平成 12 年度

学士学位論文

秘密分散法に基づく ファイル保管コマンドの実装

Implementation of an archive command using Secret Sharing Scheme

1010440　　舟橋釈仁

指導教員　　菊池 豊

2001 年 2 月 5 日

高知工科大学 情報システム工学科

要 旨

秘密分散法に基づく
ファイル保管コマンドの実装

舟橋釈仁

データを分散暗号化する理論に秘密分散法 (SSS: Secret Sharing Scheme) がある．SSS は RAID 同様にデータを分散保管する方法で, RAID にはない高い対障害性と秘匿性を持っている．本研究の目的は SSS を実装し対障害性と秘匿性を与えるシステムを作成することである．第一の目標として SSS の一つである Shamir の (k, n) 閾値 SSS に基づく `ssar` コマンドを UNIX 上に実装した．性能測定を行ったところ, 用途によっては十分実用になることが判明した．現在の実装では暗号化の単位であるブロック長は 2Byte が限界であり、これを大きくするためには原始根を求める計算を高速に行う必要があることが判明した．

キーワード　秘密分散法, 対障害性, 秘匿性, RAID5

Abstract

Implementation of an archive command using Secret Sharing Scheme

FUNAHASHI Sekiji

Secret Sharing Scheme, or SSS, is a scheme of cryptology, that distributes of data . SSS is scheme of data storage same as RAID, which have against obstacle property and secrecy property The purpose of this research is to creating system which give data against obstacle property and secrecy property First target is to implement an encrypt/decrypt command which is `ssar` on UNIX based on SSS . It turns out that the command is useful in some applications because of a performance result of the command. Block size limits of crypt unit is 2 Byte on this implement. In order to block size enlarge It need to high transaction of seeking primitive root.

key words SSS , opposite obstacle nature, Secrecy nature, RAID5, Shamir, k, nthreshold SSS, sssar

目次

第 1 章　はじめに **1**

第 2 章　秘密分散法 **3**

2.1　秘密分散法 3

2.2　(k, n) 閾値 SSS 4

2.3　(k, d, n)ramp 型 SSS 5

2.4　分散情報のサイズの特性 5

2.5　(k, n) 閾値 SSS のアルゴリズム 6

2.6　乱数 7

第 3 章　SSS と RAID5 **8**

3.1　SSS の特徴 8

3.2　RAID5 の特徴 9

3.3　SSS と RAID の比較 9

第 4 章　SSS の実装 **12**

4.1　仕様 12

4.1.1　基本方針 12

4.1.2　仕様 13

4.1.3　コマンド形式 13

4.1.4　実行例 14

4.2　設計 16

4.2.1　シェアファイルの形式 16

4.2.2　暗号化 17

4.2.3　復号化 18

4.3 実装 18
4.3.1 ブロック長 18
4.3.2 多倍長整数型 19
4.3.3 素数計算 19
4.3.4 原始根計算 19

第 5 章 性能測定 **20**
5.1 素数と原始根を求める時間のブロック長の違いによる比較 20
5.2 ファイルサイズによる処理速度の比較 21
5.3 ブロック長の違いによるシェアファイルのサイズ比較 22

第 6 章 考察 **23**
6.1 素数と原始根を求める計算量 23
6.2 処理速度 24
6.3 シェアファイルのサイズ 25

第 7 章 まとめ **26**
7.1 結果 26
7.2 2 の拡大体における SSS 26
7.3 ネットワーク転送機能の付加 27
7.4 ファイルシステムへの応用 27

謝辞 **28**

参考文献 **29**

付録 A 視覚型秘密分散法 **31**

第 1 章

はじめに

さまざまな分野に IT 技術が用いられるようになり, 情報の効率的な管理や高速処理などが可能になった. しかし, 同時に IT 技術は特有の危険も内包しており特に情報の保管に際しては十分な対策を行わなければならず, 情報の保管に対する安全性はますます重要になってきている.

コンピュータ上のデータを保管する際はさまざまな要素を考える必要がある. その中でデータの損失を防ぐ対障害性は重要である. 例えば, コンピュータ上のデータは最終的にハードディスクなどの物理媒体に保管される. しかし, ハードディスクのような物理媒体には物理的な経年変化が伴い, データ損失の可能性が生じる. このようなハードウェアに依存したデータ損失の危険を減らすためには保管場所の分散などの対策をする必要がある. テープにデータの複製を作成することなどが一般的に行われている.

しかし, 対障害性を得るためにデータの複製を作成すると複製データが不正利用される危険があるため, 複製データに対して秘匿性を与える必要がある. 複製データを暗号化して保管することなどは一般的に行われている.

分野によっては, たとえコストが若干余計にかかろうともデータに十分な秘匿性や対災害性を持たせたいという要望が出て来た. 例えば、住民基本台帳等を管理する地方自治体や保健・医療・福祉等の重要な個人情報を扱う組織である. 災害が起ったとすると, その地方にあるコンピュータや複製データの入ったテープなどすべてが壊れる可能性がある. そのためデータを遠隔地に分散して保管するなどの災害対策をすることが望まれる. そして, こうした場合には同時に分散したデータを不正利用されないように高いセキュリティの仕組みも必要である.

データに秘匿性と対障害性を与える暗号理論に秘密分散法 (SSS: Secret Sharing Scheme) がある [10][2]. SSS では情報に対して秘匿性を得ることを目的とした暗号化と対障害性を得ることを目的とした分散を同時に行うことができる.

SSS の時間計算量や空間計算量は大きく, これまでは実用性が低いと考えられていた. しかし, 近年プロセッサの処理性能の向上とハードディスクの低価格化により SSS を使用したデータ保管手法が現実味を帯びて来ている.

本研究の目的は SSS を基にしたデータ分散手法を実装し, 保管データに対障害性と秘匿性を与えるシステムを作成することである. 最終的には SSS に基づいたファイルシステムの実装を予定している. 今回はその第 1 段階として SSS に基づく ssar コマンドを作成しその基本特性を調べた.

本論文では最初に SSS 理論について説明を行い, SSS と分散型データ保管の手法である RAID との比較をした. 次に SSS の実装である ssar コマンドについて仕様, 設計, 実装, 性能測定を行った結果を説明する. そして, 性能測定の結果より見つかったいくつかの課題について考察を行う. 最後にこれらの課題の解決について今後の予定を述べる. また, 付録において SSS 理論の一つである視覚型 SSS について説明する.

第 2 章

秘密分散法

秘密分散法 (SSS: Secret Sharing Scheme) の理論は Shamir[10] と Blakey[2] により独立に (k, n) 閾値法が提案されたことにより始まり, 現在も構成法や特性に関する多くの研究が行われている.

本章では, まず SSS の特性について説明を行う. 次に SSS における分散情報のサイズについて詳しく説明する. また, 本研究で使用する Shamir により提案された (k, n) 閾値 SSS のアルゴリズムを説明する.

説明には一部「現代暗号」[18] と「秘密分散共有法とその応用」[17] から引用を行った.

2.1 秘密分散法

秘密分散法 (SSS) とは, 秘密情報からいくつかの分散情報を作成し, 特定の分散情報を集めると元の秘密情報を得ることができる. しかし, それ以外の分散情報からは元の秘密情報を得ることができないという理論である.

SSS では, 秘密情報を得るこができる分散情報の集合をアクセス集合と呼び, アクセス集合において一つでも分散情報がなくなると秘密情報が得ることができなるなるような集合を極小アクセス集合と呼ぶ. また, アクセス集合以外の集合, つまり秘密情報を得ることができない分散情報の集合を非アクセス集合と呼ぶ.

SSS は非アクセス集合の特性により大きく二つに分かれる. 非アクセス集合において部分的な情報を得ることができる不完全秘密分散法と非アクセス集合からは部分的な情報を全く得ることができない方式の完全秘密分散法である.

例えば, Shamir により提案された (k,n) 閾値秘密分散法は完全秘密分散法に分類される. そして, ramp 型秘秘密分散法 [14, 12] は不完全秘密分散法に分類される.

SSS の理論にはさまざまな種類があり, 構成方について幾何学的な手法を用いるもの [11] やベクトル空間を利用したもの [6], マトロイド理論を利用したもの [5, 16], グラフ理論を使用したもの [4], ブール関数を利用したもの [1] などがある. また, 特殊な SSS として視覚型 SSS[8, 9] や分散データに検証機能を持たせた検証可 SSS[7] などがある [18].

2.2 (k,n) 閾値 SSS

完全秘密分散法に分類される方式である (k,n) 閾値 SSS を説明する.

(k,n) 閾値 SSS は秘密情報を n 個の分散情報に符号化し, 任意の分散情報を k 個集めると元の秘密情報を復元できる. しかし, 分散情報が $k-1$ 個以下であれば元の秘密情報を復元できないという特徴を持っている. また, $k-1$ の分散情報から部分情報は復元できない.

このとき分散情報 $w_i(i = 1,2,\ldots,n)$ は秘密情報 S と素数 p, そして乱数 $r_j(j = 1,2,\ldots,k-1)$ を用いて $k-1$ 次の多項式 2.1 から以下のように表せる. また, この時 x_i は同じ値を含まない.

$$w_i = f(x_i) \quad (i = 1,2,\ldots,n)$$

$$f(x) = S + r_1 x + r_2 x^2 + \ldots + r_{k-1} x^{k-1} \pmod{p} \tag{2.1}$$

これらの式を用いて (k,n) 閾値 SSS の原理を説明すると次のようになる.

式 2.1 において $y = f(x)$ とすると秘密情報 S は y 軸切片を示している. 式 2.1 は $k-1$ 次の多項式なので, 分散情報 $w_i(i = 1,2,\ldots,n)$ が k 個集まると k 個の座標点から多項式 $y = f(x)$ が一意に求まる. しかし, 分散情報を $k-1$ 個集めただけでは $k-1$ 個の座標点しか分からず多項式が一意に定まらないので, 全ての y 切片を通る可能性がある. そのため秘密情報 S が分からない.

2.3 (k, d, n)ramp 型 SSS

不完全秘密分散法には (k, d, n) ランプ型 SSS がある．(k, d, n) ランプ型 SSS は n 個の分散情報のうち k 個以上集めると秘密情報を得ることができ, $k - d$ 以下集めただけでは秘密情報が全く分からない．しかし, 分散情報を $k - d$ より多く k 個未満集めると秘密情報の部分的な情報が分かる．

秘密情報 S を $S = S_1 + \ldots + S_l$ とし, $k > d$ が成り立っているとする．このとき分散情報 $w_i (i = 1, 2, \ldots, n)$ は秘密情報の部分的な値 $S_l (l = 0, 1, \ldots, d)$ と素数 p, そして乱数 $r_j (j = 1, 2, \ldots, k - d)$ と $k - 1$ 次の多項式 2.2 から以下の ように表せる．また, この時 x_i は同じ値を含まない．

$$w_i = f(x_i) \quad (i = 1, 2, \ldots, n)$$

$$f(x) = S_0 + S_1 x + S_2 x^2 + \ldots + S_d x^d + r_1 x^{d+1} + \ldots + r_{k-d} x^{k-1} \pmod{p} \tag{2.2}$$

これらの式を用いて (k, d, n) ランプ型 SSS の原理を説明すると次のようになる．式 2.2 は $k - 1$ 次の多項式なので, 分散情報 $w_i (i = 1, 2, \ldots, n)$ が k 個集まると k 個の座標点から多項式 $y = f(x)$ が一意に求まり, 秘密情報の部分的な値 S_l がすべて求まるので秘密情報 S の値が求まる．また, 分散情報を $k - d$ 個以上集めると集めると $S_0 \ldots S_{k-d}$ の値について関係式が得られるため秘密情報 S の部分的な情報を得ることができる．

2.4 分散情報のサイズの特性

完全秘密分散法において分散情報のサイズは定理 2.3 を満たす [17].

$$\text{分散情報のサイズ} \geq \text{秘密情報のサイズ} \tag{2.3}$$

定理 2.3 より完全秘密分散法において最も効率よく分散情報を作成した場合, そのサイズは秘密情報と同じサイズになる．また,「秘密分散共有法とその応用」[17] によれば不完全秘密分散法において分散情報のサイズは定理 2.4 を満たす.

$$\text{分散情報のサイズ} \geq \text{秘密情報のサイズ}/d \tag{2.4}$$

定理 2.4 より不完全秘密分散法を使えば分散情報のサイズを秘密情報のサイズより小さくすることが可能である. しかし, 代わりに分散情報の安全性を低くしてしまう.

このように秘密分散法において分散情報におけるサイズと安全性はトレードオフの関係にある. そのため, 分散情報の安全性を低くすることなしに分散情報のサイズを秘密情報のサイズより小さくすることは原理的に不可能である. 安全性を犠牲にしない方式の中で Shamir の (k,n) 閾値法は分散情報のサイズが秘密情報のサイズと同じになるのでサイズの面からみて最も効率的な方式である.

本研究では, 分散情報の安全性を低くすることはデータ保管に適さないと考え, 完全秘密分散法を選択し, 完全秘密分散法の理論としてサイズの面で最も効率的な Shamir の (k,n) 閾値法を採用した.

2.5 (k,n) 閾値 SSS のアルゴリズム

本節では Shamir の (k,n) 閾値 SSS のアルゴリズムについて説明する. Shamir の (k,n) 閾値 SSS では暗号化, 復号化ともに演算は素数 p の有限体上で行われる. 最初に秘密情報 S より大きな素数 p とその原始根 α を求める. 次に暗号化では式 2.5 のような行列 G をつくる. この式の 2 列目である α^1〜α^n は分散情報を識別するために使われ, ID と呼ばれる. そして, 式 2.5 を使い式 2.6 より分散情報を計算する. また, 式 2.6 において $r_j(j=1,2,\ldots,k-1)$ は $GF(p)$ 上の乱数とする.

$$G=\begin{bmatrix} 1 & \alpha^1 & \alpha^2 & \cdots & \alpha^{k-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{k-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^n & (\alpha^n)^2 & \cdots & (\alpha^n)^{k-1} \end{bmatrix} \tag{2.5}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \equiv \begin{bmatrix} 1 & \alpha^1 & \alpha^2 & \cdots & \alpha^{k-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{k-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^n & (\alpha^n)^2 & \cdots & (\alpha^n)^{k-1} \end{bmatrix} \begin{bmatrix} S \\ r_1 \\ \vdots \\ r_{k-1} \end{bmatrix} \pmod p \tag{2.6}$$

復号化では分散情報 w_i を閾値の数だけ集め, 式 2.5 の逆行列を含んだ式 2.7 の行列計算を行い秘密情報 S を求める. 式 (2.7) において行の数は閾値の数になり正方行列になり, j_1〜j_k は 1〜k までの数である.

$$\begin{bmatrix} S \\ r_{j_1} \\ \vdots \\ r_{j_{k-1}} \end{bmatrix} \equiv \begin{bmatrix} 1 & \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^{k-1} \\ 1 & \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^{k-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{j_k} & (\alpha^{j_k})^2 & \cdots & (\alpha^{j_k})^{k-1} \end{bmatrix}^{-1} \begin{bmatrix} w_{j_1} \\ w_{j_2} \\ \vdots \\ w_{j_k} \end{bmatrix} \pmod{p} \qquad (2.7)$$

2.6 乱数

視覚型 SSS では分散画像のデータを人間の目で確認できる. 例えば質の悪い乱数を使うと分散画像に対して一定のパターンを見ることができる. そのため, 最悪の場合分散情報から全体の情報を予測できる.

このように乱数を使う場合はその質が重要である. SSS は基本的にすべての理論において乱数を使うので, 良質な乱数が必要である.

第 3 章

SSS と RAID5

データを冗長にすることにより対障害性を持たせた分散型データ保管法には SSS の他に RAID[15] がある.

これらの技術はデータをいくつかのハードディスに分散して保管する. 分散されたデータは冗長になりハードディスクが壊れてもデータを失うことが少なくなり, 単一のハードディスクに保管するより安全性が高い.

本章では SSS と RAID5 について特徴を説明した後, 両者の比較を行う. また, 冗長性を得るために暗号化したデータを複製して分散データを作成するような方法についても比較する.

3.1 SSS の特徴

分散データの数を n とすると SSS では分散データをつくる時に $0 < k \leq n$ の範囲で任意に閾値 k を設定できるため, データの保管に適した安全性を柔軟に設定することができる.

また, SSS は既存の暗号化技術と併用することが容易で, 高いレベルでの秘匿性を得る事ができる. 例えば, 秘密情報から $(2, 2)$ 閾値 SSS 使用して分散データを作成する. 次にそれぞれの分散データに別々の暗号理論を使用してさらに暗号化を行う. この場合, 秘密情報が復元する条件は, それぞれの分散データにかけられた暗号が解読され, なおかつ 2 つの分散データがそろったときに限られる.

SSS の欠点は分散データの合計のサイズと処理速度である. SSS で最も効率的に分散データが作成されたとしても, そのサイズの合計は元のデータに比べて最低 n 倍大きくなる. ま

た, SSS は暗号化するときに式 2.6 を計算しなければならない. しかし, 大きな数の原始根を求める計算は高速化が難しいため処理速度が遅くなる.

3.2 RAID5 の特徴

ハードディスクはコンピュータ構成する部品の中で機械的な動作をする部分である. そのため, CPU やメモリに比べ動作速度が遅く, 信頼性が低いといわれている. このような問題を安価なハードディスクを使用してディスクアレイを行うことにより解決する技術が RAID である. RAID は複数のハードディスクを使ってアクセスを分散させることにより, 高速化と高信頼性を得る技術である.

RAID には構成により 0〜5 の RAID レベルがある. この中でよく使われているのは RAID 0 (ストライピング), 1 (ミラーリング), 5 (分散データガーディング) である. 分散保管の構成法としては RAID3, RAID4, RAID5 がある. そのなかで RAID5 はブロック単位でストライピングを行い, しかもパリティチェックを複数のディスクに書き込むため RAID3 や RAID4 より高速に動作する分散保管法である. つまり RAID3, RAID4 の弱点を補うようになっている. 以降 RAID5 を単に RAID と呼ぶ.

RAID ではパリティによるデータ復元を行うため, 2 台以上のハードディスクがクラッシュするとデータの復元ができない. しかし, 分散データの合計は元のデータに比べてパリティのサイズ大きくなるだけである.

また, 分散データに対して暗号化が行われないので, それぞれの分散データの中身を見ると, 元のデータの部分的な内容が分かる.

3.3 SSS と RAID の比較

SSS と RAID の比較を表 3.1 に示す. 表で x は元のデータのサイズを示している. また, α と γ は暗号化にともなうデータの増加分であり, β は RAID のパリティによる増加分を示す. また, 元のデータを単純に暗号化し, そのデータを複製して分散データとした場合も表に

併記した.

	SSS	RAID	暗号化複製
分散データ合計サイズ	$nx + \alpha$	$x + \beta$	$nx + \gamma$
冗長分散データ数	$n - k$	1	$n - 1$
部分情報安全性	高	低	中
全体情報安全性	高	高	中
処理速度	遅い	速い	中

表 3.1 SSS と RAID と素朴な暗号化複製との比較

まず, 分散データ合計サイズについて比較する. SSS では合計サイズは最低でも元のデータの n 倍になるのに対し, RAID ではパリティチェックの分増えるだけであるため RAID の方が優れている.

次に冗長分散データ数を比較する. 冗長分散データ数とは元のデータを戻すことが可能であるという条件の下で分散データを失う事ができる数である. SSS は $n - k$ と非常に柔軟に設定できる. しかし, RAID は, パリティチェックのため冗長性が低く 2 つ以上の分散データが損失する と元のデータに復元しない. よって SSS の方が優れている.

次に部分情報安全性と全体情報安全性について比較する. 部分情報安全性というのは, 分散データをハッキングされた時, 部分的な情報を解読されたかどうかで安全性を比較した値である. また, 全体情報安全性とは分散データをハッキングされた時, 全体の情報が解読されたかどうかで安全性を比較した値である. SSS は分散データの安全性が高い. SSS では分散と同時に暗号化が行われる. この暗号化により分散データが閾値個集まらなければ全体的な情報は解読されないし, 分散データをみても部分的な情報も解読されない. 一方, RAID には暗号化機能がついていない. 分散データをみると, 元のデータの部分的な情報が解読される. しかし, 分散データは部分的な情報しか持っていないため全体の情報を予測する事は難しい. 暗号化複製においては分散データが全体の情報を持つため RAID と SSS の中間程度と考える.

最後に処理速度について比較する．SSS では処理速度は非常に遅い．しかし, RAID では複数のハードディスクを使用するので単一のハードディスクによる処理に比べ速くなる．また, 暗号化複製はその中間程度になる．

以上の理由により, SSS は処理速度が遅いことや分散データの合計サイズが大きいといった欠点を持つものの対障害性や安全性の面において RAID にはない優れた特徴をもつデータ分散手法であるといえる．

第 4 章

SSS の実装

Shamir の (k, n) 閾値 SSS を与えられたソースファイルに対して適用する `ssar` (Secret Sharing Archiver) コマンドを作成した. 本章では `ssar` コマンドの仕様と設計, 実装について述べる. 本論文では, 以降 `ssar` を実行した結果得られる分散データのことをシェアデータと呼び, シェアデータを格納したファイルをシェアファイルと呼ぶ.

4.1 仕様

本節では, コマンドを実装する際に立てた方針と仕様について説明し, コマンドの実行例を示す.

4.1.1 基本方針

今回の実装は SSS アルゴリズムを使用した初めての実装であり, 基本的な特性を調べる事を目的とした.

SSS の特徴はシェアデータが閾値個存在すれば秘密情報を復元でき, シェアデータが閾値個存在しなければ復元できない性質にある. 今回の実装では SSS の特徴を失わないようにシェアファイルは暗号化と復号化におけるすべての情報を持つようにした.

使用するアルゴリズムは Shamir の (k, n) 閾値法である. 第 2 章で説明を行ったように Shamir の (k, n) 閾値法は理論上は出力するシェアデータサイズが秘密情報と同じになる. これは, サイズの面からみて最も効率のよい方式である.

SSS アルゴリズムでは素数計算, 原始根計算, 行列計算が主な処理である. 秘密情報のサ

イズと深い関係にある素数計算と原始根計算は非常に時間がかかる事が分かっている．その計算時間は SSS アルゴリズムにおいて大きな割合を占める．しかし, 今回は, 素朴に実装して処理にかかる時間の特性をみた．また, 行列計算も特別な高速化をしないで, 素朴に実装した．

また, 秘密情報のサイズや閾値, シェアファイルの数などのパラメータはコマンドオプションで指定できるようにした．

4.1.2 仕様

SSS アルゴリズムは暗号化と復号化の 2 つに大きくわかれているので, それぞれの動作を以下のように定めた．

- 暗号化部分

  ソースファイルが入力されたらパラメータに従い SSS アルゴリズムを適用し暗号化を行いシェアファイルを出力する．
- 復号化部分

  シェアファイルが入力されると必要な情報をシェアファイルから読み出し復号化を行った後, ソースファイルを復元する．

仕様のイメージを図 4.1 に示す．

4.1.3 コマンド形式

コマンドを実行するとソースファイルからオプションにより設定した数のシェアファイルを出力する．シェアファイルの名前はオプションで指定された basename に 0〜シェアファイル数 −1 までの番号を組み合わせて作成する．以下に暗号化の構文を示しオプションについて説明する．

```
ssar -c -s number -t number -b number -f basename sourcefilename
```

-c 暗号化の指示

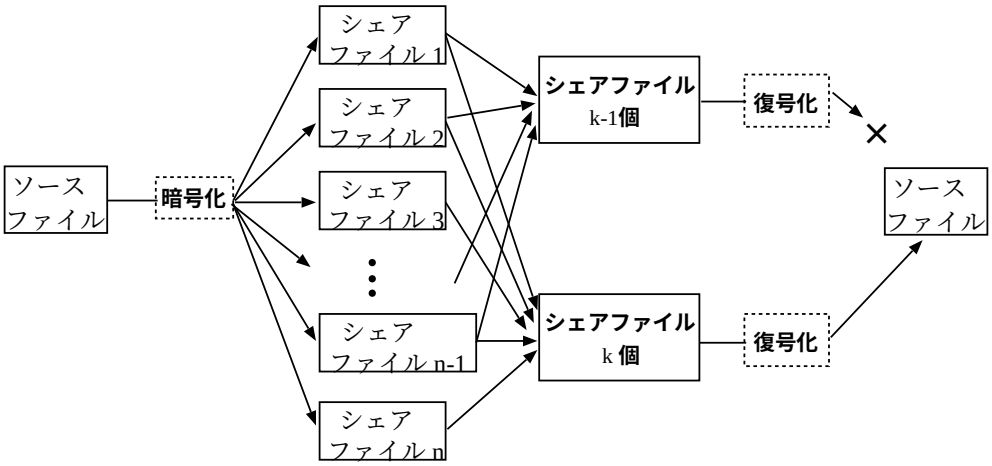

図 4.1 SSS による暗号化と復号化

-s シェアファイルの数

-t 閾値の値

-f シェアファイルにつける basename

-b 暗号化を行うブロック長の指定

次に復号化の時はコマンドを実行すると閾値個のシェアファイルから元のソースデータをオプションで指定されたファイル名で出力する. 以下に復号化の構文を示しオプションについて説明する.

```
ssar -x -f decodefilename sharefilename1 sharefilename2 ...
```

-x 復号化の指示

-f 復元するファイルの名前

4.1.4 実行例

ssar コマンドを実行した場合を暗号化と復号化について示す. 最初に暗号化の時の実行例を示す. 以下ではまず, ls コマンドによりファイル afo があるとする. 次に ssar コマンドを-c オプションにより暗号化モードで実行する. ここで-s オプションでシェアファイル数を 5, -t で閾値を 3, -b でブロック長 2Byte, -f でシェアファイルのベースの名前を bfo

として, 最後に afo をソースファイルとしている. そうするとシェアファイル bfo[01234] が作成される. ls コマンドにより確認する.

```
> ls
afo
> ssar -c -s 5 -t 3 -b 2 -f afo bfo
> ls
afo bfo0 bfo1 bfo2 bfo3 bfo4
```

次に復号化の例を示す. ssar コマンドを-x オプションつけ復号化モードで実行する. -f で復元するファイル名である cfo を指定する. 閾値の数のシェアファイルを任意に選ぶが例では bfo1, bfo3, bfo4 を使用した. cfo というファイルが作成されるので ls コマンドで確認している. 次にこの cfo というファイルとソースファイルである afo を cmp コマンドにより比べると全く同じである事が分かり, 元のファイルが復元された事が分かる.

```
> ssar -x -f cfo bfo1 bfo3 bfo4
> ls
afo bfo0 bfo1 bfo2 bfo3 bfo4 cfo
> cmp afo cfo
>
```

以下ではシェアファイルが閾値より少なかった例を示す. シェアファイル bfo[01234] の中から 2 個選んで入力しても閾値より少ないので, usage メッセージをだして終了する.

```
> ls
> afo bfo0 bfo1 bfo2 bfo3 bfo4
> ssar -x -f cfo bfo1 bfo3
 usage:ssar {-c | -x} [-s number ] [-t number ]
                    [-b number ] -f filename  inputfile ...
>
```

また, シェアファイルが閾値より多く入力された場合, 先頭から閾値個のシェアファイルだけを使用して動作する.

4.2 設計

ssar コマンドの機能を暗号化と復号化に分けて設計した. また, 実装ではシェアファイルにすべての情報が保存される. そこで, 保存されるデータの形式も設計した.

4.2.1 シェアファイルの形式

シェアファイルは SSS アルゴリズムにおいて作成されるシェアデータの他に *ID*, 暗号化に用いた素数, シェアファイルの数, 閾値, basename などの情報を持つ. シェアファイルの保存形式を表 4.1 のようにした. この保存形式では素数と ID とシェアデータ以外の情報は ascii 形式で保存されるためページャなどで確認することができる.

HEAD 部分にはシェアファイルを復元するための付加的な情報を格納する. 1 行目に HEAD タグをつけ, HEAD 情報であることを示す. そして, シェアファイルの数, 閾値, basename を格納する.

BODY 部分は復号に必要なデータの集合である. 最初に BODY タグをつけ, シェアデータを格納する. BODY 部分の一番最初に素数, 2 番目に *ID*, 3 番目以降にシェアデータを格納する.

	格納する情報
HEAD 情報	[HEAD] タグ シェアファイルの数 閾値 basename
BODY 情報	[BODY] タグ 素数 *ID* シェアデータ ⋮ シェアデータ

表 4.1 シェアファイルの形式

4.2.2 暗号化

暗号化は次の流れで処理される.

1 オプション解析

2 HEAD 情報をシェアファイルに書き込む

3 ブロック長より大きな素数 p を計算し, シェアファイルに書き込む.

4 素数 p の原始根を計算しシェアファイルに書き込む.

5 乱数 r_x を関数により求めるか/dev/random から求める.

6 式 2.5 の計算を行い *ID* を求め, シェアファイルに書き込む.

ここで原始根 $\alpha, \alpha^2 \cdots \alpha^n$ がそれぞれ $ID_1 \cdots ID_n$ となる.

7 ソースファイルからブロック長単位で秘密情報データ S を読み込む.

8 式 2.6 の行列計算によりシェアデータ w_i を求める SSS 暗号計算を行う.

9 シェアファイルにシェアデータ w_i をバイナリで書き出す.

10 ソースファイルの終りまで 7〜10 を繰り返す.

4.2.3 復号化

復号化は次の流れで処理される.

1 オプション解析

2 HEAD 情報を読み込む.

3 シェアファイルから素数と *ID* を読み込む.

4 式 2.5 の逆行列を求める.

5 シェアファイルからシェアデータ w_i を読み込む.

6 SSS 復号計算を行う.

式 2.7 の計算を行う. ここで $\alpha^{j_1}, \ldots, \alpha^{j_k}$ はシェアファイルの *ID* である.

7 ソースファイルとして秘密情報データ S を書き込む.

8 シェアファイルの終りまで 5〜7 を繰り返す.

4.3 実装

ssar コマンドにおいて, 主要な計算部分は素数計算と原始根計算と行列計算である. 本節ではこれらの計算の基本単位であるブロックと素数計算, 原始根計算について詳しく説明を行う.

4.3.1 ブロック長

プログラムではソースファイルから秘密情報データ S を一定のブロック長で読み込む. このブロックというのは暗号化の単位であり, ブロック長が長いと分散データを推測するための計算量が増えるため安全といえる. プログラムではオプションによりブロック長を 1〜512Byte (UNIX における標準的な入出力データサイズ) を選択できるにした. ブロック長の指定は b オプションで行われ, 何も指定されないときはデフォルトの 1Byte になる.

4.3.2 多倍長整数型

ブロック長が 512Byte のとき 10 進整数に直すと 0〜$2^{4096}-1$ の整数になる．このような大きな整数を扱うために GMP3.0 (The GNU Multiple Precision Arithmetic Library) を使用した．GMP とは多倍長の整数, 多倍長の有理数, 多倍長の浮動小数を扱うためのライブラリである．入出力, 乱数の取得, 各種計算関数などを持っている．GMP では多倍長整数を mpz 型としてもち, これを int 型の列として実現している．

今回のプログラムでは, 秘密情報 S やシェアデータ w_i などを mpz 型にしている．

4.3.3 素数計算

素数を求める計算には確率的素数判定法と確定的素数判定法がある．確定的素数判定法は処理が確率的素数判定法より処理が遅い一方で確実に素数を判定する．反対に確率的素数判定法では素数でない可能性が残るものの確定的素数判定法より処理が高速である．

今回は処理速度を優先して確率的素数判定法を使用した．この場合素数でない可能性が残る．しかし, 素数でなければ原始根計算の部分で判別されるので, もう一度素数計算をやり直すように実装した．

確率的素数判定法の実装は GMP のライブラリの中の素数を求める $mpz\_probab\_prime\_p()$ 関数を使用した．この関数では Miller-Rabins 法が使用されている．

4.3.4 原始根計算

原始根とは $p-1$ 乗したときの値だけが 1 となるような数である．プログラムでは最初に 2 の 1 乗から 2 の $p-1$ 乗までの計算を行い, 計算結果をハッシュ構造に保存する．そして, すべての計算した値を比べ, 同じ値が 2 個以上あれば原始根でないと判断する．原始根でない場合は 3 の 1 乗から 2 の $p-1$ 乗までの計算を行い, 同じ値がないか調べる．もし, 同じ値が 2 個以上あれば, さらに 4 を試す．これを原始根がみつかるか, $p-1$ まで行う．$p-1$ まで行って見付からなければ p が素数でないと判定できるので改めて素数を求め直す．

第 5 章

性能測定

本章では本研究で実装した ssar コマンドの性能測定を行った結果について説明する.

Shamir の (k, n) 閾値 SSS を実装したときに問題となるのは処理速度とシェアサイズであるため, 暗号化と復号化の処理速度とシェアサイズについて詳しく調べた. また, 素数と原始根を求める時間はブロックの値が大きくなると急激に大きくなることが分かっているため, どれくらい大きくなるか計算時間を調べた. また, 実行時間は計りたい部分の最初と終りに $clock()$ 関数を用いて実行時からの時間経過を取得し, その差分を計算することにより得た.

測定には CPU が Pentium3 600MHz, メモリ 128MB のマシンを使用し, OS には FreeBSD4.1.1 を使用した. また, コンパイラは gcc version 2.95.2 を使用した.

5.1 素数と原始根を求める時間のブロック長の違いによる比較

ブロック長を大きくしていった時の素数と原始根を求める時間を計測した. 具体的にはブロック長が 1Byte と 2Byte の時の処理時間を計測した. 結果を表 5.1 に示す.

ブロック長 (Byte)	素数と原始根を求める時間 (s)
1	0
2	39

表 5.1 ブロック長の違いによる処理時間の変化

ブロック長が 3Byte の場合にはプロセスが扱えるメモリ上限を越えてしまったため計測

することができなかった. よってブロック長が 3Byte 以上の時の素数と原始根を求める時間を計測する事ができなかった.

5.2　ファイルサイズによる処理速度の比較

ここでは ssar コマンドを実行したときの暗号化と復号化にかかる処理速度を計測した. 具体的にはブロック長を一定にして 1MB〜100MB のファイルを入力し, 入力ファイルサイズの違いによる暗号化と復号化にかかる処理時間を計測した. ブロック長が 1Byte の時と 2Byte の時の結果を図 5.2 に, 3Byte 以上の時は 5.1 節であげた理由により計測できなかった.

ブロック長 (Byte)	入力ファイル (MB)	処理時間 (s)	復号時間 (s)
1	1	29	15
	3	89	48
	10	298	161
	33	987	532
	100	2988	1614
2	1	55	8
	3	89	24
	10	203	82
	33	583	274
	100	1685	829

表 5.2　ファイルサイズの違いによる処理時間:ブロック長

5.3　ブロック長の違いによるシェアファイルのサイズ比較

実装におけるシェアファイルのサイズを調べ理論とどれだけ違うか比較した．具体的にはブロック長を大きくしていき, それぞれについて 1MB〜100MB の入力ファイルを与え, シェアファイルのサイズの違いを計測した．ブロック長が 1Byte の時と 2Byte の時の結果を表 5.3 に示す.

また, 3Byte 以上の時は 5.1 節であげた理由により計測できなかった.

入力ファイルサイズ (MB)	出力ファイルサイズ (MB)	
	ブロック長 1Byte	ブロック長 2Byte
1	5	3
3	15	9
10	52	31
33	173	103
100	524	314

表 5.3　ブロック長の違いによる出力ファイルサイズ

第 6 章

考察

本章では性能測定の結果に対して考察を行った. 最初に素数と原始根を求める計算量について説明し, 次に処理速度について説明し, 最後にシェアサイズについて説明する. 最後に ssar コマンドについて説明する.

6.1 素数と原始根を求める計算量

今回のプログラムではブロック長が 1Byte, 2Byte の時は順調に計算することができた. しかし, ブロック長が 3Byte のとき, 全体の 8 分の 1 まで計算したところでプログラムが止まった. 止まった原因は, 原始根を計算する過程において, プロセスが使用可能なメモリより大きなメモリを必要としたためである.

また, SSS の全体の計算量において支配的なのは原始根を求める計算である. 例えば今回の実装で 3Byte の原始根を求める部分の計算が止まるまでの時間は 10 時間であったので, 順調に計算できたとすると 80 時間程度と予想する. 次回は計算量を減らすためにアルゴリズムの改良も試す予定である. しかし, 原始根を求める計算は数学的に本質的な問題であり, アルゴリズムにより高速化することは難しい. そのため劇的な効果は期待できないと予想される.

今回のアルゴリズムでは, 原始根を判別する処理の中間データをメモリ上に保持している. この中間データをファイルに保存することにより止まらなくすることができるできる. しかし, 処理時間がかなり長くなると予想される.

6.2　処理速度

性能測定の結果よりブロック長が 2Byte の時の暗号化における処理時間 $f(x)$ はファイルサイズの大きさ x(MB) と次のような関係式が成り立つ．ここで傾きは 16 秒/MB であり，グラフの y 切片は素数と原始根を計算する時間で，39 秒である．

$$16x + 39 \tag{6.1}$$

同様に復号化の時は以下のような式が得られる．

$$8.2x \tag{6.2}$$

これらの式の傾きはブロック長と関係しており，ブロック長を大きくすることにより傾きを小さくすることができる．

また，測定結果から暗号化と復号化のそれぞれにおいてファイルサイズと処理時間の両対数グラフを作成すると図 6.1 のようになる．この図から，処理速度はファイルサイズが増えてもリニアに推移しており実用的といえる．

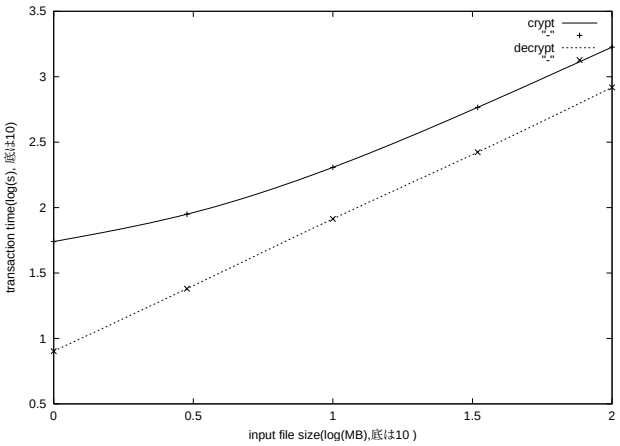

図 6.1　ファイルサイズの違いによる処理時間

6.3 シェアファイルのサイズ

今回の実装では Shamir の (k, n) 閾値 SSS を採用したのでシェアファイルのサイズはソースファイルのサイズと同じになるはずである. しかし, 測定結果を見るとシェアファイルのサイズはソースファイルより大きくなっている. なぜならシェアデータを格納する mpz 型を保存する時に, 数値自体に加えて 4Byte のオーバヘッドが生まれるためである.

このオーバヘッドはブロック長と関係がある. 例えば, 各分散ファイルのサイズはブロック長が 1 バイトの時, ソースファイルの 5 倍になり, ブロック長が 2 バイトの時ソースファイルの 3 倍になった. ブロック長が大きくなると mpz のオーバヘッドが相対的に小さくなる. よって, シェアファイルのサイズをソースファイルに近づけることができる.

第 7 章

まとめ

本章では Shamir の (k, n) 閾値 SSS を実装した結果をまとめる. また, 今後の課題について説明を行う.

7.1 結果

本研究では SSS がどういう理論であるかを説明し, データ分散型の保管手法である RAID と比較を行った. また, Shamir の (k, n) 閾値 SSS の実装において仕様や設計, そして実装の演算に対して説明を行った. そして実装したコマンドの性能測定を行いそれに対して考察を行った.

今回の実装ではブロック長が小さい時, 処理時間はファイルサイズに対してリニアに推移するので十分実用的な範囲に収まった. しかし, ブロック長が大きい時は計算できず, 計算できたとしても処理時間が非常に大きくなると予想できた.

ブロック長を大きくすることには, シェアサイズの効率化やハッキングされた時にデータが洩れにくくなるといった利点がある. そのため, 今後は 2 の拡大体 [13] における SSS を実装し, ブロック長を大きくし, その時の処理時間の短縮をはかる.

7.2 2 の拡大体における SSS

SSS においては, 素数と原始根を求める計算は必須であり, 大きな素数とその原始根を求める計算は非常に時間がかかる. この問題を解決するための一つの方法は 2 の拡大体上で SSS を使うことである. 2 の拡大体 [13] を使うと原始根を求める計算が不要になる. よって,

劇的な効率改善が期待できる. ただし, 今度は 2 の拡大体上の加法乗法の計算量が支配的になる. そのため, 実装する際には工夫する必要がある. 例えば, 最初に有限体上の全ての元に対する加法と乗法の表を構成する方法が有望だろう.

7.3 ネットワーク転送機能の付加

作成したシェアファイルを安全に保管するためには, ftp などで遠隔地にシェアファイルを送らなければならない. シェアファイルが遠隔地にあるとすると, ソースファイルを復元するためには, 閾値個のシェアファイルを遠隔地から持って来る必要がある. この作業を毎回手動で行うのは手間がかかる. 次のバージョンアップでは, 遠隔地との通信機能を持たせ, シェアファイルの遠隔地への転送と遠隔地からの転送を自動で行う機能を付加する予定である.

7.4 ファイルシステムへの応用

今回のプログラムはファイルに対して明示的に SSS を適用し, 作成されたシェアファイルをローカルホストのハードディスクに保存している. SSS をファイルシステムに実装することにより, すべてのデータに対してユーザが指示することなく分散暗号化を行うことが可能になる. また, シェアファイルをリモートのハードディスクに保存する機能も自動的に行うようにし, ユーザに SSS を使用していることを意識させないようにしたい.

謝辞

本研究を行うにあたり, 御指導, 御助言を頂いた福本 昌弘 講師に深く感謝致します.

また, 本研究の全過程を通じて御指導, 御教示を頂いた菊池 豊 助教授 に深く感謝致します.

加えて, 本研究において御助力をいただいた福本研究室の秋山 由佳氏, 庄田 亜輝氏に心から感謝いたします.

最後に, 本論文をまとめるにあたって御協力頂いた菊池研究室の諸氏に感謝いたします.

参考文献

[1] J. Benaloh and J. Leichter. Generalized Secret Sharing and Monotone Functions. In *Proc of Crypto'88*, LNCS403, pp. 27–35. Springer-Verlag, 1990.

[2] G.R Blakley. Safeguarding Cryptographic Keys. *Proc. of AFIPS 1979 Nat. Computer Conf.*, Vol. 48, pp. 313–317, 1979.

[3] R. Blakley, G and Meadow C. Security of ramp schemes. In *Proc. of Crypto'88*, LNCS403, pp. 252–268. Springer-Verlag, 1990.

[4] C. Blundo, A. De Santis, D.R. Stinson, and U. Vaccaro. Graph Decompositions and Secret Sharing Schemes. In *Proc. of Eurocrypt'92*, LNCS658, pp. 1–24. Springer-Verlag, 1993.

[5] D.M. Brickell, E.F.and Davenport. On the Classification of Ideal Secret Sharing Schemes. *Jounal of Cryptography*, Vol. 5, pp. 123–134, 1991.

[6] E.F. Brickell. Some Ideal Secret Sharing Schemes. *Jounal of Comb. Math. and Comb.comp.*, Vol. 9, pp. 105–113, 1989.

[7] B. Chor, S. Goldwasser, S. Michli, and B. Awebuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. *Proc. of FOCS*, pp. 383–395, 1985.

[8] T. Katoh and H. Imai. On Reducing the Share Size of Visual Secret Sharing Schemes. *Proc. of 1996 Int.Sym. on Inform. Theory and Its Appl.*, pp. 67–70, 1996.

[9] M. Naor and .A Shamir. A:Visual Cryptography. In *Proc. of Crypto'94*, LNCS950, pp. 3–12. Springer-Verlag, 1995.

[10] A. Shamir. How to share asecret. *Communication of the ACM*, Vol. 22, No. 11, pp. 612–613, 1979.

[11] G.j. Simmons. How to (Really) Share a Secret. In *Proc. of Crypt'90*, LNCS403, pp. 390–448. Springer-Verlag, 1990.

[12] H Yamamoto. On Secret Sharing Communication Systems with Two or Three Channels. *IEEE Trans. on Inform.Teory*, Vol. IT-32, No. 3, pp. 387–393, May 1986.

[13] 渡辺敬一, 草場公邦. 代数の世界, 3. 体. 代数の世界, 1994.

[14] 山本博資. (k,l,n) 閾値秘密分散システム. 電子通信学会論文誌, Vol. J68-A, No. 9, pp. 945–952, September 1985.

[15] 宇野俊夫. デイスクアレイテクノロジ RAID. エーアイ出版, 2000.

[16] 上原輝昭, 西関隆夫, 岡本栄司, 中村勝洋. マトロイド的アクセス構造を持つ秘密共有法. 電子通信学会論文誌, Vol. J69-A, No. 9, pp. 1124–1132, September 1986.

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

[18] 岡本龍明, 山本博資. 現代暗号, 14. 秘密分散法. 産業図書, 1997.

付録 A

視覚型秘密分散法

視覚型秘密分散法 (視覚型 SSS) は復号に計算を一切必要としない SSS の理論である．秘密情報を白黒画素の画像とし, 暗号化により n 枚の白黒画素の画像を作成する．これを分散画像と呼ぶ．また, 復号は分散画像を閾値 k の数だけ重ね合わせる事により元の画像が浮かびあがるようになる．

視覚型 SSS の構成の一つは暗号化において 1 つの画素に対して 4 つの subpixel で置き換えを行い分散画素を作成する．この置き換えは分散データを重ね合わせた時, 黒い画素だったところは 4 つの subpixle がすべて黒くなるようになり, 白い画素だったところは subpixel がすべて黒にならずに白い subpixel が残るようにする．

例として元の画像から 3 枚の分散画像を作成し, 3 枚揃うと元に戻る時の説明を行う．画素の置き換えは図 A.1 に示すパターンを使用して行う．

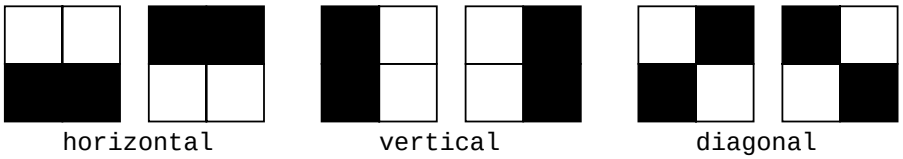

図 A.1　画素置き換えのパターン

元の画素が黒だった場合, 最初に horizontal と vertical と diagonal との中からランダムに一つを選ぶ．ここでは, horizontal の左側を選んだとする．次に残りの vertical と diagonal から 1 つをランダムに選ぶ．ここでは diagonal の右側を選んだとする．3 つの分散画素のうち 2 つが決まると残り一つは分散画素を重ね合わせた時 4 つの subpixel がすべて黒くなるように決定するので vertical の右側に決まる．

元の画素が白だった場合は最初に horizontal と vertical と diagonal の中からランダムに一つを選ぶ．ここでは, vertical の右側を選んだとする．次に残りの horizontal と diagonal から 1 つをランダムに選ぶ．ここでは diagonal の左側を選んだとする．3 つの分散画素のうち 2 つが決まると, 残り一つは分散画素を重ね合わせた時 4 つの subpixel の中に白が残るように決定するので horizontal の左側に決まる．

このように黒画素と白画素を置き換えると分散画像を必要枚数重ね合わせた時, 元の画像において黒画素であった部分が人間の目で黒く浮かびあがって見える．

視覚型 SSS において図 A.1 のパターンを使用すると分散画素のパターンは全部で 24 通りになる．分散画像は 24 通りの中から一つをランダムに選ぶ作業を繰り返すことにより作成されている．