信州大学工学部

学士論文

秘密分散法のラテン方陣を用いた表現に関する基礎的検討

指導教員　西新 幹彦 准教授

学科　電気電子工学科
学籍番号　12T2099K
氏名　吉森　祐希

2016 年 2 月 22 日

目次

1　はじめに

重要な情報は紛失と漏洩の危険にさらされているため, 管理方法が大切になる. 例えば, 情報の紛失を防ぎたいならば, 大量のコピーを取ればいいが数が増える分, 漏洩する危険が高くなる. では, コピーを取らずに管理をした場合はというと, そのデータを紛失すると復元することができなくなってしまう. このように紛失と漏洩に対する対策は一見互いに矛盾しており, 両方同時に対策することは難しいように思われる. しかし, この問題を解決する方法として秘密分散法という手法がある. 秘密分散法の仕組みでは, 守りたい秘密情報をもとに, ディーラーと呼ばれる者が n 個のシェアと呼ばれるものを作成する. 作成されたシェアはユーザーに配布される. もし, 秘密情報が必要になった場合には n 人のうち k 人のユーザーが集まることで元の秘密情報が復元できる. また, $k-1$ 人のユーザーが集まっても秘密情報を復元することはできない仕組みになっている. この方法は完全型 (k,n) しきい値法と呼ばれる.

この完全型 (k,n) しきい値法の実現法として, 有限体上で Shamir の多項式補間法を用いることでシェアと復号器を作成する方法が有名である. しかし, 文献 [3] によれば, 有限体を用いることなくラテン方陣を用いて $(2,3)$ しきい値法を表現できる場合があることから, 有限体を定義できないサイズの有限集合でも (k,n) しきい値法が構成できる可能性が指摘されている.

本研究ではラテン方陣を用いて (2,3) しきい値法を表現することによって, Shamir の多項式補間法では求めることができないサイズ 3 の (2,3) しきい値法について調べた. また, 有限体が定義できない 6 をサイズとする (2,3) しきい値法が存在するかどうかを調べた.

2　確率的な秘密分散法

はじめに, 確率的な秘密分散法の定義について確認する.

2.1　定義

秘密情報 S およびシェア $W_j(j=1,\cdots,n)$ があり, いずれもある有限集合上の値をとるものとする. ここで、S, W_j は共に確率変数とする.

定義 1　秘密情報 S とシェア $(W_1, W_2, \ldots, W_n)$ が次の 2 条件を満たすとき完全型 (k,n) しきい値型秘密分散法をなすという [1].

1. 任意の相違なる k 個のシェア $(W_{j_1}, W_{j_2}, \ldots, W_{j_k})$ から S が正しく復号できる. つまり,

$$H(S|W_{j_1}, W_{j_2}, \ldots, W_{j_k}) = 0 \tag{1}$$

が成り立つ.

2. 任意の $k-1$ 個のシェアから秘密情報 S は全く得られない. つまり,

$$H(S|W_{j_1}, W_{j_2}, \ldots, W_{j_{k-1}}) = H(S) \tag{2}$$

が成り立つ.

上式で定義される完全型 (k, n) しきい値型秘密分散法のことを本論文では確率的な (k, n) 秘密分散法, または単に確率的な秘密分散法と呼ぶ.

2.2 Shamir の多項式補間法

確率的な秘密分散法の有名な例として Shamir の多項式補間法 [2] という手法がある. この方法による秘密情報からシェアを作る手順と秘密情報を復元する手順を説明する.

はじめに, 秘密情報やシェアは有限体 $\mathbb{F}$ 上に値をとるとする. 秘密情報 S の分布は任意でよい. 係数 $a_1, a_2, \cdots, a_{k-1}$ を $\mathbb{F}$ 上に一様に分布する確率変数とし, 多項式

$$f(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1 x + S \tag{3}$$

を定める. これを元に, ディーラーが互いに相異なる非零の $x_i \in \mathbb{F}(i = 1, 2, \ldots, n)$ に対し,

$$W_i = f(x_i) \tag{4}$$

を作る. この W_i をシェアとし、ユーザーにそれぞれ秘密裏に配布する. $S, a_1, \ldots, a_{k-1}$ については非公開とする.

シェア保有者が協力して k 個のシェアを集めることによって, 式 (3) をもとめることができ, 秘密情報 S を復元できる. つまり, 定義 1.1 が成り立つ. さらに, $k-1$ 個のシェアからでは未知数が k 個ある式 (3) を求められず, 秘密情報 S を一意に定めることができないため, 復元できない. つまり, 定義 1.2 が成り立つ.

3 非確率的な秘密分散法

ここで, 文献 [3] に基づき，非確率的な秘密分散法の定義を導入する.

3.1 非確率的な秘密分散法の定義

秘密情報 s および w_j がある有限集合 $\mathbb{F}$ 上の値をとるものとする.

定義 2　秘密情報 s とシェア $(w_1, w_2, \cdots, w_n)$ が次の 2 条件を満たすとき非確率的な秘密分散法をなすという.

1. ある写像 $\psi : (\mathbb{N} \times \mathbb{F})^k \to \mathbb{F}$ が存在し,

$$s = \psi(j_1, w_{j_1}, j_2, w_{j_2}, \ldots, j_k, w_{j_k}) \tag{5}$$

が成り立つ. ここで $\mathbb{N} \triangleq \{1, \cdots, n\}$ とする。. ただし、$j_1, j_2, \ldots, j_k$ は互いに相異なる. この時の ψ は復号器としての役割を果たす.

2.$\mathbb{F} = \psi(j_1, w_{j_1}, j_2, w_{j_2}, \ldots, j_{k-1}, w_{j_{k-1}}, j_k, \mathbb{F})$ となる. すなわち,

$$g(\cdot) = \psi(j_1, w_{j_1}, j_2, w_{j_2}, \ldots, j_{k-1}, w_{j_{k-1}}, j_k, \cdot) \tag{6}$$

が全単射となる.

式 (5) は k 個のシェアから秘密情報を復元できることを意味する. また, 式 (6) が全単射の関係にあるということは k 個目の値を取り替えると, 得られる値が $\mathbb{F}$ 上で重複することなく取り変わるということを意味する. つまり, $k-1$ 個のシェアが集まったとしても, 秘密情報について情報が得られないということである. また, 秘密情報 s とシェア $(w_1, w_2, \cdots, w_n)$ の値は非公開情報だが, 復号器 ψ は非公開情報を含んでいないため公開してもかまわない.

3.2 非確率的な秘密分散法の例

2.2 節で説明した Shamir の多項式補間法は非確率的な秘密分散法でもある. そのことを以下の通り示す. シェアの作り方は基本的に 2.2 節と同じで, 秘密情報 s と係数 $a_1, a_2, \cdots, a_{k-1}$ の値は非公開であるが, これらは確率変数ではない.

秘密を復元するための復号器は次のようにして導くことができる. まず, 式 (3)(4) で定まる k 個のシェアを行列で表現すると

$$\begin{bmatrix} w_{j_1} \\ w_{j_2} \\ \vdots \\ w_{j_k} \end{bmatrix} = \begin{bmatrix} x_{j_1}^{k-1} & x_{j_2}^{k-2} & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{j_k}^{k-1} & x_{j_k}^{k-2} & \ldots & 1 \end{bmatrix} \begin{bmatrix} a_{k-1} \\ a_{k-2} \\ \vdots \\ s \end{bmatrix} \tag{7}$$

となる. ここで, 右辺の正方行列を [X] と置く.[X] はヴァンデルモンド行列となっているため逆行列が必ず存在する. この証明は付録に記す. 式 (7) の両辺に左から $[X]^{-1}$ をかけると,

$$\begin{bmatrix} a_{k-1} \\ a_{k-2} \\ \vdots \\ s \end{bmatrix} = \begin{bmatrix} x_{j_1}^{k-1} & x_{j_2}^{k-2} & \ldots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ x_{j_k}^{k-1} & x_{j_k}^{k-2} & \ldots & 1 \end{bmatrix}^{-1} \begin{bmatrix} w_{j_1} \\ w_{j_2} \\ \vdots \\ w_{j_k} \end{bmatrix} \tag{8}$$

となる. $[X]^{-1}$ の k 行目部分を $[\cdots]$ と置くと

$$\begin{bmatrix} \ldots \end{bmatrix} \begin{bmatrix} w_{j_1} \\ w_{j_2} \\ \vdots \\ w_{j_k} \end{bmatrix} = s \tag{9}$$

となる．これは $j_1, w_{j_1}, j_2, w_{j_2}, \ldots, j_k, w_{j_k}$ の値から秘密情報 s を復号する方法を示しており，式 (5) で表される復号器の構造に他ならない．つまり，

$$s = \begin{bmatrix} \ldots \end{bmatrix} \begin{bmatrix} w_{j_1} \\ w_{j_2} \\ \vdots \\ w_{j_k} \end{bmatrix} = \psi(j_1, w_{j_1}, j_2, w_{j_2}, \ldots, j_k, w_{j_k}) \tag{10}$$

となり，定義 2.1 が満たされる．

また，x_i が相異なる非零の値であることから $[\cdots]$ の各要素も非零であることがいえる．この証明も付録に記す．$[\cdots]$ の各要素が非零であることから，s の値は $w_{j_1}, \cdots, w_{j_k}$ のすべてに依存し，有限体の演算の性質から s と w_{j_k} が 1 対 1 であることがわかる．このことから定義 2.2 が満たされることがわかり，$k-1$ 個のシェアから秘密情報 S は復元できない．以上のことから Shamir の多項式補間法が非確率的な秘密分散法となっていることが示された．

4　ラテン方陣を用いた秘密分散法

4.1　ラテン方陣

本論文ではシェアのサイズと秘密情報 S のサイズが等しいと仮定しているので，復号器 ψ は複数のラテン方陣で表現できる．n 行 n 列の表の各行各列に，それぞれ n 種類の記号が重複しないように現れる表のことを n 次のラテン方陣またはサイズ n のラテン方陣と言う．例として，図 1 にサイズ 5 のラテン方陣の例を示す．図 1 に示したように，ラテン方陣には各行各列に同じ数字が表れない性質がある．この性質の特徴から，シェアをラテン方陣のマスの座標とすることで，任意の k 個のシェアで秘密情報の一点が定まる．つまり，秘密情報が復元される．また，$k-1$ 個のシェアではラテン方陣内の取りうる値に全ての数字が現れるような復号器を実現できる．

0	1	2	3	4
1	2	3	4	0
2	3	4	0	1
3	4	0	1	2
4	0	1	2	3

図 1　サイズ 5 のラテン方陣の例

4.2　ラテン方陣を用いた (2,2) しきい値法

図 1 のラテン方陣に座標を与えたものを図 2 に示す. 図 2 において, 座標部分がシェアを, ラテン方陣が復号器を表す. また, シェアは図 2 のようにそれぞれ w_1, w_2 とする. 図 2 を用いてシェアを作ったり秘密情報を復号できることを説明する.

まず, 秘密情報が s であると仮定する. 次に, 一方のシェア w_1 の値を他人に知られないようにディーラーが勝手に選ぶ (確率的に選択するという意味ではない). また, 他方のシェア w_2 の値はラテン方陣の w_1 行 w_2 列の値が秘密情報 s と等しくなるように選ぶ. 例として, 秘密情報 s が 3 だとする. 次に, ディーラーがシェア w_1 の値を 2 に決めたとする. ここで図 2 を見ると, $w_1 = 2$ で秘密情報が 3 の場合もう一方のシェア w_2 の値は 1 であることが分かる. つまり, シェアの値は、$w_1 = 2, w_2 = 1$ と設定される.

次に, 秘密情報がどのように復号されるかを考える. まず, シェア w_1 のみを入手した場合を考える. シェア w_1 から得られる情報は, 図 2 よりシェア w_2 のとりうる値と秘密情報のとりうる値は全単射となっているため, w_2 の値に対して同じ数の秘密情報が出現し, w_1 のみでは秘密情報を復号できない. シェア w_2 のみを入手した場合も同様に, 秘密情報は全く分からない. 2 つのシェア $w_1 = 2, w_2 = 1$ を入手した場合は, 図 2 より, 2 つのシェアの値から 1 マスの座標が求まり, そのマスの情報が秘密情報であるため, 2 つのシェアから秘密情報が得られる. この場合は秘密情報として 3 が得られる.

このようにして, 指定されたサイズの任意のラテン方陣を用いて非確率的な $(2, 2)$ 秘密分散法を表現することができる.

		w_2				
		0	1	2	3	4
	0	0	1	2	3	4
	1	1	2	3	4	0
w_1	2	2	3	4	0	1
	3	3	4	0	1	2
	4	4	0	1	2	3

図 2　非確率的な $(2, 2)$ しきい値法の復号表の例

4.3　ラテン方陣を用いた (2,3) しきい値法

4.2 節で説明した方法を拡張することで, (2,3) しきい値法もラテン方陣を用いて表現できる. (2,3) しきい値法は, 3 個のシェアのうち 2 個のシェアを集めることで秘密情報が復元できるという方法である. つまり, 2 個のシェアの取りうる組み合わせをそれぞれラテン方陣で表現す

ることで (2,3) しきい値法も表現できる. つまり, 秘密情報を復号できるシェアの組み合わせは 3 通りであるため, 3 個のラテン方陣を用いて表現できる. しかし, 3 個のラテン方陣は任意でよいというわけではなく, どのシェアの組み合わせでも同じ秘密情報が復号できるような 3 個のラテン方陣でなければならない. ここで, どのシェアの組み合わせでも同じ秘密情報が復号できることを整合性を満たしている, または整合性が取れているという. 非確率的な (2,3) しきい値法は, 整合性を満たした 3 個のラテン方陣の組み合わせを作ることで構成できる. 従って, 2.2 節で紹介した Shamir の多項式補間法で構成された (2,3) しきい値法も非確率的な (2,3) しきい値法であるので 3 個のラテン方陣で表現できる. 以下で Shamir の多項式補間法で構成された (2,3) しきい値法を整合性の取れた 3 個のラテン方陣で表現できることを確認する.

3 つのシェアを w_1, w_2, w_3 とし, 3 つのシェアの取る値は位数 5 の有限体 $\mathbb{F}$ の元とする. まず, 1 次多項式 $f(x) = ax + s$ を考える. この式を元に 3 つのシェアを作るための式は

$$
\begin{aligned}
w_1 &= f(1) = a + s \\
w_2 &= f(2) = 2a + s \\
w_3 &= f(3) = 3a + s
\end{aligned}
$$

となる. この 3 式から係数 a を消去して S についての式に直すと

$$
\begin{aligned}
s &= 2w_1 - w_2 \\
s &= 3w_2 - 2w_3 \\
s &= 4w_1 + 2w_3
\end{aligned}
$$

となる. この 3 式より, 図 3 の復号器 ψ となるラテン方陣の表を作ることができる.

ここで, 図 3 の表が整合性を満たしているかを確認する. 例として, 1 個目の表 (w_1-w_2 の表) における $w_1 = 2, w_2 = 1$ のマスにあたる 3 を秘密情報に設定する. したがって, シェア w_1, w_2 の値は $w_1 = 2, w_2 = 1$ となる. 次に, 2 個目の表 (w_2-w_3 の表) において $w_2 = 1$ の行に注目すると, その行における 3 にあたるマスの w_3 の値は 0 であるため, $w_3 = 0$ と求まる. 最後に, 3 個目の表 (w_1-w_3 の表) において $w_1 = 2, w_3 = 0$ のマスを確認すると, 秘密情報 3 が復号されているため, どのシェアの組み合わせでも同じ秘密情報が復号されていることが確認できる. 同様にして, どのマスを秘密情報としてもすべてのラテン方陣で正しい秘密情報が復号できることがわかり, 整合性を満たすことが確認できる.

一般に, Shamir の多項式補間法に限らず, 任意の (2,3) しきい値法は 3 個のラテン方陣で表すことができる. さらに $n \geq 2$ に対し, 任意の $(2, n)$ しきい値法も整合性を満たす複数のラテン方陣で表現できる.

		w_2				
		0	1	2	3	4
	0	0	4	3	2	1
	1	2	1	0	4	3
w_1	2	4	3	2	1	0
	3	1	0	4	3	2
	4	3	2	1	0	4

		w_3				
		0	1	2	3	4
	0	0	3	1	4	2
	1	3	1	4	2	0
w_2	2	1	4	2	0	3
	3	4	2	0	3	1
	4	2	0	3	1	4

		w_3				
		0	1	2	3	4
	0	0	2	4	1	3
	1	4	1	3	0	2
w_1	2	3	0	2	4	1
	3	2	4	1	3	0
	4	1	3	0	2	4

図 3　非確率的な $(2,3)$ しきい値法の復号表の例

5　サイズ 3 の (2,3) しきい値法

Shamir の多項式補間法で (2,3) しきい値法を考える場合, 有限体のサイズが 4 以上でなければならない. その理由はサイズが 3 以下の場合について考えると分かる. サイズが 2 以下の場合, そもそも 3 個のシェアが作れない. サイズが 3 の場合, 3 個のシェアを作ることができるが, 補間法の性質から 1 つのシェアが秘密情報そのものとなってしまい, そのシェア 1 つから秘密情報が復元できてしまうため, $(2,3)$ しきい値法にならない.

そこで, Shamir の多項式補間法や有限体で表現されるという制約を取り除き, 3 章で定義した非確率的秘密分散法の意味でサイズ 3 の (2,3) しきい値が存在するかどうか調べた. その方法はプログラムによる探索であるが, 4 章の考察から, この問題はラテン方陣を探索する問題に帰着する.

5.1　探索アルゴリズム

探索アルゴリズムの概要を以下に示す.

step1　1 個目のラテン方陣 (w_1-w_2 の表) を作成.

step2　2 個目のラテン方陣 (w_2-w_3 の表) を作成を開始する.

step3　2 個目のラテン方陣の 1 行目のマスから順にマスの値を候補の中から仮決めしていく.

step4　step3 で仮決めされた値から, 整合性が取れるように 3 個目のラテン方陣のある 1 マスの値を決める.

step5　2 個目と 3 個目のラテン方陣の全マスを確認し, 候補が無いマスを探す. 無い場合 step6 に移る. 有る場合 step7 に移る.

step6　2 個目と 3 個目のラテン方陣の全マスが埋っているかを確認する. 埋まっている場合, 2 個目のラテン方陣と 3 個目のラテン方陣が同時に完成する. 埋まってない場合, この時点での 2 個目と 3 個目のラテン方陣のコピーを取り step3 に移る.

step7　最後に取ったコピーの状態に戻り, step3 に移る.

次に, 上記それぞれの作業について詳しく説明する.

1. step1 の説明

   まず, ラテン方陣を作るために下図のような表を用意する

		w_2					
		0		1		2	
w_1	0	0 2	1	0 2	1	0 2	1
	1	0 2	1	0 2	1	0 2	1
	2	0 2	1	0 2	1	0 2	1

図 4　ラテン方陣を作るために用意した表

図 4 は全てのマスに 0,1,2 の 3 つの候補がある表を表す. まず, 図 4 の表の 1 行 1 列目から 4 つの候補 (0,1,2 の何れか) から 1 つの値を仮決めする. 例として 0 に仮決めしたとする. 次に, 仮決めされたマスのある列と行の全てのマスの候補から仮決めの値を消去する. この処理を行うことで, 各行各列にその値は 1 度しか現れなくなり, ラテン方陣となる条件を満たす. つまり, 1 行 1 列のマス以外の 1 行目のマスと 1 列目のマスの候補から 0 を消去するため, 図 4 の表は図 5 のようになる.

		w_2		
		0	1	2
w_1	0	0 2	1 2	1 2
	1	1 2	0 1 2	0 1 2
	2	1 2	0 1 2	0 1 2

図 5　ラテン方陣を作る途中経過を表した表

このように, 1 行目のマスから順に仮決めをしていき全てのマスの値が決まるとラテン方陣が完成する.

2. step2,step3 の説明

   1 個目の表を作成する作業と同様に, 2 個目の表のある 1 マスの値を候補の中から仮決めする.

3. step4 の説明

   1 個目のラテン方陣が準備された状態で 2 個目の表のある 1 マスの値が決定すると, その値は整合性の性質を用いて 3 個目の表に反映することができる. このことを具体例を用いて紹介する.

   図 6 を 1 個目のラテン方陣とする.

		w_2		
		0	1	2
w_1	0	0	1	2
	1	1	2	0
	2	2	0	1

図 6 w_1-w_2 の表 (1 個目のラテン方陣)

次に, 2 個目の表の 1 行 1 列目のマスの値を 2 と仮決めした場合 ($w_2 = 0, w_3 = 0$ の場合, 秘密情報 $s = 2$), 2 個目の表の様子は図 7 のようになる.

		w_3					
		0		1		2	
				0	1	0	1
	0	2					
		0	1	0	1	0	1
w_2	1			2		2	
		0	1	0	1	0	1
	2			2		2	

図 7　w_2-w_3 の表 (2 個目のラテン方陣)

ここで図 6 より, $w_2 = 0$ において $s = 2$ となるような w_1 の値は 2 であることが分かる. つまり, $w_1 = 2, w_3 = 0$ のマスの値は 2 であることが確定する. また, $w_1 = 2, w_3 = 0$ のマス (w_1-w_3 の表の 3 行 1 列目のマス) の値は 2 であることが確定したことから 3 行 1 列目のマス以外の 3 行目と 1 列目の全てのマスの候補から 2 が消去される. この時の 3 個目の表の様子を図 8 に示す.

		w_3					
		0		1		2	
		0	1	0	1	0	1
	0			2		2	
		0	1	0	1	0	1
w_1	1			2		2	
				0	1	0	1
	2	2					

図 8　w_1-w_3 の表 (3 個目のラテン方陣)

しかし, $w_2 = 0, w_3 = 0$ のマス (図 7 の 1 行 1 列目のマス) の値の 2 だけが 3 個目の表に反映されるわけではない. $w_2 = 0, w_3 = 0$ のマスの値が 2 と確定すると, $w_2 = 0, w_3 = 0$ のマスには 0 は絶対に当てはまらないことは自明である. つまり, 2 個目の表の $w_2 = 0, w_3 = 0$ のマスの 0 に対応する 3 個目の表の $w_1 = 0, w_3 = 0$ のマスの 0 は候補から消去される. 2 個目の表の $w_2 = 0, w_3 = 0$ のマスの 1 に関しても同様の処理が行われる. この処理を行った後の 3 個目の表の様子を図 9 に示す.

		w_3		
		0	1	2
w_1	0	1 2	0 1 2	0 1 2
	1	0 2	0 1 2	0 1 2
	2	2	0 1	0 1

図 9　w_1-w_3 の表 (3 個目のラテン方陣)

また, 図 9 の $w_1 = 2, w_3 = 0$ のマスの候補から消去された 0,1 についても同様に 2 個目の表に反映される. この処理後の 2 個目の表の様子を図 10 に示す.

		w_3		
		0	1	2
w_2	0	2	0 1	0 1
	1	1 2	0 1 2	0 1 2
	2	0 2	0 1 2	0 1 2

図 10　w_2-w_3 の表 (2 個目のラテン方陣)

このように, 2 個目のある 1 マスの値が決まると上記で説明した一連の処理が行われる.

4. step5 の説明

step4 での一連の処理が行われた後, 2 個目の表と 3 個目の表の全てのマスのそれぞれの候補の数を数える作業を行う.

		w_3		
		0	1	2
w_2	0	2	1 2	0 2
	1	1 2	0 2	2
	2	2	2	1

		w_3		
		0	1	2
w_1	0	1 2	2	0
	1	0 2	1 2	2
	2	2	0 2	1

図 11　w_2-w_3 の表 (2 個目のラテン方陣) と w_1-w_3 の表 (3 個目のラテン方陣)

もし, この作業中に図 11 の 1 行 2 列目のマスのように候補が 1 つも無いマスが発見された場合, そのマスの値を決めることができなくなので, ラテン方陣を作れない. 従って, step8 の作業に移行する. 2 個目の表と 3 個目の表の全てのマスに候補が 1 つでも存在した場合, step6 の作業に移行する.

5. step6 の説明

step5 までの作業を終えた段階で, もし図 12 のように全てのマスの値が定まっていた場合, (2,3) しきい値法が成り立つ 3 個のラテン方陣の組み合わせが完成する. また, 見つかったラテン方陣の組はアルゴリズムの出力として出力される.

		w_3		
		0	1	2
w_2	0	0	1	2
	1	2	0	1
	2	1	2	0

		w_3		
		0	1	2
w_1	0	0	2	1
	1	2	1	0
	2	1	0	2

図 12　w_2-w_3 の表 (2 個目のラテン方陣) と w_1-w_3 の表 (3 個目のラテン方陣)

もし図 13 のように候補が 2 個以上あるマスがある場合, この状態の表の両方のコピーを取って step3 に戻り, 候補が 2 個以上あるマスの値を仮決めして一連の処理を行う.

		w_3		
		0	1	2
w_2	0	2	0 1 2	0 1
	1	1	0 1 2	0 1 2
	2	0	0 1 2	0 1 2

		w_3		
		0	1	2
w_1	0	1	0 1 2	0 1 2
	1	0	0 1 2	0 1 2
	2	2	0 1	0 1

図 13　w_2-w_3 の表 (2 個目のラテン方陣) と w_1-w_3 の表 (3 個目のラテン方陣)

6. step7 の説明

図 11 の状態になる前のコピーを図 14 に示す.

		w_3					
		0		1		2	
w_2	0	2		0	1	0	1
	1		1	2		2	
	2	0		0	1	0	1
				2		2	

		w_3					
		0		1		2	
w_1	0		1	0	1	0	1
				2		2	
	1	0		0	1	0	1
				2		2	
	2	2		0	1	0	1

図 14　w_2-w_3 の表 (2 個目のラテン方陣) と w_1-w_3 の表 (3 個目のラテン方陣)

step5 で図 11 のような候補の無いマスが発見された場合, 図 11 の状態の表を捨て図 14 のコピーの状態に切り替え, step3 に戻り一連の作業を行う.

以上のアルゴリズムを実装したプログラムを用いてサイズ 3 の (2,3) しきい値法の探索を行った. 探索に用いたプログラムを付録 B に記す.

5.2　結果と考察

プログラムで探索した結果, 72 通りの整合性の取れたラテン方陣の組み合わせが見つかった. プログラムで見つかったサイズ 3 の $(2,3)$ しきい値法の 1 つを図 15 に示す.

		w_2		
		0	1	2
w_1	0	0	1	2
	1	1	2	0
	2	2	0	1

		w_3		
		0	1	2
w_2	0	0	1	2
	1	2	0	1
	2	1	2	0

		w_3		
		0	1	2
w_1	0	0	2	1
	1	2	1	0
	2	1	0	2

図 15　非確率的な $(2,3)$ しきい値法 (サイズ 3) の復号表の例

ここで, 見つかったラテン方陣の組み合わせに基づき，サイズ 3 の (2,3) しきい値法について考察する. すでに述べた通り, Shamir の多項式補間法ではサイズ 3 の (2,3) しきい値法を実現できない. しかしこれは, シェアと秘密情報の関係が有限体を使って表すことが否定されたわけではない. そこで, このことを以下のように検証した. 図 15 の例で使われているラテン方陣をそれぞれシェア w_1, w_2, w_3 と秘密情報 S の関係式で表してみると

$$w_1 + w_2 = S$$
$$2w_2 + w_3 = S$$
$$2w_1 + 2w_3 = S$$

と書くことができる. このようにラテン方陣からシェアと秘密情報の関係式が導出できたことから, サイズ 3 のラテン方陣を全て数式で表せるのではないかと予想した.

サイズ 3 のラテン方陣の数は全部で 12 個存在する. 各ラテン方陣を検証した結果, サイズ 3 のラテン方陣は何れも

$$w_1 + w_2 + b = S$$
$$2w_1 + w_2 + b = S$$
$$w_1 + 2w_2 + b = S$$
$$2w_1 + 2w_2 + b = S$$

という形の式で表せることが分かった. ここで, b は定数項である. このような形の式は定数項の値を変えることで 12 個存在する. この 12 個の式は 12 個のラテン方陣と 1 対 1 で対応することが分かった. 付録 A.3 にサイズ 3 の全てのラテン方陣と対応する式を示す.

次に, 整合性を満たす 72 通りのラテン方陣の組み合わせについても法則があるかを検証する. まず, 整合性を満たす 3 個のラテン方陣を作れるかどうかは, 1 個目と 2 個目のラテン方陣を用意すれば, その 2 つから確認することができる. では, 1 個目と 2 個目のラテン方陣の選び方は 12 通りの中から任意に 2 つ選んで良いのか, または選び方には法則があるのかということに疑問が出る. そこで, 72 通りの組み合わせ全てを確認することによりこの疑問について検証してみた. その結果, 整合性を満たす 3 個のラテン方陣を構成するためには 1 個目のラテン方陣と 2 個目のラテン方陣には 2 つのパターンの選び方があることが確認された.

パターン 1　1 個目のラテン方陣を図 17 の (1),(3),(4),(5),(7),(11) の表の中から 1 つ選び, 2 個目のラテン方陣を図 17 の (2),(3),(4),(5),(6),(12) の表の中から 1 つ選ぶ

パターン 2　1 個目のラテン方陣を図 17 の (2),(6),(8),(9),(10),(12) の表の中から 1 つ選び, 2 個目のラテン方陣を図 17 の (1),(7),(8),(9),(10),(11) の表の中から 1 つ選ぶ

ここで, 1 つ目のパターンの組み合わせは 36 通りであり, 2 つ目のパターンの組み合わせも 36 通りである. また, 2 つのパターンは互いに排反であるため, この 2 つのパターンの合計は 72 通りとなり整合性を満たす 3 個のラテン方陣の組み合わせ数と一致する. 従って, 1 個目と 2 個目のラテン方陣は 12 通りの中から任意に 2 つ選んでよいというわけではなく, 上記に書いたような選び方があることが分かった. しかし, なぜこのような 2 つのパターンになるのかは分からなかったため, 今後の課題としたい.

6　サイズ 4 の $(2, 3)$ しきい値法

5 章では, 3 個のラテン方陣の組み合わせを確認することで, Shamir の多項式補間法では作ることができないサイズ 3 の (2,3) しきい値法が作れて, それを全て数式で表せることが分

かった. そこで, 他のサイズの (2,3) しきい値法に関しても, Shamir の多項式補間法ではない方法で作ることができる (2,3) しきい値法が存在し, それらを数式化できないかと考えた. この章では, サイズ 4 の (2,3) しきい値法の全ての組み合わせを数式化できるかを考えた.

サイズ 4 の (2,3) しきい値法の全てはサイズ 4 をラテン方陣の組み合わせによって作ることができる. つまり, サイズ 4 の (2,3) しきい値法を全て数式化するためには, サイズ 4 のラテン方陣を数式化できればよい. 5.1 節で説明したアルゴリズムの step1 の機能を用いて探索した結果, サイズ 4 のラテン方陣は全部で 576 通り見つかった. 一方, 6 章のように ラテン方陣を表す 1 次式を考えるとそれらは

$$
\begin{aligned}
w_1 + w_2 + b &= S \\
w_1 + 2w_2 + b &= S \\
w_1 + 3w_2 + b &= S \\
2w_1 + w_2 + b &= S \\
2w_1 + 2w_2 + b &= S \\
2w_1 + 3w_2 + b &= S \\
3w_1 + w_2 + b &= S \\
3w_1 + 2w_2 + b &= S \\
3w_1 + 3w_2 + b &= S
\end{aligned}
$$

のようになり, その数を考えるとシェアの係数の組み合わせの数 (9 通り) と定数項 b=0,1,2,3(4 通り) の組み合わせから, 36 通り存在することがわかる. したがって, 576 通りの内 36 個の 1 次式を用いて表すことができるが, 残りの 540 通りのラテン方陣は 1 次式で表せない. 数式のクラスを拡張することによってサイズ 4 の全てのラテン方陣を表すことができるのか, という課題が出てくるが, 本論文ではこれ以上立ち入らない.

また, 上記の 1 次式で表せないラテン方陣を用いた (2,3) しきい値法の 1 例を図 16 に示す. 図 16 のように, 1 次式では表せないラテン方陣を用いた (2,3) しきい値法は存在するので, サイズ 4 の (2,3) しきい値法は位数 4 の有限体上での整合性の取れた 3 個の式から構成できるかは今のところ分からない.

		w_2			
		0	1	2	3
	0	0	1	2	3
	1	1	0	3	2
w_1	2	3	2	1	0
	3	2	3	0	1

		w_3			
		0	1	2	3
	0	0	1	2	3
	1	2	3	0	1
w_2	2	1	0	3	2
	3	3	2	1	0

		w_3			
		0	1	2	3
	0	0	1	2	3
	1	3	2	1	0
w_1	2	2	3	0	1
	3	1	0	3	2

図 16　1 次式で表せないラテン方陣を用いた (2,3) しきい値法の例

7　サイズ 6 の $(2,3)$ しきい値法

4.3 節では有限体の演算を用いることにより, 整合性を満たした 3 個のラテン方陣の作り方の一例を説明した. 一方, 有限体の演算を用いることができるのは有限体が定義できるサイズの場合だけである. つまり, 有限体が定義できないサイズの (2,3) しきい値法は有限体の演算で求めることが不可能である. しかし, 有限体が定義できないサイズのラテン方陣自体は複数存在する. そこで, 有限体が定義できないサイズであっても整合性を満たす 3 個のラテン方陣が存在するのではないかということが文献 [3] で問題提起されている.

本研究では, ラテン方陣の組み合わせをプログラムを用いて調べることにより, 有限体の定義できない 6 をサイズとする (2,3) しきい値法が存在するかを探索した.

5 章のアルゴリズムを用いて探索した結果, サイズ 6 のラテン方陣を用いた (2,3) しきい値法は存在しないことが確認された. 有限体を定義できないその他のサイズに対する探索は今後の課題である.

8　まとめ

確率的な秘密分散法と非確率的な秘密分散法の定義について確認し, ラテン方陣を用いて (2,3) しきい値法が整合性を満たした 3 個のラテン方陣で表現できることを確認した.

このことから, Shamir の多項式補間法の性質上構成することができないサイズ 3 の (2,3) しきい値法でも 3 個のラテン方陣の整合性を確認することでサイズ 3 の (2,3) しきい値法が構成できるのではないかと予想しプログラムを用いてそのような組み合わせを探索した. その結果, サイズ 3 の (2,3) しきい値法は 72 通り存在することが分かった. また, サイズ 3 のラテン方陣は全部で 12 個存在し, それら全ては位数 3 の有限体上の 1 次式で書けるため, サイズ 3 の (2,3) しきい値法は位数 3 の有限体上での整合性を満たす 3 個の 1 次式で構成できることが分かった.

また, 文献 [3] で推測されていた有限体が定義できないサイズの (2,3) しきい値法についても 3 個のラテン方陣の整合性を確認することで有限体が定義できないサイズの (2,3) しきい値法が構成できるのではないかということについても検証した. 有限体の定義できない 6 をサイズとする (2,3) しきい値法をプログラムで探索した結果, サイズ 6 の (2,3) しきい値法は存在しないことが確認された.

今後の課題として, 3 以外のサイズの (2,3) しきい値法における Shamir の多項式補間法では構成できない組み合わせについての検討と有限体が定義できないサイズの (2,3) しきい値法の探索が残っている.

謝辞

この研究を進めるにあたり指導していただいた西新幹彦准教授に感謝する.

参考文献

[1] 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 年.

[2] Adi Shamir, “How to Share a Secret”, Communications of ACM, no22, pp.612-613, Nov. 1979.

[3] 大草健人,「秘密分散法の非確率的な定義について」, 信州大学工学部学士論文, 2015 年.

付録 A　補題の証明

A.1　ヴァンデルモンド行列に逆行列が存在することの証明

ヴァンデルモンド行列とは以下のような行列のことをいう.

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_k \\ x_1^2 & x_2^2 & \dots & x_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_k^{k-1} \end{bmatrix} \tag{11}$$

3.2 節の [X] は (11) 式を転置しただけであるのでヴァンデルモンド行列である. ヴァンデルモンド行列に逆行列が存在することを証明する. 逆行列が存在するための条件はヴァンデルモンド行列の行列式が非零であることを示せばよい. よって, (11) 式の行列式を変形する.

$$\begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_k \\ x_1^2 & x_2^2 & \dots & x_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_k^{k-1} \end{vmatrix} = \begin{vmatrix} 1 & 1 & \dots & 1 \\ 0 & x_2 - x_1 & \dots & x_k - x_1 \\ 0 & x_2^2 - x_1 x_2 & \dots & x_k^2 - x_1 x_k \\ \vdots & \vdots & \ddots & \vdots \\ 0 & x_2^{k-1} - x_1 x_2^{k-2} & \dots & x_k^{k-1} - x_1 x_k^{k-2} \end{vmatrix} \tag{12}$$

(12) 式の変形では, 各行からその 1 つ前の行に x_1 をかけたものを引いている. また, 行列式の性質から次のようになる.

$$= 1 \cdot \begin{vmatrix} x_2 - x_1 & x_3 - x_1 & \dots & x_k - x_1 \\ x_2^2 - x_1 x_2 & x_3^2 - x_1 x_3 & \dots & x_k^2 - x_1 x_k \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-1} - x_1 x_2^{k-2} & x_3^{k-1} - x_1 x_3^{k-2} & \dots & x_k^{k-1} - x_1 x_k^{k-2} \end{vmatrix} \tag{13}$$

ここで, (13) 式を整理すると

$$= \begin{vmatrix} x_2 - x_1 & x_3 - x_1 & \dots & x_k - x_1 \\ x_2(x_2 - x_1) & x_3(x_3 - x_1) & \dots & x_k(x_k - x_1) \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-2}(x_2 - x_1) & x_3^{k-2}(x_3 - x_1) & \dots & x_k^{k-2}(x_k - x_1) \end{vmatrix} \tag{14}$$

$$= (x_2 - x_1)(x_3 - x_1) \cdots (x_k - x_1) \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_2 & x_3 & \dots & x_k \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-2} & x_3^{k-2} & \dots & x_k^{k-2} \end{vmatrix} \tag{15}$$

(15) 式における行列式はヴァンデルモンド行列の形をしているため, 上で行った手順で同様に変形していくと

$$= \begin{matrix} (x_2 - x_1)(x_3 - x_1)\cdots(x_k - x_1) \\ \times(x_3 - x_2)(x_4 - x_2)\cdots(x_k - x_2) \\ \vdots \\ \times(x_k - x_{k-1}) \end{matrix} \tag{16}$$

となり, 全ての項は互いに異なる x 同士の引き算であるため (16) 式は非零であることがわかる. この結果から, ヴァンデルモンド行列の行列式は非零であることから, ヴァンデルモンド行列には逆行列が存在することが証明できる.

A.2 $[X]^{-1}$ の k 行目部分の要素がすべて非零であることの証明

$$[X] = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_k \\ x_1^2 & x_2^2 & \dots & x_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_2^{k-1} & \dots & x_k^{k-1} \end{bmatrix} \tag{17}$$

とする. $[X]^{-1}$ は逆行列の定義より次のように書ける.

$$[X]^{-1} = \frac{1}{detX} \begin{bmatrix} \triangle_{11} & \triangle_{12} & \dots & \triangle_{1k} \\ \triangle_{21} & \triangle_{22} & \dots & \triangle_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ \triangle_{k1} & \triangle_{k2} & \dots & \triangle_{kk} \end{bmatrix} \tag{18}$$

ここで, $\triangle_{ij}$ は $[X]$ の (i,j) 余因子 (x_{ij} の余因子) である. 付録 A.1 より, ヴァンデルモンド行列の行列式は非零なので, $detX \neq 0$ である. よって, (18) 式の右辺の行列の k 行目の要素が全て非零であることを示せばよい.

余因子の定義より,$\triangle_{k1}, \triangle_{k2}, \dots, \triangle_{kk}$ はそれぞれ以下のように書ける.

$$[\triangle_{k1}] = \begin{vmatrix} x_2 & x_3 & \dots & x_k \\ x_2^2 & x_3^2 & \dots & x_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-1} & x_3^{k-1} & \dots & x_k^{k-1} \end{vmatrix} = x_2 x_3 \cdots x_k \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_2 & x_3 & \dots & x_k \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-2} & x_3^{k-2} & \dots & x_k^{k-2} \end{vmatrix} \tag{19}$$

$$[\triangle_{k2}] = \begin{vmatrix} x_1 & x_3 & \dots & x_k \\ x_1^2 & x_3^2 & \dots & x_k^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-1} & x_3^{k-1} & \dots & x_k^{k-1} \end{vmatrix} = x_1 x_3 \cdots x_k \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_3 & \dots & x_k \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{k-2} & x_3^{k-2} & \dots & x_k^{k-2} \end{vmatrix} \tag{20}$$

$$\vdots$$

$$[\triangle_{kk}] = \begin{vmatrix} x_2 & x_3 & \dots & x_k - 1 \\ x_2^2 & x_3^2 & \dots & x_{k-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-1} & x_3^{k-1} & \dots & x_{k-1}^{k-1} \end{vmatrix} = x_2 x_3 \cdots x_{k-1} \begin{vmatrix} 1 & 1 & \dots & 1 \\ x_2 & x_3 & \dots & x_{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_2^{k-2} & x_3^{k-2} & \dots & x_{k-1}^{k-2} \end{vmatrix} \tag{21}$$

従って, $[X]^{-1}$ の k 行目の要素は全てヴァンデルモンド行列の行列式で書けるので, 非零であることが分かる.

A.3 サイズ 3 のラテン方陣

(1)		w_2		
		0	1	2
	0	0	1	2
w_1	1	1	2	0
	2	2	0	1

(1)$w_1 + w_2 = S$

(2)		w_2		
		0	1	2
	0	1	0	2
w_1	1	0	2	1
	2	2	1	0

(2)$w_1 + w_2 + 1 = S$

(3)		w_2		
		0	1	2
	0	2	0	1
w_1	1	0	1	2
	2	1	2	0

(3)$w_1 + w_2 + 2 = S$

(4)		w_2		
		0	1	2
	0	0	1	2
w_1	1	2	0	1
	2	1	2	0

(4)$2w_1 + w_2 = S$

(5)		w_2		
		0	1	2
	0	1	0	2
w_1	1	2	1	0
	2	0	2	1

(5)$2w_1 + w_2 + 1 = S$

(6)		w_2		
		0	1	2
	0	2	0	1
w_1	1	1	2	0
	2	0	1	2

(6)$2w_1 + w_2 + 2 = S$

(7)		w_2		
		0	1	2
	0	0	2	1
w_1	1	1	0	2
	2	2	1	0

(7)$w_1 + 2w_2 = S$

(8)		w_2		
		0	1	2
	0	1	2	0
w_1	1	0	1	2
	2	2	0	1

(8)$w_1 + 2w_2 + 1 = S$

(9)		w_2		
		0	1	2
	0	2	1	0
w_1	1	0	2	1
	2	1	0	2

(9)$w_1 + 2w_2 + 2 = S$

(10)		w_2		
		0	1	2
	0	0	2	1
w_1	1	2	1	0
	2	1	0	2

(10)$2w_1 + 2w_2 = S$

(11)		w_2		
		0	1	2
	0	1	2	0
w_1	1	2	0	1
	2	0	1	2

(11)$2w_1 + 2w_2 + 1 = S$

(12)		w_3		
		0	1	2
	0	2	1	0
w_1	1	1	0	2
	2	0	2	1

(12)$2w_1 + 2w_2 + 2 = S$

図 17　サイズ 3 のラテン方陣

付録 B　ソースコード

B.1　サイズ 6 の (2,3) しきい値法を探索するプログラム

```
#include <stdio.h>

char sq1[36] = {
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
};
char sq2[36] = {
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
};
char sq3[36] = {
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
    0x3f, 0x3f, 0x3f, 0x3f, 0x3f, 0x3f,
};

int
bit_count(unsigned int x0)
{
    unsigned int x1 = (x0 & 0x55555555) + ((x0 & 0xaaaaaaaa) >> 1);
    unsigned int x2 = (x1 & 0x33333333) + ((x1 & 0xcccccccc) >> 2);
    unsigned int x3 = (x2 & 0x0f0f0f0f) + ((x2 & 0xf0f0f0f0) >> 4);
    unsigned int x4 = (x3 & 0x00ff00ff) + ((x3 & 0xff00ff00) >> 8);
    unsigned int x5 = (x4 & 0x0000ffff) + ((x4 & 0xffff0000) >> 16);
    return(x5);
}

void
printlatin(char *sq)
{
    int i;

    for (i = 0; i < 36; i++){
        printf("%3x", sq[i]);
        if (i % 6 == 5){
            printf("\n");
        }
    }
    printf("\n");

    return;
}

void
print3latins(char sq1[], char sq2[], char sq3[])
{
    int i, j;
```

```
    for (i = 0; i < 6; i++){
        for (j = 0; j < 6; j++){
            printf("%3x", sq1[i * 6 + j]);
        }
        printf("  ");
        for (j = 0; j < 6; j++){
            printf("%3x", sq2[i * 6 + j]);
        }
        printf("  ");
        for (j = 0; j < 6; j++){
            printf("%3x", sq3[i * 6 + j]);
        }
        printf("\n");
    }
    printf("\n");
    return;
}

int
marubatsu()
{
    int i, j, k, s, a, b, n, w1, w2, w3;
    int min = 7;
    int i_min = -1;
    int n_latin;

    for (i = 0; i < 36; i++){
        b = bit_count(sq2[i]);
        if (b != 1) continue;
        k = (i / 6) * 6;
        for (j = 0; j < 6; j++){
            if (k + j == i) continue;
            sq2[k + j] &= ~sq2[i];
        }
        k = i % 6;
        for (j = 0; j < 6; j++){
            if (k + j * 6 == i) continue;
            sq2[k + j * 6] &= ~sq2[i];
        }
        w2 = i / 6;
        w3 = i % 6;
        for (w1 = 0; w1 < 6; w1++){
            if (sq1[w1 * 6 + w2] == sq2[i]){
                sq3[w1 * 6 + w3] = sq2[i];
                k = w1 * 6;
                for (j = 0; j < 6; j++){
                    if (k + j == w1 * 6 + w3) continue;
                    sq3[k + j] &= ~sq2[i];
                }
                for (k = 0; k < 6; k++){
                    if (k * 6 + w3 == i) continue;
                    sq2[k * 6 + w3] &= ~sq1[w1 * 6 + k];
                }
            }
            else {
                sq3[w1 * 6 + w3] &= ~sq2[i];
            }
        }
    }

    for (i = 0; i < 36; i++){
        int b = bit_count(sq3[i]);
        if (b == 0) break;
        b = bit_count(sq2[i]);
        if (b == 0) break;
        if (b < 2 || b >= min) continue;
        min = b;
        i_min = i;
```

```
    }
    if (i < 36){
        return(0);
    }
    if (min == 7){
        print3latins(sq1, sq2, sq3);
        return(1);
    }

    n_latin = 0;
    for (i = 0; i < 6; i++){
        if (sq2[i_min] & (1 << i)){
            char sq2_dup[36], sq3_dup[36];
            for (j = 0; j < 36; j++){
                sq2_dup[j] = sq2[j];
                sq3_dup[j] = sq3[j];
            }
            sq2[i_min] = 1 << i;
            n_latin += marubatsu();
            for (j = 0; j < 36; j++){
                sq2[j] = sq2_dup[j];
                sq3[j] = sq3_dup[j];
            }
        }
    }
    return(n_latin);
}

int
deduce(char *sq)
{
    int app;
    int i, j, s;

    app = 0;
    for (i = 0; i < 36; i++){
        if (bit_count(sq[i]) != 1) continue;
        s = (i / 6) * 6;
        for (j = s; j < s + 6; j++){
            if (j == i) continue;
            if (sq[j] & sq[i]){
                sq[j] &= ~sq[i];
                app++;
            }
        }
        s = i % 6;
        for (j = s; j < s + 36; j += 6){
            if (j == i) continue;
            if (sq[j] & sq[i]){
                sq[j] &= ~sq[i];
                app++;
            }
        }
    }
    return(app);
}

int
latin()
{
    int i, idx, n, num;
    int bc;

    while (deduce(sq1) > 0){
        ;
    }

    n = 7;
    for (i = 0; i < 36; i++){
```

```
        if (sq1[i] == 0){
            return(0);
        }
        bc = bit_count(sq1[i]);
        if (bc > 1 && bc < n){
            n = bc;
            idx = i;
        }
    }
    if (n == 7){
        int num;
                printlatin(sq1);
        num = marubatsu();
        return(num);
        //      return(1);
    }

    num = 0;
    for (i = 0; i < 6; i++){
        char sq1_dup[36];
        int j;
        if ((sq1[idx] & (1 << i)) == 0) continue;
        for (j = 0; j < 36; j++){
            sq1_dup[j] = sq1[j];
        }
        sq1[idx] = 1 << i;
        num += latin();
        for (j = 0; j < 36; j++){
            sq1[j] = sq1_dup[j];
        }
    }
    return(num);
}

int
main()
{
    int num;

    num = latin();
    printf("num: %d\n", num);
    return(0);
}
```