公開鍵暗号、電子署名そして鍵共有

情報セキュリティ大学院大学　有田正剛

平成 19 年 8 月 8 日

目 次

第1章　まえがき

公開鍵暗号、電子署名そして鍵共有技術は、公開鍵暗号系の 3 大要素技術である。この 3 大要素技術が現代の電子情報社会を支える基盤の一つであるのは周知の通りである。今や我たちはそれとは意識せずに日々色々な場面で (Internet では SSL、高速道路では ETC、また駅では Suica で) これらの技術を用いている。

これら公開鍵暗号、電子署名そして鍵共有技術には共通のパターンがある。第 1 段階として、まず、数論的な難問を利用して基本的安全性をもつ技術を組み立てる。この基本的安全性は、私たちや私たちが作り出したコンピュータにとって整数論が易しくはないことを積極的に利用することで実現される。

しかし、数論的な難問から直接的に得られる基本的安全性は、暗号技術としては不十分なことが多い。乱数やハッシュ関数などを用いた暗号学的なテクニックによって、基本的安全性を、より強力な攻撃者に対しても通用するものへと強化するのが第 2 段階である。

この入門講義では、公開鍵暗号、電子署名そして鍵共有技術について、この第 1 段階と第 2 段階からなるパターンをなるべく明確にして、紹介したいと思う。

第2章　公開鍵暗号

2.1　はじめに

公開鍵暗号は予め秘密を共有しないでおいて秘密の通信を行う方法である。そのような公開鍵暗号をつくる上で、数学の一分野である整数論がとても役に立つ。ここでは例として Rabin 暗号を取り上げ、それが整数の平方演算を利用して作られることをみる。そして、私たちや私たちが作り出したコンピュータにとって整数を因数分解することが困難であるために、Rabin 暗号が暗号として成立する（すなわち、暗号文から元のメッセージを求めることができない）のをみる。

しかし、公開鍵暗号をさらに「強秘匿」なものにするには、整数論だけでは足りない。我々にとって整数論的課題のどこが難しいのかを見極め、乱数をうまく用いて、ある工夫を施す必要がある。そのような工夫を用いて、Rabin 暗号を強化することで、強秘匿な暗号（すなわち、暗号文から元のメッセージのどのような部分情報も求めることができない暗号）である Blum-Goldwasser 暗号が作られることをみる。

2.2　公開鍵暗号の基本機能

暗号とは、秘密の通信を行うために利用される技術である。暗号には、大きく分けて、共通鍵暗号と公開鍵暗号がある。共通鍵暗号は、「予め共有した秘密を用いて秘密の通信を行う方法」である。予め共有した秘密は共通鍵と呼ばれる。送信者はメッセージを共通鍵で暗号化し、暗号文を受信者に送る。暗号文を受け取った受信者はそれを共通鍵で復号してもとのメッセージを得る。予め共有した秘密があるなら、それを用いれば秘密通信ができだろうことは、具体的な方法はともかくとして、もっともなことである。

一方、公開鍵暗号は「予め秘密を共有しないで秘密の通信を行う方法」である。これは、一見すると受けいれがたく、そんなことができるものか、と思えるが、我たちやコンピュータの能力が有限であることを利用すれば実現できる。

公開鍵暗号では各自の鍵は公開鍵と私有鍵の組からなる。公開鍵は（そ

の所有者への）メッセージの暗号化に用い、私有鍵は（その所有者が自分宛の）暗号文の復号に用いる。ある公開鍵で暗号化された暗号文は、それとセットになっている私有鍵でしか、復号されないようになっている。

公開鍵暗号を用いて、実際にメッセージを暗号化して送るには、以下のようにする。まず、受信者は自分の公開鍵を、その名の通り、公開する。送信者は、公開された受信者の公開鍵を手に入れ、この公開鍵を用いてメッセージを暗号化し、受信者に送る。暗号文を受け取った受信者はそれを自分の私有鍵で復号してもとのメッセージを得る。ここで、この受信者以外は、メッセージの暗号化に用いられた公開鍵に対応する私有鍵を持っていないので、この暗号文を復号することはできない。

2.3　公開鍵暗号の定義

公開鍵暗号は、鍵生成アルゴリズム G、暗号化アルゴリズム E そして復号アルゴリズム D の 3 つの効率的なアルゴリズムの組と定義される。

鍵生成アルゴリズム G は、セキュリティパラメータ k を入力すると公開鍵 e と私有鍵 d の組を出力する：

$$(e, d) \leftarrow \mathsf{G}(k).$$

ここで、セキュリティパラメータ k とは暗号の強度を指定するパラメータである。

暗号化アルゴリズム E は、公開鍵 e とメッセージ m を入力すると暗号文 c を出力する：

$$c \leftarrow \mathsf{E}(e, m).$$

復号アルゴリズム D は、私有鍵 d と暗号文 c を入力するとメッセージ m を出力する：

$$m \leftarrow \mathsf{D}(d, c). \tag{2.1}$$

ただし、

完全性 G によって生成された任意の鍵ペア (e, d) と任意のメッセージ $m(\in \{0,1\}^{l(k)})$ に対して

$$\mathsf{D}(d, \mathsf{E}(e, m)) = m$$

でなければならない。ここで、$l(k)$ はセキュリティパラメータ k に対応するメッセージ長を表し、$\{0,1\}^{l(k)}$ は長さ $l(k)$ のバイナリ文字列全体を表

す。すなわち、完全性とは、正直に暗号化して復号したら必ずもとのメッセージが得られるということを意味する。

また、暗号文 $c(\leftarrow \mathsf{E}(e,m))$ を入手した敵が、c および公開鍵 e を用いて、m を計算できるようでは暗号といえない。つまり、

一方向性 どのような効率的なアルゴリズム A も e と c から m を求めることはできない

ことが必要である。

公開鍵暗号を作るとは、完全性と一方向性を同時に満たす 3 つのアルゴリズムの組 $(\mathsf{G},\mathsf{E},\mathsf{D})$ を求めることに他ならない。

2.4　Rabin 暗号

公開鍵暗号の一例として、Rabin 暗号を紹介する。Rabin 暗号は整数を法とする平方演算の特性を利用してつくられる。

2.4.1　必要な整数論

まず、必要となる整数論的な事実を述べる。

n を異なる 2 つの素数 p,q の積である整数とする。$a \bmod n$ は整数 a を整数 n でわった余りを表す。

整数 n に対して $\mathbb{Z}_n = \{0,1,\ldots,n-1\}$ とする。Q_n を n を法とする平方剰余の集合とする。すなわち、Q_n はある $b(\in \mathbb{Z}_n)$ について $a = b^2 \bmod n$ となる $a(\in \mathbb{Z}_n)$ の集合である。n を法とする平方剰余 a に対して、$a = b^2 \bmod n$ となる b を、a の n を法とする平方根と呼ぶ。

事実 1 整数 $n(=pq)$ の素因数分解を効率的に実行するアルゴリズムが存在しないならば、n を法とする平方根を計算する効率的なアルゴリズムも存在しない。

事実 2 任意の素数 p を法とするならば、平方根を計算する効率的なアルゴリズムが存在する。すなわち、ある効率的なアルゴリズム Sqrt があって、素数 p と、p を法とする平方剰余の集合 Q_p からランダムに選ばれた a を入力として、$a = b^2 \bmod p$ となる b を出力する: $b \leftarrow \mathsf{Sqrt}(p,a)$.

事実 1 と事実 2 は、整数 $n(=pq)$ に対して、平方関数 $y = \mathrm{RABIN}_n(x) = x^2 \bmod n$ がトラップドア付き一方向関数であることを示している。

すなわち、順方向の $y = \mathrm{RABIN}_n(x)$ の計算は、平方 x^2 を計算し n で割った余りを計算するだけなので効率的に実行できる。しかし、事実 1 よ

り、逆に y から $x = \mathrm{RABIN}_n^{-1}(y)$ を求める効率的なアルゴリズムは (素因数分解が困難ならば) 存在しない。

ところが、n の素因子 p, q が与えられれば、そのような逆方向の計算も効率的に実行できる: $x = \mathrm{RABIN}_n^{-1}(p, q, y)$. (この p, q のような一方向性を破る情報をトラップドアと呼ぶ。) 実際、アルゴリズム Sqrt を用いて y の $\bmod p$ での平方根 x_p を計算する。同様に、y の $\bmod q$ での平方根 x_q を計算する。(x_p, x_q) を中国人の剰余定理を用いて $\bmod n$ にリフトして y の n を法とする平方根 x を得る。

2.4.2　暗号方式

Rabin 暗号 $Rabin = (\mathsf{G}, \mathsf{E}, \mathsf{D})$ は、トラップドア付き一方向関数 $y = \mathrm{RABIN}_n(x)$ を用いて、以下のように構成される。

鍵生成アルゴリズム $\mathsf{G}(k)$
: それぞれ k ビットの異なる素数 p, q をランダムに生成する。$n = pq$ を公開鍵 e とし、素数 p, q を私有鍵 d とする。

暗号化アルゴリズム $\mathsf{E}(e, m)$
: 公開鍵 e から n を取り出して、$c = m^2 \bmod n$ を計算し、c を出力する。

復号アルゴリズム $\mathsf{D}(d, c)$
: 私有鍵 d から p と q を取り出して、アルゴリズム Sqrt を用いて c の $\bmod p$ での平方根 $\pm m_p$ を計算する。同様に、c の $\bmod q$ での平方根 $\pm m_q$ を計算する。$(\pm m_p, \pm m_q)$ を中国人の剰余定理を用いて $\bmod n$ にリフトして (得られる 4 通りのメッセージ候補から)m を得て、出力する。

トラップドア付き一方向関数 $y = \mathrm{RABIN}_n(x)$ の n を公開鍵、トラップドア p, q を私有鍵とするわけである。

Rabin 暗号が完全性条件を満たしているのは、暗号化は平方演算で復号は平方根演算なので、明らかである。一方向性条件は、整数 $n(= pq)$ の素因数分解が困難であるとの仮定のもと、事実 1 からしたがう。

2.5　暗号の強秘匿性

2.5.1　Rabin 暗号の問題点

Rabin 暗号を用いて秘密通信をしてみる。今、送信者 Alice が受信者 Bob に (ある案件について) 賛成か反対かのいずれかを他者には秘密にして伝え

たいとする。あらかじめ、賛成は整数 m_{yes} で反対は整数 m_{no} で表すことになっている (このことについて Alice と Bob は事前に同意しているとする)。Alice は賛成しているとして、そのことを以下のように Rabin 暗号で Bob に伝える。Alice は、Bob の公開鍵 n_B を入手し、$c = m_{yes}^2 \bmod n_B$ を計算し、暗号文として c を Bob に送信する。Bob は、暗号文 c を受信し、自身の私有鍵 p_B, q_B を用いて c の n_B を法とする平方根を求めて、Alice のメッセージ m_{yes} を得る。この送信の途中で敵 Eve が c を覗き見しているかも知れないが、敵 Eve は暗号文 c と Bob の公開鍵 n_Bを 入手しても、対応する私有鍵 p_B, q_B を持たないので、c の n_B を法とする平方根、すなわち m_{yes} を求めることはできない、はずだった。

ところが、このケースでは、敵 Eve は c から Alice のメッセージ m_{yes} を簡単に求めることができる。Eve はまず m_{no} を Bob の公開鍵 n_B で暗号化して c' を求める。c' と c を比較するとそれらは (異なるメッセージ m_{no} と m_{yes} を暗号化しているので) 異なるので、c に暗号化されていたのは m_{yes} であることが分かる。

ここでは、メッセージのパターンが m_{yes} と m_{no} の 2 通りという極端な設定で考え、そのせいでメッセージが丸ごとばれてしまったが、メッセージがいく通りあってもやはり不都合がある。実際、Eve が自分でメッセージ m' を選択し、それを暗号化した結果 c' と c を比較すれば、m' が c に暗号化されているかどうかがわかる。これを繰り返せば、(c に暗号化された)m のとり得る範囲を狭めていくことができる。Rabin 暗号では、暗号文からもとのメッセージに関する情報 (例えば m と m' が等しいか否かの１ビット) が漏れているということである。

2.5.2　強秘匿性の定義

前節で Rabin 暗号では暗号文からもとのメッセージに関する情報が漏れてしまっているのを見た。Rabin 暗号を、もとのメッセージに関するどのような情報も漏れないよう、強化することができ、そのようにして得られる暗号として 2.6 節で紹介する Blum-Goldwasser 暗号 [1] がある。

しかし、その前に、ゴールを明確にしたい。「暗号文からもとのメッセージに関するどのような情報も漏れていない」とはどういうことか？個々の暗号方式に依存せず、普遍的でかつ容易に検証できる定義がほしい。それは暗号の強秘匿性と呼ばれ、つぎにように敵 A と挑戦者 C との間のゲームを用いて定式化することができる。

強秘匿性 公開鍵暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ に対して、敵 A と挑戦者 C との間でゲームを行う。ルールは以下の通り。まず、挑戦者 C が鍵生成アルゴリズム G を実行し、鍵ペア (e, d) をつくる。挑戦者 C は公開鍵 e を敵

A に渡す。e を受け取ったら、敵 A は何らかの計算をしてメッセージの組 (m_0, m_1) を選択し、挑戦者 C に渡す (ただし、m_0 と m_1 は同じ長さとする)。これに対して挑戦者 C は m_0 か m_1 かのいずれかをランダムに選び（これを m_b とかく）公開鍵 e で暗号化して暗号文 c をつくる: $c \leftarrow \mathsf{E}(e, m_b)$. 挑戦者 C は暗号文 c を敵 A に渡す。敵 A は受け取った暗号文 c が m_0 を暗号化したものか、m_1 を暗号化したものかどちらかを推測し、推測結果 b' を出力する。推測が当たったら ($b = b'$ なら) 敵 A の勝ち (そうでないなら挑戦者 C の勝ち) とする。

このとき、どのような敵 A もせいぜい $1/2$ の確率でしか勝てないならば、公開鍵暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ は**強秘匿**と呼ばれる。

もし、公開鍵暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ がメッセージの情報を 1 ビットも漏らしていないならば、敵 A がどんなに強力でも、c が m_0 を暗号化したのか m_1 を暗号化したのか、全く分からず、b' をあてずっぽうで出力するしかない。このとき敵 A が勝つ確率は $1/2$ である。

一方、公開鍵暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ がメッセージの情報を何らかの形で漏らしているならば、巧みな敵 A が、その漏洩を突くようなメッセージの組 (m_0, m_1) を選択して挑戦者 C に提出すれば、その敵 A は高い確率で勝つはずである。

2.6　Blum-Goldwasser 暗号

Rabin 暗号を強化することで、強秘匿な暗号である Blum-Goldwasser 暗号 [1] が得られる。強化のポイントは、暗号化に乱数を導入することと hard-core ビットの利用である。

2.4 節で見たように、関数 $y = \mathrm{RABIN}_n(x) (= x^2 \bmod n)$ は一方向関数 (の有力候補) である。しかし、y は x の全てを隠せているわけではない。実際、(正しい原像である)x と (x とは異なる)x' が与えられたとき、どちらが y の原像かと問われれば、$x^2 \bmod n$ を計算してその結果が y になることを確認すれば、正しい x がどちらか分かってしまう。このため、Rabin 暗号は強秘匿な暗号にはならないのだった。

ところが、$y (= \mathrm{RABIN}_n(x) = x^2 \bmod n)$ は、x の最下位ビット $x_0 = \mathrm{LSB}(x)$、すなわち x が偶数かそれとも奇数かということ、は隠せていることが知られている。つまり、y が与えられても、0 と 1 のどちらが x の最下位ビットなのか分からない。このことを関数 $y = \mathrm{RABIN}_n(x)$ は hard-core ビットとして最下位ビット $\mathrm{LSB}(x)$ をもつという。関数 $y = \mathrm{RABIN}_n(x)$ に限らず、どのような一方向関数も hard-core ビットをもつ（ように変形できる）ことが知られている。

Blum-Goldwasser $= (\mathsf{G}, \mathsf{E}, \mathsf{D})$ は以下のように構成される。

鍵生成アルゴリズム $\mathsf{G}(k)$

それぞれ k ビットの、8 でわって 7 余る、異なる素数 p, q をランダムに生成する。$n = pq$ を公開鍵 e とし、素数 p, q を私有鍵 d とする。

暗号化アルゴリズム $\mathsf{E}(e, m)$

m のビット長を l とする。公開鍵 e から n を取り出して、

1. $\mathbb{Z}_n$ からランダムに z を選び、$x_0 = z^2 \bmod n$ を求める。
2. $i = 1, \ldots, l$ に対して、$x_i = x_{i-1}^2 \bmod n$ を計算する。
3. $mask = (\mathrm{LSB}(x_0), \mathrm{LSB}(x_1), \ldots, \mathrm{LSB}(x_{l-1}))$ を求め、$c_1 = mask \oplus m$ を計算する ($\oplus$ はビット毎の排他的論理和)。
4. $c = (c_1, x_l)$ を出力する。

復号アルゴリズム $\mathsf{D}(d, c)$

私有鍵 d から p と q を取り出して、$c = (c_1, y)$ に対し、

1. p, q を用いて平方根演算を繰り返し、$y(= x_l)$ から x_0 を求める。
2. $i = 1, \ldots, l$ に対して、$x_i = x_{i-1}^2 \bmod n$ を計算する。
3. $mask = (\mathrm{LSB}(x_0), \mathrm{LSB}(x_1), \ldots, \mathrm{LSB}(x_{l-1}))$ を求め、$m = mask \oplus c_1$ を計算し出力する。

Blum-Goldwasser 暗号では、Q_n のランダム要素 x_0 をとり、$x_{i+1} = \mathrm{RABIN}_n(x_i)$ の hard-core ビット $\mathrm{LSB}(x_i)$ を利用して、(メッセージ m と同じ長さの) ビット列 $mask = (\mathrm{LSB}(x_0), \mathrm{LSB}(x_1), \ldots, \mathrm{LSB}(x_{l-1}))$ を生成し、$mask$ をメッセージ m に足しこむことによって暗号文 $c_1 = mask \oplus m$ を生成している。ただし、c_1 だけでは誰も復号できなくなるので、x_l を暗号文に添付している: $c = (c_1, x_l)$.

$c = (c_1, x_l)$ を受け取った正当な復号者は、私有鍵 p, q を用いて平方根演算を繰り返し、x_l から x_0 を求めることができる。一旦 x_0 が分かれば、暗号化のときと全く同じ手続きで $mask$ を生成すれば、$m = mask \oplus c_1$ によってメッセージ m が得られる。

一方、敵が暗号文 $c(= (c_1, x_l))$ に含まれる x_l を見ても、hard-core ビットの性質から c_1 を作るのに用いた $mask$ の各ビットは分からない。実際、$mask = (\mathrm{LSB}(x_0), \mathrm{LSB}(x_1), \ldots, \mathrm{LSB}(x_{l-1}))$ の各ビットを見ると

- $\mathrm{LSB}(x_{l-1})$ は $x_l(= x_{l-1}^2 \bmod n)$ の hard-core ビット
- $\mathrm{LSB}(x_{l-2})$ は $x_{l-1}(= x_{l-2}^2 \bmod n)$ の hard-core ビット

- ...
- LSB(x_0) は $x_1(= x_0^2 \bmod n)$ の hard-core ビット

となっている。敵は x_l を知っているが、hard-core ビットの性質から、LSB(x_{l-1}) は全く分からない。敵は、x_{l-1} の部分情報を知っているかもしれないが、LSB(x_{l-2}) は全く分からない。(x_{l-1} を知っていても LSB(x_{l-2}) は全く分からないから。) 以下同様に繰り返し、敵は、x_1 の部分情報を知っているかもしれないが、LSB(x_0) は全く分からない。

以上から、$mask$ は敵にとって未知の (メッセージ m と同じ長さの) 乱数列に等しいことが分かった。すると、$c = mask \oplus m$ は m の情報を完全に隠してしまうので、強秘匿性の定義のゲームに現れた攻撃者 A に勝ち目はなく、Blum-Goldwasser 暗号 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ は強秘匿となる。

2.7　おわりに

公開鍵暗号の例として、整数を法とした平方関数の一方向性を利用した、Rabin 暗号を紹介し、それを hard-core ビットを利用して強化することで強秘匿な Blum-Goldwasser 暗号が構成されることを見た。

ここでは、攻撃モデルとしては選択メッセージ攻撃しか扱わなかった。選択メッセージ攻撃とは、敵がメッセージを選択し、それに対する暗号文が与えられる攻撃モデルである。より強力な攻撃モデルとして、敵が暗号文を選択し、それに対するメッセージが与えられる攻撃である選択暗号文攻撃がある。また、安全性定義についても、強秘匿性のほかに頑健性と呼ばれる安全性がある。

第3章　電子署名

3.1　はじめに

電子署名はメッセージ（対象とする電子データ）の真正性を、誰もが検証できる形で、保証する電子データである。署名を作れるのは本人のみだが、検証はだれでもできるという状況を実現する。そのような電子署名をつくる上で、数学の一分野である整数論がとても役に立つ。ここでは例として RSA 署名を取り上げ、それが整数のべき乗剰余演算を利用して作られることをみる。そして、私たちや私たちが作り出したコンピュータにとってべき乗剰余演算の逆関数を計算することが困難であるために、RSA 署名が署名として成立する（すなわち、与えられたメッセージに対してその署名を偽造できない）のをみる。

しかし、電子署名をさらに「存在的偽造不可」なものにするには、整数論だけでは足りない。ハッシュ関数と呼ばれる暗号学的な圧縮関数を用いて強化する必要がある。そのような工夫により RSA 署名を強化した例として RSA-FDH 署名を紹介する。そして、RSA-FDH 署名が、ランダムオラクルモデルと呼ばれるある種のヒューリスティックスのもとで、選択メッセージ攻撃に対し存在的偽造不可であること（すなわち、敵がいくつかのメッセージの署名を入手したとしても、それら以外のどのようなメッセージに対してもその署名を偽造できないこと）の証明をスケッチする。

3.2　電子署名の基本機能

電子署名はメッセージ（対象とする電子データ）の真正性を、誰もが検証できる形で、保証する電子データである。署名を作れるのは本人のみだが、検証はだれでもできるという状況を実現する。

電子署名では各自は自分用の署名鍵と検証鍵の組をもつ。署名鍵は秘密に保管し、検証鍵は公開する。署名鍵はメッセージの署名を生成するのに用い、検証鍵は署名を検証するに用いる。ある署名鍵で生成された署名は、それとセットになっている検証鍵でしか、受理されないように作られる。

電子署名を用いて、実際にメッセージに署名を添付して送るには、以下

のようにする。送信者 Alice は自分の検証鍵 pk_A を公開しておく。Alice は、自身の署名鍵 sk_A を用いてメッセージ m の署名 σ を生成し、メッセージ m とともに受信者 Bob に送る。メッセージと署名の対 (m, σ) を受け取った Bob は、Alice の検証鍵 pk_A を入手し、その検証鍵 pk_A を用いて (m, σ) の正当性をチェックする 正当なときのみメッセージ m を受けいれる。

ここで、Alice の署名鍵 sk_A は秘密に保管され、検証鍵 pk_A は公開されているので、

- 署名検証は Bob を含めてだれにでもできる。
- 署名生成は (署名鍵を知っている)Alice にしかできない。

ということが期待される。これが本当に正しいのなら、受信者 Bob から見て、受け取ったメッセージ m は確かに Alice が署名したと確信でき、また、送信者 Alice から見ると、メッセージ m に署名したということを、Bob のみならず第 3 者に対しても否定できないことになる。

3.3　電子署名の定義

電子署名は、鍵生成アルゴリズム Gen、署名生成アルゴリズム Sign そして署名検証アルゴリズム Vrfy の 3 つの効率的なアルゴリズムの組と定義される。

鍵生成アルゴリズム Gen は、セキュリティパラメータ k を入力すると検証鍵 pk と署名鍵 sk の組を出力する：

$$(pk, sk) \leftarrow \mathsf{Gen}(k).$$

ここで、セキュリティパラメータ k とは署名の強度を指定するパラメータである。

署名生成アルゴリズム Sign は、署名鍵 sk とメッセージ m を入力すると署名 σ を出力する：

$$\sigma \leftarrow \mathsf{Sign}(sk, m).$$

署名検証アルゴリズム Vrfy は、検証鍵 pk とメッセージ m と署名 σ を入力とし、0 または 1 を出力する (0 は NG、1 は OK):

$$0/1 \leftarrow \mathsf{Vrfy}(pk, m, \sigma). \tag{3.1}$$

ただし、

完全性 Gen によって生成された任意の鍵ペア (pk, sk) と任意のメッセージ m に対して

$$\mathsf{Vrfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$$

でなければならない。これは、正しく作られた署名は必ず受理されることを意味する。

また、安全性条件として

偽造不可能性 どのような効率的なアルゴリズム F も、与えられた m に対して (sk を知らずに)pk だけを用いて妥当な署名 σ をつくることはできない

ことが必要である。

電子署名を作るとは、完全性と偽造不可能性を同時に満たす 3 つのアルゴリズムの組 $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ を求めることに他ならない。

3.4　RSA 署名

電子署名の一例として、RSA 署名を紹介する。RSA 署名は整数を法とする「べき乗剰余関数」の特性を利用してつくられる。

3.4.1　必要な整数論

まず、必要となる整数論的な事実と仮定を述べる。

n を異なる 2 つの素数 p,q の積である整数とする。$a \bmod n$ は整数 a を整数 n でわった余りを表す。また、a と b の差が n で割り切れるとき、a と b を n を法として等しいといい、$a \equiv b \pmod{n}$ とかく。

整数 n に対して $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ とする。$l = \phi(n) = (p-1)(q-1)$ とする。

事実 3 $ed \equiv 1 \pmod{l}$ ならば任意の $a(\in \mathbb{Z}_n)$ に対して $(a^e)^d \equiv (a^d)^e \equiv a \pmod{n}$.

仮定 1 (RSA 仮定) $n(= pq)$ を十分大とする。y を $\mathbb{Z}_n$ からランダムに選ぶ。このとき、e, y, n を与えられて $y = x^e \bmod n$ となる x を求める効率的なアルゴリズムは存在しない。

仮定 1 は RSA 仮定と呼ばれ、その正しさは一般につよく信じられている。

事実 3 と仮定 1 は、整数 $n(=pq)$ に対して、べき乗剰余関数 $y = \mathrm{RSA}_{n,e}(x) = x^e \bmod n$ がトラップドア付き一方向関数であることを示している。

すなわち、順方向の $y = \mathrm{RSA}_{n,e}(x)$ の計算は、Square-And-Multiply 法と呼ばれるアルゴリズムを用いて効率的に実行できる。しかし、仮定 1 より、逆に y から $x = \mathrm{RSA}^{-1}_{n,e}(y)$ を求める効率的なアルゴリズムは存在しない。

ところが、n の素因子 p, q が与えられれば、そのような逆方向の計算も効率的に実行できる: $x = \mathrm{RSA}^{-1}_{n,e}(p, q, y)$. (この p, q のような一方向性を破る情報をトラップドアと呼ぶ。) p, q を知っていると、$l = (p-1)(q-1)$ を計算でき、この l を用いれば、$d = e^{-1} \bmod l$ によって e から d を計算できるので、$x = y^d \bmod n$ が求める x である。実際、事実 3 より、$x^e \equiv (y^d)^e \equiv y \pmod n$ なので、$y = x^e \bmod n$ である。(これは、仮定 1 が成り立つためには、$n(=pq)$ の素因数分解が困難であることが必要なことを示している。)

3.4.2　署名方式

RSA 署名 $RSA = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ は、トラップドア付き一方向関数 $y = \mathrm{RSA}_{n,e}(x)$ を用いて、以下のように構成される。

鍵生成アルゴリズム $\mathsf{Gen}(k)$

それぞれ k ビットの異なる素数 p, q を生成する。$n = pq, l = (p-1)(q-1)$ とし、$ed \equiv 1 \pmod l$ となる e, d を生成する。(n, e) を検証鍵 pk とし、(n, d) を署名鍵 sk とする。

署名生成アルゴリズム $\mathsf{Sign}(sk = (n, d), m)$

署名鍵 sk から n, d を取り出して、$\sigma = m^d \bmod n$ を計算し、σ を出力する。

署名検証アルゴリズム $\mathsf{Vrfy}(pk = (n, e), m, \sigma)$

検証鍵 pk から n と e を取り出して、$m' = \sigma^e \bmod n$ を計算する。結果 m' が m に等しければ 1 を異なれば 0 を出力する。

RSA 署名の完全性は事実 3 から直ちに従う。偽造不可能性は、与えられた m に対してその署名 σ は $m = \sigma^e \bmod n$ を満たす整数 σ なので、仮定 1 そのものである。

3.5　選択メッセージ攻撃における存在的偽造不可能性

3.5.1　RSA 署名の問題点

先に見たように、RSA 署名は与えられた (検証鍵 $pk = (n, e)$ と) メッセージ m に対して偽造不可能性をもつ。すなわち、どのような (署名鍵をもたない、効率的な) 敵も与えられた m に対して $\sigma^e = m \bmod n$ となる σ を生成することはできない。しかし、与えられた m に限らず、とにかく検証式 $\sigma^e = m \bmod n$ を満足するような (m, σ) ならすべて偽造と認めるならば (このような偽造を存在的偽造と呼ぶ)、RSA 署名はいくらでも偽造可能である。実際、σ をランダムに生成し、$m = \sigma^e \bmod n$ によって m を求め、(m, σ) を出力すればよい。このとき m もランダムになるので通常意味のあるメッセージとはならないが、アプリケーションによってはこのようなランダムメッセージに対する署名であっても意味をもつ場合がある。

また、アプリケーションによっては、敵が自ら選んだいくつかのメッセージに対する正当な署名を何らかの手段で入手できるような場合があり得る。このような状況における敵の攻撃を選択メッセージ攻撃と呼ぶが、この攻撃のもとでは RSA 署名は与えられた m に対してさえ偽造可能である。実際、敵 F は次にようにして、(自分で選んだ) たった 2 つのメッセージ m_1, m_2 に対する署名 σ_1, σ_2 を入手すれば、与えられた m の署名 σ を計算することができる。

敵 F は検証鍵 $pk = (n, e)$ とメッセージ m を入力として受け取り、次のように動作する。まず、乱数 $r(\in \mathbb{Z}_n)$ を生成し、$m_1 = mr \bmod n$ を計算し、m_1 の正当な署名 σ_1 を入手する。次に、$m_2 = r^{-1} \bmod n$ を計算し、m_2 の正当な署名 σ_2 を入手する。最後に、σ_1 と σ_2 の積 $\sigma = \sigma_1\sigma_2 \bmod n$ を計算し、(m, σ) を出力する。

この σ に対して、$\sigma^e \equiv (\sigma_1\sigma_2)^e \equiv \sigma_1^e\sigma_2^e \equiv m_1m_2 \equiv m \bmod n$ となるので、(m, σ) は妥当である。

3.5.2　選択メッセージ攻撃における存在的偽造不可能性の定義

前節で RSA 署名では存在的偽造を防げないこと、また選択メッセージ攻撃においては (存在的と限らない) 偽造そのものを防げないことを見た。RSA 署名を、次節で見る RSA-FDH 署名のように、選択メッセージ攻撃において存在的偽造すらできないように強化することができる。

しかし、その前に、「選択メッセージ攻撃において存在的偽造すらできない」ということを、つぎにように敵 F と挑戦者 C との間のゲームを用いて定式化する。

選択メッセージ攻撃における存在的偽造不可能性

電子署名 (Gen, Sign, Vrfy) に対して、敵 F と挑戦者 C との間でゲームを行う。ルールは以下の通り。まず、挑戦者 C が鍵生成アルゴリズム Gen を実行し、鍵ペア (pk, sk) をつくる。挑戦者 C は検証鍵 pk を敵 F に渡す。pk を受け取ったら、敵 F は何らかの計算をしてメッセージ m_i を選択し、挑戦者 C に渡す。これに対して挑戦者 C は署名鍵 sk を用いて m_i の署名 σ_i を計算する: $\sigma_i \leftarrow \mathsf{Sign}(sk, m_i)$. 挑戦者 C は署名 σ_i を敵 F に渡す。このような m_i と σ_i のやりとりを敵 F は挑戦者 C との間で好きなだけ繰り返すことができるものとする。敵 F は受け取った (複数の)σ_i の情報を用いて、何らかのメッセージと偽造署名の対 (m, σ) を出力する。もしも m がどの m_i とも異なり、σ が m の妥当な署名 (すなわち $\mathsf{Vrfy}(pk, m, \sigma) = 1$) ならば、敵 F の勝ち (そうでないなら挑戦者 C の勝ち) とする。

このとき、どのような敵 F もほとんど 0 に等しい確率でしか勝てないならば、電子署名 (Gen, Sign, Vrfy) は**選択メッセージ攻撃において存在的偽造不可能**であると呼ぶ。

3.6　RSA-FDH 署名

RSA 署名において、メッセージにハッシュ関数を施すことで、より偽造が困難な RSA-FDH 署名 [2] が得られる。ここで、ハッシュ関数 H とは (暗号学的な) 圧縮関数であって、全てのビット列の集合 $\{0,1\}^*$ から $\mathbb{Z}_n$ への関数である：$H : \{0,1\}^* \to \mathbb{Z}_n$. ただし、どのような効率的なアルゴリズムも $H(x) = H(y)$ となる異なる x, y を見つけることはできないとする (衝突困難性)。

RSA-FDH 署名=(Gen, Sign, Vrfy) は以下の通り。

鍵生成アルゴリズム $\mathsf{Gen}(k)$ /* RSA 署名と同じ */

それぞれ k ビットの異なる素数 p, q を生成する。$n = pq, l = (p-1)(q-1)$ とし、$ed \equiv 1 \pmod{l}$ となる e, d を生成する。(n, e) を検証鍵 pk とし、(n, d) を署名鍵 sk とする。

署名生成アルゴリズム $\mathsf{Sign}(sk = (n, d), m)$

署名鍵 sk から n, d を取り出して、$\sigma = H(m)^d \bmod n$ を計算し、σ を出力する。

署名検証アルゴリズム $\mathsf{Vrfy}(pk = (n, e), m, \sigma)$

検証鍵 pk から n と e を取り出して、$h' = \sigma^e \bmod n$ を計算する。結果 h' が $h = H(m)$ に等しければ 1 を異なれば 0 を出力する。

RSA-FDH 署名に対しては、RSA 署名に対して成立した先の攻撃はうまくいかない。

まず、存在的偽造について見る。先と同様に、σ をランダムに生成し、$h = \sigma^e$ とする。偽造を完成させるにはこの h に対して、さらに $h = H(m)$ となる m を求めなければならない。これは衝突困難性より不可能である。(実際、h から $h = H(m)$ となる m を計算できるアルゴリズム I が存在するとすると、ランダムな x に対して $h = H(x)$ とし、h を I に入力すると $h = H(m)$ となる m が得られ、衝突ペア m, x が得られてしまう。)

また、先の選択メッセージ攻撃も、やはり衝突困難性より $H(m) = H(m_1)H(m_2)$ とはならないので、うまくいかない。(実際、$m = m_1 m_2 \bmod n$ である任意の m_1, m_2 に対していつも $H(m) = H(m_1)H(m_2)$ となるようだと、$H(m)$ の値は m そのものよりも $m \bmod n$ で決まっている。)

しかし、だからといってどのような選択メッセージ攻撃を行っても存在的な偽造ができないといえるだろうか? ランダムオラクルモデルと呼ばれるヒューリスティックに頼ればそのようにいえる。

Theorem 1 *RSA 仮定 (仮定 1) が真ならば、RSA-FDH 署名はランダムオラクルモデルのもとで選択メッセージ攻撃において存在的偽造不可能である。*

まず、ランダムオラクルモデルについて述べる。ランダムオラクルモデルとは、アルゴリズムの実行モデルであって、各アルゴリズムにおけるハッシュ関数 $H(\cdot)$ の実行を、ランダムオラクル O_H への問い合わせに置き換えてしまうモデルである。各アルゴリズムは m に対して $H(m)$ を自力では計算せず、なぜか必ずランダムオラクル O_H へ m を問い合わせる。m を受け取ったランダムオラクル O_H は m に対して乱数 $h(\in \mathbb{Z}_n)$ を割り当てて $H(m)$ の値として返す。h を受け取ったアルゴリズムは $h = H(m)$ として以降の計算を通常どおり続行する。

ハッシュ関数 $H(\cdot)$ には衝突困難性が求められるので、通常、その値 $h = H(m)$ は乱数にしか見えないようなものになる。それなら $H(\cdot)$ の計算をランダムオラクル O_H への問い合わせに置き換えても安全性の検証上、一定の意味があるだろうという発想である。

つぎに、定理 1 の証明のスケッチを紹介する。選択メッセージ攻撃のもとで RSA-FDH 署名の存在的偽造に成功する敵 F が存在したと仮定する。すなわち、ある敵 F があって、F は先にみたゲームを挑戦者 C との間で実行し、無視できない確率で挑戦者 C に勝つとする。このゲームにおいて、F はランダムオラクルに何回か問い合わせを行い (ランダムオラクル

モデル)、また挑戦者 C に対していくつかのメッセージの署名を要求し、最終的にこれらの結果を用いて RSA-FDH 署名の偽造署名 (m, σ) を出力する。我々は、このような F があれば、それを用いて RSA 関数の逆関数の値を計算できる敵 I を作り出せることを示す。つまり、I は、F が署名を偽造する能力を利用して RSA 関数の逆関数の値を計算するわけである。しかし、このような I の存在は RSA 仮定に反するので定理の仮定のもとで許されない。そのようなことになってしまったのは上のような敵 F が存在すると仮定したためである。よって、F は存在せず、RSA-FDH 署名はランダムオラクルモデルのもとで選択メッセージ攻撃において存在的偽造不可能である。以上が、証明の大枠である。

目標である (RSA 仮定に対する) 敵 I は (RSA-FDH 署名に対する) 敵 F を用いて以下のように構成される。(敵 F がランダムオラクルに問い合わせを行う回数を q とする。) I には入力として n, e, y が与えられる。I の目的は $y = \mathrm{RSA}_{n,e}(x)$ となる x を求めることである。I は以下のようにして、F に対して挑戦者 C の役割を演じ、F の能力を利用する。I はまず 1 以上 q 以下の範囲の乱数 i^* を生成しておく。I は自身のサブルーチンとして F を起動する。つぎに I は挑戦者 C として $pk = (n, e)$ を RSA-FDH 署名の検証鍵ということにして F に与える。F は RSA-FDH 署名に対する攻撃を開始する。攻撃の途中で、F はランダムオラクルにハッシュ値を問い合わせたり、挑戦者 C に署名を問い合わせたりする。これらに対して I は以下のように返答する。

- ランダムオラクルへの (i 回目の) 問い合わせ m_i に対して
  I は $i = i^*$ かどうかみる。もしそうならば I は (ランダムオラクルからの返答として)y を返す。そうでないならば、x_i を $\mathbb{Z}_n$ からランダムに選択し、$y_i = \mathrm{RSA}_{n,e}(x_i)$ を計算し、y_i を返す。

- 挑戦者 C への署名問い合わせ m に対して
  I は $m = m_i$ となる i を求める。(F は m をすでにランダムオラクルに尋ねているとしてよい。その方が F にとって情報が増えて有利だから。) $i = i^*$ なら I は (RSA 仮定に対する) 攻撃をあきらめる。そうでないなら、I は x_i を (m の署名として)F に返答する。

これらの問い合わせ結果に基づき、F は何らかの計算を行って最後に (m^*, σ) を出力する。これに対し、$m^* = m_{i^*}$ なら I は σ を出力して終了する。(ここでも F は m^* をすでにランダムオラクルに尋ねているとしてよい。その方が F にとって情報が増えて有利だから。) そうでないなら I は攻撃をあきらめる。

上で、F は I の中で起動されているので、F にとって I がその世界の全てである。そのため、I は F に対してランダムオラクルや挑戦者の役割を演じている。もしもこの演技（シミュレーション）がまずければ、F はまともに動かず、その能力を利用できない。まず、I が F に与える入力 $pk = (n, e)$ は RSA-FDH 署名における公開鍵として妥当である。次に、I によるランダムオラクルのシミュレーションを検証する。I は $i = i^*$ のとき y を答えている。この y はもともと I への課題として $\mathbb{Z}_n$ からランダムに選ばれている。よって F からみてもランダムな $\mathbb{Z}_n$ の要素なのでシミュレーションは OK である。(ランダムオラクルは $\mathbb{Z}_n$ のランダムな要素を返すのだった。) $i \neq i^*$ のときは、I は x_i を $\mathbb{Z}_n$ からランダムに選択しているので、$y_i = \mathrm{RSA}_{n,e}(x_i)$ も $\mathbb{Z}_n$ のランダムな要素である。よって、やはり OK である。さらに、I による署名問い合わせに対する応答のシミュレーションを検証する。上で I は問い合わせ $m = m_i$ に対して $i \neq i^*$ のとき x_i を答えている。この答えは m に対する署名として正しい。実際、$(x_i)^e = y_i = H(m_i)$ である。ここで辻褄があうように I は $H(m_i)$ の値を $y_i (= x_i^e)$ に決めていたのである。(I がこのような “ズル” をしていようとは F からはまったく見えない。)

以上から、I は F に対してランダムオラクルや挑戦者の役割をほぼうまく演じていることがわかった。(うまくいかないケースは I が署名問い合わせに応答する際に $i = i^*$ となってしまう場合である。この場合は、I はあきらめてしまうのだった。しかし、i^* はランダムに選ばれているので、F が偽造の対象として m_{i^*} を選択する確率は無視できない程度 $(1/q$ 以上$)$ にある。このとき、ゲームのルールにより、F が m_{i^*} の署名を問い合わせることはない。) よって、F は無視できない確率で攻撃に成功し、妥当なメッセージと署名の組 (m^*, σ) を出力する。ここで、もしも $m^* = m_{i^*}$ となっていると、I によるランダムオラクルのシミュレーションの仕方から、$\sigma^e = H(m^*) = y$ なので、σ が目的の $(y = \mathrm{RSA}_{n,e}(x)$ となる$)x$ である。もちろん、$m^* = m_{i^*}$ となるとは限らないが、i^* はランダムに選ばれているので、$m^* = m_{i^*}$ となる確率は $1/q$ 以上で無視できない程度にはあり、RSA 仮定を破るには十分である。

第4章　鍵共有プロトコル

4.1　はじめに

鍵共有プロトコルとはセッション鍵を共有するためのプロトコルである。セッション鍵 k とは一時的な使い捨ての秘密の鍵情報を意味し、鍵共有プロトコルを実行した 2 者 P_1, P_2 の間でだけ共有されて、その後の P_1, P_2 の通信を秘匿したり (例えば k を鍵としてブロック暗号を用いる)、また認証したり (例えば k を鍵として認証子を用いる) するために使われる。

ここでは、(数多ある) 鍵共有プロトコルの中から、2 つの典型的なプロトコルを選んで紹介する。そのうちの 1 つは公開鍵暗号を用いる鍵共有法で、SSL にも用いられている。もう 1 つは DH 鍵共有法と呼ばれるもので IPSec に用いられている。

それら 2 つのプロトコルを縦糸にし、横糸として鍵共有プロトコルの安全性を考える。鍵共有プロトコルの安全性は、一見すると単純に見える。セッション鍵 k を守ればいいわけなので、「プロトコルを実行している P_1, P_2 のメッセージのやり取りを敵 A が覗き見しても、敵 A にはセッション鍵 k がわからない」こととしておけば十分に思える。

ところが、鍵共有プロトコルはそれ自身では意味を持たず、他のプロトコルやシステム（「親プロトコル」と呼ぶ）で部品として用いられて初めて意味をもつプロトコルである。しかも、その親プロトコルは、1 つだけでなく、同時に多数の鍵共有プロトコルを部品として実行することも珍しくない。そのため、鍵共有プロトコルは、同時に複数のコピーがいくらかの文脈を共有した状態で実行される。このような状態での実行を「コンカレントな」実行と呼ぶが、このコンカレントな実行において安全性を達成するのはなかなか困難なことである。鍵共有プロトコルがどのようにしてコンカレントな実行に対処しているのか見る。

4.2　鍵共有プロトコルとは?

鍵共有プロトコルとは「セッション鍵」を共有するための 2 者間プロトコルである。鍵共有プロトコルを実行する、2 つのパーティ P_1 と P_2 は、いくつかのメッセージを交換し、それら交換したメッセージをもとにし

て、それぞれ、あるビット列 k を計算して出力する。ここで、P_1 も P_2 も同じ k を出力できなければならない。この k をセッション鍵と呼ぶ。

このセッション鍵 k は一時的な使い捨ての、P_1 と P_2 の間だけの秘密情報を意図したもので、通常、その後の P_1 と P_2 間の通信を秘匿したり (例えば k を鍵としてブロック暗号を用いる)、また認証したりする (例えば k を鍵として認証子を用いる) ために使われる。

このようにセッション鍵 k は鍵共有プロトコルを実行した 2 者 P_1, P_2 だけが知るべき情報なので、鍵共有プロトコルに以下の安全性条件 SK を求めるのは自然である。

安全性条件 SK　鍵共有プロトコルを実行する 2 者 P_1, P_2 のやり取りするメッセージをすべて敵 A が覗き見しても、敵 A にはセッション鍵 k のどのような部分情報もわからない。

ここで、単に敵 A にはセッション鍵 k がわからないだけでなく、セッション鍵 k のどのような部分情報（例えば k の第 3 ビット目 k_3 が 1 であるとか、$k_1 \oplus k_7 = 0$ であるとか) もわからないことを要求していることに注意する。これが必要なのは、例えば、この k を鍵にしてブロック暗号でメッセージを暗号化して秘匿通信するような場合を考えると明らかである。ブロック暗号の安全性は鍵が完全に秘密であるときにしか保証されない。

4.3　2 つの鍵共有プロトコル：基本形

2 つの鍵共有プロトコル（の基本形）を紹介する。1 つは公開鍵暗号ベースの鍵共有プロトコル EB、もう 1 つは Diffie-Hellman 鍵共有プロトコル DH である。

4.3.1　公開鍵暗号ベースの鍵共有プロトコル EB

用いる公開鍵暗号を $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ とし (G:鍵生成、E:暗号化、D:復号アルゴリズム)、n をセキュリティパラメータ（安全性を指定するパラメータ）とする。P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得る。

1° P_2 は、G を実行して鍵ペア (e, d) を生成する: $(e, d) \leftarrow \mathsf{G}(n)$. 公開鍵 e を P_1 に送る。

2° P_1 は、ランダムな n ビットのビット列 k を生成し、それを鍵 e で暗号化して暗号文 c を得る：$k \xleftarrow{\$} \{0,1\}^n$,　$c \leftarrow \mathsf{E}_e(k)$. c を P_2 に送り、k を出力する。

3° P_2 は、c を復号して k を得る: $k \leftarrow \mathsf{D}_d(c)$. k を出力する。

プロトコルを実行するたびに鍵生成 $(e,d) \leftarrow \mathsf{G}(n)$ を実行するのは非効率なので、予め鍵を計算しておき、保存しておいた (同じ)e を送る（あるいはディレクトリに公開しておき、P_1 はそれから e を入手する）ことにしても構わない。

敵 A が P_1, P_2 間でやり取りされるメッセージ e と c を入手しても、公開鍵暗号 $(\mathsf{G},\mathsf{E},\mathsf{D})$ の安全性より (c に暗号化されている)k は分からない。ただし、安全性条件 SK を満たすためには、k のどのような部分情報も A にはわからない必要があるため、公開鍵暗号 $(\mathsf{G},\mathsf{E},\mathsf{D})$ は強秘匿 (第 1 章「公開鍵暗号」参照) である必要がある。

SSL では上の公開鍵暗号として RSA-PKCS#1 が使われている。

4.3.2　Diffie-Hellman 鍵共有プロトコル DH

p を十分大きな素数とし、$g(\in \{0,1,\ldots,p-1\})$ を、p を法とする位数が素数 q である整数 (すなわち、g は q 乗して初めて (p でわったあまりが)1 となる数) とする。P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得る。

1° P_1 は、$\{1,2,\ldots,q-1\}$ の範囲からランダムに x を選ぶ: $x \xleftarrow{\$} \{1,2,\ldots,q-1\}$. g の x 乗を計算し X とする: $X = g^x \bmod p$. X を P_2 に送る。

2° P_2 は、$\{1,2,\ldots,q-1\}$ の範囲からランダムに y を選ぶ: $y \xleftarrow{\$} \{1,2,\ldots,q-1\}$. g の y 乗を計算し Y とする: $Y = g^y \bmod p$. Y を P_1 に送る。

3° P_1 は、$k = Y^x \bmod p$ を計算し、k を出力する。

4° P_2 は、$k = X^y \bmod p$ を計算し、k を出力する。

ここで、まず、

$$Y^x \equiv g^{xy} \equiv X^y \bmod p$$

であることに注意する。これより、P_1 と P_2 は確かに同じ k を出力することが分かる。

安全性については、数論的な仮定が必要となる。

CDH 仮定 $\{1,2,\ldots,q\}$ の範囲からランダムに a,b を選ぶとき、どのような効率的なアルゴリズムも、$g^a \bmod p$ と $g^b \bmod p$ を与えられても $g^{ab} \bmod p$ は計算できない。

DDH 仮定 $\{1, 2, \ldots, q\}$ の範囲からランダムに a, b, c を選ぶとき、どのような効率的なアルゴリズムも、$g^a \bmod p$ と $g^b \bmod p$ を与えられて $g^{ab} \bmod p$ と $g^c \bmod p$ を区別（することすら）できない。

DDH 仮定のもとで、鍵共有プロトコル DH は安全性条件 SK を満たす。P_1, P_2 間でやり取りされるメッセージ $X = g^x \bmod p$,$Y = g^y \bmod p$ からセッション鍵 $k = g^{xy} \bmod p$ を求めることは、まさに CDH および DDH 仮定で問題とされていることと同じだからである。

4.4　コンカレントな環境における鍵共有プロトコルの安全性

ここで、鍵共有プロトコルの本来の目的に戻る。鍵共有プロトコルはそれ自身では意味がない。ただ k を共有しても使われないなら意味がない。鍵共有プロトコルは、他のプロトコルやシステム（「親プロトコル」と呼ぶ）で部品として用いられて初めて意味をもつプロトコルである。

しかも、その親プロトコルは、1 つだけでなく、同時に多数の鍵共有プロトコルを部品として実行することも珍しくない。そのため、鍵共有プロトコルは、同時に複数のコピーが (いくらかの文脈を共有した状態で) 実行されることになる。しかも、敵 A はただメッセージを覗き見るだけとも限らない。敵は、同時に実行されている数多くの鍵共有プロトコルの参加者としても潜んでいる可能性がある（正当な参加者として参加している場合もあれば、なりすましている場合もある)。さらに、敵 A はすでに終了した過去の鍵共有プロトコルに関しては、そのセッション鍵 k を入手できる場合もある（鍵共有プロトコルを利用するアプリケーションがセッション鍵をばらしてしまう場合など)。このような鍵共有プロトコルの（より現実に近い）実行環境をコンカレントな環境と呼ぶ。

コンカレントな環境にも耐えられる、より強力な安全性条件 CSK を設定する。

安全性条件 CSK　コンカレントな実行環境、すなわち、

(設定 1) 鍵共有プロトコルは、同時に複数のコピーが (いくらかの文脈を共有した状態で) 実行される、

(設定 2) しかも、敵 A 自身もそれらの任意の鍵共有プロトコルの実行に（正当な参加者としてあるいは誰かに成りすまして）参加できる、

(設定 3) さらに、敵 A はすでに終了した過去の任意の鍵共有プロトコルを選択し、そのセッション鍵を入手できる

環境において、

(条件 1) 鍵共有プロトコルの実行において、P_1 と P_2 さえ正しく正直に振舞えば、(陰日向で敵 A がどのように振舞おうとも) P_1 と P_2 の間で正しくセッション鍵が共有され、かつ

(条件 2) (敵 A の参加していない) 鍵共有プロトコル実行の全てのメッセージを敵 A が覗き見しても、(設定 3 の手段を用いてセッション鍵を入手しない限り) 敵 A にはセッション鍵 k のどのような部分情報もわからない。

4.4.1　プロトコル DH に対する中間者攻撃

4.3.2 節の DH 鍵共有プロトコル DH は安全性条件 CSK を満たさない。実際、コンカレント環境においては、以下のような中間者攻撃が成立する。

1° P_1 は正直に $X(= g^x)$ を計算し、P_2 になりすました攻撃者 M に送る。

2° M は $X'(= g^{x'})$ を自身で計算し、P_1 になりすまして P_2 に送る。

3° P_2 は正直に $Y(= g^y)$ を計算し、P_1 になりすました攻撃者 M に送る。

4° M は $Y'(= g^{y'})$ を自身で計算し、P_2 になりすまして P_1 に送る。

以上の実行は、P_1 と P_2 の視点からは全く正常に見えていることに注意する。ところが、P_1 は $k_1 = g^{xy'}$ を、P_2 は $k_2 = g^{x'y}$ を出力し、この k_1, k_2 は異なるので、安全性条件 CSK の条件 1 が破られている。さらに悪いことに M は上の k_1 も k_2 も知っている。

4.4.2　プロトコル EB に対する再送攻撃

4.3.1 節の公開鍵暗号ベースの鍵共有プロトコル EB は安全性条件 CSK を満たさない。実際、コンカレント環境においては、以下のような再送攻撃が成立する。ここでは、プロトコル EB において P_1 は予め P_2 の公開鍵 e を入手しているとする。(したがって、プロトコル EB で送受信されるメッセージは P_1 から P_2 へ送られる暗号文 c のみとなる。)

1° P_1 と P_2 は正直にプロトコル EB を実行する。このとき、P_1 から P_2 へ送られる暗号文 c を敵 R が入手する。

2° 敵 R は正当な送信者として P_2 との間でプロトコル EB を実行する。ただしこの際、上で入手した暗号文 c を P_2 へ送る。

3° 2° の R と P_2 によるプロトコル EB の実行に対し、R は設定 3 の手段を用いてそのセッション鍵 k を入手する。(k は C の復号文。)

4° 敵 R は 1° で P_1 と P_2 が得たセッション鍵は k であることを知る。

敵 R は覗き見したメッセージを自分自身のメッセージとして再送することで安全性条件 CSK の条件 2 を破ってしまった。

この攻撃において、P_2 は同じ暗号文 c を異なる送信者 P_1 と R から 2 度受信しているので、R からの攻撃に気づくのではないか、と思われるかもしれない。しかし、P_2 は人ではなく、単にアルゴリズムなので、予めコードとして書かれた手続きしか実行しない。したがって、受信した c が再送されたものかどうかのチェックはしないため、R からの攻撃を検知できない。

4.5　2 つの鍵共有プロトコル：発展形

ここでは、2 つの鍵共有プロトコル EB と DH をコンカレント環境に耐えられるよう強化して、鍵共有プロトコル EB-SIG と DH-SIG を得る [3]。強化の方針は、以下の通りである。

プロトコルの各メッセージにセッション識別子を含める。

プロトコル一つ一つの実行をセッションと呼ぶ。コンカレント環境では同時に複数のセッションが発生するので、それらの異なるセッションに属するメッセージが混じってしまわないよう、各メッセージにセッション識別子を含める。

プロトコルの各メッセージに送信者識別子を含める。

異なる送信者が生成したメッセージが混じってしまわないよう、各メッセージに送信者識別子を含める。

プロトコルの各メッセージの送信に認証メカニズムを組み込む。

折角、セッション識別子や送信者識別子をメッセージに含めても、それを敵に悪用されては意味がなくなる。そのようなことを防ぐため、各メッセージの送信にチャレンジレスポンスに基づく認証メカニズムを組み込む。具体的には、送信者 P_1 が受信者 P_2 にメッセージ m を送るところを、各パーティはデジタル署名スキームを利用できるものとして、以下のように変更する。まず受信者 P_2 の方から送信者 P_1 に乱数 N を送り、それに対して、送信者 P_1 がメッセージ m に m と N の連結文 (m, N) の署名文 s を添付して送るようにする。これによって、m は確かに今現在の P_1 が送ったことを受信者 P_2 は正しく確信することができる。

4.5.1　鍵共有プロトコル DH-SIG

各パーティはデジタル署名スキームが使用できるものとし、P_i のメッセージ m に対する署名文を $SIG_i(m)$ で表す。各パーティ P_i の識別子を (記号を節約して)P_i と書く。セッション識別子を s とする。セッション識別子 s は親プロトコルから与えられると考える。

P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得る。

1° P_1 は、$\{1, 2, \ldots, q-1\}$ の範囲からランダムに x を選ぶ: $x \stackrel{\$}{\leftarrow} \{1, 2, \ldots, q-1\}$. g の x 乗を計算し X とする: $X = g^x \bmod p$. (P_1, s, X) を P_2 に送る。

2° P_2 は、$\{1, 2, \ldots, q-1\}$ の範囲からランダムに y を選ぶ: $y \stackrel{\$}{\leftarrow} \{1, 2, \ldots, q-1\}$. g の y 乗を計算し Y とする: $Y = g^y \bmod p$. そして P_2 は、$(P_2, s, Y, SIG_2(P_2, s, Y, X, P_1))$ を P_2 に送る。

3° P_1 は、署名文 $SIG_2(P_2, s, Y, X, P_1)$ の正しさを確認する。(もし正しくなければプロトコル実行を拒否する。) そして P_1 は、$(P_1, s, SIG_1(P_1, s, X, Y, P_2))$ を P_2 に送ると同時に、$k = Y^x$ を計算し、(s, k) を出力する。

4° P_2 は、署名文 $SIG_1(P_1, s, X, Y, P_2)$ の正しさを確認する。(もし正しくなければプロトコル実行を拒否する。) P_2 は、$k = X^y$ を計算し、(s, k) を出力する。

鍵共有プロトコル DH-SIG には、プロトコル SIG に対して成立した中間者攻撃 (4.4.1 節) は通用しない。DH-SIG ではメッセージが (ランダムチャレンジに対応する) 署名文で守られているため、敵 M が P_1 や P_2 になりすましてメッセージを送ることができないためである。例えば署名文 $SIG_2(P_2, s, Y, X, P_1)$ には送信者と受信者の識別子が含まれていることに注意する。

4.5.2　鍵共有プロトコル EB-SIG

各パーティはデジタル署名スキームが使用できるものとし、P_i のメッセージ m に対する署名文を $SIG_i(m)$ で表す。各パーティ P_i の識別子を (記号を節約して)P_i と書く。セッション識別子を s とする。セッション識別子 s は親プロトコルから与えられると考える。さらに、$f_\kappa(\cdot)$ を擬似ランダム関数 (κ を知らないものにはランダム関数と区別がつかない関数) とする。

P_1 と P_2 は以下を実行して、それぞれセッション鍵 k を得る。(各パーティは予め必要な公開鍵と検証鍵を入手しているとする。)

1° P_2 は、ランダムな n ビットのビット列 N を生成し、(P_2, s, N) を P_1 に送る。

2° P_1 は、ランダムな n ビットのビット列 κ を生成し、それを鍵 e で暗号化して暗号文 c を得る：$k \xleftarrow{\$} \{0,1\}^n$,　$c \leftarrow \mathsf{E}_e(\kappa)$. そして P_1 は、$(P_1, s, c, SIG_1(P_1, s, c, N, P_2))$ を P_2 に送り、$k = f_\kappa(P_1, P_2, s)$ を出力する。

3° P_2 は、署名文 $SIG_1(P_1, s, c, N, P_2)$ の正しさを確認する。(もし正しくなければプロトコル実行を拒否する。) c を復号して κ を得る: $\kappa \leftarrow \mathsf{D}_d(c)$. $k = f_\kappa(P_1, P_2, s)$ を出力する。

鍵共有プロトコル EB-SIG には、プロトコル EB に対して成立した再送攻撃 (4.4.2 節) は通用しない。敵 R が先と全く同じようにメッセージ (P_1, s, c, SIG_1) をまるごと再送しても、新しいランダムチャレンジ N に対応していないので署名がおかしくなる。敵 R が c だけ再利用し、署名文は新しいランダムチャレンジ N に対応して自身で再計算して添付しても、セッション鍵 $k = f_\kappa(P_1, P_2, s)$ は、新しいセッション識別子 s(その他) にリンクしているので、元の値とは違うランダムな値になる。

4.6　おわりに

コンカレント環境における鍵共有プロトコルのフォーマルな安全性定義がいくつか提案されている。そのもとで鍵共有プロトコル EB-SIG、DH-SIG はその安全性が適切な仮定のもとで証明されている [3]。

ただし、鍵共有プロトコルは、理論と実際が最も鋭く対応するプロトコルといえると思う。そのフォーマルな安全性定義についても、どこまでモデルに現実的な状況を取り入れられているか、また、理論的になりすぎて安全性要件として不自然に厳しくなっていないか、などについて議論が多くある。

関連図書

[1] M. Blum and S. Goldwasser, An efficient probabilistic public-key encryption scheme that hides all partial information, pp. 289-302, Crypto '84, LNCS 196.

[2] M. Bellare and P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols. Proceedings of the First Annual Conference on Computer and Commmunications Security, ACM, 1993.

[3] R. Canetti and H. Krawczyk, Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels, Eurocrypt 01, 2001.