平成 13 年度

学士学位論文

秘密分散における効率的かつ高速な計算処理

Fast Calculation for Secret Sharing Scheme

1020293　　嶋岡 哲夫

指導教員　　福本 昌弘

2002 年 2 月 8 日

高知工科大学 情報システム工学科

要 旨

秘密分散における効率的かつ高速な計算処理

嶋岡 哲夫

情報を安全に保管する方法のひとつとして秘密分散法があげられる．

過去の研究では，構成法に拡大体を用いて秘密分散法を実現した．この方法では，拡大体上の 2 項演算には演算表を用いているが，この演算表は膨大な容量を要する．

本研究では，拡大体の性質を考慮して四則演算は場合分けして行っている．加法については，直接 2 項演算と表引きするものとを場合分けしている．このとき用いる演算表は可換律を考慮して縮小を行っている．また，減法については，加法と演算結果が同じになるために考慮する必要がない．乗法および除法については，べき表現により直接 2 項演算している．これにより，乗法表と除法表を用いる必要がなくなった．

本研究の結果，実行の際の必要な容量を省き，拡大体上で効率よく秘密分散法を実行できる．

キーワード　秘密分散法，(k, n) しきい値法，拡大体，四則演算

Abstract

Fast Calculation for Secret Sharing Scheme

Tetsuo SHIMAOKA

The Secret Sharing Scheme(SSS) is a method to keep information out for safety.

In the past paper, The SSS is executed on the extension field $\boldsymbol{GF}(2^m)$.

On the extension field $\boldsymbol{GF}(2^m)$, it can't execute on two term calculation.

Then, it made operational tables for four operation rules solution of which can't execute on two term calculation. However, operational tables are large size, so they need enormous memory.

In this paper, two term calculation is distinguished character of the extension field $\boldsymbol{GF}(2^m)$.

Case of addtion, directly two term calculation and reference to operational table. then, this table is reduced by commutative law. Moreover, results of addtion and subtraction are equivalent, so subtraction doesn't need to consider.

Case of multiplication and division, it is possible exponential laws to calculate,so it doesn't need multi table and div table.

As a result,when SSS executes,it save extra memory.So,it is efficiency of SSS to executes on the extension field $\boldsymbol{GF}(2^m)$.

multiplication and division doesn't need to operational table, and addtion's table is less than half.

key words Secret Sharing Scheme(SSS), (k,n)threshold SSS, extension field, four operation rules

目次

図目次

表目次

第 1 章

序論

1.1　研究の背景・目的

現代社会では多くの情報が電子化され，コンピュータ等で保存されている．電子化された情報はペーパーベースの情報と比較すると，検索や編集を容易に実行でき，保存においても場所を取らないという利点を持っている．

その反面，コンピュータでの保存は，次のような危険をともなう．

- 情報の損失について

  破壊・損失・改ざん etc
- 情報の開示について

  漏洩などのプライバシーの侵害 etc

情報の管理法は大きく集中管理と分散管理に分類される．

集中管理はひとつのシステムで管理を行うため，管理自体は容易だが，システムに支障が起きた場合，情報そのものを失う可能性がある．仮に復元できたとしても膨大な労力を必要とする．

集中管理に対し，分散管理はコピーを作成して複数のシステムで管理する．そのため情報を紛失した場合は，バックアップとして保存しているものを他のシステムから取り出せばよい．しかし，情報そのものが重要であり一部の人間以外に公開したくない場合，コピーを作成した分だけ情報が露見する可能性が高くなり、コピーの作成は好ましくない．

第三者に情報の内容をわからないようにする方法として暗号がある．暗号化された情報は

第三者がアクセスしても意味のない符号文でしかない．そのため，内容が露見することはない．しかし，災害などによって情報を紛失した場合，情報が復元できない．また，誰かに暗号を解読された場合は内容が漏洩してしまう．

これらの危険から情報を安全に保存する一手段として秘密分散法があげられる．秘密分散法に関する研究には以下のようなものがあげられる [5].

- 特性に関する研究
  - アクセス構造
  - エントロピー
- 構成法に関する研究
  - 幾何学
  - ベクトル空間
  - マトロイド理論
  - グラフ理論
  - ブール関数

文献 [12] では，コンピュータで秘密分散法を実現することを考慮して，構成法に 2 の拡大体を用いた．拡大体上の四則演算は，実数体等と異なり，直接 2 項演算をすることができない．文献 [12] は，演算表を作成し表引きによりこの問題を解決したが，この演算表には容量の問題がある．そこで，本研究は拡大体の性質に着目し，実行の際の四則演算の効率化および高速化を検討する．

1.2　本論文の概要

本論文の構成について述べる．

第 2 章では，秘密分散法がどのようなものか説明する．また，秘密分散法の数学的知識として体論についても説明する．第 3 章では，拡大体上の秘密分散法を説明する．そして第 4 章で本研究のまとめを記述する．

第 2 章

秘密分散法と体論

2.1 概要

秘密分散法の概念，特徴と利点および種類について説明する．また，一般に秘密分散法は素数 p を法とする有限体 (素体) 上で実現されるため，体論についても説明する．そして，実現方法を説明し，その問題点を指摘する．本研究は素体ではなく拡大体上で秘密分散法を実現するため，章末に文献 [12] から素体と拡大体を比較する．

2.2 秘密分散法

秘密分散法を例を交えて説明する．

市役所に金庫があり，これを開ける権利を持つ人間が 5 人いるとする．このとき，5 人全員に鍵を渡してしまうと，各々が 1 人で簡単に金庫を開けることができるため，危険でありあまり好ましくない．そこで，鍵を持つ権利者 5 人のうち 3 人以上が合意したときのみ，金庫が開くようにしたい．秘密分散法はこのようなシステムを実現する．すなわち，情報を n 人で分散管理し (管理する人を参加者と呼ぶ)，このうち k 人が合意したときのみ情報の内容が確認できるという管理法である．もちろん，合意が $k-1$ 人以下ならばその内容というのは全くわからない [1].

秘密分散法は情報 S を次の条件を満たすように n 個の分散情報 $w_1, w_2, \cdots, w_n$ に分割する．この処理を分散符号化という．

ここで分散符号化する人をディーラ，作成された分散情報をシェア，シェアを受け取る人

2.2　秘密分散法

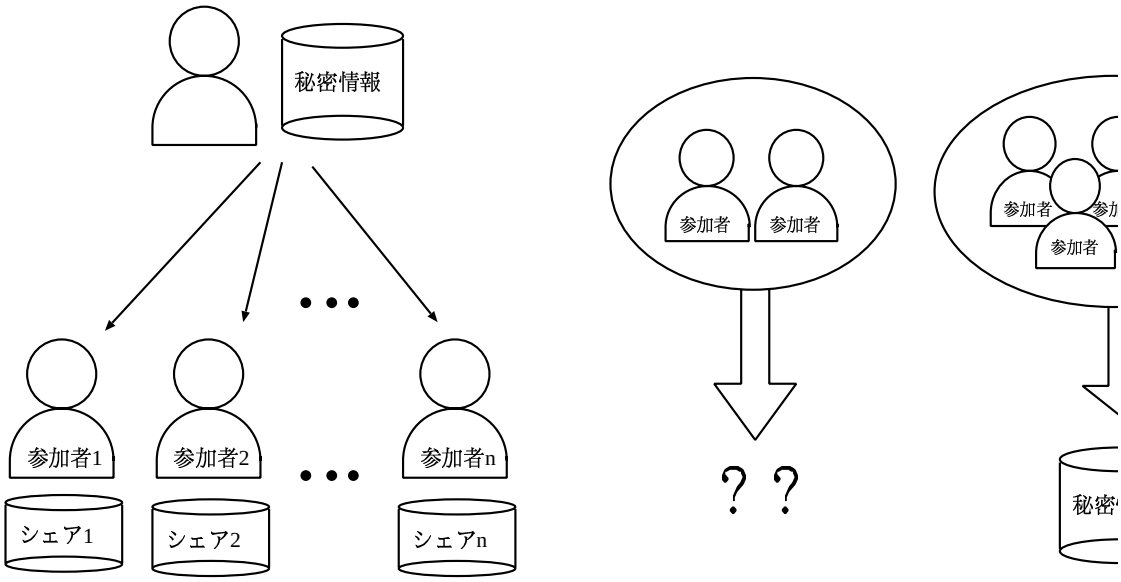

図 2.1　秘密分散法

を参加者と呼ぶこととする.

- シェア w_i が k 個集まれば S を算出可能
- シェア w_i が $k-1$ 個以下では情報不足で S の算出不可能

秘密情報 S を分散符号化して n 個のシェアを作り，n 人にそれぞれシェアを渡し各々で保管する.

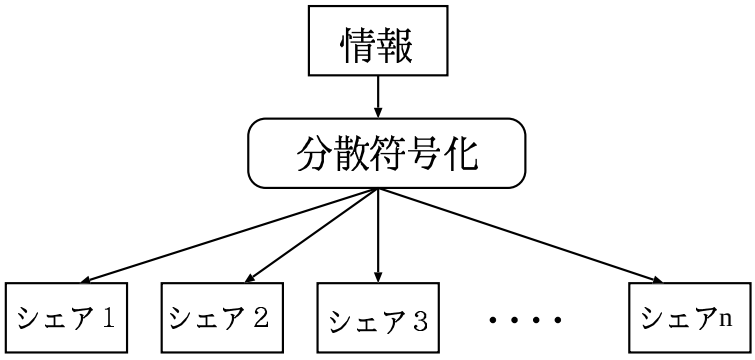

図 2.2　分散符号化

秘密情報を復元したいときは，任意の k 個シェアを集めて処理を行えばもとの情報を得ることができる．この処理を復号化という.

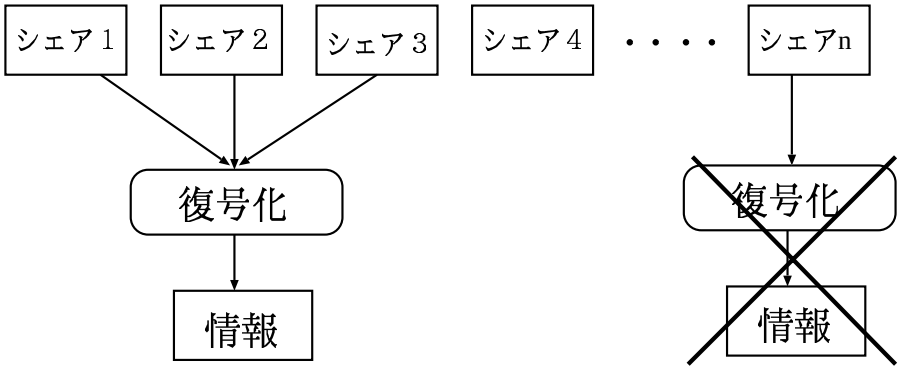

図 2.3　復号化

2.3　特徴と利点

重要な情報を保管する場合，情報の紛失，第三者による情報の破壊や改ざんなどの危険があるが，秘密分散法を用いるとこのような問題を解決することが可能となる．

それぞれのシェアはそれひとつからはもちろんのこと，どの $k-1$ 個を取り出しても，もとの情報は得られないため，シェアが $k-1$ 個以下なら漏洩しても情報は復号されない．すなわち，k 人未満のユーザが集まって秘密情報を復号しようとしても不可能である．また，もとの情報を復号するために，n 個全てのシェアを集める必要はない．したがって，保管しているシェア 1 つに不正にアクセスしたとしても，シェアは符号化されているため情報の内容を知ることはできないし，k 個のシェアが集まらない限り，もとの情報の内容が露見することはない．もちろん，$n-k$ 個までならシェアに破壊および紛失が起きても，残りのシェアからもとの情報は復号できる．秘密分散法は上に述べたような特徴によって，情報の漏洩や紛失に対処することができるという利点を持つ．

2.4　秘密分散法の種類

秘密分散法にはいくつか種類がある．

1. (k, n) しきい値法 [1]

　これまでの話はこのしきい値法をもとに説明した．すなわち，n 人のうち k 人が合意す

ればもとの情報を復号できるが，合意が $k-1$ 人以下では復号できないというものである．k と n の値をしきい値と呼び，この秘密分散法を (k,n) しきい値法と呼ぶ．

一方，シェアを保持する参加者に対してレベルを与えたものもある．例えば，情報を 10 個に分散符号化し， 3 人の課長，7 人の平社員にそれぞれシェアを渡す．復号の際は，課長なら 2 人，平社員なら 6 人集まれば復号可能といった具合になる．

2. 満場一致法 [1]

k と n の値が等しい場合，すなわちシェアを保持している参加者全員が合意したときのみ情報を復号できるというものである．この秘密分散法を満場一致法という．全員のシェアが集まらないと復号できないため，盗聴に対する安全性は高いが，1 個でもシェアが消えてしまうともとの情報を復号できないという欠点がある．

3. (k,d,n) ランプスキーム [1]

(k,n) しきい値法と構造は似ているが，ランプスキームは一部の参加者集合に対して，部分的な情報を入手することを許している．参加者集合がそれぞれシェアを持ち寄ったとき，例えば秘密情報 100bit のうち下位 50bit だけは分かるというように部分的な情報を手にいれることができる．(k,d,n) ランプスキームは

(a) n 人のうち k 人で復号可能

(b) $k-d$ 人以下では復号不可能

(c) $k-d$ 人より多く k 人未満なら部分的情報を入手可能

という条件を満たす．シェアは保管しておかなければならない情報である．ゆえにシェアサイズ[*1]は小さい程良いとするならば，ランプスキームにおけるシェアサイズの定理

$$[\text{シェアサイズ}] \geq [\text{秘密情報のサイズ}]/d$$

より，d の値を大きくすれば良い．d の値を大きくするとは多くの人に部分的情報を漏

[*1] シェアファイルの大きさ

らすことになる．したがってシェアサイズと情報の安全性のバランスをとる必要がある．

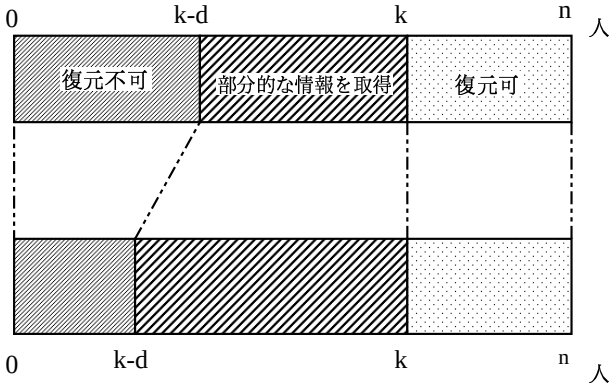

図 2.4　(k, d, n) ランプスキームにおいて d を大きくした場合

上記の 3 種の特徴を考慮した上で，本研究では (k, n) しきい値法を利用する．

満場一致法では，参加者全員のシェアが集まらなければ復号できないため，1 個でもシェアを無くしてしまうと復元できず，情報紛失への耐性がない．つまり，複数箇所にシェアを保管したとき，1 箇所でも災害などの被害を受けたら秘密情報を復号できない．また，ランプスキームのように部分的な情報を与えないものとする．

2.5　素体

秘密分散法は素体 $\boldsymbol{GK}(p)$ 上で実現されるため，実際の実現方法を説明する前にここで少し体論についてふれておく．

2.5.1　体

二つの演算 (加法と乗法) の与えられた集合 $\boldsymbol{K}$ において、任意の $a, b, c \in \boldsymbol{K}$ が以下の条件[*2]を満足するとき、この集合 $\boldsymbol{K}$ を体 (可換体) と呼ぶ.

[*2] ここでは混乱をさけるために $\boldsymbol{K}$ から元 0 を除いたという意味の記号に $*$ を用い，乗法演算子には $\cdot$ を用いる.

体の条件

1. 演算が閉じている

   $a+b \in \boldsymbol{K}$ ，$a \cdot b \in \boldsymbol{K}$

2. 単位元の存在

   加法について $a+0=0+a=a$ を満たすような加法単位元 0(ゼロ元) が存在する．

   また，加法単位元 0 を除いた $\boldsymbol{K}^*$に対して $a \cdot 1 = 1 \cdot a = a$ を満たす乗法単位元 1 が存在する．

3. 逆元の存在

   加法について $a+b=b+a=0$ を満たす加法逆元 $b(=-a)$ が存在する．

   また，加法単位元 0 を除いた $\boldsymbol{K}^*$に対して $a \cdot b = b \cdot a = 1$ を満たす乗法逆元 $b(=a^{-1})$ が存在する．

4. 可換律を満足する

   $a+b=b+a \quad a \cdot b = b \cdot a$

5. 結合律を満足する

   $a+(b+c)=(a+b)+c \quad a(b \cdot c) = (a \cdot b)c$

6. 分配律を満足する

   $a(b \cdot c) = a \cdot b + a \cdot c \quad (b+c)a = b \cdot a + c \cdot a$

体上では四則演算 (加減乗除) が定義される．例えば，実数全体の集合は上記の条件を満たし体をなす．この実数体のように元が無数に存在するものを無限体という．本研究は，そのような無限の元をもつ体ではなく，有限個の元から成る有限体を用いる．有限体はガロア体ともよばれ，記号 $\boldsymbol{GF}(q)$ で表す．q は元の数を表し，位数とよばれる．

2.5.2　素体

位数が素数 p である有限体 $\boldsymbol{GF}(p)$ を素体という．素体上では法を p として剰余演算が行われる．演算を行った後，p で剰余をとればよい．

例として表 2.1 に $\boldsymbol{GF}(5)$ の演算を示す.

表 2.1　$\boldsymbol{GF}(5)$ の演算表

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$*$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

2.6　実現方法

1979 年に創案された Shamir の (k, n) しきい値法 [4] の構築方法について説明する.

分散符号化する方法について記述する.

参加者には，自分に与えられたシェア以外の内容を知ることができないようにシェアを配布する．この n 人の参加者の集合を

$$\boldsymbol{H} = \{H_i \,|\, i = 1, 2, \cdots, n\} \tag{2.1}$$

と表す.

このとき処理は素体 $\boldsymbol{GF}(p)$ (2.5 参照) 上で実行される．ここで p は素数であり，$p \geq n + 1$ を満たす．情報 S も $\boldsymbol{GF}(p)$ の元とする.

さて，実際の処理だが，ディーラは情報 S を分散符号化するために，最大次数 $k - 1$ のランダムな多項式 $f(x)$ を構築する．そして，作成された多項式の定数項を S とする．それぞれの参加者はこの多項式の一点 $(x_i, f(x_i)) = (x_i, w_i)$ を得る．この w_i がシェアである．また，x_i は ID といい，参加者は与えられたシェアとその ID を保管する.

ここで，x_i は各々異なった値でなくてはいけない．したがって，その値に原始根 (2.21 参

照) を用いる.

分散符号化の手順を以下にまとめる.

初期設定

1. $\boldsymbol{GF}(p)$ の元から，0 ではない別々のものを n 個選びだし，各々を $x_i \quad (i = 1, 2, \cdots, n)$ と記述する．このとき，$p \geq n + 1$ であることに注意する.
2. $i = 1, 2, \cdots, n$ に対し，値 x_i を H_i に与える．x_i の値は公開されている.

分散符号化

1. $\boldsymbol{GF}(p)$ から $k - 1$ 個の元 $r_j \quad (j = 1, 2, \cdots, k - 1)$ をランダムに選ぶ.
2. $w_i = f(x_i)$ を以下の式で計算する．$(i = 1, 2, \cdots, n)$

$$f(x_i) = S + \sum_{j=1}^{k-1} r_j {x_i}^j \pmod{p} \tag{2.2}$$

3. シェア w_i を H_i に与える．$(i = 1, 2, \cdots, n)$

次に，秘密情報を復号する方法について記述する.

k 人の参加者 $H_{i_1}, H_{i_2}, \cdots, H_{i_k}$ が集まって S を求めたいとする．参加者はそれぞれが x_i と w_i の値を持っており，ディーラによって決められた多項式 $f(x)$ により，

$$f(x_{i_j}) = w_{i_j} \tag{2.3}$$

が得られることを知っている．$f(x)$ の定義から最大次数は $k - 1$ であるから $f(x)$ は

$$f(x) = S + r_1 x + r_2 x + \cdots + r_{k-1} x^{k-1} \tag{2.4}$$

と表される．ここで係数 $r_1, r_2, \cdots, r_{k-1}$ は分散符号化を行なった $\boldsymbol{GF}(p)$ の元である．S は分散符号化された情報であり，$\boldsymbol{GF}(p)$ の元である．k 人が持ち寄った IDx_i とシェア w_i 及び多項式 $f(x)$ から，k 個の未知数 $S, r_1, r_2, \cdots, r_{k-1}$ の線形方程式が k 個得られる．この線形連立方程式は

$$
\begin{aligned}
S + r_1 x_{i_1} + r_2 x_{i_1} + \cdots + r_{k-1} {x_{i_1}}^{k-1} &= w_{i_1} \\
S + r_1 x_{i_2} + r_2 x_{i_2} + \cdots + r_{k-1} {x_{i_2}}^{k-1} &= w_{i_2} \\
S + r_1 x_{i_3} + r_2 x_{i_3} + \cdots + r_{k-1} {x_{i_3}}^{k-1} &= w_{i_3} \\
&\vdots \\
S + r_1 x_{i_k} + r_2 x_{i_k} + \cdots + r_{k-1} {x_{i_k}}^{k-1} &= w_{i_k}
\end{aligned}
\tag{2.5}
$$

となり，これの行列で表すと

$$
\begin{bmatrix}
1 & x_{i_1} & {x_{i_1}}^2 & \cdots & {x_{i_1}}^{k-1} \\
1 & x_{i_2} & {x_{i_2}}^2 & \cdots & {x_{i_2}}^{k-1} \\
1 & x_{i_3} & {x_{i_3}}^2 & \cdots & {x_{i_3}}^{k-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{i_k} & {x_{i_k}}^2 & \cdots & {x_{i_k}}^{k-1}
\end{bmatrix}
\begin{bmatrix}
S \\ r_1 \\ r_2 \\ \vdots \\ r_{k-1}
\end{bmatrix}
=
\begin{bmatrix}
w_{i_1} \\ w_{i_2} \\ w_{i_3} \\ \vdots \\ w_{i_k}
\end{bmatrix}
\tag{2.6}
$$

となる．これらの線形方程式は互いに独立であるため一意的に解が求まり，S の値が明かになる．同時に $k-1$ 個の乱数 $r_1, r_2, \cdots, r_{k-1}$ の値も求められる．これらの線形方程式は独立でなければ解を求めることはできない．この線形方程式が常に独立になることは証明されている (付録 A 参照)．なお，式 (2.6) における係数行列を $\boldsymbol{G}$ とおく．

$$
\boldsymbol{G} =
\begin{bmatrix}
1 & x_{i_1} & {x_{i_1}}^2 & \cdots & {x_{i_1}}^{k-1} \\
1 & x_{i_2} & {x_{i_2}}^2 & \cdots & {x_{i_2}}^{k-1} \\
1 & x_{i_3} & {x_{i_3}}^2 & \cdots & {x_{i_3}}^{k-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & x_{i_k} & {x_{i_k}}^2 & \cdots & {x_{i_k}}^{k-1}
\end{bmatrix}
\tag{2.7}
$$

復号化の手順を以下にまとめる．

初期設定 k 人が各々の持つ ID x_{i_j} とシェア w_{i_j}　$(1 \leq j \leq k)$ を持ち寄る．

復号化

1. 以下の式に x_{i_j} と w_{i_j} を代入．

$$
f(x) = S + r_1 x + r_2 x^2 + \cdots + r_{k-1} x^{k-1} = w_{i_j} \tag{2.8}
$$

2. 1 を k 人分繰り返す (線形方程式を k 個作成する)
3. k 個の線形方程式から分散符号化されたもとの情報 S，分散符号化のときにランダムに $k-1$ 個選ばれた $\boldsymbol{GF}(p)$ の元 $r_1, r_2, \cdots, r_{k-1}$ を求める．S の内容が得られる．

分散符号化と復号化の簡単な例を以下に示す.

分散符号化の例

p=7，k=3，n=5 とする．これは $\boldsymbol{GF}(7)$ 上で $(3,5)$ しきい値法を実行することを意味する．公開されている $x_1, x_2, \cdots, x_5$ の値は原始根 3 を用いて．$x_i = 3^i \pmod 7$ を与える．この場合，各々の値は $x_1 = 3, x_2 = 2, x_3 = 6, x_4 = 4, x_5 = 5$ となる．係数行列 $\boldsymbol{G}$ は

$$\boldsymbol{G} = \begin{bmatrix} 1 & 3 & 3^2 \\ 1 & 2 & 2^2 \\ 1 & 6 & 6^2 \\ 1 & 4 & 4^2 \\ 1 & 5 & 5^2 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \\ 1 & 2 & 4 \\ 1 & 6 & 1 \\ 1 & 4 & 2 \\ 1 & 5 & 4 \end{bmatrix} \pmod 7 \tag{2.9}$$

となる．$S = 3$ とし，$k-1$ 個すなわち 2 個の乱数を $r_1 = 1, r_2 = 1$ とする．式 (2.2) を用いてシェアを求める.

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 \\ 1 & 2 & 4 \\ 1 & 6 & 1 \\ 1 & 4 & 2 \\ 1 & 5 & 4 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 5 \end{bmatrix} \pmod 7 \tag{2.10}$$

保管する (ID，シェア) は $(3,1), (2,2), (6,3), (4,2), (5,5)$ となる.

復号化の例

上記で分散符号化した 3 個のシェアから S の値を求める．公開されている $x_1, x_2, \cdots, x_5$ の値は $x_1 = 3, x_2 = 2, x_3 = 6, x_4 = 4, x_5 = 5$ であった．3 人の参加者 H_1, H_3, H_4 がそれ

ぞれのシェア $w_1 = 1, w_3 = 3, w_4 = 2$ を集めると式 (2.4) より，多項式 $f(x)$ は

$$f(x) = S + r_1 x + r_2 x^2 = w \tag{2.11}$$

となる．集まった 3 組の ID とシェア $(x_1, w_1) = (3, 1), (x_3, w_3) = (6, 3), (x_4, w_4) = (4, 2)$ より $f(x_1), f(x_3), f(x_4)$ を求める．まず参加者 H_1 のシェアによって求められる多項式は，$x_1 = 3$ を用いて，

$$f(3) = S + 3r_1 + 3^2 r_1 = 1 \tag{2.12}$$

となる．同様に H_3, H_4 のシェアから 2 つの多項式が求められる．

$$f(6) = S + 6r_1 + 6^2 r_2 = 3 \tag{2.13}$$

$$f(4) = S + 4r_1 + 4^2 r_2 = 2 \tag{2.14}$$

こうして求められた 3 つの多項式を行列で表すと

$$\begin{bmatrix} 1 & 3 & 3^2 \\ 1 & 6 & 6^2 \\ 1 & 4 & 4^2 \end{bmatrix} \begin{bmatrix} S \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix} \tag{2.15}$$

となる．演算はすべて $\boldsymbol{GF}(7)$ 上で行われる．この連立方程式を解くと，

$$\begin{bmatrix} S \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix} \tag{2.16}$$

が求められる．よって $S = 3$ が得られた．

2.7　問題点と改良

参加者が保持する ID となる $x_i \quad (i = 1, 2, \cdots, n)$ に原始根を用いるため，これを求める計算が必要である．また，素体 $\boldsymbol{GF}(p)$ 上で実行する際の素数 p を求めなければいけない．しかし，一般的に原始根および素数を求めるには膨大な計算時間が必要である．そのため，本研究では拡大体上で秘密分散法を実現することを考える．そこで次に文献 [12] から拡大体の有用性を記述する．

2.8　拡大体の有用性

2.8.1　拡大体

位数 q が素数 p の m 乗 $(m \geq 2)$ であるような有限体を考える．有限体 $\boldsymbol{GF}(p^m)$ は素体の場合と同じような方法では作ることができない．

$q = 2^2 = 4$ の場合を例に挙げてみる．$\bmod 4$ では 2 の乗法逆元 2^{-1} が存在しない．これは 2 による除算ができないことを意味している．

実数体から複素数体を導く方法を考えてみる．複素数体は実数体に虚数単位 i を付加すると同時に体となるために必要な元 $a + bi$ を全て実数体に付加することで作られている．このとき，最初に付加した虚数単位 i は，実数体上の既約多項式 $x^2 + 1 = 0$(これ以上は因数分解が不可能な最小の多項式) の解 (根) である．このように，体 $\boldsymbol{F}$ に $\boldsymbol{F}$ 上の既約多項式の解を付加して作られる体を拡大体という．なお，$\boldsymbol{F}$ の元を係数とする多項式を $\boldsymbol{F}$ 係数多項式または $\boldsymbol{F}$ 上多項式という．

複素数体と同様に拡大体 $\boldsymbol{GF}(p^m)$ も素体 $\boldsymbol{GF}(p)$ を拡大して作ることができる．$\boldsymbol{GF}(p)$ に $\boldsymbol{GF}(p)$ 上の m 次既約多項式の解を 1 つ付加して体を作ることによって m 次拡大体 $\boldsymbol{GF}(p^m)$ が得られる．拡大体 $\boldsymbol{GF}(p^m)$ は，

$$a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \tag{2.17}$$

の形をもつ $m - 1$ 次以下の多項式の元をもつ．体を作る際の既約多項式を $p(x)$ と表す．

2.8.2　2 の拡大体

現在のデジタルコンピュータは内部処理を 2 進数で行う．$\boldsymbol{GF}(2)$ はふたつの元 $0, 1$ をもつ体である．そのために実用上は $\boldsymbol{GF}(2^m)$ のような体があると便利である．

$\boldsymbol{GF}(2)$ から $\boldsymbol{GF}(2^2)$ を導いてみる．まず，$\boldsymbol{GF}(2)$ 上の元 $0, 1$ を係数に持つ既約多項式 $x^2 + x + 1$ を用意する．この多項式の解を α とし，これを $\boldsymbol{GF}(2)$ に付加して，さらに体になるために必要な元を付加する．ここで，この体は $\boldsymbol{GF}(2)$ の元である 0 と 1 を含まなくてはならない．そして α も元である．α を付加した集合が体になるためには演算に対して閉じ

ていなくてはいけない．すなわち，α のべき $(\alpha^2, \alpha^3, \cdots, \alpha^{2^m-1})$ を集合に付加する必要がある．α は x^2+x+1 の解としたので

$$\alpha^2 + \alpha + 1 = 0 \tag{2.18}$$

を満足する．これは変形して

$$\alpha^2 = -\alpha - 1 = \alpha + 1 \tag{2.19}$$

となり，この $\alpha^2 = \alpha + 1$ を基に α のべきが次のように求められる．ここで，$\boldsymbol{GF}(2)$ 上の多項式なので，係数は 1 もしくは 0 にしかなりえないため，$1+1=0$ に注意する．

$$\begin{aligned}
\alpha^0 &= \quad 1 \\
\alpha^1 &= \alpha^0 \cdot \alpha^1 = 1 \cdot \alpha = \alpha \\
\alpha^2 &= \alpha + 1 \\
\alpha^3 &= \alpha^2 \cdot \alpha = (\alpha+1)\alpha = \alpha^2 + \alpha = (\alpha+1) + \alpha = 1 = \alpha^0 \\
&\vdots
\end{aligned} \tag{2.20}$$

$\alpha^3 = 1$ のように，別の元と同じ値になるものは体の元として含む必要がないため，この体の元は $0, 1, \alpha, \alpha^2$ の 4 つになる．このようにして作られた体が $\boldsymbol{GF}(2)$ の 2 次拡大体 $\boldsymbol{GF}(2^2)$ である．実際に，この体が四則演算の可能な体になっているかどうかは，加法表・乗法表を作成することにより確かめられる．表 2.2 に $\boldsymbol{GF}(2^2)$ の加法表と乗法表を示す．

表 2.2　$\boldsymbol{GF}(2^2)$ の演算表

$+$	0	1	α	α^2
0	0	1	α	α^2
1	1	0	α^2	α
α	α	α^2	0	1
α^2	α^2	α	1	0

$*$	0	1	α	α^2
0	0	0	0	0
1	0	1	α	α^2
α	0	α	α^2	1
α^2	0	α^2	1	α

同様に m 次拡大体 $\boldsymbol{GF}(2^m)$ について考える．$0, 1$ のみを係数とする m 次既約多項式の

解を α とすると，m 次拡大体 $\boldsymbol{GF}(2^m)$ の元は $0, \alpha^0, \alpha^1, \alpha^2, \cdots, \alpha^{2^m-2}$ で構成され，α は

$$\alpha^{2^m-1} = 1 \tag{2.21}$$

を満たす.

α のベキが全て異なる元となるとき α を原始根という．$\alpha^2, \alpha^3, \cdots, \alpha^{2^m-2}$ のように原始根 α のべきで $\boldsymbol{GF}(2^m)$ の元を表すことをべき表現という．べき表現を用いると $\boldsymbol{GF}(2^m)$ における乗算が指数法則 (付録 B 参照) を用いて容易に実行できる．つまり，$\alpha^i, \alpha^j (0\ leqi \leq 2^{m-2}, 0 \leq j \leq 2^{m-2})$ の積を

$$\alpha^i \alpha^j = \alpha^{i+j} \tag{2.22}$$

として計算できる．ただし，$i + j$ は $2^m - 1$ で剰余をとる.

m 次既約多項式を

$$p(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_1 x + 1 \tag{2.23}$$

のように表す．係数 $f_1, \cdots, f_{m-1}$ は $\boldsymbol{GF}(2)$ の元である．$F(\alpha) = 0$ より α^m を

$$\alpha^m = f_{m-1}\alpha^{m-1} + \cdots + f_1\alpha + 1 \tag{2.24}$$

のように $1, \alpha, \cdots, \alpha^{m-1}$ の 1 次結合で表すことができる．これを用いて α^i $(i = 0, 1, \cdots, 2^m - 2)$ を $\boldsymbol{GF}(2)$ の元を係数とする $1, \alpha, \cdots, \alpha^{m-1}$ の 1 次結合で

$$\alpha^i = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0 \tag{2.25}$$

のように表すことができる．これを用いれば加算を簡単に実行できる．$1, \alpha, \cdots, \alpha^{m-1}$ の係数同士を $\boldsymbol{GF}(2)$ 上で加えればよい．この加算は，式 (2.25) の係数のみを並べた

$$(a_{m-1}, \cdots, a_1, a_0) \tag{2.26}$$

という $\boldsymbol{GF}(2)$ 上の m 次元ベクトルの加算と同じであり，拡大体上の加算を簡単に行うために，元 α^i を $\boldsymbol{GF}(2^m)$ 上の m 次元ベクトルとして表しておくとよい．これを $\boldsymbol{GF}(2^m)$ の元のベクトル表現という．元 0 のベクトルは $(0, 0, \cdots, 0)$ である.

$\boldsymbol{GF}(2^m)$ の減算は，加算と同様にできる．なぜならば，$\boldsymbol{GF}(2^m)$ の任意の元 x の加法に関する逆元 $-x$ は自分自身と一致する．

このように $\boldsymbol{GF}(2^m)$ の加算や減算はベクトル表現を用いて，乗算や除算はべき表現を用いて行うことができる．このべき表現とベクトル表現は，$\boldsymbol{GF}(2^m)$ を作る際の既約多項式をもとにして対応づけられる．対応表の例を表 2.3 に示す．表 2.3 は既約多項式 x^4+x+1 を用いて $\boldsymbol{GF}(2^4)$ を作ったときの対応表である．

表 2.3　$\boldsymbol{GF}(2^4)$ の対応表

べき表現	多項式表現				ベクトル表現
0				0	(0,0,0,0)
α^0				1	(0,0,0,1)
α^1			α		(0,0,1,0)
α^2		α^2			(0,1,0,0)
α^3	α^3				(1,0,0,0)
α^4			α	$+1$	(0,0,1,1)
α^5		α^2	$+\alpha$		(0,1,1,0)
α^6	α^3	$+\alpha^2$			(1,1,0,0)
α^7	α^3		$+\alpha$	$+1$	(1,0,1,1)
α^8		α^2		$+1$	(0,1,0,1)
α^9	α^3		$+\alpha$		(1,0,1,0)
α^{10}		α^2	$+\alpha$	$+1$	(0,1,1,1)
α^{11}	α^3	$+\alpha^2$	$+\alpha$		(1,1,1,0)
α^{12}	α^3	$+\alpha^2$	$+\alpha$	$+1$	(1,1,1,1)
α^{13}	α^3	$+\alpha^2$		$+1$	(1,1,0,1)
α^{14}	α^3			$+1$	(1,0,0,1)

2.8.3　2 の拡大体上の演算

コンピュータでの効率的な計算処理を行うためには，演算を 2 進数で表現することが望ましい．$\boldsymbol{GF}(2^m)$ では，0, 1 のみで表現が可能となる．したがって，元は長さ m のビットベクトル $(a_{m-1}, \cdots, a_1, a_0)$ として表現できる．たとえば，ビットベクトル $(1, 1, 0, 0, 1)$ は $\boldsymbol{GF}(2^m)$ における多項式 $x^4 + x^3 + 1 (1 \cdot x^4 + 1 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x + 1)$ と対応している．すなわち，ベクトル表現と多項式表現が対応する．

$\boldsymbol{GF}(2^m)$ における演算は，$\boldsymbol{GF}(p)$ における演算よりも空間と時間の面でより効率的である．p を $2^{m-1} < p < 2^m$ であるような素数であるとすると，$\boldsymbol{GF}(p)$ の元は同様に長さ m のビットベクトルとして表現できる．この対応の仕方は 2 進数表現から 10 進数表現への変換と同様に行える．たとえば，$(1, 1, 0, 0, 1)$ は整数 $2^4 + 2^3 + 1 = 25$ と対応するようにする．全ての 2^m ビットのベクトルは，$\boldsymbol{GF}(2^m)$ の元に対応するが，$\boldsymbol{GF}(p)$ については，あてはまらないことがわかる．$\boldsymbol{GF}(2^m)$ では，同じ空間を用いて $\boldsymbol{GF}(p)$ で表すよりも多くの元を表現できる．

次に具体的な数を用いて $\boldsymbol{GF}(2^m)$ と $\boldsymbol{GF}(p)$ における演算方法に関して両者の比較を行う．$\boldsymbol{GF}(2^m)$ の元 a, b をベクトル表現を用いて $a = (a_{m-1}, \cdots, a_0)$，$b = (b_{m-1}, \cdots, b_0)$ とおく．$\boldsymbol{GF}(2^m)$ においての加算は，各ビットの排他的論理和 (表 2.4 参照) をとればよい．ここで，$c = a + b$ とすると

$$c_i = a_i \oplus b_i \quad (i = 0, 1, \cdots, m-1) \tag{2.27}$$

のときに $c = (c_{m-1}, \cdots, c_0)$ となる．

以下に例を示す．$a = (1, 0, 1, 0, 1)$ および $b = (0, 1, 1, 0, 0)$ とする．$\boldsymbol{GF}(2^m)$ において，$c = a + b$ は次のように計算される．

$$\begin{aligned} a &= 10101 \\ \underline{b &= 01100} \\ c &= 11001 \end{aligned} \tag{2.28}$$

$\boldsymbol{GF}(p)$ において $p = 31$ の場合の $a + b$ の計算を行うとする．加算では繰り上がりがおこ

表 2.4　排他的論理和の真理値表

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

り，31 を法とする結果を得るために除算が必要になる．

1.　a と b を足す

$$\begin{aligned} a &= 10101 \quad (21) \\ \underline{b &= 01100 \quad (12)} \\ c &= 100001 \quad (33) \end{aligned} \tag{2.29}$$

2.　31 で割って剰余をとる

$$c = 00010 \quad (2) \tag{2.30}$$

$\boldsymbol{GF}(2^m)$ において $a * b$ を計算するとき，積 $a * b$ は既約多項式 $p(x)$ で除される必要がある．積 $d = a * b$ は多項式の和で表される．

$$d = \sum_{i=0}^{m-1} (a_i * b) x^i \pmod{p(x)} \tag{2.31}$$

$$a * b = \begin{cases} b = b_{m-1}x^{m-1} + \cdots + b_0 & a_i = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

以下に例を示す．

$a = (1, 0, 1)$ とする．既約多項式 $p(x) = x^3 + x + 1$　(2 進数の 1011) を用いて作られた $\boldsymbol{GF}(2^3)$ において a を 2 乗するとき，積 $d = a * a$ は以下のように計算される．

2.8　拡大体の有用性

1. $a * a$ を計算する

$$\begin{array}{ccccc} & & 1 & 0 & 1 \\ & & 1 & 0 & 1 \\ \hline & & 1 & 0 & 1 \\ & 0 & 0 & 0 & \\ 1 & 0 & 1 & & \\ \hline 1 & 0 & 0 & 0 & 1 \end{array} \tag{2.33}$$

2. $p(x) = (1, 0, 1, 1)$ で割る

$$\begin{array}{rcccc} & & & & 1 \quad 0 \\ \hline 1 \quad 0 \quad 1 \quad 1 &) & 1 & 0 & 0 \quad 0 \quad 1 \\ & & 1 & 0 & 1 \quad 1 \\ \hline & & & & 1 \quad 1 \quad 1 \quad = d \end{array} \tag{2.34}$$

a を $\boldsymbol{GF}(7)$ において 2 乗するとき，演算は同様に行うが，乗除での加算と減算において繰り上がりが必要になる．

次に $\boldsymbol{GF}(2^m)$ における $a * b$ の計算方法を示す．

$a = (1, 1, 1), b = (1, 0, 0)$ とする．積 $d = a * b$ は既約多項式 $p(x) = 1011 \quad (x^3 + x + 1)$ を用いて作られる．

1. $a * b$ を計算する

$$\begin{array}{ccccc} & & 1 & 1 & 1 \\ & & 1 & 0 & 0 \\ \hline & & 0 & 0 & 0 \\ & 0 & 0 & 0 & \\ 1 & 1 & 1 & & \\ \hline 1 & 1 & 1 & 0 & 0 \end{array} \tag{2.35}$$

2. $p(x) = (1, 0, 1, 1)$ で割る

$$\begin{array}{rcccc} & & & & 1 \quad 1 \\ \hline 1 \quad 0 \quad 1 \quad 1 &) & 1 & 1 & 1 \quad 0 \quad 0 \\ & & 1 & 0 & 1 \quad 1 \quad 0 \\ \hline & & & 1 & 0 \quad 1 \quad 1 \\ & & & 1 & 0 \quad 1 \quad 1 \\ \hline & & & & 1 \quad = d \end{array} \tag{2.36}$$

$\boldsymbol{GF}(2^m)$ において b を a で除するためには，a^{-1} と表記する a の逆数 a^{-1} を計算し，b に a^{-1} を乗ずる．

これらから，$\boldsymbol{GF}(2^m)$ における多項式法は $\boldsymbol{GF}(p)$ における整数算法よりも効率的であるといえる．なぜならば繰り上がりがなく，除算は加算や減算を要しないからである．

第 3 章

拡大体上の秘密分散法

3.1 概要

拡大体 $\boldsymbol{GF}(2^m)$ 上で秘密分散法を実現する．$\boldsymbol{GF}(2^m)$ 上で効率よく秘密分散法を行なうために実行の際の四則演算の効率化および高速化を目指す．ここで，$\boldsymbol{GF}(2^m)$ 上の減法の演算結果は加法の演算結果と同じであるため，減法については考えない．したがって加法・乗法・除法の 3 つの演算について考える．

文献 [12] では．四則演算について各演算表を作成することにより表引きのみで演算結果を得ることができた．表引きとは，演算表の 2 つの元 α^i, α^j の交点を見れば，その演算結果が得られるものである．

演算表は全ての値について各演算結果を記述している，すなわち，$\boldsymbol{GF}(2^m)$ の位数 2^m から各演算表は $2^m \times 2^m$ の 2 次元配列で表現される．つまり，その要素数は 2^{2m}．そのため，拡大次数 m が増加すると指数関数的に容量が大きくなり，加法・乗法・除法の全ての演算に演算表を用いることは望ましくない．

本研究では，演算表の縮小のために，各々の演算の特徴を考慮し，直接計算するもの，表引きするものと場合分けで計算処理を行う．

演算表を作成するアルゴリズムは，文献 [12] のものをそのまま用いるが，演算表に関しては最終的に保管する指針である．保管が実現すれば，演算表作成時の作成時間は秘密分散法を実行するための計算時間には含まれない．そのため，本研究が指す計算処理には作成時間は含まない．

本小節では前半部分で各演算法について説明し，後半部分で拡大体上での (k, n) しきい値

法の実現方法を説明する．

3.2　演算の定義

拡大体上では，素体上で行われているような演算をすることができないため，演算の定義をする必要がある．

本研究は，基本的に 2 項演算の内容により場合分けを行い演算方法を確定する．これはどの演算においても項に 0 が含まれるなど演算結果が明らかなものがあるためである．乗法および除法においては，べキ表現における指数法則をもとに計算処理を行う．加法については，加法表を作成し表引きと直接計算の両方を用いて計算を行う．

このため，2 項演算の引数はべき表現で与えることとする．

3.2.1　べき表現-ベクトル表現対応表作成

第 2 章からべき表現で表された $\boldsymbol{GF}(2^m)$ の元は，m 桁のベクトル表現に変換することが可能であった．この例を示す．

$$\begin{aligned} \alpha^0 &\rightarrow (0,0,1) \\ \alpha^1 &\rightarrow (0,1,0) \\ &\vdots \end{aligned} \tag{3.1}$$

α^m には，$\boldsymbol{GF}(2^m)$ の法である既約多項式 $p(x)$ の係数のみを取り出して並べたベクトル表現を対応させる．α^i は $\alpha^{i-1} * \alpha^1$ を計算して求める．対応表のベクトル表現は配列で表しており，加法表作成時の

1. べき表現をベクトル表現に変換

   (例)　$\alpha^0 \rightarrow (0,0,1)$
2. ベクトル表現をべき表現に変換

   (例)　$(1,0,0) \rightarrow \alpha^2$

の作業を行うために，べき表現とベクトル表現を対応させた表が必要となる．対応表の参照方法は 2 通りある．

1. べき表現を指定すると，対応するベクトル表現を返す.
2. ベクトル表現を指定すると，対応するべき表現を返す.

ベクトル表現をべき表現に変換する場合は別の表を用いる．べき表現-ベクトル表現対応表は，ベクトル表現を 2 進数値の配列で表している．この表を用いてベクトル表現をべき表現に直す場合，数値を比較する段階で処理が複雑になってしまう．ベクトル表現を指定してそれに対応するべき表現を求めるとき，指定されたベクトル表現と表の中のベクトル表現が等しいところのべき表現を返す仕組みになっている．しかし，このとき配列と配列が等しいかどうかの判断が必要になる．したがって，対応表のベクトル表現を配列だけでなく，整数型で表した整数型対応表を作成する．表 3.1 に作成される対応表の例を示す.

整数型対応表を用いてベクトル表現をべき表現に変換する手順は以下のようになる.

1. 配列で表されたベクトル表現値 x を整数型ベクトル表現 x' に変換する.

   (例)$(0, 1, 0, 0, 1) \rightarrow 1001$
2. x' と等しい整数型対応表の整数型ベクトル表現を探す.
3. $x' = y$ となる y がみつかれば，y に対するべき表現を返す.

対応表を作成する手順を示す.

べき表現-ベクトル表現対応表作成手順

1. m 次既約多項式 $p(x)$ を定める.
2. α^m を $p(x)$ と対応させる.

$$p(x) = b_m x^m + b_{m-1} x^{m-1} + \cdots + b_1 x + b_0 \tag{3.2}$$

とすると，

$$\alpha^m = b_{m-1} x^{m-1} + \cdots + b_1 x + b_0 \tag{3.3}$$

とおく．したがって，ベクトル表現では，

$$\alpha^m \rightarrow (b_{m-1}, \cdots, b_1, b_0) \tag{3.4}$$

3. $\alpha^i \quad (i = 1, 2, \cdots, m-1)$ を $(c_{m-1}, \cdots, c_1, c_0)$ と対応させる．ただし，

$$c_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \tag{3.5}$$

4. $\alpha^{i-1} * \alpha^1$ の値を求め，α^i へ格納する．

5. 配列で表されたベクトル表現を整数型に変換する．

6. 変換した値をべき表現-整数型-ベクトル表現対応表に格納する．

表 3.1 $\boldsymbol{GF}(2^3)$ の対応表

べき表現	多項式表現			ベクトル表現
α^0			1	$(0,0,1)$
α^1		α		$(0,1,0)$
α^2	α^2			$(1,0,0)$
α^3		α	$+1$	$(0,1,1)$
α^4	α^2	$+\alpha$		$(1,1,0)$
α^5	α^2	$+\alpha$	$+1$	$(1,1,1)$
α^6	α^2		$+1$	$(1,0,1)$

3.2.2 加法

加法は，場合分けで表引きと直接計算を行う．加法表の作成については，秘密分散法を行う前に対応表を用いて作成しておく．本研究で用いる加法表は，文献 [12] の加法表を縮小したものである．本研究は次のような加法の特徴を考慮して加法表を縮小する．表 3.2 に $2^m \times 2^m$ サイズで作成される加法表の例を示す．

表 3.2　$\boldsymbol{GF}(2^3)$ の加法表

$+$	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
0	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	α^0	0	α^3	α^5	α^1	α^5	α^4	α^2
α^1	α^1	α^3	0	α^4	α^0	α^2	α^6	α^5
α^2	α^2	α^6	α^4	0	α^5	α^1	α^3	α^0
α^3	α^3	α^1	α^0	α^5	0	α^6	α^2	α^4
α^4	α^4	α^5	α^2	α^1	α^6	0	α^0	α^3
α^5	α^5	α^4	α^6	α^3	α^2	α^0	0	α^1
α^6	α^6	α^2	α^5	α^0	α^4	α^3	α^1	0

1. $i = j$ のとき，$\alpha^i + \alpha^j = 0$
2. 0 にどんな元を $+$ しても値は変わらない
3. $\alpha^i + \alpha^j = \alpha^j + \alpha^i (i \neq j)$

1 は $\boldsymbol{GF}(2)$ 上の多項式なので，係数は $0, 1$ のみで，2 は 0 と合同であることから導ける．

$$\alpha^i + \alpha^i = (1 + 1)\alpha^i = 0 \tag{3.6}$$

2 が成り立つことは一般的に明らかである．

このため，$1, 2$ の場合は，容易に演算結果が求められると考え，直接計算を行う．3 の場合は表引きを行う．加法表には，3 の特徴を考慮して $\alpha^i + \alpha^j (i > j)$ の演算結果のみを記述する．これは $\alpha^i + \alpha^j$ の演算結果がわかれば項が反転しただけの $\alpha^j + \alpha^i$ の演算結果も同じなので加法表を参照する上で片方の演算結果だけわかればさしつかえない．

表 3.2 を例にどれだけ加法表が縮小されるかを見てみる．

$1, 2$ の直接計算するものは加法表に記述する必要がないため，1 から α^i 行 0 列と 0 行 α^i 列の要素を消去する．このとき，消去される要素数は $2^{2m} - 2 * 2^m$ となる．

2 から $i = j$，つまり，対角線要素を消去する．このとき，消去される要素数は $(2^{2m} - 2 * 2^m) - 2^m$ である．

3 から $\alpha^i + \alpha^j (j \geq i)$ の要素を消去する．このとき，消去される要素数は $\frac{((2^{2m} - 2 * 2^m) - 2^m)}{2}$ となる．

以上から 2^{2m} だった要素数が半分以下の $\sum_{x=1}^{2^m-2} x$ まで減った．具体的には表 3.3 のようである．

表 3.3　加法表の要素数の比較

拡大次数 m	文献 [12]	本研究
4	256	105
6	4096	1953
8	65536	32385
12	16777216	8382465

実際の例を表で見てみると，表 3.2 の加法表は以下の表 3.4 のように変わる．

表 3.4　縮小した $\boldsymbol{GF}(2^3)$ の加法表

$+$	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0		α^3	α^6	α^1	α^5	α^4	α^2
α^1			α^4	α^0	α^2	α^6	α^5
α^2				α^5	α^1	α^3	α^0
α^3					α^6	α^2	α^4
α^4						α^0	α^3
α^5							α^1

ここで，実際の加算演算表の要素はべき表現をベクトル表現に変換し，加算を行う．その結果をまたベクトル表現に変換する．

実際には

$$\begin{aligned} \alpha^0 + \alpha^1 &\rightarrow 001 + 010 \\ &= 011 \\ &\rightarrow \alpha^3 \end{aligned} \tag{3.7}$$

のように求める.

こうして求めた値を加法表に格納していく．以下に加法表を作成する手順を示す.

加法表作成手順

1. $i = 0$ から $2^m - 2$ まで 1 を繰り返す.
2. $j = 1$ から $2^m - 1$ まで 2 を繰り返す.
3. $\alpha^i + \alpha^j$ を計算する.
   （a）対応表を用いて α^i と α^j をベクトル表現 b_i, b_j に変換する.
   （b）b_i と b_j の各桁の排他的論理和をとる.
   （c）求めたベクトル表現を整数型対応表を用いてべき表現へ変換する.
   （d）べき表現を α^i 行 α^j 列に代入する.

以上を用いて加法の演算は次の手順にした.

加法実行手順

1. どちらかの項が 0 を含んでないか確かめる.
2. 1 を満足したら 0 でない方の元を演算結果として返す.
   満足しない場合 3 へ
3. 元が同じか確かめる.
4. 3 を満足したら 0 を演算結果として返す
   満足しない場合 5 へ
5. 加法表から $\alpha^i + \alpha^j$ の値を取り出す.
   演算表に値が存在すれば，その値を演算結果として返し，存在しなければ項を反転して再び値を取り出す.

3.2.3　乗法・除法

拡大体上の乗法は 2.8.2 に述べたように，べき表現を利用して簡単に計算できる．元 α^i と α^j の積は指数法則を用いて，

$$\alpha^i * \alpha^j = \alpha^{i+j} \tag{3.8}$$

で計算される (付録 B 参照)．$i+j$ は 2^m-1 で剰余をとる．

$\boldsymbol{GF}(2^m)$ において，全ての元は 0 を掛けると 0 になる．

$$\alpha^i * 0 = 0 \quad (i = 0, 1, \cdots, 2^m - 2) \tag{3.9}$$

よって乗法は，0 を含む場合と含まない場合で場合分けをする．

乗法実行手順
1. 片方もしくは両方の元が 0 でないか調べる．
2. 1 を満足したら 0 を演算結果として返す．
満足しない場合 3 へ
3. $\alpha^i * \alpha^j$ を計算する．
$(i+j \pmod{2^m-1})$ を計算する)

除法は乗法から以下のように導くことができる．

$$\alpha^i / \alpha^j = \alpha^i * \frac{1}{\alpha^j} = \alpha^i * \alpha^{-j} \tag{3.10}$$

もちろん，このときも $i+(-j)$ は 2^m-1 で剰余をとる．すなわち，除法 a/b は，割る数 b の逆元 b^{-1} を用いた乗法 $a * b^{-1}$ と定義する．

ここで，乗法逆元についてふれる．乗法逆元は $a*b = b*a = 1$ を満たす b であった．実際を考えると α^i の乗法逆元とは演算結果が 1 であるわけだから

$$\alpha^i * \frac{1}{\alpha^i} = 1 \tag{3.11}$$

より $\frac{1}{\alpha^i}$ と表現される．つまり $(\alpha^i)^{-1}$ が α^i の逆元である．しかし，$\boldsymbol{GF}(2)$ の元というのは一般的にべきが負であるものはないように見える．これは乗法逆元 $(\alpha^i)^{-1}$ は $\alpha^{(m-1)-i}$ と表現され常に $i \leq m-1$ が成立しているためである．

除法実行手順

1. 片方もしくは両方の元が 0 でないか調べる.
2. 1 を満足したら 0 を演算結果として返す.

   満足しない場合 3 へ
3. $\alpha^i * \alpha^{-j}$ を計算する.

   $(i-j \pmod{2^m-1})$ を計算する)

3.3　対応表の効率化

3.2 において対応表は加算表作成時にしか，参照されないことがわかっている．概要 3.1 でも述べた通り，加法演算表は保管する指針である．つまり，加法演算表の保管が実現すると対応表は再度演算表を作成するまでは必要でなくなる．そこで，保管が実現したと仮定して対応表に関する効率化のアルゴリズムをいくつか説明する.

一時的保管アルゴリズム

これは今回のように演算表を作成して加法表を作成する．しかし，対応表は加算表が完成すれば，その容量というのはむだになる．そこで，この対応表は一時保存領域のような場所に保管をしておいて加法表が完成したら対応表を消去するようなアルゴリズムがあれば便利である．まとめると次のようになる.

1. 対応表を作成する
2. 対応表を用いて加法表を作成する
3. 加法表が完成すると対応表を削除する

変換作業を用いた加法表作成アルゴリズム

対応表の作業は次のふたつであった.

1. べき表現をベクトル表現に変換
2. ベクトル表現をべき表現に変換

つまり，対応表作成時のこのふたつのアルゴリズムを加法表アルゴリズムにおいて参照作業と置き換えればいいわけである．

このふたつのアルゴリズムを比較すると一時保管アルゴリズムは，その容量の問題があるが演算表が完成すると加法表作成時には参照だけであるので加法表を作成する作成時間は短くて済む．これに対して，加法表作成アルゴリズムに統合する場合は加法表の要素数 2^{2m} 分，$\alpha^i + \alpha^j$ を行なうので変換作業は $2^{2m} \times 3$ 回行なわなくてはいけない．

ふたつのアルゴリズムを比べると容量と作成時間がトレードオフの関係になる．保存が実現した場合，演算表を作成するのは問題が起きたときのみなので，一概にどちらが有用であるということは言えない．ユーザの所有するコンピュータが一時記憶領域等において優れているならば，一時的保管アルゴリズムを採用した方が有用であり，逆に計算処理能力において優れているなら変換作業を用いた加法表作成アルゴリズムを採用する方が有用であると考えられる．

3.4　拡大体上の (k, n) しきい値法の実現

3.4.1　分散符号化

分散符号化処理では，秘密情報から，k 個以上集めると復号できるが $k-1$ 個以下では復号できないような n 個のシェアを作成する．

作成した分散符号化アルゴリズムの条件，制約を以下に記す．

1. $\boldsymbol{GF}(2^m)$ の場合，入力できる情報は 0 から $2^m - 1$ の正整数値である．
2. しきい値 k, n の値は $k < n$ かつ $n < 2^m$ を満たす正整数である．
3. 乱数は $\boldsymbol{GF}(2^m)$ の元とする．
4. 拡大体を決定する既約多項式はあらかじめ設定されている値を使用する．
5. 分散符号化処理に必要な係数行列 $\boldsymbol{G}$ を作成し，かつ ID 設定に使用される原始根を 1 つ定めている．これにより ID は $1, 2, \cdots, n$ になる．

拡大体上の分散符号化の手順を以下に示す.

初期設定

1. $\boldsymbol{GF}(2^m)$ 対応表，加法表を作成する.
2. ユーザがしきい値 $k, n \in \boldsymbol{GF}(2^m)$ を入力する.

分散符号化

1. ユーザが秘密情報 $S \in \boldsymbol{GF}(p)$ と $\boldsymbol{GF}(2^m)$ からランダムに選んだ $k-1$ 個の元 $r_j \quad (j = 1, 2, \cdots, k-1)$ を入力する.
2. $x_i = i$ とする．$(i = 1, 2, \cdots, n)$
3. $w_i = f(x_i)$ を以下の式で計算する．ただし演算は直接行わず，作成した演算アルゴリズムにしたがって行う．$(i = 1, 2, \cdots, n)$

$$f(x_i) = S + \sum_{j=1}^{k-1} r_j {x_i}^j \tag{3.12}$$

4. IDx_i とシェア w_i を出力する．$(i = 1, 2, \cdots, n)$

まず指定された拡大次数 m から $\boldsymbol{GF}(2^m)$ の演算表を作成する．必要な値がすべて入力されると ID が定められ，分散符号化処理を開始する．分散符号化処理は素体上で利用した式と同じ式を用いるが，演算は拡大体上の演算を行う．ここでの四則演算は各演算アルゴリズムを行って演算結果を求める．処理が終了すると最後に，求めた ID とシェアを出力する.

3.4.2　復号化

分散符号化された ID とシェア k 個から，秘密情報を復号する．作成した復号化アルゴリズムの条件，制約を以下に記す.

1. 拡大体を決定する既約多項式はあらかじめ設定されている値を使用する.
2. 拡大次数 m は分散符号化されたときと同じ値を用いる．つまり同じ演算表を使う．異

なる場合は正しく復号できない.

3. しきい値 k は分散符号化されたときと同じ値を用いる．異なる場合は正しく復号できない.

初期設定

1. k 人が各々の持つ $ID = x_{i_j}$ と w_{i_j}　$(1 \leq j \leq k)$ を持ち寄る.
2. $\boldsymbol{GF}(2^m)$ の対応表，加法表を作成する.
3. ユーザが，分散符号化したときと同じしきい値 $k \in \boldsymbol{GF}(2^m)$ を入力する.

復号化

1. ユーザが k 個の ID とシェアを入力する.
2. 以下の式に x_{i_j} と w_{i_j} を代入して線形方程式を求める.

$$f(x) = S + r_1 x + r_2 x^2 + \cdots + r_{k-1} x^{k-1} \tag{3.13}$$

3. 2 を k 人分繰り返す.
4. k 個の線形方程式から k 個の未知数 $S, r_1, r_2, \cdots, r_{k-1}$ を求める．演算結果はすべて演算表から取り出す.
5. 秘密情報 S と $k-1$ 個の乱数を出力する.

まず，指定された拡大次数 m から $\boldsymbol{GF}(2^m)$ の演算表を作成する．必要な値がすべて入力されると ID が定められ，復号化処理を開始する．連立方程式はガウスの消去法 (付録 C 参照) を用いて計算する．ただし四則演算はすべて各々の演算アルゴリズムを行って演算結果を求める．処理が終了すると最後に情報と $k-1$ 個の乱数を出力する.

3.5　演算量の比較

ここで，まず本研究と文献 [12] の分散符号化に要する演算量を比較する．分散符号化に要する演算量はしきい値 (k, n) に依存していることから表 3.5 のようになる.

表 3.5　分散符号化に要する演算量の比較

	文献 [12]	本研究
表引き	$k(n-1)+kn$	$2k(n-1)$
判定式	0	$3k(n-1)+kn$
乗算	0	kn

- k(n-1) ： 分散符号化に要する加算処理回数
- kn ： 分散符号化に要する乗算処理回数

実際に表 3.5 のしきい値に値を入れてみると表 3.6 のようになる.

表 3.6　分散符号化にようする演算量の比較 (具体例)

しきい値	昨年度	本研究
$(3,5)$	27	90
$(17,30)$	1003	3485
$(25,47)$	2303	7990

次に，復号化に要する演算量を比較する．復号化は分散符号化と異なり，しきい値 k のみに依存する．これは k 個の線形方程式から秘密情報を得るためである.

実際に表 3.7 の k に値を入れてみると表 3.8 のようになる.

本研究は文献 [12] と比較して場合分けをしてる分だけ演算量が多くなってしまった．しかし，文献 [12] の演算量は変わらないが本研究の演算量というのは最大の場合であり，実際には表 3.6,3.8 に示している値の 3 分の 2 程度になると思われる．例えば，表 3.6,3.8 から，日本の住民データを 47 都道府県で分散管理し，そのうち 25 個シェアが集まれば住民データを復号できるという大規模な管理を考えても，演算量の増加分は問題になる程ではないことがわかる.

表 3.7　復号化に要する演算量の比較

	文献 [12]	本研究
表引き	$2(\sum_{x=1}^{k} x(x-1) + 2\sum_{x=1}^{k} x) + k + \sum_{x=1}^{k+1} x$	$2(\sum_{x=1}^{k} x(x-1) + 2\sum_{x=1}^{k} x)$
判定式	0	$4(\sum_{x=1}^{k} x(x-1) + 2\sum_{x=1}^{k} x) + k + \sum_{x=1}^{k+1} x$
乗算	0	$\sum_{x=1}^{k} x(x-1) + 2\sum_{x=1}^{k} x$
除算	0	$k + \sum_{x=1}^{k+1} x$

表 3.8　復号化に要する演算量の比較 (具体例)

k	昨年度	本研究
3	56	174
17	4958	15810
25	13500	46150

3.6　評価

今回作成した演算アルゴリズムは，演算表を用いて表引きのみで実行されていた文献 [12] よりも演算量が増加したことは否めない．しかし，各々の演算は比較的簡単な処理のみである．また，演算量はしきい値 (k, n) に依存する．そのために増加した演算量が問題になる程ではない．加法演算表については，用いられる演算表が分散符号化と復号で同じものを用いるので分散符号化のときに作成したものを保存しておけば復号の際に演算表を作成する必要がなくなる．ただしこの場合，演算表の保管方法を考えなくてはいけない．対応表は加算表作成時に用いられるだけで秘密分散を実行する際には参照されることがないので保管の対象となるのは加法表のみである．

保管に関しては，拡大次数も一緒に記憶しておくのが望ましい．これは保管している演算

表に問題が起きた場合，分散符号化に使った拡大次数さえわかれば再度同じ演算表を作成することができるためである．これで演算表の紛失・破壊などに対応できる．演算表が改ざんされた場合は，この事実を発見する方法が必要である．事実さえわかれば再度演算表を作成することで問題を解決できる．

また，秘密分散法を実現する拡大体がべき表現において 0 を含まないものを用いることが可能であれば，本研究の演算量は半分程度になる．$\boldsymbol{GF}(2^m)$ の利点である $0, 1$ のみで表現が可能という点は，ベクトル表現を用いる場合であるため，この利点が消えるわけではない．

演算に関しては，加法も直接加算できる手段を考える必要があると思われる．現状では，演算表を保管する手段は考慮されてない．つまり，現段階では毎回，実行前に演算表作成をしなくてはいけない．分散符号化の際の加法表の参照回数は最大で $2k(n-1)$ 回である．実際の加法表の要素数を考えると参照されない要素の方が多い．つまり，保管が実現しなかった場合，演算表は無駄な要素の方が多くなる．

具体的な加算の方法としては，$\boldsymbol{GF}(2)$ を m 次に拡大する方法からその一般式を発見し，発見した式にベクトル表現を用いて加算を行うということが考えられる，このときの演算は $0, 1$ のみを用いるため，演算量も膨大な量にはならないと予想される．

第 4 章

結論

4.1 結論

本研究では秘密分散法を拡大体上で実現する際の計算処理を効率的かつ高速に実現することを主な研究対象とした．これに対して，拡大体上の演算方法の特徴を考慮して場合わけし，2 項演算の内容により最適と思われる演算を定義した．

結果を演算量のみに限定すると今回の演算アルゴリズムは場合分けをしている分，文献 [12] の表引きアルゴリズムより演算量が多いことは否めない．しかし，本研究が提案した演算アルゴリズムは比較的簡単な演算処理しかしていない．また，演算量はしきい値 (k, n) に依存するため，処理速度および演算量の増加分が問題になることはない．

4.2 今後の課題

本研究では，拡大体上の秘密分散法における計算処理を定義したにすぎない．実用化に向けてさらに発展させる必要がある．今後の課題を列挙する．

1. 拡大体上の秘密分散法の完成

   (ファイルからの入力，シェアファイルの作成，しきい値と拡大次数値の保管など)
2. 演算表の保管と保管されている演算表が常に正しいものであるかの検証
3. 直接加算処理をする演算アルゴリズムの検討
4. 分散符号化が正しく行われたかどうかの判定
5. 他の暗号システムとの併用

4.2 今後の課題

暗号化された情報は，暗号が解読されたときには情報が全て露見してしまう．秘密分散では，k 個のシェアを集めれば簡単に秘密情報を復元できてしまう．これらの欠点を補うために，多数の暗号システムと秘密分散法を組合せれば，より安全なシステムとなる．

謝辞

本研究を遂行するにあたり、終止御指導御鞭撻を承りました高知工科大学情報システム工学科の福本昌弘助教授に謹んで感謝致します.

本論文を御審議してくださる島村和典教授，菊池豊助教授，情報システム工学科の先生方に謹んで感謝致します.

何かとお世話になることが多かった妻鳥貴彦助手に謹んで感謝致します.

研究室内において 1 年間暖かく見守ってくれた秋山由佳さん・舟橋釈仁さんに謹んで感謝致します.

この 1 年間ともに研究生活を送った山本真弘さんに深く感謝致します.

休憩相手になってくれたのっしぃ〜こと登伸一さん，ひらっちこと平山正治さん，はっしぃ〜こと橋本正和さんにも深く感謝します.

そして，とっちぃ〜コンピュータこと栃木隆道さん，亀本学さんありがとう.

参考文献

[1] 尾形わかは, 黒沢馨, “秘密分散法とその応用,” 電子情報通信学会誌, Vol.82, No.12, pp.1228–1236, Dec.1999.

[2] 今井秀樹, 現代暗号とマジックプロトコル, サイエンス社, 2000.

[3] 尾形わかは, 岡田光司, 黒沢馨, “秘密分散共有法,” ComputerToday, No.86, pp.18-23, サイエンス社, 1998.

[4] D.R.Stinson, (櫻井幸一 訳), 暗号理論の基礎, 共立出版, 1996.

[5] 岡本 龍明, 山本 博資, 現代暗号, 産業図書,1979

[6] 井関 清志, 集合と論理, 新曜社,1979.

[7] 大矢雅則編著, 井上啓, 佐藤圭子, 情報数理入門, サイエンス社, 1999.

[8] 松坂和夫, 代数系入門, 岩波書店, 1976.

[9] Hajime Kaji, 工科系のための初等整数論入門, 倍風館,2000

[10] 杉原 厚吉, 今井 敏行, 工学のための応用代数, 共立出版,1999

[11] 新妻 弘, 木村 哲三, 群・環・体 入門, 共立出版,1999

[12] 庄田 亜輝,“秘密分散法における効率的なアルゴリズム”, 平成 12 年度高知工科大学学士学位論文,Feb.2001

付録 A

線形方程式が独立であることの証明

ここでは, 式 (2.6) の線形方程式が互いに独立であることを証明する.

行列 $\boldsymbol{G}$ は

$$\det \boldsymbol{G} = \prod_{1 \leq j < t \leq k} (x_{i_t} - x_{i_j}) \pmod{p} \tag{A.1}$$

で定められる.

$x_i \quad (i = 0, 1, \cdots, k-1)$ はそれぞれ異なる値である. よって $x_{i_t} - x_{i_j} \neq 0$ となる. なぜならば, この乗算は体 $\boldsymbol{GF}(p)$ 上で計算されるからである. 体上の乗算では, 0 でない項の積は常に 0 でない. したがって $\det \boldsymbol{G} \neq 0$ となる. 係数行列 $\boldsymbol{G}$ の行列式が 0 でないので, 式 (2.6) は体上で一意の解を持つ.

付録 B

べき表現において指数法則が成り立つことの証明

ここでは, 式 (2.22) の指数法則が成り立つことを証明する. $\alpha^m \cdot \alpha^n = \alpha^{m+n}$ を場合わけによって示す.

1. m,n のいずれかが 0 のとき

   $m = 0$,e を単位元とすると

   左辺 $= \alpha^m \cdot \alpha^n = \alpha^0 \cdot \alpha^n = e \cdot \alpha^n$

   右辺 $= \alpha^{m+n} = \alpha^{0+n} = \alpha^n$

2. m と n がどちらも正の整数のとき

   $\alpha^m \cdot \alpha^n = \overbrace{(\alpha \cdots \alpha)}^{m}\overbrace{(\alpha \cdots \alpha)}^{n} = \alpha^{m+n}$

3. m と n の 2 つとも負の整数のとき

   $m < 0, n < 0$ であるから,$m' = -m, n' = -n$ とおけば,$m' > 0, n' > 0$. よって,

   $$\begin{aligned} \alpha^m \cdot \alpha^n &= (\alpha^{-1})^{m'}(\alpha^{-1})^{n'} \\ &= (\alpha^{-1})^{m'+n'} \\ &= \alpha^{(m'+n')} \\ &= \alpha^{m+n} \end{aligned}$$

4. m と n のいずれかが負で他方が正の整数のとき

   $m > 0, n < 0$ として $n' = -n > 0$ とおけば

   $\alpha^m \cdot \alpha^n = \alpha^m(\alpha^{-1})^{n'} = \overbrace{(\alpha \cdots \alpha)}^{m}\overbrace{(\alpha^{-1} \cdots \alpha^{-1})}^{n'}$

(a) $(m > n')$ のとき　$\alpha^{m-n'} = \alpha^{m+n}$

(b) $(m = n')$ のとき　$e = \alpha^0 = \alpha^{m+n}$

(c) $(m < n')$ のとき　$(\alpha^{-1})^{n'-m} = \alpha^{-n'+m} = \alpha^{m+n}$

ゆえに, この場合にも $\alpha^m \cdot \alpha^n = \alpha^{m+n}$ が成り立つ

付録 C

ガウスの消去法

連立 1 次方程式

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned} \tag{C.1}$$

の解法を示す.

1. まず,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad , \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{C.2}$$

とおく. 連立方程式を

$$Ax = b \tag{C.3}$$

と行列で表す.

2. $\widehat{A} = [A|b]$ とおくと,

$$\widehat{A} = [A|b] = \left[\begin{array}{cccc|c} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & b_n \end{array}\right] \tag{C.4}$$

と表される. 行列の基本変形

(a) ある行を定数倍する.

(b) ある行を他の行に加えて, それを新たな行とする.

(c) 2 つの行を入れ換える.

を行って,

$$\widehat{A} \to \left[\begin{array}{cccccc|c} 1 & a'_{12} & a'_{13} & a'_{14} & \cdots & a'_{1n} & b'_1 \\ 0 & 1 & a'_{23} & a'_{24} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & 1 & a'_{34} & \cdots & a'_{3n} & b'_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & b'_n \end{array}\right] \tag{C.5}$$

の形に変形する. この変形操作を前進消去とよぶ.

3. 前進消去により, $Ax = b$ は以下のように変形される.

$$\left[\begin{array}{cccccc} 1 & a'_{12} & a'_{13} & a'_{14} & \cdots & a'_{1n} \\ 0 & 1 & a'_{23} & a'_{24} & \cdots & a'_{2n} \\ 0 & 0 & 1 & a'_{34} & \cdots & a'_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{array}\right] \tag{C.6}$$

この式から n 個の方程式が得られる.

$$\begin{aligned} x_1 + a'_{12}x_2 + a'_{13}x_3 + a'_{14}x_4 + \cdots + a'_{1n}x_n &= b'_1 \\ x_2 + a'_{23}x_3 + a'_{24}x_4 + \cdots + a'_{2n}x_n &= b'_2 \\ x_3 + a'_{34}x_4 + \cdots + a'_{3n}x_n &= b'_3 \\ &\vdots \\ x_n &= b_n \end{aligned} \tag{C.7}$$

なお, x_n の値が $x_n = b_n$ と確定しているので, この値を用いて方程式の解 x_i を求めていくことが可能である. この処理を後退代入という.