平成 12 年度

学士学位論文

秘密分散法における効率的なアルゴリズム

Efficient Algorithm for the Secred Sharing Scheme

1010398　　庄田亜燁

指導教員　　福本昌弘

2001 年 2 月 5 日

高知工科大学 情報システム工学科

要 旨

秘密分散法における効率的なアルゴリズム

庄田亜輝

コンピュータへの情報の保管には, 盗聴やデータ紛失などの危険がともなう. これらへの耐性を高め, 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る. 本研究では, 秘密分散法を効率良く実行するためのアルゴリズムを検討する. 演算処理の高速化を目的として, 拡大体上での秘密分散法の実現とその演算処理方法の検討を行う.

キーワード　秘密分散法, (k, n) しきい値法, 有限体, 拡大体

Abstract

Efficient Algorithm for the Secred Sharing Scheme

Aki SHODA

Risk, such as eavesdropping and data loss, follows on storage of the information on a computer. The Secred Sharing Scheme(SSS) is one of the safty method of data conservation. In this paper, we consider efficent algorithm for the SSS. we conduct an investigation into SSS for the extension field $\boldsymbol{GF}(2^m)$ and method of calculation.

key words Secred Sharing Scheme(SSS), (k,n)threshold SSS, finite field, extension field

目次

第 1 章　序論　1

1.1　研究の背景と目的 1

1.2　本論文の概要 2

第 2 章　秘密分散法と体論　3

2.1　まえがき 3

2.2　秘密分散法とは 3

2.3　特徴と利点 5

2.4　秘密分散法の種類 5

2.5　実現方法 8

2.6　問題点と改良 12

2.7　体論 12

2.7.1　素体 12

2.7.2　拡大体 14

2.7.3　2 の拡大体 14

2.7.4　2 の拡大体上の演算 17

第 3 章　拡大体上の秘密分散法　22

3.1　まえがき 22

3.2　演算の効率化 22

3.2.1　乗法表作成 23

3.2.2　べき表現-ベクトル表現対応表作成 24

3.2.3　加法表作成 26

3.2.4　除法表作成 28

3.3 拡大体上の (k, n) しきい値法の実現 30
3.3.1 分散符号化 30
3.3.2 復号化 31
3.4 評価 32

第 4 章 結論 **34**
4.1 結論 34
4.2 今後の課題 34

謝辞 **36**

参考文献 **37**

付録 A 線形方程式が独立であることの証明 **38**

付録 B ガウスの消去法 **39**

図目次

1.1 情報保管時のリスク 1

2.1 秘密分散法 4

2.2 分散符号化 4

2.3 復号化 5

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 7

表目次

2.1 $\boldsymbol{GF}(5)$ の演算表 13
2.2 $\boldsymbol{GF}(2^2)$ の演算表 15
2.3 $\boldsymbol{GF}(2^4)$ の対応表 18

3.1 $\boldsymbol{GF}(2^3)$ の乗法表 24
3.2 $\boldsymbol{GF}(2^3)$ の対応表 26
3.3 $\boldsymbol{GF}(2^3)$ の加法表 28
3.4 乗法表を除法表に変換 29
3.5 $\boldsymbol{GF}(2^3)$ の除法表 29

第 1 章

序論

1.1　研究の背景と目的

現在では多くの情報が電子化されてコンピュータ内に保存されている．電子化された情報は検索や編集が容易にできるという利点を持つ．しかしその一方で, 重要な情報をコンピュータに保存するときにいくつかの危険がともなう．例えば秘密情報が他人に見られる危険性, 保存している情報が何らかの原因によって消滅してしまう危険性などがある．

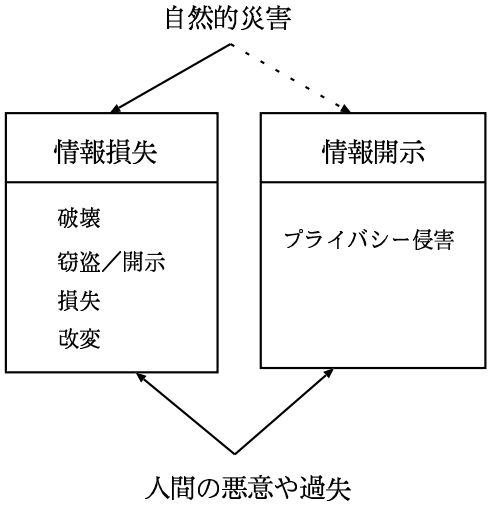

図 1.1　情報保管時のリスク

情報を保存しているシステムに支障が起きた場合には, 情報が消えてしまって取り出せなくなる可能性がある．このような情報の紛失への対処として, 情報を分散管理する方法がある．重要な情報は 1ヶ所だけでなく, いくつかのシステムにコピーを保存しておく．情報が消えてしまったら, バックアップとして保存している情報を取り出せばよい．しかし, 情報が非

常に重要なものである場合, コピーそのものを人に渡してしまうことは危険である. コピーを作っていくつかの場所に配布すれば, それだけ他人に見られる危険性を増加させることになる.

秘密情報を他人に見られないようにする方法として暗号がある. 秘密情報を暗号化することによって, 他人が重要な情報にアクセスしても内容が露見しない. しかし災害などによる情報紛失の際には情報が復元できず, 暗号が解かれた場合は秘密情報が漏洩してしまう.

情報を安全に保管するためには秘密情報の盗用と紛失の両方に対して有効な方法を考えなければならない. これらへの耐性を高める保管方法のひとつとして秘密分散法がある. 本研究では秘密分散法に着目し, 効率良く実行するためのアルゴリズムを検討する.

1.2　本論文の概要

ここでは本論文の構成を簡単に述べる.

秘密分散法については, どのようなものであるのか, また何故この 2 つの危険に対して有効であるのか第 2 章で説明する. また, 秘密分散法は素体上で実現されるため, 基礎知識として体に関する説明を行う. 第 3 章では実際に構築した拡大体上の秘密分散法について記述する. 第 4 章では研究のまとめを述べる.

第 2 章

秘密分散法と体論

2.1 まえがき

まず秘密分散法の定義, 特徴や構築方法について説明する. 秘密分散法は素体上で実現されるが, 本研究では拡大体上での実現を試みる. その違いを説明するために素体と拡大体の違いも説明しておく.

2.2 秘密分散法とは

銀行に金庫があり, 金庫を開ける権利を持つ人が 3 人いるとする. しかし, この 3 人それぞれに鍵を渡すと, その 3 人は金庫を開けようと思えば 1 人でも開けられることになり危険である. よって, 権利を持つ 3 人のうち任意の 2 人が合意したときのみ, 金庫を開けることができるようにしたい. 秘密分散法はこのようなシステムを実現する. つまり秘密分散法とは, 秘密情報を n 人のグループで分散管理し, n 人のうちの任意の k 人が合意したときだけ元の情報を閲覧できるが, $k-1$ 人が秘密情報を復元しようとしても情報は得られないような情報の保管方法のことである. また, 保管する分散情報をシェアという [1].

秘密分散法では分散符号化と復号化とよばれる処理を行う.

ある秘密情報 S を分散して保管するとき, この秘密情報を以下の条件を満たすように n 個のシェア $w_1, w_2, \cdots, w_n$ に分割する.

1. w_i のうちの k 個があれば S を算出できて, かつ
2. w_i のうち $k-1$ 個以下では情報不足で S の算出が不可能であること.

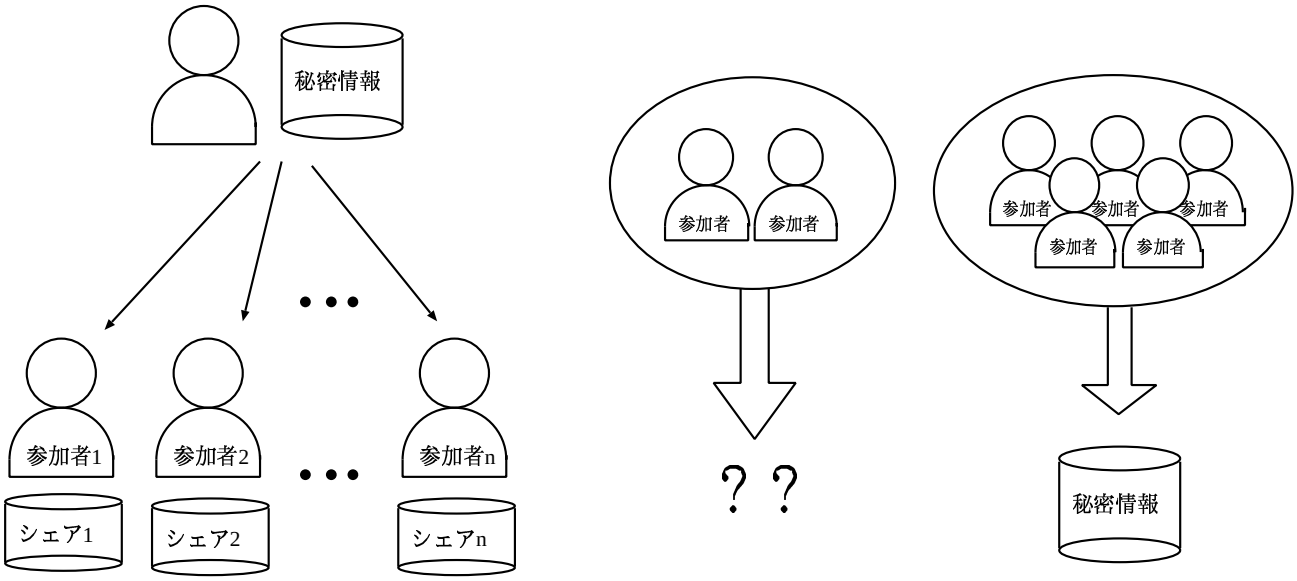

図 2.1　秘密分散法

このシェアの作成処理のことを分散符号化という．秘密情報 S を分散符号化して n 個のシェアを作り, それぞれのシェアを n 人によって分散管理する．

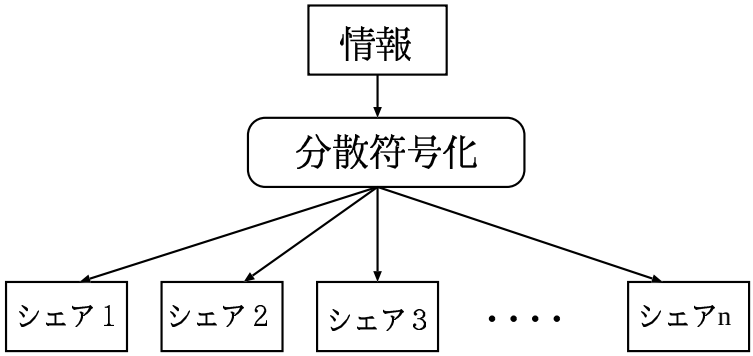

図 2.2　分散符号化

秘密情報を復元したいときには, 任意の k 人それぞれが持つシェアを集めて処理を行えば元の情報が復元できる．この処理を復号化という．

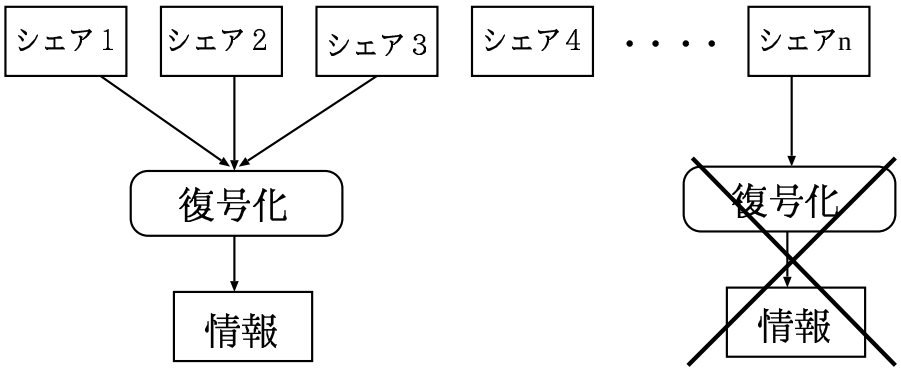

図 2.3　復号化

2.3　特徴と利点

重要な情報を保管する場合, 情報の紛失, 悪意を持った第 3 者による情報の破壊や盗難などの危険性がある. 情報の紛失に対してはコピーを複数作っておけばよい. しかしコピーが増えれば情報破壊や盗難の危険が増してしまう. 秘密分散法はこのような問題を解決することができる.

それぞれのシェアは, どの $k-1$ 個を取り出して見たとしても元の情報が得られないように作られている. よって, シェアが $k-1$ 個以下漏洩しても秘密情報は復元されない. 即ち k 人未満のユーザが共謀して秘密情報を復元しようとしても不可能である. また, n 個に分散された情報を復元するために, 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, シェアは分散符号化されているため情報を知ることはできない. k 個のシェアを集めない限り, 元の情報の秘密は守られる. シェアが破壊された場合も, 残りのシェアから元の情報を復元できる. 同様に, 保管していたシェアが消えてしまっても, 残りのシェアを用いて復元が可能である. 秘密分散法は上に述べたような特徴によって, 秘密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]

   n 人のうち k 人が合意すれば情報を復元できるが, $k-1$ 人では元の情報は得られない. k と n の値をしきい値と呼び, この秘密分散法を (k, n) しきい値法という. 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. 秘密情報を 10 個に分散符号化し, 3 人の課長, 7 人の平社員にシェアを渡す. 課長なら 2 人, 平社員なら 6 人集まれば復元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元できる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元できないため, 盗聴に対する安全性は高い. 一方, 1 個でもシェアが消えてしまうと元の情報を復元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 100 のうち下位 50buit 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 人で復元可能

   (b) $k-d$ 人以下では復元不可能

   (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

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

[*1] シェアファイルの大きさ

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.

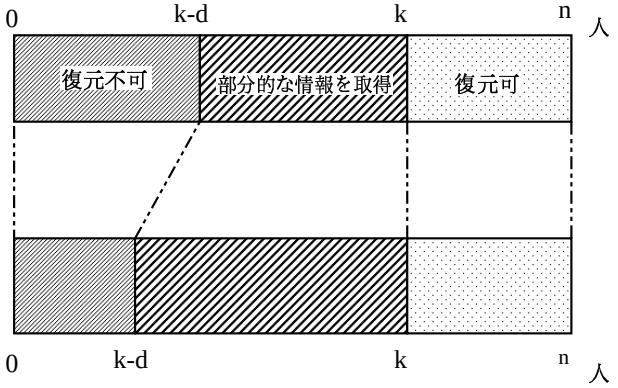

図 2.4　(k, d, n) ランプスキームにおいて d を大きくした場合

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. 現在考えている秘密分散法を利用した分散管理システムは, 秘密情報を分散符号化し, 作成したシェアを複数の地域で分散管理するものである. (k, n) しきい値法の特徴である $n - k$ 個以下のシェアを失っても残りの k 個から秘密情報を復元可能であることを利用すれば, 例えば, 災害によるシステムの破壊が起こってシェアが消えてしまった場合でも, シェアが k 個以上残ってさえいれば秘密情報を復元できる. そしてシェアを複数箇所に保管することによって, 不正なアクセスによってシェアが盗まれたとしても, k 箇所のシェアを集めない限り秘密情報を知られることはない. どこかでシェアが盗まれた場合, 残ったシェアからもう一度秘密情報を復元して分散符号化し直せば, さらにシェアを盗まれたとしてもそれらのシェアから秘密情報を復元することは不可能となる.

満場一致法では, n 人のうち n 人全員が集まらなければ復元できないため, 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元できない. 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 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.

秘密情報を分散符号化する人をディーラといい. 作成されたシェアを受け取る人を参加者という. 参加者が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配布される. n 人の参加者の集合を

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

と表すことにする.

処理は素体 $\boldsymbol{GF}(p)$ (2.7.1 参照) 上で行われる. ここで p は素数であり, $p \geq n + 1$ を満たす. 秘密情報 S も $\boldsymbol{GF}(p)$ の元とする. ディーラは秘密情報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. その際は, 多項式の定数項を秘密情報 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. x_i は ID といい, 参加者はシェアと ID を保管する.

分散符号化の手順を以下にまとめる.

初期設定

1. ディーラは $\boldsymbol{GF}(p)$ の元から, 0 ではない別々のものを n 個選びだし, $x_i \quad (i = 1, 2, \cdots, n)$ と記述する. $p \geq n + 1$ であることに注意する.
2. $i = 1, 2, \cdots, n$ に対し, ディーラは値 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. ディーラが秘密情報 $S \in \boldsymbol{GF}(p)$ を分散保管したいものとする. ディーラは $\boldsymbol{GF}(p)$ 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. ディーラは $w_i = f(x_i)$ 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. ディーラはシェア w_i を H_i に与える. $(i = 1, 2, \cdots, n)$

次に, k 人の参加者集合が秘密情報を復元する方法について記す.

k 人の参加者 $H_{i_1}, H_{i_2}, \cdots, H_{i_k}$ が集まって秘密情報 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,

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

が成立することを知っている. $f(x)$ の最大次数は $k-1$ である. よって $f(x)$ 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 $r_1, r_2, \cdots, r_{k-1}$ は未知の $\boldsymbol{GF}(p)$ の元である. S は秘密情報であり, 同じく未知の $\boldsymbol{GF}(p)$ の元である. k 人が持ち寄った x_i と 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, これは行列を用いて,

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

と表すことができる. これらの線形方程式は互いに独立であるため一意に解が得られ, 秘密情報 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

の線形方程式が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 (2.5) における係数行列を $\boldsymbol{G}$ とおく.

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

以下に復号化の手順をまとめる.

初期設定

1. k 人が各々の持つ x_{i_j} と 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.

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

2. 1 を k 人分繰り返す.
3. 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. 秘密情報 S 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 $x_1, x_2, \cdots, x_5$ の値を $x_1 = 3, x_2 = 2, x_3 = 6, x_4 = 4, x_5 = 5$ とする. 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. 秘密情報 $S = 3$ 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

体 $\boldsymbol{GF}(7)$ 上で $(3,5)$ しきい値法を実行して 3 個のシェアから秘密情報を求める. 公開されている $x_1, x_2, \cdots, x_5$ の値は $x_1 = 3, x_2 = 2, x_3 = 6, x_4 = 4, x_5 = 5$ である. 3 人の参加者 H_1, H_3, H_4 がそれぞれのシェア $w_1 = 1, w_3 = 3, w_4 = 2$ を持ち寄ったとする. 式 (2.4) より, 多項式 $f(x)$ は

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

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. よって秘密情報 $S = 3$ が得られた.

2.6 問題点と改良

参加者が保持する ID となる $x_i \quad (i = 1, 2, \cdots, n)$ の値はすべて異なるものでなければならない. したがって原始根 2.7.3 を用いる. そのため, 原始根を求める計算が必要となる. また, $\boldsymbol{GF}(p)$ で実行するときの素数 p を求めなければならない. しかし原始根や素数を求めるには膨大な時間がかかってしまう. これらの問題点を解決するために拡大体上で実現することを考える. 次に体と拡大体について説明する.

2.7 体論

秘密分散法は素体 $\boldsymbol{GF}(p)$ 上で実現されるが, 本研究では拡大体上での実現を目指す. したがって, 2.7 では素体と拡大体の違いを主として体論について述べておく.

2.7.1 素体

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. ここではそのような無限の元をもつ体

表 2.1 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

ではなく, 有限個の元から成る有限体について述べる. 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. q はどんな数でもよいわけではなく, 素数 p または素数のべき p^m (p : 素数, m : 自然数) のときに限って体となる.

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. 加法・乗法は演算を行った後, p で割った余りをとればよい. 減算 $b-a$ は, b に a の加法に関する逆元 $-a$ を加える. 加法逆元 $-a$ とは

$$(-a)+a=0 \pmod{p} \tag{2.17}$$

を満たす元である.[*2] 零でない元 a による除算 b/a は, 元 b に a の乗法に関する逆元 a^{-1} を掛ければよい. 乗法逆元 a^{-1} は

$$a^{-1}a=1 \pmod{p} \tag{2.18}$$

を満たす元である.

たとえば $\boldsymbol{GF}(2)$ は 2 つの元 $0,1$ をもつ体である. これは 2 進数と対応する. ここでは例として表 2.1 に $\boldsymbol{GF}(5)$ の演算を示す.

[*2] $x=y \pmod{z}$ とは $x-y$ が z で割り切れることを意味する. "x と y が z を法として合同" と読む. このような式を合同式という.

2.7.2 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．これはつまり 2 による除算ができないことを意味している．よってこのような体を作るためには別の方法が必要である．

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, 実数体上の多項式 $x^2 + 1$ の 1 つの解である複素数の虚数単位 i を実数体に付加して作られる．ここで使う多項式は既約なものすなわちそれ以上因数分解されない最小の多項式でなければならない．実数体に虚数単位 i を付加して, さらにそれが体となるために必要な元を全て付け加えたものが複素数体である．このように体 $\boldsymbol{F}$ に, $\boldsymbol{F}$ 上で既約な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 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,

$$a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0 \tag{2.19}$$

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．

2.7.3 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．2 進数は $\boldsymbol{GF}(2)$ と対応している．よって $\boldsymbol{GF}(2^m)$ はコンピュータへの適用という点で有効であるため, $\boldsymbol{GF}(2^m)$ を利用することにする．$\boldsymbol{GF}(2)$ 上の多項式 $x^2 + x + 1$ は既約である．この多項式の解を α とし, $\boldsymbol{GF}(2)$ に α を付加した体を作成する．この体はまず $\boldsymbol{GF}(2)$ 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．元同士の乗算の結果が体に含まれなければならないから, α のべき $\alpha^2, \alpha^3, \cdots$ はすべてこの体の元となる．α は $x^2 + x + 1$ の解としたので,

$$\alpha^2 + \alpha + 1 = 0 \tag{2.20}$$

表 2.2 $\mathrm{GF}(2^2)$ 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

が得られる．これは変形して

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

となる．これにより α のべきが次のように求められる．

$$\begin{aligned} \alpha^0 &= 1 \\ \alpha^1 &= \alpha \\ \alpha^2 &= \alpha + 1 \\ \alpha^3 &= \alpha^2 + \alpha = 1 = \alpha^0 \\ &\vdots \end{aligned} \tag{2.22}$$

$\alpha^3 = 1$ 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．よってこの体の元は $0, 1, \alpha, \alpha^2 (= \alpha + 1)$ の 4 つになる．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加法表・乗法表を作成することにより確かめられる．この演算表を作る基本となる式は $\alpha^2 = \alpha + 1$ である．たとえば $\alpha^2 + \alpha$ は

$$\alpha^2 + \alpha = \alpha + 1 + \alpha = (1 + 1)\alpha + 1 = 1 \tag{2.23}$$

のように計算される．$\boldsymbol{GF}(2)$ 上の多項式であるので, 係数は 1 もしくは 0 にしかなりえない．$1 + 1 = 0$ になる点に注意する．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．表 2.2 に $\boldsymbol{GF}(2^2)$ の加法表と乗法表を示す．

$\boldsymbol{GF}(2^2)$ は $\boldsymbol{GF}(2)$ 係数 2 次既約多項式 $x^2 + x + 1$ の解 α のべき $\alpha^0, \alpha^1, \alpha^2$ と 0 の 4 つの元で構成できることが示された．同様に $\boldsymbol{GF}(2)$ の m 次拡大体 $\boldsymbol{GF}(2^m)$ も, $\boldsymbol{GF}(2)$ 係数

m 次既約多項式の解 α とそのべき $\alpha^0, \alpha^1, \alpha^2, \cdots, \alpha^{2^m-2}$ と 0 の 2^m 個の元で構成できる. α を 2^m-1 乗すれば再び $\alpha^0 = 1$ に戻る. つまり,

$$\alpha^{2^m-1} = 1 \tag{2.24}$$

になる. この既約多項式の解 α を $\boldsymbol{GF}(2^m)$ の原始根という. 厳密に述べると原始根とは, $\boldsymbol{GF}(2^m)$ の 0 以外の全ての元 $\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$ に対して,

1. $\alpha^m = 1$
2. $\alpha^n \neq 1 \quad (0 < n < m)$

となる元のことである. $\alpha^2, \alpha^3, \cdots$ 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 $\boldsymbol{GF}(2^m)$ における乗算が容易に実行できる. α^{2^m-1} だから, α^i, α^j の積を

$$\alpha^i \alpha^j = \alpha^{i+j} \tag{2.25}$$

として計算できる. ただし $i+j$ は 2^m-1 で剰余をとる.

α は m 次既約多項式

$$p(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_1 x + 1 \tag{2.26}$$

の解とする. 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

$$\alpha^m = f_{m-1}\alpha^{m-1} + \cdots + f_1 \alpha + 1 \tag{2.27}$$

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 $\alpha^i \quad (i = 0, 1, \cdots, 2^m-2)$ を $\boldsymbol{GF}(2)$ の元を係数とする $1, \alpha, \cdots, \alpha^{m-1}$ の 1 次結合で

$$\alpha^i = a_{m-1}\alpha^{m-1} + \cdots + a_1 \alpha + a_0 \tag{2.28}$$

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 (2.28) の係数のみを並べた

$$(a_{m-1}, \cdots, a_1, a_0) \tag{2.29}$$

という $\boldsymbol{GF}(2)$ 上の m 次元ベクトルの加算と同じである．よって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からである．すなわち

$$-x = x \tag{2.30}$$

となる．

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．なお，この体を

$$\boldsymbol{GF}(2^4) = \frac{\boldsymbol{GF}(2)}{x^4+x+1} \tag{2.31}$$

と表す．

2.7.4 2 の拡大体上の演算

コンピュータでの効率的な処理のためには，演算を 2 進数で表現することが望ましい．そこで $\boldsymbol{GF}(2^m)$ を考える．$\boldsymbol{GF}(2^m)$ の元は $\boldsymbol{GF}(2)$ 係数多項式である．すなわち多項式の係数は 2 進数の 0 と 1 である．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．そして，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 と対応しているというように，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．この対応の仕方は正整数の 2 進法表示法を用いて，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．同じ

表 2.3 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)

空間を用いて $\boldsymbol{GF}(p)$ で表すよりも多くの元を $\boldsymbol{GF}(2^m)$ で表現できる.

次に演算方法に関して両者の比較を行う. $\boldsymbol{GF}(2^m)$ における演算方法については前に述べたが, ここでは具体的な数を用いて $\boldsymbol{GF}(2^m)$ と $\boldsymbol{GF}(p)$ における演算の違いを記述する.

$\boldsymbol{GF}(2^m)$ 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をとればよい. $c = a + b$ とすると,

$$c_i = a_i \oplus b_i \quad (i = 0, 1, \cdots, m-1) \tag{2.32}$$

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.

$$\begin{array}{r} a = 10101 \\ \underline{b = 01100} \\ c = 11001 \end{array} \tag{2.33}$$

$\boldsymbol{GF}(p)$ 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り, 31 を法とする結果を得るために除算が必要になる.

1. ステップ 1　a と b を足す

$$\begin{array}{r} a = 10101 \quad (21) \\ \underline{b = 01100 \quad (12)} \\ c = 100001 \quad (33) \end{array} \tag{2.34}$$

2. ステップ 2　31 で割って剰余をとる

$$c = 00010 \quad (2) \tag{2.35}$$

$\boldsymbol{GF}(2^m)$ 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.

$$d = \sum_{i=0}^{m-1} (a_i * b) x^i \pmod{p(x)} \tag{2.36}$$

$$a * b = \begin{cases} b = b_{m-1} x^{m-1} + \cdots + b_0 & a_i = 1 \\ 0 & otherwise \end{cases} \tag{2.37}$$

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.

1. $a * a$ を計算する

$$\begin{array}{ccccc} & & 1 & 0 & 1 \\ & & 1 & 0 & 1 \\ \hline & & 1 & 0 & 1 \\ & 0 & 0 & 0 & \\ 1 & 0 & 1 & & \\ \hline 1 & 0 & 0 & 0 & 1 \end{array} \tag{2.38}$$

2. $p(x) = (1,0,1,1)$ で割る

$$\begin{array}{cccccc} & & & & 1 & 0 \\ \hline 1 & 0 & 1 & 1 \ ) & 1 & 0 & 0 & 0 & 1 \\ & & & & 1 & 0 & 1 & 1 & \\ \hline & & & & & 1 & 1 & 1 & = d \end{array} \tag{2.39}$$

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

$$\begin{array}{ccccc} & & 1 & 1 & 1 \\ & & 1 & 0 & 0 \\ \hline & & 0 & 0 & 0 \\ & 0 & 0 & 0 & \\ 1 & 1 & 1 & & \\ \hline 1 & 1 & 1 & 0 & 0 \end{array} \tag{2.40}$$

2. $p(x) = (1,0,1,1)$ で割る

$$\begin{array}{cccccc} & & & & 1 & 1 \\ \hline 1 & 0 & 1 & 1 \ ) & 1 & 1 & 1 & 0 & 0 \\ & & & & 1 & 0 & 1 & 1 & 0 \\ \hline & & & & & 1 & 0 & 1 & 1 \\ & & & & & 1 & 0 & 1 & 1 \\ \hline & & & & & & & 1 & = d \end{array} \tag{2.41}$$

$\boldsymbol{GF}(2^m)$ 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 まえがき

第 2 章で述べてきた $\boldsymbol{GF}(p)$ 上の秘密分散法を, 拡大体 $\boldsymbol{GF}(2^m)$ 上で実現する. 拡大体 $\boldsymbol{GF}(2^m)$ 上で秘密分散法を実行する際の高速な演算処理を実現するために, 拡大体上の演算方法を検討する. 具体的には,

1. 四則演算の効率化
2. $\boldsymbol{GF}(2^m)$ 上で (k, n) しきい値法を実行するアルゴリズムの検討

を行う. 四則演算の効率化を目指すが, 減法演算に関しては考えない. これは $a, b \in \boldsymbol{GF}(2^m)$ の減法 $a - b$ の結果は加法 $a + b$ と等しくなるためである. したがって考慮する演算は, 加法, 乗法, 除法の 3 つとする.

作成した演算表を用いて拡大体上で秘密分散法を構築する. 構築方法は素体上での構築方法と同じであるが, 演算方法は異なる. 本研究のテーマは拡大体上で秘密分散法を実現する際の演算処理である. したがって本研究では, ファイル入力・出力や正しく処理されたかどうかの確認などは考えないこととする. 分散符号化と復号化に分けてその仕組みを記述する.

3.2 演算の効率化

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. よって演算表を作成して演算の定義を行う. 計算前にあらかじめ演算表を作成しておくことによって, 演算は表を引くだけでよくなるため, 演算の効率化が可能であ

る. 演算表では $2^m \times 2^m$ の 2 次元配列で表現される. 加法表と乗法表では, 加算または乗算をする 2 つの元 α^i, α^j の交点を見れば, その演算結果である $\alpha^i + \alpha^j$, $\alpha^i * \alpha^j$ の値が得られる. 除法は加法や乗法とは異なり可換ではないため, 縦に被除数, 横に除数をとり, その演算結果 $\alpha^i \div \alpha^j$ をその交点に格納している.

3.2.1 乗法表作成

拡大体上の乗法演算は 2.7.3 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,

$$\alpha^i * \alpha^j = \alpha^{i+j} \tag{3.1}$$

で計算される. $i+j$ 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.

$$\alpha^i * 0 = 0 \quad (i = 0, 1, \cdots, 2^m - 2) \tag{3.2}$$

よって乗法表では, 0 との積を表す部分は全て 0 となり, その他の乗法表の α^i 行 α^j 列には α^{i+j} を代入する. 以下に乗法表の作成方法を示す. 手順 5, 6 では, 加法表が対称になることを利用して, 表作成にかかる演算量を少なくしている.

乗法表の作成手順

1. $\bmod p$ の計算を行う関数を用意しておく.
2. $i = 0$ から $2^m - 2$ まで 3 を繰り返す.
3. $j = i$ から $2^m - 2$ まで 4,5,6 を繰り返す.
4. $\alpha^i * \alpha^j$ を計算する. $(i + j \pmod{2^m - 1})$ を計算する)
5. α^i 行 α^j 列に計算した値を代入する.
6. α^j 行 α^i 列に計算した値を代入する.

表 3.1 に作成される乗法表の例を示す. これは $\boldsymbol{GF}(2^3) = \frac{\boldsymbol{GF}(2)}{x^3+x+1}$ の乗法表である. 以降に示す演算表の例は全て, 既約多項式 $x^3 + x + 1$ より作られた $\boldsymbol{GF}(2^3)$ の演算表である.

$\alpha^i * \alpha^j$ を計算するとき, α^i 行 α^j 列の要素を返す.

表 3.1 GF (2^3) の乗法表

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

3.2.2 べき表現-ベクトル表現対応表作成

べき表現で表された $\boldsymbol{GF}(2^m)$ の元は, m 桁のベクトル表現で表すことができる. 以下に例を示す.

$$\begin{aligned} \alpha^0 &\rightarrow (0,0,1) \\ \alpha^1 &\rightarrow (0,1,0) \\ &\vdots \end{aligned} \tag{3.3}$$

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, 加法表を作るときに利用する. 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.

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.

以下に, 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.

$$p(x) = b_m x^m + b_{m-1} x^{m-1} + \cdots + b_1 x + b_0 \tag{3.4}$$

とすると,

$$\alpha^m = b_{m-1} x^{m-1} + \cdots + b_1 x + b_0 \tag{3.5}$$

とおく. したがってベクトル表現では,

$$\alpha^m \rightarrow (b_{m-1}, \cdots, b_1, b_0) \tag{3.6}$$

となる.

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,

$$c_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \tag{3.7}$$

4. $\alpha^{i-1} * \alpha^1$ の値を乗算表から引き, 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 3.2 に作成される対応表の例を示す.

表 3.2 GF (2^3) の対応表

べき表現	多項式表現			ベクトル表現
0			0	$(0,0,0)$
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		$+1$	$(1,0,1)$

3.2.3 加法表作成

加法表は対応表を用いて作成する. $0 + 0 = 0$ であるので 0 行 0 列の要素は 0 になる. また, どの元に 0 を足してもその値は変わらない.

$$\alpha^i + 0 = \alpha^i \tag{3.8}$$

であるので, α^i 行 0 列と 0 行 α^i 列の要素はすべて α^i となる. また, 同じ元同士を加算した値は 0 である.

$$\alpha^i + \alpha^i = (1+1)\alpha^i = 0 \tag{3.9}$$

2 の拡大体なので係数は 0 か 1 のみで, 2 は 0 と合同である. α^i 行 α^i 列つまり表の対角線上の要素はすべて 0 になる. その他の要素は,

$$\begin{aligned} \alpha^0 + \alpha^1 &\rightarrow 001 + 010 \\ &= 011 \\ &\rightarrow \alpha^3 \end{aligned} \tag{3.10}$$

のように求める.

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. 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. 0 行 0 列に 0 を代入する.
2. α^i 行 0 列, 0 行 α^i 列に α^i を代入する.　$(i = 0, 1, \cdots, 2^m - 1)$
3. α^i 行 α^i 列に 0 を代入する.
4. $i = 1$ から $2^m - 1$ まで 5 を繰り返す.
5. $j = 1$ から $2^m - 1$ まで 6 を繰り返す.
6. $\alpha^i + \alpha^j$ 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,α^j 行 α^i 列に代入する.

表 3.3 に作成される加法表の例を示す.

表 3.3 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

3.2.4 除法表作成

$\alpha^i \div \alpha^j$ の計算結果を α^i 行 α^j 列の要素へ格納する. $0 \div \alpha^i, \alpha \div 0$ の値を 0 とする. 除法表のその他の要素は乗法表から値を求める. 乗法表における

$$\alpha^i * \alpha^j = \alpha^h \tag{3.11}$$

は,

$$\alpha^j = \alpha^h \div \alpha^i \tag{3.12}$$

と変形できる. この式変形を表で示すと表 3.4 のようになる.

表 3.4 の右表は $\alpha^3 \div \alpha^2$ の値が α^1 であることを示しており, $\alpha^2 \div \alpha^3$ の値ではないことに注意する. 以下に除法表を作成する手順を示す.

除法表作成手順

1. 0 行 α^i 列, α^i 行 0 列に 0 を代入する.
2. $i = 1$ から $2^m - 2$ まで 3 を繰り返す.
3. $j = 1$ から $2^m - 2$ まで 4,5 を繰り返す.

表 3.4　乗法表を除法表に変換

$*$	$\cdots$	α^1	α^2	α^3	$\cdots$
$\vdots$		$\vdots$			
α^1		$\vdots$			
α^2	$\cdots$	α^3			
α^3					
$\vdots$					

$\div$	$\cdots$	α^1	α^2	α^3	$\cdots$
$\vdots$			$\vdots$		
α^1			$\vdots$		
α^2			$\vdots$		
α^3	$\cdots$	$\cdots$	α^1		
$\vdots$					

$$\alpha^2 * \alpha^1 = \alpha^3 \quad \rightarrow \quad \alpha^1 = \alpha^3 \div \alpha^2$$

表 3.5　GF (2^3) の除法表

$\div$	0	α^0	α^1	α^2	α^3	α^4	α^5	α^6
0	0	0	0	0	0	0	0	0
α^0	0	α^0	α^6	α^5	α^4	α^3	α^2	α^1
α^1	0	α^1	α^0	α^6	α^5	α^4	α^3	α^2
α^2	0	α^2	α^1	α^0	α^6	α^5	α^4	α^3
α^3	0	α^3	α^2	α^1	α^0	α^6	α^5	α^4
α^4	0	α^4	α^3	α^2	α^1	α^0	α^6	α^5
α^5	0	α^5	α^4	α^3	α^2	α^1	α^0	α^6
α^6	0	α^6	α^5	α^4	α^3	α^2	α^1	α^0

4. 乗法表を参照して $\alpha^i * \alpha^j$ の値 α^h を取り出す.

5. 除法表の α^h 行 α^i 列に α^j を代入する.

表 3.5 に作成される除法表の例を示す.

3.3 拡大体上の (k, n) しきい値法の実現

3.3.1 分散符号化

分散符号化処理では, 秘密情報から, k 個以上集めると復元できるが $k-1$ 個以下では復元できないような 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. $\boldsymbol{GF}(2^m)$ の乗法表, 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, 作成した演算表から演

算結果を取り出す. $(i = 1, 2, \cdots, n)$

$$f(x_i) = S + \sum_{j=1}^{k-1} r_j {x_i}^j \tag{3.13}$$

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. 分散符号化で扱う演算は加法・乗法であるため, 乗法表・対応表・加法表を作成するが除法表は作成しない. 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. ここでは表引き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.

3.3.2 復号化

分散符号化された ID とシェア k 個から, 秘密情報を復元する.

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元できない.
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元できない.

初期設定

1. k 人が各々の持つ $ID = x_{i_j}$ と $w_{i_j} \quad (1 \leq j \leq k)$ を持ち寄る.
2. $\boldsymbol{GF}(2^m)$ の乗法表, 対応表, 加法表, 除法表を作成する.
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.

$$f(x) = S + r_1 x + r_2 x^2 + \cdots + r_{k-1} x^{k-1} \tag{3.14}$$

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. 分散符号化で扱う演算は加法・乗法・除法であるため, 乗法表・対応表・加法表・除法表を作成する. 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 B 参照) を用いて計算する. ただし演算はすべて表引き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.

3.4 評価

今回は分散符号化, 復号化を行うたびに演算表を作成しているが, 実際は全く同じ演算表を用いなければならないため, 最初に 1 度演算表を作成しておき, 両方の処理でこの演算表を用いると効率が良い. 演算表を保存しておけば処理を行う際に演算表を作成する必要がなくなる. これによって処理はほとんど表引きだけで行われるため, 処理にかかる時間を大幅に減らすことが可能となる. ただしこの場合には演算表の保管方法が問題となる. 分散符号化で使われた拡大次数をシェアファイルに書き込むなど, 何らかの方法で保管しておけば同じ演算表を再度作成することができるため, 演算表を紛失した場合には対処できる. 演算表を改ざんされた場合に対しては, それを発見する方法が必要である. 改ざんされていることさえわかれば, 保管している拡大次数値を用いて再度演算表を作成すればよい.

乗法表を作成すれば, 除法演算結果を乗法表から取り出すことが可能である. しかし乗法

表を用いて除法演算を行う場合, 表への平均アクセス回数は $2^m/2$ 回となる. 除法表を作成した場合, 除法演算は表へ 1 回アクセスしただけで演算結果が求められる. 前に述べたように演算表を保管しておく場合, 演算表作成にかかる時間は処理時間には含まれない. したがって除法表を作成しておくことで, さらに演算処理にかかる時間を短縮できる. ただし除法表を保管しておく容量が必要となる.

第 4 章

結論

4.1 結論

秘密情報を保管する際, 盗聴や喪失などの危険がある. これらへの対処方法のひとつとして秘密分散法がある. 本研究では, 秘密分散法を効率良く実行する手段として拡大体上での実現を試みた. 素体上で秘密分散法を実現する際に素数と原始根を求める演算に時間がかかる点, 拡大体が 2 進数とうまく対応する性質をもっているためコンピュータ内の演算において空間を有効に利用できる点などを考慮したものである.

本研究では拡大体上で秘密分散法を実現する際の演算処理を効率化することを主な課題とした. これに対し, 前もって演算表を作成しておき, 演算の際には表から値を取り出すことによって処理の効率化を実現した. 拡大体上の演算方法を考慮して, 表の作成方法と参照方法を定義した. この演算表を (k, n) しきい値法で利用すれば, 分散符号化・復号化処理では表引きをすることで直ちに演算結果を求められる.

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演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

3. 分散符号化が正しく行われたかどうかの判定
4. 他の暗号システムとの併用

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. 秘密分散法では 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, 終始御指導並びに御鞭撻を賜わりました高知工科大学情報システム工学科の福本昌弘講師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深く感謝致します.

また, 本論文の文書スタイルファイルを作成してくださった橋本学氏, 井上富幸氏, 中平拓司氏に心より感謝致します.

本研究を遂行するにあたり, 御助言を賜わりました電子・光システム工学科の関口晃司助教授に深謝致します.

最後に, 御協力を頂いた福本研究室の皆様方にも深く感謝致します.

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] 土居範久, 小山謙二, コンピュータセキュリティ, 共立出版, 1986.

[6] Neal Koblitz, (林彬 訳), 暗号の代数理論, シュプリンガー・フェアラーク東京, 1999.

[7] D.E.R. デニング, (上園忠弘, 小嶋格, 奥島晶子 訳), 暗号とデータセキュリティ, 培風館, 1988.

[8] 今井秀樹, 情報理論, 昭晃堂, 1984.

[9] 芹沢正三, C による初等整数論, 森北出版, 1993.

[10] 遠山啓, 初等整数論, 日本評論社, 1972.

[11] 渡辺啓一, 草場公邦, 代数の世界, 朝倉書店, 1994.

[12] 寺田文行, 数理・情報系のための代数系の基礎, サイエンス社, 1990.

[13] 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.

[14] 松坂和夫, 代数系入門, 岩波書店, 1976.

付録 A

線形方程式が独立であることの証明

ここでは, 式 (2.5) 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 (2.5) は体上で一意の解を持つ.

付録 B

ガウスの消去法

連立 1 次方程式

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

の解法を示す.

1. まず,

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

とおく. 連立方程式を

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

と行列で表す.

2. $\widehat{A} = [A|b]$ とおくと,

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

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,

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

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.

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

この式から n 個の方程式が得られる.

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

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.