

平成21年度

論文題目

n チャネルメッセージ伝送方式による
暗号化通信

神奈川大学 工学部 電子情報フロンティア学科

学籍番号 200602825

栗山 知也

指導担当者 木下 宏揚 教授

目次

第1章	序論	5
1.1	背景	5
1.2	本研究の目的	7
第2章	数学的基礎	8
2.1	合同式	8
2.2	合同式の除法	9
2.2.1	一次合同式	9
2.2.2	逆元	10
2.3	ユークリッドの互除法	11
2.4	ラグランジェの補間公式	14
第3章	暗号理論の基礎	15
3.1	公開鍵暗号	15
3.2	秘密分散共有法	17
3.3	ハッシュ関数	18
第4章	nチャンネルメッセージ伝送方式	20
4.1	PSMT(Perfectly Secure Message Transmission)	21
4.1.1	PSMTにおける安全性の定義	21
4.1.2	PSMTの歴史	22
4.2	ASMT(Almost Secure Message Transmission)	23
4.2.1	ASMTにおける安全性の定義	23

4.2.2	ASMTの歴史	23
4.2.3	Basicプロトコル	24
4.2.4	Basicプロトコルの通信量	24
第5章	提案プロトコル	26
5.1	改良プロトコル	26
5.1.1	改良プロトコルの計算量	27
5.1.2	改良プロトコルの通信量	28
5.2	プログラム	29
5.2.1	作成したプログラムの構造	29
5.2.2	各オブジェクトの構造	30
5.3	ネットワーク上での実装	36
5.3.1	仮想ネットワークの利点	37
5.3.2	実装上の問題	37
5.3.3	問題の解決案	37
5.3.4	実装計画	38
第6章	結論	40
	謝辞	41
	参考文献	42
	質疑応答	44

目次

1.1	従来の公開鍵暗号方式	5
1.2	n チャンネルメッセージ伝送方式	5
3.1	秘密分散共有法	17
4.1	n チャンネルメッセージ伝送方式	20
4.2	1-round 方式と 2-round 方式	21
5.1	内部構造のイメージ	30
5.2	仮想ネットワークのイメージ	39
5.3	実際のネットワークのイメージ	39

表 目 次

4.1 PSMTの歴史	22
-----------------------	----

第1章

序論

1.1 背景

従来の公開鍵暗号方式では公開鍵の正当性を証明するために認証局のような信頼できる第三者機関が必要であった(図1.1)。それに対して n チャネルメッセージ伝送方式では事前の鍵が不要なため第三者機関も必要ない。そこで本論文では n チャネルメッセージ伝送方式に着目している。 n チャネルメッセージ伝送方式とは n 本の通信路を使用して安全に送信する暗号化通信方式である。もし n 本のうちの何本かに文書を盗聴・改ざんする敵が潜んでいても、残りの通信路の情報を用いて文書を復号することができる(図1.2)。

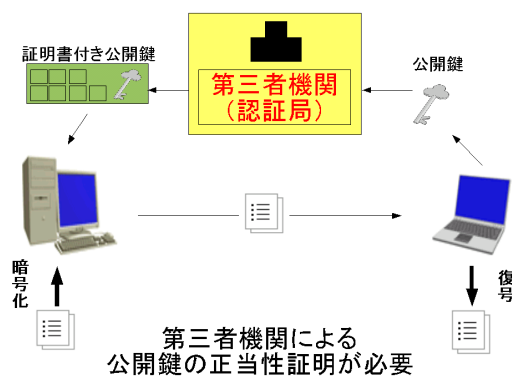


図 1.1 従来の公開鍵暗号方式

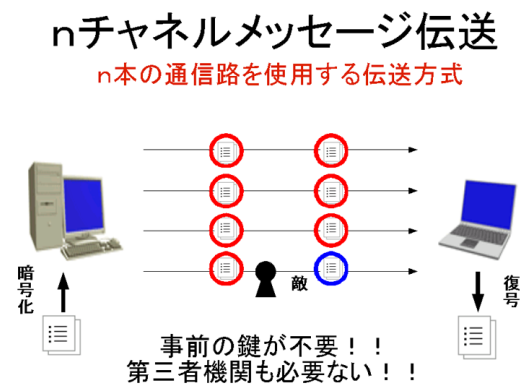


図 1.2 n チャネルメッセージ伝送方式

n チャンネルメッセージ伝送方式には PSMT (Perfectly Secure Message Transmission) という方式がある。PSMT の安全性の定義は以下のとおりである。

1. 敵は送信メッセージに関する情報を何も得られない。(盗聴耐性)
2. 受信者がメッセージを正しく受信できる確率が 100 % である。(改竄耐性)

また、送信者が受信者に 1 回送信するだけで済む方式を 1-round 方式、送信者と受信者が相互に r 回やり取りを行う方式を r -round 方式と呼ぶ。

PSMT は 1993 年に Dolev ら [1] によって提案された。彼らは、敵が n 本の通信路のうち t 本に潜んでいるとしたときに PSMT プロトコルが存在するための必要十分条件は、1-round 方式では $n \geq 3t + 1$ 、2-round 方式では $n \geq 2t + 1$ であることを証明した。PSMT においては、 $n \geq 3t + 1$ でなければ使えない 1-round 方式よりも、 $n \geq 2t + 1$ で使える 2-round 方式のほうが優れている。そこで、1-round 方式における必要十分条件を $n \geq 2t + 1$ に改善するために生まれたのが ASMT (Almost Secure Message Transmission) である。

ASMT の安全性の定義は以下のとおりである。

1. 敵は送信メッセージに関する情報を何も得られない。(盗聴耐性)
2. 受信者がメッセージを正しく受信できる確率が $1 - \delta$ 以上である。(改竄耐性)
3. 受信者が正しく受信できない確率が δ であり、そのとき受信者は failure を出力できる。(失敗検知能力)
4. 敵が t 本の通信路を遮断しても受信者は残りの通信路で得た情報だけからメッセージを受信できる。(遮断耐性)

ASMT は 2004 年に Srinathan ら [2] によって提案されたが、そのプロトコルには間違いがあった。その後 2007 年に Kurosawa ら [3] によって厳密に定義された。そのなかで $n = 2t + 1$ のときの通信効率の限界が示され、限界に近い通信量で通信できるプロトコルが提案された。上記のプロトコルは計算量が指数関数的であるという問題があった。その問題点を改善し、計算量を多項式時間にした Basic プロトコルを 2008 年に木下研究室が提案した。

1.2 本研究の目的

本論文ではBasicプロトコルの通信量を改善した改良プロトコルを提案する。それに加えて改良プロトコルをFlash(ActionScript)で実装し、正しく動作することを検証する。また、提案したプロトコルをネットワーク上で動かすことを最終的な目標とする。

第2章

数学的基礎

2.1 合同式

整数環 Z において、整数 m と n を整数 h で割った余りが等しいとき

$$m \equiv n \pmod{h}$$

と記して、 m と n とを同一視することを h を法 (modulo h 、略して \pmod{h}) とし
て合同という。具体例として $h = 5$ である場合、例えば $12 \equiv 7 \equiv 2 \pmod{5}$ であ
るから 12、7、2 は 5 を法として同一視される。すなわち合同式を用いると、無
数に存在する整数をある整数で割り、同じ余りを有するものどうしを同様の
性質をもつ数として分類し、取り扱うことができるのである。

また合同式では整数 a, b, c, d 、自然数 m に対して以下のことが成り立つ。

1. $a \equiv a \pmod{m}$ (反射律)
2. $a \equiv b \Rightarrow b \equiv a \pmod{m}$ (対称律)
3. $a \equiv b, b \equiv c \Rightarrow a \equiv c \pmod{m}$ (推移律)
4. $a \equiv b, c \equiv d \pmod{m} \Rightarrow a + c \equiv b + d \pmod{m}$
 $a - c \equiv b - d \pmod{m}$
 $ac \equiv bd \pmod{m}$

5. もし $ac \equiv bc \pmod{m}$ かつ $\gcd(c, m) = 1$ ならば $a \equiv b \pmod{m}$
合同式については [4]、[5] を参照。

2.2 合同式の除法

合同式は加法・減法・乗法・については良い性質を持ち、扱いも簡単である。しかし除法についてはそうではない。そもそも合同式は整数の範囲での記法である。整数を整数で割った場合、整数にならないこともあるので、合同式で除法を考えると注意が必要である。例えば、 $ab \equiv ac \pmod{m}$ は両辺 a で割って $b \equiv c \pmod{m}$ とすることができるとは限りません。実際 $3 \times 5 \equiv 15 \equiv 9 \equiv 3 \times 3 \pmod{6}$ ですが、 $5 \equiv 3 \pmod{6}$ ではない。しかし除法を注意深く、上手に解釈すると除算ができる場合もある。そのために、合同式の一次方程式(一次合同式)を考えてみる。

2.2.1 一次合同式

a, b を整数とするとき、 $ax \equiv b \pmod{m}$ となる x を求めることを一次合同式を解くという。合同式の解は m を法として決まるため x が解で $x \equiv x' \pmod{m}$ ならば、 x' も解になる。一次合同式は解を持たないこともある。例えば $3x \equiv 1 \pmod{6}$ は解を持たない。実際この場合 x に解があったとすると、合同式の定義から、 $3x - 1$ が 6 で割り切れることになり矛盾となる。解の存在については、次のことが成立する。

- 一次合同式 $ax \equiv b \pmod{m}$ が解 x を持つための必要十分条件は a と m の最大公約数 $\gcd(a, m)$ が b を割り切ることである。

[証明]

(必要性)

x を $ax \equiv b \pmod{m}$ の解とする。このとき、 $ax - b$ は m で割り切れるので、その商を k とすると、 $ax - b = km$ と表される。 b と km を移項すると、 $ax - km = b$ となり、左辺は $\gcd(a, m)$ で割り切れる。したがって、 $\gcd(a, m)$ は右辺 b を割り切る。

(十分性)

$\gcd(a, m)$ は右辺 b を割り切るとし、その商を s とする。このとき $b = s \times \gcd(a, m)$ である。ここで後述する拡張ユークリッドの互除法の結果を使うと、

$$ax' + my' = \gcd(a, m)$$

となる x' と y' が存在する。この両辺に s を掛けると

$$asx' + msy' = s \times \gcd(a, m) = b$$

となる。ここで $x = s \times x'$ と置くと、 $ax - b = -msy'$ であるから、 $ax \equiv b \pmod{m}$ となる。すなわち、 x が解になる。

この証明から解 x は拡張ユークリッドの互除法から計算できることが分かる。上の性質で $b = 1$ の場合が特に重要である。この場合 $\gcd(a, m)$ が 1 を割り切るとは $\gcd(a, m) = 1$ に他ならない。よって次のことが成立する。

- 一次合同式 $ax \equiv 1 \pmod{m}$ が解 x を持つための必要十分条件は a と m の最大公約数 $\gcd(a, m) = 1$ となることである。

この場合、解 x は m を法にして唯一つしかないことも分かる。すなわち、 x と x' が共に $ax \equiv 1 \pmod{m}$ の解だとすると、 $x' \equiv x \pmod{m}$ となる。実際 $ax \equiv 1 \pmod{m}$ 、 $ax' \equiv 1 \pmod{m}$ とすると、 $axx' \equiv x' \pmod{m}$ となる。したがって、

$$x' \equiv axx' \equiv ax'x \equiv 1 \times x \equiv x \pmod{m}$$

となる。このことから m を法とする逆元を定義することができる。

2.2.2 逆元

$\gcd(a, m) = 1$ となるとき、一次合同式 $ax \equiv 1 \pmod{m}$ の解 x が m を法にして唯一つ存在する。その x を m を法とする a の逆元という。 m を法とする a の逆

元は $\gcd(a, m) = 1$ となる a に対してのみ定義される。また m を法とする a の逆元は整数としては唯一つではないが、 m を法にした場合は唯一つである。この m を法とする a の逆元 x は整数であるから、他の数に掛けることができる。ここで $ax \equiv 1 \pmod{m}$ であるから、 x を掛けることは m を法として考えている限りは a で割ることを意味する。したがって合同式で除法を行う場合は、次のように逆元を用いて行う。

$$a \div b \equiv a \times (b \text{ の逆元}) \pmod{m}$$

合同式の除法については [6] を参照。

2.3 ユークリッドの互除法

ユークリッドの互除法は、2つの自然数の最大公約数を導き出すアルゴリズムである。素因数分解に比べて効率よく計算できる。互除法で2つの自然数 $a, b (a > b)$ の最大公約数を見つけるには、次の手続きを用いる。

1. a を b で割り、余り r とする。
2. $r = 0$ の場合は、最大公約数は b であり、手続きは終わりになる。
3. $r \neq 0$ の場合は、 a と b の組を b と r に置き換えて、最初の手続きにもどる。

つまり、この1から3の手続きを繰り返して、余りが0になったときに割った数が、最大公約数となるわけである。言い換えれば、余り0を得たときの直前のステップで得た余りが、最大公約数ということになる。

例として、1365と77の最大公約数をユークリッドの互除法で求めてみる。

$$1365 = 17 \times 77 + 56 \quad (\leftarrow 1365 \div 77 = 17 \text{ 余り } 56 \text{ の計算による})$$

$$77 = 1 \times 56 + 21 \quad (\leftarrow 77 \div 56 = 1 \text{ 余り } 21 \text{ の計算による})$$

$$56 = 2 \times 21 + 14 \quad (\leftarrow 56 \div 21 = 2 \text{ 余り } 14 \text{ の計算による})$$

$$21 = 1 \times 14 + \underline{7} \quad (\leftarrow 21 \div 14 = 1 \text{ 余り } 7 \text{ の計算による})$$

$$14 = 2 \times \underline{7} + 0 \quad (\leftarrow 14 \div 7 = 2 \text{ 余り } 0 \text{ の計算による})$$

となり、最大公約数は7となる。

次に互いに素な20と17で、互除法を用いて最大公約数を求めてみる。

$$20 = 1 \times 17 + 3 \tag{2.1}$$

$$17 = 5 \times 3 + 2 \tag{2.2}$$

$$3 = 1 \times 2 + 1 \tag{2.3}$$

$$2 = 2 \times 1 + 0$$

最大公約数は、当然ながら1なので、互除法を使う必要がないように感じられるが、その結果を求める過程の式に大きな利用価値がある。まず式(2.1)、(2.2)、(2.3)を移項して、次の3つの式を得る。

$$20 - 1 \times 17 = 3 \tag{2.4}$$

$$17 - 5 \times 3 = \underline{2} \tag{2.5}$$

$$3 - 1 \times \underline{2} = 1 \tag{2.6}$$

次に式(2.6)の2に式(2.5)を代入して、3と17に注目してくくる。

$$3 - 1 \times \underline{2} = 3 - 1 \times (17 - 5 \times 3) = 6 \times \underline{3} - 1 \times 17 = 1 \tag{2.7}$$

さらに、式(2.7)の3に式(2.4)を代入し、20と17に注目してくくる。

$$6 \times \underline{3} - 1 \times 17 = 6 \times (20 - 1 \times 17) - 1 \times 17 = 6 \times 20 - 7 \times 17 = 1$$

この一連の手続きから得た結果を、次のように書き換える。

$$20 \times 6 + 17 \times (-7) = 1$$

上の式は、 $ax + by = c$ という形になっていて、 a 、 b 、 c 、 x 、 y にあたる数はすべて整数である。このような形の方程式は一次不定方程式といい、整数解の x

と y を求めるものである。つまり、ユークリッド互除法の計算過程を利用することで、 $a = 20$ 、 $b = 17$ のとき、一次不定方程式の整数解 $(x, y) = (6, -7)$ が得られるということが示されている。この方法は拡張ユークリッドの互除法と呼ばれ、非常に利用価値の高いアルゴリズムである。一般に a と b を 0 でない整数とし、 a と b の最大公約数を c とすると、一次不定方程式

$$ax + by = c$$

は、整数解 (x_1, y_1) を持ち、解の 1 組は、拡張ユークリッドの互除法を用いて求めることができる。ただし、一次不定方程式の解は、1 組だけではない。方程式すべての整数解は、任意の整数 k を用いて次のように表される。

$$(x, y) = \left(x_1 + k \cdot \frac{b}{c}, y_1 - k \cdot \frac{a}{c} \right) \quad (2.8)$$

式 (2.8) に示す解の公式を用いれば、一次不定方程式 $20x + 17y = 1$ のすべての整数解は、次のようになる。

$$(6 + 17k, -7 - 20k) \quad (2.9)$$

$k = -1$ の場合、解は $(x, y) = (-11, 13)$ である。これを一次不定方程式 $20x + 17y = 1$ に代入する。

$$20 \times (-11) + 17 \times 13 = 1$$

移項して、式を整える。

$$17 \times 13 = 1 + 11 \times 20 \quad (2.10)$$

式 (2.10) をよく見ると、実は、次式と同じ意味であることがわかる。

$$17 \times 13 \equiv 1 \pmod{20} \quad (2.11)$$

2.2.2 で、 $ax \equiv 1 \pmod{m}$ の場合、「 x は m を法とする a の逆元である」と説明した。すなわち、式 (2.11) は 20 を法にして 13 が 17 の乗算に対する逆元であることを意味する。つまり、拡張ユークリッドの互除法を使えば、合同式での逆元が効率よく導き出すことができるのである [7]。

2.4 ラグランジェの補間公式

xy 座標上に k 個の点 $(i, f(i))$ が与えられたとき、それら全てを通る $k - 1$ 次関数 $f(x)$ は唯一に定まる。この $f(x)$ を求める方法としてラグランジェの補間公式と呼ばれる次の公式がある。

$$f(x) = \lambda_1(x)f(i_1) + \cdots + \lambda_k(x)f(i_k)$$

ただし

$$\lambda_j(x) = \frac{(x - i_1) \cdots (x - i_{j-1})(x - i_{j+1}) \cdots (x - i_k)}{(i_j - i_1) \cdots (i_j - i_{j-1})(i_j - i_{j+1}) \cdots (i_j - i_k)}$$

例えば、二次関数 $f(x) = ax^2 + bx + c$ 上の点を $(x_1, f(x_1))$ 、 $(x_2, f(x_2))$ 、 $(x_3, f(x_3))$ として、ラグランジェの補間公式を用いると $f(x)$ は以下のように表せる。

$$f(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}f(x_3)$$

ラグランジェの補間公式については [8] を参照。

第3章

暗号理論の基礎

3.1 公開鍵暗号

公開鍵暗号は、その概念が1976年にディフィーとヘルマンによって提案された。具体的な公開鍵暗号化方式の最初のものは、1978年にリベスト (R.L.Rivest)、シャミア (A.Shamir)、エードルマン (L.Adleman) により提案され、RSA暗号と呼ばれた。その後、ElGamal暗号、楕円曲線暗号など多くの公開鍵暗号が発明されたが、現在でもRSA暗号が最も広く用いられている。

公開鍵暗号を用いる場合には、各ユーザは自分の固有の秘密鍵と公開鍵の対を生成する。いま、AがBに暗号化通信をしたい場合には、Bはまず、自分の公開鍵をAに伝える。Aはこの公開鍵を用いてメッセージを暗号化し、Bに送る。Bは自分の秘密鍵を用いて、Aから送られた暗号文を復号できる。この方式の利点は秘密鍵がユーザごとに一つだけであるから、その管理が非常に楽になることである。しかし、この公開鍵暗号には大きな問題が二つある。

一つは計算量の問題である。暗号化鍵と復号鍵の非対称性を実現するために、公開鍵暗号の暗号化、復号にはかなりの計算量を要することになる。このため、公開鍵暗号は大量の情報の高速な守秘伝送には向いていない。もう一つは公開鍵の管理の問題である。暗号化の場合でもデジタル署名の場合でも、A

やBの公開鍵が間違いなく本人のものであることが保証されないかぎり、安全な通信や認証が行えない。このような保証を与える仕組みの一つが公開鍵認証基盤 (PKI : public key infrastructure) である。これは認証機関 (CA : certification authority) が各ユーザに発行する公開鍵証明書によって、本人の公開鍵であることの認証を行う方式であり、電子政府の基盤となっている方式である [9]。最後に公開鍵暗号系の中でも重要なものをいくつか簡単に紹介する [10]。

RSA

RSAの安全性は大きな整数の因数分解の難しさに基づいている。

Merkle-Hellman ナップサック

これに関係したシステムは部分集合和問題の難しさに基づいている。しかしながら、様々なナップサック系の全てが危険であることが明らかにされている。(Chor-Rivest 暗号系を除く)

McEliece

McEliece 暗号系は、代数的符号理論に基づいており、いまだ安全であると信じられている。この理論は線形符号の復号問題に基づいている。

ElGamal

ElGamal 暗号系は有限体上の離散対数問題の難しさに基づいている。

Chor-Rivest

これも“ナップサック”型暗号系の一種であるが、まだ安全だとみなされている。

3.2 秘密分散共有法

秘密鍵を紛失から守るためには、そのコピーを作って複数の場所に保管しておくことが望ましい。しかし、コピーの数を多くすると盗難の危険が増大してしまう。一方、コピーの数を少なくすると、すべてを紛失してしまう危険が増大する。この相矛盾する二つの問題を解決する方法が、秘密分散共有法である。

秘密分散共有法は、秘密 s の保有者(ディーラ)と、複数の分散管理者の間で行われる。以降、ディーラを D 、 n 人の分散管理者を P_1, \dots, P_n で表す。秘密分散共有法は図 3.1 のように分散段階と再構成段階から構成される。

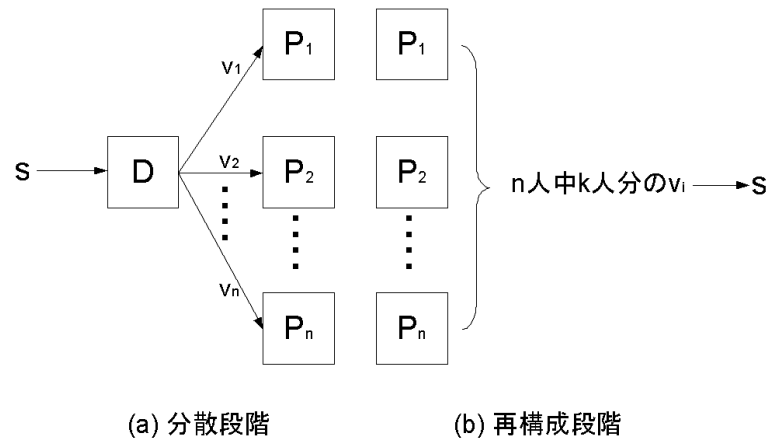


図 3.1 秘密分散共有法

分散段階

図(a)のように、 D は、秘密 s から、 n 個のシェアと呼ばれる v_1, \dots, v_n を計算し、 v_i を P_i に与える。

再構成段階

図(b)のように、 n 人の分散管理者のうち何人かが集まり、シェアから s を求める。

(k, n) しきい値秘密分散共有法 (しきい値法) では、 v_1, \dots, v_n は次の条件を満たすように作成されていなければならない。

(条件 1) v_1, \dots, v_n のうち、任意の k 個から元の秘密 s を復元できる。

(条件 2) どの $k - 1$ 個を集めても、 s について何も分からない。

秘密分散共有法について詳しくは [8] を参照。

3.3 ハッシュ関数

長いメッセージを短い k ビットに圧縮する関数をハッシュ関数という。ハッシュ関数は、デジタル署名など、暗号の多くの分野で利用される。

衝突困難なハッシュ関数

ハッシュ関数 H においては、長いメッセージを短くするので、必ず

$$H(x) = H(x')$$

となる衝突ペア (x, x') が存在する。そのような衝突ペア (x, x') を効率よく見つけるのが困難なハッシュ関数を衝突困難なハッシュ関数と呼ぶ。上記のことも含めてハッシュ関数には以下のような安全性が定義されている [10]。

Collision Resistance(衝突困難性)

ハッシュ関数 H は、 $x' \neq x$ かつ $H(x') = H(x)$ であるような x と x' を見つけることが計算量的に困難である。

Preimage Resistance(一方向性)

$H(x) = z$ である x を見つけることが計算量的に不可能である。また、このようなハッシュ関数 H を一方向性であるという。

Second Preimage Resistance

x に対して $H(x') = H(x)$ であるような x とは別の x' を見つけることが計算量的に困難である。

固定長圧縮関数

k ビットより長い一定の長さのメッセージを、 k ビットに圧縮する関数を固定長圧縮関数と呼ぶ。実際のハッシュ関数の設計においては、長いメッセージに対して固定長圧縮関数を繰り返し適用することにより、可変長の入力に対するハッシュ値を求める、という方針がとられている。

以下に代表的なハッシュ関数について簡単に説明する。

MD4

衝突を有するが、任意の長さの平文を 128 ビットに圧縮する方法で、32 ビットを単位とした演算をする。非常に高速である。

MD5

MD4 をさらに高速にしたものである。

SHA-1

米国政府のシステム調達基準である FIPS 180-1 の中で規定されている代表的なハッシュ関数である。 2^{64} ビット未満の任意の長さのメッセージを 160 ビットに圧縮する。

ハッシュ関数については [8]、[10]、[11] を参照。

第4章

nチャネルメッセージ伝送方式

従来の公開鍵暗号方式では公開鍵の正当性を証明するために認証局のような信頼できる第三者機関が必要であった。それに対してnチャネルメッセージ伝送方式では事前の鍵が不要なため第三者機関も必要ない。そこで本論文ではnチャネルメッセージ伝送方式に着目している。

nチャネルメッセージ伝送方式は文書をn本の通信路を使用して安全に送信する暗号化通信方式である。もしn本のうちの何本かに文書を盗聴・改竄する敵が潜んでいても、残りの通信路の情報を用いて文書を復号することができる(図4.1)。nチャネルメッセージ伝送方式にはPSMTとASMTという方式がある。

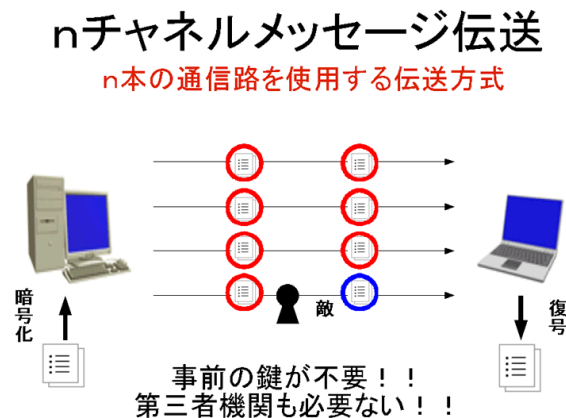


図 4.1 nチャネルメッセージ伝送方式

4.1 PSMT(Perfectly Secure Message Transmission)

4.1.1 PSMTにおける安全性の定義

n チャンネルメッセージ伝送方式において次の2つの条件を満たしたものをPSMTと呼ぶ。

1. 敵は送信メッセージに関する情報を何も得られない。(盗聴耐性)
2. 受信者がメッセージを正しく受信できる確率が100%である。(改竄耐性)

また、送信者が受信者に1回送信するだけで済む方式を1-round方式、送信者と受信者が相互に r 回やり取りを行う方式を r -round方式と呼ぶ(図4.2)。

このとき敵が n 本の通信路のうち t 本に潜んでいるとしたときにPSMTプロトコルが存在するための必要十分条件は、1-round方式では $n \geq 3t + 1$ 、2-round方式では $n \geq 2t + 1$ であることが証明されている。

1-round



2-round

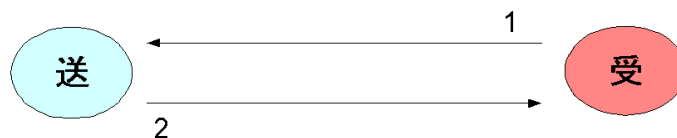


図 4.2 1-round方式と2-round方式

4.1.2 PSMTの歴史

PSMTは1993年にDolevら[1]によって提案された。彼らは、敵が n 本の通信路のうち t 本に潜んでいるとしたときにPSMTプロトコルが存在するための必要十分条件は、1-round方式では $n \geq 3t + 1$ 、2-round方式では $n \geq 2t + 1$ であることが証明し、また、それぞれの通信量が $O(n)$ 、 $O(2^n)$ のプロトコルを提案した。その後2-round方式は1996年にSayeedら[12]が通信量 $O(n^3)$ のプロトコルを提案し、2006年にはAgarwalら[13]が大量の文書を送ることを前提に通信量 $O(n)$ で計算量が指数関数的であるプロトコルを提案した。そして2008年にはKurosawaら[14]が通信量 $O(n)$ 、計算量 $O(n^3)$ となるプロトコルを提案した。このように2-round方式の通信量、計算量は改善されていった。一方、1-round方式は、通信量 $O(n)$ 、計算量が多項式時間となるプロトコルをDolevらが最初に提案しており、既にそれが $n \geq 3t + 1$ における最善のプロトコルであった。PSMTにおいては、 $n \geq 3t + 1$ でなければ使えない1-round方式よりも、 $n \geq 2t + 1$ で使える2-round方式のほうが優れている。そこで、1-round方式における必要十分条件を $n \geq 2t + 1$ に改善するために生まれたのが次に述べるASMTである。

	PSMT	
	2-round	1-round
Dolev 1993	$n \geq 2t + 1$ が必要十分条件 通信量： $O(2^n)$	$n \geq 3t + 1$ が必要十分条件 通信量： $O(n)$
Sayeed 1996	通信量： $O(n^3)$	
Agarwal 2006	通信量： $O(n)$ ただし大量の文書を送ることが前提 計算量：指数的	
Kurosawa 2008	通信量： $O(n)$ 計算量： $O(n^3)$	

表 4.1 PSMTの歴史

4.2 ASMT(Almost Secure Message Transmission)

4.2.1 ASMTにおける安全性の定義

ASMTにおける安全性の定義は以下のとおりである。

1. 敵は送信メッセージに関する情報を何も得られない。(盗聴耐性)
2. 受信者がメッセージを正しく受信できる確率が $1 - \delta$ 以上である。(改竄耐性)
3. 受信者が正しく受信できない確率が δ 以下であり、そのとき受信者は failure を出力できる。(失敗検知能力)
4. 敵が t 本の通信路を遮断しても受信者は残りの通信路で得た情報だけからメッセージを受信できる。(遮断耐性)

PSMTと比較して主に異なる点は定義2においてメッセージを正しく受信できる確率が $1 - \delta$ となっており、失敗した時はそれを検知できるという定義3が加わっている点である。

4.2.2 ASMTの歴史

ASMTは2004年にSrinathanら[2]によって提案されたが、そのプロトコルには間違いがあった。その後2007年にKurosawaら[3]によって厳密に定義された。そのなかで $n = 2t + 1$ での通信効率の限界が以下のように示された。

$$|X_i| \leq (|S| - 1)/\delta + 1 \quad (4.1)$$

X_i : channel(i) を流れる情報の集合

S : 秘密情報の集合

δ : 失敗確率

また、Kurosawaらは通信効率の限界に近い通信量で通信できるプロトコルも提案した。そのプロトコルの通信効率は失敗確率を ϵ とすると、

$$|X_i| = \frac{|S| - 1}{\delta} + 1 > \frac{|S| - 1}{\epsilon} + 1$$

$$\text{ただし } \epsilon = \left\{ \binom{n}{t+1} - \binom{n-t}{t+1} \right\} \delta$$

となっている。

4.2.3 Basic プロトコル

Kurosawaらが提案したプロトコルは計算量が指数関数的であるという問題があった。木下研究室がこれを改善して多項式時間となるBasicプロトコルを提案した。

4.2.4 Basic プロトコルの通信量

Basicプロトコルの特徴はハッシュ関数 H を用いる点であった。敵のSecond Preimage Attackが成功したときがこのプロトコルの失敗となる。敵は t 個のハッシュ値にSecond Preimage Attackをするので、このプロトコルの失敗確率 ϵ は、

$$\epsilon = [\text{ハッシュ関数 } H \text{ への } \textit{SecondPreimageAttack} \text{ が成功する確率}] \times t \quad (4.2)$$

ここで H の出力値のビット数を h とすると、出力値は 2^h 通りとなり、 H がランダムオラクル(ランダムに値を出力する)と仮定すると H へのSecond Preimage Attackが成功する確率は $1/2^h$ となる。したがって式(4.2)は

$$\epsilon = \frac{t}{2^h} \quad (4.3)$$

となる。さらに秘密 s の長さを q ビットとおくと秘密の集合 S との関係は

$$|S| = 2^q \quad (4.4)$$

通信効率の限界式 (式 (4.1)) と式 (4.3)、式 (4.4) より

$$|X_i| \frac{|S| - 1}{\epsilon} + 1 = \frac{2^q - 1}{t/2^h} + 1 = \frac{2^h(2^q - 1)}{t} + 1 \quad (4.5)$$

となるので、Basic プロトコルの実際の $|X_i|$ が式 (4.5) の右辺に近ければ通信効率が限界に近いと言える。次にビット数で評価するために両辺の \log_2 をとる。

$$\begin{aligned} \log_2 |X_i| &= \log_2 \left\{ \frac{2^h(2^q - 1)}{t} + 1 \right\} \approx \log_2 \left\{ \frac{2^h(2^q - 1)}{t} \right\} \\ &= \log_2 2^h + \log_2(2^q - 1) - \log_2 t \approx h + q - \log_2 t \end{aligned} \quad (4.6)$$

一方、このプロトコルの X_i とはチャンネル ch-i を使って送信者が送る $f(i)$ と $H(f(1)) \sim H(f(n))$ である。秘密 s の長さを q ビットとしているので $f(i)$ も q ビットであり、ハッシュ値は h ビットであるので $|X_i|$ は $q + hn$ ビットとなる。よって式 (5.5) は

$$q + hn = \log_2 |X_i| \approx h + q - \log_2 t$$

となる。両辺を比較すると左辺の hn と右辺の h との差が大きいことがわかる。したがって Basic プロトコルは通信効率の限界に近いとは言えない。そこで本研究では Basic プロトコルの通信効率を改善したプロトコルを提案する。

第5章

提案プロトコル

この章では Basic プロトコルの通信効率を改善した改良プロトコルを提案する。

5.1 改良プロトコル

改良プロトコルでは Basic プロトコルと違い、1度に m 個の s_i を送ることで通信効率を改善している。改良プロトコルの手順は以下のとおりである。

但し、 H はハッシュ値であり、安全なハッシュ関数を用いるとする。

$n = 2t + 1$ (n : 通信路の数、 t : 敵の数)、送信する秘密情報を s 、 P を大きな素数とする。

送信者

1. $f_1(x) \sim f_m(x)$ をランダムで決める。 ($f_i(x) = s_i + a_{i1}x + a_{i2}x^2 + \dots + a_{it}x^t$)
2. $\mathbf{F}_1 = f_1(1) \| f_2(1) \| f_3(1) \| \dots \| f_m(1)$
 $\mathbf{F}_2 = f_1(2) \| f_2(2) \| f_3(2) \| \dots \| f_m(2)$
 \vdots
 $\mathbf{F}_n = f_1(n) \| f_2(n) \| f_3(n) \| \dots \| f_m(n)$ とおく。

3. ハッシュ値 $H(F_1) \sim H(F_n)$ を計算する。
4. 各チャンネル ch-i に F_i 、 $H(F_1) \sim H(F_n)$ を送る。

敵

- $F_1 \sim F_n$ のうち、 t 個しか知らない。
→ F_i 中の $f_i(x)$ は t 次関数なので t 点からは s について何も分からない。
- ここでハッシュ関数 H は一方向性があると仮定するので $H(m)$ から m を逆算できない。
→ ハッシュ値からは s について何も分からない。

受信者

1. $F'_1 \sim F'_n$ を得る。
2. n 本の通信路のうち半分以上の $t+1$ 本は正しい情報であることを利用し、多数決で正しい $H(F_1) \sim H(F_n)$ を得る。
3. $H(F'_1) \sim H(F'_n)$ を計算し、 $H(F_1) \sim H(F_n)$ と等しいか調べる。
4. $H(F_i) = H(F'_i)$ となる F_i は $t+1$ 個以上ある。
→ ラグランジェの補間公式を用いて $f_i(x)$ を復元し、 s を得られる。
5. もし $f_i(x)$ が復元できなければ failure を出力する。

5.1.1 改良プロトコルの計算量

改良プロトコルでは1度に m 個の秘密を送るので、Basic プロトコルの約 m 倍の計算量が必要である。しかし、 $m = O(n)$ であれば多項式時間のものを m 倍しても多項式時間であるので、改良プロトコルの計算量は多項式時間である。

5.1.2 改良プロトコルの通信量

改良プロトコルでは1度に m 個の秘密 s を送るので s の長さは qm ビットとなる。また、使用するハッシュ関数は変わらないのでハッシュ関数 H の出力値は h ビットのままである。

Basic プロトコルの通信効率の限界式 (4.5) は

$$|X_i| \frac{|S| - 1}{\epsilon} + 1 \frac{2^q - 1}{t/2^h} + 1 = \frac{2^h(2^q - 1)}{t} + 1$$

であった。

通信効率の限界式より

$$|X_i| \frac{|S| - 1}{\epsilon} + 1 \frac{2^{qm} - 1}{t/2^h} + 1 = \frac{2^h(2^{qm} - 1)}{t} + 1$$

となる。ビット数で評価するため両辺の \log_2 をとると

$$\begin{aligned} \log_2 |X_i| \log_2 \left\{ \frac{2^h(2^{qm} - 1)}{t} + 1 \right\} &\approx \log_2 \left\{ \frac{2^h(2^{qm} - 1)}{t} \right\} \\ &= \log_2 2^h + \log_2(2^{qm} - 1) - \log_2 t \approx h + qm - \log_2 t \end{aligned}$$

一方、このプロトコルの $|X_i|$ とはチャンネル $\text{ch-}i$ を使って送信者が送る F_i と $H(F_1) \sim H(F_n)$ である。ここで F_i は qm ビットであり、 $H(F_1) \sim H(F_n)$ は h ビットなので

$$qm + hn = \log_2 |X_i| \quad h + qm - \log_2 t$$

$q = h$ 、 $m = n$ とすると

$$2hn = \log_2 |X_i| \quad h + hn - \log_2 t > hn - \log_2 n$$

オーダーで評価すると

$$O(hn) = O(\log_2 |X_i|) \quad \Omega(hn - \log_2 n)$$

ここで n を十分に大きい値だとすると

$$O(hn) = O(\log_2 |X_i|) \quad \Omega(hn - \log_2 n) \rightarrow \Omega(hn)$$

となり、オーダで考えると等しいことがわかる。したがって改良プロトコルは通信効率の限界に近いと言える。

5.2 プログラム

改良プロトコルをプログラム実装した結果、正しく動作した。最終的な目標をネットワーク上での実装としているため、それに伴う様々な準備を行った。

5.2.1 作成したプログラムの構造

今回プログラムの作成はFlash(ActionScript)を使用した。Flashは暗号関係のライブラリ(as3crypto[15]参照)が充実しており、これにより多倍長とハッシュ関数の導入が可能になった。流れとしてはmainファイルで全体を管理し、処理を各オブジェクトで行う。送信側の処理をSendObject、受信側をReceiveObject、改竄をFalsifyObjectの三つに分割している。

暗号化から復号までの流れ

mainファイルで任意bit数の素数Pを生成する。

送りたい情報Sの長さをm個に分割

送信オブジェクトで送信セットを作成

改竄オブジェクトで送信セットを改竄

受信オブジェクトで受信した情報を復号

復号された情報を表示

その構造を下図に示す。

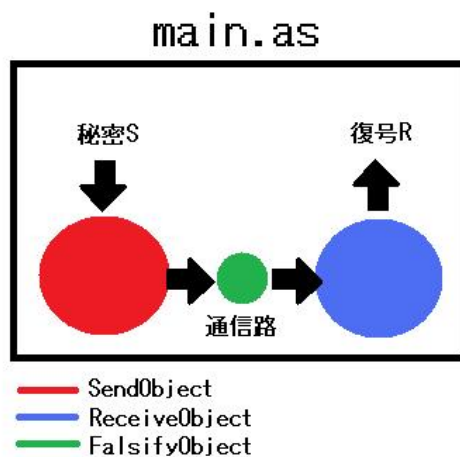


図 5.1 内部構造のイメージ

5.2.2 各オブジェクトの構造

継承した新しいクラスは作らずに BigInteger クラスを使用した。

送信側オブジェクトの手順

1. 1 ~ 9 行目 ランダム係数 a を生成
2. 11 ~ 46 行目 素数 P と a を使い $f(x)$ を作成
3. 48 ~ 59 行目 ハッシュ値を計算
4. 61 ~ 78 行目 F_i とハッシュ値で送信セットを作成

ソースコード 5.1 SendObject

```

1 //ランダム係数a[i][j]の生成
2
3 for( i=0;i<S_loop;i++ ) {
4     a[i] = new Array(2); //2次元配列に
5     a[i][0] = new BigInteger(GenerateKey(256)); //256 bitのランダム数生成
6     a[i][0] = a[i][0].mod(P); //配列はメソッドが検出されない
7     a[i][1] = new BigInteger(GenerateKey(256)); //256 bitのランダム数生成
8     a[i][1] = a[i][1].mod(P);
9 }
10
11 //Fs_send[i][j]の作成
12
13 for( i=0;i<S_loop;i++ ) {
14     S_bigpiece[i] = new BigInteger(S_piece[i]);
15     //BigIntegerに変換して代入
16 }
17
18 for( i=0;i<S_loop;i++ ) {
19     Fs_send[i] = new Array(5); //Fs_sendを2次元配列に
20 }
21
22 for( i=0;i<S_loop;i++ ) {
23     for( j=0;j<5;j++ ) {
24         g = new BigInteger(j.toString()); //gにjを代入
25         BItemp0 = new BigInteger("1");
26         g = g.add(BItemp0); //gを+1しておく(1^5で使いたい為)
27
28         BItemp0 = a[i][0]; //a[0]
29         BItemp1 = BItemp0.multiply(g); //a[0]*g
30
31         BItemp0 = a[i][1]; //a[1]
32         BItemp2 = g.multiply(g); //g*g
33         BItemp2 = BItemp2.multiply(BItemp0); //a[1]*g^2
34
35         BItemp0 = new BigInteger("0"); //f(g) = 0
36         BItemp0 = BItemp0.add(BItemp1); //f(g) = a[0]*g
37         BItemp0 = BItemp0.add(BItemp2); // f(g) = a[0]*g + a[1]*g^2
38         BItemp0 = BItemp0.add(S_bigpiece[i]);
39         // f(g) = S + a[0]*g + a[1]*g^2
40
41         BItemp0 = BItemp0.remainder(P);
42         // f(g) = S + a[0]*g + a[1]*g^2 mod P
43
44         Fs_send[i][j] = BItemp0;
45     }
46 }

```



```

47
48 //ハッシュ値H_send[i]の生成
49
50 for( i=0;i<5;i++ ) {
51     H_string[i] = Fs_send[0][i].toString();
52     if ( S_loop > 1 ) { //2つ以上に分割されているかどうか
53         H_string[i] = H_string[i].concat(Fs_send[1][i].toString());
54         if ( S_loop > 2 ) { //3つ以上に分割されているかどうか
55             H_string[i] = H_string[i].concat(Fs_send[2][i].toString());
56         }
57     }
58     H_send[i] = GenerateSHA1(H_string[i]);
59 }
60
61 //送信セットの作成
62
63 total = 5 + S_loop; // ハッシュの個数 + 分割個数
64
65 for( i=0;i<total;i++ ) {
66     F_send[i] = new Array(10); // F_sendを2次元配列に
67 }
68
69 for( k=0;k<5;k++ ) { //5セット作る
70     for( i=0;i<S_loop;i++ ){
71         F_send[k][i] = Fs_send[i][k];
72         //送信セットの頭からFs_sendを入れていく
73     }
74     for( j=S_loop;j<total;j++ ) {
75         F_send[k][j] = H_send[j-S_loop];
76         //Fs_sendの次にハッシュ値を入れる
77     }
78 }

```

受信オブジェクトの手順

1. 12~ 34行目 多数決で正しい $H(F_1) \sim H(F_n)$ を決める
2. 49~ 74行目 ハッシュ値を自身が計算したハッシュ値と比較する
3. 76~106行目 ラグランジェの補間公式で復号する
4. 108~114行目 復号された m 個の情報を結合して目的の情報 S を得る

ソースコード 5.2 ReceiveObject

```
1     for ( i=0;i<total;i++ ) {
2         F_receive[i] = new Array(10); //F_receiveを2次元配列に
3     }
4
5     for( i=0;i<5;i++ ) {
6         for( j=0;j<total;j++ ) {
7             F_receive[i][j] = F_send[i][j];
8             //受信セットに改竄した送信セットを代入
9         }
10    }
11
12    //多数決で正しいH_receive[i]を決定
13
14    j = 1;
15    i = 0;
16    k = 0;
17    flag = 0;
18    while( j < 5 ) {
19        flag = 0;
20        for( j=S_loop;j<total;j++ ) {
21            for( k=i+1;k<5;k++ ) {
22                if( F_receive[i][j] == F_receive[k][j] ) {
23                    flag++;
24                }
25            }
26            if ( flag >= 2 ) {
27                H_receive[j-S_loop] = F_receive[i][j];
28            }else{
29                i++;
30                j--;
31                flag = 0;
32            }
33        }
34    }
35
36    //f_receive[i][j]を取り出す
37
38    for( i=0;i<total;i++ ) {
39        f_receive[i] = new Array(10); //f_revceiveを2次元配列に
40    }
41
42    for( i=0;i<S_loop;i++ ) {
43        for( j=0;j<5;j++ ) {
44            f_receive[i][j] = F_receive[j][i];
45            //受信セットからf(g)を取り出す
46        }
47    }
```

```
48
49 //取り出したf_receiveのハッシュ値を計算
50
51 for( i=0;i<5;i++ ) {
52     H_string[i] = f_receive[0][i].toString();
53     //H_stringに分割した一つ目を代入
54     if ( S_loop > 1 ) { //2つ以上に分割されているかどうか
55         H_string[i] = H_string[i].concat(f_receive[1][i].toString());
56         //H_stringに分割した二つ目を追加
57         if ( S_loop > 2 ) { //3つ以上に分割されているかどうか
58             H_string[i] = H_string[i].concat(f_receive[2][i].toString());
59             //H_stringに分割した三つ目を追加
60         }
61     }
62     H_calculate[i] = GenerateSHA1(H_string[i]);
63     //f[g]を繋げたH_stringをGenerateSHA1()に代入してハッシュ値を得る
64 }
65
66 //ハッシュ値の比較
67
68 flag = 0;
69 for( i=0;i<5;i++ ) {
70     if( H_receive[i] == H_calculate[i] ) {
71         select[flag] = i + 1;
72         flag++;
73     }
74 }
75
76 //復号処理 ラグランジェの補間公式
77
78 for( i=0;i<S_loop;i++ ) {
79     select[0] = new BigInteger(select[0].toString());
80     select[1] = new BigInteger(select[1].toString());
81     select[2] = new BigInteger(select[2].toString());
82     BItemp5 = new BigInteger("0");
83     //初期化しておかないと足し算で足してしまう
84     for( k=0;k<=2;k++ ) {
85         BItemp3 = new BigInteger("1");
86         //1番目のループの掛け算で0にならないように代入
87         for( j=0;j<=2;j++ ) {
88             if( j!=k ) {
89                 BItemp0 = modsubtract(BIzero,select[j],P);
90                 // x - x2
91                 BItemp1 = modsubtract(select[k],select[j],P);
92                 // x1 - x2
93                 BItemp2 = moddivide(BItemp0,BItemp1,P);
94                 // ( x - x2 ) / ( x1 - x2 )
```

```

95             BItemp3 = modmultiply(BItemp3,BItemp2,P);
96             // (( x - x2 ) / ( x1 - x2 )) * (( x - x3 ) / ( x1 - x3 ))
97             }
98         }
99         BItemp4 = modmultiply(f_receive[i][select[k]-1],BItemp3,P);
100 // BItemp4[0] = f(x1) * (( x - x2 ) / ( x1 - x2 )) * (( x - x3 ) / ( x1 - x3 ))
101         BItemp5 = modadd(BItemp5,BItemp4,P);
102         // BItemp4[0]+BItemp4[1]+BItemp4[2]
103     }
104     decryption[i] = BItemp5;
105     //次のループでBItemp5が初期化されるので保管
106 }
107
108 S_decryption = decryption[0].toString();
109 if ( S_loop > 1 ) {
110     S_decryption = S_decryption.concat(decryption[1].toString());
111     if ( S_loop > 2 ) {
112         S_decryption = S_decryption.concat(decryption[2].toString());
113     }
114 }

```

改竄オブジェクトの手順

1. 1~17行目 敵の潜んでいる経路をランダムに決定
2. 20~45行目 同じ長さのランダムな文字列に置き換える

ソースコード 5.3 FalsifyObject

```

1 //敵の位置決定
2
3 i = 0;
4 flag = 0;
5 while ( i<2 ) {
6     flag = 0;
7     attack[i] = Math.floor(Math.random()*100)%5; //1~5までのランダムな数
8     for( j=0; j<i; j++ ) {
9         if ( attack[j] == attack[i] ) {
10            flag = 1;
11        }
12    }
13    if ( flag == 0 ) {
14        i++;
15    }
16
17 }

```

```
18
19
20 // 改竄
21
22 for ( i=0;i<total;i++ ) {
23     F_receive[i] = new Array(10); // F_receiveを2次元配列に
24 }
25
26 for( i=0;i<2;i++ ) {
27     for( j=0;j<total;j++ ) {
28         F_send[attack[i]][j] = new BigInteger(GenerateKey(160)).mod(P);
29         // 敵の位置に変数total分160bitを生成して代入
30     }
31     for ( k=0;k<S_loop;k++ ) {
32         F_send[attack[i]][k] = new BigInteger(GenerateKey(256)).mod(P);
33         // 敵の位置に変数S_loop分62文字を生成して代入
34     }
35 }
36
37 for ( i=0;i<total;i++ ) {
38     F_receive[i] = new Array(10); // F_receiveを2次元配列に
39 }
40
41 for( i=0;i<5;i++ ) {
42     for( j=0;j<total;j++ ) {
43         F_receive[i][j] = F_send[i][j]; // 受信セットに改竄した送信セットを代入
44     }
45 }
```

5.3 ネットワーク上での実装

本研究はネットワーク上での実装を最終目標としている。しかし、いきなり既存のネットワーク上で実装することはプロトコルの仕様以外の部分でも問題が生じる。そのため段階的に実装していく必要がある。次の段階として仮想ネットワーク上での実装を目指す。

5.3.1 仮想ネットワークの利点

次の段階として仮想ネットワークでの実装を目指す理由（利点）は実際のインターネット構造を無視できることにある。仮想ネットワークの場合既存のネットワークと比べてトラフィックや経路障害もない。それに加えて経路決定の問題（後述）をある程度緩和することができる。

5.3.2 実装上の問題

実装する上での問題は

1. 通信プロトコル
2. 経路決定の問題

である。通信プロトコルはTCP/IPの上に乗せるソフトウェアは何を使うのか、といった問題。経路決定の問題というのはプロトコルの仕様上 n 本それぞれ別々の経路を通る必要があるが、ルータは自身の持つ経路表に基づいた最適な経路を選択するため経路が全て一緒になってしまうことである。

5.3.3 問題の解決案

データを送る方法はFlashを使用する。Flashは容易にXMLにアクセスするメソッド等が用意されているためこれを利用する。またFlashはWEBベースであるためOSへの依存度が低く、既存の仮想ネットワーク構築ソフト上での動作が期待できる。サーバ（受信者）側にXML形式のファイルを経路の数だけ用意し、そこにクライアント（送信者）側がデータ（文字列）を書き込む。これによりサーバ（受信者）は必要な分のXMLファイルを使って復号することができる。経路決定の問題はソースルーティングの技術を使うことにより解決する。ソースルーティングとはルータが普段が自動的に選択している経路

を送信者が明示的に指定することができる技術である。途中のすべての経路を指定する「ストリクトソースルーティング」といくつかの経路を通過することを指定して、それ以外は途中のルータに任せる「ルーズソースルーティング」の2種類があるが、ここではストリクトソースルーティングを使う。実際のインターネットは大規模なネットワークであるため設定が困難であるが、仮想ネットワークでは最小構成でTCP/IPのネットワークを組むことができるため容易に使用できる。これにより経路が目的地に近づくにつれて一緒の経路を通過してしまう可能性を減らすことができる。一緒の経路を通ることはセキュリティ面において非常に危険であり、第三者に復号される可能性が高くなる。最小構成を図5.2、実際のインターネットを図5.3に示した。

5.3.4 実装計画

まず初めに仮想ネットワークを構築するソフトを決定する。選択する上での条件としては

1. 複数のホスト、ルータが設置できる
2. 設置したホストそれぞれでWebが使用可能
3. ソースルーティングに対応している

ことが必要である。次に、サーバクライアントでXMLファイルを読み書きするプログラムを作成する。そして、今回作った改良プロトコルにXMLファイルから文字列を読み出し復号するオブジェクトに代入する改良を加える。最後にソースルーティングによって経路指定を実装する。Linuxであれば「ポリシールーティング」などの機構を利用する。「ポリシールーティング」とは指定条件の packets に特定のルーティングを行う技術であり、ソースルーティングとも呼ばれている。

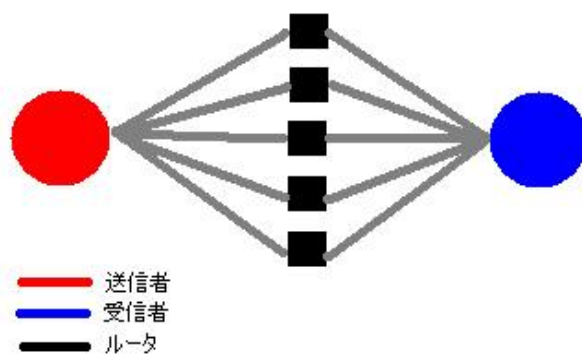


図 5.2 仮想ネットワークのイメージ

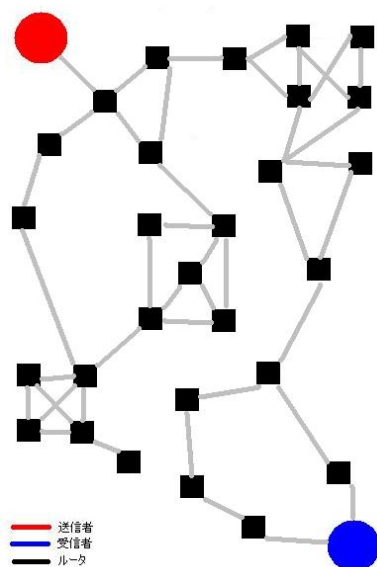


図 5.3 実際のネットワークのイメージ

第6章

結論

改良プロトコルによって通信効率を限界近くにすることができた。また、Flash(ActionScript)で実装した結果、正しく動作することが検証した。ネットワーク上での実装へ近づけ、仮想ネットワークでの実装方法を提案した。

謝辞

本研究を行なうにあたり、終始熱心に御指導していただいた木下宏揚教授、鈴木一弘先生に心から感謝致します。良き研究生生活を送らせていただいた木下研究室の方々に深く感謝致します。

参考文献

- [1] DANNY DOLEV、CYNTHIA DWORK、ORLI WAARTS、MOTI YUNG
”Perfectly Secure Message Transmission”
Journal of the Association for Computing Machinery、Vol.40,No.1、
pp.17-47(1993)
- [2] K. Srinathan、Arvind Narayanan、C. Pandu Rangan ”Optimal Perfectly Secure
Message Transmission”
CRYPTO 2004、LNCS 3152、 pp.545-561(2004)
- [3] Kaoru KUROSAWA、Kazuhiro SUZUKI、Members ”Almost Secure (1-
Round,n-Channel) Message Transmission Scheme”
IEICE TRANS. FUNDAMENTALS、VOL.E92-A,NO.1(2009)
- [4] 笠原正雄、佐竹賢治：”誤り訂正符号と暗号の基礎数理”、コロナ社(2004)
- [5] 澤田秀樹：”暗号理論と代数学”、海文堂(1997)
- [6] ”整数の合同” [http://www2.cc.niigata-u.ac.jp/takeuchi/tbasic/BackGround/-
Cong.html](http://www2.cc.niigata-u.ac.jp/takeuchi/tbasic/BackGround/-Cong.html)
- [7] 三谷政昭、佐藤伸一、ひのきいでろう：”マンガでわかる暗号”、オーム社
(2007)
- [8] 黒澤馨、尾形わかは：”現代暗号の基礎数理”、コロナ社(2004)
- [9] 今井秀樹：”情報・符号・暗号の理論”、コロナ社(2004)

-
- [10] Douglas R. Stinson、櫻井幸一：“暗号理論の基礎”、共立出版株式会社(1996)
- [11] DOUGLAS R. STINSON：“CRYPTOGRAPHY THEORY AND PRACTICE THIRD EDITION”、Chapman & Hall/CRC(2006)
- [12] HASAN MD. SAYEED、HOSAME ABU-AMARA ”Efficient Perfectly Secure Message Transmission in Synchronous Networks”
INFORMATION AND COMPUTATION 126、pp.53-61(1996)、ARTICLE NO.0033
- [13] Saurabh Agarwal、Ronald Cramer、Robbert de Haan ”Asymptotically Optimal Two-Round Perfectly Secure Message Transmission”
CRYPTO 2006、LNCS 4117、pp.394-408(2006)
- [14] Kaoru Kurosawa、Kazuhiro Suzuki ”Truly Efficient 2-Round Perfectly Secure Message Transmission Scheme”
Advances in Cryptology、EUROCRYPT 2008 LNCS 4965、pp.324-340(2008)
- [15] ”as3crypto” <http://code.google.com/p/as3crypto/>

質疑応答