令和 2 年度 卒業研究 論文
秘密分散の実装における速度・容量性能の評価

三浦夢生

2021 年 2 月

概要

様々なシステムやサービスがディジタル・オンライン化される流れにある現代社会において，情報の紛失や盗難への対策は重要性を増すばかりである．

これに対する技術として「秘密分散」があげられる．秘密分散とは元の秘密情報をアルゴリズムに基づいて作成した分散情報を管理する者や端末に配布・保管し，必要な数だけ集めて復元する技術のことである．

本研究では秘密分散技術のうち，加法的秘密分散・(k, n) しきい値秘密分散・(k, L, n) しきい値秘密分散を，シェルスクリプト及び Python を用いて実装し，秘密分散における一連の処理にかかった時間と生成されたシェアの平均サイズをまとめ，結果をもとに各アルゴリズムについて評価・考察を行う．

目次

第 1 章

はじめに

様々なシステムやサービスがディジタル・オンライン化される流れにある現代社会において，情報の紛失や盗難への対策は重要性を増すばかりである．例として，機密情報の複数人による管理や各地に配置されたストレージで情報の保管をする場合などがある．また，オンライン決済やネットバンキングの普及により管理しなければならない機密情報なども増えている．

これらに対する技術として「秘密分散」があげられる．秘密分散とは元の秘密情報をアルゴリズムに基づいて作成した分散情報を管理する者や端末に配布・保管し，必要な数だけ集めて復元する技術のことである．このとき，元の秘密情報から分散情報を作成する者・端末のことを「ディーラー」，配布された分散情報を保守・管理する者・端末を「参加者」，一連の処理において扱われる分散情報を「シェア」という．アルゴリズムごとに設定された数だけシェアを集めなければ，元の秘密情報を得ることはできない．この技術は秘密情報を分散して管理したり，複数人で秘密情報の完全性を担保したりする性質から，クラウドサービスやブロックチェーンと相性が良いため広く用いられている．

この技術の代表例として Shamir の提案した (k, n) しきい値秘密分散 [1] があげられる．これは元の秘密情報から n 個のシェアを生成し，k 個のシェアから秘密情報の復元ができるが，$k-1$ 以下の個数のシェアからは元の情報は全く得られない手法である．

本研究では，加法的秘密分散 [2]・(k, n) しきい値秘密分散 [1]・(k, L, n) しきい値秘密分散 [3][4] の，3 通りのアルゴリズムを実装し，実行速度及び生成されたシェアの平均サイズについて評価し，いずれが優れたアルゴリズムか，高速化の余地や改善点について考察を行う．

第 2 章

秘密分散

本章では，今回実装した秘密分散アルゴリズムの概要について述べる．

2.1 加法的秘密分散 [2]

加法的秘密分散は n 個のシェアに対して n 個のシェアからのみ元の情報が復元できるため，(n, n) しきい値秘密分散ともいえる．生成されるシェアのサイズは，生成する乱数の範囲によって決まる．

この手法はまず乱数を用いて，シェア $s_i, i = 2, 3, \cdots, n$ を以下のように生成する．

$$
\begin{aligned}
s_2 &= r_1 \\
s_3 &= r_2 \\
&\vdots \\
s_n &= r_{n-1}
\end{aligned}
$$

次に秘密情報 S を用いて以下のようにシェア s_1 を生成する．

$$s_1 = S - (s_2 + s_3 + \cdots + s_n)$$

復元の際には，参加者に配布したシェアを全て集めて足し合わせることで秘密情報 S を得る．

上記の計算を素数 p を標数とする体の上で行う手法もある．また用いる演算が加算のみなので，環上で計算を行うこともできる．発展させた手法として，シェアを一人に複数個割り当てることで後述の (k, n) しきい値秘密分散を実現する手法 ((k, n) 複製型秘密分散という) も示されている．[2]

2.2 (k, n) しきい値秘密分散 [1]

この手法は開発者の名前をとって Shamir の秘密分散とも呼ばれる．(k, n) しきい値秘密分散は n 個のシェアに対して k 個のシェアから元の秘密情報が復元できるが，$(k-1)$ 個以下のシェアからは元の秘密情報に関する情報は全く得られない．生成されるシェアは元の秘密情報と同等のサイズとなる．

この手法は秘密情報 S を定数項とする，ランダムな係数をもつ $k-1$ 次多項式を生成する．

$$f(x) = S + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

また参加者に番号 i を割り振り，$f(i)$ を計算し，シェアとして参加者に渡す．復元には k 個のシェア $(i, f(i)), i = 1, 2, \cdots, k$ を持ち寄り，k 個の式を立てて連立方程式を解くことで秘密情報 S が求められる．た

だし，参加者の番号によらず，決められた数を集めればよい．

$$\begin{cases} f(1) = S + a_1 \cdot 1 + a_2 \cdot 1^2 + \cdots + a_{k-1} \cdot 1^{k-1} \\ f(2) = S + a_1 \cdot 2 + a_2 \cdot 2^2 + \cdots + a_{k-1} \cdot 2^{k-1} \\ \qquad\qquad \vdots \\ f(i) = S + a_1 \cdot i + a_2 \cdot i^2 + \cdots + a_{k-1} \cdot i^{k-1} \end{cases}$$

このときラグランジュ補間を用いて $x = 0$ の場合を計算するとよい．

$$L(x) = \sum_{i=0}^{k} y_i l_i(x)$$

$$l_i(x) = \prod_{j=0, j \neq i}^{k} \frac{x - x_j}{x_i - x_j}$$

$k-1$ 個以下のシェアからは k 個の式は立たず，解が求まらないため元の秘密情報に関する情報が得られないことは容易にわかる．コンピュータ上で連続値は扱えず，またラグランジュ補間において除算を用いているため以上の計算を有限体上で行うと都合がよい．

2.3 (k, L, n) しきい値秘密分散 [3][4]

Shamir の (k, n) しきい値秘密分散を拡張した (k, L, n) しきい値秘密分散 は n 個のシェアに対して k 個のシェアから元の秘密情報が得られる．$k - L$ 個以下のシェアからは元の情報に関する情報は得られず (k, n) しきい値秘密分散と比べて各シェアのサイズが $1/L$ になる利点をもつが，$k - l(0 \leq l \leq L-1)$ 個のシェアからは断片的に元の秘密情報に関する情報が得られてしまう．しかし，各シェアのサイズが $1/L$ になる性質は非常に強力であり，実用性に長けているといえる．

この手法は秘密情報 S を以下のように L 個に分割し，0 から $L-1$ 次の項の係数として用いる．

$$S = s_1 || s_2 || \cdots || s_L$$

また L から $k-1$ 次の項には乱数を係数として用いて以下の多項式を生成する．

$$\begin{aligned} f(x) = s_1 + s_2 x + \cdots + s_L x^{L-1} \\ + a_0 x^L + a_1 x^{L+2} + \cdots + a_{k-L-1} x^{k-1} \end{aligned}$$

次に，参加者に番号 i を割り振り，$f(i)$ を計算し，シェアとして参加者に渡す．復元も (k, n) しきい値秘密分散と同様に連立方程式を解くことで秘密情報の断片を得た後，分割した際と逆の手順で結合を行い，元の秘密情報を得る．

$$\begin{cases} f(1) = s_1 + s_2 \cdot 1 + \cdots + s_L \cdot 1^{L-1} \\ \qquad + a_0 \cdot 1^L + a_1 \cdot 1^{L+2} + \cdots + a_{k-L-1} \cdot 1^{k-1} \\ f(2) = s_1 + s_2 \cdot 2 + \cdots + s_L \cdot 2^{L-1} \\ \qquad + a_0 \cdot 2^L + a_1 \cdot 2^{L+2} + \cdots + a_{k-L-1} \cdot 2^{k-1} \\ \qquad\qquad \vdots \\ f(i) = s_1 + s_2 \cdot i + \cdots + s_L \cdot i^{L-1} \\ \qquad + a_0 \cdot i^L + a_1 \cdot i^{L+2} + \cdots + a_{k-L-1} \cdot i^{k-1} \end{cases}$$

この手法における計算も有限体上で構成することでコンピュータによる実行が可能となる．

第 3 章

実装

本章では，今回作成したソースコードについて，シェルスクリプト部と Python 部にわけて説明を行う．

今回は Zsh5.8 及び Python3.7.9 の環境でソースコードの作成を行った．

プログラム全体の流れを図 3.1 に示す．シェルスクリプトで秘密ファイルの前後処理と，Python コードの実行を行っている．Python では実際の秘密分散処理とファイル入出力を行っている．

また，秘密ファイルとして「This is the Secret!」という 1 行の文を 1 万行並べた 200KB のテキストファイルを用いた．

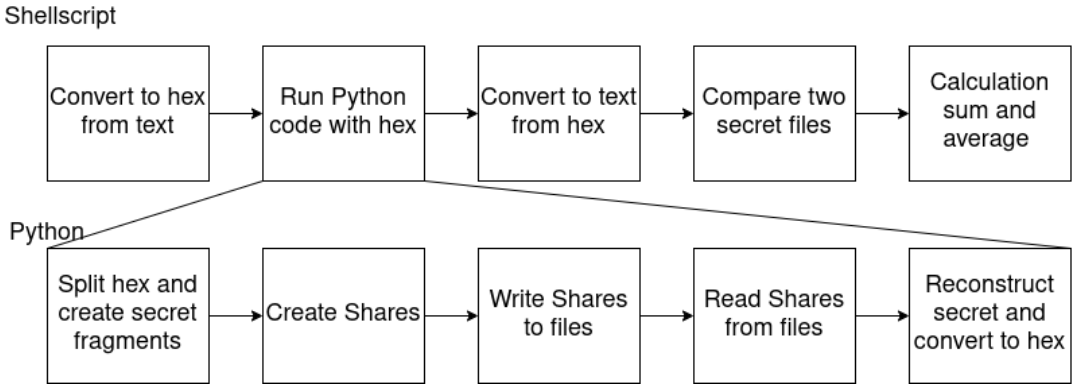

図 3.1　実装したプログラム全体の流れ.

3.1　シェルスクリプト部

シェルスクリプトのみの流れを示したものを図 3.2 に示す.

まず，引数として受け取った秘密ファイルを xxd コマンドで 16 進数に変換し，1 行あたり 512 ビットずつ書き込んでファイルとして保存する．次に，16 進数のファイルを Python のプログラムに渡して実行する．このとき，time コマンドを用いて実行することで，実行にかかった時間を計測している．その後，Python のプログラムによって生成された 16 進数のファイルを xxd コマンドでテキストに変換し，元のテキストファイルと再構築された秘密ファイルの差分を diff コマンドで比較する．最後に，Python によって生成された各シェアのサイズから，合計と平均を求めて表示する．

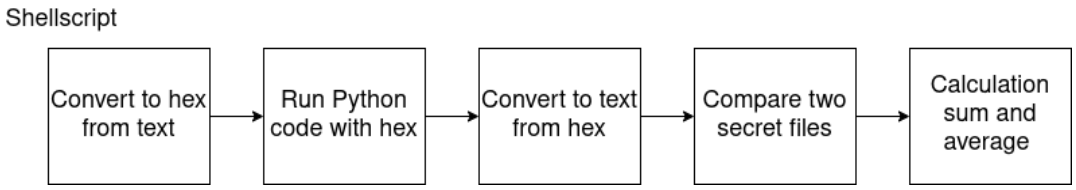

図 3.2　実装したシェルスクリプトの流れ.

3.2　Python 部

Python のソースコードのみの流れを示したものを図 3.4 に示す.

ファイル入出力や秘密情報の分割については秘密分散と切り離すため別のプログラムを作成し，それぞれのアルゴリズムにインポートして使用している.

リスト A.6 のファイルでは，各秘密分散には 16 進数を 4 文字ずつ読み込み，64 ビット分の情報を一つの 10 進数の秘密情報として渡している．リスト A.7 では，秘密分散によって生成されたシェアを，インデックスごとに分けてファイルに書き込んでいる.

リスト A.8 では，設定された数だけシェアを読み込んで，配列にしている．このとき，シェアは 1 から設定された数まで順番に利用される．リスト A.9 では，再構築された 10 進数の秘密情報の配列を 16 進数に変換し，再構築後の 16 進数の秘密情報としてファイルに書き込んでいる.

どの秘密分散においても，一つの 10 進数の秘密情報に対して，設定された数だけシェアを作成している．また，今回用いている乱数のシード値は固定した.

加法的秘密分散については，生成するシェア数を $n = 11$ とし，用いる乱数の範囲を $0 \leq r \leq 2^{16} - 1$ とした．復元には $n = 11$ 個のシェアを読み込み，単純な加算を用いた.

(k, n) しきい値秘密分散では再構築可能なシェア数を $k = 4$，生成するシェア数を $n = 11$ に設定し，有限体の標数は $p = 65537$ とした．復元には $x = 0$ におけるラグランジュ補間を用いた．図 3.3 に示すように，$x = 0$ において得られる関数の値は多項式の定数項とした秘密情報と一致するため，復元が可能となっている.

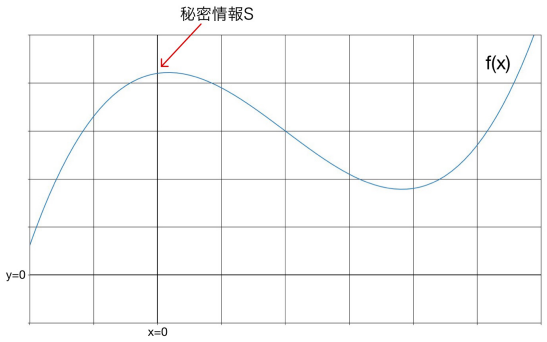

図 3.3　秘密情報取得の概念図.

(k, L, n) しきい値秘密分散では再構築可能なシェア数を $k = 4$，分割パラメータを $L = 2$，生成するシェア

数を $n = 11$ に設定し，有限体の標数は $p = 65537$ とした．復元はヴァンデルモンド行列による逆行列計算を用いた．ヴァンデルモンド行列とは，以下に示す連立方程式のうち，係数行列のような形式のものを指す．

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{k-1} & x_{k-1}^2 & \cdots & x_{k-1}^{k-1} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_L \\ a_0 \\ a_1 \\ \vdots \\ a_{k-L-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{k-1} \end{pmatrix}$$

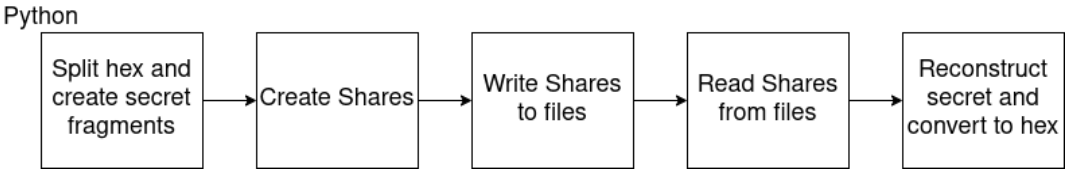

図 3.4　実装した Python の流れ．

第 4 章

結果・考察

Python 部のみの実行時間を計測した結果と平均シェアサイズを表 4.1 に示す．実行時間は秘密情報の分散処理及び復元処理が終わるまでの時間を示す．また，平均シェアサイズはシェルスクリプト実行によって表示されたサイズの上位から 4 桁目を四捨五入した値を示す．

表 4.1　実行時間及びシェアの平均サイズ．

アルゴリズム	実行時間 [s]	平均サイズ [KB]
加法的秘密分散	2.714	約 603
(k, n) しきい値秘密分散	6.956	約 583
(k, L, n) しきい値秘密分散	2145.62	約 583

平均シェアサイズについては，どの秘密分散においても，最大サイズが 16 ビットにほぼ収まっているため大きな違いは見られなかった．加法的秘密分散は各シェアを $0 \leq r \leq 2^{16} - 1$ の範囲でランダムに生成し，除算を行っていないため，演算を有限体上で行い，演算結果がほぼ 16 ビット以内になる他の二つと比べて平均シェアサイズが大きくなっている．また今回の実装方法の場合，(k, L, n) しきい値秘密分散と (k, n) しきい値秘密分散の平均シェアサイズがほぼ同等となり，シェアサイズが $1/L$ になるという性質の効果が見られなかった．これは，(k, L, n) しきい値秘密分散において，一つの秘密情報を分割してから秘密分散処理を行っているためだと考えられる．解決法として，一つの秘密情報を分割するのではなく，指定されたパラメータの分だけ秘密情報を用いて秘密分散処理を行うことがあげられる．

実行時間について，乱数の生成や加減算のみを行うシンプルな加法的秘密分散が最も速い結果となった．次点で (k, n) しきい値秘密分散が速く，(k, L, n) しきい値秘密分散は，今回秘密情報の断片を取り出すのに逆行列計算を用いているため非常に時間がかかった結果となった．加法的秘密分散は復元処理に単純な加算のみを用いているため $O(n)$ であり，秘密ファイルのサイズを増やしたときに実行時間は線形増加すると考えられる．対して，(k, n) しきい値秘密分散は復元処理にラグランジュ補間を用いているため $O(n^2)$，(k, L, n) しきい値秘密分散は復元処理に逆行列計算を用いているため $O(n^3)$ と，計算量の多い処理をしているため，非線形増加をすると考えられる．

第 5 章

まとめ

本研究では，加法的秘密分散・(k, n) しきい値秘密分散・(k, L, n) しきい値秘密分散及び実行やファイル入出力に関する周辺プログラムを実装し，実行速度及び生成されたシェアの容量について評価・考察した．本研究の実装手法・評価項目においては実行時間が二番目に速く，シェアサイズが比較的小さい (k, n) しきい値秘密分散が最も優れているという結果となった．

今後の展望として，まず (k, L, n) しきい値秘密分散について，与えられた秘密情報を再度分割するのではなく，L 個ずつ秘密分散の処理にかけることでシェアのサイズ削減ができる．また，逆行列計算よりも計算量の少ないアルゴリズムを検討することでより高速な動作が期待できる．

加法的秘密分散に対しても，有限体上や有限環上で実装ができるため，その条件も他の 2 つと合わせた場合に結果にどのような影響が出るのかを比較することもあげられる．

次に，比較対象として，AONT 秘密分散 [5] や Krawczyk の方式 [6] などを実装することで，より多くのアルゴリズムがらより良いものを選べる．AONT(All-Or-Nothing Transform) 秘密分散とは，”All-Or-Nothing form”という形式の擬似メッセージを用いた暗号を，秘密分散に応用した手法である．”All-Or-Nothing form”とは，メッセージ $m_1, m_2, \cdots, m_s$ に対し，以下のように計算されたメッセージ m'_i のことである．

$$m'_i = m_i \oplus E(K', i) i = 1, 2, \cdots, s$$

ここで，K' は大きな空間からランダムに選んだ秘密鍵である．上記の操作は，逆変換が可能であるため秘密分散に応用ができる．

Krawczyk の方式とは，秘密情報を暗号化し，暗号文と秘密鍵をそれぞれ別のアルゴリズムで分散処理し，分散後の情報のペアをシェアとする手法である．

また評価項目として，一つのシェアから得られる秘密情報の情報量や，秘密の分散・再構築時にかかった時間の項目，各秘密分散について乱数の範囲やパラメータを変更した際の項目を設けたり，様々な大きさの秘密ファイルを秘密分散処理したりすることで，より多角的な評価・考察が可能となる．

一つの情報量から得られる秘密情報の情報量を追加することで，安全性と実行時間のトレードオフについての検討などができる．また，その他の項目追加で，秘密分散処理のどの部分に時間がかかるのかなど，詳しい性質の評価につながる．

参考文献

[1] A Shamir, “How to Share a Secret”, Communications of the ACM, Vol.22 pp612–613, 1979.

[2] 大原一真, “秘密分散を用いた秘密計算”, システム/制御/情報, Vol.63, pp71–76, 2019.

[3] 山本博資, “(k,L,n) しきい値秘密分散システム”, 電子通信学会論文誌, Vol.J68-A, pp945–952, 1985.

[4] 千田浩司・五十嵐大・菊池亮・濱田浩気, “計算量的秘密分散およびランプ型秘密分散のマルチパーティ計算拡張”, 研究報告コンピュータセキュリティ, Vol.2012-CSEC-58, pp1–5, 2012.

[5] Ronald L. Rivest, “All-or-Nothing Encryptionand the Package Transform”, Fast Software Encryption, Vol.LNCS 1267, pp210–218, 1997.

[6] Hugo Krawczyk, “Secret Sharing Made Short”, CRYPTO’93, Vol.LNCS 773, pp136–146, 1994.

付録 A

ソースコード

本研究で作成したソースコードを以下に示す.

A.1 シェルスクリプト部

結果に影響を出さないため，秘密分散のソースコード以外は同一のプログラム構造を用いている.

今回は例として加法的秘密分散の際に用いるシェルスクリプトを示す．他 2 つの秘密分散では 15 行目の Python ファイル名，41 行目のメッセージ部分を変更して利用している．リスト A.1 はシェルスクリプト内で用いている time コマンドのフォーマットを zsh の設定ファイルから一部抜粋して示している．またリスト A.2 内の time コマンドの引数は，ファイルパスを省略している.

このシェルスクリプトは，実行するために対象となるファイルを用意し，引数として渡す必要がある．また，実行すると”secret.hex”，”secret_reconst.hex”，”secret_reconst.txt”，”sharesize_sum.txt”が生成される.

Listing A.1 .zshrc(一部抜粋)

```
TIMEFMT=$'\n===============\nCPU :%P\nuser :%*Us\nsystem :%*Ss\ntotal :%*Es\n
    ===============\n'
```

Listing A.2 ASS.sh

```
#! /bin/zsh

source ~/.zshrc

if [ $# -eq 1 ]; then
        FILE1="secret.hex"
        if [ -e $1 ]; then
                xxd -p $1 > ${FILE1}
                echo "Succeeded convert to hex."
        else
                echo "Error occurred.(1)"
        fi

        if [ -e ${FILE1} ]; then
                time Python3.7 AdditiveSecretSharing.py ${FILE1}
```

```
16                  echo "Succeeded Secret Sharing."
17          else
18                  echo "Error occurred.(2)"
19          fi
20
21          FILE2="secret_reconst.hex"
22          FILE3="secret_reconst.txt"
23          if [ -e ${FILE2} ]; then
24                  touch ${FILE3}
25                  xxd -p -r ${FILE2} > ${FILE3}
26                  echo "Succeeded convert to text."
27          else
28                  echo "Error occurred.(3)"
29          fi
30
31          if [ -e ${FILE3} ]; then
32                  diff -s $1 ${FILE3}
33                  echo "Succeeded reconstruct."
34          else
35                  echo "Error occurred.(4)"
36          fi
37
38          ls -la | grep 'Share' | cut -d ' ' -f 5 > sharesize_sum.txt
39          awk '{ s += $size }; END { print "sum="s"[KB],", "average="s/NR"[KB]"}' <
                sharesize_sum.txt
40  else
41          echo "usage: ./ASS.sh [file]"
42  fi
```

A.2 Python 部

Python では大きく分けて 2 種類のソースコードを作成した．はじめに秘密分散のソースコード，次にファイル入出力関係のソースコードを示す．

どの秘密分散においてもファイル入出力は同じファイルを用いている．

A.2.1 加法的秘密分散

Listing A.3 AdditiveSecretSharing.py

```
1  import random
2  import file_split as fsplit
3  import file_output as foutput
4  import file_read as fread
5  import file_reconst as freconst
6
```

```
7  #
8  # create share
9  #
10 def create_share(_secret, _n):
11     share = []
12     for i in range(1, _n):
13         r = random.randint(0, 2**16-1)
14         share.append(r)
15     s = _secret - sum(share)
16     share.insert(0, s)
17     return share
18
19 #
20 # combine share
21 #
22 def combine_share(_share):
23     s = sum(_share)
24     return s
25
26 def main():
27     #
28     # define some constant
29     #
30     secret = fsplit.hex_to_int()
31     n = 11
32     random.seed(0)
33
34     #
35     # disperse imformation
36     #
37     shares = []
38     for i in range(len(secret)):
39         shares.append(create_share(secret[i], n))
40
41     #
42     # write file
43     #
44     temp = []
45     for i in range(len(shares[0])):
46         for j in range(len(shares)):
47             temp.append(shares[j][i])
48         foutput.write_share('Share', i + 1, temp)
49         temp = []
50
51     #
52     # read file
```

```
53      #
54      shares = fread.read_share('Share', n)
55
56      #
57      # reconstruct imformation
58      #
59      re_s = []
60      for i in range(len(shares)):
61          s = combine_share(shares[i])
62          if secret[i] == s:
63              re_s.append(s)
64      freconst.int_to_hex(re_s)
65
66  if __name__ == '__main__':
67      main()
```

A.2.2 (k, n) しきい値秘密分散

Listing A.4 ShamirSecretSharing.py

```
1  import random
2  import file_split as fsplit
3  import file_output as foutput
4  import file_read as fread
5  import file_reconst as freconst
6
7  #
8  # generate server ids
9  #
10 def generate_serverId(_n, _prime):
11     serverId = [i + 1 for i in range(_prime - 1)]
12     random.shuffle(serverId)
13     for i in range(_prime -_n):
14         serverId.pop(0)
15     return serverId
16
17 #
18 # generate coefficient of polynomial
19 #
20 def generate_polynomial(_secret, _k, _prime):
21     fx = [_secret]
22     for i in range(_k - 1):
23         fx.append(random.randint(0, _prime - 1))
24     return fx
25
26 #
```

```
27 # create share
28 #
29 def create_share(_serverId, _fx, _prime):
30     share = []
31     for i in _serverId:
32         temp = 0
33         for j in range(len(_fx)):
34             temp += _fx[j] * i ** j
35         temp %= _prime
36         share.append(temp)
37     return share
38 
39 #
40 # calculation of Lagrange Interpolation
41 #
42 def lagrange_interpolation(_dataX, _dataY, _prime):
43     dataNum = len(_dataX)
44     x = 0
45     l = 0
46     L = 0
47     for i in range(dataNum):
48         l1 = base_polynomial(dataNum, i, x, _dataX, _prime)
49         l2 = base_polynomial(dataNum, i, _dataX[i], _dataX, _prime)
50         temp1, l2_inv, temp2 = xgcd(l2, _prime)
51         l = l1 * l2_inv
52         L += _dataY[i] * l
53     L %= _prime
54     return L
55 
56 #
57 # calculation of base polynomial for Lagrange Interpolation
58 #
59 def base_polynomial(_dataNum, _i, _x, _dataX, _prime):
60     l = 1
61     for k in range(_dataNum):
62         if _i != k:
63             l *= _x - _dataX[k]
64     l = l % _prime
65     return l
66 
67 #
68 # calculation of inverse element on Galois Field
69 #
70 def xgcd(_a, _b):
71     if _a == 0:
72         return _b, 0, 1
```

```
73      else:
74          g, y, x = xgcd(_b%_a, _a)
75          return g, x - (_b//_a)*y, y
76
77  #
78  # choose share randomly by the number of shareNum and make list
79  #
80  def choose_share(_serverId, _w, _n, _shareNum):
81      useShare = [i for i in range(_n - 1)]
82      random.shuffle(useShare)
83      for i in range(_n - _shareNum):
84          useShare.pop(0)
85      print(f'using share number = {useShare}')
86      dataX = []
87      dataY = []
88      for i in useShare:
89          dataX.append(serverId[i - 1])
90          dataY.append(w[i - 1])
91      return dataX, dataY
92
93  def main():
94      #
95      # define some constant
96      # secret:original secret k:key num n:share num prime:prime
97      #
98      secret = fsplit.hex_to_int()
99      k = 4
100     n = 11
101     prime = 65537
102     random.seed(0)
103
104     #
105     # split secret
106     # generate server id and n degree polynomial then calculate share
107     #
108     fx = []
109     shares = []
110     serverId = generate_serverId(n + 1, prime)
111     for i in range(len(secret)):
112         fx = generate_polynomial(secret[i], k, prime)
113         shares.append(create_share(serverId, fx, prime))
114
115     #
116     # write file
117     #
118     for i in range(len(shares[0])):
```

```
119         temp = [serverId[i]]
120         for j in range(len(shares)):
121             temp.append(shares[j][i])
122         foutput.write_share('Share', i + 1, temp)
123 
124     #
125     # read file
126     #
127     shares = fread.read_share('Share', k+1)
128     dataX = []
129     dataY = []
130     for i in range(len(shares[0])):
131         dataX.append(shares[0][i])
132     for i in range(1, len(shares)):
133         temp = []
134         for j in range(len(shares[0])):
135             temp.append(shares[i][j])
136         dataY.append(temp)
137 
138     #
139     # reconstruct information
140     #
141     re_s = []
142     for i in range(len(dataY)):
143         s = lagrange_interpolation(dataX, dataY[i], prime)
144         if secret[i] == s:
145             re_s.append(s)
146     freconst.int_to_hex(re_s)
147 
148 if __name__ == '__main__':
149     main()
```

A.2.3 (k, L, n) しきい値秘密分散

Listing A.5 RampSecretSharing.py

```
1  import random
2  from sympy import Matrix
3  import file_split as fsplit
4  import file_output as foutput
5  import file_read as fread
6  import file_reconst as freconst
7  
8  #
9  # generate server ids
10 #
```

```
11 def generate_serverId(_n, _prime):
12     serverId = [i+1 for i in range(_prime-1)]
13     random.shuffle(serverId)
14     for i in range(_prime-_n):
15         serverId.pop(0)
16     return serverId
17 
18 #
19 # generate coefficient of polynomial
20 #
21 def generate_polynomial(_secret, _k, _L, _prime):
22     secBit = format(_secret, 'b')
23     bitLen = len(secBit)
24     if bitLen%8 != 0:
25         secBit = format(_secret, '0' + str(bitLen + (8 - bitLen%8)) + 'b')
26         bitLen = len(secBit)
27     fx = []
28     for i in range(_L):
29         temp = secBit[i*int(bitLen/_L):(i+1)*int(bitLen/_L)]
30         fx.append(int(temp, 2))
31     for i in range(_L, _k-1):
32         fx.append(random.randint(0, _prime-1))
33     return fx
34 
35 #
36 # create share
37 #
38 def create_share(_serverId, _fx, _prime):
39     share = []
40     for i in _serverId:
41         temp = 0
42         for j in range(len(_fx)):
43             temp += _fx[j] * i**j
44         temp %= _prime
45         share.append(temp)
46     return share
47 
48 #
49 # reconstruct secrets
50 #
51 def reconst_secret(_dataX, _dataY, _prime, _k, _L):
52     num = len(_dataX)
53     xList = []
54     for x in _dataX:
55         for i in range(num):
56             xList.append((x ** i) % _prime)
```

```
57      xMat = Matrix(num, num, xList)
58      yMat = Matrix(num, 1, _dataY)
59      xMat = xMat.inv_mod(_prime)
60      ansMat = (xMat * yMat) % _prime
61
62      bit = ''
63      for i in range(_L):
64          temp = format(int(ansMat[i]), 'b')
65          if len(temp)%8 != 0:
66              temp = format(int(ansMat[i]), '0' + str(len(temp) + (8 - len(temp)%8)) +
                    'b')
67          bit += temp
68      secret = int(bit, 2)
69      return secret
70
71  #
72  # choose share randomly by the number of shareNum and make list
73  #
74  def choose_share(_serverId, _w, _n, _shareNum):
75      useShare = [i for i in range(_n-1)]
76      random.shuffle(useShare)
77      for i in range(_n-_shareNum):
78          useShare.pop(0)
79      dataX = []
80      dataY = []
81      for i in useShare:
82          dataX.append(_serverId[i-1])
83          dataY.append(_w[i-1])
84      return dataX, dataY
85
86  def main():
87      #
88      # define some constant
89      # secret:original secret k:key num n:share num prime:prime
90      #
91      secret = fsplit.hex_to_int()
92      k = 4
93      L = 2
94      n = 11
95      prime = 65537
96      random.seed(0)
97
98      #
99      # split secret
100     # generate server id and n degree polynomial then calculate share
101     #
```

```
102     fx = []
103     shares = []
104     serverId = generate_serverId(n+1, prime)
105     for i in range(len(secret)):
106         fx = generate_polynomial(secret[i], k, L, prime)
107         shares.append(create_share(serverId, fx, prime))
108 
109     #
110     # write file
111     #
112     for i in range(len(shares[0])):
113         temp = [serverId[i]]
114         for j in range(len(shares)):
115             temp.append(shares[j][i])
116         foutput.write_share('Share', i + 1, temp)
117 
118     #
119     # read file
120     #
121     shares = fread.read_share('Share', k + 1)
122     dataX = []
123     dataY = []
124     for i in range(len(shares[0])):
125         dataX.append(shares[0][i])
126     for i in range(1, len(shares)):
127         temp = []
128         for j in range(len(shares[0])):
129             temp.append(shares[i][j])
130         dataY.append(temp)
131 
132     #
133     # reconstruct information
134     #
135     re_s = []
136     for i in range(len(dataY)):
137         s = reconst_secret(dataX, dataY[i], prime, k, L)
138         if secret[i] == s:
139             re_s.append(s)
140     freconst.int_to_hex(re_s)
141 
142 if __name__ == '__main__':
143     main()
```

A.2.4 ファイル入出力関係

ここでは順に「16 進数の秘密ファイルを 10 進数の秘密情報に分割するソースコード」,「シェアをファイルに書き込むソースコード」,「シェアを指定された数だけ順に読み込むソースコード」,「再構築された 10 進数の秘密情報を 16 進数に変換するソースコード」を示す.

Listing A.6 file_split.py

```
import sys

def hex_to_int():
    if len(sys.argv) < 2:
        print('too few arguments.')
        sys.exit()
    elif len(sys.argv) > 2:
        print('too many arguments.')
        sys.exit()

    file_name = sys.argv[1]
    HEX = ''
    with open(file_name, mode='r') as s:
        for l in s:
            HEX += l
    HEX = HEX.replace('\n', '')

    size = 4
    DEC = []
    h = ''
    temp = ''
    for i in range(len(HEX)):
        h = hex(int(HEX[i], 16))
        temp += h[2:]
        if i % size == size - 1:
            DEC.append(int('0x'+temp, 16))
            temp = ''

    return DEC
```

Listing A.7 file_output.py

```
def write_share(_name, _number, _shares):
    file_name = str(_name) + str(_number) + '.txt'
    with open(file_name, mode='w') as f:
        for i in range(len(_shares)):
            f.write(str(_shares[i])+'\n')
```

Listing A.8 file_read.py

```
def read_share(_name, _n):
    shares = []
    for i in range(_n):
        temp = []
        filename = str(_name) + str(i + 1) + '.txt'
        with open(filename, mode='r') as f:
            for l in f:
                l = l[:-1]
                temp.append(int(l))
        shares.append(temp)
    shares = [[shares[i][j] for i in range(len(shares))] for j in range(len(shares[0]))]
    return shares
```

Listing A.9 file_reconst.py

```
def int_to_hex(DEC):
    re_HEX = ''
    for i in range(len(DEC)):
        h = str(format(DEC[i], 'x'))
        re_HEX += h

    with open('secret_reconst.hex', mode='w') as f:
        f.write(re_HEX)
```