RSA 公開鍵における情報埋め込みサイズの上限に関する考察

北原　基貴 †　　安田　貴徳 ‡　　西出　隆志 †　　櫻井　幸一 †

† 九州大学大学院システム情報科学府情報学専攻
819-0395 福岡市西区元岡 744 番地
kitahara@itslab.inf.kyushu-u.ac.jp

‡ 九州先端科学技術研究所
814-0001 福岡市早良区百道浜 2-1-22

あらまし RSA 暗号において，公開鍵 N の中に効率的に情報を埋め込む手法が Lenstra により提案された．それ以来，Lenstra の提案した埋め込みアルゴリズムを用いて鍵供託を行ったり，公開鍵をより使いやすいものに変える手法が提案されてきた．しかし，Lenstra は埋め込み情報量の上限について公開鍵長の半分程度まで効率的に埋め込むことができると主張していたが，具体的なビット数に関しては言及していなかった．本研究では Lenstra アルゴリズムを用いた場合の効率的に埋め込みのできるビット長の限界を見積り，また埋め込むビット長と公開鍵 N からアルゴリズムの成功確率を見積もる式を理論解析と実験により算出した．

Limit of Embeddable Information Size for RSA public key

Motoki Kitahara†　　Takanori Yasuda‡　　Takashi Nishide†　　Kouichi Sakurai†

†Kyushu University
744 Motooka, Nishi-ku, Fukuoka-ward, Fukuoka 819-0395, JAPAN
kitahara@itslab.inf.kyushu-u.ac.jp

‡Institute of Systems, Information Technologies and Nanotechnologies (ISIT)
2-1-22 Momochihama Sawara-ku, Fukuoka-ward, Fukuoka 814-0001, JAPAN

Abstract In RSA encryption, Lenstra proposed the method that information can be embed in the public key N efficiently. Since then, many methods have proposed, such as additional key escrow function and a public key becoming visible. Lenstra make an assertion that embeddable size is up to half of the length of a public key, but he did not mention the strict length. In this paper, we examined Lenstra algorithm both in theoretical analysis and in implementation analysis, and calculated the limit of the length of embeddable information efficiently.

1　序章

1.1　背景

RSA 暗号は公開鍵暗号の代表として現在最も広まっている暗号の一つである．RSA 暗号の特徴の一つとして，公開鍵 N に情報を埋め込むことができるという点がある．この情報は長さ以外に制限がないため，どんな情報でも埋め込むことができる．Vanstone と Zuccherato が最初に RSA 公開鍵に情報を埋め込むアルゴリズムを提案した [2]．このアルゴリズムでは素因数の桁数まで情報を埋め込んでしまうと，埋め込む情報の素因数が必要となり，速度の面で効率の悪い手法だった．埋め込む情報を少なくすることで，情報量とのトレードオフにはなるが，速度を効率化することもできた．その後 1998 年に Lenstra が速度の面でも効率的であり，素因数の桁数まで情報を埋め込むことのできる手法

を提案した [1]．このアルゴリズムは埋め込む情報量が素因数の桁数近くまでであればオリジナルの RSA における鍵生成アルゴリズムと同程度に高速に実行できる手法であった．

1.2 研究の動機について

埋め込みアルゴリズムを用いた場合，どんな情報でも埋め込むことができるため，応用を考えた場合 1 ビットでも長い方が望ましい．また，コンピュータの進歩により公開鍵に必要なビット数は増加しており，NIST でも 2011 年からは 2048 ビットの鍵を用いることを推奨している．このことは，埋め込む情報も多くすることができることを意味している．今後ますます多くの応用例が提案されていくと想定される．実用を考えた場合，最も必要になるのは処理速度であり，現状速いとは言えない RSA 暗号よりも時間のかかるものは使いづらく，RSA と同じオーダで鍵生成を行うことができる Lenstra アルゴリズムが埋め込みアルゴリズムを用いる場合では有用である．これらの点により，Lenstra アルゴリズムに対して厳密な評価を行った．

1.3 関連研究

Lenstra が実用的な埋め込みアルゴリズムを提案して以来，多くの応用例が提案されてきた．Lenstra アルゴリズムはオリジナルの RSA 暗号と同程度の効率で鍵生成を行うことができ，作られた公開鍵も RSA 暗号と互換性があるため，既存の手法を拡張して何らかの機能を付加するといった手法が多く提案されている．

埋め込みアルゴリズムの最も有益な利用法の一つとして，管理者に自身の秘密鍵を伝える鍵供託が Joye により提案されている [3]．秘密鍵情報を管理者と共有した秘密鍵で暗号化し，利用者自身の公開鍵に埋め込むことにより，秘密鍵を持つ管理者のみが復号でき，利用者の秘密鍵を得ることができるという手法である．これにより管理者は利用者全員の秘密鍵を得ることができ，管理が容易になる．もし我々の秘密鍵を公開鍵に埋め込もうとした場合，RSA 暗号では秘密鍵情報である素因数の桁数は公開鍵の半分となる．後に述べるが，Lenstra アルゴリズムにおいて公開鍵の桁数の半分まで情報を埋め込んでしまうと問題が生じる．そのため，この論文では秘密鍵情報を短くして埋め込んでいる．また，同じ論文にて，公開鍵を短くする方法も提案されている．この手法は種となる小さな値と一方向性ハッシュを用い，この種に対してハッシュをとった値を公開鍵の埋め込み情報として使うものである．受信者は種に対するハッシュ値を埋め込み情報として公開鍵を作成し，種とハッシュそのもの，そして埋め込み部以外の値を保存する．送信者はこれらの値を取ってきて，種となる値とハッシュから埋め込み情報を得，保存されている埋め込み部以外の値と組み合わせることにより，公開鍵を得ることができる．この手法では，ハッシュの出力値に対して，種となる値とハッシュそのものの値の合計との差分が公開鍵として削減できる量となる．

別の例として，公開鍵を写真に置き換え，ユーザビリティを高める手法が提案されている [4]．この手法では，利用者は自分の用意した写真とほぼ同じ見た目の写真を公開鍵として用いることができる．予め用意した写真のピクセルの濃さ情報のうち，大きな影響を与えている上位ビットは変えず，埋め込み情報として公開鍵を作成することで，作成した公開鍵を写真として表示した場合に元の画像と似たものにできるというもの．結果としても利用者から見ると判別不可能であり，誰の公開鍵かをひと目で判断できるため，利用や管理が容易になるという利点がある．

また，我々は証明書を公開鍵に埋め込む手法の提案を行っている [5]．この手法では埋め込まれた証明書を参照することで，ID ベース暗号と同様に公開鍵だけで公開鍵の確認ができるため，帯域を節約することが可能となっている．

1.3.1 Lenstra のアルゴリズム

Lenstra は RSA における公開鍵に，効率的に情報を埋め込む手法を提案した [1]．この論文では，Lenstra は公開鍵の上位ビットに埋め込む手法と，下位ビットに埋め込む手法の両方を提

案していた．このセクションでは，上位ビットに埋め込む手法に関して議論を行う．

Lenstra の提案したアルゴリズム
埋め込み情報を I とし，$L = |N| - |s|$ とする．d^x は d 進数固定基数 x 桁の数を意味し，$|N|$ は N のビット長を意味するとする．

1. $d = 2$ とし，$n' = I * d^L$ を求める．この時，s は K の長さとする．
2. L 以下の桁数である素数 p をランダムに選ぶ
3. p との積が n' と最も近くなる整数 q を求める
4. $q = q' + m$ を満たし，q が素数となる数を求める．この時，m は最小の数を取る．
5. $n = pq$ を計算し，上位ビットが I と等しければ n, p, q を返す．等しくなければ 2 に戻る．

このアルゴリズムは，入力として公開鍵 N の桁数と埋め込み情報の 2 つを必要とし，p, q は埋め込み情報によって決まる．しかし，実用を考えた場合，公開鍵 N の素因数である p, q は同じ長さになったほうがよい．よって，以下のアルゴリズムを用いる．こちらでは公開鍵長と埋め込み情報から，適切なビット長を持った p, q, N が生成される．
今回用いたアルゴリズム

1. $N' = I \parallel 00...0$ を計算する．この時，N' の桁数が N と等しくなるように生成する．
2. ランダムに素数 p を計算する．この時，p の桁数が N の半分となるように生成する．
3. $Q' = \lceil N'/p \rceil$ を計算する
4. $q = Q' + V$ が素数となるような最小の V を求める．
5. $N = pq$ を計算する．ここで出た N に関して，上位ビットが I と等しくなっていれば N を公開鍵として出力する．等しくなければ 2 に戻る．

最終的に，埋め込み情報 I を含んだ N が RSA 暗号における公開鍵として生成される．

このアルゴリズムの問題点として，埋め込み情報は公開鍵の素因数である q より小さくなければならなということがあげられる．また，効率を考えた場合は更にある程度の冗長ビット長が必要となる．

このアルゴリズムは，同じ埋め込み情報を用いて実行したとしても，(2) で選ぶ p の素数の値を変えることにより，別の公開鍵を作成することができる．よってこの p を選ぶ範囲が十分にあれば，同じ公開鍵を作成してしまうこともなく，安全に作成できるといえる．また，N と p を適切に選ぶことにより，q の長さも p とほぼ同じにすることができる．このアルゴリズムでは，N の半分の長さの p を選んだ場合，半分程度の確率で q の長さは p と同じになり，半分程度の確率で q の方が 1 ビット大きいものとなる．

1.4 解決すべき問題について

Lenstra アルゴリズムはそれまでの方式と比較すると高速であり，かつ多くの情報を埋め込むことができる．このアルゴリズムを見て分かるとおり，ステップ 5 で得られる N の上位ビットが I と一致しない限り，何度も p を取り換え，ステップ 2〜5 を繰り返されることになる．すなわち，ランダムに取った一つの p に対し，N の上位ビットが I と一致する確率が Lenstra アルゴリズムの効率性に直接かかわってくる．そしてこの確率は埋め込み情報の桁数が大きくなるに連れ，小さくなることが経験的に分かっている．そこでこの確率と埋め込み情報の桁数に関する関係を明らかにすることが Lenstra アルゴリズムの効率的設計のために必要となる．

1.5 我々の貢献について

我々は Lenstra アルゴリズムにおいて，ランダムに選ばれた 1 つの素数 p から作られた N が I と一致する上位ビットを持つ確率を算出した．そしてこの確率を用いて，効率的に公開鍵を出すことのできる限界となる埋め込み情報のサイズを算出した．我々はこの確率の算出のために素数定理を用いた理論的解析と素数定理に含ま

れる誤差の実験的見積りを行った。その結果，この確率は N のビット長と埋め込み情報の桁数を用いた式で表せることが分かった．これらにより，利用者は公開鍵長からどの程度の情報まで埋め込むか知ることができ，また埋め込み情報のサイズから埋め込みを効率的に行うことのできる成功確率も知ることが可能となった．

Lenstra アルゴリズムを効率的に実行するための成功確率

N が 2048 ビットと仮定すると，埋め込み情報の長さが.1013 ビットのとき 87% ，埋め込み情報の長さが 1012 ビットのとき，98% で成功する．

1.6 先行研究との比較

Lenstra は自身の論文にて，処理速度における最適の場合と最悪の場合のオーダを算出していた．しかし，いつ最悪の場合が起きるのか，厳密には言及しておらず，埋め込み情報と一つの素因数が近い場合，とだけ言っていた．本論文では，我々は RSA における公開鍵の２つの素因数が同程度の長さの場合に限定し，Lenstra アルゴリズムにおいて現れる繰り返し操作の回数の期待値を見積もった．その結果，Lenstra アルゴリズムの効率性をより正確に見積もることが可能になった．

2 準備

2.1 表記法

本論文では，以下の表記法を用いる．

文字の結合 $a \parallel b$ は a, b ２つの文字の結合を表す．

文字列の長さ $|a|$ は a の桁数を表す．

切り上げ $\lceil a \rceil$ は a に対して，小数点以下を切り上げた値を表す．

平均 $\bar{a}$ は a の平均を表す．

2.1.1 環境設定

本論文では以下の前提を仮定する．

- 埋め込み情報としてランダムな値を用いる．
- 素数生成における擬陽性は 2^{-100} を超えないものとし，偽陰性はないとする．

3 解析

3.1 理論解析

1.4 で Lenstra アルゴリズムの効率性はランダムに選ばれた p から作られた N が I と一致する上位ビットを持つ確率と関係することを述べた．この確率は N の上位ビットが I と一致するような q の存在する確率と言い換えることができる．そこでこの節では，このような q の存在確率を素数定理を用いて理論的に見積もる．素数定理は近似公式なので，この節では誤差を含んだ形でしか述べることはできない．しかし，後の節では，この誤差を実験的に見積もり，より正確な確率を与える．

まず，このような q の存在確率に関して次のように解析できる．Lenstra アルゴリズムにおいて，R を $p*(Q'+R)$ の上位ビットが I と一致する最大の自然数とすると，素数 q は（存在するならば）$Q', Q'+1, ..., Q'+R$ の中になければならないことが分かる．この区間の中の整数が素数となる確率が整数の取り方によらず一定だと仮定し、その確率を X とすると上のような条件を満たす q の存在確率は

$$1-(1-X)^R$$

と書くことができる．

以下で X と R を理論的に見積もる．

3.2 X に関する理論的見積り

上の解析ではある区間で整数が素数になる確率が一定で，その確率も分かっていることを仮定した．この確率は以下の素数定理を使うと分かる．

定理 3.1 (素数定理（非公式）) 数値 x が与えられた場合，x が素数となる確率は $1/\log_e x$ となる．

この素数定理は通常の素数定理ではなく，素数のランダム分布を仮定した非公式なものである．そのため，この仮定の正しさを証明する必要がある．今回は実験にてこの差があるかを求める．

3.2.1 X の見積り

先ほどの q の存在確率の見積もりでは $\log_e q$ が現れた．これを $|q|$ を用いた式で書き直しておいた方が使い勝手が良い．$\log_e q$ は $|q|$ で書くことが可能で，実際，$1/\log_e q$ は $C_1 = \log_e q/|q|$ とおくと，$(\log_e q)^{-1} = (C_1|q|)^{-1}, \log_e 2 \leq C_1 < \log_e 2\,|q+1|\,/\,|q|$ と表せる．

3.3 R に関する理論的見積り

秘密鍵となる素数のビット長（どちらの素数も同じ長さと仮定している）から埋め込みたい情報のビット長の差を冗長ビット長と呼び、 r でその値を表すことにする．試行できる回数 R は，2^r とほぼ等しい

この近似に関して，更に検証を行う．

3.3.1 R の見積もり

N_0 を繰り上がりが起こった時の N の値，$2^{l-1} \leq p < 2^l$，埋め込む情報部以外を乱数ではなくすべて 0 と仮定すると，$N_0 - N' = 2^{l+r}$ と表すことができる．よって $2^r < (N_0 - N')/p \leq 2^{r+1}$ が成り立つ．$C_2 = r+1-\log_2((N_0 - N')/p)$ とおくと，$(N_0 - N')/p = 2^{r+1-C_2}$ と書ける．これらの値より，試行回数 $(N_0 - N')/p$ は $C_2, 2^r \leq C_2 < 2^{r+1}$ と表せる．

3.4 理論解析により分かったこと

以上の理論解析により次が分かった．

定理 3.2 *Lenstra* アルゴリズムにおいて，1 つの p を固定したとき，N の上位ビットが I と一致するような q が存在する確率は次のように表される．

$$1 - (1 - (C_1 * |q|)^{-1})^{2^{r+1}-C_2}$$

但し，C_1 は $\log_e 2 \leq C_1 < ((|q|+1)/|q|)\log_e 2$ を満たす実数であり，C_2 は $2^r \leq C_2 < 2^{r+1}$ を満たす実数である．

3.5 実装

3.5.1 概要

理論解析では非公式な解釈を用いているため，素数定理を用いた q の存在確率の見積りが正しいかどうか検証する必要がある．ここでは，Lenstra アルゴリズムを実行し，ステップ (4) にて冗長ビット長を変えた場合に得られる素数をすべてカウントする．この行為を何度も繰り返した時の平均値と，理論解析で得られた値との差を算出することで，理論解析で用いた素数定理が誤差なく使用できるかの検証を行う．

さらなる解析として，我々が Lenstra アルゴリズムを使用することを考えた場合，検証に用いる素数そのものではなく，公開鍵長や埋め込み情報のみを用いて成功確率を出したい．このため，実験では冗長ビット長と公開鍵長を変えつつ実験を行い，$\log_e q$ を $|q|$ のみを用いて表す．

3.5.2 目的

今回用いている素数定理では，非公式な仮定を置いている．この値には誤差が含まれており，実用上問題がないのか検証を行う必要があるといえる．よって実際に Lenstra アルゴリズムを動かした場合に出る値と，素数定理を用いて算出した理論的な数値との間にどの程度誤差があるのか算出することで誤差を見積もる．

また，理論解析では C_1, C_2 という値を用いて Lenstra アルゴリズムのおおよその成功確率を見積もった．実験では実際に Lenstra アルゴリズムを動かした場合の値を算出することができるため，この値と理論値との比較を行うことで，C_1，C_2 の正確な値を算出する．

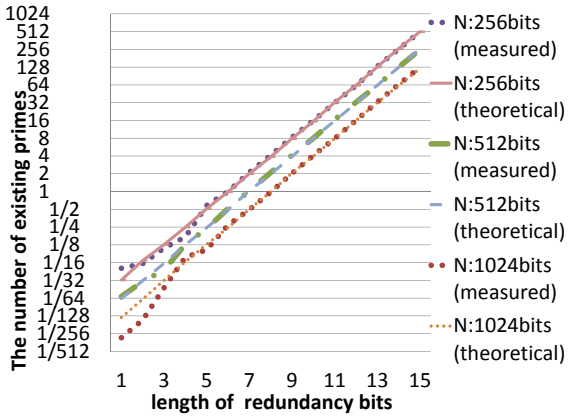

図 1: 存在する素数の個数

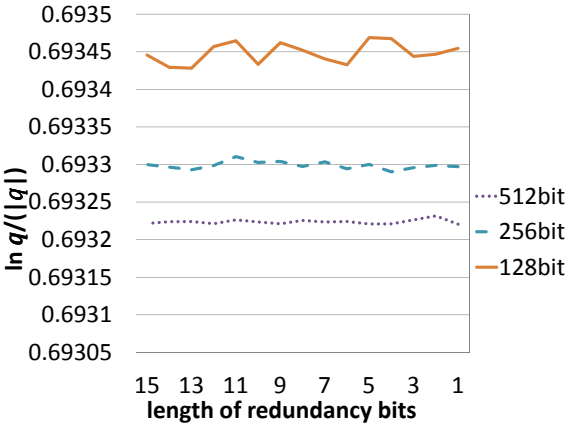

図 2: $\log_e q/q$

3.5.3 素数定理の検証

素数定理の非公式な仮定を用いており，この値が実用に足る精度を持っているかを調べる．Lenstra アルゴリズムの成功確率をそのまま用いた場合，冗長ビット長を増やしていくに従い，確率がほぼ 1 に漸近するため，値が正確であるかの検証が難しいという問題がある．よって，より精度を保つことのできる値で検証を行う．

今回はこの値として，Lenstra アルゴリズムのステップ (4) にて出力される素数の個数を検証に用いる．冗長ビット長が十分に大きい場合においても，素数である確率が一定であると仮定すると，生成される素数の個数は冗長ビット長に応じて増えていくと想定される．この値を理論解析で求めると，探索できる範囲と素数である確率の積と等しくなるため，$((N_0-N')/p)(\log_e q)^{-1}$ となる．この値と Lenstra アルゴリズムを実行した時の値を比較する．以下の図がその結果となる．

図 1 より，理論解析で求めた値とアルゴリズムを用いての実効値がほぼ等しいとわかる．冗長ビット長が少ない場合において，多少誤差が出ている場合があるが，こちらは Lenstra アルゴリズムを用いた場合に，ほぼ素数が見つからなかったため，施行回数が少なくなってしまったためである．Lenstra アルゴリズムにおいて冗長ビット長が少なくなると処理時間は $\mathcal{O}(pq)$ に漸近してしまうことが原因にあげられる．一つの素数を見つけるのにこのオーダがかかってしまい，統計的に誤差を無視できる程度の素数を集めることができなかった．更に試行回数を増やすことで精度を高めることができる．

しかし十分に Lenstra アルゴリズムを用いた場合の実験値は $((N_0-N')/p)(\log_e q)^{-1}$ と等しく，$(\log_e q)^{-1}$ が q が素数である確率を表しているといえる．

3.5.4 素数となる可能性

素数定理より，q という値を与えられた際にその q が素数である確率は $(\log_e q)^{-1}$ となり，理論解析より，$(\log_e q)^{-1} = (C_1 |q|)^{-1}$ であるため，C_1 を見積もる．

Lenstra のアルゴリズムを用いた際の素数となった q の値について，$|q| \le \log_2 q < |q+1|$ より，$1 \le \log_2 q/|q| < |q+1|/|q|$ である．ここで，$\log_2 q = \log_e q/\log_e 2$ だから，$\log_e 2 \le \log_e q/|q| < \log_e 2\,|q+1|/|q|$ となる．$C_1 = \log_e q/|q|$ とおくと，$(\log_e q)^{-1} = (C_1 |q|)^{-1}$ と q のビット長と定数で表すことができるため，C_1 の値を見積もる．

$\log_e q/|q|$ を縦軸に，冗長ビット長を横軸にとったものが以下のグラフである．

図 2 より，冗長ビット長を増やしたとしても，$\log_e q/|q|$ の値は変わらない．また，q のビット長を増やすにしたがって $\log_e q/|q|$ の値は減少している．暗号で用いることを考えた場合，RSA の公開鍵長には最低でも $N = 1024$ を使用するため，この値は 0.694 で抑えることができるとい

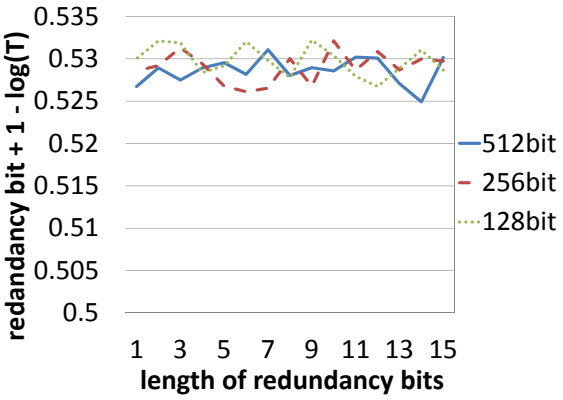

図 3: $r+1-\log_2 T$

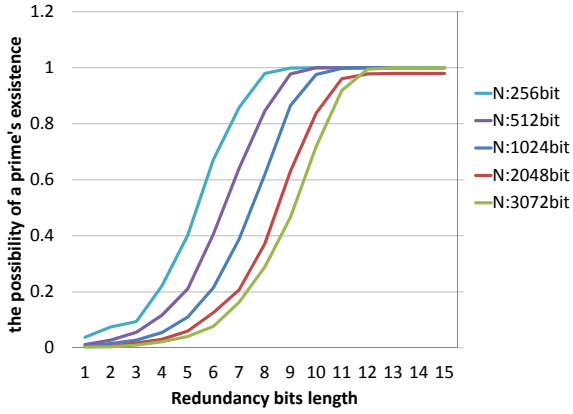

図 4: 素数生成の成功確率

える．この結果より，$(\log_e q)^{-1}$ は $(0.694\,|q|)^{-1}$，N の素因数 p,q をそれぞれ同じ長さと仮定すると，$(0.347\,|N|)^{-1}$ と表すことができる．

3.5.5 試行回数

ここでは，施行できる回数について考察する．Lenstra アルゴリズムにおいて，施行できる回数 T は $(N_0-N')/p$ である．理論解析より，$T=2^{r+1}-(N_0-N')/p$ であり，$C_2=\log_2 T$ とする．ここでは，この C_2 の値を実験値との比較により見積もる．

C_2 を縦軸に，r の値を横軸にとったのが以下のグラフとなる．

図 3 より，C_2 の値は冗長ビット長には依存せず，また公開鍵長にも依存していないことがわかる．どの場合においても 0.53 ± 0.005 であり，C_2 の値は定数で見積もることができる．ここでは最終的な確率の下限を見積もるため，C_2 の値を 0.5 とする．$C_2=r+1-\log_2((N_0-N')/p)$ より，$(N_0-N')/p=2^{r+1-C_2}$ と表すことができるため，試行回数は $2^{r+1-0.5}$ より，$2^{r+0.5}$ となる．

3.6 結論

定理 3.4 における C_1，C_2 は実験的に計算することができ，試行回数は $2^{r+0.5}$，素数である可能性は $(0.347\,|N|)^{-1}$ となる．これらの値より，Lenstra アルゴリズムの成功確率は $1-(1-(0.347\,|N|)^{-1})^{2^{r+0.5}}$ と表すことができる．

この式から導かれる確率を表 1 にまとめている．

4 Lenstra アルゴリズムの成功確率

Lenstra アルゴリズムの成功確率は，上記で述べた通り $1-(1-(0.347\,|N|)^{-1})^{2^{r+0.5}}$ となる．この値を元に，Lenstra アルゴリズムにおいて公開鍵がどの程度のビット長の時どの程度冗長ビット長が必要なのか，計算する．

以下の図が公開鍵のビット長と冗長ビット長を変化させた時の成功確率を表したグラフとなる．

図 4 から，冗長ビット長は，使用する公開鍵の長さに対して log を取った場合に等しくなる程度取れば成功確率が 99% 程度になる．また，確実性を持たせたければ 1bit 余分に取るとその成功確率は 1 に漸近する．また，この結果より以下の定理が成り立つ．

定理 4.1 (埋め込みに必要な冗長ビット長) 公開鍵長を N とすると，*Lenstra* アルゴリズムを効率的に成功させるために必要な冗長ビット長は $\log_2|N|$ となる．

	冗長ビット長														
公開鍵長	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
512 ビット	.007	.015	.034	.061	.119	.225	.400	.640	.870	.983	.999	.999	1.00	1.00	1.00
1024 ビット	.003	.007	.015	.031	.061	.119	.225	.399	.639	.870	.983	.999	.999	1.00	1.00
2048 ビット	.001	.003	.007	.015	.031	.061	.119	.225	.399	.639	.869	.983	.999	.999	1.00
4096 ビット	.001	.001	.003	.007	.015	.031	.061	.119	.224	.399	.639	.869	.983	.999	.999

表 1: Lenstra アルゴリズムにおける効率的な終了確率

5 結論

RSA 暗号における埋め込みアルゴリズムに関して，効率的な生成限界埋め込み長に関して論じた．実験結果から公開鍵 N に情報を埋め込む場合，$\log_2 |N|$ 程度の冗長ビット長を取れば効率的に埋め込むことができるといえる．

また，今後の課題としては実際に使うアプリケーションにおいて，この冗長ビット長を加味した場合にどの程度影響があるのかの調査を行うことを考えている．

謝辞

本研究の一部は，日本学術振興会 科学研究費補助金 基盤 B（課題番号 23300027）による補助のもとで行われた．また，本研究の一部は科学技術振興機構 「暗号アルゴリズム解析と数理学に基づくネットワークセキュリティ強化の評価」の研究プロジェクトの助成を受けたものである．

参考文献

[1] A. Lenstra.“Generating RSA moduli with a predetermined portion” *In Advances in Cryptology-Asiacrypt' 98*, pp. 1 10. Springer, 1998.

[2] S.A. Vanstone and R.J. Zuccherato. “Short RSA keys and their generation” *Journal of Cryptology*, Vol. 8, No. 2, pp. 101 114, 1995.

[3] M. Joye “RSA moduli with a predetermined portion: Techniques and applications” *In Proceedings of the 4th international conference on Information security practice and experience*, pp. 116 130. Springer-Verlag, 2008.

[4] CS Laih and KY Chen, “Generating visible RSA public keys for pki” *International Journal of Information Security*, Vol. 2, No. 2, pp. 103 109, 2004.

[5] Motoki Kitahara, Takashi Nihide, Kouichi sakurai, ”A Method for Embedding Secret Key Information in RSA Public Key and Its Application”, Sixth International Workshop on Advances in Information Security, 2012