

電子政府情報セキュリティ技術開発事業

素数生成アルゴリズムの調査・開発

調査報告書



平成 15 年 2 月

情報処理振興事業協会

セキュリティセンター

目次

1	はじめに	4
1.1	暗号における素数の重要性	4
1.2	本調査報告書の目的	4
1.3	本調査報告書の構成	5
2	確率的素数判定法	5
2.1	概要	5
2.2	Fermat 法	6
2.3	Solovay-Strassen 法	7
2.4	Miller-Rabin 法	8
2.5	Lucas-Lehmer 法	9
2.6	Frobenius-Grantham 法	10
3	確定的素数判定法	11
3.1	概要	11
3.2	試行割算法	12
3.3	Proth-Pocklington-Lehmer-Selfridge(PPLS) 法	12
3.4	Kraitchik-Lehmer 法	13
3.5	Elliptic Curve Primality Proving (ECP) 法	14
3.6	Cohen-Lenstra 法	15
3.7	Agrawal-Kayal-Saxena-Lenstra(AKSL) 法	16
3.8	AKS での予想に基づく方法	17
4	素数生成法	17
4.1	概要	17
4.2	擬似乱数生成法	18
4.3	素数判定法に基づく方法	18
4.3.1	概要	18
4.3.2	Random Choice	19
4.3.3	Incremental Search	20
4.3.4	FIPS 186-2 DSS の方法	20
4.4	直接素数を構成する方法	21
4.4.1	Shawe-Taylor 法	21
4.4.2	Maurer 法	22
5	各種アルゴリズムの比較	22
5.1	概要	22
5.2	計算量比較	23
5.3	計算機資源比較	26
5.4	実用性	27

6	実装性評価	27
6.1	高速化手法	27
6.1.1	べき乗剰余演算	28
6.1.2	Miller-Rabin 法における基底のとり方	30
6.1.3	小さな素数による試行割算	30
6.2	速度評価	31
6.2.1	べき乗剰余演算速度	31
6.2.2	2のべき乗剰余演算速度	32
6.2.3	素数生成速度	33
6.3	高速化のための最適設定	34
6.4	フリーソフトウェアとの比較	35
7	暗号鍵としての素数の安全性条件	36
7.1	RSA 暗号	36
7.1.1	素因数分解アルゴリズム	36
7.1.2	素因数分解に対して安全な素数の要件	37
7.2	DSA 署名	38
7.3	楕円曲線暗号	38
8	標準化動向	39
8.1	ANSI	39
8.2	FIPS	39
8.3	CRYPTREC	39
9	電子政府セキュリティシステムにおける素数生成法の要件について	40
9.1	素数判定の確実性および素数生成の確実性と一様性	40
9.2	素数生成の実装性	41
9.3	素数を暗号鍵として用いる場合の安全性条件	41
9.4	今回開発すべきソフトウェアの要求仕様	42
9.4.1	素数判定ソフトウェア	42
9.4.2	素数生成ソフトウェア	43
9.4.3	パラメータ安全性判定ソフトウェア	43
9.5	CMVP への利用に向けて	44
10	まとめ	45

1 はじめに

1.1 暗号における素数の重要性

現在利用されているほとんどの公開鍵暗号は、素数の数学的性質にその安全性の基礎をおいている。たとえば RSA 暗号においては、素数はユーザの秘密鍵の一部であり、したがって素数であることの確実性、すなわち真に素数であるかどうかは個人のプライバシーに直結する。また DSA 署名においては、素数が情報セキュリティシステム参加者全員で共有されるため、その素数が真に素数であるかだけでなく、それが正しく生成されたものであるかどうか、すなわち意図的に弱い素数を生成するようなことがおこなわれていないことが万人に証明できなければ、情報セキュリティシステムの真の社会的信頼性は得られない。具体的には、最近実運用が開始された住民基本台帳システムにおいては、多数の利用者が公開鍵暗号を継続的に利用する。そのため、例えば RSA 暗号秘密鍵の大量かつ高速な生成、つまり安全な素数の高速な生成法は、利便性が高く安定したシステムには必須となっている。また、今後 e-Japan 構想が実現を目指すと思われる電子選挙においても同様に安全な素数生成は、そのシステムの質を規定する重要な要素技術である。

暗号において用いられる素数は大きな値であり、例えば RSA 暗号においては安全性上の理由から 512 ビットあるいは 1024 ビットという長いデータが必要となる。また同じく RSA 暗号においては、暗号化や復号に必要な時間は、一般に素数の長さの 3 乗のオーダーなのに対して、素数生成に必要な時間は、現在知られている最も高速な方式を用いても 4 乗のオーダーとなり、現状では、高速な PC を用いても長い素数の生成には秒単位の時間が必要となる。RSA 暗号では、素数はユーザごとに一度生成すれば一定期間それを用いることが前提であるため、素数生成は、暗号化や復号と同程度の速度は必ずしも必要ではないが、上記にあげた国民全体が参加するような大規模システムを考えた場合には、素数生成の高速化はシステム運用上の重要な課題である。

1.2 本調査報告書の目的

本調査報告書は、安全で信頼できる電子政府セキュリティシステムを構築するために、素数に関連する事項の必要条件について調査することを目的とする。

調査は理論的観点および実装上の観点から実施した。

本調査報告で対象とする素数判定法、素数生成法に対して報告書末尾に記された参考文献、又は関連 URL を中心にして、調査を行った。ここで挙げられた文献以外にも関連があると思われるものに関しては調査を行った。それら関連文献一覧に関しては、R. Crandall と C. Pomerance の著書 [19]、H. C. Williams の著書 [69] などの巻末に付された文献一覧を参照した。また、ANSI 等各種標準で制定されている素数判定および素数生成法についても調査を行った。また、RSA 暗号や DSA 署名など、実際の暗号ごとに、素数に要求される要件についても調査した。

実装性については、べき乗剰余演算など、素数判定および素数生成において必要となる演算アルゴリズムの高速化方法についても検討した。素数判定法および素数生成法を実際にソフトウェアで実装することにより、速度およびメモリの両観点から考察を行った。

以上のような調査および実装実験によって、電子政府セキュリティシステムにおける使用にふさわしい素数判定、素数生成の方法、および実装方法に関する指針を示す。

1.3 本調査報告書の構成

本調査報告書の構成は次の通りである。まず2節から4節までで、各種素数判定法および素数生成法の理論的調査を行う。2節では確率的素数判定法、3節では確定的素数判定法、4節では素数生成法に関して調査する。次に5節において各種方法の理論的側面および実装的側面における比較考察を行う。各アルゴリズムの有効性について比較検討し、使用すべき素数判定法、素数生成法の指針を与える。6節では、素数判定法および素数生成法の実装性を考察する。高速化手法を示し、実際に速度計測を行い、高効率実装を実現するための方法を示す。7節では素数を実際の暗号用途に用いる場合に必要となる要件について考察する。8節において、素数判定法および素数生成法に関する標準化動向についてまとめる。9節では、日本における電子政府で、安全で信頼できるセキュリティシステムを構築するために、素数に関連する必要条件について考察する。最後に、10節において本調査報告書のまとめとして、電子政府セキュリティシステムにおける素数判定法および素数生成法の要件をまとめる。

2 確率的素数判定法

2.1 概要

本報告書で扱う確率的素数判定法は、表1にあるものであるがそれらは全て“素数であるための必要条件”を利用している。つまり

n が素数であれば、ある条件 Cond_i (複数個 $i = 1, 2, \dots$) を満たす。

という形の定理を基にアルゴリズムを構成している。また、これは、 Cond_i の否定が合成数であるための十分条件であることと同値であることに注意しておく。典型的にはアルゴリズム1のように記述される。¹

つまり、 t 個の Cond_i に関して調べたところ、合成数とは判定されなかったものを “probable prime” として出力するアルゴリズムである (つまり正確には合成数判定アルゴリズムである)。ここで問題となるのは n は合成数であるにも関わらず Cond_i を満たしてしまう場合がどれくらいあるかということである (このような時 n は “ \sim 擬素数” という。 \sim は Cond_i により異なる)。それら条件 (に付随するパラメータ) を n に対する偽証拠 (false witness) と呼ぶことにする ([23] 等参照。[43] では liar と呼ばれている。正確な定義は各判定法の説明の項で記述する)。本節では判定法の性能を比較する測度の1つとして、(1) 式で定義される条件付き確率 “ $(c \rightarrow \text{pp})$ 誤り確率” (合成数が probable prime と判定される確

¹probable prime に関しては適当な日本語訳が見つからなかったので、英語のまま以後も参照する。

アルゴリズム 1 確率的素数判定法

入力 n (奇数), t (判定法の繰り返し回数)

出力 “probable prime” か “composite”

- 1: for $i = 1$ to t do
 - 2: if n は i 番目の必要条件 Cond_i を満たさない then
 - 3: “composite” を出力.
 - 4: end if
 - 5: end for
 - 6: 上記ループを, 途中出力なしに終了したら, “probable prime” を出力.
-

率) を考え, その上界値を与える (2) 式を評価する (その正確な定義も各小節で各判定法に応じた記述する).

$$(c \rightarrow \text{pp}) \text{ 誤り確率} := \text{Prob} \left[\begin{array}{l} n \text{ が probable prime} \\ \text{と判定される} \end{array} \middle| \begin{array}{l} n \text{ は合成数} \end{array} \right] \quad (1)$$

$$\left(\leq \max_{n: \text{合成数}} \left\{ \frac{n \text{ に対する偽証拠の総数}}{\text{Cond}_i \text{ の総数}} \right\} \right) \quad (2)$$

注意すべきことは, (2) 式はあくまでも “(c→pp) 誤り確率” の上界値であり, 測度である “(c→pp) 誤り確率” そのものを与えているわけではないことであり, (2) 式の値が小さければ小さいほど確率的素数判定法として優れていると判断できるが, (ある程度) 大きいからと言って必ずしも “(c→pp) 誤り確率” が大きいとは言えないことである. しかし, (1) 式の値を直接求められない以上, 小さい (2) 式の値を有する (正しく評価できる) ということ自体が良い確率的素数判定法としての条件であると見る見方が一般的に取られていると思われる.

本節内の各小節で扱う確率的素数判定法に関連して参照している文献またはそのために調査した文献の主なものを表 1 に示す.

2.2 節	Fermat 法 (F 法)	[3, 35, 43]
2.3 節	Solovay-Strassen 法 (SS 法)	[35, 43, 65]
2.4 節	Miller-Rabin 法 (MR 法)	[4, 7, 23, 35, 43, 46, 47, 59]
2.5 節	Lucas-Lehmer 法 (LL 法)	[4, 5, 8, 19, 54, 55, 56, 60]
2.6 節	Frobenius-Grantham 法 (FG 法)	[4, 29, 30, 51, 55]

表 1: 2 節内各小節に関連して参照・調査した主な文献

2.2 Fermat 法

Fermat 法で基礎とする定理は

定理 1 n が素数であれば, $1 \leq a < n$ である整数 a (よって $\text{gcd}(a, n) = 1$) に対して

$$a^{n-1} \equiv 1 \pmod{n} \quad (3)$$

である.

というものであり, Fermat の小定理と呼ばれるものである. つまり, 各 a に対する定理 1 の後半部を 2.1 節での Cond_i としてアルゴリズム 1 に適用したものを Fermat 法と呼ぶ.

また, その際に含まれる演算も高速べき乗計算法を用いれば, $O((\log n)^3)$ 回のビット演算で計算可能であり, 暗号応用に用いることができる時間で実行可能となる.

定義 1 合成数 n に対して, 条件式 (3) を満たす $a(1 \leq a < n)$ を (Fermat-) 偽証拠という. また, この時 n は a を基底とする (Fermat-) 擬素数であるという. n に対する偽証拠の集合を $FW(n)$ で, その個数を $\#FW(n)$ で表す (以後も集合の前に $\#$ を付すことでその要素数を表す).

次の定理 2 は, Fermat 判定法が有効でない奇合成数 n が無限に存在することを示している.

定理 2 ([3]) 奇合成数 n であって, $\#FW(n) = \varphi(n)$ である n は無限個存在する. ここで $\varphi(n)$ は $a (1 \leq a < n, \gcd(a, n) = 1)$ の個数とする.

つまり, Fermat 法に関しては (2) 式を用いた (c→pp) 誤り確率の評価は自明なものであり, 以降の各節の判定法に比べて優位性を示すことはできない.

上記のような正整数 n は Carmichael 数と呼ばれている. 実際, 例えば $561 = 3 \cdot 11 \cdot 17$ が Carmichael 数であることが知られている.

Fermat 法は, ほとんどの (確率的) 素数判定法の雛形であるが, 上記の理由により, 実際に用いられることはない.

2.3 Solovay-Strassen 法

Fermat 法の弱点 (Carmichael 数の存在) を克服するために考案された素数判定法が Solovay-Strassen 法である. Solovay-Strassen 法は, 以下の定理 (Euler の規準と呼ばれる) を基礎とする.

定理 3 n が素数であれば, $1 \leq a < n$ である整数 a (よって $\gcd(a, n) = 1$) に対して,

$$a^{(n-1)/2} \equiv \left(\frac{a}{n}\right) \pmod{n} \quad (4)$$

である. ここで $\left(\frac{a}{n}\right)$ は Jacobi 記号である.

n が素数 p の時には, Jacobi 記号 $\left(\frac{a}{p}\right)$ は $p \nmid a$ であれば, $x^2 \equiv a \pmod{p}$ が $1 \leq x < p$ の範囲で解を持てば 1, そうでなければ -1 , $p|a$ の時には 0 と定義される. 一般の $n = \prod p_i^{e_i}$ に対しては, $\left(\frac{a}{n}\right) = \prod \left(\frac{a}{p_i}\right)^{e_i}$ と定義される. Jacobi 記号 $\left(\frac{a}{n}\right) (1 \leq a < n)$ の計算は, 平方剰余の相互律を用いることで $O((\log n)^3)$ のビット演算量で効率的に実行できる ([35] 参照).

そのアルゴリズム記述は, 2.1 節での Cond_i に, 各 a に対する定理 3 の後半部を用いて, アルゴリズム 1 に適用することで得られる.

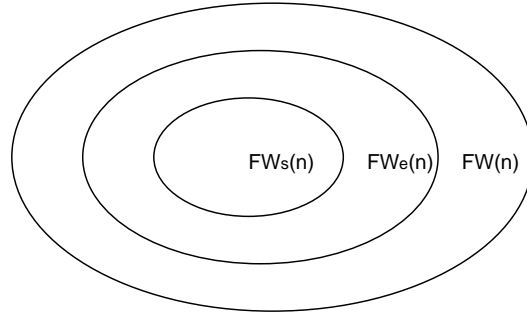


図 1: 偽証拠集合間の包含関係 ([43] より引用)

定義 2 奇合成数 n に対して, 条件式 (4) を満たす a ($1 \leq a \leq n, \gcd(a, n) = 1$) を Euler-偽証拠という. また, この時 n は a を基底とする Euler-擬素数であるという. n に対する Euler-偽証拠の集合を $FW_e(n)$ で表す.

その時, 全ての奇合成数 n に対して, $\frac{\#FW_e(n)}{\varphi(n)} \leq \frac{1}{2}$, つまり t 個基底 a を取った Solovay-Strassen 法の (c→pp) 誤り確率 $\leq \left(\frac{1}{2}\right)^t$ となることが知られている ([35] 参照). また, 全ての奇合成数 n に対して, $FW_e(n) \subset FW(n)$ となる ([35, 43], 図 1 参照) ことが知られていることから Solovay-Strassen 法は Fermat 法よりも優れているといえる.

しかし, Solovay-Strassen 法より誤り確率の点でも計算効率の点でも優れた方法が次に述べる Miller-Rabin 法である.

2.4 Miller-Rabin 法

Miller-Rabin 法は以下の定理を基礎とした確率的素数判定法である.

定理 4 n が素数であれば, $n - 1 = 2^s t$ (但し t は奇数) とすると, $1 \leq a < n$ である整数 a (よって $\gcd(a, n) = 1$) に対して,

$$\begin{aligned} & a^t \equiv 1 \pmod{n} \text{ が成立するか} \\ & \text{ある } i \ (0 \leq i < s) \text{ に対して } a^{t2^i} \equiv -1 \pmod{n} \text{ が成立する.} \end{aligned} \quad (5)$$

Fermat 法, Solovay-Strassen 法同様, 各 a に対する上記定理の後半部を Cond_i に用いて, アルゴリズムが構成される.

定義 3 奇合成数 n に対して, 条件式 (5) を満たす a ($1 \leq a \leq n, \gcd(a, n) = 1$) 強偽証拠 (MR-偽証拠) という. また, この時 n は a を基底とする強擬素数であるという. n に対する強偽証拠の集合を $FW_s(n)$ で表す.

その時, 全ての奇合成数 n に対して, $\frac{\#FW_s}{\varphi(n)} \leq \frac{1}{4}$ となる, つまり t 個基底 a を取った Miller-Rabin 法の (c→pp) 誤り確率 $\leq \left(\frac{1}{4}\right)^t$ となることが知られている ([35] 参照). また, 全ての奇合成数 n に対して, $FW_s(n) \subset FW_e(n)$ となる ([35, 43], 図 1 参照) ことが知られている.

つまり、ここまでの素数判定法の中では、誤り確率という点から最も優れた方法であることがわかる。また、その計算効率に関しても、Fermat 法と同等かそれ以上の高効率を達成できる。そして、またアルゴリズム記述の簡単さの故もあって、現在最も一般的に用いられている確率的素数判定法である。

Miller-Rabin 法に関しては更に詳しく素数生成法に用いた時に合成数を素数と誤る確率についても調べられていて ([23]), それに関しては 4.3.2 節を参照。

また、Solovay-Strassen 法, Miller-Rabin 法に関しては「成立するであろうと信じられている一般リーマン予想 (GRH) という数論での予想の成立を仮定すると, $1 < a < \min\{n, 2(\log n)^2\}$ である全ての a を基底として probable prime であれば素数である」ことが知られている ([7, 46]).

2.5 Lucas-Lehmer 法

Lucas-Lehmer 法は次の定理に基づいている。

定理 5 n が素数で, 整数 P, Q に対し $\Delta = P^2 - 4Q$ として $n \nmid 2Q\Delta$ (よって $\gcd(2Q\Delta, n) = 1$) であれば

$$U_{n - \left(\frac{\Delta}{n}\right)} \equiv 0 \pmod{n} \quad (6)$$

である。ここで, U_i は $U_0 = 0, U_1 = 1, U_i = PU_{i-1} - QU_{i-2} (i \geq 2)$ という線形漸化式によって定義される。

前節までと同様, 各 P, Q に対する上記定理の後半部を Cond_i に用いて, アルゴリズムが構成される。

また, $U_{n - \left(\frac{\Delta}{n}\right)} \pmod{n}$ の計算は, P, Q, Δ が n に比べて十分小さければ, 法 n での普通のべき乗剰余計算の約 2 倍の計算時間で可能である (その方法に関しては例えば [4] 参照)。

[5] の偽証拠に関する結果を引用するために, 次の定理を述べる。

定理 6 ([5]) n が素数で, 整数 P, Q に対し $\Delta := P^2 - 4Q$ として $n \nmid 2Q\Delta$ (よって $\gcd(2Q\Delta, n) = 1$) であれば $n - \left(\frac{\Delta}{n}\right) = 2^s t$ (但し t は奇数) として

$$U_t \equiv 0 \pmod{n} \text{ が成立するか} \\ \text{ある } i (0 \leq i < s) \text{ に対して } V_{2^i t} \equiv 0 \pmod{n} \text{ が成立する.} \quad (7)$$

である。ここで, U_i は定理 5 中と同様に, V_i は $V_0 = 2, V_1 = P, V_i = PV_{i-1} - QV_{i-2} (i \geq 2)$ という線形漸化式によって定義される。

定理 5, 6 を踏まえて次の定義を行う。

定義 4 奇合成数 n と, n と互いに素な整数 Δ に対して, 条件式 (6) を満たす (P, Q) ($1 \leq P, Q < n, \gcd(Q, n) = 1, P^2 - 4Q \equiv \Delta \pmod{n}$) を Lucas-偽証拠ということにする。また, この時 n は (P, Q) を基底とする Lucas-擬素数であるという。 n に対する Lucas-偽証拠の集合を $FW_l(n)$ で表す。また, 上記の条件式 (6) を条件式 (7) に置き換えて, 同様に強 Lucas-偽証拠, 強 Lucas-擬素数, $FW_{sl}(n)$ が定義される。

上の定義の下で, ある特殊な合成数 n を除いた合成数 n に対しては, $\frac{\#FW_{st}(n)}{n} \leq \frac{4}{15}$ (t 個基底を取った Lucas-Lehmer 法の (c→pp) 誤り確率 $\leq (\frac{4}{15})^t$) であることが [5] で示されている. この上界値は, それほど魅力的でない値であるが, あくまでこれは理論的上界値 (最悪誤り確率) であって, 実際の $\frac{\#FW_s(n)}{n}$ の (平均) 値はずっと小さいと思われる (2.1 節参照).

また, この Lucas-Lehmer 法は次の点から 2.2, 2.3, 2.4 節で述べた方法とは違う特質を備える. Fermat 法, Solovay-Strassen 法, Miller-Rabin 法に関する偽証拠の集合の間には $FW_s(n) \subset FW_e(n) \subset FW(n)$ という包含関係があった. つまり, 素数を選び分ける能力という点で, Miller-Rabin 法に他の方法は適わない. しかし, Lucas-Lehmer 法を用いれば, 強擬素数を合成数と判定することが可能となる. 例えば, $25 \cdot 10^9$ 以下の 2 を基底とする擬素数は全て, あるパラメータ P, Q に対する Lucas-Lehmer 法を用いれば合成数であることが示される ([56] 参照). また, 逆に 2 を基底とする擬素数でなくても Lucas 擬素数となるものも存在する.

このことから, 素数選り分けには, Miller-Rabin 法と Lucas-Lehmer 法を組み合わせると有効性を高めることが可能ではないかと思われる. 実際, ANSI X9.80[4] では, Miller-Rabin 法を数個の基底に関して実行した後に, Lucas-Lehmer 法を 1 度行うことが推奨されている. 現在までに, 5 を法として 2 又は 3 であり, 2 を基底とした擬素数で, かつ $(P, Q) = (1, -1)$ に対する Lucas 擬素数は見つかっていない ([54] 参照).

2.6 Frobenius-Grantham 法

Frobenius-Grantham 法は次の定理に基づいている.

定理 7 n が素数で, $f(X)$ を有限体 F_n を係数とする次数 d , 判別式 Δ のモニック多項式として $\gcd(f(0)\Delta, n) = 1$ とする. $f_0(X) := f(X)$, $F_i(X) := \gcd(X^{n^i} - X, f_{i-1}(X))$, $f_i(X) := f_{i-1}(X)/F_i(X)$ ($1 \leq i \leq d$) として, また各 i ($1 \leq i \leq d$) に対して, $n^i - 1 = 2^r s$ とした時に, $F_{i,0}(X) := \gcd(F_i(X), X^s - 1)$, $F_{i,j}(X) := \gcd(F_i(X), X^{2^{j-1}s} + 1)$ ($1 \leq j \leq i$) とし, $S := \sum_{2|i} \frac{\deg(F_i(X))}{i}$ と定義する. この時,

$$f_d(X) = 1 \text{ かつ } i | \deg(F_i(X)) \quad (1 \leq i \leq d) \quad (8)$$

$$F_i(X) | F_i(X^n) \quad (2 \leq i \leq d) \quad (9)$$

$$(-1)^S = \left(\frac{\Delta}{n}\right) \quad (\text{Jacobi 記号}) \quad (10)$$

$$\prod_{j=0}^r F_{i,j}(X) = F_i(X) \text{ かつ } i | \deg(F_{i,j}(X)) \quad (0 \leq j \leq i) \quad (11)$$

前節までと同様に, 各 $f(X)$ に対する定理 7 の条件式 (8), (9), (10) を Cond_i に用いて構成されたアルゴリズムを Frobenius-Grantham 法と呼び, Cond_i に更に条件式 (11) を加えたアルゴリズムを強 Frobenius-Grantham 法と本報告書ではいうことにする. [29] では, $d = 2$ 時の本報告書の強 Frobenius-Grantham 法が 2 次 Frobenius テストと呼ばれている (詳細なアルゴリズム記述は [29] を参照). また, ANSI X9.80[4] でも, $d = 2$ の多項式 $f(x)$ を用いることが推奨されている.

その時 ($d = 2$), 1 つの多項式 $f(X)$ に対する判定法の漸近計算時間は, 1 つの基底に対する Miller-Rabin 法の漸近計算時間の約 3 倍である ([29] 参照). また, [51] で, 実際的には Miller-Rabin 法の計算時間の約 4.5 倍かかると指摘されている.

定義 5 奇合成数 n に対して,

$$M(n) = \left\{ (P, Q) \in \mathbb{Z}^2 \left\{ \begin{array}{l} 0 \leq P, Q < n, \\ \left(\frac{P^2+4Q}{n}\right) = -1 \text{ と } \left(\frac{Q}{n}\right) = 1 \text{ が共に成立するか} \\ 1 < \gcd(P^2 + 4Q, n) < n \text{ が成立するか} \\ 1 < \gcd(Q, n) < n \text{ が成立するか} \end{array} \right. \right\}$$

として, 条件式 (8)~ (11) を満たす $(P, Q) \in M(n)$ (又はその時の 2 次多項式 $f(X) = X^2 - PX - Q$) を強 Frobenius-偽証拠ということにする. また, この時 n は (P, Q) (又はその時の 2 次多項式 $f(X) = X^2 - PX - Q$) を基底とする強 Frobenius-擬素数であるという. n に対する強 Frobenius-偽証拠の集合を $FW_{sf}(n)$ で表す.

Frobenius-Grantham 法の特長は, 上記の $\frac{\#FW_{sf}}{\#M(n)}$ を評価することによって, t 回繰り返しの強 Frobenius-Grantham 法の (c→pp) 誤り確率の上界値が $(1/7710)^t$ と 2.2 節から 2.5 節までの判定法と比べて低く与えられていることである.

それにより, 漸近的実行時間が等しい 3 基底に対する Miller-Rabin 法の (c→pp) 誤り確率 $1/4^3 = 1/64$ とは比較にならない効率性を達成できる. また, 1 次の $f(X)$ を用いると, 2.2 節から 2.4 節までの判定法と同等のものを構成できたり, あるパラメータに対する 2 次多項式 $f(X)$ を基底とする強 Frobenius 擬素数は, 別のパラメータに対応する強 Lucas 擬素数になっていたりする ([30] 参照) など, 2.2 節から 2.5 節までの確率的素数判定法を含む方法になっているのは, 重要な特長である.

しかし, 本節の Frobenius-Grantham 法では, 素数生成法の時に重要な誤り確率評価がうまくできないという難点を持っている (このことについては 4.3.2, 5.2 節参照).

また, [51] において, 更に (c→pp) 誤り確率の上界値を低くできる方法が提案されている (詳細は [51] 参照).

3 確定的素数判定法

3.1 概要

確定的素数判定法に関しては, [55] にまとまったサーベイがある. 本報告書では, ANSI X 9.80[4] の項目に従ってそれら方法の記述を行う.

確定的素数判定法は, 全て “素数であるための十分条件” を利用している. つまり

$$\begin{array}{l} \text{奇数 } n \text{ がある仮定 Assump のもとで,} \\ \text{ある条件 Cond を満たせば, } n \text{ は素数である.} \end{array} \quad (12)$$

という形式の定理をアルゴリズムの基礎としている. 以降の各小節において, 各々のアルゴリズムの基礎となる定理を上記の形式に従って解釈していくが, 3.2, 3.6 節以外では,

Cond の成否が $\log n$ の多項式時間 (多項式回のビット演算) でチェックできるものになっており, 3.6 節の方法では, それは “ほとんど” 多項式時間でチェックできる.

本節内の各小節で扱う確定的素数判定法に関連して参照している文献またはそのために調査した文献の主なものを表 2 に示す.

3.2 節	試行割算法	[4, 34]
3.3 節	Proth-Pocklington-Lehmer-Selfridge 法 (PPLS 法)	[4, 19, 55]
3.4 節	Kraitchik-Lehmer 法 (KL 法)	[4, 55, 57]
3.5 節	Elliptic Curve Primality Proving 法 (ECP 法)	[4, 6, 26, 37, 48, 49, 63]
3.6 節	Cohen-Lenstra 法 (CL 法)	[1, 4, 17, 18, 44, 45, 55, 67]
3.7 節	Agrawal-Kayal-Saxena-Lenstra 法 (AKSL 法)	[2, 10, 50, 55, 66]
3.8 節	AKS[2] での予想に基づく方法 (AKS 予想法)	[2, 32]

表 2: 3 節内各小節に関連して参照・調査した主な文献

3.2 試行割算法

定理 8 奇数 n が $p \leq n^{1/2}$ である全ての素数で割り切れなければ, n は素数である.

試行割算法は, 言うまでもなく素数候補 n を順に小さい素数によって割ってみて, 素数性を判定する方法である. その最悪計算量は, $n^{1/2}$ に比例して与えられることはすぐわかるが, 平均計算量になると考察を要する. つまり, ランダムに与えられた正整数の最小素因数の大きさを見積もることが必要になる. 知られた結果としては, 約半数のランダム正整数に対し, $n^{0.35}$ 以下の整数による試行割算を繰り返せば合成数は合成数と判定されることがわかっている ([34]).

以上は理論的側面の話であるが, 実際的には試行割算は, 他の確率的, 確定的素数生成法の前処理部分として用いられることでその効率性を高めるために用いられている.

3.3 Proth-Pocklington-Lehmer-Selfridge(PPLS) 法

PPLS 法はいわゆる $n - 1$ 法と $n + 1$ 法の組み合わせである ([55] 参照).

本報告書では ANSI X9.80[4] での順序に基づいて記述を行っているため前後するが, 3.4 節の定理 10 が $n - 1$ 法の雛形であることに注意しておく.

定理 9 奇数 $n > 1$ に対して, 整数 a, F_1 ($F_1 | n - 1$) が

$$a^{n-1} = 1 \pmod n \text{ かつ全ての素数 } q | F_1 \text{ に対し } \gcd(a^{(n-1)/q} - 1, n) = 1 \quad (13)$$

となっているとする. また, 整数 P, Q ($\gcd(Q, n) = 1, \Delta := P^2 - 4Q$ として $\left(\frac{\Delta}{n}\right) = -1$) と整数 F_2 ($F_2 | n + 1$) が

$$U_{n+1} \equiv 0 \pmod{n} \text{ かつ全ての素数 } q | F_2 \text{ に対し } U_{(n-1)/q} \in (\mathbb{Z}/(n))^* \quad (14)$$

となっていて, $F = \text{lcm}[F_1, F_2] > n^{1/2}$ となっていれば, n は素数である.

3.1 節の形式での Assump は $F_1 | n - 1, F_2 | n + 1, F = \text{lcm}[F_1, F_2] > n^{1/2}$ となる F_1, F_2 の素因数分解を知っていることであり, Cond は式 (13) と式 (14) を満たすことである.

また, [55] では, 式 (13), 式 (14) を基にした素数判定法をそれぞれ “ $n - 1$ ” 法, “ $n + 1$ ” 法と呼んでいる.

ANSI[4] のアルゴリズムでは, P, Q の選択に当たっては, $\Delta = 5, -7, 9, -11, 13, -15, 17, \dots$ に対し, $(P, Q) = (1, \frac{1-\Delta}{4})$ という手順で探索を行っていくと記述されている.

3.4 Kraitchik-Lehmer 法

Kraitchik-Lehmer 法は, 以下の定理 10 を基礎にしたアルゴリズムである.

定理 10 奇数 $n > 1$ に対し, $n - 1$ の素因数分解が既知とする. $n - 1$ の各素因数 q に対して, 以下の条件を満たす整数 a が存在するならば, n は素数である.

$$a^{n-1} \equiv 1 \pmod{n}, \quad a^{(n-1)/q} \not\equiv 1 \pmod{n} \quad (15)$$

本方法では, Assump として, $n - 1$ の素因数を全て知っていることが必要であり, Cond は式 (15) である.

Assump を仮定しなければならないことは Kraitchik-Lehmer 法の最大の難点であるが, また, もし $n - 1$ の素因数分解を知っていたとしても, (15) 式にあるような a を容易に見つけることができるかという問題も存在する. それに関して [55] では次の式を挙げて, ヒューリスティックには問題ないとしている.

$$200560490131 \text{ 以上の整数 } n \text{ に対して } \varphi(n - 1) > \frac{n}{2 \log \log n} \quad (16)$$

$\varphi(\cdot)$ の定義は定理 2 を参照. n が素数の時は, $\varphi(n - 1)$ は n の全ての素因数 q に対して (15) 式を満たす a の総数を表すので, ランダムな a ($1 < a < n$) に対して順次 (15) 式の成否を $2 \log \log n$ 回チェックすれば成立する a が見つかる期待される.

Kraitchik-Lehmer 法は PPLS 法よりも柔軟性, 有効性に欠けるものであるが, 実装が PPLS 法より容易であるという特長を持つ.

また, [57] において, 定理 10 は “素数判定問題” が NP (非決定性多項式時間) 問題であることの証明に用いられている. NP 問題であるとは, 何らかの “付加情報” があれば多項式時間で実行できるアルゴリズムを持つ問題のことである. 今の場合, $n - 1$ の全ての素因数 q と a がその付加情報に当たる. しかし, 更に q が素数であることにも証明をつけなければ, その付加情報が正しいものであることは保証されない. それ故, q が素数であることに對する同様の付加情報も付加する. このように帰納的に定理 10 を適用して, 元々の n

の素数性に対する付加情報を構成するというのが, [57] で行われていることであり, 更にこのように構成した付加情報の検証が多項式時間で行い得ることも示している.

上述の付加情報のことを“素数性証明書 (primality certificate)” という. 3.5 節の ECPP 法も類似の素数性証明書を用いた方法であるが, 本節のアルゴリズムでは素数性証明書の検証過程のみをアルゴリズムにしているのに対して, 3.5 節では, 素数性証明書の生成過程もアルゴリズムに含めたものとなっている. これは, 楕円曲線有理点群 (3.5 節参照) が有限体乗法群とは違ってフレキシブルに選択できることに起因している.

また現在では, [2] において素数判定問題が P 問題 (決定的多項式時間問題) であることまで示されたことを付け加えておく (3.7 節参照).

3.5 Elliptic Curve Primality Proving (ECPP) 法

まずは, 楕円曲線に関する一般的事項を簡単に述べる.

素数 n と $a, b \in \mathbb{Z}$ ($0 \leq a, b < n, 4a^3 + 27b^2 \not\equiv 0$) に対して次式で定義される代数曲線を楕円曲線と呼び, 本報告書では $E_{a,b}$ で表す

$$Y^2 = X^3 + aX + b.$$

楕円曲線有理点群 $E_{a,b}(n)$ とは式 (17) を満たす $(x, y) \in \mathbb{F}_n$ 全体の集合と無限遠点と呼ばれる点 O の和集合に群構造 (加算) が定義されたものである.

その加算の計算法に関しては, [6, 26, 37, 55] 等多数の参考文献があるためそれらを参照することとするが, 無限遠点が加算の単位元であることには注意しておく. また, 正整数 M と $P = (x, y) \in E_{a,b}(n)$ に対して, MP で P を M 回加算した点 $\underbrace{P + \dots + P}_{M \text{ 回加算}}$ を表すこととする.

楕円曲線を用いた素数判定法は, 以下の定理 ([26] 参照) に基づく.

定理 11 $\gcd(n, 6) = 1$ である $n \in \mathbb{Z}_{>1}$ と $\gcd(4a^3 + 27b^2, n) = 1$ である $a, b \in \mathbb{Z}$ に対して, $P = (x, y) \in (\mathbb{Z}/n\mathbb{Z})^2$ が $y^2 \equiv x^3 + ax + b \pmod{n}$ を満たすとする. また, $F \in \mathbb{Z}$ は $F > (n^{1/4} + 1)^2$ とする. n が素数であるかどうかに関わらず n が素数である時の楕円曲線上の演算法に従って, P の定数倍を計算するとする. その時, FP , 及び F の全ての素因数 q に対する $(F/q)P$ の計算途中で法 n での除算計算に失敗しなくて,

$$FP = O, \text{ 及び } F \text{ の全素因数 } q \text{ に対して } (F/q)P \neq O \quad (17)$$

であるなら, n は素数である.

定理 10 中の式 (15) と定理 11 中の式 (17) とは式 (15) では乗算になっているところが, 式 (17) では加算になっている事を除けば非常に類似していることに注意する. 実際, それらの証明法の構成は類似している.

定理 11 を形式 (12) に従って解釈すると, Assump としては, (適当な) 楕円曲線のパラメータ a, b (と点 P) を知っていることと F の素因数を全て知っていることが必要であり, Cond は式 (17) である.

3.4 節では判定対象とする整数 n に対して, $n-1$ の素因数を知っていることが Assump として必要であったのに対して, ECPP 法では素因数分解を知らなければいけない F を複数取り替えることが可能であることが特長である. 特に F として素数であるもの, 即ち定理 11 中で $F = q$ であるものを選択することが可能である.

また, 定理 11 中での条件である $F > (n^{1/4} + 1)^2$ を素数であり (17) 式を満たすように得るのには, 楕円曲線 $E_{a,b}(n)$ に対する位数計算アルゴリズムというアルゴリズムが必要になるが, 位数計算は [63] にある方法を用いれば多項式時間で実行できる.

よって, 3.4 節後半部に記述されたように素数性証明書の検証 ((17) 式) だけでなく, (帰納的な) 素数性証明書の構成も効率的に可能となる.

そのように定理 11 を帰納的に適用して構成したアルゴリズムを (Goldwasser-Kilian の) ECPP 法と呼び, [26] によれば, それは, $\log n$ ビットの入力に対して $1 - O(2^{-\log n^{1/\log \log \log n}})$ の確率で (確率的) 多項式時間で動作する.

実際的には [63] でのアルゴリズムの代わりに, 虚数乗法論に基づいたアルゴリズム [6] を用いることによりヒューリスティックな計算量が $O((\log n)^{6+\epsilon})$ (但し, ϵ は任意の正数) である方法が用いられる ([37] 参照).

[48] によると, DEC 3000 (125 MHz) 上で, 512 ビット素数 50 個の判定時間を統計して平均 35.81 秒, 最悪 68.53 秒, 最良 18.17 秒という結果が述べられている.

3.6 Cohen-Lenstra 法

Cohen-Lenstra 法は, 別名ヤコビ和テストとも呼ばれ, 以下の定理 12 に基づいて構成される. ここで, 本方法理解に不可欠のヤコビ和に関する数学の用語の定義を若干行う.

素数 p, q を $p^k \parallel q-1$ となるものとし, ディリクレ指標 $\chi_{q,p} : \mathbb{F}_q^* \rightarrow \mathbb{Q}[\zeta_{p^k}]$ は $\chi_{q,p}(\mathbb{F}_q^*) = \langle \zeta_{p^k} \rangle$ となるものとする. また, 整数 a, b を $ab(a+b) \not\equiv 0 \pmod p$ かつ $(a+b)^p \not\equiv a^p + b^p \pmod{p^2}$ となるものとする (このような a, b はいつも存在する). この時, ヤコビ和 $J(\chi_{q,p}^a, \chi_{q,p}^b)$ は以下で定義される.

$$J(\chi_{q,p}^a, \chi_{q,p}^b) := - \sum_{y=0}^{q-1} \chi_{q,p}^a(y) \chi_{q,p}^b(1-y)$$

また, 体の拡大 $\mathbb{Q}(\zeta_{p^k})/\mathbb{Q}$ のガロア群 $G = \text{Gal}(\mathbb{Q}(\zeta_{p^k})/\mathbb{Q})$ による群環 $\mathbb{Z}[G]$ の元 α を次式で定義する. ここで, n は素数判定を行う奇数 (> 1) とし, $y \in \mathbb{Q}$ に対し, $[y]$ は y を超えない最大整数とする.

$$\alpha = \sum_{x=1, p \nmid x}^{p^k} \left[\frac{nx}{p^k} \right] \sigma_x^{-1} \in \mathbb{Z}[G]$$

但し, ここで $\sigma_x \in G$ は, $\sigma_x : \zeta_{p^k} \rightarrow \zeta_{p^k}^x$ で定まる体同型とする.

また, (小さい) 奇素数の有限集合 E と各 $p \in E$ に対して決める整数 $a_p > 0$ を以下の条件が成立するように定める.

$$t := 2 \prod_{p \in E} p^{a_p}, \quad s := \prod_{q \text{ は素数}, q-1 \mid t} q^{v_q(t)+1} \quad (v_q \text{ は } q \text{ での付値})$$

とした時 $s > n^{1/2}$.

定理 12 ([67]) 次の 4 条件が成立すれば, n は素数であり, そうでなければ合成数である.

1. s の全ての素因数 q に対して $q^{(n-1)/2} \equiv \pm 1 \pmod{n}$
2. s の全ての素因数 q と $q-1$ の全ての素因数 p に対して $J(\chi_{q,p}^a, \chi_{q,p}^b)^\alpha \equiv \zeta \pmod{n}$ となる. (但し, ζ は 1 の p^k 乗根である).
3. s の全ての素因数 q に対して $c^{(q-1)/2} \equiv -1 \pmod{n}$ となる $c \in \mathbb{Z}$ が存在する.
4. n のすべての非自明な因数 r は, どの i ($0 \leq i < t$) によっても $r \equiv n^i \pmod{s}$ となっていない

定理 12 中の各条件は効率的に判定することができる. よって, 本方法の場合は, 形式 (12) での Assump にあたるものは存在せず, Cond は上記各条件の全てである.

本判定法の鍵となるのは, 条件 2 であり, これは Solovay-Strassen 法の判定法の拡張とみなせる. 条件 2 中で計算負荷の大きい部分はヤコビ和の $\log n^{O(\log \log \log n)}$ 回の計算であり, この回数評価により多項式時間判定法にはなっていない.

この漸近計算量評価にも関わらず, 本方法 (の発展形) は, 実際的には確定的な判定法では最速な判定法になっている. [1, 17, 18, 39] 等の先駆的な仕事の後, 本方法は “円分環素数判定法 (cyclotomy primality proving)” として発展していて, [45] では, DEC Alpha 500 上で 10 進 200 桁の素数 n の判定時間を複数得て, その統計によって平均 23.83 秒, 最悪 41.40 秒, 最良 13.70 秒という結果が述べられている.

また, [45] p.106 表 3 では, 円分環素数判定法及び ECPP の速度比較がなされており, ここでは ECPP の値は [48] よりの引用値を用いている. その表では実測時間と H/W に異存しない比較値が与えられていて, 10 進 160 桁数に対して H/W に異存しない比較値では円分環素数判定法は ECPP 法の 9 倍の速度性能を達成している (また, それらデータから [45] で用いた H/W の性能を推測すると [48] で用いたものの高々 2 倍の性能と推測される. このことは 5.2 節の表 7 でのデータを算出する時に用いた).

3.7 Agrawal-Kayal-Saxena-Lenstra(AKSL) 法

2002 年 8 月に [2] において, 素数判定問題に対して (決定的) 多項式時間アルゴリズムがあることが示された. 更に [10] において改良された方法が与えられた. 本報告書ではその方法を Agrawal-Kayal-Saxena-Lenstra(AKSL) 法と呼ぶことにする. その鍵となる定理は以下のものである.

定理 13 n, r を正整数とする. $S(\subset \mathbb{Z})$ を (ある) 有限集合とする. 次の条件が全て満たされれば n は素数である.

1. n は a^c (但し, $a, c \in \mathbb{Z}, c > 1$) とはなっていない.
2. n は $(\mathbb{Z}/r\mathbb{Z})^*$ の原始根である.
3. $b \neq b'$ である全ての $b, b' \in S$ に対して $\gcd(n, b - b') = 1$
4.
$$\binom{\varphi(r) + \#S - 1}{\#S} \geq n^{\lfloor \sqrt{\varphi(r)} \rfloor}$$
5. 全ての $b \in S$ に対して, $(x + b)^n = x^n + b$ が $(\mathbb{Z}/n\mathbb{Z})[x]/(x^r - 1)$ 内で成立する.

本方法では, 形式 (12) での Assump にあたるものは存在せず, Cond は上記条件の全てであり多項式時間でその成否を判定できる.

$s = \#S$ として, [2, 10] にあるように高速スカラ乗算法, 高速フーリエ変換を用いて得られる漸近計算量を考えると, それは $\tilde{O}(rs(\log n)^2)$ である. ここで $\tilde{O}(t(n)) := O(t(n)\text{poly}(\log(t(n))))$ とする. 定理 13 の条件を満たす r, s は, $r = O((\log n)^6)$, $s = O((\log n)^4)$ の範囲で見つかることには数学的に証明がつく. 実際的には, $r = O((\log n)^2)$, $s = O((\log n)^2)$ の範囲で見つかると思われる ([10] 参照) ので, 時間計算量は, 厳密には $\tilde{O}((\log n)^{12})$ であり, 予想としては $\tilde{O}((\log n)^6)$ である.

[50] によると, 700MHz PC 上, 10 進 9 桁素数に対して, AKSL 法が約 6 分 9 秒で実行できたとある. これを 1024 ビットに換算すると, 約 65 万年となって現実的には不可能な数字となる.

よって, AKSL 法それ自体は, 実用的な確定的素数判定法にはなり得ない. しかし, その発見自体が新しいことからより時間計算量の小さいアルゴリズムへの改良は今後も続けられていくであろう (例えば [11, 66] 参照).

次節で扱う方法は予想に基づくものではあるが, 計算時間を大幅に削減できるので, 本報告書では注目して調査する.

3.8 AKS での予想に基づく方法

[2] 6 節の予想によると, 本来の AKS 法より大幅に計算時間を減らすことが可能になる. その予想は以下の通りである.

予想 1 $n \in \mathbb{Z}_{>1}$ に対し, 次の条件を満たす $r \in \mathbb{Z}_{>0}$ ($r \nmid n$) が存在すれば n は素数である.

$$(x-1)^n \equiv x^n - 1 \pmod{(x^r - 1, n)} \text{ かつ } n^2 \not\equiv 1 \pmod{r}$$

本方法は, $\tilde{O}((\log n)^3)$ の計算量で見積もられている ([2]). 予想 1 は, 定理 13 で $S = \{1\}$ としたものであり, かつ r に関する条件は $r \nmid n$ かつ $n^2 \not\equiv 1 \pmod{r}$ だけであり, 定理 13 の条件 4 に比べると, かなり小さい r を選択することが可能であることがわかる. 実際 r は $[2, 4 \log n]$ という区間内で取ることが可能である ([2]). [32] においては, $r \leq 100$ かつ $n \leq 10^{10}$ の範囲では, 予想 1 が成り立つことが計算機実験によって確かめられている.

本方法 (及び AKSL 法それ自体) は, 前節でも述べたように, 理論的及び実際の側面に関して改良が今後も続けられていくと思われる方法なので, その理論的側面はもちろん実装性能に関しても引き続き注目していく必要がある.

4 素数生成法

4.1 概要

素数生成法とは目的とするビット長の素数を出来るだけランダムに生成する方法である.

素数生成法は大きく 2 節, 3 節で述べられた素数判定法を用いる方法 (4.3 節参照) と直接素数を構成する方法 (4.4 節参照) に大別される. いずれにおいても擬似乱数生成法を用いる必要が生じる. 擬似乱数生成法に関しては, それ自身で議論, 考察すべき点が多いため, 本報告書においてその詳細な記述はしないが, 典型的な方法として, 4.2 節で, [24] において記述されている DSA 用の素数生成時に使用している乱数生成法を記述する.

本節内の各小節で扱う生成法に関連して参照している文献またはそのために調査した文献の主なものを表 3 に示す.

4.3.2 節	Random Choice (RC 法)	[4, 16, 22, 23, 33]
4.3.3 節	Incremental Search (IS 法)	[4, 12, 13]
4.3.4 節	FIPS 186-2 DSS の方法	[24]
4.4.1 節	Shawe-Taylor 法 (ST 法)	[4, 43, 64]
4.4.2 節	Maurer 法 (M 法)	[4, 41, 42, 43]

表 3: 4 節内各小節に関連して参照・調査した主な文献

4.2 擬似乱数生成法

[24] の Appendix 3 に記述されている擬似乱数生成法は, [20] 5.3.1 節に現時点唯一の監視状態にある擬似乱数生成法として記述されていて, かつ CRYPTREC の 2002 年度推奨暗号リスト案 [58] にも掲載されている.

その詳細な記述は上記文献を参照することとするが, [24] に記述された方法は, ハッシュ関数 SHA-1 に基づく方法とブロック暗号 DES に基づく方法の 2 種があることに注意しておく. そして, [20] では, ハッシュ関数 SHA-1 を用いる場合を監視状態アルゴリズムとしている. また, 同 [20] ではハッシュ値が 160 ビットの SHA-1 の安全性にも言及して, 現時点で格段の安全性の欠陥とはならないが, NIST で今後制定されていくであろう次世代 SHA(SHA-256, SHA-384, SHA-512) を SHA-1 の代わりに用いることを推奨している.

4.3 素数判定法に基づく方法

4.3.1 概要

確率的素数判定法に基づいて素数を生成する時には, 4.3.2 節及び 4.3.3 節で与えられるアルゴリズム 2, 3 又はそれらアルゴリズム中の Miller-Rabin 法を他の確率的素数判定法に置き換えたアルゴリズムによって probable prime と判断された整数が合成数である確率が問題になるので, (18) 式で定義される “(pp → c) 誤り確率” を考える.

$$(pp \rightarrow c) \text{ 誤り確率} := \text{Prob} \left[\begin{array}{l} n \text{ は合成数} \\ n \text{ が probable prime} \\ \text{と判定される} \end{array} \right] \quad (18)$$

この確率の正しい評価の重要性は [9] で指摘された. 暗号用途を念頭においた場合には, 2.1 節で定義された (c → pp) 誤り確率より (pp → c) 誤り確率の方がより重要である.

[33]にあるように、 $\log n$ ビットの素数を生成する時には、 $\log n$ ビット以下の奇数が素数である確率を α として、(2) 式で定義される上界値を β とすれば、(19) 式で (pp \rightarrow c) 誤り確率が与えられることが簡単にわかる。

$$(\text{pp} \rightarrow \text{c}) \text{ 誤り確率} \leq \frac{(1 - \alpha)\beta}{\alpha + (1 - \alpha)\beta} \quad (19)$$

しかし、(19) 式の評価は実際にはあまり役に立たないので、次節以降に見られるように (pp \rightarrow c) 誤り確率のもっと良い評価を与えようとする研究が行われてきた。

また、 $\pi(x)$ (x は正実数) で x を超えない素数の個数を表すと、

$$\pi(x) \sim \frac{x}{\log x} \quad (x \rightarrow \infty) \quad (\text{素数定理})$$

である。つまり、大変大雑把に言えば、 x の周りにはおおよそ $\log x$ の確率で素数があると期待できる。その事実によって、素数判定法を用いて素数を生成することが可能になっていることに注意する。その厳密な解析は非常に重要であるが、本報告書ではこれ以上は直接触れない。以下の各節での (pp \rightarrow c) 誤り確率評価の際には、この素数分布に関連する問題は各論文で扱われている ([12, 23] 等, (19) 式の α にも注意)。

4.3.2 Random Choice

Random Choice 素数生成法とは 2 節又は 3 節内で述べた各素数判定法を用いて素数生成を行う方法の 1 つである。

本報告書では、主に Miller-Rabin 法を用いた Random Choice 素数生成法について述べる。それはアルゴリズム 2 で与えられる素数生成法である。

アルゴリズム 2 Miller-Rabin 法を用いた Random Choice 素数生成法

入力 k (ビット長), t (判定法の繰り返し回数)

出力 n (k ビットの probable prime)

- 1: repeat
 - 2: k ビットの奇数 n をランダムに選ぶ.
 - 3: n に対して, t 回繰り返し Miller-Rabin 素数判定法を行う.
 - 4: until n が “probable prime” と判定される.
 - 5: n を出力.
-

アルゴリズム 2 で出力された n が合成数である (pp \rightarrow c) 誤り確率 $p_{k,t}$ の上界値は表 4 に示される。表 4 の各エントリは、 $-\log_2 p_{k,t}$ に対して理論的に得られた下界値である。この表は、[16]にあるものを抜粋したものであり、例えば、アルゴリズム 2 を用いて 600 ビットの素数を (pp \rightarrow c) 誤り確率 2^{-80} 以下で得たい場合には $p_{600,2} \leq 2^{-82}$ であるからその繰り返し回数 t を 2 とすればよいことがわかる。

本節及び次節のアルゴリズムを高速化する方法は種々知られている。それらについては、6.1 節で述べる。

また, Miller-Rabin 法以外に小さい ($pp \rightarrow c$) 誤り確率を有する方法として, [22] にあるような Frobenius-Grantham 法 (2.6 節) を基にした方法も考案されている. その方法の有効性は, 今後 研究者らによって検証されていく必要がある.

$k \setminus t$	1	2	3	4	5	6	7	8	9	10
100	7	17	23	28	32	35	38	41	44	46
150	11	22	30	36	41	46	50	53	56	60
200	14	27	36	43	49	54	59	63	67	71
250	16	32	42	49	56	62	68	72	77	81
300	19	36	46	55	63	69	75	81	86	90
350	28	39	51	60	69	76	82	88	94	99
400	37	46	55	65	74	82	89	95	101	107
450	46	54	62	70	79	88	95	102	108	114
500	56	63	70	78	85	93	101	108	115	121
550	65	72	79	86	93	100	107	114	121	128
600	75	82	88	95	102	108	115	121	128	135

表 4: Random Choice による $-\log_2 p_{k,t}$ の既知下界値

4.3.3 Incremental Search

Incremental Search 素数生成法とは前節同様, 各素数判定法を用いた素数生成法であり, 4.3.2 節の方法の高速化と言える. Miller-Rabin 法を用いた Incremental Search 素数生成法は, アルゴリズム 3 で与えられる素数生成法で与えられる.

[13] においては, Incremental Search を用いることで 25% 高速化できたと述べられている ([13] には, 使用した H/W について記述がないことや, それは 10 年以上前の論文であることを注意しておく).

アルゴリズム 3 により合成数が出力される ($pp \rightarrow c$) 誤り確率を $q_{k,t,c,k}$ とする. また, [12] においては表 4 の時と同様に $-\log_2 q_{k,t,c,k}$ に対する理論的に得られる下界値が表にされている. そこでの値を見ると, Incremental Search を用いた場合の $-\log_2 q_{k,t,c,k}$ に対する既知下界値は, 表 4 で与えられた $-\log_2 p_{k,t}$ に対する既知下界値より小さいことがわかる. しかし, 注意すべきことは [12] での表はあくまで理論的に得ることができた下界値を一覧にしている, 実際の $-\log_2 q_{k,t,c,k}$ の値と近いとは限らないことである. 例えば, その表を見ると, $c = 1, k = 100, t = 1$ の欄のエントリが 0 になっている. それより $q_{100,1,100} \leq 1$ であることが従うが, この不等式は自明である. つまり, [12] での理論的評価では正確に評価できなかった部分が残っているということである.

4.3.4 FIPS 186-2 DSS の方法

FIPS 186-2 DSS[24] の方法は, 繰り返し Miller-Rabin 法を基にした素数生成法であるが, その中で特徴的なのは, 素数候補となる数の生成にハッシュ関数を用いている点であると思われる. そのことにより, 素数候補を一様に与えられると共に, 素数候補乱数のシード

アルゴリズム 3 Miller-Rabin 法を用いた Incremental Search 素数生成法

入力 k (ビット長), t (判定法の繰り返し回数), c (インクリメント回数を指定するパラメータ)

出力 n (k ビットの probable prime)

- 1: k ビットの奇数 $n_0(= n)$ をランダムに選ぶ.
 - 2: **while** $n < n_0 + c \cdot k$ **do**
 - 3: n に対して, t 回繰り返し Miller-Rabin 素数判定法を行う.
 - 4: n が “probable prime” と判定されれば n を出力.
 - 5: $n \leftarrow n + 2$.
 - 6: **end while**
 - 7: “fail” を出力.
-

さえわかっているならば, 後から生成された素数が恣意的に構成されたものでないことを確認できる.

4.4 直接素数を構成する方法

4.4.1 Shawe-Taylor 法

Shawe-Taylor 法は, 3.3 節定理 9 中の (13) 式を基にしたアルゴリズムである. 本節で用いる形で次のように述べておく.

定理 14 n は奇数で, 以下のような $n - 1$ の奇素因数 q ($q > n^{1/2}$) と整数 a が存在すれば, n は素数である.

$$a^{n-1} \equiv 1 \pmod{n} \text{ かつ } \gcd(a^{(n-1)/q} - 1, n) = 1 \quad (20)$$

素数 q が与えられていたら, $2R < q$ となるような整数 R を (できるだけ大きく) ランダムにとつては $n = 1 + 2Rq$ を構成して, その n, q に対して, (20) 式を満たす $a \in \mathbb{Z}$ が存在するかどうかチェックする. もし見つければ定理 14 より n は素数であることがわかる. つまり, 元々の素数 q のほぼ 2 倍のビット長を持つ素数 n を構成することができたことがわかる. ここで, (20) 式を満たす a を見つけることに関する計算時間に関しては, 3.4 節の (16) 式前後でも述べたように効率的に実行可能であると期待できる.

この手順を帰納的に用いて, 目的とするビット長の素数を構成するのが Shawe-Taylor 法である. 詳細な記述は ANSI X9.80[4] 又は [64] を参照. また, 次節で述べる Maurer 法のアルゴリズム記述アルゴリズム 4 でのステップ 4 において常に r を $1/2$ に設定することでも Shawe-Taylor 法のアルゴリズム記述は行える.

また, 例えば [4] では, 40 ビットの素数を生成した時の以下の例を述べている.

まずランダムな 21 ビットの素数 $q = 1832371$ を試行割算法 (3.2 節) を用いて生成し, $R = 263840$ ($R < q$) に対し, $n = 1 + 2Rq = 96690559281$ (40 ビット) を構成する. そして, $a = 41767655812$ に対し, (20) 式の成立を確かめることで, 40 ビットの素数 n を得ている.

3.5 節の ECPP 法でも, ある素数の素数性証明を, 短いビット長の素数の素数性証明に帰着していったことに注意する. 但し, 3.5 節でも触れたように定理 14 の類似である定理 9, 定理 10 を基にした各方法 (PPLS 法, Kraitchik-Lehmer 法) は無条件では効率的な素数判定法とはならなかったことにも注意する.

4.4.2 Maurer 法

Maurer 法も Shawe-Taylor 法と同様, 定理 14 を帰納的に用いてアルゴリズムが構成されるが, アルゴリズム 4 では, パラメータ $r (0.5 \leq r \leq 1)$ によって, 各段階で構成される素数 n に関して, $n - 1$ の最大素因数 q の大きさをランダムにとることができる.

$x \in \mathbb{R}$ を十分大きいとして, n を $[1, x]$ からランダムにとってきた時, $r (0.5 \leq r \leq 1)$ に対し n の最大素因数 n_1 が x^r 以下になる確率 $s = s(r)$ は, 近似的に $1 + \log r$ で与えられることが知られている ([42]).

これによって, アルゴリズム 4 (M_Random_Prime) 中の $s = 1 + \log r$ をランダムにとることで, 生成される素数の一様性を確保することができる. つまり, $0.5 \leq r \leq 1$ である r に対し,

$$s = s(r) = \text{Prob}[n_1 \leq x^r | n \leftarrow_R [1, x]] \sim 1 + \log r \quad (x \rightarrow \infty) \quad (21)$$

という評価を利用する.

アルゴリズム 4 Maurer 法 (M_Random_Prime)

入力 $b \in \mathbb{Z}_{>0}$

出力 b ビット素数 y

- 1: if $b < 20$ then
 - 2: ランダムな b ビット素数を試行割算法により生成し, それを出力する.
 - 3: else
 - 4: s が $[0, 1]$ 内で一様になるように (21) 式に従って $r (0.5 \leq r \leq 1)$ を選択する
 - 5: $q \leftarrow \text{M_Random_Prime}(\lfloor r \cdot b \rfloor + 1)$
 - 6: R を $n = 2Rq + 1$ が b ビットになるようにランダムに選び, (20) 式の成否チェックにより素数判定して素数 n が得られるまで繰り返し, 素数 n が得られたら, それを出力.
 - 7: end if
-

5 各種アルゴリズムの比較

5.1 概要

前節までに調査した結果を基にして, 表 5 の各項目に関して各素数判定・生成法の比較検討を行う. また, 各項目に関して記述されている節番号も併記する. 以下の項目で“確率的か決定的”とあるのは, アルゴリズムが確率的な記述であるか決定的な記述である

かということ (乱数を用いるか用いないか) を指すのであり, 素数判定法として確率的か確定的であるかということとは異なる (つまり, この違いは 4.2 節で触れた (擬似) 乱数生成法を必要とするかどうかということになる)。

数論仮定なし漸近的計算量 (確率的か決定性的かも含めて)	5.1, 5.2 節
数論仮定あり漸近的計算量 (確率的か決定性的かも含めて)	5.1, 5.2 節
確率的素数判定法の場合は, その $(c \rightarrow pp)$ 誤り確率	5.2 節
実時間量 (512, 1024 ビットというような具体的な値に対して)	5.2 節
メモリ使用量	5.3 節
実装の難易度	5.4 節
素数性証明書 (primality certificate) の有無	5.2 節
生成された素数の一様性	5.4 節

表 5: 素数判定・生成法の比較項目

本報告書では, ほとんどのアルゴリズムは上記の意味で確率的アルゴリズムとして扱うが, 試行割算法, Cohen-Lenstra 法, AKSL 法, AKS での予想を用いた方法に関しては決定性的アルゴリズムとして扱う。

5.2 節で各アルゴリズムの計算量を素数候補数 n の (自然) 対数 $\log n$ の関数として表すが, $\log n$ 程度の大きさの数の乗算は $(\log n)^2$ のビット演算で行えるという前提で評価する。つまり, 本報告書では暗号用途での素数判定, 生成法を考察するのでその前提で比較するのが妥当と思われるからである (よって, AKSL 法, AKS 予想法に関しては, 3.7, 3.8 節と少し異なる漸近計算量を考える)。

また, AKSL 法では, 法とする多項式の次数が $O((\log n)^2)$ 程度と大きいので, 高速フーリエ変換を使った多項式乗算計算量を採用する。AKS での予想を用いた方法に関しても, 漸近計算量は AKSL 法に準じて高速多項式乗算法を適用する。

5.2 計算量比較

表 6 は, 確率的素数判定法の計算時間を比較した表であり, 表 7 は, 確定的素数判定法の計算時間を比較した表である。それらの説明は, 2, 3 節中の各アルゴリズムの説明の項を参照することとする。また, AKS 予想法は, あくまで 3.8 節 予想 1 が正しいことを仮定して多項式時間確定的素数判定法になるので, その点では 2.4 節末尾でも述べたように Solovay-Strassen 法, Miller-Rabin 法と同様であるが, AKSL 法との比較の観点から表 7 に掲載する。

確率的素数判定法の比較であるが, Fermat 法, Solovay-Strassen 法に関しては, Miller-Rabin 法と比べて $((c \rightarrow pp)$ 誤り確率, $(pp \rightarrow c)$ 誤り確率のどちらかを考慮に入れても) 計算時間で優位性を示すことはできない。

	t 基底テスト時の漸近計算量と (c→pp) 誤り確率に対する既知上界値	予想 (GRH) の下での漸近計算量	実際の計算時間
F 法	$\frac{(\frac{3}{2} + o(1))t(\log n)^3}{1}$	—	MR 法と同等かそれ以上の時間
SS 法	$\frac{(\frac{3}{2} + o(1))t(\log n)^3}{(\frac{1}{2})^t}$	$2(\log n)^5$	MR 法と同等かそれ以上の時間
MR 法	$\frac{(\frac{3}{2} + o(1))t(\log n)^3}{(\frac{1}{4})^t}$ 以下,	$2(\log n)^5$	Pentium IV 1.8 GHz 上で, 1024 ビット素数 (テスト基底 2 個) に対し, 約 70m 秒.
LL 法	$\frac{(3 + o(1))t(\log n)^3}{(\frac{4}{15})^t}$	—	1 つの基底に対するテストは MR 法の約 2 倍の時間
FG 法	$\frac{(\frac{9}{2} + o(1))t(\log n)^3}{(\frac{1}{7710})^t}$	—	1 つの基底に対するテストは MR 法の約 4.5 倍の時間

表 6: 確率的素数判定法の計算時間比較

また, Lucas-Lehmer 法に関しては, 表に見る通り, Miller-Rabin 法に比べての特に優れた点が, 少なくとも現時点では判っていないので, それ自体を Miller-Rabin 法の代替として考えることは難しいが, 2.5 節でも述べたような Miller-Rabin 法との併用は十分意味があると思われる.

Frobenius-Grantham 法に関しては, その (c→pp) 誤り確率評価は Miller-Rabin 法より低いのであるが, (pp→c) 誤り確率評価に関しては, (19) 式からわかる以上の評価は得られていない. また, 2.6 節でも述べたように最近, [22] において, (pp→c) 誤り確率評価を有する拡張法が得られたが, 比較的最近のことであるので, それは今後, 研究者によって検討されなければならない. それに対して, Miller-Rabin 法は長年, 確率的素数判定法として一般に使われてきた実績があるので, 今後も最も一般に使われ続けていくと思われる.

	漸近計算量	仮定となる条件 (Assump) 等	実際の計算時間
試行割算法	$O(n^{1/2}(\log n)^2)$	—	—
PPLS 法	$O((\log n)^4)$	$n-1, n+1$ の部分的分解が既知である時のみ適用可能	1024 ビット素数に対して, MR 法の約 1000 倍 (1.8GHz で 70 秒) 以下の時間
KL 法	$O((\log n)^4)$	$n-1$ の素因数分解が既知である時のみ適用可能	1024 ビット素数に対して, MR 法の約 1000 倍 (1.8GHz で 70 秒) 以下の時間

ECPP 法	AM法のヒューリスティックな計算量は $O((\log n)^{6+\epsilon})$.	左記計算量評価が成立しない素数の確率は $O(2^{-(\log n)^{\frac{1}{\log \log \log n}}})$.	DEC 3000 上, 512 ビット素数に対し平均 35.81 秒, 最悪 68.53 秒, 最良 18.17 秒. $O((\log n)^6)$ の計算量を使って 1024 ビット時の時間を見積もると平均約 2292 秒
CL 法	ある定数 c に対し, $O((\log n)^{c \log \log \log n})$.	—	DEC Alpha 500 上, 10 進 200 桁素数に対し平均 23.83 秒, 最悪 41.40 秒, 最良 13.70 秒. 1024 ビット素数に対する同環境での時間見積りは 455.95 秒. ²
AKSL 法 ³	$\tilde{O}((\log n)^{13})$. ヒューリスティックには $\tilde{O}((\log n)^7)$.	—	700MHz PC 上, 10 進 9 桁素数に対し, 約 6 分 9 秒. 1024 ビットに換算すると, 約 65 万年.
AKS 予想法 ³	$\tilde{O}((\log n)^4)$	3.8 節 予想 1 が正しいことを仮定	Pentium IV, 1.8GHz 上, $r \leq 100$ と取れば, 1024 ビット素数に対し, 約 62 秒以下. $r \sim 4000$ なら約 2 日以下と見積もられる.

表 7: 確定的素数判定法の計算時間比較

試行割算法は, 単独で暗号鍵 (パラメータ) 生成用途の素数判定法として用いられることはないのであるが, 実装に関して述べた後節でもみるように他の各種判定法と複合して用いられることがあるので, その計算量に関しても一瞥する必要がある。

PPLS 法, Kraitchik-Lehmer 法に関しては, 表 7 にもあるように, その方法は無条件では一般の素数判定法にはならないので, (少なくとも現時点では) 暗号鍵 (パラメータ) 生成用途に用いるのは困難である。

ECPP 法, Cohen-Lenstra 法に関しては, [48, 45] が 1998 年発表の論文であることを考えると, 現在一般に用いられている PC では表 7 の数字より更に短時間で 512, 1024 ビット数の判定が可能であろうと思われることに注意しておく。よって, 計算時間という点からみるなら, それほど多くない個数の素数を生成するには実用的になってきたと思われる。但し, 大量の暗号鍵生成という点では依然考慮が必要であろう。計算時間以外の問題点に関しては, 5.4 節参照。

また, 実際的に Cohen-Lenstra 法 (の改良) の方が, ECPP 法より高速である ([45]) 反面, ECPP 法は素数性証明書を生成できること ([48] 及び 3.5 節参照) に注意する。

²[36] にあるプログラムを用いて 512 ビット素数の判定を行わせたところ 47 秒かかり, 1024 ビット素数の判定には 10 分 8 秒 (608 秒) かかり $\log \log \log 512 = 0.604 \dots$, $\log \log \log 1024 = 0.6606 \dots$ であるので, 考えている範囲で $\log \log \log n$ を定数として $c \log \log \log n$ の値を逆算すると, 3.6933... と見積もられる。

³漸近計算量に関しては, 5.1 節内の説明を参照。

また、素数生成法に関しては、素数を構成する Maurer 法は、[43] では、繰り返し回数 $t = 1$ の Miller-Rabin 法より少し速度性能が悪いがほぼ同等とあり、あまり利点がないと述べられている。

真にランダムな鍵を選択するためにも、敢て Maurer 法のような生成される素数が真に一様ではない方法を用いる必要はないと思われる。

そして素数判定法を用いた素数生成法を用いる場合には、Random Choice とするか Incremental Search にするかは速度を重視するかどうかによって、使用環境に応じて選択すべきと思われる。

5.3 計算機資源比較

各確率的又は確定的素数判定法に関しての使用メモリ容量比較を行う。

- 各確率的素数判定法に関しては、計算途中において $\log n$ ビットの整数を数個 (せいぜい 10 個) 保持できる領域があれば良いので、通常の PC 上で素数判定法としてのみ用いる時には、メモリに関する制約はほとんどないと言ってよい。
- 確定的素数判定法の PPLS 法と Kraitichik-Lehmer 法に関しても、通常の PC 上ではメモリの制約はほとんどないと言っていいと思われる。
- 試行割算法を行う時には、割る素数の表をどこまで用意しておくかということで、メモリ容量に関して注意が生じ得る。
- ECPP 法では、楕円曲線演算を行う時にはさほどメモリに関して制約は生じないと思われるが、位数が既知の楕円曲線を得るために、[6] の方法では、(虚 2 次体の) 判別式という負整数でパラメータ付けされているある代数的数の定義多項式 (即ち、その係数) のデータが必要となる。[49] には 2244 個の判別式に関するデータが添付されており、その総容量は約 10M バイトである。現在のメモリの通常の容量を考慮に入れると、ECPP 法は、通常の PC 上での実行に関しては特に制約はないと言っていいと思われる。
- Cohen-Lenstra 法の場合には、[44] でも指摘されているように定理 12 中のパラメータ q が大きくなると、通常の PC に搭載されるようなメモリ容量を超える危険性があるが、[36] を用いた計算機実験を行ってみるとわかるように n が 512, 1024 ビットの整数であるような暗号用途では、その心配はいらぬ。実際に PC 上で [36] を動作させ、(Windows XP のタスクマネージャ機能を用いて、) メモリ使用量を調査した所、高々 3M バイトで 512, 1024 ビット整数に対して動作していることが確認できた。これらのことは Cohen-Lenstra 法が、暗号用途に使用する場合にはテーブルファイルを考慮に入れても現実的なメモリ容量で動作できることを示している。
- AKSL 法の場合、その計算の主な部分は多項式の (円分多項式を法とする) べき乗算であり、512, 1024 ビットの n に対して扱う多項式次数が大きくなり得るので、通常考えられるようなメモリ容量限界を超える可能性がある。[10] で指摘されているよう

に定理 13 のパラメータ r は現実的には $(\log n)^2$ のオーダーで取り得ると思われるので、 r を仮に $(\log n)^2$ としてみると、 n が 1024 ビットの時には、 r はほぼ 10^6 と考えられ、その多項式の係数は各々 1024 ($\sim 10^3$) ビットの整数であるので、多項式を表現するだけで、はぼ 10^9 ビット ($\sim 1.2 \cdot 10^8$ バイト = $1.2 \cdot 100$ M バイト) 必要になると考えられる (これは、あくまでも漸近関数から見積もった値であることに注意する)。これ自体現実的な数値ではあるが、メモリ容量を配慮した実装をする必要が生じることがわかる (しかし、AKSL 法に関して、より問題になるのは 5.2 節でも述べたように時間計算量であることに注意しておく)。

- AKS での予想を用いた方法に関しては、 r は小さい整数に設定できるので、通常の PC 上で動かす場合にはメモリに関する問題は生じないと思われる。

5.4 実用性

大量の鍵生成を短時間に要請する一般的な使用法に最も適した方法は、5.2 節の表 6、表 7 から確率的素数判定法、特に Miller-Rabin 法を用いるのが最も良いと思われる。確定的素数判定法においても、Cohen-Lenstra 法 (の拡張) や ECPP 法のように許容できる計算時間で遂行できるものもあるが、それら方法は、[4] の記述にもあるようにアルゴリズムの複雑さ故に、プログラム中のバグ検出が大変であり、実用的に用いるのには現段階では考慮を要する。

直接素数を生成する方法に関しては、生成される素数の一様性を考慮した Maurer 法を用いた場合でさえも、完全に一様に選ぶことはできない。また、[43] で述べられているように、速度性能を考慮しても 4.4 節で述べた方法には、4.3 節での方法と比べてあまり利点がないと思われる。

6 実装性評価

前節までの我々の調査によって、様々な素数判定アルゴリズムのうち、確率的素数判定法である Miller-Rabin 法は効率の良い素数判定法であることが確かめられた。また、現在の暗号システムへの応用としては、Miller-Rabin 法が最も広く使用されていると思われる。さらに、8 節で述べるように、Miller-Rabin 法は様々な標準でも採用されている。

そこで、本節では Miller-Rabin 法に基づく素数生成法の実装性について考察する。特に、実際の暗号において最もよく必要とされる 512 ビットと 1024 ビットの素数を生成するための最適な実装方法を、速度およびメモリの観点から実験的に確かめる。また、既存のフリーソフトウェアの素数生成関数との比較考察を行う。

6.1 高速化手法

Miller-Rabin 法のアルゴリズムを見ると、高速化に関して考慮すべき事項に次のものがあることがわかる。

- べき乗剰余演算の高速化
- Miller-Rabin 法の基底のとり方
- Miller-Rabin 法の前処理として小さな素数で試行割算を行うこと

Miller-Rabin 法においては，定理 4 で見たようにランダムに選ばれた基底に対するべき乗剰余演算を行うことが必要である．べき乗剰余演算は，RSA 暗号においても必要な演算である．その計算量はビット数の 3 乗に比例することから，Miller-Rabin 法においても RSA 暗号においても，計算時間の大部分を占める演算部となり，高速化が課題となる．べき乗剰余演算の高速化に関連して，Miller-Rabin 法の基底の取り方を工夫することも考察する．

また，与えられた大きな数は素因数として小さな素数を含む確率が高い [61] ことから，Miller-Rabin 法を実行する前に小さな素数での試行割算を行うことにより合成数判定を行うことも有力な高速化手法である．

次節以下で，これらの高速化手法に関して考察する．

6.1.1 べき乗剰余演算

べき乗剰余演算は，Miller-Rabin 法や RSA 暗号などで計算時間が支配的となる演算部である．そのため，様々なべき乗剰余演算アルゴリズムが提案されている [28, 43]．Sliding Window 法は，高速なべき乗剰余演算アルゴリズムの 1 つである．本報告書においては Sliding Window 法によるべき乗剰余演算をベースに，Miller-Rabin 法の高速化を検討する．

まず，べき乗剰余演算アルゴリズムのうち，最も基本的な左 Binary 法をアルゴリズム 5 に示す．

アルゴリズム 5 左 Binary 法によるべき乗剰余演算

入力 正整数 x, n, t ビットの整数 $k = (k_{t-1}k_{t-2} \cdots k_1k_0)_2$, ($k_i \in \{0, 1\}$, $k_{t-1} = 1$)

出力 $y \equiv x^k \pmod n$

- 1: $A \leftarrow x$
 - 2: **for** i from $t - 2$ down to 0 **do**
 - 3: $A \leftarrow A^2$
 - 4: **if** $k_i = 1$ **then**
 - 5: $A \leftarrow A \cdot x$
 - 6: **end if**
 - 7: **end for**
 - 8: A を出力する
-

べき乗剰余演算においては，入力された基底 x の乗算および自乗算が繰り返し実行される．自乗算はべき指数 k のビット長に応じた回数，乗算は k のビット “1” の数に応じた回数が必要である．自乗算の演算量を S ，乗算の演算量を M とし，べき指数 k のビット長を t とすると，左 Binary 法の計算量は，平均

$$(t - 1)S + \frac{t}{2}M$$

となる。

べき乗剰余演算においては、自乗算はべき指数のビット長分必要となるが、アルゴリズムを工夫することにより乗算の回数は減らすことができる。乗算回数を減らすことが、べき乗剰余演算の高速化につながる。Binary 法においては乗算はべき指数の 1 ビットごとに行わなければならないが、複数ビットごとにまとめて乗算するように工夫されたアルゴリズムに Ary 法 [28, 43] がある。

Ary 法をさらに改良したアルゴリズムが Sliding Window 法である。そのアルゴリズムをアルゴリズム 6 に示す [28, 43]。

アルゴリズム 6 Sliding Window 法によるべき乗剰余演算

入力 正整数 x, n, t ビットの整数 $k = (k_{t-1}k_{t-2}\cdots k_1k_0)_2$, ($k_i \in \{0, 1\}$, $k_{t-1} = 1$), Window 幅 w

出力 $y \equiv x^k \pmod{n}$

- 1: 事前計算
 - 2: $x_1 \leftarrow x, x_2 \leftarrow x^2$
 - 3: **for** i from 1 to $(2^{k-1} - 1)$ **do**
 - 4: $x_{2i+1} \leftarrow x_{2i-1} \cdot x_2$
 - 5: **end for**
 - 6: 本計算
 - 7: $A \leftarrow 1, i \leftarrow t - 1$
 - 8: **while** $i \leq 0$ **do**
 - 9: **if** $k_i = 0$ **then**
 - 10: $A \leftarrow A^2, i \leftarrow i - 1$
 - 11: **else**
 - 12: $i - l + 1 \leq w$ かつ $e_l = 1$ であるような最長のビット列 $k_i k_{i-1} \cdots k_l$ を見つける
 - 13: $A \leftarrow A^{2^{i-l+1}} \cdot x_{(k_i k_{i-1} \cdots k_l)_2}$
 - 14: $i \leftarrow l - 1$
 - 15: **end if**
 - 16: **end while**
 - 17: A を出力する
-

Ary 法も Sliding Window 法もアルゴリズムへの入力として window 幅 w を設定する。事前計算ステップにおいて window 幅に応じた数の入力 x のべき乗を計算する。window 幅 w を大きくすると、事前計算ステップの演算量が増加し、本計算ステップの演算量は乗算回数が減るので減少する。window 幅 w を小さくすれば、事前計算ステップの演算量は減少し、本計算ステップの演算量は増加する。したがって、適切な window 幅を決めることが必要である。適切な window 幅は、法 n のビット長や実際の実装における乗算の時間などによって決まる。したがって、実際の実装によって、実験的にビット長に応じた最適幅を確かめることが必要である。

また、事前計算された x のべき乗値は、べき乗剰余演算が終了するまでメモリに保持しておかなければならない。したがって、高速化の代償として必要メモリ量が増加すること

に注意が必要である．メモリ増加量は window 幅に依存する．

なお，Sliding Window 法において Window 幅 w が 1 の場合は，アルゴリズムは Binary 法と等価となる．

6.1.2 Miller-Rabin 法における基底のとり方

Miller-Rabin 法ではランダムに選ばれた複数の基底に対して合成数判定を行うが，その基底は 2 としてもよいことに着目する．基底を 2 とすると，左 Binary 法アルゴリズム 5 はアルゴリズム 7 のように変形される．

アルゴリズム 7 左 Binary 法による 2 のべき乗剰余演算

入力 正整数 n ， t ビットの整数 $k = (k_{t-1}k_{t-2}\cdots k_1k_0)_2$ ， $(k_i \in \{0, 1\}, k_{t-1} = 1)$

出力 $y \equiv 2^k \pmod n$

- 1: $A \leftarrow 2$
 - 2: **for** i from $t - 2$ down to 0 **do**
 - 3: $A \leftarrow A^2$
 - 4: **if** $k_i = 1$ **then**
 - 5: $A \leftarrow A \cdot 2$ (シフト演算でよい)
 - 6: **end if**
 - 7: **end for**
 - 8: A を出力する
-

ループ中，べき指数のビット “1” において，通常の Binary 法では乗算が行われるが，2 のべき乗の場合乗じる数が 2 であるので，乗算をシフト算で代用できる．シフト算は乗算と比べて高速である．

したがって，基底 2 の場合に対する Sliding Window 法よりも基底 2 専用のアルゴリズム 7 が高速になれば，Miller-Rabin 法において基底を選ぶ時，最初は 2 とすることは高速化につながる．(Sliding Window 法においては，基底が 2 であってもシフト算は適用できないことに注意．)

6.1.3 小さな素数による試行割算

ランダムに選ばれた数は，素因数として小さな素数をもつ確率が高い [61]．したがって，Miller-Rabin 法に基づく素数生成アルゴリズム 2 (Random Choice) およびアルゴリズム 3 (Incremental Search) 中，Miller-Rabin 法による素数判定を行う前に，小さな素数による試行割算を行って合成数をふるい落とすことは有効である．

この際，どの程度の数の小素数に対して試行割算を実行することが最も高速になるかは，実験的に確かめる必要がある．Random Choice による素数生成の場合は，ある程度理論的な解析 [13] がなされており， r を試行割算を行う素数の大きさの上限とすれば，次

の式を満たすようにとるのが最適と見積もられている。

$$r = \frac{R_2}{D \log(R_2/D)} \quad (22)$$

ここで、 R_2 は 2 を基底とした Miller-Rabin 素数判定 1 回に要する時間であり、 D は試行割算時の小素数 1 個での割算の時間とする。

6.2 速度評価

前節で述べた高速化手法について、ソフトウェア実装により有効性を実験的に確認する。また、window 幅、小素数の数などの最適値を導出する。

実験に用いた速度計測環境を表 8 に示す。

CPU	Pentium IV 1.6 GHz
言語	C 言語
OS	RedHat Linux 8.0
コンパイラ	gcc version 3.2
コンパイルオプション	-O2
多倍長演算ライブラリ	GNU MP version 4.1.2
RAM	512 MB

表 8: 速度計測環境

なお、速度計測は、いずれの実験もランダムに生成した 10,000 個に対する平均時間を算出した。ただし、素数生成速度計測の場合は、10,000 個の素数を生成した時に、1 個生成するのに必要な時間の平均値を算出した。また、乱数生成には、CRYPTREC で決められた電子政府用暗号リスト案 [58] にリストされている、SHA-1 に基づく乱数生成アルゴリズムを使用した。

6.2.1 べき乗剰余演算速度

まず、べき乗剰余演算を実装し、その速度を計測する。べき乗剰余演算のアルゴリズムには、前節で述べたように Sliding Window 法を用いる。window 幅を 1 から 8 まで変化させ、速度を計測する。512 ビットの場合を表 9 に、1024 ビットの場合を表 10 に示す。なお、window 幅が 1 の場合は、アルゴリズムは Binary 法と等価になる。

表 9, 10 には速度と共に必要な乗算回数および自乗算回数の計測も示している。window 幅の増減は、事前計算ステップの計算量と本計算ステップの計算量とのトレードオフを生じさせることがわかる。512 ビットの場合は window 幅 5、1024 ビットの場合は window 幅 6 が最高速であった。

Window 幅	事前計算		本計算		速度 (m 秒)	メモリ (バイト数)
	自乗算回数	乗算回数	自乗算回数	乗算回数		
1	0	0	511	255.75	7.03	64
2	1	1	511	170.43	6.28	128
3	1	3	511	127.84	5.79	256
4	1	7	511	102.44	5.60	512
5	1	15	511	85.52	5.50	1,024
6	1	31	511	73.29	5.55	2,048
7	1	63	511	64.22	5.77	4,096
8	1	127	511	57.12	6.35	8,192

表 9: べき乗剰余演算 $y \equiv x^k \pmod{n}$ の速度 (512 ビット)

Window 幅	事前計算		本計算		速度 (m 秒.)	メモリ (バイト数)
	自乗算回数	乗算回数	自乗算回数	乗算回数		
1	0	0	1023	511.76	45.46	128
2	1	1	1023	340.96	40.12	256
3	1	3	1023	255.98	37.42	512
4	1	7	1023	204.90	35.93	1,024
5	1	15	1023	170.84	35.08	2,048
6	1	31	1023	146.40	34.85	4,096
7	1	63	1023	128.25	35.21	8,192
8	1	127	1023	113.98	36.83	16,384

表 10: べき乗剰余演算 $y \equiv x^k \pmod{n}$ の速度 (1024 ビット)

また，表中には必要なメモリ量も示した．メモリ量は，

事前計算ステップで計算される x のべき乗の個数 $\times x$ のバイト数

として評価した．速度的に最適な場合の必要メモリ量は，512 ビットべき乗剰余演算の場合 1 キロバイト，1024 ビットべき乗剰余演算の場合 4 キロバイトである．この量は通常の PC およびワークステーションにおいては問題にならない量である．しかし，IC カードなどに実装する場合は実装されるメモリ量に応じて window 幅を減らす必要性があり得る．

6.2.2 2 のべき乗剰余演算速度

次に，べき乗剰余演算において基底が 2 の場合のアルゴリズム 7 の速度を計測する．速度計測結果を表 11 に示す．

512 ビットの場合も 1024 ビットの場合も，Sliding Window 法によるべき乗剰余演算よりも高速である．したがって，Miller-Rabin 法において選ぶ基底は，最初は 2 とすること

ビット長	自乗回数	乗算回数	速度 (m 秒)
512 ビット	511	256.14	4.50
1024 ビット	1023	512.07	29.11

表 11: 2 のべき乗剰余演算 $y \equiv 2^k \pmod{n}$ の速度

で Miller-Rabin 法を高速化できる。

6.2.3 素数生成速度

次に Miller-Rabin 法に基づく素数生成の速度を計測する。べき乗剰余演算には、前節で得られた速度的な最適設定である次の設定を用いる。

- 512 ビットの場合：window 幅 5 による Sliding Window 法
- 1024 ビットの場合：window 幅 6 による Sliding Window 法

また、基底は最初は 2 を選び、基底 2 のべき乗剰余演算アルゴリズム 7 を用いる。

まず、Miller-Rabin 法を実行する前に小さな素数による試行割算を行う際の、素数の数の最適値を導出する。評価式 (22) によれば、素数の数の最適値は与えられた数を小素数で割る速度と、Miller-Rabin 法において基底 2 の場合の Miller-Rabin1 回の速度とで評価できる。基底 2 の場合の速度は、基底 2 のべき乗剰余演算速度 (表 11) とほぼ同じである。そこで、与えられた数を小素数で割る速度計測する。結果を表 12 に示す。

512 ビット	0.609 μ 秒
1024 ビット	1.215 μ 秒

表 12: 多倍長整数 (512 ビットまたは 1024 ビット) を単ワード素数で割る速度

表 12 と表 11 と評価式 (22) とから、試行割算における最適な小素数の数を見積もると次のようになる。

- 512 ビットの場合：1898 個
- 1024 ビットの場合：5475 個

実際の実装上の最適値は、実験的に求める必要がある。そこで、この評価で得られた個数の周辺について、実際の実装により速度を計測し、最適値を求める。小素数の数を変化させ、それに対する素数生成速度を表 13、表 14 に示す。

素数生成は、Random Choice による方法、Incremental Search による方法の両方について実装した。また、繰り返し回数は、誤判定確率が 2^{-80} 程度になるように設定した。512 ビットの素数生成の場合は、Random Choice と Incremental Search の両方において試行割算を行う小素数の数は 500 個の場合が最高速であった。また、1024 ビットの場合は、Random Choice の場合は 2500 個、Incremental Search の場合は 4200 個が最高速であった。

小さな素数の数	速度 (m 秒)	
	Random Choice	Incremental Search
10	128.790	121.478
100	80.257	76.061
400	71.977	67.939
500	71.932	63.630
600	71.989	68.242
1000	74.677	70.447
10000	167.063	151.133

表 13: Miller-Rabin 法に基づく素数生成速度 (512 bit , 繰り返し回数=6)

小さな素数の数	速度 (秒)	
	Random Choice	Incremental Search
10	3.375	3.408
1000	1.449	1.409
2400	1.393	—
2500	1.391	—
2600	1.393	—
4100	—	1.331
4200	—	1.330
4300	—	1.333
5000	1.445	1.438
10000	1.569	1.560

表 14: Miller-Rabin 法に基づく素数生成速度 (1024 bit , 繰り返し回数=3)

6.3 高速化のための最適設定

以上の実験結果より, Miller-Rabin 法に基づく素数生成を行う場合の, 高速化のための最適設定を表 15 にまとめる. また, その時に必要なメモリ量も記載する. メモリ量は, べき乗剰余演算における事前計算テーブルの大きさと, 試行割算における小素数の数とから算出した.

なお, 速度上最適な設定は, 実際の多倍長演算関数の速度などによって変わり得るものである. 本調査において用いた多倍長演算ライブラリ GMP と実装環境 (表 8) では, 速度上の最適設定は表 15 の通りであった. この設定は, 多くの実装環境における最適設定として一応の目安となるが, 環境が変わればその実環境上で速度計測を行うことにより, 最適設定を探ることが望ましい.

また, 速度上最適な設定であっても, IC カードなどメモリをあまり搭載できない実装環境においては, 使用可能なメモリ量に応じて設定を変える必要がある. 表 15 の設定では数キロバイトのメモリを必要とする. この量のメモリを搭載できない場合は, べき乗剰

素数生成法	ビット長	Sliding Window 法の Window 幅	素数テーブルの大きさ	メモリ使用量
Random Choice	512	5	500	1.5 Kbyte
	1024	6	2500	6.5 Kbyte
Incremental Search	512	5	500	1.5 Kbyte
	1024	6	4200	8.2 Kbyte

表 15: Miller-Rabin 法に基づく素数生成法の高速化のための最適設定 (繰り返し回数:512 ビットの時 6, 1024 ビットの時 3)

余演算における window 幅を小さくすることなどを検討する必要がある。

6.4 フリーソフトウェアとの比較

本稿では、多倍長演算ライブラリとしてフリーソフトウェアである GNU MP version 4.1.2(GMP)[25] を用いた。GMP は高速な多倍長演算ライブラリとして知られ、GMP を用いて暗号アルゴリズムの実装研究を報告している論文も多い。GMP 自体にも

- Miller-Rabin 法による素数判定関数 `mpz_probab_prime_p()` と、
- これを用いた Incremental Search による素数生成関数 `mpz_nextprime()`

がある。

素数判定関数では、与えられた数 n に対して、Miller-Rabin 法を実行する前に、 $(n$ のビット長/30)² 未満の素数に対する試行割算を実行している。また、素数生成関数においては、Miller-Rabin 法をサブルーチンとして呼ぶ時は、繰り返し回数は常に 2 としている。繰り返し回数 2 に固定という設定は、特に 512 ビットの素数判定を行う場合十分な回数ではなく、生成された素数を暗号用途に使うべきではない。

GMP の素数生成関数は、暗号用途に使うことを目的としていないので、そのまま使うことには注意が必要である。(素数判定関数単独なら、繰り返し回数を入力パラメータとして設定できる。)

また、GMP の素数生成関数を用いて素数生成の速度を計測した結果、同測定環境において素数 1 個を生成する平均時間は、

- 512 bit : 182.660 m 秒
- 1024 bit : 1961.600 m 秒

であった。本稿における実装の方が、誤判定確率を考慮して繰り返し回数を多く適切に設定しているにも関わらず、高速である。

7 暗号鍵としての素数の安全性条件

素数は暗号アルゴリズムにおいて重要な役割を果たす。RSA 暗号ではユーザの秘密鍵として用いられ、DSA 署名や楕円曲線暗号では、公開鍵として用いられる。RSA 暗号、DSA 署名および楕円曲線暗号は各種標準でも採用され、広く使用されている暗号である。そこで、本節では、この3つの暗号に用いる素数の要件についてまとめる。

7.1 RSA 暗号

RSA 暗号はビット長の等しい2つの異なる素数 p, q の積 n を素因数分解することの難しさに安全性の根拠をおいている。素数 p と q は秘密鍵であり、法 n を素因数分解できれば RSA 暗号は完全に破られる。したがって、RSA 暗号において使用する素数 p と q に関する要件は、

- $n(= pq)$ を素因数分解することが難しいこと

である。

7.1.1 素因数分解アルゴリズム

素因数分解アルゴリズムに関する研究は数多くあるが、アルゴリズムは、計算量が分解したい合成数 n の大きさで決まるもの(合成数依存型)と、 n の素因数 p の性質によって決まるもの(素因数依存型)の2種類に大別される。前者には

- 一般数体ふるい法 [38]

などがあり、後者には

- 楕円曲線法 [40]
- Fermat 法 [14]
- Pollard の $p - 1$ 法 [53]
- Williams の $p + 1$ 法 [68]

などがある。RSA 暗号に用いる素数は、これらの素因数分解アルゴリズムに対して計算量的に素因数分解が難しくなるように選ばなければならない。これらの素因数分解アルゴリズムに対する安全性条件として、次のような条件が考えられる。

- p と q は素数であること。
- p と q は大きく、かつビット長が等しいこと(一般数体ふるい法および楕円曲線法に対する安全条件)。
- p と q は大きさの差が小さくないこと(Fermat 法に対する安全条件)。

- $p - 1$ と $q - 1$ は大きな素数で割り切れること ($p - 1$ 法に対する安全条件) .
- $p + 1$ と $q + 1$ は大きな素数で割り切れること ($p + 1$ 法に対する安全条件) .

一般数体ふるい法の計算量は

$$O(\exp(c(\log_e n)^{1/3}(\log_e \log_e n)^{2/3})), \quad c = 1.901$$

であり, 素因数依存型アルゴリズムで最も高速な楕円曲線法の計算量は

$$O(\exp(c\sqrt{\log_e p \log_e \log_e p}) \cdot (\log_2 n)^2), \quad c = 1.414$$

である [20] . 一般数体ふるい法の計算量は n のビット長で評価され, 楕円曲線法の計算量は p のビット長で評価される . RSA 暗号のように $n = pq$ という型の合成数の素因数分解が難しくなるような p と q の要件を考える場合は, 上記 2 つの計算量を考慮すればよい .

なお, $p - 1$ 法や $p + 1$ 法に対して強い素数, すなわち $p - 1$ と $p + 1$ がともに大きな素数を選んだとしても, より高速な楕円曲線法に対する防御とはならない . 素数 p が $p - 1$ 法や $p + 1$ 法に対して強い素数かどうかということは, 楕円曲線法の効率には影響しない [62] . また, ランダムに生成された大きな素数であれば, $p - 1$ 法や $p + 1$ 法が効率的に働く素数が生成される確率は無視できる [62] .

また, 2 つの素数 p と q の大きさの差が小さい場合は $n = pq$ の素因数分解は, \sqrt{n} 付近の素数で試行割算することにより $n = pq$ を効率的に素因数分解できる (Fermat 法 [14]) . しかし, ランダムに選ばれた 2 つの大きな素数であれば, Fermat 法が効率的に働く 2 つの素数が生成される確率は無視できる .

7.1.2 素因数分解に対して安全な素数の要件

電子政府で使用する暗号アルゴリズムを評価している CRYPTREC においては, 素因数分解問題の難しさに関する評価も行っている . CRYPTREC での評価によれば, 一般数体ふるい法および楕円曲線法に対して安全な素数は, 現時点では, p と q のビット長は 512 ビット以上, すなわち $n = pq$ のビット長は 1024 ビット以上とされている [20] .

以上をまとめれば, RSA 暗号に用いる素数の要件は,

- 素数 p と q はビット長が等しく, かつ $n(= pq)$ は 1024 ビット以上

となる . これは CRYPTREC Report 2001(暗号技術評価報告書)[20] で述べられている要件である . ただし, これは現時点での安全条件であり, 計算機性能の向上や素因数分解アルゴリズムの改良が行われた場合は, この要件は大きなビット長へ変更しなければならないことに注意 . 素因数分解問題の難しさに関するより詳細な解説は, CRYPTREC Report 2001(暗号技術評価報告書)[20], および今年度末出版が予定されている CRYPTREC Report 2002(暗号技術評価報告書) を参照のこと .

7.2 DSA 署名

DSA 署名は NIST により規格化されたデジタル署名方式である [24]。DSA 署名は有限体の乗法群上の離散対数問題を解くことの困難性に安全性の根拠をおいている。離散対数問題とは一般の有限群 G に対して定義される次のような問題である。

定義 6 (離散対数問題) G を有限群とする。 G の元 g と $u = g^x$ ($x \in \mathbb{Z}$) が与えられたとき、 x を求める問題を離散対数問題という [20]。

暗号アルゴリズムでは、有限群 G として有限体の乗法群、または有限体上定義された楕円曲線有理点群を用いることが多い。 DSA 署名は有限体の乗法群上の離散対数問題を利用している。 離散対数問題を解くアルゴリズムには、

- Pohlig-Hellman 法 [52]
- Index-Calculus 法 [27]

などがある。

DSA 署名においては、上記離散対数問題を解くアルゴリズムや他のアルゴリズムに対しても安全になるように、2つの素数 p, q を生成する手順が決められている。したがって、仕様書 [24] で定められた手順によって生成された素数であることが DSA 署名で使用するべき素数の要件となる。

DSA 署名に用いる素数 p と q が持つ性質、すなわち DSA 署名に用いる素数の要件は次の通りである。

- p と q は素数であること
- p は 1024 ビットであること
- q は 160 ビットであること
- $p - 1$ は q で割り切れること

現時点の DSA 署名の仕様 [24] においては、素数 p のビット長は 1024 が推奨されている。

なお、素数 p のビット長の上限は仕様上 1024 に制限されているが、将来の計算機の性能向上を考慮して素数のビット長をより大きくするように DSA 署名は仕様変更される予定である。 DSA 署名の仕様が変更された場合は、変更された仕様に合わせて上記要件は変更する必要がある。

7.3 楕円曲線暗号

楕円曲線暗号においては、素数は曲線を定義するパラメータとして使用される。楕円曲線暗号が安全性の根拠とする楕円曲線離散対数問題は、素数 p のみならず、曲線の他のパラメータも合わせて困難さが評価できる。素数 p のみから評価できる安全性条件には p は素数であることおよび p は大きいこと、が挙げられる。楕円曲線離散対数問題が定義さ

れる楕円曲線有理点群の大きさは、 p の大きさでおおよそ決まるため、 p の大小は楕円曲線離散対数問題の安全性の一応の目安となる。(p のみからでは完全な安全性評価はできない) CRYPTREC においては、楕円曲線離散対数問題の安全性評価も行っており、 p は160ビット以上を推奨している [20] .

したがって、楕円曲線暗号に用いる素数の要件は次の通りである .

- p は素数であること
- p は160ビット以上であること

8 標準化動向

8.1 ANSI

ANSI においては ANSI X 9.80, “American National Standard for Financial Services - Prime Number Generation, Primality Testing and Primality Certificates” [4], において素数判定法および素数生成法が標準化されている . 確率的素数判定法, 確定的素数判定法, およびそれら素数判定法に基づく素数生成法, 構成的に素数を生成する方法などが記載されている . これらのアルゴリズムに関しては, 全て本調査報告書において調査されている .

8.2 FIPS

NIST では, デジタル署名の標準, FIPS 186-2, “Digital Signature Standard (DSS)” において, DSA 署名用の素数を生成するアルゴリズムが記述されている . そのアルゴリズムは, 4.3.4 節で述べたように, Miller-Rabin 法に基づく素数生成法である . 擬似乱数生成関数としては, 一方向性関数を用いる方法が記述されている . 一方向性関数としては, SHA-1 を用いる方法と DES を用いる方法が記述されている .

8.3 CRYPTREC

日本においては, 高度情報通信ネットワーク社会形成基本法に基づく e-Japan 重点計画が推進されている . 電子政府におけるセキュリティ確保のために, 暗号評価プロジェクト “CRYPTREC” において, 電子政府で使用すべき暗号アルゴリズムの評価を行っている [21] .

公開鍵暗号は, CRYPTREC の公開鍵暗号技術評価小委員会において評価が行われ, 2002 年 11 月, 電子政府で使用すべき暗号のリスト案が作成された [58] . リスト案に挙げられている公開鍵暗号アルゴリズムを, 安全性の根拠とする数論問題によって分類すると表 16 のようになる .

CRYPTREC では現時点では素数判定法および素数生成法に関するリストは作成していないが, リスト案に記載されている暗号は, いずれも素数を秘密鍵または公開鍵として必要とする .

安全性の根拠とする数論問題	暗号アルゴリズム名
素因数分解問題	RSASSA-PKCS-v1_5, RSA-PSS, RSA-OAEP, RSAES-PKCS-v1_5
離散対数問題	DSA, DH
楕円曲線離散対数問題	ECDSA, ECDH, PSEC-KEM

表 16: 電子政府で使用すべき公開鍵暗号のリスト案 (2002 年 11 月 28 日)

リスト案に掲載された暗号へ用いる素数の安全条件としては、7 節で述べた、RSA 暗号の場合の安全性条件、DSA 署名の場合の安全性条件、または楕円曲線暗号の場合の安全性条件のいずれかが適用できる。

9 電子政府セキュリティシステムにおける素数生成法の要件について

電子政府においてはセキュリティを確保するために暗号を用いることが必須である。前節で見たように、電子政府で使用することが推奨される暗号アルゴリズムのリスト案に掲載されている公開鍵暗号は、いずれも素数を必要とする。本節では、電子政府セキュリティシステムにおいて使用する素数生成法の要件について、次の観点でまとめる。

- 素数判定の確実性および素数生成の確実性と一様性
- 素数生成の実装性
- 素数を暗号鍵として用いる場合の安全性条件

これらの観点から、9.4 節で今回開発すべきソフトウェアの要求仕様を述べる。さらに 9.5 節において、暗号モジュールの評価制度である CMVP において素数判定、素数生成法を用いる場合の要件についてもまとめる。

9.1 素数判定の確実性および素数生成の確実性と一様性

確実に素数が生成されることおよび一様性について、次の要件が必要となる。

- Miller-Rabin 法など確率的判定法を用いる場合は、誤判定確率パラメータ (繰り返し回数) を適切に設定する。
- 正しく実装されていることを検査する。特に Cohen-Lenstra 法など確定的判定法は、そのアルゴリズムの複雑さから、正しく実装することが一般には難しいと考えられるため、実装の正しさの検査方法自体が確立されてから使用することが望ましい。
- 擬似乱数生成法は適切なアルゴリズムを用いる。

Miller-Rabin 法等，確率的判定法を適用する際は，本稿の実装で適用したように，Miller-Rabin 法の繰り返し回数は，ビット長に応じて適切に設定されるような実装法を採用することは必須である．

また，生成された素数が何らかの特徴を持つことがわかっている場合，例えば素因数分解が容易に行える可能性がある．素数を一様にランダムに生成することが必要で，そのためには，擬似乱数を適切なアルゴリズムによって生成することが必要である．CRYPTREC において例示されている擬似乱数生成アルゴリズム [58] などを使用することが望ましい．

9.2 素数生成の実装性

実装性に関しては，速度とメモリのトレードオフが生じる．素数生成の実装法を選択する場合，

- 必要とされる速度
- 使用可能なメモリ量

の両方を考慮して決める必要がある．

素数生成は，暗号化や署名生成と比べて時間のかかる演算である．多くのユーザに短時間で暗号鍵を発行するような場面では，素数生成の高速性は重要な要件である．6 節において考察したように，高速に素数生成を行うための高速べき乗剰余演算など各種高速アルゴリズムを実装し，各アルゴリズムにおける，window 幅など設定パラメータも最適に選ぶ必要がある．

また，安全性上の理由から，IC カード上で鍵生成 (素数生成) を行う場合もある．IC カードは一般に実装されるメモリが少なく，RAM は数百バイト程度のこともある．したがって，6 節の実装における window 幅等の設定は，使用できるメモリに応じて選択しなければならない．

9.3 素数を暗号鍵として用いる場合の安全性条件

電子政府において使用されることが推奨される暗号のリスト案に記載されている公開鍵暗号は，表 16 に記述した通りである．これらの公開鍵暗号が安全性の根拠としている数論問題は，表 16 に記載した通り，

- 素因数分解問題
- 離散対数問題
- 楕円曲線離散対数問題

のいずれかである．

したがって，生成された素数を CRYPTREC 推奨暗号へ用いる場合の安全性を検証するとすれば，7 節で述べた，RSA 暗号の場合の安全性条件，DSA 署名の場合の安全性条

件，または楕円曲線暗号の場合の安全性条件のいずれかを暗号アルゴリズムに応じて検証すればよい．

また，9.1 節で述べたような確実に一様に素数を生成することが，安全な電子政府セキュリティシステムを構築する上で重要な要件である．

9.4 今回開発すべきソフトウェアの要求仕様

前節までで素数生成法に関する様々な考察を行った．これらの考察に基づき，今回開発する素数判定ソフトウェア，素数生成ソフトウェア，素数を暗号鍵として用いた場合の安全性判定ソフトウェア (パラメータ安全性判定ソフトウェア) の要求仕様について述べる．

9.4.1 素数判定ソフトウェア

素数判定ソフトウェアに実装する素数判定アルゴリズムとして，次のものが挙げられる．

- 確率的素数判定法
 - Miller-Rabin 法
 - Solovay-Strassen 法
- 確定的素数判定法
 - AKSL 法
 - AKS 予想法

前節までの考察で述べたように，Miller-Rabin 法は誤り確率や実装効率において優れた素数判定アルゴリズムであり，実装することは必須である．また，電子政府，その他暗号実用においては，現時点では Miller-Rabin 法が最も優れた素数判定アルゴリズムであると考えられる．

ただし，9.5 節において述べたように，実装された Miller-Rabin 法モジュールが正しく素数判定を行っているかどうかを検査することは必須である．前述のように，アルゴリズムが簡便で，より実装誤りが起きにくい Solovay-Strassen 法を実装しておくことも有効であると思われる．Miller-Rabin 法モジュールと Solovay-Strassen 法モジュールとの判定結果が一致するかどうかを検査することは，Miller-Rabin 法モジュールの実装の正しさ (バグが無いこと) を検証する 1 つの手段と考えられる．

AKSL 法 (3.7 節参照) は最近になって発明された確定的判定法で，多項式時間で確定的に素数判定が行える初めてのアルゴリズムという点で画期的である．現時点では，暗号において使用するような 512 ビット程度の大きさの素数を判定することは現実的な時間で行えないが，3.7 節でも述べたように，AKSL 法の理論上の改良は現在もなお活発に研究が行われている．したがって，新しい画期的なアルゴリズムである AKSL 法の実装性を把握しておくことは必要であると考えられる．また，[2] において述べられている予想が正しいと仮定した素数判定法である AKS 予想法も，3.8 節において述べたように，理論的に

も実装的にも改良が続けられている素数判定法である。したがって、AKS 予想法の実装性を把握しておくことは必要であると思われる。

9.4.2 素数生成ソフトウェア

素数生成ソフトウェアにおいては、前節 9.4.1 において挙げた素数判定法に基づく素数生成アルゴリズムを実装すればよい。素数生成ソフトウェアに実装すべき素数生成アルゴリズムとして、次のものが挙げられる。

- 確率的素数判定法に基づく素数生成法
 - Miller-Rabin 法に基づく Random choice による方法
 - Miller-Rabin 法に基づく Incremental search による方法
 - Solovay-Strassen 法に基づく Random choice による方法
 - Solovay-Strassen 法に基づく Incremental search による方法
- 確定的素数判定法に基づく素数生成法
 - AKSL 法に基づく Random choice による方法
 - AKSL 法に基づく Incremental search による方法
 - AKS 予想法に基づく Random choice による方法
 - AKS 予想法に基づく Incremental search による方法

素数生成法には Random choice による方法と Incremental search による方法があるが、実際に使用する時は、素数の一様性や素数生成速度 (6 節参照) から選択すればよい。また、素数生成においては乱数を生成することが必要であるが、CRYPTREC において例示されているような適切な擬似乱数生成アルゴリズム [58] を用いることが必要である。なお、AKS 予想法は [2] において述べられている予想が正しいと仮定した素数判定法である。

9.4.3 パラメータ安全性判定ソフトウェア

7 節で述べたように、公開鍵暗号として代表的なものは RSA 暗号と DSA 署名と楕円曲線暗号である。また、CRYPTREC において挙げられた電子政府で利用されることが推奨される暗号のリスト案に記載されている公開鍵暗号は、RSA 暗号、DSA 署名、楕円曲線暗号のいずれかに分類されるか、または素数の要件としてはこの 3 つの暗号に対する素数の要件のいずれかと同じである。したがって、開発すべきパラメータ安全性ソフトウェアに必要とされる仕様としては、この 3 つの暗号に対する素数の安全性を検証できることである。楕円曲線暗号へ用いる素数 p の要件は、 p の素数性とビット長であるので、後述する RSA 暗号のパラメータ安全性判定項目と同じである。したがって、開発すべきソフトウェアは、

- RSA 暗号の秘密鍵として用いる場合の素数の要件

- DSA 署名の公開鍵として用いる場合の素数の要件

を検証できればよいと考えられる．それぞれの要件は7節で述べた通りである．

したがって，パラメータ安全性判定ソフトウェアに要求される仕様は，RSA 暗号に素数を用いる場合の安全性判定機能と DSA 署名に素数を用いる場合の安全性判定機能とを持つことであり，それぞれの安全性判定項目は，7節で述べた通り次のようになる．

RSA 暗号のパラメータ安全性判定項目

- p と q の素数性判定
- p と q のビット長判定

DSA 署名のパラメータ安全性判定項目

- p と q の素数性判定
- p は 1024 ビットであることの判定
- q は 160 ビットであることの判定
- $p - 1$ は q で割り切れることの判定

9.5 CMVP への利用に向けて

Cryptographic Module Validation Program(CMVP) は，1995 年に NIST(米国) および CSE(カナダ) により創設された暗号モジュールの評価制度である．CMVP においては，開発された暗号モジュールが正しく動作するか，FIPS 140-2 で要求された基準を満足するかなどの検証を行う．

現在 FIPS 140-2 では，素数判定に対する要求基準は定められていない．しかしながら，素数判定/生成モジュールの作成においては，要求基準が定められていないからといって検証が不要であるということの意味しているわけではない．わが国で同様の制度を創設する際には独自に検証項目を設定する必要があると考えられている．

Miller-Rabin 法は効率の良い素数判定法であり，CMVP 用にも適していると思われるが，実際に実装された Miller-Rabin 法モジュールが確実に動作しているかどうかを検証する必要がある．検証手段の 1 つとして，別な素数判定法，例えば Miller-Rabin 法よりもアルゴリズムが単純でバグが混入しにくい Solovay-Strassen 法 (2.3 節) を用いて，両者の素数判定結果が一致することを確認することが考えられる．ただし，素数判定の誤り確率が十分小さくなるように基底の数 (繰り返し回数) を選択した上で両者の判定結果を比較しなければならない．

また，素数生成モジュールが恣意的に素因数分解が容易な素数を生成していないか，あるいは与えられた暗号鍵である素数が素因数分解が容易なものになっていないかを検証する手法の確立も必要である．前節までに述べたように，“適切な手法”で生成された大きな

素数であれば素因数分解が困難である。したがって、適切な手法で生成された素数であるかどうかを検証することが CMVP においては必要と考えられる。

RSA 暗号においてはこのような手順は規格化されていない。DSA 署名の場合は FIPS 186-2 において素数生成のアルゴリズムが規定されており、FIPS 186-2 に記述された手順で生成された素数であることを検証する手段が記述されている。DSA 署名の素数生成では、一方向性関数 (SHA-1) を使用しており、SHA-1 へは乱数が入力される。乱数値を保存しておくことにより、素数生成手順を再現し、恣意的に生成された素数でないことを確認できる。楕円曲線暗号においても素数を含めた曲線パラメータが適切な手順で生成されたことを後から検証できるような手順を規格化することが必要と考えられる。

10 まとめ

本調査報告書では、安全で信頼できる電子政府セキュリティシステムを構築するために、素数に関連する事項の必要条件について調査を実施した。

理論的調査および実際の実装実験により得られた結論をまとめると次のようになる。

- Miller-Rabin 法は、誤り確率、実装効率ともに優れており、現時点では電子政府において使用する素数判定アルゴリズムとして最も相応しい。ただし Miller-Rabin 法を使用する際は、ビット長に応じた適切なセキュリティパラメータ (繰り返し回数) を設定することが必要である。
- 実装プラットフォームを考慮して、速度性能および必要メモリ量が最適になるような演算アルゴリズムを適用することが必要である。
- 品質の高い擬似乱数を生成する擬似乱数生成アルゴリズムを使用することが必要である。

また、AKSL 法や AKS 予想法など、新しく画期的な確定的素数判定法については、今後も理論的観点および実装的観点での研究の進展が予想される。したがって、AKSL 法および AKS 予想法は今後も動向を注意深く追っていくことが必要と思われる。

安全で信頼できるセキュリティシステムを構築するためには、素数に関連する事項以外にも、数多くの必要条件がある。セキュリティシステムの安全性の根幹を支える要素技術の一つは、暗号アルゴリズムであり、その暗号アルゴリズムおよび暗号鍵を強固なものとするために、安全で信頼できる素数、素数判定法、および素数生成法を使用することは重要な要件である。

参考文献

- [1] L. M. Adleman, C. Pomerance and R. S. Rumely, “On distinguishing prime numbers from composite numbers”, *Ann. Math.* 117 (1983), 173-206.
- [2] M. Agrawal, N. Kayal, N. Saxena, “Primes is in P”, preprint, available at <http://www.cse.iitk.ac.in/primality.pdf>.
- [3] W. R. Alford, A. Granville and C. Pomerance, “There are infinitely many Carmichael numbers”, *Ann. Math.* 140 (1994), 703-722.
- [4] ANSI X 9.80, “*American National Standard for Financial Services - Prime Number Generation, Primality Testing and Primality Certificates*”, American National Standard Institute, 2001.
- [5] F. Arnault, “The Rabin-Monier theorem for Lucas pseudoprimes”, *Math. Comp.* 66 (1997), 861-881.
- [6] A. O. L. Atkin and F. Morain, “Elliptic curves and primality proving”, *Math. Comp.* 61 (1993), 29-68.
- [7] E. Bach, “Explicit bounds for primality testing and related problems”, *Math. Comp.* 35 (1990), 355-380.
- [8] R. Bailie and S. S. Wagstaff, Jr., “Lucas pseudoprimes”, *Math. Comp.* 35 (1980), 1391-1417.
- [9] P. Beauchemin, G. Brassard, C. Crepeau, C. Goutier and C. Pomerance, “The generation of random numbers that are probably prime”, *Journal of Cryptology* 1 (1988), 53-64.
- [10] D. J. Bernstein, “An exposition of the Agrawal-Kayal-Saxena primality-proving theorem”, preprint.
- [11] P. Berrizbeitia, “Sharpening *Primes is in P* for a large family of numbers”, preprint, available at <http://xxx.yukawa.kyoto-u.ac.jp/abs/math.NT/0211334>.
- [12] J. Brandt, I. Damgard, “On generation of probable primes by incremental search”, *CRYPTO* 92 (1992), LNCS 740, 358-370.
- [13] J. Brandt, I. Damgard and P. Landrock, “Speeding up prime number generation”, *ASIACRYPT* 91 (1991), LNCS 739, 440-449.
- [14] J. Brillhart, “Fermat’s factoring method and its variants,” *Congressus Numerantium*, vol. 32, (1981), 29-48.

- [15] J. Brillhart, D. H. Lehmer, J. Selfridge, B. Tuckerman and S. S. Wagstaff, Jr., “Factorizations of $bn \pm 1$ for $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers”, Contemporary Mathematics, vol. 22, American Mathematical Society, Second Edition, 1988.
- [16] R. J. Burthe, Jr., “Further investigations with the strong probable prime test”, *Math. Comp.* 65 (1996), 373-381.
- [17] H. Cohen and A. K. Lenstra, “Implementation of a new primality test”, *Math. Comp.* 48 (1987), 103-121.
- [18] H. Cohen and H. W. Lenstra, Jr., “Primality testing and Jacobi sums”, *Math. Comp.* 42 (1984), 297-330.
- [19] R. Crandall and C. Pomerance, “*Prime Numbers: A Computational Perspective*”, Springer, 2001.
- [20] “*CRYPTREC Report 2001 (暗号技術評価報告書)*”, 情報処理振興事業協会 (IPA) 通信放送機構 (TAO).
- [21] IPA, TAO, “CRYPTREC ホームページ,” <http://www.ipa.go.jp/security/enc/CRYPTREC/>
- [22] I. Damgard, G. S. Frandsen, “An extended quadratic Frobenius primality test with average case error estimates”, BRICS Report, RS-01-45, Univ. of Aarhus, Denmark, 2001.
- [23] I. Damgard, P. Landrock and C. Pomerance, “Average case error estimates for the strong probable prime test”, *Math. Comp.* 61 (1993), 177-194.
- [24] FIPS 186-2, “*Digital Signature Standard (DSS)*”, Federal Information Processing Standards Publication 186-2, U.S. Department of Commerce/ N.I.S.T., January 27 2000.
- [25] GNU MP home page, <http://www.swox.com/gmp/>.
- [26] S. Goldwasser and J. Kilian, “Almost all primes can be quickly certified”, *Proc. 18th STOC.* (1986), 316-329.
- [27] D. M. Gordon, “Discrete logarithm in $GF(p)$ using the number field sieve,” *SIAM Journal on Discrete Math.*, vol. 6 (1993), 124–138.
- [28] D. M. Gordon, “A survey of fast exponentiation methods,” *Journal of Algorithms*, vol. 27, (1998), 129–146.
- [29] J. Grantham, “A probable prime test with high confidence”, *Journal of Number Theory* 72 (1998), 32-47.

- [30] J. Grantham, “Frobenius pseudoprimes”, *Math. Comp.* 70 (2001), 873-891.
- [31] J. Grantham, “There are infinitely many Perrin pseudoprimes”, preprint, available at <http://www.pseudoprime.com/jgpapers.html>.
- [32] N. Kayal, N. Saxena, “Towards a deterministic polynomial-time primality test”, Technical Report, IIT Kanpur, 2002, available at <http://www.cse.iitk.ac.in/research/btp2001/primality.html>.
- [33] S. -H. Kim and C. Pomerance, “The probability that a random probable prime is composite”, *Math. Comp.* 61 (1989), 721-741.
- [34] D. E. Knuth and L. T. Pardo, “Analysis of a simple factorization algorithm”, *Theoret. Comput. Sci.*, 3 (1976), 321-348.
- [35] N. Koblitz, “*A Course in Number Theory and Cryptography*”, GTM 114, Springer Verlag, Second Edition, 1994.
- [36] Y. Kida, “Cohen-Lenstra method UBASIC program”, available at <http://www.rkmath.rikkyo.ac.jp/~kida/ubasic.htm>
- [37] A. K. Lenstra and H. W. Lenstra, Jr, “Algorithms in number theory”, J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Elsevier Science Publishers, 1990, 674-715.
- [38] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, “The number field sieve,” *Proc. 22nd STOC*, (1990), 564–572.
- [39] H. W. Lenstra, Jr, “Primality testing algorithms (after Adleman, Rumely and Williams), *Séminaire Bourbaki* 33 (1980/1981), no. 576, LNM Vol. 901, Springer. 1981, 243-257.
- [40] H. W. Lenstra, Jr., “Factoring integers with elliptic curves,” *Ann. Math.*, vol. 126, (1987), 649–673.
- [41] U. Maurer, “Some number-theoretic conjectures and their relation to the generation of cryptographic primes”, C. Mitchell. Editor, *Cryptography and Coding II*, volume 33 of Institute of Mathematics & Its Applications (IMA), 173-191, Clarendon Press, 1992.
- [42] U. Maurer, “Fast generation of prime numbers and secure public-key cryptographic parameters”, *Journal of Cryptology* 8 (1995), 123-155.
- [43] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, “*Handbook of Applied Cryptography*”, CRC press, 1997.

- [44] P. Mihăilescu, “Cyclotomy of rings and primality testing”, dissertation 12278, ETH Zürich, 1997 available at <http://www.inf.ethz.ch/~mihailles>.
- [45] P. Mihăilescu, “Cyclotomy primality proving - recent developments”, *ANTS III*, LNCS Vol. 1423, Springer. 1998, 95-110.
- [46] G. L. Miller, “Riemann’s hypothesis and tests for primality”, *Journal of Computer and System Sciences*, 13 (1976), 300-317.
- [47] L. Monier, “Evaluation and comparison of two efficient probabilistic primality testing algorithms”, *Theoret. Comput. Sci.*, 12 (1980), 97-108.
- [48] F. Morain, “Primality proving using elliptic curves: an update”, *ANTS III*, LNCS Vol. 1423, Springer. 1998, 111-127.
- [49] F. Morain, “E CPP package”, available at <http://ultralix.polytechnique.fr/Labo/Francois.Morain/Prgms/ecpp.english.html>.
- [50] F. Morain, “Primalite theorique et primalite pratique ou AKS vs. E CPP”, available at <http://www.lix.polytechnique.fr/Labo/Francois.Morain/aks-f.pdf>
- [51] S. Müller, “A probable prime test with very high confidence for $n \equiv 1 \pmod{4}$ ”, *ASIACRYPT 2001* (2001), LNCS 2248, 87-106.
- [52] S. C. Pohlig, M. E. Hellman, “An improved algorithm for computing logarithm over $GF(p)$ and its Cryptographic Significance,” *IEEE Trans. Information Theory*, IT-24, No. 1, (1978), 106–110.
- [53] J. M. Pollard, “Theorems on factorization and primality testing,” *Proc. Cambr. Philos. Soc*, vol. 76, 1974, 521–528.
- [54] C. Pomerance, “Are there counter-examples to the Baillie-PSW primality test”, Letter to A. Lenstra, available at <http://www.pseudoprime.com/dopo.ps>.
- [55] C. Pomerance, “Primality testing: variations on a theme of Lucas”, to appear in the proceedings of an MSRI workshop, J. Buhler and P. Stevenhagen, eds. available at <http://cm.bell-labs.com/cm/ms/who/carlp/pub.html>.
- [56] C. Pomerance, J. L. Selfridge and S. S. Wagstaff, Jr., “The pseudoprimes to $25 \cdot 10^9$ ”, *Math. Comp.* 35 (1980), 1003-1026.
- [57] V. R. Pratt, “Every prime has a succinct certificate”, *SIAM J. Comput.*, 4 (1975), 214-220.
- [58] 総務省, 経済省, “「電子政府」における調達のための推奨すべき暗号のリスト案に対する意見募集,” 2002.11.28, http://www.soumu.go.jp/s-news/2002/021128_5.html.

- [59] M. O. Rabin, “Probabilistic algorithm for testing primality”, *Journal of Number Theory* 12 (1980), 128-138.
- [60] P. Ribenboim, “*The Book of Prime Number Records*”, Springer Verlag, Second Edition, 1989.
- [61] H. Riesel, “*Prime Numbers and Computer Methods for Factorization*”, Birkhauser, Second Edition, 1994.
- [62] R. L. Rivest, R. D. Silverman, “Are ‘strong’ primes needed for RSA ?,” Cryptology ePrint Archive, Report 2001/007, (2001), available at <http://eprint.iacr.org>.
- [63] R. Schoof, “Elliptic curves over finite fields and the computation of square roots mod p ”, *Math. Comp.* **44**, 1985, 483-494.
- [64] J. Shawe-Taylor, “Generating strong primes”, *Electronics Letters* 22 (July 31, 1986), 875-877.
- [65] R. Solovay and V. Strassen, “A fast Monte-Carlo test for primality”, *SIAM J. Comput.*, 6 (1977), 84-85, Erratum in 7 (1978), 118.
- [66] J. F. Voloch, “Improvements to AKS”, preprint, available at <http://www.ma.utexas.edu/users/voloch/preprint.html>.
- [67] L. C. Washington, “*Introduction to Cyclotomic Fields*”, GTM 83, Springer Verlag, Second Edition, 1996.
- [68] H. C. Williams, “A $p + 1$ method of factoring,” *Math. Comp.*, vol. 39, No. 159, (1982), 225–234.
- [69] H. C. Williams, “*Edouard Lucas and Primality Testing*”, Can. Math. Soc. series of monographs and advanced text, Wiley-Interscience, 1998.