信州大学工学部

学士論文

多項式補間法による強いランプ型しきい値秘密分散法

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　04T2050F
氏名　滝澤 克則

2009 年 2 月 28 日

目次

1 序章

今日では，あらゆる情報がディジタル化され，ハードディスクなどの記憶装置に保管されている．それらの情報は記憶装置の故障や自然災害による記憶装置の破壊で情報が取り出せなくなったり，侵入により情報漏洩する可能性がある．情報を故障や破壊，紛失から守るためには，そのコピーを作って保管しておけば良い．しかし，コピーが多いと情報の漏洩の可能性が高くなってしまう．この問題を解決する方法として，秘密分散法 [1][2] がある．

秘密分散法とは，秘密情報を n 個に分散させて管理する方式である．(k, n) しきい値秘密分散法では，n 個の分散情報 (シェア) のうちから任意の k 個のシェアを集めれば秘密情報を復元できるが，任意の $k-1$ 個のシェアからでは，秘密情報について何も分からないという特徴がある．このように，秘密分散法は，故障や破壊にも漏洩にも強い情報管理ができる．

本論文では, 第 2 章で秘密分散法の一つである Shamir により提案された (k, n) しきい値秘密分散法について述べる．第 3 章ではランプ型しきい値秘密分散法について，Shamir の多項式補間法をランプ型秘密分散法に拡張した方法では，強いランプ型秘密分散法にならない例をあげる．第 4 章では違った方法で拡張すれば強いランプ型秘密分散法になることを示す．第 5 章では強いランプ型であることを確かめる検証とファイルを秘密情報としてシェアを作って復元するときの工夫について述べ，また秘密分散法がどのようなことに使えるか考えた．第 6 章でまとめを述べる．

2 (k, n) しきい値秘密分散法

(k, n) しきい値法とは，秘密情報を n 個に分散化し，k 個以上集めた場合は，一意に秘密情報を求める事ができるが，k 個未満の場合では，秘密情報を求める事ができないという方法である．

2.1 (k, n) しきい値秘密分散法の定義

秘密情報 s，分散情報（シェア）$v_i (1 \leq i \leq n)$ が有限体 $GF(q)$ 上の値を取るとする．s が GF(q) 上の確率変数であるとき，v_i も確率変数となる．

秘密情報 s のシェア $(v_1, v_2, \cdots, v_n)$ が次のような 2 つの条件を満たすとき，(k, n) しきい値秘密分散法という．

- 条件 1
  任意の k 個のシェア $v_{i_1}, v_{i_2}, \cdots, v_{i_k}$ から s が復元できる．これより次式が成り立つ．

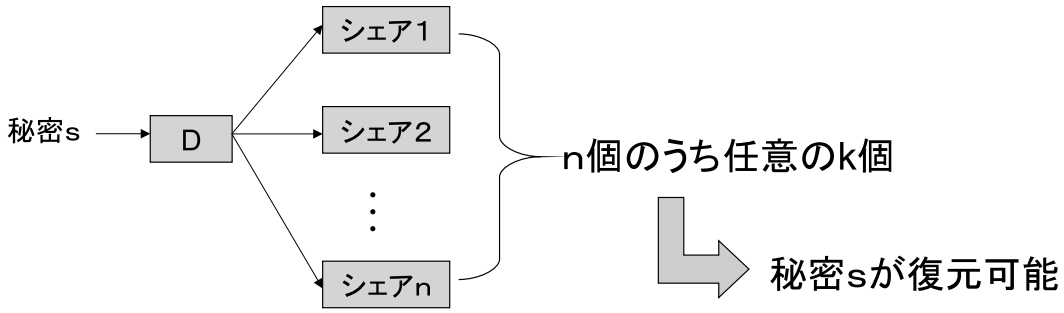

図 1　秘密分散法のモデル

$$H(s|v_{i_1}, v_{i_2}, \cdots, v_{i_k}) = 0 \tag{1}$$

- 条件 2
  任意の $k-1$ 個のシェア $v_{i_1}, v_{i_2}, \cdots, v_{i_{k-1}}$ からでは，s について何もわからない [3]. これより次式が成り立つ.

$$H(s|v_{i_1}, v_{i_2}, \cdots, v_{i_{k-1}}) = H(s) \tag{2}$$

2.2　実現方法

実現法として，多項式補間法を利用した Shamir の方法がよく知られている [1]．秘密情報 s の保有者 (ディーラー) が複数の分散情報を生成し, それらを複数の管理者に個別に管理させる．次からディーラーを D，s 人の分散管理者を P_1，P_2，…，P_n で表す．Shamir の方法で $(2, n)$ しきい値法を例として示す．$(2, n)$ しきい値法は図 2 のようになる．

分散段階では，秘密 s に対して a_1 をランダムに選び，$f(x) = s + a_1 x$ とおく．次に，v_i を $P_i (1 \leq i \leq n)$ に与える．

再構築段階で P_1, P_2 が集まったとすると，2 点 $(1, v_i), (2, v_2)$ を通る直線 $f(x)$ が一意に決まる．よって，$s = f(0)$ が求まる．しかし，P_i 1 人だけでは直線は求められないので，s について何も分からない．実際は実数上ではなく，有限体上で行う．

一般には次のようにすることができる．

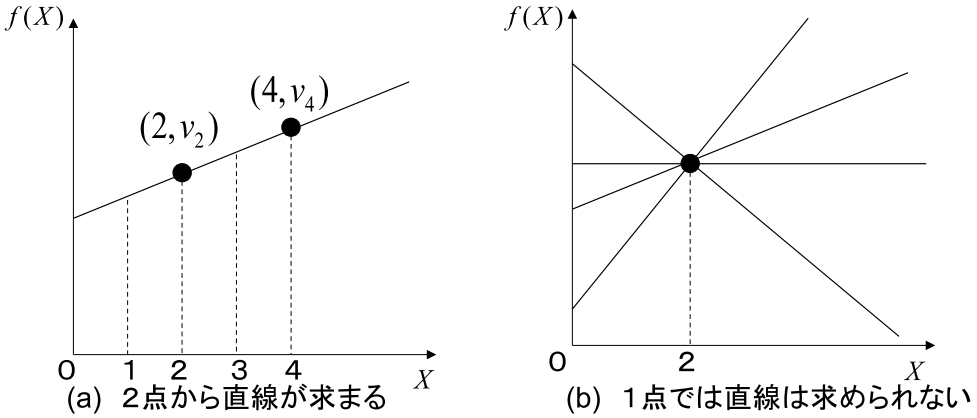

図 2　実現方法

- 分散段階
  $p > \max(s, n)$ となる素数 p を選び，$f(0) = s$ となる $GF(p)$ 上の $k-1$ 次の多項式を使い，

$$f(x) = s + a_1 x + \cdots + a_{k-1} x^{k-1} \bmod p \tag{3}$$

  $a_1, a_2, \cdots, a_{k-1}$ をランダムに選び，$v_i = f(i)$ を計算し，シェア v_i を $P_i (1 \leq i \leq n)$ に与える．
- 再構築段階
  k 人の分散管理者 $p_{i_1}, \cdots, p_{i_k}$ が集まったとき

$$v_{i_j} = f(i_j) \bmod p \tag{4}$$

  が $j = 1, \cdots, k$ で満たされる $k-1$ 次の多項式 $f(x)$ を考えれば $(i, (f(i))$ の組が k 個できるので任意の $f(x)$ が復元できる．よって，$f(0) = s$ なので，任意の k 人が集まると秘密 s を復元できる．また，どの $k-1$ 個が集まっても $f(x)$ が求まらないので，s について何も分からないことが分かる [4].

3 (k, L, n) しきい値秘密分散法

(k, L, n) しきい値法とは，任意の k 個以上のシェアが集まれば秘密を求めることができるが，任意の $k - L$ 個未満の場合では，秘密を求める事ができないという方法である．また，集めるシェアの数が多くなるにつれて秘密情報の曖昧さが徐々に減少していくことからランプ型秘密分散法と言われる．ここで，エントロピーの考え方による (k, n) しきい値秘密分散法とランプ型秘密分散法の違いを図 3 に示す．図 3 で，横軸は集めたシェアの数, 縦軸は秘密情報に関するエントロピーの大きさを表す．(a) は (k, n) しきい値秘密分散法で n が k 個集まればエントロピーは 0 となり s が分かるが, k 個未満だとエントロピーは $H(s)$ となり s について何も分からない．(b) はランプ型しきい値秘密分散法で n が k 個集まればエントロピーは 0 となり, $k - L$ 個以下だとエントロピーは $H(s)$ となるが, $k - L$ 個より多くなっていくと段々曖昧さが小さくなっていく．ランプ型秘密分散法は, シェアのサイズが (k, n) しきい値法に比べて，$1/L$ に抑えることができるので符号化効率が良くなる方法である．

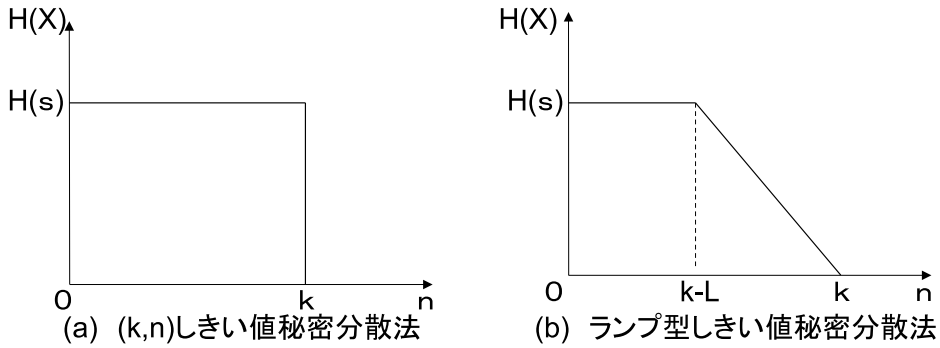

図 3　エントロピーによる考え方

3.1 強い (k, L, n) 符号

秘密を $s^L = (s_1, s_2, \cdots, s_L)$ とし，n 個のシェアを $v_1, v_2, \cdots, v_n$ とする．

- 条件 1
  $0 \leq t \leq L$ の t に対して，任意の $k - t$ 個のシェア $v_{j_1}, v_{j_2}, v_{j_3}, \cdots, v_{j_{k-t}}$ が次式を満たす符号を (k, L, n) 符号と言う．

$$H(s^L | v_{j_1}, v_{j_2}, \cdots, v_{j_{k-t}}) = \frac{t}{L} H(s^L) \tag{5}$$

- 条件 2

  $1 \leq t \leq L$ の t に対して，任意の $k-t$ 個のシェア $v_{j_1}, v_{j_2}, \cdots, v_{j_{k-t}}$ と，$s^L = (s_1, s_2, \cdots, s_L)$ の任意の t 個の組 $(s_{u_1}, s_{u_2}, \cdots, s_{u_t})$ が次式を満たす．

$$H(s_{u_1}, s_{u_2}, \cdots, s_{u_t} | v_{j_1}, v_{j_2}, \cdots, v_{j_{k-t}}) = \frac{t}{L} H(s^L) \tag{6}$$

  条件 1 も条件 2 も満たす符号を強い (k, L, n) 符号と言い，それ以外の (k, L, n) 符号を弱い (k, L, n) 符号と言う [5]．

3.2 拡張方法

Shamir の多項式補間法をランプ型秘密分散法に拡張する方法には，係数を秘密として拡張する方法がある．多項式 $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$ を用意する．ここで，秘密を $s^L = (s_1, s_2, \cdots, s_L)$ とすると，秘密は $s_1 = a_0, s_2 = a_1, \cdots, s_L = a_{L-1}$ となる．このように，係数を秘密として拡張したとき強いランプ型秘密分散法にならない例がある．例えば，$(4, 2, n)$ ランプ型秘密分散法を考える．秘密情報を s_1, s_2 とし，a_1, a_2 を独立な乱数とし，GF(17) 上で次の多項式を用意する．

$$f(x) = s_1 + s_2 x + a_1 x^2 + a_2 x^3 \tag{7}$$

分散情報は $v_i = f(i)$ で作れる．このとき，３個のシェア v_3, v_6, v_{15} は次のようになる．

$$v_3 = s_1 + 3s_2 + 9a_1 + 10a_2 \tag{8}$$
$$v_6 = s_1 + 6s_2 + 2a_1 + 12a_2 \tag{9}$$
$$v_{15} = s_1 + 15s_2 + 4a_1 + 9a_2 \tag{10}$$

以上より

$$5s_2 = 7v_3 - 8v_2 + v_{15} \tag{11}$$

を満たすことが分かる．これより，シェア v_3, v_6, v_{15} から秘密情報の一部の情報 s_2 が完全に復元されてしまうので，強いランプ型秘密分散法と言えない [6]．

4　拡張方法の提案

4.1　符号の作り方

多項式 $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ を使い, 秘密情報を $f(0), f(1)$ とする．乱数 $f(2), f(3)$ を使い，係数 a_0, a_1, a_2, a_3 を決め，シェアを作る．シェアは $v_i = f(i)(i \geq 4)$ となる．シェアが持つ情報は $(i, f(i))$ だけである．係数を決める式は次のようになる．

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0^2 & 0^3 \\ 1 & 1 & 1^2 & 1^3 \\ 1 & 2 & 2^2 & 2^3 \\ 1 & 3 & 3^2 & 3^3 \end{pmatrix}^{-1} \begin{pmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{pmatrix} \tag{12}$$

決まった係数からシェア v_i が作れる．n 個のシェアから任意の 4 個 $f(x_1), f(x_2), f(x_3), f(x_4)$ 集めれば，秘密情報を求められる．次の方程式

$$\begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{13}$$

より

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \end{pmatrix}^{-1} \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ f(x_4) \end{pmatrix} \tag{14}$$

以上より係数が求められるので，$f(x)$ が決まる．これにより秘密が復元できる．

4.2　強い符号であることの証明

ここで,

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ 1 & x_4 & x_4^2 & x_4^3 \end{pmatrix} \tag{15}$$

これは，ヴァンデルモンドの行列と呼ばれ一般式は次のようになる．

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{pmatrix} \tag{16}$$

このように定義される行列は次のような式が成り立つ.

$$\det A = \Pi_{1\leq i<j\leq n}(x_j - x_i) \tag{17}$$

証明は次のようになる.

$$f(x_1, x_2, \cdots, x_n) = \det A \tag{18}$$

とおくと，f は交代式であり，$\Pi_{i<j}(x_j - x_i)$ で割り切れる.

$$\begin{aligned} f(cx_1, cx_2, \cdots, cx_n) &= c * c^2 \cdots c^{n-1} f(x_1, x_2, \cdots, x_n) \\ &= c^{n(n-1)/2} f(x_1, x_2, \cdots, x_n) \end{aligned} \tag{19}$$

また,

$$\begin{aligned} \Pi_{i<j}(x_j - x_i) = (x_2 - x_1)(x_3 - x_1)\cdots(x_{n-1} - x_1)(x_n - x_1) \\ (x_3 - x_2)\cdots(x_{n-1} - x_2)(x_n - x_2) \\ \cdots \\ (x_{n-1} - x_{n-2})(x_n - x_{n-2}) \\ (x_{n-x} - x_{n-1}) \end{aligned} \tag{20}$$

であるので，次数は $n(n-1)/2$ である．ここで，$f = c\Pi_{i<j}(x_j - x_i)$ で $1x_2x_3\cdots x_n^{n-1}$ の項の係数をみると，f についても $\Pi_{i<j}(x_j - x_i)$ についても 1 である．よって,

$$\det A = f = \Pi_{1\leq i<j\leq n}(x_j - x_i) \tag{21}$$

以上より，$\det A \neq 0$ であるのでヴァンデルモンドの行列は正則なので，ただ一つの解しか持たない．また，n 個より少ない方程式からでは解は求められない [7][8]．よって，この方法で拡張すると強いランプ型秘密分散法ができる.

5 検証

5.1 符号の強さの検証

$GF(2^8)$ の上で多項式 $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ を使い，秘密を $f(0), f(1), f(2)$(各 1 バイト）とし，$f(3)$ を決めて係数 a_0, a_1, a_2, a_3 を求め，求めた係数よりシェアを作る．この作ったシェアを 4 個集めれば秘密 $f(0), f(1), f(2)$ を求めることができる．ここで，シェアを 3 個集めたときどの程度秘密を求めることができるか $\{f(0), f(1), f(2)\}$ のパターンを調べた．また，シェアを 2 個集めたときも調べた．これは実際にプログラムを作って調べた．3 個

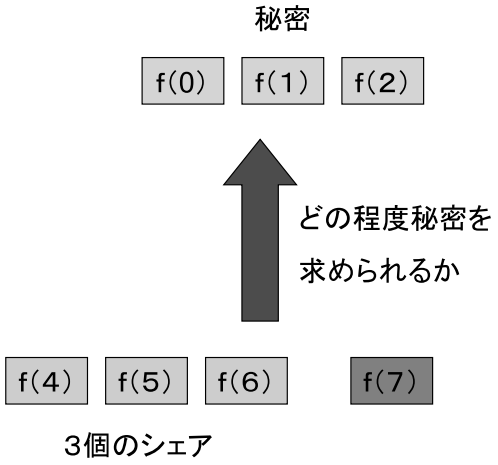

図 4　秘密を探る検証のモデル

のシェアを使って秘密を探ったとき，$\{f(0), f(1), f(2)\}$ のパターンは 256 通りあった．また，$f(0), f(1), f(2)$ はそれぞれ 0〜255 の数が 1 回ずつしか出て来なかった．

2 個のシェアを使って秘密を探ったとき，$\{f(0), f(1), f(2)\}$ のパターンは 256^2 通りあった．また，$f(0), f(1), f(2)$ はそれぞれ 0〜255 の数が 256 回ずつ出てきた．256 通りや 256^2 通りなら秘密を探られる可能性が高いので，実際は $GF(2^8)$ の大きさを大きくしたり，多項式の項数を多くして秘密を求めることができるシェアの数を増やせば実用上問題なくできる．

このことから，強いランプ型符号というのは $\{f(0), f(1), f(2)\}$ のパターンで見ると，徐々に秘密となる情報は特定されるが，$f(0), f(1), f(2)$ のそれぞれを見ても同じ回数でてくるので $f(0), f(1), f(2)$ の値は特定されず，その中の 1 ビットも特定されないことが分かる．

5.2　ファイルを秘密情報としてシェアを作る場合

コンピュータに保存してあるファイルを秘密情報としてシェアを作り，復元するとき次のようなことが考えられる．例えば，一つの秘密 $f_s(0)$ を 1 バイトとし，秘密が 5 バイトのとき二つの多項式 $f_1(x), f_2(x)$ を使うと秘密は，$f_1(0), f_1(1), f_1(2), f_2(0), f_2(1), f_2(2)$ となり，どれか 1 つに余計な情報が 1 バイト入ってしまう．そのため，復元したときに余計な情報も入ってしまうと考えられる．それを解決するために次のようなことを考えた．秘密を $(f_1(0), f_1(1), f_1(2)), \ldots, (f_{n-1}(0), f_{n-1}(1), f_{n-1}(2)), (f_n(0), f_n(1), f_n(2))$ とし，一番最後の秘密 $f_n(0)$ をその前の秘密がいくつ必要な情報であるかを表す数にし，$f_n(1), f_n(2)$ はダミーとする．$f_n(0) = 2$ なら $f_{n-1}(0)$ と $f_{n-1}(1)$ が必要な情報となり，$f_{n-1}(2)$ はダミーとなる．

このように考え，実際にプログラムでファイルのシェアを 4 つ作り，元に復元できることを

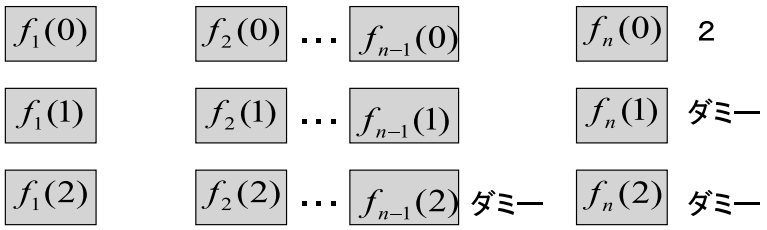

図 5　ファイルを秘密情報としてシェアを作る場合のモデル

確認した．これで，ファイルの大きさに関係なくファイルを秘密情報としてシェアを作り，余計な情報を入れないで復元できる．

5.3　シナリオ

これまで研究してきたことからどのようなことに利用できるか考えた．

5.3.1　RAID6 システムへの利用

RAID6 システムとは，複数のハードディスクで構成されたストレージにおいて，どの 2 台のディスクが同時に故障してもすべてのデータを復元できるシステムのことである．例えば，ハードディスクA，B，P，Qがそれぞれ異なる 4 台のパソコンにある場合を考える．A，B はユーザデータ，P，Qはパリティデータと言い，パリティデータはユーザデータ復元のためのデータでともにA，Bに依存する．また，PとQは有限体上の異なる一次式として定義すればよい．この RAID6 システムは，もし 1 台のパソコンからデータを取り出されたら，それがA，Bなら情報の一部が完全に漏れることがあり，P，QでもA，Bからなる式なので，ある程度情報が漏れてしまう．

ここで，秘密分散法で RAID6 システムを実現すれば情報の一部が漏れなくてすむ．次のような多項式を使う．

$$f(x) = a_0 + a_1 x \tag{22}$$

秘密を $f(0), f(1)$ とし，4 台のハードディスクに格納するデータを $f(2), f(3), f(4), f(5)$ とすれば，4 つのうち 2 つ集まれば復元できる．また，それぞれのハードディスクは $f(0), f(1)$ とは全く関係ない値なので，1 つが取り出されたとしても情報の一部が漏れることはない．

5.3.2　シェアの持つ数を変える

人によってシェアを持つ数を変えることで，秘密情報を集まる人によって復元できる人数を変えられる．例えば，10 個シェアを作り，4 個集まれば復元できるとする．Aは 4 個，B，C

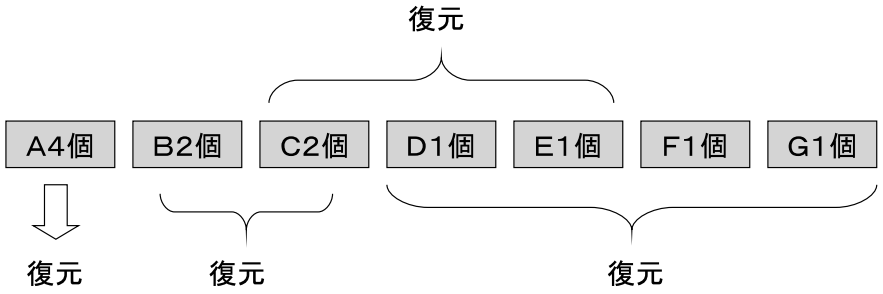

図 6　シェアの持つ数を変えたとき

は 2 個，D，E，F，Gは 1 個シェアを持つとする．すると，Aは一人で秘密情報が得られ，BとCでは二人で秘密情報が得られ，BとDとEでは三人で秘密情報が得られ，DとEとFとGでは四人で秘密情報が得られる．このように，持てるシェアの数を人により変えれば，復元できる人数を変えられる．実際には，会社などの役職に応じて持てるシェアの数を変えれば，うまく利用できる．

6　まとめ

係数を秘密として拡張する方法では強いランプ型しきい値秘密分散法はできないが，$f(x)$の値を秘密として拡張すれば強いランプ型しきい値秘密分散法ができることを示した．また，秘密を復元するとき余計な情報が入らないように工夫すれば，問題なく実現できることを示した．

謝辞

本研究を行うにあたって, 丁寧にご指導してくださった西新幹彦准教授に感謝の意を表する．

参考文献

[1] A.Shamir,“How to share a secret”,Communications of the ACM, 22, pp.612-613, Nov.1979.

[2] G.R.Blakley,“Safeguarding cryptographic keys”,AFIPS 1979 Nat.Computer Conf., vol.48, pp.313-317, 1979.

[3] 山本博資,「秘密分散法とそのバリエーション」, 数理解析研究所講究録,1361 巻, pp.19-31, 2004.
[4] 黒澤馨, 尾形わかは, 現代暗号の基礎数理, コロナ社,2004.
[5] 山本博資,「(k,L,n) しきい値秘密分散法」, 電気通信学会論文誌, vol.J68-A, no.9, pp.945-952, 1985.
[6] 岩本貢, 山本博資,「一般アクセス構造に対する強い秘密保護特性をもつランプ型秘密分散法」,The 27th Symposium on Information Theory and Its Applications (SITA 2004), pp.331-334, Dec.2004.
[7] 藤原良, 神保雅一, 符号と暗号の数理, 共立出版株式会社, 1993.
[8] 砂田利一, 行列と行列式, 岩波書店, 2003

付録 A　ソースコード

A.1　有限体 $GF(2^8)$ を作るプログラム

```
// 有限体クラス
class GF256
{
private:
unsigned char num; // number

public:
GF256(){}
GF256(unsigned char n): num(n) {}
GF256 operator+(const GF256 &b);
GF256 operator*(const GF256 &b);
GF256 operator/(const GF256 &b);
operator int();
};

inline GF256 GF256::operator+(const GF256 &b)
{
return(GF256(num ^ b.num));
}

GF256 GF256::operator*(const GF256 &b)
{
unsigned char a7b = (num & 0x80)? b.num: 0;
unsigned char a6b = (num & 0x40)? b.num: 0;
unsigned char a5b = (num & 0x20)? b.num: 0;
unsigned char a4b = (num & 0x10)? b.num: 0;
unsigned char a3b = (num & 0x08)? b.num: 0;
unsigned char a2b = (num & 0x04)? b.num: 0;
unsigned char a1b = (num & 0x02)? b.num: 0;
unsigned char a0b = (num & 0x01)? b.num: 0;
unsigned char y0 =
(a0b)^(a1b<<1)^(a2b<<2)^(a3b<<3)^(a4b<<4)^(a5b<<5)^(a6b<<6)^(a7b<<7);
unsigned char y1 =
(a1b>>7)^(a2b>>6)^(a3b>>5)^(a4b>>4)^(a5b>>3)^(a6b>>2)^(a7b>>1);
unsigned char p = y0;
p ^= (y1 << 4) & 0xf0;
p ^= (y1 << 3) & 0xf8;
p ^= (y1 << 2) & 0xfc;
p ^= (y1 >> 0) & 0x0f;
p ^= (y1 >> 1) & 0x08;
```

```
p ^= (y1 >> 2) & 0x14;
p ^= (y1 >> 3) & 0x04;
p ^= (y1 >> 4) & 0x03;
p ^= (y1 >> 5) & 0x03;
p ^= (y1 >> 6) & 0x01;
return(GF256(p));
}

GF256 GF256::operator/(const GF256 &b){
unsigned char c[8];
unsigned char e[8];

c[7] = (b.num << 0) ^ (b.num >> 4) ^ (b.num >> 5) ^ (b.num >> 6);
c[6] = (b.num << 1) ^ (b.num >> 3) ^ (b.num >> 4) ^ (b.num >> 5);
c[5] = (b.num << 2) ^ (b.num >> 2) ^ (b.num >> 3) ^ (b.num >> 4);
c[4] = (b.num << 3) ^ (b.num >> 1) ^ (b.num >> 2) ^ (b.num >> 3) ^
(b.num >> 7);
c[3] = (b.num << 4) ^ (b.num >> 1) ^ (b.num >> 2) ^ (b.num >> 4) ^
(b.num >> 5);
c[2] = (b.num << 5) ^ (b.num >> 1) ^ (b.num >> 3) ^ (b.num >> 5) ^
(b.num >> 6);
c[1] = (b.num << 6) ^ (b.num >> 2) ^ (b.num >> 6) ^ (b.num >> 7);
c[0] = (b.num << 7) ^ (b.num >> 1) ^ (b.num >> 5) ^ (b.num >> 6) ^
(b.num >> 7);
for (int i = 0; i < 8; i++){
e[i] = !!(num & (1 << i));
}

for (int i = 0; i < 8; i++){
int j;
for (j = i; j < 8 && !(c[j] & (1 << i)); j++){
;
}
if (j >= 8){
return(GF256(0)); // divided by zero
}
unsigned char tmp;
tmp = c[i], c[i] = c[j], c[j] = tmp;
tmp = e[i], e[i] = e[j], e[j] = tmp;
for (j = 0; j < 8; j++){
if (i != j && (c[j] & (1 << i))){
c[j] ^= c[i];
e[j] ^= e[i];
}
}
}
unsigned char div = 0;
for (int i = 0; i < 8; i++){
div |= (e[i])? 0x80 >> i: 0;
}
return(GF256(div));
}

inline GF256::operator int()
{
return(num);
}

//-----------------------------------------------------------
// 有理的有限体クラス
class RGF256
{
private:
GF256 num; // numerator
GF256 den; // denominator

public:
RGF256(){}
RGF256(unsigned char n): num(n), den(1) {}
```

```
RGF256(GF256 n, GF256 d): num(n), den(d) {}
RGF256 operator+(const RGF256 &b);
RGF256 operator*(const RGF256 &b);
RGF256 operator/(const RGF256 &b);
RGF256 &operator=(const unsigned char b);
operator int();
};

inline RGF256 RGF256::operator+(const RGF256 &b)
{
return(RGF256(num * b.den + den * b.num, den * b.den));
}

inline RGF256 RGF256::operator*(const RGF256 &b)
{
return(RGF256(num * b.num, den * b.den));
}

inline RGF256 RGF256::operator/(const RGF256 &b)
{
return(RGF256(num * b.den, den * b.num));
}

inline RGF256 &RGF256::operator=(const unsigned char b)
{
num = GF256(b);
den = GF256(1);
return(*this);
}

inline RGF256::operator int(){
return((int)(num / den));
}
```

A.2 シェア３つから秘密を探るプログラム

```
#include <iostream>
#define MATRIXSIZE 4

int main(void)
{

    RGF256 m0123[MATRIXSIZE][MATRIXSIZE] = {
        {1, 0, 0, 0}, {1, 1, 1, 1}, {1, 2, 4, 8}, {1, 3, 5, 15}};
    RGF256 m0123i[MATRIXSIZE][MATRIXSIZE] = {
        {1, 0, 0, 0}, {123, 1, 142, 244}, {0, 122, 244, 142}, {122, 122, 122, 122}};
    RGF256 m4567[MATRIXSIZE][MATRIXSIZE] = {
        {1, 4, 16, 64}, {1, 5, 17, 85}, {1, 6, 20, 120}, {1, 7, 21, 107}};
    RGF256 m4567i[MATRIXSIZE][MATRIXSIZE] = {
        {27, 28, 18, 20}, {136, 242, 125, 7}, {245, 143, 1, 123}, {122, 122, 122, 122}};

    int p0[256];
    int p1[256];
    int p2[256];
    int p3[256];

    for (int i = 0; i < 256; i++){
        p0[i] = 0;
        p1[i] = 0;
        p2[i] = 0;
        p3[i] = 0;
    }

    RGF256 f0(1);
    RGF256 f1(2);
```

```
    RGF256 f2(3);
    RGF256 f3(4);

    RGF256 a0 = m0123i[0][0] * f0 + m0123i[0][1] * f1 + m0123i[0][2] * f2 + m0123i[0][3] * f3;
    RGF256 a1 = m0123i[1][0] * f0 + m0123i[1][1] * f1 + m0123i[1][2] * f2 + m0123i[1][3] * f3;
    RGF256 a2 = m0123i[2][0] * f0 + m0123i[2][1] * f1 + m0123i[2][2] * f2 + m0123i[2][3] * f3;
    RGF256 a3 = m0123i[3][0] * f0 + m0123i[3][1] * f1 + m0123i[3][2] * f2 + m0123i[3][3] * f3;

    RGF256 f4 = m4567[0][0] * a0 + m4567[0][1] * a1 + m4567[0][2] * a2 + m4567[0][3] * a3;
    RGF256 f5 = m4567[1][0] * a0 + m4567[1][1] * a1 + m4567[1][2] * a2 + m4567[1][3] * a3;
    RGF256 f6 = m4567[2][0] * a0 + m4567[2][1] * a1 + m4567[2][2] * a2 + m4567[2][3] * a3;
    RGF256 f7 = m4567[3][0] * a0 + m4567[3][1] * a1 + m4567[3][2] * a2 + m4567[3][3] * a3;

    for (int i = 0; i < 256; i++){
        RGF256 f7a(i);

        RGF256 a0a = m4567i[0][0] * f4 + m4567i[0][1] * f5 + m4567i[0][2] * f6 + m4567i[0][3] * f7a;
        RGF256 a1a = m4567i[1][0] * f4 + m4567i[1][1] * f5 + m4567i[1][2] * f6 + m4567i[1][3] * f7a;
        RGF256 a2a = m4567i[2][0] * f4 + m4567i[2][1] * f5 + m4567i[2][2] * f6 + m4567i[2][3] * f7a;
        RGF256 a3a = m4567i[3][0] * f4 + m4567i[3][1] * f5 + m4567i[3][2] * f6 + m4567i[3][3] * f7a;

        RGF256 f0a = m0123[0][0] * a0a + m0123[0][1] * a1a + m0123[0][2] * a2a + m0123[0][3] * a3a;
        RGF256 f1a = m0123[1][0] * a0a + m0123[1][1] * a1a + m0123[1][2] * a2a + m0123[1][3] * a3a;
        RGF256 f2a = m0123[2][0] * a0a + m0123[2][1] * a1a + m0123[2][2] * a2a + m0123[2][3] * a3a;
        RGF256 f3a = m0123[3][0] * a0a + m0123[3][1] * a1a + m0123[3][2] * a2a + m0123[3][3] * a3a;

        std::cout << (int)f0a << ' ' << (int)f1a <<  ' ' << (int)f2a << ' ' << (int)f3a << '\n';

        p0[(int)f0a]++;
        p1[(int)f1a]++;
        p2[(int)f2a]++;
        p3[(int)f3a]++;

    }

    for (int k = 0; k < 256; k++ ) {
        std::cout << k << ' ' << p0[k] << ' ' << p1[k] << ' ' << p2[k] << ' ' << p3[k] << '\n';
    }

    return(0);
}
```

A.3 ファイルからシェアを４つ作るプログラム

```
#include <iostream>
#include <fstream>
#define MATRIXSIZE 4

int main(int argc, char *argv[])
{

    RGF256 m0123[MATRIXSIZE][MATRIXSIZE] = {
        {1, 0, 0, 0}, {1, 1, 1, 1}, {1, 2, 4, 8}, {1, 3, 5, 15}};
    RGF256 m0123i[MATRIXSIZE][MATRIXSIZE] = {
        {1, 0, 0, 0}, {123, 1, 142, 244}, {0, 122, 244, 142}, {122, 122, 122, 122}};
    RGF256 m4567[MATRIXSIZE][MATRIXSIZE] = {
        {1, 4, 16, 64}, {1, 5, 17, 85}, {1, 6, 20, 120}, {1, 7, 21, 107}};
    RGF256 m4567i[MATRIXSIZE][MATRIXSIZE] = {
        {27, 28, 18, 20}, {136, 242, 125, 7}, {245, 143, 1, 123}, {122, 122, 122, 122}};

    if (argc != 2){
        std::cerr << "引数の数が違います。\n";
        exit(1);
    }
    std::ifstream secret_file;
```

```
secret_file.open(argv[1], std::ios::in | std::ios::binary);
if (secret_file.fail()){
    std::cerr << "open error: " << argv[1] << '\n';
    exit(1);
}

std::ofstream share[4];
share[0].open("share4.dat", std::ios::out | std::ios::binary | std::ios::trunc);
share[1].open("share5.dat", std::ios::out | std::ios::binary | std::ios::trunc);
share[2].open("share6.dat", std::ios::out | std::ios::binary | std::ios::trunc);
share[3].open("share7.dat", std::ios::out | std::ios::binary | std::ios::trunc);
if (share[0].fail() || share[1].fail() || share[2].fail() || share[3].fail()){
    std::cerr << "open error: share*.dat\n";
    exit(1);
}

char char_buf;
char_buf = 4;
share[0].write(&char_buf, 1);
char_buf = 5;
share[1].write(&char_buf, 1);
char_buf = 6;
share[2].write(&char_buf, 1);
char_buf = 7;
share[3].write(&char_buf, 1);

RGF256 f0, f1, f2, f3;
RGF256 a0, a1, a2, a3;
RGF256 f4, f5, f6, f7;
char in_buffer[3];
int tail, tail2;

for (; secret_file.read(in_buffer, 3), (tail = secret_file.gcount()) > 0; tail2 = tail){
    switch (tail){
    case 3:
        f0 = in_buffer[0];
        f1 = in_buffer[1];
        f2 = in_buffer[2];
        f3 = 0; // random number
        break;
    case 2:
        f0 = in_buffer[0];
        f1 = in_buffer[1];
        f2 = 0; // random number
        f3 = 0; // random number
        break;
    case 1:
        f0 = in_buffer[0];
        f1 = 0; // random number
        f2 = 0; // random number
        f3 = 0; // random number
        break;
    default:
        break;
    }
//係数を決める
    a0 = m0123i[0][0] * f0 + m0123i[0][1] * f1 + m0123i[0][2] * f2 + m0123i[0][3] * f3;
    a1 = m0123i[1][0] * f0 + m0123i[1][1] * f1 + m0123i[1][2] * f2 + m0123i[1][3] * f3;
    a2 = m0123i[2][0] * f0 + m0123i[2][1] * f1 + m0123i[2][2] * f2 + m0123i[2][3] * f3;
    a3 = m0123i[3][0] * f0 + m0123i[3][1] * f1 + m0123i[3][2] * f2 + m0123i[3][3] * f3;
//シェアを求める
    f4 = m4567[0][0] * a0 + m4567[0][1] * a1 + m4567[0][2] * a2 + m4567[0][3] * a3;
    f5 = m4567[1][0] * a0 + m4567[1][1] * a1 + m4567[1][2] * a2 + m4567[1][3] * a3;
    f6 = m4567[2][0] * a0 + m4567[2][1] * a1 + m4567[2][2] * a2 + m4567[2][3] * a3;
    f7 = m4567[3][0] * a0 + m4567[3][1] * a1 + m4567[3][2] * a2 + m4567[3][3] * a3;

    char_buf = f4;
    share[0].write(&char_buf, 1);
    char_buf = f5;
```

```
        share[1].write(&char_buf, 1);
        char_buf = f6;
        share[2].write(&char_buf, 1);
        char_buf = f7;
        share[3].write(&char_buf, 1);
    }

    f0 = tail2;
    f1 = 0; // random number
    f2 = 0; // random number
    f3 = 0; // random number

//係数を決める
    a0 = m0123i[0][0] * f0 + m0123i[0][1] * f1 + m0123i[0][2] * f2 + m0123i[0][3] * f3;
    a1 = m0123i[1][0] * f0 + m0123i[1][1] * f1 + m0123i[1][2] * f2 + m0123i[1][3] * f3;
    a2 = m0123i[2][0] * f0 + m0123i[2][1] * f1 + m0123i[2][2] * f2 + m0123i[2][3] * f3;
    a3 = m0123i[3][0] * f0 + m0123i[3][1] * f1 + m0123i[3][2] * f2 + m0123i[3][3] * f3;
//シェアを求める
    f4 = m4567[0][0] * a0 + m4567[0][1] * a1 + m4567[0][2] * a2 + m4567[0][3] * a3;
    f5 = m4567[1][0] * a0 + m4567[1][1] * a1 + m4567[1][2] * a2 + m4567[1][3] * a3;
    f6 = m4567[2][0] * a0 + m4567[2][1] * a1 + m4567[2][2] * a2 + m4567[2][3] * a3;
    f7 = m4567[3][0] * a0 + m4567[3][1] * a1 + m4567[3][2] * a2 + m4567[3][3] * a3;

    char_buf = f4;
    share[0].write(&char_buf, 1);
    char_buf = f5;
    share[1].write(&char_buf, 1);
    char_buf = f6;
    share[2].write(&char_buf, 1);
    char_buf = f7;
    share[3].write(&char_buf, 1);

    share[0].close();
    share[1].close();
    share[2].close();
    share[3].close();
    secret_file.close();

    return(0);
}
```

A.4 作ったシェアから復号するプログラム

```
#include <iostream>
#include <fstream>
#define MATRIXSIZE 4

int main(int argc, char *argv[])
{

    RGF256 m0123[MATRIXSIZE][MATRIXSIZE] = {
        {1, 0, 0, 0}, {1, 1, 1, 1}, {1, 2, 4, 8}, {1, 3, 5, 15}};
    RGF256 m0123i[MATRIXSIZE][MATRIXSIZE] = {
        {1, 0, 0, 0}, {123, 1, 142, 244}, {0, 122, 244, 142}, {122, 122, 122, 122}};
    RGF256 m4567[MATRIXSIZE][MATRIXSIZE] = {
        {1, 4, 16, 64}, {1, 5, 17, 85}, {1, 6, 20, 120}, {1, 7, 21, 107}};
    RGF256 m4567i[MATRIXSIZE][MATRIXSIZE] = {
        {27, 28, 18, 20}, {136, 242, 125, 7}, {245, 143, 1, 123}, {122, 122, 122, 122}};

    if (argc != 2){
        std::cerr << "引数の数が違います。\n";
        exit(1);
    }
    std::ofstream secret_file;
    ecret_file.open(argv[1], std::ios::out | std::ios::binary | std::ios::trunc);
```

```
    if (secret_file.fail()){
        std::cerr << "open error: " << argv[1] << '\n';
        exit(1);
    }

    std::ifstream share[4];
    share[0].open("share4.dat", std::ios::in | std::ios::binary);
    share[1].open("share5.dat", std::ios::in | std::ios::binary);
    share[2].open("share6.dat", std::ios::in | std::ios::binary);
    share[3].open("share7.dat", std::ios::in | std::ios::binary);
    if (share[0].fail() || share[1].fail() || share[2].fail() || share[3].fail()){
        std::cerr << "open error: share*.dat\n";
        exit(1);
    }

    char char_buf;
    share[0].read(&char_buf, 1);
    if (char_buf != 4){
        std::cerr << "share number error: [0]\n";
        exit(1);
    }
    share[1].read(&char_buf, 1);
    if (char_buf != 5){
        std::cerr << "share number error: [1]\n";
        exit(1);
    }
    share[2].read(&char_buf, 1);
    if (char_buf != 6){
        std::cerr << "share number error: [2]\n";
        exit(1);
    }
    share[3].read(&char_buf, 1);
    if (char_buf != 7){
        std::cerr << "share number error: [3]\n";
        exit(1);
    }

    char in_buffer[4];
    int in_counter;

    RGF256 f4, f5, f6, f7;
    RGF256 a0a, a1a, a2a, a3a;
    RGF256 f0a, f1a, f2a, f3a;

    char out_buf1[4];
    char out_buf2[4];

    in_counter = 0;
    share[0].read(in_buffer + 0, 1);
    in_counter += share[0].gcount();
    share[1].read(in_buffer + 1, 1);
    in_counter += share[1].gcount();
    share[2].read(in_buffer + 2, 1);
    in_counter += share[2].gcount();
    share[3].read(in_buffer + 3, 1);
    in_counter += share[3].gcount();
    if (in_counter != 4){
        std::cerr << "error in " << __LINE__ << '\n';
        exit(1);
    }

    f4 = in_buffer[0];
    f5 = in_buffer[1];
    f6 = in_buffer[2];
    f7 = in_buffer[3];
//シェアから係数を求める
    a0a = m4567i[0][0] * f4 + m4567i[0][1] * f5 + m4567i[0][2] * f6 + m4567i[0][3] * f7;
    a1a = m4567i[1][0] * f4 + m4567i[1][1] * f5 + m4567i[1][2] * f6 + m4567i[1][3] * f7;
    a2a = m4567i[2][0] * f4 + m4567i[2][1] * f5 + m4567i[2][2] * f6 + m4567i[2][3] * f7;
```

```
    a3a = m4567i[3][0] * f4 + m4567i[3][1] * f5 + m4567i[3][2] * f6 + m4567i[3][3] * f7;
//秘密を求める
    f0a = m0123[0][0] * a0a + m0123[0][1] * a1a + m0123[0][2] * a2a + m0123[0][3] * a3a;
    f1a = m0123[1][0] * a0a + m0123[1][1] * a1a + m0123[1][2] * a2a + m0123[1][3] * a3a;
    f2a = m0123[2][0] * a0a + m0123[2][1] * a1a + m0123[2][2] * a2a + m0123[2][3] * a3a;
    f3a = m0123[3][0] * a0a + m0123[3][1] * a1a + m0123[3][2] * a2a + m0123[3][3] * a3a;
    out_buf1[0] = f0a;
    out_buf1[1] = f1a;
    out_buf1[2] = f2a;
    out_buf1[3] = f3a;

    in_counter = 0;
    share[0].read(in_buffer + 0, 1);
    in_counter += share[0].gcount();
    share[1].read(in_buffer + 1, 1);
    in_counter += share[1].gcount();
    share[2].read(in_buffer + 2, 1);
    in_counter += share[2].gcount();
    share[3].read(in_buffer + 3, 1);
    in_counter += share[3].gcount();
    if (in_counter != 4){
        std::cerr << "error in " << __LINE__ << '\n';
        exit(1);
    }

    f4 = in_buffer[0];
    f5 = in_buffer[1];
    f6 = in_buffer[2];
    f7 = in_buffer[3];
//シェアから係数を求める
    a0a = m4567i[0][0] * f4 + m4567i[0][1] * f5 + m4567i[0][2] * f6 + m4567i[0][3] * f7;
    a1a = m4567i[1][0] * f4 + m4567i[1][1] * f5 + m4567i[1][2] * f6 + m4567i[1][3] * f7;
    a2a = m4567i[2][0] * f4 + m4567i[2][1] * f5 + m4567i[2][2] * f6 + m4567i[2][3] * f7;
    a3a = m4567i[3][0] * f4 + m4567i[3][1] * f5 + m4567i[3][2] * f6 + m4567i[3][3] * f7;
//秘密を求める
    f0a = m0123[0][0] * a0a + m0123[0][1] * a1a + m0123[0][2] * a2a + m0123[0][3] * a3a;
    f1a = m0123[1][0] * a0a + m0123[1][1] * a1a + m0123[1][2] * a2a + m0123[1][3] * a3a;
    f2a = m0123[2][0] * a0a + m0123[2][1] * a1a + m0123[2][2] * a2a + m0123[2][3] * a3a;
    f3a = m0123[3][0] * a0a + m0123[3][1] * a1a + m0123[3][2] * a2a + m0123[3][3] * a3a;
    out_buf2[0] = f0a;
    out_buf2[1] = f1a;
    out_buf2[2] = f2a;
    out_buf2[3] = f3a;

    in_counter = 0;
    share[0].read(in_buffer + 0, 1);
    in_counter += share[0].gcount();
    share[1].read(in_buffer + 1, 1);
    in_counter += share[1].gcount();
    share[2].read(in_buffer + 2, 1);
    in_counter += share[2].gcount();
    share[3].read(in_buffer + 3, 1);
    in_counter += share[3].gcount();

    while (in_counter == 4){
        secret_file.write(out_buf1, 3);
        out_buf1[0] = out_buf2[0];
        out_buf1[1] = out_buf2[1];
        out_buf1[2] = out_buf2[2];
        out_buf1[3] = out_buf2[3];

        f4 = in_buffer[0];
        f5 = in_buffer[1];
        f6 = in_buffer[2];
        f7 = in_buffer[3];
    //シェアから係数を求める
        a0a = m4567i[0][0] * f4 + m4567i[0][1] * f5 + m4567i[0][2] * f6 + m4567i[0][3] * f7;
        a1a = m4567i[1][0] * f4 + m4567i[1][1] * f5 + m4567i[1][2] * f6 + m4567i[1][3] * f7;
        a2a = m4567i[2][0] * f4 + m4567i[2][1] * f5 + m4567i[2][2] * f6 + m4567i[2][3] * f7;
```

```
        a3a = m4567i[3][0] * f4 + m4567i[3][1] * f5 + m4567i[3][2] * f6 + m4567i[3][3] * f7;
    //秘密を求める
        f0a = m0123[0][0] * a0a + m0123[0][1] * a1a + m0123[0][2] * a2a + m0123[0][3] * a3a;
        f1a = m0123[1][0] * a0a + m0123[1][1] * a1a + m0123[1][2] * a2a + m0123[1][3] * a3a;
        f2a = m0123[2][0] * a0a + m0123[2][1] * a1a + m0123[2][2] * a2a + m0123[2][3] * a3a;
        f3a = m0123[3][0] * a0a + m0123[3][1] * a1a + m0123[3][2] * a2a + m0123[3][3] * a3a;
        out_buf2[0] = f0a;
        out_buf2[1] = f1a;
        out_buf2[2] = f2a;
        out_buf2[3] = f3a;

        in_counter = 0;
        share[0].read(in_buffer + 0, 1);
        in_counter += share[0].gcount();
        share[1].read(in_buffer + 1, 1);
        in_counter += share[1].gcount();
        share[2].read(in_buffer + 2, 1);
        in_counter += share[2].gcount();
        share[3].read(in_buffer + 3, 1);
        in_counter += share[3].gcount();
    }
    if (in_counter != 0){
        std::cerr << "error in " << __LINE__ << '\n';
        exit(1);
    }
    secret_file.write(out_buf1, out_buf2[0]);

    share[0].close();
    share[1].close();
    share[2].close();
    share[3].close();
    secret_file.close();

    return(0);
}
```