut berhasil, total koleksi data yang dibangkitkan oleh metode tersebut disusun dalam sertifikat primalitas, $C(N)$. Memverifikasi $C(N)$, dan dengan demikian membuktikan bahwa N adalah bilangan prima, jauh lebih cepat daripada membangkitkan sertifikat tersebut.

Pada input integer $N \geq L$, dengan $\gcd(N, 6) = 1$, algoritma pembuktian primalitas kurva eliptis harus dimulai dengan inisialisasi berikut.

a) Set $S = \{ N \}$ (himpunan integer yang mensyaratkan sertifikat primalitas), dan set $Certs = \{ \}$ (sertifikat tanpa konten).

   Pada langkah-langkah berikut, berbagai tes primalitas diterapkan pada integer r yang dipilih dari himpunan S. Jika suatu tes (untuk sementara waktu) menerima primalitas r, tes tersebut akan mengembalikan sertifikat $C(r)$, dan, boleh jadi, sekumpulan kemungkinan prima $\{ q_i \}$, dalam hal ini sertifikat harus tidak digunakan untuk membuktikan bahwa *r* adalah bilangan prima, kecuali setiap q_i telah terbukti bilangan prima. Jika perlu, algoritma harus beralih secara rekursif, mencoba membangkitkan sertifikat primalitas untuk masing-masing bilangan yang kemungkinan prima. Dalam proses membangkitkan sertifikat tersebut, kemungkinan prima tambahan boleh ditambahkan ke himpunan integer yang mensyaratkan sertifikat. Algoritma harus berlanjut hingga tidak ada lagi kemungkinan prima untuk diproses atau pemrosesan dibatalkan karena beberapa integer yang memerlukan sertifikat ditentukan sebagai bilangan komposit.

b) Pilih nilai r dari S dan set $S = S - \{ r \}$ (yaitu menghapus elemen r dari S).

   1) Terapkan:

— tes primalitas Pocklington pada r (lihat D.2) dan, jika tes tersebut tidak meyakinkan, terapkan juga tes primalitas Lucas deterministik pada r (lihat D.4.2); atau

— tes Brillhart-Lehmer-Selfridge ke r (lihat D.5).

Dalam kedua kasus tersebut, ketika pengetesan dilakukan, biarkan kemungkinan prima $\geq L$ muncul (dengan jumlah besar) dalam faktorisasi parsial $r - 1$ dan/atau $r + 1$.

2) Jika pengetesan mengindikasikan r adalah komposit, kembalikan "r komposit" (bersama dengan nilai r) dan stop.

3) Jika pengetesan tidak meyakinkan, lanjutkan ke langkah c).

4) Jika pengetesan menunjukkan bahwa r adalah bilangan prima, maka gabungkan $C(r)$ dengan *Certs*, dengan $C(r)$ adalah sertifikat untuk r yang dikembalikan oleh berhasilnya tes tersebut, dan himpunan $S = S \cup \{q_i\}$, dengan $\{q_i\}$ adalah himpunan kemungkinan bilangan prima (jika ada) yang muncul dalam faktorisasi $r - 1$ dan/atau $r + 1$ (yang mana pun yang digunakan dalam pengetesan); jika S tidak kosong, ulangi langkah b). Jika tidak, kembalikan "N prima" bersama dengan $C(N) = Certs$, dan stop.

c) Bangkitkan kurva eliptis E yang diberikan oleh $y^2 = x^3 + ax + b$ untuk beberapa integer a, b.

1) Terapkan tes primalitas kurva eliptis terhadap r (lihat D.6). Saat pengetesan dilakukan, biarkan kemungkinan bilangan prima $\geq L$ muncul (dengan multiplisitas) dalam faktorisasi parsial orde E_r (kurva E direduksi modulo r).

2) Jika tes tersebut mengindikasikan bahwa r adalah komposit, kembalikan "r komposit" bersama dengan nilai r dan stop.

3) Jika tes tersebut tidak meyakinkan, ulangi langkah c) dengan pilihan kurva eliptis yang baru.

4) Jika tes tersebut mengindikasikan bahwa *r* adalah bilangan prima, maka gabungkan $C(r)$ ke $Certs$, yang mana $C(r)$ adalah sertifikat untuk r yang dikembalikan oleh berhasilnya tes tersebut, dan himpunan $S = S \cup \{q_i\}$, dengan $\{q_i\}$ adalah himpunan kemungkinan bilangan prima (jika ada) yang muncul dalam faktorisasi orde E_r; jika S tidak kosong, lanjutkan ke langkah b). Jika tidak, kembalikan "N adalah bilangan prima" bersama dengan $C(N) = Certs$, dan stop.

Faktor kemungkinan prima (jika ada) yang muncul pada langkah b) dan c) disyaratkan lebih besar atau sama dengan batas *trial division*. Primalitas dari faktor yang lebih kecil harus dipastikan dengan menggunakan *trial division*.

Selama rekursi, tes pada langkah b) dapat dilewati. Hal tersebut disertakan dalam deskripsi algoritma untuk tujuan efisiensi.

Jika primalitas N tidak meyakinkan, tes dapat dijalankan kembali. Tes tersebut tidak mungkin dijalankan kembali secara penuh jika hasil parsial pengerjaan sebelumnya tertahan (misalnya, nilai *Certs* ketika proses pengetesan dihentikan dan nilai komposit r yang menyebabkan terminasi lebih awal). Cukup dengan memundurkan langkah rekursi satu tingkat sebelum titik

kegagalan. Memvariasikan pilihan kurva pada langkah c) dan/atau melewatkan langkah b) kemungkinan besar akan menghasilkan kumpulan bilangan prima yang berbeda untuk dites. Dalam hal r adalah komposit, tes kurva eliptis pada langkah c) kemungkinan besar menjadi tidak meyakinkan sehingga algoritma dapat gagal untuk dihentikan. Sebagai tindakan pencegahan, batasan jumlah iterasi pada langkah c) dapat diterapkan.

CATATAN Kurva eliptis E pada langkah c) dapat dibangkitkan dengan menggunakan metode CM yang dijelaskan dalam D.8.

7.2.3 Verifikasi sertifikat primalitas kurva eliptis

Sertifikat primalitas kurva eliptis untuk integer $N \geq L$, dengan $\gcd(N, 6) = 1$, adalah sekumpulan sertifikat (saling bergantung), $C(N) = \{C_i\}$, yang masing-masing C_i menegaskan primalitas beberapa integer $r_i \geq L$, dan salah satu r_i tepat sama dengan N. Setiap C_i merupakan sertifikat yang dihasilkan dari pelaksanaan tes Pocklington (D.2.2), tes deterministik Lucas (D.4.2), tes Brillhart-Selfridge-Lehmer (D.5) ataupun tes kurva eliptis (D.6).

Jika C_i menegaskan primalitas r_i berbasis (sebagian) asumsi primalitas dari satu atau lebih integer lainnya $q_j \geq L$, maka sertifikat C_i (dan karenanya primalitas r_i) harus hanya berhasil diverifikasi setelah primalitas masing-masing integer dari q_j tersebut diterima melalui verifikasi sertifikat mereka, yang juga harus termasuk dalam $C(N)$. Jika C_i menegaskan primalitas r_i berbasis (sebagian) asumsi primalitas integer yang kurang dari L, maka *trial division* harus digunakan untuk memverifikasi primalitas integer tersebut sebagai bagian dari verifikasi C_i.

Sertifikat primalitas kurva eliptis $C(N) = \{C_i\}$ diterima hanya jika setiap C_i diterima (dan salah satu r_i tepat sama dengan N). Jika verifikasi Ci gagal, maka sertifikat primalitas kurva eliptis untuk N harus ditolak.

Verifikasi masing-masing sertifikat C_i harus dilakukan sebagaimana dispesifikasikan dalam Lampiran D.

7.3 Sertifikat primalitas berbasis algoritma Shawe-Taylor

Jenis sertifikat primalitas C ini adalah kumpulan sertifikat $\{ C_i \}$ yang harus dihitung (hanya) selama proses pembangkitan bilangan prima menggunakan proses yang dijelaskan dalam 8.4.2 (Algoritma Shawe-Taylor). Sertifikat C_i merupakan hasil tes Pocklington (lihat D.2.2) yang pembuktian primalitasnya mempunyai struktur sebagai berikut:

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

dengan r_i, q_i, dan a_i adalah bilangan bulat. Sertifikat C_i, yang menyatakan bahwa r_i adalah bilangan prima, diverifikasi setelah melewati pemeriksaan di C.2 dan q_i terbukti sebagai bilangan prima (ganjil) (misalnya, dengan *trial division* atau dengan memverifikasi sertifikatnya itu sendiri, yang juga telah termasuk dalam C). Sertifikat terakhir yang diverifikasi, misalkan $C_1 = (N, q_i, a_i)$, berisi nilai N. Jika semua C_i diverifikasi (dan terkonfirmasi berantai hingga N), maka C itu sendiri terverifikasi dan N diterima sebagai bilangan prima.

8 Pembangkitan bilangan prima

8.1 Umum

Pasal ini menspesifikasikan dua pendekatan untuk pembangkitan bilangan prima. Pendekatan pertama adalah dengan menggunakan tes primalitas Miller-Rabin probabilistik pada kandidat

yang dipilih secara acak (8.3). [Secara opsional, kemungkinan prima, N, yang dibangkitkan dengan metode tersebut selanjutnya menjadi terbukti prima dengan membangkitkan sertifikat primalitas kurva eliptis (7.2) untuk N.] Pendekatan kedua untuk pembangkitan bilangan prima adalah dengan menggunakan metode deterministik Shawe-Taylor (8.4) yang menghasilkan integer yang diketahui sebagai bilangan prima. Metode ini juga dapat menghasilkan sertifikat primalitas sebagai bagian dari proses pembangkitan tersebut.

Metode dalam pasal ini membangkitkan bilangan prima pada interval $(2^{k-1}, 2^k)$ untuk beberapa k. Untuk membangkitkan bilangan prima dengan kendala tambahan, seperti membangkitkan bilangan prima dalam interval yang lebih terbatas dan/atau bilangan prima yang memenuhi kondisi kongruensi tertentu, metode pada B.2 harus digunakan. Contoh bilangan prima yang dibangkitkan dengan kendala tambahan diberikan dalam Lampiran E.

Teknik-teknik ini harus hanya diterapkan untuk membangkitkan bilangan prima yang lebih besar atau sama dengan batasan *trial division* L. Membangkitkan bilangan prima yang kurang dari L dapat dilakukan secara sederhana melalui pemilihan integer yang kurang dari L dan pengetesan primalitas menggunakan *trial division*.

8.2 Persyaratan

Agar algoritma pembangkitan bilangan prima sesuai dengan dokumen ini, algoritma tersebut harus:

— membangkitlkan bit acak menggunakan pembangkit bit acak deterministik yang memenuhi spesifikasi ISO/IEC 18031;

— membangkitkan bilangan acak dari sekuens bit acak menggunakan metode konversi yang dispesifikasikan dalam Lampiran C atau ISO/IEC 18031;

— memastikan, dalam kasus bilangan prima yang tidak dapat dibuktikan (8.3), bahwa probabilitas galat suatu komposit lulus sebagai bilangan prima paling banyak adalah 2^{-100}.

Lampiran A berisi informasi lebih lanjut tentang bagaimana algoritma di 8.3 harus memenuhi properti galat 2^{-100}. Untuk aplikasi yang bilangan prima yang dibangkitkannya perlu dirahasiakan, hal berikut juga berlaku:

— entropi (rahasia) yang digunakan dalam pembangkit bit acak untuk menghasilkan keluaran bilangan harus memenuhi persyaratan ISO/IEC 18031 dan harus paling sedikit berupa B bit dengan B adalah tingkat keamanan aplikasi tersebut;

— jika kendala tambahan ditempatkan pada bilangan prima yang dibangkitkan (B.2), kendala tersebut harus dibatasi sehingga tidak mempengaruhi keamanan sistem dan persyaratan dalam Lampiran B harus berlaku.

Pada kasus pembangkitan bilangan prima untuk digunakan dalam RSA, B sebaiknya 112 untuk bilangan prima 1024-bit, 128 untuk bilangan prima 1536-bit, 192 untuk bilangan prima 3840-bit, dan 256 untuk bilangan prima 7680-bit. ISO/IEC 18031 memerlukan entropi minimum 120 bit untuk menghindari kolisi. Pada kasus RSA, kolisi dapat mengarah ke pengulangan bilangan prima untuk modul RSA yang berbeda. Bilangan prima seperti itu dapat diperoleh kembali dengan menghitung gcd di antara modul RSA.

CATATAN Membatasi jumlah total kendala diperlukan untuk menghindari serangan gaya Coppersmith [7][14], misalnya serangan ini dapat memfaktorkan moduli RSA jika kira-kira setengah bit faktor prima diketahui. Jika kendalanya (seperti jumlah bit yang diketahui) terbatas, maka serangan ini tidak berlaku. Sebagai contoh, membutuhkan integer $n < 2^k$ terletak pada interval $(2^{k-2}, 2^{k-1}, 2^k)$ dengan $n \bmod 4 = 3$ dan $\gcd(n-1, e) = 1$ untuk beberapa eksponen enkripsi ganjil yang memberikan

kurang dari 5 bit kendala. Jika intervalnya diganti dengan $(2^{k-1}\sqrt{2}, 2^k)$, kendala totalnya kurang dari 4,5 bit. Membangkitkan bilangan k–bit n dengan $n \bmod q = 1$ untuk beberapa integer q berukuran m bit yang dibangkitkan secara acak (dan rahasia) menempatkan kira-kira dua bit kendala pada n (karena n dapat dinyatakan sebagai $n = 1 + u\,q$ dengan u, q keduanya adalah bilangan $k - m$, m-bit). Selanjutnya mensyaratkan q menjadi bilangan prima menambahkan beberapa bit kendala tambahan, kira-kira $\log_2(m) - 0{,}528$ bit berdasarkan teorema bilangan prima.

Saat membangkitkan kembali bilangan prima (rahasia) dari *seed* tetap, kehati-hatian sebaiknya diberikan untuk menghindari serangan *side-channel*.

8.3 Menggunakan tes primalitas Miller-Rabin

8.3.1 Umum

Probabilitas galat suatu komposit lulus sebagai bilangan prima bergantung pada jumlah iterasi yang digunakan dalam tes primalitas Miller-Rabin. Lampiran A harus diikuti untuk menentukan jumlah iterasi yang diperlukan.

Subpasal 8.3.2 dan 8.3.3 menjelaskan dua algoritma yang memenuhi persyaratan 8.2.

8.3.2 Pencarian acak

Algoritma berikut dapat digunakan untuk menghasilkan suatu bilangan prima k-bit.

a) Bangkitkan bilangan acak N sedemikian hingga $2^{k-1} < N < 2^k$. Jika N genap, tambah N sebanyak 1.
b) Terapkan tes primalitas Miller-Rabin. Jika diterima, kembalikan N dan stop. Jika tidak, lanjutkan ke langkah a).

8.3.3 Pencarian inkremental

Algoritma berikut dapat digunakan untuk membangkitkan *bilangan* prima *k*-bit. Algoritma tersebut berbeda dari algoritma yang diberikan di 8.3.2 dalam cara memilih kandidat untuk pengetesan primalitas. Jika kandidat N (ganjil) yang dipilih secara acak tidak diterima oleh tes primalitas Miller-Rabin, beberapa integer ganjil konsekutif berikut ($N + 2, N + 4$, dst.) dites sebelum memilih nilai lain untuk N secara acak. Hasilnya, algoritma ini dapat memiliki beberapa keunggulan praktis (dalam hal efisiensi implementasi) dibandingkan 8.3.2. Suatu parameter yang dilambangkan μ membatasi jumlah integer ganjil konsekutif yang dites antara pilihan acak N.

a) Bangkitkan bilangan acak N sedemikian hingga $2^{k-1} < N < 2^k$. Jika N genap, tambah N sebanyak 1.

b) Set $Max = \min\{\, 2^k - 1, N + 2\mu \,\}$ untuk beberapa nilai μ yang dipilih dengan tepat, misalnya, $\mu = 10 \ln(2^k)$

c) Jika N diterima oleh tes primalitas Miller-Rabin, kembalikan N dan stop.

d) Set $N = N + 2$.

e) Jika $N > Max$, lanjutkan ke langkah a). Jika tidak, lanjutkan ke langkah c).

Daripada menaikkan kandidat bilangan prima N pada langkah d) sebanyak 2, orang dapat menerapkan tes Miller-Rabin pada elemen-elemen dalam sekuens $N, N + 2, N + 4$, ..., Max yang bertahan melalui prosedur *sieving* yang dijelaskan dalam C.1. Proses *sieve* mengeliminasi kandidat bilangan prima yang habis dibagi bilangan-bilangan prima kecil yang digunakan dalam *sieve*.

CATATAN Nilai yang disarankan sebesar $10 \ln (2^k)$ untuk μ berbasis teorema Bilangan Prima yang memberikan perkiraan $2^k / ln(2^k)$ untuk jumlah bilangan prima kurang dari 2^k. Faktor ekstra 10 diberikan untuk meningkatkan kemungkinan bahwa bilangan prima terdapat dalam interval $[N, N + 2\mu]$. Secara khusus, Lemma 4 dari Acuan [5] menduga bilangan prima tersebut ada dengan probabilitas sekitar $1 - \exp(-2 \cdot c), c = 10$.

8.3.4 Prima dengan suatu sertifikat primalitas kurva eliptis

Suatu bilangan prima bersertifikat dapat dibangkitkan dengan terlebih dahulu membangkitkan bilangan prima menggunakan metode yang dijelaskan dalam 8.3.2 dan 8.3.3, dan kemudian menghitung sertifikat primalitas kurva eliptis untuk bilangan tersebut, seperti yang dijelaskan dalam 7.2.

8.4 Menggunakan metode deterministik

8.4.1 Umum

Metode deterministik dalam subpasal ini mengonstruksi bilangan terbukti prima dengan ukuran yang diinginkan dengan mengonstruksi bilangan terbukti prima yang lebih kecil yang dapat dibuktikan secara rekursif. Algoritma pembangkitan bilangan prima tersebut memanggil dirinya sendiri secara berulang kali untuk membangkitkan sepertiga ukuran (atau setengah ukuran) dari prima dan rekursi ini hanya stop ketika ukuran bilangan prima tersebut sedemikian hingga suatu kandidat bilangan prima acak dapat dibuktikan sebagai bilangan prima melalui *trial division*, setelah itu rekursi dibatalkan dengan mengonstruksi bilangan prima berukuran tiga kali lipat (atau berukuran dua kali lipat) yang terbukti merupakan bilangan prima berdasarkan konstruksi.

8.4.2 Algoritma Shawe-Taylor

Algoritma Shawe-Taylor membangkitkan bilangan prima acak N dari awal. Algoritma tersebut harus diimplementasikan menggunakan algoritma yang terdeskripsi. Algoritma ini mengambil integer k sebagai input, jumlah bit dalam bilangan prima yang disyaratkan. Algoritma mengembalikan bilangan N dan sertifikat primalitas jika diminta.

Algoritma ini disebut rekursif, sedemikian hingga untuk suatu panjang bit yang diberikan $j \geq \log_2(L)$, terdapat:

a) prima j'-bit q dengan $j = \lceil j/3 \rceil + 1$ (atau $\lceil j/2 \rceil + 1$) dan sertifikat primalitas $C(q)$ (yang mencakup sertifikat bilangan prima yang lebih kecil yang digunakan untuk mengonstruksi q) jika diminta.

dari mana bilangan prima j-bit p dibangun adalah sebagai berikut:

b) Pilih bilangan bulat x secara acak dari interval $(2^{j-1}, 2^j - 2q]$

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

d) Terapkan tes primalitas Pocklington terhadap p, dengan $p - 1 = F R$ serta $F = q, R = t$.

jika tes merespons "p prima";

respons p bersama dengan sertifikat $C(p)$ (jika sertifikat diminta) dan stop.

lain jika $p < 2j - 2q$;

set $p = p + 2q, t = t + 2$, dan lanjutkan ke langkah d).

lainnya lanjutkan ke langkah b).

Ketika $j < \log_2(L)$, j-bit bilangan prima q_0 harus dikonstruksi menggunakan *trial division*. Sertifikat C untuk k-bit bilangan prima N adalah koleksi sertifikat yang dibangkitkan pada langkah d) dan sertifikat *trial division* $C_0(q_0)$.

Algoritma ini dapat diimplementasikan dengan terlebih dahulu mengeset $j_n = k$ (untuk beberapa n), dan menghitung secara rekursif panjang bit $j_{i-1} = \lceil j_i/3 \rceil + 1$ (atau $\lceil j_i/2 \rceil + 1$) hingga $j_0 < \log_2(L)$, yang menjadi dasar j_0-bit bilangan prima q_0 dikonstruksi menggunakan *trial division*. Bilangan prima q_0 kemudian digunakan untuk mengonstruksi j_1-bit bilangan prima q_1 melalui langkah b) hingga d). Bilangan prima q_1 kemudian digunakan untuk mengonstruksi j_2-bit bilangan prima q_2, dan seterusnya hingga bilangan prima *k*-bit $N = q_n$ dibangkitkan.

Daripada menaikkan kandidat bilangan prima p pada langkah c) sebesar $2q$ [pada langkah d)], untuk beberapa J pada $((2^{j-1} - p) / 2q, (2^j - p) / 2q)$ tes Pocklington mungkin diterapkan pada elemen dalam sekuens $p, p + 2q, p + 4q, \ldots, p + 2Jq$ yang bertahan melalui prosedur *sieving* yang dijelaskan dalam C.1. Proses *sieving* mengeliminasi kandidat bilangan prima yang habis dibagi bilangan prima kecil yang digunakan dalam *sieve*.

CATATAN Penggunaan nilai alternatif $j' = \lceil j/2 \rceil + 1$, pada langkah a) ekuivalen dengan algoritma Shawe-Taylor dalam ISO/IEC 18032:2005 (dan diadaptasi dari Acuan [18]). Pilihan $j' = \lceil j/3 \rceil + 1$ memanfaatkan versi teorema Pocklington yang lebih umum yang diberikan di D.2, dan menghasilkan algoritma yang lebih efisien karena langkah rekursif yang lebih sedikit.

Lampiran A
(normatif)
Probabilitas galat

A.1 Umum

Untuk tes probabilistik apa pun, jumlah iterasi yang perlu dilakukan bergantung pada probabilitas galat dari satu iterasi tes tersebut. Probabilitas galat dari satu iterasi bervariasi bergantung pada apakah bilangan yang akan dites telah dipilih secara acak, atau apakah bilangan terpilih tersebut memiliki sifat khusus.

Untuk tes Miller-Rabin, estimasi galat kasus terburuk (A.2) diberikan untuk probabilitas bahwa aplikasi berulang-ulang dari tes tersebut menerima suatu bilangan input komposit yang diberikan. Estimasi galat kasus terburuk harus diterapkan kecuali dinyatakan lain dalam dokumen ini.

Untuk integer besar dan dipilih secara acak, estimasi galat kasus rata-rata yang baik dapat diberikan, yang secara signifikan mengurangi jumlah iterasi Miller-Rabin yang dilakukan. Estimasi galat pada A.3 hanya berlaku untuk metode pencarian acak pada 8.3.2. Dalam kasus metode pencarian inkremental 8.3.3, berlaku probabilitas galat yang berbeda[5]. Untuk memastikan metode pencarian inkremental mencapai probabilitas galat 2^{-100} yang disyaratkan, jumlah iterasi Miller-Rabin yang dilakukan harus ditingkatkan sebesar 1. Jadi, sebagai contoh, kandidat bilangan prima 1.024-bit mensyaratkan setidaknya 5 iterasi. Selanjutnya, jika jumlah iterasi Miller-Rabin direduksi untuk salah satu metode pencarian, kandidat bilangan prima tersebut juga sebaiknya lulus iterasi tunggal dari tes probabilistik Lucas (D.3) sebelum diterima sebagai (kemungkinan) prima. Karena, hingga saat ini, belum ada contoh komposit yang diketahui lulus iterasi tunggal dari Miller-Rabin (dengan basis/saksi (*witness*) $a = 2$) dan iterasi tunggal dari tes Lucas, yaitu bilangan prima semu Baillie-PSW[15].

Secara umum, estimasi tersebut bergantung pada dua parameter:

— k, panjang bit dari bilangan yang dibangkitkan; dan

— t, berapa kali tes diulang pada masing-masing kandidat

A.2 Estimasi galat kasus terburuk untuk *t* tes primalitas Miller-Rabin

Probabilitas suatu bilangan komposit diterima oleh tes t (independen) Miller-Rabin paling banyak $(¼)^t$ (lihat Acuan[9]). Jadi, untuk memastikan suatu probabilitas galat paling banyak 2^{-100}, t harus memenuhi $t \geq 50$.

A.3 Estimasi galat kasus rata-rata untuk *t* tes primalitas Miller-Rabin

Tabel A.1 dan Tabel A.2 berisi estimasi logaritma basis 2 dari probabilitas galat kasus rata-rata untuk t tes Miller-Rabin. Entri yang digarisbawahi sesuai dengan jumlah minimum tes Miller-Rabin independen disyaratkan untuk mencapai ambang batas 2^{-100}. Umpamanya, entri untuk k = 256 dan t = 16 adalah −101, menunjukkan bahwa probabilitas galat untuk 16 tes Miller-Rabin dari suatu bilangan 256-bit adalah 2^{-101}. Sub-tabel kedua menunjukkan bahwa 4 tes Miller-Rabin dari suatu bilangan 1024-bit membatasi probabilitas galat menjadi 2^{-109}. Entri tersebut diturunkan dari formula yang disediakan dalam Lampiran F Acuan [10] berbasis hasil dalam Acuan [5] dan [9].

Tabel A.1 — Estimasi galat kasus rata-rata untuk *t* tes primalitas Miller-Rabin (*k* = 256)

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

Tabel A.2 — Estimasi galat kasus rata-rata untuk tes primalitas Miller-Rabin ($k \geq 512$)

*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

Estimasi galat kasus rata-rata untuk t iterasi dari tes tersebut mengasumsikan bahwa setiap iterasi menggunakan basis acak. Oleh karena itu, jika seseorang pertama kali menggunakan basis tetap, misalnya a = 2, dan kemudian t-1 basis acak, maka probabilitas galat yang tercantum di sini sebaiknya digunakan seolah-olah hanya t-1 iterasi yang telah dilakukan.

Untuk panjang bit 4.096 atau lebih besar, t = 1 memenuhi batas probabilitas galat 2^{-100}. Namun, implementasinya dapat menginginkan untuk mempertimbangkan penggunaan *round* tambahan sebagai tindakan pencegahan karena kondisi yang disyaratkan untuk penggunaan tabel ini tidak dipenuhi secara ketat.

Secara umum, menerapkan *round* tambahan tes primalitas Miller-Rabin diperbolehkan. Sebagai contoh, aplikasi tertentu boleh menginginkan estimasi galat kasus rata-rata sesuai dengan tingkat keamanan yang diinginkan untuk menggunakan bilangan prima. Selanjutnya, untuk tingkat keamanan tetap ukuran bilangan prima dapat berbeda bergantung pada algoritma kunci publik. Sebagai contoh, bilangan prima 3.072-bit p yang digunakan dalam Diffie-Hellman (atas suatu *field* prima) menawarkan keamanan sekitar 128 bit, sedangkan faktor prima untuk modulus RSA 3.072-bit adalah bilangan 1.536-bit. Jadi, dalam kasus Diffie-Hellman, akan mensyaratkan 3 *round* pengetesan primalitas Miller-Rabin untuk mencocokkan tingkat keamanan 128 bit, dan akan mensyaratkan 4 *round* untuk masing-masing faktor prima modulus RSA.

CATATAN Probabilitas kasus rata-rata menurun seiring dengan bertambahnya panjang bit bilangan prima.

Lampiran B
(normatif)
Membangkitkan bilangan prima dengan kondisi sampingan

B.1 Umum

Dalam beberapa kasus, diperlukan untuk membangkitkan bilangan prima yang memenuhi kondisi ekstra tertentu. Sebagai contoh:

— dalam aplikasi kriptografis tertentu, bisa jadi diperlukan bahwa bilangan prima yang dibangkitkan harus kongruen dengan 3 modulus 4. Sebagai contoh, dalam protokol identifikasi Feige-Fiat-Shamir, modulusnya adalah hasil kali dua bilangan prima yang tunduk pada pembatasan ini. Selanjutnya, dalam aplikasi RSA apa pun yang menggunakan eksponen (publik) e, hanya bilangan prima p dengan $\gcd(p-1, e) = 1$ yang harus digunakan sebagai faktor modulus RSA (karena e sebaiknya invertibel dalam Z_N);

— bilangan prima tersebut sebaiknya berada dalam interval tertentu (misalnya, untuk RSA, bilangan prima tersebut mungkin tidak cukup hanya terdiri dari k bit, tetapi hasil perkaliannya dengan bilangan prima kedua sebaiknya menghasilkan bilangan $2k$-bit).

Dalam konstruksi yang dijelaskan di bawah, integer x dibangkitkan dengan tunduk pada kondisi kongruensi dan/atau kondisi ukuran tertentu. Jika x dibangkitkan secara acak berdasarkan kendala ini dan dites primalitasnya menggunakan tes primalitas Miller-Rabin, dokumen ini memperbolehkan pengurangan jumlah iterasi yang diberikan pada Tabel A.2 ditambah satu. Bilangan kemungkinan prima tersebut juga sebaiknya lulus iterasi tunggal dari tes probabilistik Lucas, D.3. Artinya, jumlah iterasi yang sama yang diperbolehkan untuk metode pencarian inkremental 8.3.3 berlaku. Hal tersebut mengikuti teorema Dirichlet[13] bahwa bilangan prima didistribusikan secara merata di seluruh kelas kongruensi (yang koprima terhadap modulus tersebut), sehingga kendala tambahan ini tidak akan memengaruhi analisis kasus rata-rata yang digunakan di A.3.

Perlu diperhatikan bahwa modulus kongruensi tidak terlalu besar dan/atau ukuran intervalnya terlalu kecil. Jika bilangan prima yang dibangkitkan untuk digunakan dalam RSA, jumlah total bit kendala yang dihasilkan dari pembatasan ukuran interval dan/atau kondisi kongruensi harus tidak melebihi 20 untuk menghindari serangan gaya Coppersmith (lihat CATATAN pada 8.2). Sebagai contoh, mensyaratkan bilangan $n < 2^k$ terletak pada interval $(2^{k-2} + 2^{k-1}, 2^k)$ dan memenuhi $n \bmod 4 = 3$ menghasilkan 4 bit kendala: ukuran interval mensyaratkan dua bit teratas dari n harus diset dan kondisi kongruensinya mensyaratkan dua bit signifikan terkecil untuk diset.

Bilangan acak harus dikonstruksi menggunakan pembangkit bit acak deterministik yang memenuhi persyaratan ISO/IEC 18031 dan suatu metode konversi (dari sekuens bit menjadi bilangan) dalam Lampiran C atau ISO/IEC 18031.

B.2 Restriksi kongruensi pada bilangan prima

B.2.1 Umum

Dokumen ini menjelaskan pembangkitan bilangan prima berbasis kondisi kongruensi.

B.2.2 Restriksi kongruensi dan pencarian inkrimental atau acak

Misalkan m adalah modulus, e adalah integer ganjil positif, dan r adalah integer yang memenuhi $0 \leq r < m$. Diberikan tes primalitas T, suatu bilangan prima k-bit N–yang memenuhi $N \bmod m = r$ dan $\gcd(N-1, e) = 1$ harus dibangkitkan sebagai berikut.

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)* (pencarian inkrimental);

or

go to step b) (pencarian acak).

Daripada menaikkan kandidat bilangan prima *p* pada langkah e) sebanyak m, untuk beberapa $J < (2^k - p) / 2m$, *T* dapat diterapkan pada elemen dalam sekuens $p, p + 2m, p + 4m, \ldots, p + 2Jm$ yang lolos prosedur *sieving* yang dijelaskan dalam C.1. Proses *sieve* mengelimininasi kandidat bilangan prima yang habis dibagi bilangan prima kecil yang digunakan dalam *sieve* tersebut.

Pada kasus hanya kendala $N \bmod m = r$ yang diperlukan, maka langkah d) dapat dihilangkan. Jika hanya kendala $\gcd(N-1, e) = 1$ yang diperlukan, maka langkah a) dapat diterima pada pengesetan $m = 2$ dan $r = 1$.

CATATAN 1 Langkah e) memperbolehkan dua metode pencarian yang berbeda, bertambah sebesar *m* atau membangkitkan kandidat baru secara acak (tunduk pada kondisi kongruensi tetap).

CATATAN 2 Untuk membangkitkan bilangan prima p sehingga $p \bmod m$ terletak pada beberapa himpunan bagian S dari $\{1, 2, \ldots, m-1\}$, seseorang dapat lebih dulu memilih r acak di S dan kemudian menerapkan algoritma di atas untuk mengonstruksi bilangan prima p yang memenuhi $p \bmod m = r$.

CATATAN 3 Untuk modulus *m* ganjil, langkah a) diperlukan untuk memastikan integer p pada langkah c) ganjil.

CATATAN 4 Kehati-hatian perlu diberikan dalam mempertimbangkan pilihan e, r, dan m. Sebagai contoh, jika m dan r bilangan genap, kandidat bilangan prima *p* pada langkah c) selalu genap dan algoritma gagal mengembalikan bilangan prima. Demikian pula, nilai $e = 3$, $r = 1$, $m = 3$ mengakibatkan tidak ada bilangan prima yang dikembalikan karena kondisi $N \bmod 3 = 1$ memaksa $\gcd(N-1, e) = 3$.

Penerapan umum dari algoritma di atas mengikuti.

a) Jika T sama dengan tes primalitas Miller-Rabin, maka algoritma mengembalikan kemungkinan bilangan prima acak N tunduk pada kondisi bahwa $N \bmod m = r$. Kasus trivial $m = 2$ dan $r = 1$, memastikan kandidat bilangan prima bernilai p pada langkah c) adalah ganjil.

b) Kasus T sama dengan tes Pocklington (agar m = bilangan prima dengan ukuran yang diperlukan dan $r = 1$) digunakan dalam algoritma Shawe-Taylor (lihat 8.4.2). Jika diminta, algoritma juga dapat mengembalikan sertifikat primalitas (Pocklington).

c) Dengan diberikan dua bilangan prima bantu (*auxiliary prime*) p_1 dan p_2, algoritma di atas tersebut dapat digunakan untuk mengembalikan bilangan prima N dengan p_1 membagi N-*1* dan p_2 membagi N + *1* dengan pengesetan $m = p_1 p_2$ dan $r = \left((p_2{}^{-1} \bmod p1) \cdot p_2 - (p_1{}^{-1} \bmod p_2) \cdot p_1\right) \bmod m$. Sebagai contoh, jika T = tes Brillhart-Lehmer-Selfridge dan kondisi yang diperlukan terpenuhi, algoritma tersebut merespons bilangan terbukti prima N (dan sebuah sertifikat primalitas, jika diminta). Jika T sama dengan tes primalitas Miller-Rabin, dikembalikan suatu bilangan prima probabilistik.

B.2.3 Restriksi kongruensi dan algoritma Shawe-Taylor

Dalam rekursi terakhir algoritma Shawe-Taylor (8.4.2), bilangan prima *k*-bit N yang dikembalikan tersebut memenuhi $N \bmod 2q = 1$ untuk beberapa bilangan prima q. Hal ini dapat dimodifikasi untuk memastikan kondisi kongruensi tambahan $N \bmod m = r$ (dengan $m \neq q$) sebagai berikut. Jika m prima terhadap $2q$, mengeset $m_0 = 2qm$, $r_0 = (((2q)^{-1} \bmod m) \cdot 2q + (m^{-1} \bmod 2q) \cdot m) \bmod m_0$. Jika 2 membagi m (perhatikan bahwa *r* sebaiknya ganjil untuk memastikan p ganjil) mengeset $m_0 = qm$, $r_0 = \left((q^{-1} \bmod m) \cdot q + (m^{-1} \bmod q) \cdot m\right) \bmod m_0$. Dalam kedua kasus tersebut, integer bernilai $p = x + (r_0 - x) \bmod m_0$ memenuhi $p \bmod m = r$ dan $p \bmod 2q = 1$. Jadi, untuk rekursi akhir, langkah b) hingga d) sekarang boleh diubah menjadi yang berikut.

b) Pilih bilangan bulat *x* secara acak dari interval $(2k - 1, 2k - m_0]$.

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

d) Terapkan tes Pocklington pada p, dengan $p - 1 = F R$ dimana $F = q, R = t$.

Jika tes merespons "p *prima*", respons p bersama dengan sertifikat $C(p)$ (jika sertifikat diminta) dan stop.

Jika $p < 2k - m_0$, *set* $p = p + m_0$, $t = t + (m_0 \operatorname{div} q)$, dan lanjutkan ke langkah d)

Jika tidak, lanjutkan ke langkah b).

Jika N sebaiknya memenuhi kendala $\gcd(N - 1, e) = 1$, untuk beberapa integer *e* (seperti kasus RSA di mana e = eksponen (publik)), p pada langkah d) sebaiknya dicek untuk memenuhi $\gcd(p - 1, e) = 1$ sebelum menerapkan tes Pocklington yang lebih mahal secara komputasi.

Daripada menaikkan kandidat bilangan prima *p* pada langkah c) sebesar m_0 [pada langkah d)], untuk beberapa J pada $((2^{j-1} - p) / m_0, (2^j - p) / m_0)$, tes Pocklington dapat diterapkan ke elemen-elemen dalam sekuens $p, p + m_0, p + 2m_0, \ldots, p + J m_0$ yang bertahan melalui prosedur *sieving* yang dijelaskan dalam C.1. Proses *sieve* mengeliminasi kandidat bilangan-bilangan prima yang habis dibagi bilangan prima kecil yang digunakan dalam *sieve* tersebut.

B.2.4 Membangkitkan bilangan prima dalam suatu interval

Teks dalam dokumen ini berkaitan dengan membangkitkan k-bit bilangan prima acak, yaitu bilangan prima *p* sedemikian hingga $2^{k-1} \leq p < 2^k$. Untuk membangkitkan bilangan prima antara beberapa batas bawah A dan batas atas B sebarang, seseorang dapat menerapkan salah satu rekomendasi algoritma yang dijelaskan sebelumnya, namun mengganti 2^{k-1} dan 2^k masing-masing dengan A dan B. Bergantung pada intervalnya, hal ini juga dapat dilakukan hanya dengan pengesetan sejumlah bit-bit tinggi dari bilangan prima. Beberapa kehati-hatian sebaiknya diberikan dalam memilih nilai untuk A dan B, misalnya jika $B - A$ kecil, algoritma tertentu dapat gagal. Hal ini sudah tentu merupakan kasus untuk membangkitkan bilangan prima berdasarkan kondisi kongruensi (B.2) ketika $B - A < m$.

Aplikasi yang umum adalah membangkitkan RSA $2k$-bit modulus $n = pq$ di mana *p* dan *q* adalah *k*-bit bilangan prima. Dalam kasus ini, ambil $A = 2^{k-1}\sqrt{2}$ (atau $A = 2^{k-1} + 2^{k-2}$, yaitu membangkitkan bilangan prima dengan set dua bit tinggi) dan $B = 2^k$ mencukupi.

Untuk kasus algoritma Shawe-Taylor, restriksi interval perlu diterapkan hanya pada langkah b) pada langkah rekursi akhir (yang mengembalikan k-bit bilangan prima). Selanjutnya pengecekan $p < 2^j - 2q$ pada langkah d) diganti (pada langkah rekursi terakhir) dengan $p < B - 2q$. Demikian pula, ketika menggunakan metode *sieving* (dalam salah satu algoritma ini), interval yang disaring sebaiknya dimuat dalam $[A, B]$.

Lampiran C
(normatif)
Metode pembangkitan bilangan acak tambahan

C.1 Umum

Metode membangkitkan bilangan dari sekuens bit acak disediakan dalam ISO/IEC 18031. Dokumen ini menjelaskan tiga metode tambahan, dua diantaranya serupa dengan metode pembuangan sederhana dan metode modular sederhana dari ISO/IEC 18031 dengan pengecualian bahwa sekuens bit digunakan dalam urutan terbalik untuk membangkitkan bilangan acak. Sehubungan dengan pengurutan ini, bit keluaran pertama (acak) menjadi *most significant bit* dari bilangan yang dibangkitkan.

Metode pembuangan dan metode modular membangkitkan bilangan acak dalam interval [0, *r*) untuk beberapa batas r. Jika bilangan acak dibatasi di bawahnya oleh nilai A, metode pembangkitan bilangan acak dapat diulang hingga nilai kembalian memenuhi batas bawah A. Alternatifnya, algoritma ini dapat digunakan untuk membangkitkan bilangan acak, misalnya a, dalam $[0, r - A)$ dan sebagai gantinya mengembalikan bilangan $a + A$ [yang terletak di $[A, r)$].

C.2 Metode konversi simpel

Untuk mengonstruksi integer m-bit acak:

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

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

Dalam aplikasi dengan bilangan acak yang harus dibangkitkan dengan bit tertentu yang ditetapkan, bit yang sesuai pada langkah a) mungkin diset dengan tepat.

C.3 Metode pembuangan simpel (urutan terbalik)

Misalkan m adalah integer positif unik yang memenuhi $2^{m-1} \leq r \leq 2^m - 1$.

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

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

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

C.4 Metode modular simpel (urutan terbalik)

Misalkan m adalah integer positif unik yang memenuhi $2^{m-1} \leq r \leq 2^m - 1$, dan misalkan l adalah parameter keamanan.

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

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

c) *Return* $c \bmod r$.

Karena operator mod tidak seragam, nilai l = 64 sering kali direkomendasikan agar efeknya dapat diabaikan.

Lampiran D
(normatif)
Metode bantu (*Auxiliary methods*)

D.1 Prosedur *Sieving*

Diberikan suatu sekuens integer $S = Y_o, Y_o + h, \ldots, Y_o + J \cdot h$ untuk beberapa integer non-negatif h dan J, prosedur ini mengidentifikasi integer mana pada sekuens tersebut yang habis dibagi oleh faktor-faktor prima sampai batas tertentu K. Nilai tipikal untuk K adalah antara 10^3 dan 10^5. Lanjutkan sebagai berikut:

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* $i = (-h^{-1}Y0) \bmod p_j$.

2) *while* ($i \leq J$):

i) *set* $A[i] = 1$;

ii) *set* $i = i + p_j$.

Setelah menyelesaikan langkah c), pernyataan berikut ini benar untuk setiap integer $i \in [0, J]$:

— $Y_0 + i \cdot h$ habis dibagi suatu bilangan prima di B jika dan hanya jika $A[i] = 1$.

— Secara ekuivalen $Y_0 + i \cdot h$ tidak habis dibagi oleh satupun bilangan prima di B jika dan hanya jika $A[i] = 0$.

D.2 Tes primalitas berbasis teorema Pocklington

D.2.1 Umum

CATATAN 1 Algoritma yang diberikan di sini menghasilkan sertifikat primalitas untuk tujuan verifikasi. Jika suatu integer lulus tes ini, bilangan tersebut dijamin merupakan bilangan prima. Namun, jika digugurkan lebih awal, integer yang dimaksud dapat berupa bilangan komposit atau primalitasnya tidak dapat ditentukan.

Metode pengetesan primalitas yang disediakan di D.2 berbasis pada fakta berikut (varian dari teorema Pocklington).

Diberikan integer ganjil $N > 1$, andaikan terdapat faktorisasi $N - 1 = F \cdot R$, dengan F dan R adalah integer positif, dan faktorisasi prima dari F diketahui. Andaikan lebih lanjut bahwa integer non-negatif unik s dan r yang memenuhi $R = s'F + r$ dan $0 \leq r < F$ juga memenuhi $s < F + r$.

Jika, untuk setiap faktor prima q dari F, terdapat integer $a \in [2, N - 1]$ sedemikian hingga keduanya:

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

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

maka N bilangan prima jika dan hanya jika $s = 0$ atau $r^2 - 4s$ bukan kuadrat sempurna.
CATATAN 2 Nilai a dapat berbeda untuk setiap bilangan prima q yang membagi F.

D.2.2 Tes primalitas Pocklington

Untuk mengetes apakah integer $N \geq L$ (batas *trial division*) adalah bilangan prima dengan faktorisasi prima $F = {q_1}^{e_1} {q_2}^{e_2} \ldots {q_n}^{e_n}$ yang diberikan, dengan $N - 1 = F \cdot R$, teorema Pocklington harus diterapkan sebagai berikut.

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 $s > 0$ dan $r^2 - 4s$ *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* Tabel 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" dan 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* $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$.

Tabel D.1 Jumlah a untuk mengetes suatu q yang diberikan

q	T = jumlah a untuk dites
2	7
3	5
5 atau 7	3
$11 \leq q \leq 97$	2
$97 < q$	1

Jika N bilangan prima maka untuk suatu q yang diberikan, probabilitas terpilihnya a secara acak memenuhi kondisi 1 dan 2 dari D.2.1 adalah $1 - 1/q$. Nilai T pada Tabel D.1 diset sedemikian hingga probabilitas a gagal pada langkah b) 3) kurang dari 0,01, asalkan N adalah bilangan prima.

CATATAN 1 Pengetesan apakah $r^2 - 4s$ adalah kuadrat sempurna pada langkah a) dapat dilakukan menggunakan salah satu dari metode yang tercantum dalam D.9.

CATATAN 2 Tingkat kegagalan tes, yaitu tingkat tesnya tidak meyakinkan, dapat direduksi dengan meningkatkan nilai T yang dipilih pada langkah b) 2).

CATATAN 3 a yang dipilih pada langkah 3) dapat dipilih secara non-acak, misalnya implementasi dapat dimulai dengan $a = 2$, dan kenaikan di dalam *while loop* terhadap pangkat integer berikutnya yang tidak sempurna. Sebagai contoh, jika $a = 2$ telah dipilih pada langkah 3), nilai $a = 4$, 8, 16, ... dapat dilewati karena pengecekan 3) i) akan berlebihan, yaitu $2^{n-1} \bmod N = 1$ berimplikasi $4^{N-1} \bmod N = 1$, $8^{N-1} \bmod N = 1$, dst.

CATATAN 4 q_i yang muncul dalam faktorisasi F itu sendiri dapat disertai dengan sertifikat primalitas. Dengan asumsi sertifikat yang menyertainya benar, maka sertifikat tersebut juga disertakan dalam sertifikat C yang dikembalikan pada langkah c).

Memverifikasi sertifikat C di atas untuk N harus dilakukan sebagai berikut.

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* ${a_1}^{N-1} \bmod N \neq 1$

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

3) *If* $\gcd(\ ({a_1}^{(N-1)/q_i} - 1)\ mod\ N, N) \neq 1, N$,

return "certificate invalid, N *composite" and stop*;

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

return "certificate invalid" and stop.

4) *while* $(R \bmod q_i\ ) = 0$:

i) *set* $R = R \operatorname{div} q_i$;

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 $s = 0$ atau $r^2 - 4s$ *is a non-square, accept the certificate, return "*N *prime" and stop*.

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

CATATAN 5 Nilai F yang digunakan dalam upaya memverifikasi sertifikat $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$ dapat melibatkan pangkat yang lebih tinggi dari bilangan prima tersebut daripada nilai F yang digunakan untuk membangkitkan sertifikat itu, karena *while loop* pada langkah c) 4) mengonstruksi F terbesar yang membagi $N - 1$ dengan faktor prima dalam daftar $\{q_1, q_2, \ldots, q_n\}$. Oleh karena itu, nilai R, s, dan r yang "dipulihkan" juga dapat berbeda dari nilai yang digunakan untuk mengawali konstruksi sertifikat untuk N.

CATATAN 6 Algoritma pembangkitan dan verifikasi sebagaimana tercantum dalam dokumen ini diadaptasi dari teorema 5 Acuan[6]. Kondisi $s < F + r$ ekuivalen dengan kondisi $N < F^3 + r \cdot F^2 + r \cdot F + 1 = (F + 1)(F^2 + (r - 1) \cdot F + 1)$.

D.2.3 Tes primalitas Pocklington parsial

Jika kondisi $s < F + r$ tidak terpenuhi, primalitas bilangan yang dites masih dapat ditentukan dengan mengkombinasikan hasil dengan tes primalitas Lucas deterministik parsial (D.4) sebagaimana dispesifikasikan dalam tes Brillhart-Lehmer-Selfridge (D.5). Algoritma tersebut masih mengembalikan sertifikat "parsial" – sertifikat lengkap adalah sertifikat parsial yang dikombinasikan dengan sertifikat Lucas deterministik parsial.

Versi yang dilemahkan ini disebut sebagai tes primalitas parsial Pocklington dan memodifikasi tes primalitas Pocklington sebagai berikut.

— Kondisi $s \geq F + r$ dihilangkan pada langkah a).

— Langkah c) mengembalikan "N lulus" dengan sertifikat parsial C yang berisi nilai $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$.

Verifikasi sertifikat parsial C dimodifikasi dengan cara yang sama.

— Langkah e) pengecekan $s \geq F + r$ dihilangkan.

— Langkah e) mengembalikan "N lulus" dan nilai F yang dihitung pada langkah c) jika kondisi pada s dan $r^2 - 4s$ terpenuhi, dan mengembalikan "sertifikat tidak valid, jika tidak N komposit".

D.3 Tes primalitas Lucas probabilistik

Metode pengetesan primalitas yang disediakan di sini berbasis fakta berikut (lihat Acuan [3]): Jika N bilangan prima dan D integer dengan $D \bmod 4 = 1$ sedemikian hingga:

1) $\text{Jacobi}(D, N) = -1$;

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

maka $\text{Lucas}(D, N + 1, N) = 0$.

Untuk mengetes apakah integer ganjil $N \geq L$ (batas *trial division*) adalah bilangan prima, tes probabilistik Lucas harus diterapkan sebagai berikut.

a) *If N is a perfect square, return "N composite" and stop.*

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

c) *While* $(\text{Jacobi}(D, N) \neq -1$ *or* $\gcd(N, Q) \neq 1)$:

   1) *If* $\text{Jacobi}(D, N) = 0$ *and* $D \bmod N \neq 0$, *return* "N composite" *and stop*.

   2) *If* $\gcd(N, Q) \neq 1$ *and* $Q \bmod N \neq 0$, *return* "N composite" *and stop*.

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

d) *If* $\text{Lucas}(D, N + 1, N) = 0$, *return* "$N$ *accepted*" *and stop.*

   Otherwise, *return* "N *composite*" *and stop*.

CATATAN 1 Metode pengetesan kuadrat sempurna diberikan pada D.9, untuk menghitung simbol Jacobi pada D.10, dan $\text{Lucas}(D, N + 1, N)$ pada D.11.

CATATAN 2 Pengetesan pada langkah a) dilakukan untuk menghindari *loop* tak hingga pada langkah c) jika N adalah kuadrat sempurna.

CATATAN 3 Nilai D dimulai dengan $D = 5$ pada langkah b) sebagai $\text{Jacobi}(1, N)$ selalu 1 dan berkisar pada himpunan $\{5, -7, 9, -11, 13, -15, 17, \ldots\}$.

D.4 Tes primalitas desterministik Lucas

D.4.1 Umum

CATATAN 1 Algoritma yang diberikan di sini menghasilkan sertifikat primalitas untuk tujuan verifikasi. Jika suatu integer lulus tes ini, bilangan tersebut dijamin menjadi bilangan prima. Namun, jika dibatalkan lebih awal, integer yang dimaksud dapat berupa bilangan komposit atau primalitasnya tidak dapat ditentukan.

Metode pengetesan primalitas yang diberikan pada D.4 berbasis fakta berikut:
Diberikan integer ganjil $N > 1$, andaikan terdapat faktorisasi $N + 1 = F \cdot R$, dengan F dan R adalah integer positif, dan faktorisasi prima dari F diketahui. Andaikan lebih jauh lagi bahwa integer non-negatif unik s dan r yang memenuhi $R = s \cdot F + r$ dan $0 \leq r < F$ juga memenuhi $s + r < F$.

Jika untuk setiap faktor prima q dari F terdapat integer D dengan $D \bmod 4 = 1$ sedemikian hingga:

1) $\text{Jacobi}(D, N) = -1$;
2) $\text{Lucas}(D, N + 1, N) = 0$; dan
3) $\gcd(\text{Lucas}(D, (N + 1)/q, N), N) = 1$;

maka *N* bilangan prima jika dan hanya jika $s = 0$ atau $r^2 + 4s$ bukan kuadrat sempurna.

Pilihan D bisa berbeda untuk masing-masing q_i yang membagi F. Selanjutnya jika untuk D tertentu, kondisi 1 benar tetapi kondisi 2 gagal, maka N komposit dan pengetesan dapat dihentikan.

CATATAN 2 Metode pengetesan kuadrat sempurna diberikan pada D.9, untuk menghitung simbol Jacobi pada D.10, dan $\text{Lucas}(D, N + 1, N)$ di D.11.

D.4.2 Tes primalitas deterministik Lucas

Untuk mengetes apakah integer $N \geq L$ (batas *trial division*) adalah bilangan prima dengan faktorisasi prima $F = q_1^{e_1} q_2^{e_2} \ldots q_n^{e_n}$, yang diberikan, dengan $N + 1 = F \cdot R$, tes Lucas harus diterapkan sebagai berikut.

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

If $s + r \geq F$, *return "test inconclusive" and stop* (karena kondisi yang diperlukan untuk menerapkan tes belum terpenuhi).

If $s \neq 0$ dan $r_2 + 4s$ *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 (misalnya dari Tabel D.2), *based on the value of* q_i.
2) *Set* $j = 0$, *set* $D = 5$.
3) *While* $(\text{Jacobi}(D, N) \neq -1$ *and* $j < T)$:

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

If $D \bmod N \neq 0$, *return* "N commposite" *and stop*.

Otherwise, *set* $j = j + 1$.

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

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

5) *If* $\operatorname{Lucas}(D, N+1, N) \neq 0$:

If $Q \bmod N \neq 0$ *where* $Q = (1 - D) \operatorname{div} 4$, *return "*$N$ *composite" and stop*.

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

6) *If* $\gcd(\operatorname{Lucas}(D, (N+1)/q_i, N), N) = 1$, *set* $D_i = D$.

Otherwise, if $\operatorname{Lucas}(D, (N+1)/q_i, N) \neq 0$, *return "*$N$ *composite" and stop*.

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

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

Tabel D.2 Jumlah D untuk mengetes suatu q yang diberikan

q	T = jumlah a untuk dites
2	17
3	12
5 , 7	10
$11 \leq q \leq 23$	8
$23 < q$	7

Jika N bilangan prima maka untuk suatu q yang diberikan, probabilitas D acak yang memenuhi kondisi 1 hingga 3 dari D.4.1 adalah sekitar $(1/2) \cdot (1 - 1/q)$. Nilai T pada Tabel D.2 diset sedemikian hingga probabilitas D gagal pada langkah b) 3) kurang dari 0,01 jika N adalah bilangan prima.

CATATAN 1 q_i yang muncul dalam faktorisasi F sendiri dapat disertai dengan sertifikat primalitas. Dengan asumsi sertifikat yang menyertainya benar, maka sertifikat tersebut juga disertakan dalam sertifikat C yang dikembalikan pada langkah c).

CATATAN 2 Pengetesan $r_2 + 4s$ adalah kuadrat sempurna pada langkah a) dapat dilakukan dengan menggunakan metode deterministik pada D.9.

CATATAN 3 Kondisi pada langkah b) 5) merupakan akibat dari fakta berikut:

1) untuk bilangan prima *N* dengan $Jacobi(D, N) = -1$ dan $\gcd(Q, N) = 1$, maka $\operatorname{Lucas}(D, N+1, N) = 0$; yaitu, jika $\operatorname{Lucas}(D, N+1, N) \neq 0$, $\operatorname{Jacobi}(D, N) = -1$, dan $\gcd(Q, N) = 1$, maka N tidak dapat menjadi bilangan prima;

2) jika $1 < \gcd(Q, N) < N$, maka N bukan bilangan prima. Jadi jika $\operatorname{Lucas}(D, N+1, N) \neq 0$, $\operatorname{Jacobi}(D, N) = -1$ dan $\gcd(Q, N) < N$, atau ekuivalen dengan $Q \bmod N \neq 0$, maka N bukan bilangan prima.

Memverifikasi sertifikat C di atas untuk N harus dilakukan sebagai berikut.

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* $\text{Jacobi}(D, N) = 0$:

If $D \bmod N \neq 0$, *return "certificate invalid,* N *composite" and stop*.

Otherwise, *return "certificate invalid" and stop*.

3) *If* $\text{Jacobi}(D, N) = 1$, *return "certificate invalid" and stop*.

4) *If* $\text{Lucas}(D, N + 1, N) \neq 0$:

If $Q \bmod N \neq 0$ *where* $Q = (1 - D) \text{ div } 4$, *return "certificate invalid,* N *composite" and stop.*

Otherwise, return "certificate invalid" and stop.

5) *If* $cd(\text{Lucas}(D_i, (N + 1)/q_i, N), N) \neq 1$:

If $\text{Lucas}(D_i, (N + 1)/q_i, N) \neq 0$, *return "certificate invalid,* N *composite" and stop*.

Otherwise, return "certificate invalid" and stop.

6) *While* $(R \bmod q_i) = 0$:

i) *Set* $R = R \; div \; q_i$.

ii) *Set* $F = F \cdot qi$.

d) *Compute* (r, s) *where* $R = sF + r, s \geq 0$ *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*.

Verifikasi primalitas q_i pada langkah c) 1) harus dilakukan dengan menggunakan metode deterministik dalam dokumen ini.

CATATAN 4 Nilai F yang digunakan dalam upaya memverifikasi sertifikat $\{(N, q1, D1), (N, q2, D2), \ldots, (N, qn, Dn)\}$ dapat melibatkan pangkat yang lebih tinggi dari bilangan prima tersebut daripada nilai F yang digunakan untuk membangkitkan sertifikat tersebut, karena *while loop* pada langkah c) 6) mengonstruksi F terbesar yang membagi $N + 1$ dengan faktor prima dalam daftar $\{q_1, q_2, \ldots, q_n\}$. Oleh karena itu, nilai R, s, dan r yang "dipulihkan" juga dapat berbeda dari nilai yang digunakan untuk mengawali konstruksi sertifikat untuk N.

CATATAN 5 Algoritma pembangkitan dan verifikasi sebagaimana tercantum dalam dokumen ini diadaptasi dari teorema 17 dari Acuan [6] Kondisi $s + r < F$ ekuivalen dengan kondisi $N < F^3 - r \cdot F^2 + r \cdot F - 1 = (F-1)(F^2 + (1-r) \cdot F + 1)$.

D.4.3 Tes primalitas deterministik parsial Lucas

Jika kondisi $s + r < F$ tidak terpenuhi, primalitas bilangan yang dites masih dapat ditentukan dengan mengkombinasikan hasil dengan tes primalitas parsial Pocklington (D.2) sebagaimana dispesifikasikan dalam tes Brillhart-Lehmer-Selfridge (D.5). Algoritma tersebut masih mengembalikan sertifikat "parsial" – sertifikat lengkap adalah sertifikat parsial yang digabungkan dengan sertifikat Pocklington parsial.

Versi yang dilemahkan ini disebut sebagai tes primalitas Lucas deterministik parsial dan memodifikasi tes primalitas Lucas deterministik sebagai berikut.

— Kondisi $s + r \geq F$ dihilangkan pada langkah a).

— Langkah c) mengembalikan "N lulus" dengan sertifikat parsial C yang berisi nilai $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$.

Verifikasi sertifikat parsial C dimodifikasi dengan cara yang sama:

— Langkah e) yang memeriksa $s + r \geq F$ dihilangkan.

— Langkah f) mengembalikan "N lulus" dan nilai F yang dihitung pada langkah c) jika kondisi pada s dan $r_2 + 4s$ terpenuhi, dan mengembalikan "sertifikat tidak valid, jika tidak N komposit".

D.5 Tes primalitas Brillhart-Lehmer-Selfridge

CATATAN 1 Algoritma yang diberikan di sini menghasilkan sertifikat primalitas untuk tujuan verifikasi. Jika suatu integer lulus tes ini, bilangan tersebut dijamin merupakan bilangan prima. Namun, jika dibatalkan lebih awal, integer yang dimaksud dapat berupa bilangan komposit atau primalitasnya tidak dapat ditentukan.

CATATAN 2 Metode ini dapat digunakan untuk membangkitkan bilangan prima kuat, yaitu bilangan prima p di mana $p + 1$ dan $p - 1$ mempunyai pembagi prima yang besar.

Metode pengetesan primalitas yang disediakan di D.5 berbasis hal berikut.

Diberikan integer ganjil $N > 1$, andaikan terdapat faktorisasi $N - 1 = F_1 \cdot R_1$ dan $N + 1 = F_2 \cdot R_2$ dan faktorisasi prima dari F_1 dan F_2 keduanya diketahui.

Jika, untuk setiap faktor prima q dari F_1, terdapat integer $a \in [2, N - 1]$ sedemikian hingga keduanya:

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

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

dan untuk setiap faktor prima r dari F_2 terdapat (diskriminan) D pada $\{5, -7, 9, -11, 13, -15, 17, \ldots\}$ sedemikian hingga:

1) $\text{Jacobi}(D, N) = -1$;

2) $\text{Lukas}(D, N + 1, N) = 0$; dan

3) $\gcd(\text{Lucas}(D, (N + 1)/r, N), N) = 1$;

maka setiap pembagi positif d dari N harus memenuhi: $d \bmod \text{lcm}(F_1, F_2) = 1$ atau $d \bmod \text{lcm}(F_1, F_2) = N \bmod \text{lcm}(F_1, F_2)$.

Jadi, jika kelima kondisi 1)-2) dan 1)-3) terpenuhi, maka N bilangan prima jika:

a) $\text{lcm}(F_1, F_2) > N$;

b) $\sqrt{N} - 1 < \text{lcm}(F_1, F_2) < N$ dan $N \bmod \text{lcm}(F_1, F_2)$ bukan pembagi dari N; atau

c) tidak ada pembagi yang tepat d yang memenuhi $d \bmod \text{lcm}(F_1, F_2) = 1$.

CATATAN 3 Jika a) atau b) berlaku, maka N adalah bilangan prima karena semua pembagi sejati (*proper divisor*) dari N akan lebih besar dari $\sqrt{N}$. Jika c) berlaku, N juga bilangan prima. Jika tidak,

$N \bmod \mathrm{lcm}(F_1, F_2) = N^2 \bmod \mathrm{lcm}(F_1, F_2)$ (karena N adalah hasil kali dua pembagi sejati (*proper divisor*), yaitu $N \bmod \mathrm{lcm}(F_1, F_2) = 1$ yang bertentangan dengan c).

CATATAN 4 Jika $\mathrm{lcm}(F_1, F_2) > N^{1/3}$, maka c) dapat dites menggunakan algoritma Lenstra (lihat D.12).

CATATAN 5 Nilai a dapat berbeda untuk setiap q pembagi F_1 dan nilai D dapat berbeda untuk setiap r pembagi F_2. Tabel D.1 memberikan indikasi jumlah nilai yang akan dites untuk suatu q yang diberikan. Tabel D.2 memberikan indikasi jumlah nilai D yang akan dites untuk suatu r yang diberikan.

Untuk mengetes apakah suatu integer $N \geq L$ (batas *trial division*) adalah bilangan prima dengan faktorisasi prima $F1 = q_1^{e_1} q_2^{e_2} \ldots q_n^{e_n}$, dan $F2 = r_1^{f_1} r_2^{f_2} \ldots r_n^{f_n}$, yang diberikan dengan $N - 1 = F_1 R_1$ dan $N + 1 = F_2 R_2$, hasil di atas harus diterapkan sebagai berikut.

a) Jika $\mathrm{lcm}(F_1, F_2) \leq N^{1/3}$, stop; kondisi yang diperlukan untuk tes ini tidak terpenuhi.

b) Terapkan tes primalitas parsial Pocklington ke N menggunakan faktorisasi parsial $N - 1 = F_1 \cdot R_1$.

Jika dikembalikan *"*tes tidak meyakinkan*",* kembalikan *"*tes tidak meyakinkan*"* dan stop*.*

Jika dikembalikan *"*N komposit*",* kembalikan *"*N komposit*"* dan stop*.*

Jika dikembalikan *"*N lulus*"* dengan sertifikat parsial dengan nilai $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$. lanjutkan ke langkah berikutnya.

c) Terapkan tes primalitas Lucas deterministik parsial ke N menggunakan faktorisasi parsial $N + 1 = F_2 R_2$.

Jika dikembalikan *"test inconclusive",* kembalikan *"test inconclusive"* dan stop*.*

Jika dikembalikan *"*N *composite",* kembalikan *"*N *composite"* dan stop*.*

Jika dikembalikan *"*N *passes"* dengan sertifikat parsial dengan nilai$\{(N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$, lanjutkan ke langkah berikutnya.

d) Jika $\mathrm{lcm}(F_1, F_2) > N$, kembalikan "$N$ bilangan prima." Jika sertifikat diminta, kembalikan sertifikat dengan nilai $\{(N, a_1, q_1), (N, a_2, q_2), \ldots, (N, a_n, q_n), (N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$. Stop.

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

Jika $N \; mod \; (N \bmod \mathrm{lcm}(F_1, F_2)) = 0$, kembalikan "$N$ komposit" dan stop.

Jika tidak, kembalikan *"*N prima*."* Jika sertifikat diminta, kembalikan sertifikat dengan nilai. $\{(N, a_1, q_1), (N, a_2, q_2), \ldots, (N, a_n, q_n (N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$ Stop.

f) Terapkan tes residu Lenstra D.12 dengan input $s = \mathrm{lcm}(F_1, F_2)$ dan $n = N$.

Jika kembalian tes salah, yaitu tidak terdapat suatu pembagi tepat d dari N dengan $d \bmod \mathrm{lcm}(F_1, F_2) = 1$, kembalikan "$N$ prima." Jika sertifikat diminta, kembalikan sertifikat dengan nilai $\{(N, a_1, q_1), (N, a_2, q_2), \ldots, (N, a_n, q_n), (N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$. Stop.

Jika tidak, kembalikan "N komposit" dan stop.

Penggunaan algoritma Lenstra D.12, dapat dihindari dengan mensyaratkan hanya faktorisasi parsial yang memenuhi $\sqrt{N - 1} < \mathrm{lcm}(F_1, F_2)$.

Memverifikasi sertifikat C di atas untuk N harus dilakukan sebagai berikut.

a) Verifikasi sertifikat Pocklington parsial $\{(N, a_1, q_1), (N, a_2, q_2), \ldots, (N, a_n, q_n)\}$.

Jika mengembalikan "sertifikat tidak valid, N komposit," kembalikan "sertifikat tidak valid, N komposit" dan stop.

Jika mengembalikan "sertifikat tidak valid," kembalikan "sertifikat tidak valid" dan stop.

Jika mengembalikan "N lulus" dengan nilai F, setel $F_1 = F$ dan lanjutkan ke langkah berikutnya.

b) Verifikasi sertifikat Lucas deterministik parsial $\{(N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$.

Jika mengembalikan "sertifikat tidak valid, N komposit," kembalikan "sertifikat tidak valid, N komposit" dan stop.

Jika mengembalikan "sertifikat tidak valid," kembalikan "sertifikat tidak valid" dan stop.

Jika mengembalikan "N lulus" dengan nilai F, setel $F_2 = F$ dan lanjutkan ke langkah berikutnya.

c) Jika $\mathrm{lcm}(F_1, F_2) \leq N^{1/3}$, kembalikan "sertifikat tidak valid" dan stop.

d) Jika $\mathrm{lcm}(F_1, F_2) > N$, terima sertifikatnya, kembalikan "N prima" dan stop.

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

Jika $N \bmod (N \bmod \mathrm{lcm}(F_1, F_2)) = 0$, respons "sertifikat tidak valid, N komposit" dan stop.

Jika tidak, terima sertifikatnya, kembalikan "N prima" dan stop

f) Terapkan tes residu Lenstra, D.12, dengan input $s = \mathrm{lcm}(F_1, F_2)$ dan $n = N$.

8.5 Jika tes merespons salah, yaitu tidak ada pembagi yang tepat *d* dari *N* dengan $d \bmod \mathrm{lcm}(F_1, F_2) = 1$, terima sertifikatnya, kembalikan "*N* prime" dan stop.

Jika tidak, kembalikan "sertifikat tidak valid, *N* komposit" dan stop.

CATATAN 6 Algoritma ini diadaptasi dari teorema 20 Acuan [6] dan mengkombinasikan hasil tes "$N-1$" yang diberikan oleh tes Pocklington dan tes "$N+1$" yang diberikan oleh tes deterministik Lucas. Selain itu, kondisi 1-2 dari D.2.1 dan 1-3 dari D.4.1 berimplikasi semua pembagi d dari N memenuhi $d \bmod F_1 = 1$ dan $d \bmod F_2 = \pm 1$ (teorema 4 dan 16 Acuan [6]) tempat asal $d \bmod \mathrm{lcm}(F_1, F_2) = 1$ atau $N \bmod \mathrm{lcm}(F_1, F_2)$.

D.6 Tes primalitas kurva eliptis

Titik 0 pada kurva eliptis menunjukkan elemen identitas. Informasi tambahan mengenai kurva eliptis dapat diperoleh dari ISO/IEC 15946-1 dan Acuan [2], [4] dan [19].

CATATAN 1 Algoritma yang diberikan di sini menghasilkan sertifikat primalitas untuk tujuan verifikasi. Jika suatu integer lulus tes ini, bilangan tersebut dijamin merupakan bilangan prima. Namun, jika dibatalkan lebih awal, integer yang dimaksud dapat berupa bilangan komposit atau primalitasnya tidak dapat ditentukan.

Metode pada D.6 berbasis pada fakta berikut untuk integer N dengan $\gcd(N, 6) = 1$. Jika terdapat kurva eliptis *E* yang diberikan oleh $y^2 = x^3 + ax + b$ sedemikian hingga:

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

b) terdapat titik $P = (x, y) \neq 0$ di E_N, kurva E direduksi modulo N, orde r dimana:

1) r adalah prima;

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

maka N adalah bilangan prima.

Lihat Acuan [19] untuk properti kurva eliptis modulo bilangan komposit.

Untuk mengetes apakah integer $N \geq L$ (batas *trial division*) adalah bilangan prima dengan suatu kurva eliptis E yang diberikan dengan mendefinisikan persamaan $y^2 = x^3 + ax + b$, tes kurva eliptis harus diterapkan sebagai berikut.

a) *If* $\gcd(4a^3 + 27b^2, N)$ *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* (sehingga Z_N adalah *field*), *determine* t *= the order of the putative elliptic curve* E_N (kurva E direduksi 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 orde*r r, *is to take an arbitrary nonzero poin* $P \neq 0$ *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*.

Metode untuk penemuan titik P berorde r adalah dengan mengambil titik sembarang nonzero $Q = (x_0, y_0)$ pada E_N dan memeriksa apakah $P = (t/r)\, Q$ berorde r. Jika $P \neq 0$ dan $r \cdot P \neq 0$, maka N bukan bilangan prima dan langkah c) dapat mengembalikan "N komposit". Jika $P = 0$, maka elemen sembarang Q lainnya dapat dipilih.

Dalam kasus N bukan bilangan prima, komputasi pada langkah b) dan c) dapat gagal karena mencoba menginversi elemen Z_N yang noninvertibel; yaitu, integer s ditemukan sedemikian hingga $\gcd(s, N)$ adalah pembagi sejati dari N. Dalam kasus seperti itu, algoritma dapat mengembalikan "N komposit" dan stop. Namun, jika N hanya habis dibagi oleh faktor-faktor besar, kecil kemungkinannya untuk menemukan pembagi s dari N dengan cara ini, yaitu tes tersebut kemungkinan besar tidak meyakinkan. Sebagai tindakan pencegahan, suatu implementasi dapat melakukan pengetesan tambahan, seperti beberapa tes primalitas Miller-Rabin, pada N *sebelum* menerapkan tes kurva eliptis.

CATATAN 2 Kehati-hatian mungkin diperlukan pada langkah b), karena hal ini dapat menjadi hambatan yang memakan waktu. Komputasi t dapat dilakukan dengan menggunakan beberapa algoritma seperti algoritma Schoof[16] atau perbaikan Elkies dan Atkins[11] Namun, akan lebih efisien untuk menghindari langkah komputasi ini dengan mengonstruksi kurva berorde yang ditentukan menggunakan metode CM.[1]

Untuk mengetes apakah integer $N \geq L$ adalah bilangan prima dengan pemberian sertifikat dengan nilai (N, r, t, a, b, P), lanjutkan sebagai berikut:

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* $\gcd(4a^3 + 27b^2, N)$ *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.*

e) *Verify* P *has order* r, *i.e.* $P \neq 0$ (titik nol pada E_N) *and* $r \cdot P = 0$. *If not, return "certificate invalid" and stop. Otherwise, accept the certificate, return "*N *prime" and stop*.

D.7 Algoritma Cornacchia

Diberikan integer $M > 0$, dan integer δ dengan $1 \leq \delta < M$ dan $\gcd(\delta, M) = 1$, D.7 memberikan algoritma yang menemukan pasangan integer (V, W) yang memenuhi $M = V^2 + \delta W^2$ (asalkan ada solusinya).

a) Usaha menemukan solusi, r_1, dari kongruensi $(r_1)^2 \equiv -\delta \pmod M$ yang juga memenuhi pertidaksamaan $0 \leq r1 \leq (M/2)$. Jika tidak ada solusi yang ditemukan, batalkan (tidak ada solusi persamaan Diophantine).

b) Set $r_0 = M$, dan dengan menggunakan nilai r_1 yang ditentukan pada langkah a), perhatikan sekuens $r_0, r_1, r_2 \ldots$ yang didefinisikan secara rekursif dengan menyeting $r_i + 2 = r_i \bmod r_{i+1}$ untuk integer $i \geq 0$ Tentukan integer positif terkecil k yang $(r_k)^2 < M$.

c) Set $s = ((M - (r_k)^2)/\delta)^{1/2}$Jika s bilangan bulat, respons $(V, W) = (r_k, s)$; jika tidak, batalkan (tidak ada solusi persamaan Diophantine).

D.8 Metode multiplikasi kompleks (CM)

Algoritma yang diberikan di sini membangkitkan kurva eliptis dengan orde yang diketahui Z_N dengan N adalah bilangan prima.

Misalkan $\Delta = \{-3, -4, -7, -8, -11, -15, -19, -20, -23, -24, -31, -35, -39, -40, -43, -47, \ldots\}$ adalah himpunan diskriminan kuadratik fundamental, yaitu himpunan integer negatif d dengan $d \bmod 4 = 1$, $d \bmod 16 = 8$, atau $d \bmod 16 = 12$.

a) *Pick a discriminant d in* Δ, *and put* $D = -d$.

b) *Using Cornacchia's algorithm,* D.7, *find integers* V *and* W *such that* $4N = V^2 + DW^2$. *If no such pair of integers exists, go to step a) and select a new discriminant* d.

c) *If* $d = -3$, *set*:

$$T = \{N + 1 - V, N + 1 + V, N + 1 - 1/2 \cdot (-V + 3W), N + 1 + 1/2 \cdot (-V + 3W), N + 1 - 1/2 \cdot (V + 3W), N + 1 + 1/2 \cdot (V + 3W)\}.$$

If $d = -4$, *set*:

$$T = \{N + 1 - V, N + 1 + V, N + 1 - 2W, N + 1 + 2W\}.$$

If $d < -4$, *set*:

$$T = \{N + 1 - V, N + 1 + V\}.$$

d) *For each t in* T, *use the CM method to construct the elliptic curve* E_t *that reduced modulo* N *has order* t. *Return the set of tuples* $\{(Et, t) : t \text{ in } T\}$ *and stop.*

Kurva di E_t pada langkah d) dapat dibuat menggunakan metode CM sebagaimana dispesifikasikan dalam ISO/IEC 15946-1.

D.9 Pengetesan untuk kuadrat sempurna

D.9.1 Umum

Pasal ini berisi dua metode untuk menentukan apakah suatu integer n merupakan kuadrat sempurna. Metode pertama bersifat probabilistik sedangkan metode kedua bersifat deterministik. Metode pertama sebaiknya digunakan untuk menentukan apakah suatu integer bukan kuadrat sempurna.

D.9.2 Metode probabilistik

Metode ini memanfaatkan simbol Jacobi sebagaimana dijelaskan pada 3.4, dan mengambil input berupa integer n dan bilangan prima ganjil *p* sedemikian hingga $\gcd(n,p) = 1$. Metode ini mengembalikan "n lulus iterasi ini" atau "n bukan kuadrat sempurna".

Untuk menentukan apakah n kuadrat sempurna atau tidak, periksa apakah $\text{Jacobi}(n,p) = 1$. Jika demikian, kembalikan "n lulus iterasi ini". Jika tidak, kembalikan "n bukan kuadrat sempurna".

CATATAN Pengetesan ini bersifat probabilistik dan sebuah non-kuadrat akan lulus, katakan, 20 iterasi dari pengetesan ini terhadap 20 bilangan prima berbeda dengan probabilitas 2^{-20}.

Jika integer lulus beberapa iterasi tes ini, tes deterministik tersebut sebaiknya diterapkan untuk mendapatkan jawaban definitif.

D.9.3 Metode deterministik

Metode yang disediakan di sini berbasis metode Newton. Metode tersebut menerima integer m-bit n sebagai input dan merespons "*true*" jika n adalah kuadrat sempurna dan "*false*" jika sebaliknya.

Untuk menentukan apakah n merupakan kuadrat sempurna atau tidak, lakukan hal berikut.

a) *Set* $b = m \operatorname{div} 2, x = 2^{b+1}$ *and* $y = (x + (n \operatorname{div} x)) \operatorname{div} 2$.

b) *While* $y < x$*:*

1) *set* $x = y$;

2) *set* $y = (x + (n \operatorname{div} x)) \operatorname{div} 2$.

c) *If* $x_2 = n$, *return "true" and the root* x, *else return "false".*

D.10 Simbol Jacobi

Pada input integer a dan integer ganjil n, respons nilai $\text{Jacobi}(a,n)$.

a) *If* $\gcd(a,n) \neq 1$, *return* 0.

b) *set* $a = a \bmod n$.

c) *Write* $a = 2^e b$, *where* b *is odd and* $e \geq 0$.

d) *If* $n \bmod 8 = 1$ or 7, *set* $v = 1$. *Otherwise*, *set* $v = (-1)^e$.

e) *Set* $a = b$.

f) *While* $a \neq 1$:

1) If $a \bmod 4 = 3$ *and* $n \bmod 4 = 3$, *then set* $v = -v$;

2) *set* $r = n \bmod a$;

3) *write* $r = 2^f s$, *with* s *is odd and* $f \geq 0$;

4) *If* $a \bmod 8 = 3$ *or* 5, *set* $v = (-1)^f v$;

5) *set* $n = a$ *and* $a = s$.

g) *Return* v.

D.11 Simbol Lucas

Pada input diskriminan kuadrat D, integer positif K, dan integer ganjil N, mengembalikan nilai $\text{Lucas}(D, K, N)$.

a) *Set* $U = 1$ *and* $V = 1$.

b) *Let* $K_r K_r - 1 \ldots K_0$ *be the binary expansion* of K *where* K_r *is* 1. *The bit-length of* K *is* $r + 1$.

c) *For* i *from* r-1 *to* 0:

1) *Set* $(U, V) = ((UV) \bmod N, (V^2 + DU^2)/2 \bmod N)$;

2) *If* $K_i = 1$, *set* $(U, V) = ((U + V)/2 \bmod N, (V + DU)/2 \bmod N)$.

d) *Return* U.

CATATAN Langkah c) 1) dan c) 2) memanggil untuk evaluasi dalam bentuk $W/2 \bmod N$ dengan W adalah integer dan N diasumsikan integer ganjil. Jika W ganjil, maka $W/2 \bmod N$ dapat dikalkulasi sebagai $((W + N) \text{ div } 2) \bmod N$.

D.12 Menemukan pembagi di kelas residu (Lenstra).

Diberikan integer n dan modulus $s > n^{1/3}$ dengan $\gcd(n, s) = 1$, algoritma merespon "*true*" jika terdapat pembagi non-trivial d dari n yang memenuhi $d \bmod s = 1$. Algoritma ini merupakan kasus khusus dari hasil yang lebih umum karena Lenstra.[12]

a) *Set* $a_0 = s$, $b_0 = 0$, *and* $c_0 = 0$.

b) *Set* $a_1 = n \bmod s$, $b_1 = 1$, *and* $c1 = ((n - a_1) \text{ div } s) \bmod s$.

c) *Set* $i = 1$.

d) *While* $(a_i \neq 0)$:

1) *if* i *is even and* $a_i - 1 \bmod a_i = 0$, *set* $a_i + 1 = a_i$;

Otherwise, *set* $a_i + 1 = a_i - 1 \bmod a_i$.

2) *set* $q = (a_i - 1 - a_i + 1) \text{ div } a_i$;;

3) *set* $b_i + 1 = b_i - 1 - q\ b_i$;

4) *set* $c_i + 1 = (c_i - 1 - q\ c_i) \bmod s$;

5) *increment* i, *i.e. set* $i = i + 1$.

e) *For* $j = 0$ *to* i:

1) *Set* $c = c_j \bmod s$;

2) *If* j *is even*, *find all integer roots of the polynomials* $x^2 - (c\ s + a_j + b_j\ a_1)\ x + a_j\ b_j\ n$.

If $c > 0$, *also find all integer roots* $x^2 - ((c - s)\ s + a_j + bj\ a_1)\ x + a_j\ b_j\ n$.

Otherwise:

i) *set* $t = \lceil (2\, a_j\, b_j - c) / s \rceil$;

ii) *set* $c = c + t\, s$;

iii) *if* $c \leq n/s^2 + a_j\, b_j$, *find all integer roots of the polynomial* $x^2 - (c\, s + a_j + b_j\, a_1)\, x + a_j\, b_j\, n$.

3) *If* $a_j \neq 0$, *for each positive integer root α of the polynomials in step* e) 2):

i) *set* $d = \alpha \operatorname{div} a_j$;

ii) *if* $(d \neq 0$ *or* $d \neq 1)$ *and* $(n \bmod d = 0)$ *and* $(d \bmod s = 1)$, *return* "*true*", *divisor d*, *and* stop.

Otherwise:

i) *if* $c_j = 0$, *set* $y = 0$. *Otherwise, set* $y = (c_j - s)\ div\ b_j$;

ii) *set* $d = n \operatorname{div} (y\, s + a_1)$;

iii) *if* $(d \neq 0$ *or* $d \neq 1)$ *and* $(n \bmod d = 0)$ *and* $(d \bmod s = 1)$, *return "true", divisor d*, *and stop*.

f) *Return false*

CATATAN 1 Algoritma ini digunakan pada D.5 dengan $s = \operatorname{lcm}(F_1, F_2)$ dengan F_1 membagi $n - 1$ dan F_2 membagi $n + 1$, dan seterusnya $\gcd(n, s) = 1$ terpenuhi secara trivial

CATATAN 2 Pada langkah 3) dalam kasus $a_j = 0$, maka dari identitas pada halaman 333 Acuan [12] bahwa $b_j \neq 0$. Dengan demikian, div b_j yang berkorespondensi tidak menghasilkan pembagian oleh galat nol.

Lampiran E
(normatif)
Contoh pembangkitan prima

E.1 Umum

Dua contoh yang tersedia menggunakan mekanisme di B.2 untuk membangkitkan bilangan prima 1024-bit dengan dua set bit atas dan bawah, yaitu bilangan tersebut kongruen dengan 3 mod 4 dan terletak pada interval $(2^{1023} + 2^{1022}, 2^{1024})$. Dalam kedua kasus tersebut, kandidat non-prima dikesampingkan dengan menemukan faktor-faktor kecil melalui *sieving* atau melakukan iterasi tes Miller-Rabin dengan nilai saksi $b = 2$ (sehingga tidak ada panggilan tambahan yang dilakukan ke RBG untuk pengetesan Miller-Rabin dari nilai non-prima).

RBG deterministik yang digunakan untuk menghasilkan kandidat bilangan prima adalah Hash_DRBG yang didefinisikan dalam ISO/IEC 18031:2011, C.2, dengan SHA-256 sebagai fungsi hash. Kedua contoh dibangkitkan dengan nilai entropi 256-bit dan string personalisasi 128-bit yang sama:

Entropi_input =

```
9F25EC74 6A7616D3 CB2B0779 5A9DB21C BB5BD922 D6E2AC5A C0554BC0 46FA692A
```

Personalization_string = `8C261FD6 7E844588 2FB0EF90 7CBFB59C`

Dengan input di atas, fungsi Instantiate_Hash_DRBG membangkitkan nilai status 440-bit:

```
V = 64690D2C 271E5F32 3C0F39B2 C6838AAE 348D6DB4 F30842A3 B8158994 37EDE8A3
A11E21A2 502FB5B6 C297F71C BDB34190 FA91E4AA 2DE9C3
C = B342BFFF FB17AA21 C14C57C3 104DB01A 3A94542A C4AF4DBF 58FB2027 CD252B24
E368CBD5 EAD0C2EE 1043236E 933337F7 608428BA A55988
```

Bilangan acak dibuat menggunakan metode konversi sederhana yang dispesifikasikan dalam C.2.

E.2 Contoh pencarian inkremental Miller-Rabin

Setelah algoritma Instantiate_Hash_DRBG, fungsi Hash_DRBG dipanggil untuk membangkitkan integer x 1.022-bit:

```
022CE48F D309055C 360C8890 9B501103 B9773C95 407A2373 DA250D8F 17E5609B
8C1E7B19 B89C609F 7A03A11D 46593DB1 0822B44C 47C7D283 F460CF4E 700F503F
730D43E5 EDD183AD 7800ACD0 67144CCF 9F2936D6 141A8CF3 8A3BAF85 64A9EC7D
71DEA720 C9F61CCD 4BF72CDD 6F2776E3 D6CC2234 E8A0CC6E EC7C3AD4 6FEE41D8
```

Dua *most significant* dan dua *least significant* bit dari *x* ditetapkan, sehingga menghasilkan nilai 1024-bit:

```
C22CE48F D309055C 360C8890 9B501103 B9773C95 407A2373 DA250D8F 17E5609B
8C1E7B19 B89C609F 7A03A11D 46593DB1 0822B44C 47C7D283 F460CF4E 700F503F
730D43E5 EDD183AD 7800ACD0 67144CCF 9F2936D6 141A8CF3 8A3BAF85 64A9EC7D
71DEA720 C9F61CCD 4BF72CDD 6F2776E3 D6CC2234 E8A0CC6E EC7C3AD4 6FEE41DB
```

Nilai yang dihasilkan bertambah 4 hingga bilangan prima tersebut:

```
C22CE48F D309055C 360C8890 9B501103 B9773C95 407A2373 DA250D8F 17E5609B
8C1E7B19 B89C609F 7A03A11D 46593DB1 0822B44C 47C7D283 F460CF4E 700F503F
730D43E5 EDD183AD 7800ACD0 67144CCF 9F2936D6 141A8CF3 8A3BAF85 64A9EC7D
71DEA720 C9F61CCD 4BF72CDD 6F2776E3 D6CC2234 E8A0CC6E EC7C3AD4 6FEE48A3
```

ditemukan.

E.3 Contoh pencarian acak Miller-Rabin

Nilai entropi dan string personalisasi yang sama digunakan sebagai input ke algoritma Instantiate_Hash_DRBG yang menghasilkan nilai awal x yang sama di E.2. Dengan cara yang sama, x dimodifikasi menjadi integer 1.024-bit dengan dua *most significant* dan dua *least significant* bit. Hash_DRBG dipanggil 166 kali sebelum nilai prima tersebut:

```
CF400E9A 59E57803 35CE354D C08F296C 31B07B95 4ACF7F43 89049BB1 D8746522
179666F7 A0D78048 17AE9892 18C9D245 6ED03168 A0FB255C B3E94C71 8C5356E0
381399FF 64679BF0 DB435BCE 8F06B79C B7A00BF8 7D00571A 9CBFFA51 5D4CF3D8
637CEA0A 425235CC B6431898 4EF9A34D D417DA9C 7759568E 029D76C6 A6C8730F
```

ditemukan.

Bibliografi

[1] ANSI X9.80-2010, *Prime number generation, primality testing, and primality certificates.*

[2] Atkin A.O.L., Morain F., *Elliptic curves and primality proving*. Mathematics of Computation **61**, 1993, pp. 29-68

[3] Baillie Robert, Wagstaff Samuel S.Jr, *Lucas Psuedoprimes*. Mathematics of Computation **35**, 1980, pp 1391-1417.

[4] Blake Ian F., Seroussie Gadiel, Smart Nigel P., *Elliptic curves in cryptography*, third printing, Cambridge University Press, 2000

[5] Brandt J., Damgård I. *On generation of probable primes by incremental search*. Proceedings CRYPTO 92, LNCS 740, Springer, 1993, pp. 358-370

[6] John Brillhart D. H., Lehmer, J. L. Selfridge. *New Primality Criteria and Factorizations of* $2^m \pm 1$. Mathematics of Computation **29**(130) 1975, pp. 620-647

[7] Coppersmith Don, *Finding a Small Root of a Bivariate Integer Equation; Factoring with high bits known*. Advances in Cryptography – EUROCRYPT 1996, pp 178-189.

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

[9] Damgård Ivan, Landrock Peter, Pomerance Carl, *Average case error estimates for the strong probable prime test*. Mathematics of Computations **61**(203), 1993, pp. 177-194

[10] NIST *Digital Signature Standard* (*DSS*), Federal Information Processing Standards Publication 186-4, July 2013.

[11] Elkies N.D. *Elliptic and modular curves over finite fields and related computational issues*. Computational Perspectives on Number Theory, volume 7 of AMS/IP Stud. Adv. Math. Amer. Soc., Providence, RI, 1998.

[12] Lenstra H. W., Divisors in Residue Classes. *Mathematics of Computation* **42** (165), 1984, pp. 331-340.

[13] Marcus Daniel, *Number Fields*, third printing, Springer-Verlag, 1995.

[14] Nemec M., Sys M., Svenda P., Klinec D., Matyas V. *The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli*. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp 1631-1648.

[15] Pomerance Carl *Are there Counter-examples to the Baillie – PSW Primality test?* 1984

[16] Schoof Rene, *Counting points on elliptic curves over finite fields*. J. Theory. Nombres Bourdeaux, **7**:219-254, 1995.

[17] Stinson Douglas R., *Cryptography*, *Theory and Practice*, 2nd edition, Chapman & Hall/CRC, 2002

[18] Shawe-Taylor J., *Generating strong primes*. Electronics Letters (22) **16**, 1986, pp. 875-877

[19] Washington L., Elliptic Curves: Number Theory and Cryptography, CRC Press, 2003, 2nd ed. 2008.

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

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 $a = (a\ div\ n) \cdot n + r$.
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
$\text{Jacobi}(a, N)$	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. $\text{sgn}(D) = 1\ if\ D \geq 0$ 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 $N \bmod p = 0$ 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:

— $b^s \bmod N = 1$

— $b^s \bmod N = N - 1$; atau

— $(b^s)^{2^i} \bmod N = N - 1$, *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 $= 1$ *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 $N \, mod \, 4 = 1$.

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 ≥ 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 ≥ 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:

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

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 $C_1 = (N, q_i, a_i)$, 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 $\gcd(n-1, e) = 1$ 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 $N, N + 2, N + 4, \ldots, Max$ 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 $j = \lceil j/3 \rceil + 1 \,(\text{atau}\, \lceil j/2 \rceil + 1)$ 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 $p = x + ((1 - \text{x}) \bmod 2q), t = (p - 1) \operatorname{div} q$ (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 $p < 2j - 2q$;

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 $j_{i-1} = \lceil j_i/3 \rceil + 1$ 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	-101	-104

Table A.2 — Average-case error estimates for *t* Miller-Rabin primality tests (*k* ≥ 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 $\gcd(p-1, e) = 1$ 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 $0 \leq r < m$. Given a primality test T, a k-bit prime N satisfying $N \bmod m = r$ and $\gcd(N-1, e) = 1$ 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 $p, p + 2m, p + 4m, \ldots, p + 2Jm$ 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 mod m lies in some subset S of $\{1, 2, \ldots, m-1\}$, 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 mod 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 $r = \left(({p_2}^{-1} \bmod p1) \cdot p_2 - ({p_1}^{-1} \bmod p_2) \cdot p_1\right) \bmod m$. 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 ≠ q) as follows. If m is prime to $2q$, set $m_0 = 2qm,\ r_0 = (\,((2q)^{-1} \bmod m) \cdot 2q + (m^{-1} \bmod 2q) \cdot m) \bmod m_0$. If 2 divides m (note that r should be odd to ensure p is odd) set $m_0 = qm,\ r_0 = \left((q^{-1} \bmod m) \cdot q + (m^{-1} \bmod q) \cdot m\right) \bmod m_0$ 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 $p = x + (r_0 - x) \bmod m_0, t = (p - 1)\ div\ q$.

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 $p < 2k - m_0$, 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 $p,\ p + m_0, p + 2\,m_0\ ,\ \ldots,\ p + J\,m_0$ 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 $2^{k-1} \le p < 2^k$. 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, $(b_0, b_1, \ldots, b_{m-1})$;

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

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, $b_0, b_1, \ldots, b_{m-1}$.

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

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, $b_0, b_1, \ldots, b_{m+l-1}$.

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

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 $S = Y_o, Y_o + h, \ldots, Y_o + J \cdot h$ 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 $i = 0,1,\ldots,J, \text{set } A[i] = 0$.

c) For each p_j in B, do:

1) set $i = (-h - 1Y0) \bmod pj$.

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 $R = s' F + r$ 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, ... 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 ${a_1}^{N-1} \bmod N \neq 1$,

return "certificate invalid, N composite" and stop.

3) If $\gcd(\ ({a_1}^{(N-1)/q_i} - 1)\ mod\ N, N) \neq 1, N$,

return "certificate invalid, N composite" and stop;

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

return "certificate invalid" and stop.

4) while $(R \bmod q_i\ ) = 0$:

i) set $R = R \operatorname{div} q_i$;

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 $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$ 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 $\{q_1, q_2, \ldots, q_n\}$.. 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 $N < F^3 + r \cdot F^2 + r \cdot F + 1 = (F + 1)(F^2 + (r - 1) \cdot F + 1)$..

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 $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$.

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) $\text{Jacobi}(D, N) = -1$;

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

then $\text{Lucas}(D, N + 1, N) = 0$.

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 ($\text{Jacobi}(D, N) \neq -1$ or $\gcd(N, Q) \neq 1$):

   1) If $\text{Jacobi}(D, N) = 0$ and $D \bmod N \neq 0$, return "N composite" and stop.

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

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

d) If $\text{Lucas}(D, N + 1, N) = 0$, 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 $\text{Lucas}(D, N + 1, N)$ 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 $\text{Jacobi}(1, N)$ is always 1 and ranges over the set $\{5, -7, 9, -11, 13, -15, 17, \ldots\}$

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 $D \bmod 4 = 1$ such that:

1) $\text{Jacobi}(D, N) = -1$;
2) $\text{Lucas}(D, N+1, N) = 0$; dan
3) $\gcd(\text{Lucas}(D, (N+1)/q, N), N) = 1$;

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 $\text{Lucas}(D, N+1, N)$ 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 ($\text{Jacobi}(D, N) \neq -1$ and $j < T$):
      i) If $\text{Jacobi}(D, N) = 0$:

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

         Otherwise, set $j = j + 1$.

      ii) Set $D = -\text{sgn}(D) \cdot (\text{sgn}(D) \cdot D + 2)$, i.e. select the next value for D.
   4) If $j = T$, return "test inconclusive" and stop.
   5) If $\text{Lucas}(D, N + 1, N) \neq 0$:

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

      Otherwise, set $D = -\text{sgn}(D) \cdot (\text{sgn}(D) \cdot D + 2)$ and go to step b) 3).
   6) If $\gcd(\text{Lucas}(D, (N + 1)/q_i, N), N) = 1$, set $D_i = D$.

      Otherwise, if $\text{Lucas}(D, (N + 1)/q_i, N) \neq 0$, return "$N$ composite" and stop.

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

c) Return "N prime." If certificate is requested, return certificate with values $\{(N, q_1, D_1), (N, q_2, D_2), \ldots, (N, q_n, D_n)\}$. 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
$11 \le q \le 23$	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 $\text{acobi}(D, N) = -1$ and $\gcd(Q, N) = 1$,, then $\text{Lucas}(D, N + 1, N) = 0$; that is, if $\text{Lucas}(D, N + 1, N) \neq 0$, $\text{Jacobi}(D, N) = -1$,, and $\gcd(Q, N) = 1$, then *N* cannot be prime;

2) if $1 < \gcd(Q, N) < N$, then *N* is not prime. Thus if $\text{Lucas}(D, N + 1, N)$, $\text{Jacobi}(D, N) = -1$ and $\gcd(Q, N) < N$, or equivalently $Q \bmod N \neq 0$, 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 $\text{Jacobi}(D, N) = 0$:

If $D \bmod N \neq 0$, return "certificate invalid, N composite" and stop.

Otherwise, return "certificate invalid" and stop.

3) If $\text{Jacobi}(D, N) = 1$, return "certificate invalid" and stop.

4) If $\text{Lucas}(D, N + 1, N) \neq 0$:

If $Q \bmod N \neq 0$ where $Q = (1 - D) \text{ div } 4$, return "certificate invalid, N composite" and stop.

Otherwise, return "certificate invalid" and stop.

5) If $gcd(\text{Lucas}(D_i, (N + 1)/q_i, N), N) \neq 1$:

If $\text{Lucas}(D_i, (N + 1)/q_i, N) \neq 0$, return "certificate invalid, N composite" and stop.

Otherwise, return “certificate invalid” and stop.

6) While $(R \bmod q_i) = 0$:

i) Set $R = R\ div\ q_i$.

ii) Set $F = F \cdot qi$.

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 $\{(N, q_1, D_1), (N, q_2, D_2), \ldots, (N, q_n, D_n)\}$ 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 $\{q_1, q_2, \ldots, q_n\}$ 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 $\{(N, q_1, a_1), (N, q_2, a_2), \ldots, (N, q_n, a_n)\}$.

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 $\{5, -7, 9, -11, 13, -15, 17, \dots\}$ such that:

1) $\text{Jacobi}(D, N) = -1$;

2) $\text{Lukas}(D, N + 1, N) = 0$; and

3) $\gcd(\text{Lucas}(D, (N + 1)/r, N), N) = 1$;

then any positive divisor d of N shall satisfy: $d \bmod \text{lcm}(F_1, F_2) = 1$ or $d \bmod \text{lcm}(F_1, F_2) = N \bmod \text{lcm}(F_1, F_2)$.

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

a) $\text{lcm}(F_1, F_2) > N$;

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

c) there does not exist a proper divisor d satisfying $d \bmod \text{lcm}(F_1, F_2) = 1$.

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, $N \bmod \text{lcm}(F_1, F_2) = N^2 \bmod \text{lcm}(F_1, F_2)$ (as N is a product of two proper divisors), i.e. $N \bmod \text{lcm}(F_1, F_2) = 1$ which contradicts c).

NOTE 4 If $\text{lcm}(F_1, F_2) > N^{1/3}$, 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 $F2 = r_1^{f_1} r_2^{f_2} \dots r_n^{f_n}$ where $N - 1 = F_1 R_1 \; dan \; N + 1 = F_2 R_2$, the result above shall be applied as follows.

a) If $\text{lcm}(F_1, F_2) \leq N^{1/3}$, 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 $N + 1 = F_2 R_2$.

   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 $\{(N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$, proceed to the next step.

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

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

If $N \bmod (N \bmod \text{lcm}(F_1, F_2)) = 0$, return "$N$ composite" and stop.

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

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

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

Otherwise, return "N composite" and stop.

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

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

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

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 $\{(N, r_1, D_1), (N, r_2, D_2), \ldots, (N, r_m, D_m)\}$..

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 $\text{lcm}(F_1, F_2) \leq N^{1/3}$, return "certificate invalid" and stop.

d) If $\text{lcm}(F_1, F_2) > N$, accept the certificate, return "N prime" and stop.

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

If $N \bmod (N \bmod \text{lcm}(F_1, F_2)) = 0$, 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 $s = \text{lcm}(F_1, F_2)$ 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 $d \bmod F_1 = 1$ and

$d \bmod F_2 = \pm 1$ (theorems 4 and 16 of Reference [6]) from which it follows $d \bmod \mathrm{lcm}(F_1, F_2) = 1$ or $N \bmod \mathrm{lcm}(F_1, F_2)$.

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 $P = (t/r)\, Q$ 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.

e) Verify P has order r, i.e. $P \neq 0$ (the zero point on E_N) and $r \cdot P = 0$. If not, return "certificate invalid" and stop. Otherwise, accept the certificate, return "N prime" and stop.

D.7 Cornacchia's algorithm

Given an integer $M > 0$, and an integer δ with $1 \leq \delta < M$ and $\gcd(\delta, M) = 1$, D.7 gives an algorithm which finds a pair of integers (V, W) satisfying $M = V^2 + \delta W^2$ (provided a solution exists).

a) Attempt to find a solution, r_1, of the congruence $(r_1)^2 \equiv -\delta \ (mod\ M)$ that also satisfies the inequalities $0 \leq r1 \leq (M/2)$.. If no solution is found, abort (there is no solution of the Diophantine equation).

b) Set $r_0 = M$, and, using the value of r_1 determined in step a), consider the sequence $r_0, r_1, r_2 \ldots$ that is recursively defined by setting $r_i + 2 = r_i \ mod\ r_{i+1}$ for integers $i \geq 0$. Determine the least positive integer k for which $(r_k)^2 < M$.

c) Set $s = ((M - (r_k)^2)/\delta\ )^{1/2}$. If s is an integer, return $(V, W) = (r_k, s)$; otherwise abort (there is no solution of the Diophantine equation).

D.8 Complex multiplication (CM) method

The algorithm given here generates elliptic curves of known order over Z_N where N is prime. Let $\Delta = \{-3, -4, -7, -8, -11, -15, -19, -20, -23, -24, -31, -35, -39, -40, -43, -47, \ldots\}$ be the set of fundamental quadratic discriminants, i.e. the set of negative integers d with $d \bmod 4 = 1$, $d \bmod 16 = 8$, or $d \bmod 16 = 12$.

a) Pick a discriminant d in Δ, and put $D = -d$.

b) Using Cornacchia's algorithm, D.7, find integers V and W such that $4N = V^2 + DW^2$. If no such pair of integers exists, go to step a) and select a new discriminant d.

c) If $d = -3$, set:

$$T = \{N + 1 - V, N + 1 + V, N + 1 - 1/2 \cdot (-V + 3W), N + 1 + 1/2 \cdot (-V + 3W), N + 1 - 1/2 \cdot (V + 3W), N + 1 + 1/2 \cdot (V + 3W)\}.$$

If $d = -4$, set:

$$T = \{N + 1 - V, N + 1 + V, N + 1 - 2W, N + 1 + 2W\}.$$

If $d < -4$, set:

$T = \{N + 1 - V, N + 1 + V\}$.

d) For each t in T, use the CM method to construct the elliptic curve E_t that reduced modulo N has order t. Return the set of tuples $\{ (Et, t) : t \text{ in } T \}$ and stop.

The curve in E_t in step d) can be constructed using the CM method as specified in ISO/IEC 15946-1.

D.9 Testing for perfect squares

D.9.1 General

This clause contains two methods for determining whether an integer n is a perfect square. The first method is probabilistic whereas the second method is deterministic. The first method should be used to determine if an integer is not a perfect square.

D.9.2 Probabilistic method

This method makes use of the Jacobi symbol as described in 3.4, and takes as input an integer n and an odd prime p such that $\gcd(n, p) = 1$. The method returns "n passes this iteration" or "n is not a perfect square".

To determine whether or not n is perfect square, check if $\text{Jacobi}(n, p) = 1$. If so, return "n passes this iteration". Otherwise, return "n is not a perfect square".

NOTE This test is probabilistic in nature and a non-square will pass, say, 20 iterations of this test with respect to 20 different primes with probability 2^{-20}.

If an integer passes several iterations of this test, the deterministic test should be applied for a definitive answer.

D.9.3 Deterministic method

The method provided here is based on Newton's method. It takes as input an m-bit integer n and returns "true" if n is a perfect square and "false" otherwise.

To determine whether or not n is perfect square, do as follows.

a) Set $b = m \text{ div } 2, x = 2^{b+1}$ and $y = (x + (n \text{ div } x)) \text{ div } 2$.

b) While $y < x$:

1) set $x = y$;

2) set $y = (x + (n \text{ div } x)) \text{ div } 2$.

c) If $x^2 = n$, return "true" and the root x, else return "false".

D.10 Jacobi symbol

On input of an integer a and odd integer n, returns the value $\text{Jacobi}(a, n)$.

a) If $\gcd(a, n) \neq 1$, return 0.

b) Set $a = a \bmod n$.

c) Write $a = 2^e b$, where b is odd and $e \geq 0$.

d) If $n \bmod 8 = 1$ or 7, set $v = 1$. Otherwise, set $v = (-1)^e$.

e) Set $a = b$.

f) While $a \neq 1$:

1) if $a \, mod \, 4 = 3$ and $n \, mod \, 4 = 3$, then set $v = -v$;

2) set $r = n \bmod a$;

3) write $r = 2^f s$, where s is odd and $f \geq 0$;

4) if $a \, mod \, 8 = 3$ or 5, set $v = (-1)^f \cdot v$;

5) set $n = a$ and $a = s$.

g) Return v.

D.11 Lucas symbol

On input of a quadratic discriminant D, a positive integer K, and an odd integer N, returns the value $\mathrm{Lucas}(D, K, N)$.

a) Set $U = 1$ and $V = 1$.

b) Let $K_r K_r - 1 \ldots K_0$ be the binary expansion of K where K_r is 1. The bit-length of K is $r + 1$.

c) For i from $r - 1$ to 0:

1) set $(U, V) = ((UV) \bmod N, (V^2 + DU^2)/2 \bmod N)$;

2) if $K_i = 1$, set $(U, V) = ((U + V)/2 \bmod N, (V + DU)/2 \bmod N)$.

d) Return U.

NOTE Steps c) 1) and c) 2) call for evaluations of the form $W/2 \bmod N$ where W is an integer and N is assumed to be an odd integer. If W is odd, then $W/2 \bmod N$ can be calculated as $((W + N) \operatorname{div} 2) \bmod N$.

D.12 Finding divisors in residue classes (Lenstra).

Given an integer n and a modulus $s > n^{1/3}$ with $\gcd(n, s) = 1$, the algorithm returns “true” if there exists a non-trivial divisor d of n satisfying $d \bmod s = 1$. This algorithm is a special case of a more general result due to Lenstra.[12]

a) Set $a_0 = s$, $b_0 = 0$, and $c_0 = 0$.

b) Set $a_1 = n \bmod s$, $b_1 = 1$, and $c1 = ((n - a_1) \operatorname{div} s) \bmod s$.

c) Set $i = 1$.

d) While ($ai \neq 0$):

1) if i is even and $a_i - 1 \bmod a_i = 0$, set $a_i + 1 = a_i$;

Otherwise, set $a_i + 1 = a_i - 1 \bmod a_i$.

2) set $q = (a_i - 1 - a_i + 1) \operatorname{div} a_i$;

3) set $b_i + 1 = b_i - 1 - q \, b_i$;

4) set $c_i + 1 = (c_i - 1 - q\ c_i) \bmod s$;

5) increment i, i.e. set $i = i + 1$.

e) For $j = 0\ to\ i$:

1) Set $c = c_j \bmod s$;

2) If j is even, find all integer roots of the polynomials $x^2 - (c\ s + a_j + b_j\ a_1)\ x + a_j\ b_j\ n$.

If $c > 0$, also find all integer roots $x^2 - ((c - s)\ s + a_j + bj\ a_1)\ x + a_j\ b_j\ n$.

Otherwise:

i) set $t = \lceil (2\ a_j\ b_j - c) / s \rceil$;

ii) set $c = c + t\ s$;

iii) if $c \leq n/s^2 + a_j\ b_j$, find all integer roots of the polynomial $x^2 - (c\ s + a_j + b_j\ a_1)\ x + a_j\ b_j\ n$.

3) If $a_j \neq 0$, for each positive integer root α of the polynomials in step e) 2):

i) set $d = \alpha \operatorname{div} a_j$;

ii) if ($d \neq 0$ or $d \neq 1$) and ($n \bmod d = 0$) and ($d \bmod s = 1$), return “true”, the divisor d, and stop.

Otherwise:

i) if $c_j = 0$, set $y = 0$. Otherwise, set $y = (c_j - s) \operatorname{div} b_j$;

ii) set $d = n \operatorname{div} (y\ s + a_1)$;

iii) if ($d \neq 0$ or $d \neq 1$) and ($n \bmod d = 0$) and ($d \bmod s = 1$), return “true”, the divisor d, and stop.

f) Return false.

NOTE 1 This algorithm is used in D.5 where $s = \operatorname{lcm}(F_1, F_2)$ with F_1 dividing $n - 1$ and F_2 dividing $n + 1$, and so $\gcd(n, s) = 1$ is trivially satisfied.

NOTE 2 In step 3) in case $a_j = 0$, it follows from identities on page 333 of Reference [12] that $b_j \neq 0$. Thus, the corresponding div b_j does not result in a division by zero error.

Annex E
(informative)
Prime generation examples

E.1 General

Two examples are provided using the mechanisms in B.2 to generate a 1 024-bit prime number with both the top and low two bit sets, i.e. the number is congruent to 3 mod 4 and lies in the interval ($2^{1\,023} + 2^{1\,022}$, $2^{1\,024}$). In both cases, non-prime candidates were ruled out by finding small factors through sieving or performing an iteration of the Miller-Rabin test with the witness value $b = 2$ (and so no additional calls were made to the RBG for Miller-Rabin testing of non-prime values).

The deterministic RBG that is used to produce candidate primes is the Hash_DRBG defined in ISO/IEC 18031:2011, C.2, with SHA-256 as the hash function. Both examples are generated with the same 256-bit entropy value and 128-bit personalization string:

Entropy_input =

9F25EC74 6A7616D3 CB2B0779 5A9DB21C BB5BD922 D6E2AC5A C0554BC0 46FA692A
Personalization_string = 8C261FD6 7E844588 2FB0EF90 7CBFB59C

With the above input, the Instantiate_Hash_DRBG function generates the 440-bit state values:

V = 64690D2C 271E5F32 3C0F39B2 C6838AAE 348D6DB4 F30842A3 B8158994 37EDE8A3
A11E21A2 502FB5B6 C297F71C BDB34190 FA91E4AA 2DE9C3
C = B342BFFF FB17AA21 C14C57C3 104DB01A 3A94542A C4AF4DBF 58FB2027 CD252B24
E368CBD5 EAD0C2EE 1043236E 933337F7 608428BA A55988

Random numbers are constructed using the simple conversion method specified in C.2.

E.2 Miller-Rabin incremental search example

After the Instantiate_Hash_DRBG algorithm, the Hash_DRBG function is called to generate the 1 022-bit integer x:

022CE48F D309055C 360C8890 9B501103 B9773C95 407A2373 DA250D8F 17E5609B
8C1E7B19 B89C609F 7A03A11D 46593DB1 0822B44C 47C7D283 F460CF4E 700F503F
730D43E5 EDD183AD 7800ACD0 67144CCF 9F2936D6 141A8CF3 8A3BAF85 64A9EC7D
71DEA720 C9F61CCD 4BF72CDD 6F2776E3 D6CC2234 E8A0CC6E EC7C3AD4 6FEE41D8

The two most significant and least two significant bits of x are set, resulting in the 1 024-bit value:

C22CE48F D309055C 360C8890 9B501103 B9773C95 407A2373 DA250D8F 17E5609B
8C1E7B19 B89C609F 7A03A11D 46593DB1 0822B44C 47C7D283 F460CF4E 700F503F
730D43E5 EDD183AD 7800ACD0 67144CCF 9F2936D6 141A8CF3 8A3BAF85 64A9EC7D
71DEA720 C9F61CCD 4BF72CDD 6F2776E3 D6CC2234 E8A0CC6E EC7C3AD4 6FEE41DB

The resulting value is incremented by 4 until the prime:

```
C22CE48F D309055C 360C8890 9B501103 B9773C95 407A2373 DA250D8F 17E5609B
8C1E7B19 B89C609F 7A03A11D 46593DB1 0822B44C 47C7D283 F460CF4E 700F503F
730D43E5 EDD183AD 7800ACD0 67144CCF 9F2936D6 141A8CF3 8A3BAF85 64A9EC7D
71DEA720 C9F61CCD 4BF72CDD 6F2776E3 D6CC2234 E8A0CC6E EC7C3AD4 6FEE48A3
```

is found.

E.3 Miller-Rabin random search example

The same entropy value and personalization string are used as inputs to the Instantiate_Hash_DRBG algorithm resulting in the same initial value x in E.2. In a similar fashion, x is modified to a 1 024-bit integer with the two most significant and least significant bits set. Hash_DRBG is invoked 166 times before the prime value:

```
CF400E9A 59E57803 35CE354D C08F296C 31B07B95 4ACF7F43 89049BB1 D8746522
179666F7 A0D78048 17AE9892 18C9D245 6ED03168 A0FB255C B3E94C71 8C5356E0
381399FF 64679BF0 DB435BCE 8F06B79C B7A00BF8 7D00571A 9CBFFA51 5D4CF3D8
637CEA0A 425235CC B6431898 4EF9A34D D417DA9C 7759568E 029D76C6 A6C8730F
```

is found.

Bibliography

[1] ANSI X9.80-2010, *Prime number generation, primality testing, and primality certificates.*

[2] Atkin A.O.L., Morain F., *Elliptic curves and primality proving*. Mathematics of Computation **61**, 1993, pp. 29-68

[3] Baillie Robert, Wagstaff Samuel S.Jr, *Lucas Psuedoprimes*. Mathematics of Computation **35**, 1980, pp 1391-1417.

[4] Blake Ian F., Seroussie Gadiel, Smart Nigel P., *Elliptic curves in cryptography*, third printing, Cambridge University Press, 2000

[5] Brandt J., Damgård I. *On generation of probable primes by incremental search*. Proceedings CRYPTO 92, LNCS 740, Springer, 1993, pp. 358-370

[6] John Brillhart D. H., Lehmer, J. L. Selfridge. *New Primality Criteria and Factorizations of* $2^m \pm 1$. Mathematics of Computation **29**(130) 1975, pp. 620-647

[7] Coppersmith Don, *Finding a Small Root of a Bivariate Integer Equation; Factoring with high bits known*. Advances in Cryptography – EUROCRYPT 1996, pp 178-189.

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

[9] Damgård Ivan, Landrock Peter, Pomerance Carl, *Average case error estimates for the strong probable prime test*. Mathematics of Computations **61**(203), 1993, pp. 177-194

[10] NIST *Digital Signature Standard* (*DSS*), Federal Information Processing Standards Publication 186-4, July 2013.

[11] Elkies N.D. *Elliptic and modular curves over finite fields and related computational issues*. Computational Perspectives on Number Theory, volume 7 of AMS/IP Stud. Adv. Math. Amer. Soc., Providence, RI, 1998.

[12] Lenstra H. W., Divisors in Residue Classes. *Mathematics of Computation* **42** (165), 1984, pp. 331-340.

[13] Marcus Daniel, *Number Fields*, third printing, Springer-Verlag, 1995.

[14] Nemec M., Sys M., Svenda P., Klinec D., Matyas V. *The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli*. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp 1631-1648.

[15] Pomerance Carl *Are there Counter-examples to the Baillie – PSW Primality test?* 1984

[16] Schoof Rene, *Counting points on elliptic curves over finite fields*. J. Theory. Nombres Bourdeaux, **7**:219-254, 1995.

[17] Stinson Douglas R., *Cryptography*, *Theory and Practice*, 2nd edition, Chapman & Hall/CRC, 2002

[18] Shawe-Taylor J., *Generating strong primes*. Electronics Letters (22) **16**, 1986, pp. 875-877

[19] Washington L., Elliptic Curves: Number Theory and Cryptography, CRC Press, 2003, 2nd ed. 2008.

Informasi pendukung terkait perumus standar

[1] Komite Teknis Perumusan SNI

Komite Teknis 35-04 Keamanan Informasi, Keamanan Siber, dan Perlindungan Privasi

[2] Susunan keanggotaan Komite Teknis Perumusan SNI

Ketua : Soetedjo Joewono
Sekretaris : Didik Utomo
Anggota :
1 Pedro Libratu Putu Wirya
2 Zaenal Arifin
3 Wisnoe Prasetyo Pribadi
4 Bisyron Wahyudi
5 Satriyo Wibowo
6 Sarwono Sutikno
7 Chandra Yulistia
8 Pratama Dahlian Persadha
9 Sugi Guritman
10 Bety Hayat Susanti
11 Sari Agustini Hafman

[3] Konseptor Rancangan SNI

Gugus Kerja 2 Kriptografi dan Mekanisme Keamanan – Komtek 35-04:

Ketua : Sari Agustini Hafman
Wakil Ketua : Sugi Guritman
Sekretaris : Novita Angraini
Anggota :
1. Pinuji Prasetyaningtyas
2. Bety Hayat Susanti
3. Dory Marselly
4. Afifah
5. Fadila Paradise
6. Freddy Ajax Pratama

[4] Sekretariat pengelola Komite Teknis Perumusan SNI

Direktorat Kebijakan Teknologi Keamanan Siber dan Sandi
Badan Siber dan Sandi Negara (BSSN)