暗号の数理要綱 #8

4 RSA 暗号

4.1 RSA 暗号とは?

暗号化の前に平文は数字とみなせることを注意しておく。アルファベットが m 文字からなるとき，その文字と 1 から m までの数字を一対一に対応させる。例えばアルファベットが $\{\,\mathrm{a},\mathrm{b},\ldots,\mathrm{z}\,\}$ とすると，a に 0 を対応させ，b に 1 を対応させ，$\cdots$ z に 25 を対応させる。m 文字から成るアルファベットを使い，文字を k 個並べた文 t を考える。これを，$\{0,1,\cdots,m-1\}$ という数字を k 個並べて出来る数，すなわち k 桁の m 進数であると見なせす。すなわち平文が

$$t_{k-1}t_{k-2}\cdots t_2t_1t_0$$

である時,

$$t = t_{k-1}m^{k-1} + t_{k-2}m^{k-2} + \cdots + t_2m^2 + t_1m + t_0$$

と見なす。 ここで $t_i \in \{0,1,\cdots,m-1\}$ である。

$$t \leqq m^k - 1$$

であるから,

$$n \geqq m^k$$

となる自然数 n を取ることにより，m 種類の文字を k 個並べた文章は，必ず n より小さいある数に対応することになる。そこで，$\mathbb{Z}_n = \{[0],[1],\cdots,[n-1]\}$ の元を入れ替える変換，すなわち，ある全単射

$$f : \mathbb{Z}_n \to \mathbb{Z}_n$$

を定義することにより，文の暗号化ができることになる。

RSA 暗号は，その公開鍵と秘密鍵を次のように作成する。

(1) 非常に大きな 2 つの素数 $p,\ q$ をとり（それぞれ 10 進数でおよそ 100 桁〜200 桁くらい），$n = pq$ とする。

(2) $\varphi(n)$ は，1 から $n-1$ までの自然数の中で n と互いに素であるようなものの個数であった。$n-1$ 以下の 1 以外の数で n と共通の約数を持つものは

$$p,\ 2p,\ 3p,\ \cdots,(q-1)p,\ q,\ 2q,\ 3q,\ \cdots,(p-1)q$$

の全部で $q-1+p-1 = p+q-2$ 個なので,

$$\varphi(n) = n-1-(p+q-2) = pq-p-q+1 = (p-1)(q-1)$$

である。

(3) e を $\varphi(n)$ と互いに素であって，$1 < e < \varphi(n)$ となる自然数とする。

(4) e は $\varphi(n)$ と互いに素なので，$\mathbb{Z}^*_{\varphi(n)}$ における $[e]$ の逆元 $[d]$ が存在する。すなわち d は $1 < d < \varphi(n)$ であり，

$$de \equiv 1 \mod \varphi(n)$$

を満たす数である。 mod の記号を使わなければ，ある整数 k が存在して

$$de - k\varphi(n) = 1$$

を満たす。

(5) **公開鍵**を (n, e) とし，**秘密鍵**を d とする。n を **RSA-module**，e を **暗号化指数** (*encryption exponent*)，d を **復号化指数** (*decryption exponent*) と呼ぶ。

公開鍵 (n, e) による暗号化のプロセス： Abel は Briskorn の公開鍵 (n, e) を使って，文章を次のように暗号化する。この操作は公開鍵を使うので，誰でも行うことができる。

まず，平文 t は $0 \leqq t < n$ となるある自然数である。

$$c = t^e \mod n$$

を計算する。この c が暗号文となる。

秘密鍵 d による復号化のプロセス： Briskorn は Abel から暗号文 c を受け取る。Briskorn は自分自身の秘密鍵 d を知っているので，

$$c^d$$

を計算する。

$$de \equiv 1 \mod \varphi(n)$$

なので，ある整数 r があって，

$$de = k\varphi(n) + 1$$

と書ける。

$$c^d = (t^e)^d = t^{ed} = t^{k\varphi(n)+1} = t^{k\varphi(n)}t = \left(t^{\varphi(n)}\right)^k t \mod n$$

である。定理 3.12 (一般化されたフェルマーの小定理) より，

t と n が互いに素ならば！

$$t^{\varphi(n)} = 1 \mod n$$

であるから，

$$\left(t^{\varphi(n)}\right)^r t = t \mod n$$

となり，

$$c^d = t \mod n$$

であることがわかった。すなわち，もとの平文 t に復号化されたわけである。

t と n が互いに素ではないなら，必ずしもこうなるとは限らないが，$t < n$ であり，n より小さい数で n と共通の約数を持つ数は，上で述べたように

$$p,\ 2p,\ 3p,\ \cdots,(q-1)p,\ q,\ 2q,\ 3q,\ \cdots,(p-1)q$$

の全部で $q-1+p-1=p+q-2$ 個しかない。p と q が非常に大きな数ならば，$p+q-2$ は $n=pq$ に比べて非常に小さい。実際，p, q が共に 10^k より大きいならば，

$$\frac{p+q-2}{pq}=\frac{1}{q}+\frac{1}{p}-\frac{2}{pq}<\frac{1}{q}+\frac{1}{p}<\frac{1}{10^k}+\frac{1}{10^k}=\frac{2}{10^k}$$

である。

これがどれほど小さい割合であるのか，ということを考えてみると，例えば，$k=50$ 程度の，p, q の取り方としては，かなり小さな値にしたとしても，$\frac{2}{10^{50}}$ という数は，全人類の中からランダムに 1 人だけを選んで百万ドルを与える，という宝くじが行われた時に，5 回連続して，偶然，自分が選ばれる，という確率よりもはるかに小さな割合である。

秘密鍵による暗号化とその公開鍵による復号化： 秘密鍵 d による暗号化と，その公開鍵 (n,e) による復号化に関しては，そのプロセスは，上の公開鍵による暗号化と秘密鍵による復号化の場合と全く同じである。

n は常に公開しているので，公開鍵が (n,e) であり，秘密鍵が (n,d) であると言っても良いが，e と d の役割は，互いに $\mathbb{Z}^*_{\varphi(n)}$ における他の逆元であるというだけで，それ以外に本質的な違いはない。つまり，全く同じことを e と d を入れ替えて行っても良いのである。

RSA 実装におけるアルゴリズム上の問題： RSA を実現するためには，次の 3 つのことをどのように実行するのか，ということが問題となってくる。

(1) 非常に大きな 2 つの数 a, b に対して，

$$a^b \mod n$$

を計算すること。

(2) 100 桁〜200 桁という，非常に大きな 2 つの素数を見つけること。しかも，それは良く知られているような素数ではなく，ランダムに選ばれたものでなければならない。

(3) e の値として何を採用するか？

(1) については，そのようなことが短時間で可能であることが簡単にわかる (高速指数計算法)。

(2) は難しい問題である。100 桁という大きな数が素数であるかどうかを判定するには，素因数分解ができないことを証明しなければならない。しかし，RSA の安全性は，次節で述べるように，百桁以上の大きな数の素因数分解が極めて難しい，という点によって保障されている。これは，RSA が致命的な矛盾をその内部に孕んでいることを示しているように見える。しかしこの矛盾は，巧妙な方法で回避される。これについても，後の節で詳しく述べることにする (素数判定)。

(3) RSA 暗号では，$\varphi(n)$ と互いに素となる数 e をとって，(n,e) を公開鍵としている。この e としてどのような数をとったら良いのだろうか。

まず e は $\varphi(n)$ と互いに素でなければならないので，とりあえず $3 \leqq e \leqq \varphi(n) - 1$ となる素数をとって，e が $\varphi(n)$ を割り切るかどうかをチェックすれば，この点については問題はない。

この e は，値を公開してしまうので，特にわかりにくい数にしてしまう必要もないが，あまり小さな数をとると，暗号を破られてしまうことも知られている（低指数攻撃)。

計算のし易さのためには小さな値の方が良いが，安全性のためには，ある程度大きな値にする必要がある。実際には，$e = 2^{2^4} + 1 = 65537$ という素数が使われることが多いようである。

4.2 RSA 暗号の安全性

RSA 暗号の安全性は,

- (n, e) を知った時に d を計算できるか ？

という問題に帰着される。

- もし n を素因数分解することができれば，$n = pq$ であることがわかる。
- そうすれば，$\varphi(n) = (p-1)(q-1)$ がわかる。
- e は公開されているので，$\varphi(n)$ がわかれば $\mathbb{Z}^*_{\varphi(n)}$ における e の逆元 d は，拡張ユークリッドアルゴリズムで簡単に計算できる。
- というわけで，d を知ることができるかどうか，という問題は，n を素因数分解できるかどうか，という問題に帰着される。

大きな自然数を素因数分解するための方法はいろいろ開発されている。まず，最も単純な方法は，その数よりも小さな素数，あるいは素数である可能性がある全ての数で割って行く，という方法である。

n が合成数ならば必ず $\sqrt{n}$ に等しいかそれより小さい素因数を持つので，n を素因数分解しようと思ったら，$\sqrt{n}$ に等しいかそれよりも小さな素数で n を割ってみれば良い。

自然数 n よりも小さな素数の数は，約 $n/\log n$ 個くらいあることが知られている。

n が 10 進数で 100 桁くらいの数なら，$\sqrt{n}$ は 50 桁くらいの数になる。$\sqrt{n}$ よりも小さな素数の数は，約 $10^{50}/115$ 個くらいあるので，約 10^{48} 個くらいある。

記憶装置に記録することができる数の個数は，せいぜい 10^{15} 個程度なので，10^{50} より小さな素数を全て記録しておくことは全く不可能だが，たとえそれができたとしても，10^{48} 回の演算を現実的な時間内で行うことは不可能である。

> 現在，世界最高速のスーパーコンピューターと言われているのは，ＩＢＭの Roadrunner で，その計算速度は約 1 Peta-FLOPS である。これは，1 秒間に 10^{15} 回程度の演算ができるということだが，それでも 1 年間でもせいぜい 10^{23} 回である。このようなコンピューターを 100 億台並列にしたとしても，1 年間にできる演算回数は 10^{33} 回程度であり，10^{40} 回以上の演算を必要とする計算は，現在の計算機の基本原理の上では，現実的に不可能と言って良い。(量子コンピューターなど，全く異なる原理に基づいた方法では，どうなるのかはわからない。)

素因数分解の方法は，このような単純な方法だけではなく，いろいろなアルゴリズムが考案されている。たとえ n がかなり大きな数であったとしても，その素因数の形によっては，あるアルゴリズムによって短時間で素因数分解されてしまうこともある。実際 1995 年，公式に使われていた RSA 暗号が，アメリカの 2 人の大学生によって破られてしまったこともある。

このように RSA 暗号といえども完全なものではなく，知られている素因数分解アルゴリズムによって簡単に破られないように，注意深く p と q を選ばなければならない。

また，次のような問題も考えられる。

- n を素因数分解しないで RSA 暗号を破る方法はないのか？

この問題についても，現在のところその解答は知られていない。RSA 暗号を破るには n の素因数分解が必要であろう，と信じられているようだが，今のところ，その保障がないのもまた事実なのである。

4.3 高速指数計算法

すでに述べたように，RSA 暗号を実行するためには，非常に大きな 3 つの数 a, b, n に対して，

$$a^b \mod n$$

が高速で計算できなければならない。

> a と n が互いに素ならば，定理 3.12(一般化されたフェルマーの小定理) より，$a^{\varphi(n)} \equiv 1 \mod n$ が成り立っているが，今の場合はこれは使えない。なぜなら，n が非常に大きな数の場合，$\varphi(n)$ を計算する効率的なアルゴリズムがないからである。（そのようなアルゴリズムがあれば，RSA 暗号が解読されてしまう。）また，たとえ $\varphi(n)$ がわかったとしても，$b = k\varphi(n) + r$ とすると結局 $a^r \mod n$ を計算しなけらばならず，一般化されたフェルマーの小定理 だけでは，最終結果には至らない。

$\mod n$ での高速指数計算は次のように行う。

- まず b を 2 進数で表す。

  $$b = b_0 + b_1 2 + b_2 2^2 + b_3 2^3 + \cdots b_{m-1} 2^{m-1}$$

  ここで，b_i は 0 か 1 である。

  b が 10 進数で 200 桁くらいの数であれば，2 進数では，664 桁くらいなので，ここの m としては，1000 以下くらいの数と思って良い。

$$\begin{aligned} a^b &= a^{b_0 + b_1 2 + b_2 2^2 + b_3 2^3 + \cdots b_{m-1} 2^{m-1}} \\ &= \left(a^{b_0}\right) \left(a^2\right)^{b_1} \left(a^{2^2}\right)^{b_2} \left(a^{2^3}\right)^{b_3} \cdots \left(a^{2^{m-1}}\right)^{b_{m-1}} \end{aligned}$$

となる。 ここで

$b_i = 0, 1$ なので，b_i は，単にそれが付いている数を掛けるのか掛けないのか，を表しているのである。

この指数計算を実際に実行することは不可能である。10 進数で a が 100 桁，b が 100 桁くらいの数ならば，a^b は 10^{100}-桁くらいの数になり，記憶装置に記憶することすら到底不可能である（世界最大の記憶装置ですら，10^{18} byte 程度しかない）。

しかし我々が求めているのは $\mod n$ での値である。これらの計算を全て $\mod n$ で行うことを考えてみる。

- $R = a^b \mod n$ を求めたい。そのためには，

$$a,\ a^2,\ a^{2^2},\ a^{2^3},\ \cdots,\ a^{2^m}$$

を $\mod n$ で求めれば良い。

- $a_1 = a^2 \mod n$ とおき，帰納的に，

$$a_{k+1} = a_k^2 \mod n$$

とおく。

-

$$a^{2^{k+1}} = a^{2^k \cdot 2} = \left(a^{2^k}\right)^2$$

なので，

$$a_k = a^{2^k} \mod n$$

である。

- あとは，これらの a_k を，$b_k = 1$ の場合のみ，$\mod n$ をとりながらかけて行けばよい。すなわち，

$$R_0 = a^{b_0} \mod n$$

とおき，

$$R_k = R_{k-1} \cdot a_k^{b_k} \mod n$$

とすれば，

$$R_{m-1} = a^b \mod n$$

となる。

- 各ステップで行われるのは，100 桁程度の数の掛け算と　mod　の計算だけであり，それを 1000 回程度行うのは，それほどの計算量ではない。

  しかし RSA 暗号においては，このような指数計算が多く出てくるので，高速化を求めるならば，アルゴリズムの改良や専用プロセッサーの開発など，やるべきことはあるかもしれない。