明治大学総合数理学部

２０１６年度

卒業研究

マイナンバーカード利用者証明用電子証明書を用いた電子署名アプリケーションの開発

学位請求者　先端メディアサイエンス学科

金子曜大

目次

1　はじめに

1.1　研究背景 1

1.2　マイナンバーカード 1

1.3　PDF の署名方式 3

1.4　研究目的 4

2　実験

2.1　マイナンバーカードを用いた電子署名 5

2.2　OpenSC 5

2.3　マイナンバーカードの秘密鍵を利用する試行実験

2.3.1　実装目的 6

2.3.2　実装環境 6

2.3.3　実装内容 6

2.3.4　実験結果 7

2.2　マイナンバーカードを用いた電子署名アプリケーションの開発

2.2.1　実験環境 10

2.2.2　実験内容 10

2.2.3　実験結果 10

2.2.4　性能評価実験

2.2.4.1　評価実験内容 14

2.2.4.2　評価実験結果 14

3　おわりに 17

付録 A　検索可能秘密分散

4　はじめに

4.1　研究背景 18

4.2　研究目的 18

4.3　秘密分散とは 18

5　実装

5.1　画像の秘密分散

5.1.1　実装目的 20

5.1.2 実装環境 20
5.1.3 実装内容 20
5.1.4 実装結果 20
5.2 検索可能秘密分散システムの実装
5.2.1 実装内容 23
5.2.2 実装結果 24
5.2.3 評価 28
6 まとめ 30
参考文献

1 はじめに

1.1 研究背景

現在の生活でインターネットは欠かせない存在となっているが、そのやりとりをしている相手が、自分が思っている人とは異なり、別人がなりすました人ではないかといいう心配がある。電子署名は認証局から発行される「電子証明書」を用いて、なりすましの防止や情報の改ざんを防止することのできる技術である。

しかし従来の電子署名にも問題があり、電子署名を行う際には PC 上のファイルに秘密鍵を格納する必要があり、その秘密鍵が漏洩する恐れがあった。従来の署名ツールである Acrobat では秘密鍵は PC 上に保存され、鍵のコピーは容易く行うことができるという問題点がある。

1.2 マイナンバーカードについて

マイナンバーカードは個人番号を証明する書類や、本人確認の際の公的な身分証明書として利用できる。マイナンバーカードには IC チップが内蔵されており、公開鍵電子証明書と 2048bit の RSA 鍵が内蔵されている。電子証明書は署名用電子証明書と利用者証明用電子証明書の 2 つが内蔵されている。これらは公的個人認証サービスを利用した行政手続き等を行うときに利用する。証明書は公的個人認証サービスポータルサイト(https://www.jpki.go.jp/)でダウンロードできる利用者クライアントソフトを用いることで、表示することができる。

署名用電子証明書には氏名や、生年月日、性別等が記されており、所得税の確定申告(e-Tax)等に利用する。利用者証明用電子証明書には主体者情報、発行年月日等が記されており、ネットワーク上のマイなポータルに入る時に等に利用する。なお、この主体者情報はランダム文字列と受付端末識別記号を文字列結合した文字列である。JPKI 利用者クライアントソフトで表示した電子証明書をそれぞれ示す。図 1 が署名用電子証明書の基本情報で、図 2 が署名用電子証明書の詳細情報。図 3 が利用者証明用電子証明書の基本情報、図 4 が利用者証明用電子証明書の詳細情報である。基本情報とは JPKI 利用者クライアントソフトが表示した証明書で、詳細情報とは実際の証明書に記されたデータである。

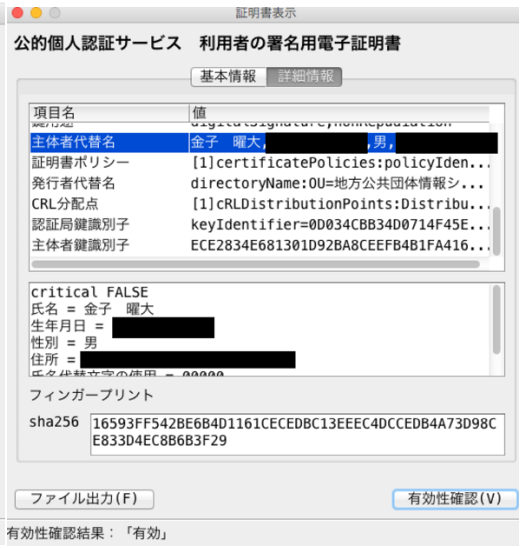


図１（左）：署名用電子証明書（基本情報）

図２（右）：署名用電子証明書（詳細情報）

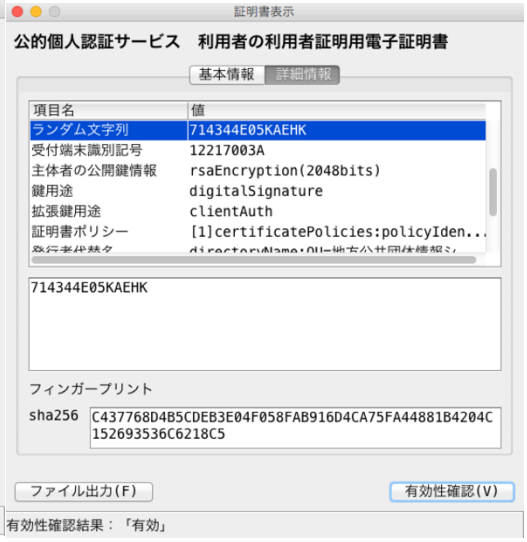


図３（左）：利用者証明用電子証明書（基本情報）

図４（右）：利用者証明用電子証明書（詳細情報）

マイナンバー制度は平成 28 年 1 月から開始された、社会保障と税と災害対策の行政手続きを目的とした制度である。また、導入時期は不確定であるが、銀行口座との紐付けや医療分野での利用、犯罪歴の記録などの導入が検討されている。しかし、マイナンバーカードの普及率は平成 28 年 12 月の時点でわずか 7.5%となっている。これらの交付率の伸び悩む理由としては、マイナンバーカードの利便性が感じられないことにあると考える。マイナンバーカードを取得することでどのような利点があるかというメリットをわかりやすく広告すべきだろう。

また、詐欺被害も報告されている。事件概要としては 70 代女性宅に男から電話があり、偽のマイナンバーを伝えられた。その後、別の男から「公的機関に寄付するため、マイナンバー貸して」と連絡があり、女性は番号を伝えた。翌日、公的機関を名乗るものから「番号を教えたことは犯罪に当たる」として、記録の書き換えを名目に金銭を要求され、現金数百万を支払ったという。マイナンバー制度は海外での実例が多いため、それらの実例を参考にすることができる。これらのことからマイナンバーカードを持ち歩く中で注意すべきこと、詐欺被害の実例などを挙げて犯罪防止を啓発すべきである。

1.3 PDF ファイルの署名方式

署名のついた PDF ファイルをテキストエディタやバイナリエディタ等で展開し、内部構造を追っていく。そうすると obj<<ByteRange[] /Contents<> …… endobj といった構造がある。このオブジェクトが署名辞書を表している。署名辞書の中には署名する際のハッシュ計算の対象と、署名値等が表示されている。ハッシュ計算の対象は ByteRange []に記されている。例えば ByteRange[0, 900, 1000, 200]と表示されていた場合は PDF の 0 バイト目から 900 バイト目までと 1000 バイト目から 1200 バイト目がハッシュ計算の対象となる。また署名値は /Contents <>に記されている。また Contents<>の続きには署名情報も記されている。署名情報は CMS フォーマットによるもので、署名者情報や証明書、執行情報等が記されている。

署名された PDF の構造を以下の図 5 に示す。

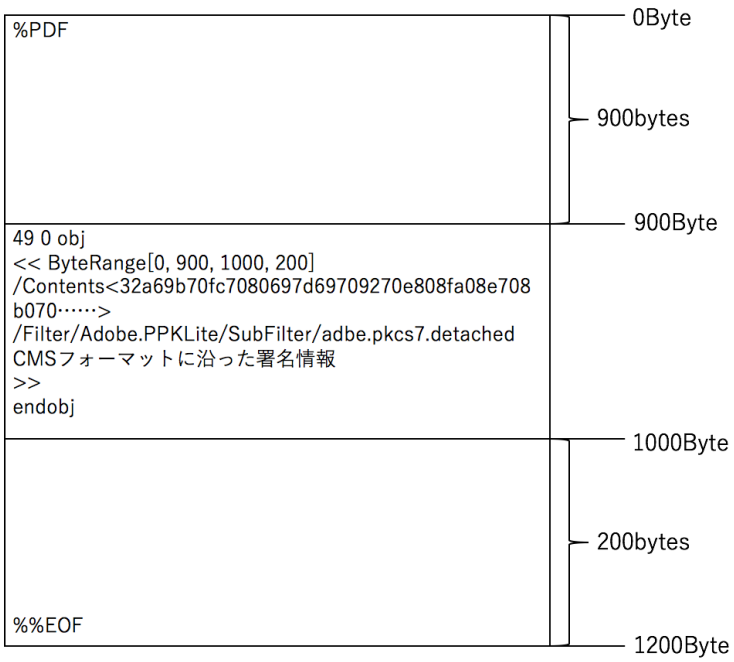


図 5 ：署名された PDF の構造

1.4 研究目的

秘密鍵を平文で格納するソフトウェアの脆弱性を解消し、安全に署名を行うソフトウェアを開発する。

公開鍵暗号方式を用いたスマートカードでは、秘密鍵が常に IC チップ内部に格納され、カードの改造を検知するとメモリをクリアする安全機構となっている。そのため、マイナンバーカードに格納されている秘密鍵が漏洩する恐れがない。そこでマイナンバーカードを用いた電子アプリケーションを開発することで秘密鍵が漏洩する脅威を防ぐ。また、マイナンバーカードの利用者証明用電子証明書を用いると、安全性が高いだけではなく、自治体から正式に認証された本人認証のある署名をすることができる。

２署名ソフト”Captain Signer”の開発

2.1 マイナンバーカードを用いた電子署名

マイナンバーカードには署名用電子証明書と利用者証明証電子証明書が内蔵されている。この利用者証明用電子証明書の秘密鍵を用いて電子署名を試みる。また、マイナンバーカードの利用者証明用電子証明書を用いていると、署名の安全性が高いだけではなく、自治体から正式に認証された本人認証のある署名となる。そのため、電子署名及び認証業務に関する法律[3]において認められた署名となる。実装のシステム構成図を以下の図６に示す。

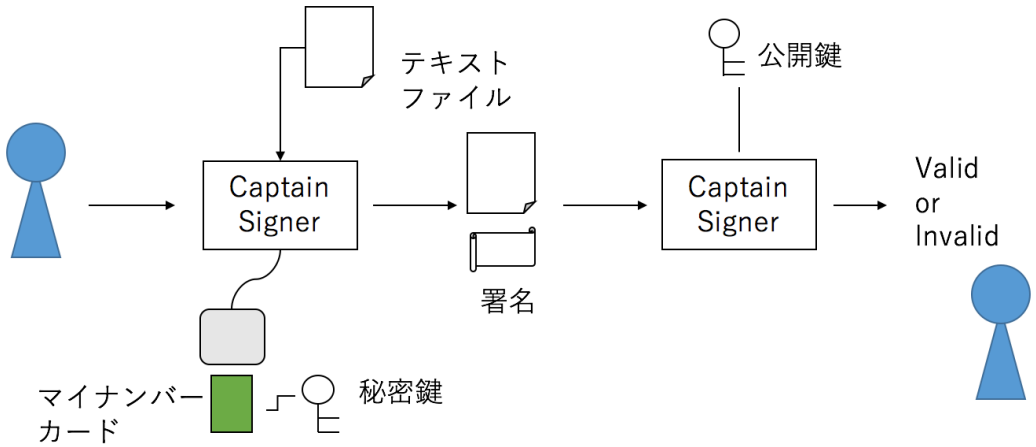


図６：システム構成図

また、マイナンバーカードと通信する API として OpenSC[4]を用いた。

2.2OpenSC

本実験ではマイナンバーカードと通信する API として OpenSC を用いた。OpenSC は OpenSC チームが作成した API であり、PKCS#15 と互換性のある IC カード、およびその他の暗号トークンの使用が可能である。マイナンバーカード以外にも、フィンランドの FINEID やエストニア EstEID など国が発行している ID カードをサポートしている。GitHub（https://github.com/OpenSC/OpenSC）でダウンロードできる。Linux、OSX、Windows 上で稼働する。

本研究では OpenSC の pkcs15-crypt コマンドと pkcs15-tool コマンドを用いた。PKCS とは RSA セキュリティにより公開されている公開鍵暗号標準のグループである。OpenSC の pkcs15-crypt はスマートカードに保存されている鍵を用いて、電子署名を計算したり、データを復号するなどの暗号化操作を実行するコマンドである。pkcs15-tool はスマートカードに格納されてい

る鍵や証明書などを読むことができるコマンドである。公開鍵情報を pem ファイルに書き込むコマンドの例を示す。

```
pkcs15-tool --read-public-key 1 > ./pubkey.pem
```

2.3 マイナンバーカードの秘密鍵を利用する試行実験

2.3.1 実装目的

マイナンバーカードの秘密鍵の利用が可能か試みる。

2.3.2 実装環境

PC

表 1 ：PC の実装環境

言語	Java
OS	OSX
メモリ	8GB

カードリーダ

表 2 ：カードリーダの製品情報

製品名	ACR39-NTTCom
適応カード	ISO7816-3 準拠カード （個人番号カード、住民基本台帳カードなど IC チップがついたタイプ）
サイズ/重量	14.0mm(H) × 72.2mm(W) × 69.0mm(D) / 57g
インターフェース	USB2.0 Full Speed(12Mbps)
電源	電圧：5V　消費電力：Max50mA
対応 OS	Windows 7, 8, 8.1, 10, Vista OSX 10.8, 10.9, 10.10

2.3.3 実装内容

マイナンバーカードを用いて署名。また検証をする。Captain Signer の中心となる処理である。ある文字列のハッシュ値からマイナンバーカードの秘密鍵を用いて署名値を算出し、署名をする。また署名値とマイナンバーカードの公開鍵から元のハッシュ値を算出し、署名を検証する実験を行った。

2.3.4 実装結果

まずは以下のコマンドの通りにテキストファイルを作成する。

```
echo ‘B94D27B9934D3E08A52E52D7DA7DABFAC484EFE37A5380EE9088F7ACE2EFCDE9’
> test.txt
```

後に作成したテキストファイルの文字を秘密鍵で暗号化するため、テキストファイルの文字は電子署名のアルゴリズムに沿って SHA-256 ハッシュ値同様の 256 ビットの 16 進数とした。

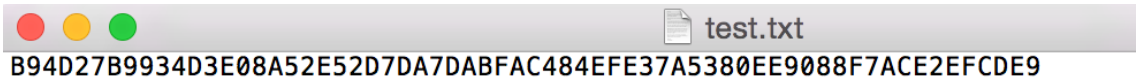


図 7 ：作成したテキストファイル

OpenSC からマイナンバーカードの秘密鍵を利用し、作成したテキストファイルから署名値を算出する。pkcs15-crypt -s で秘密鍵でテキストを暗号化し署名を生成するコマンドとなっている。ここで test.sign ファイルに署名値を書き込んでいる。

```
pkcs15-crypt --pkcs1 -s -R -i test.txt -o test.sign
```

コマンドの引数の意味としては--pkcs1 が pkcs1 の署名方式に従うことを示している。-R は 8 ビットのデータを出力すること。-i test.txt は署名対象のファイルで、-o test.sign が出力する署名ファイルとなっている。

コマンドを入力するとカードリーダから IC カードが読み込まれる。今回はマイナンバーカードの利用者証明用電子証明書にアクセスしようとしているので、利用者証明用電子証明書のパスワードが求められる。

```
Using reader with a card: ACS ACR39 ICC Reader
Enter PIN [User Authentication PIN]:
```

パスワードの入力をすると、設定した出力ファイルが得られる。今回は test.sign と署名値情報のファイルが得られている。sign ファイルは通常通りに開こうとすると文字化けしているため、

hexdump を用いて 16 進数で表示する。

```
hexdump test.sign
0000000 72 7b 3d a1 a4 32 3a c6 9c d9 66 7b c1 8f d6 60
0000010 bb cb f6 ff 10 e9 be d5 27 68 0f e9 02 cb a1 e7
0000020 4a 7e dc 3c 5a 66 fb 32 ab f8 8d ad 27 7e 68 f2
0000030 e7 42 10 43 c0 72 a8 a8 18 49 51 24 a9 94 39 b7
0000040 5b af 99 b3 f6 21 5f 79 d0 07 65 55 cb c6 a7 4c
0000050 e9 d0 e9 13 c9 73 0e a4 9f 97 04 2d 11 9b 76 4d
0000060 e3 d1e3 87 b1 cd 33 6f f4 d3 3d 79 c2 c3 ce 43
0000070 01 52 80 69 b0 6d fe 34 8b 7f 76 53 c4 35 03 2e
0000080 d8 ce cf 1a c2 89 a6 61 46 65 ec 67 19 de eb ca
0000090 89 f3 36 01 b8 3c 90 38 a1 78 43 4e 14 8b d5 f2
00000a0 82 c2 3f ef 12 e2 bb a6 5e c5 14 7c9c b5 22 37
00000b0 19 43 41 29 97 91 3b 65 2e f3 ea f8 41 14 de 54
00000c0 bc 75 fc f1 22 5c b5 2a 26 c1 75 a6 37 f5 63 c5
00000d0 2f 34 46 3e 82 29 26 5d 09 94 f0 9d 4f e7 e5 cc
00000e0 ea 4b a9 3d cd df 2a 6a 74 d2 77 47 0d fb 11 92
00000f0 35 ce 43 6e 3d 36 ab 40 c2 6e 97 6e 4c 21 e8 40
0000100
```

この sign ファイルは pkcs1 形式に基づいて作成されている。

公開鍵と署名値から元のハッシュ値を算出する。そのコマンドを以下に示す。

```
openssl rsautl -verify -in test.sign -pubin -inkey pub_key.pem
```

このコマンドの出力が以下の 16 進数の文字列となる。

```
B94D27B9934D3E08A52E52D7DA7DABFAC484EFE37A5380EE9088F7ACE2EFCDE9
```

この文字列からテキストファイルの文字列と復元した文字列が一致することから署名と復元が正しく行われていることがわかる。なおここで使われている pub_key.pem ファイルはあらかじめ用意しておいた公開鍵情報が書き込まれたファイルである。OpenSC では公開鍵を pem ファイルに容易に書き込むことができる。

```
pkcs15-tool --read-public-key 1 > ./pub_key.pem
```

pem ファイルの中身が以下の通りである。

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAszquDIfV5c+SN8304hH6
j6ksPCPML9MAj418xVSJRGZ+R/hS51KJlhHqlsKqhwfhLjguMQO0v9YsvqG2sQM5
AqxgcIDzKfwbNsHMZdfdR0z6dBk3rkiz/d+MASobC3jQ7jhFgZap+hJbGY0iFJlz
aeNrnlS9PV0glixQPtDe1dW0Zsv8g1zMhjM/LfYqRoJ/zQZa9a6hfBKLjkC1Sz16
OykNbU5E6dQyMgnVqzg5OmOpuv7/sl3nlM7yXsv8hHtUsJGmG30YlItYzJzrcCIX
WXQxkxd3Ngh4Dn5X7b1nc5uLQiZaGFhkzjoLH1fPcXSU2VuZ2rmGDmggH+eTjyE6
9QIDAQAB
-----END PUBLIC KEY-----
```

2.4 マイナンバーカードを用いた電子署名アプリケーションの開発

2.4.1 実装環境

使用した PC とカードリーダは 2.3.2 実験環境の表 1 と同様である。

2.4.2 実装内容

マイナンバーカードを用いてテキストファイルに電子署名をする。

2.4.3 実装結果

実装したシステムはテキストファイルの署名と検証が可能となった。システムは Java で書かれており、GUI は Swing クラスを用いて作成している。また OpenSC は外部コマンドとして呼び出している。

システムを起動するとファイルの選択、署名作成、検証、公開鍵の選択の 4 つのボタンのある画面が起動する。

図 8 ：システム起動例

以下、まずはテキストファイルへの署名への方法を説明する。

はじめに署名するファイルへがどのファイルか選択する必要がある。起動画面の署名ファイル参照ボタンを押すことで、ファイル選択のダイアログボックスが表示される。ファイル選択は PC 上のどのファイルにもアクセスすることができる。

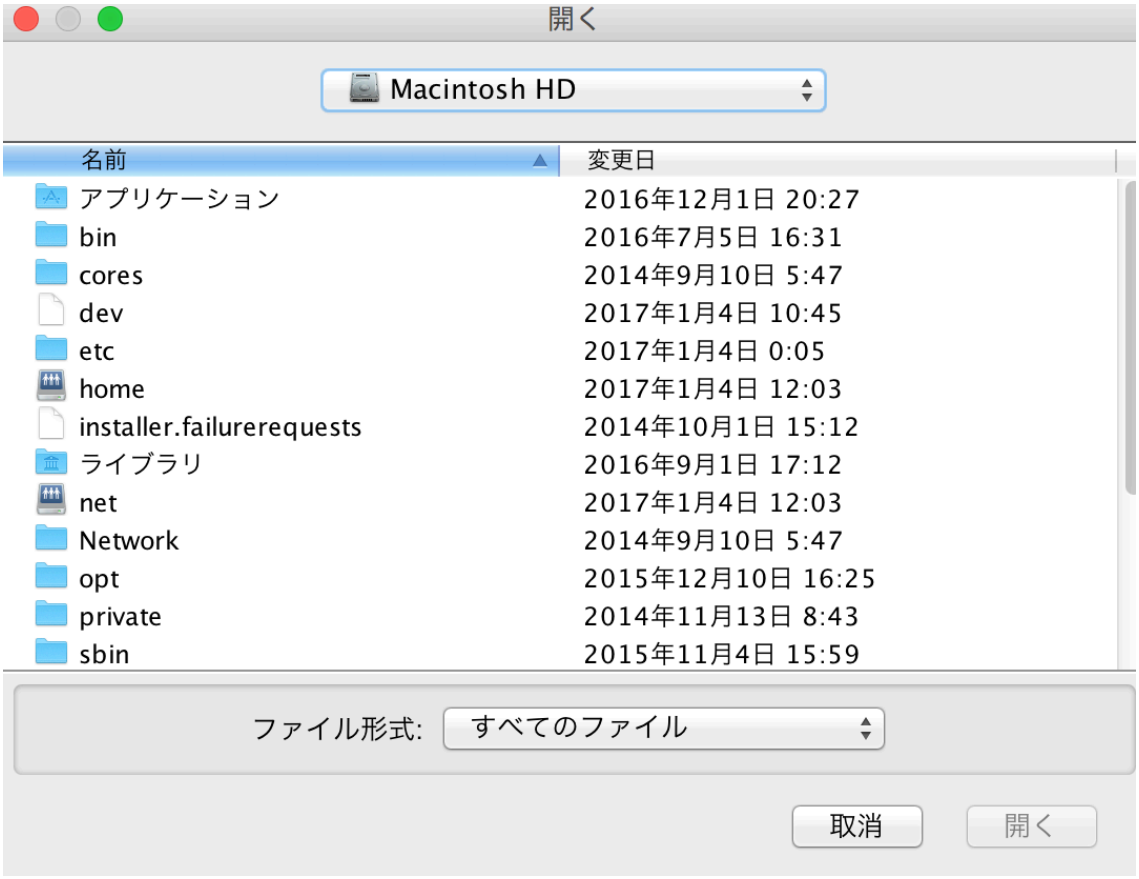


図９：ファイル選択画面

ファイル選択後、起動画面の署名作成ボタンから署名を作成する。なお、ここで例として選んだテキストファイルの画像を示す。

図１０：テキストファイルの例

署名を作成するときに、マイナンバーカードの秘密鍵を利用するためにパスワードが問われる。このパスワードはマイナンバーカード作成時に設定した利用者証明用電子証明書のパスワードである。

パスワード入力するとファイルに署名が追加される。なお、追加される文字列として CERTIFICATE と SIGNATURE の２つがある。CERTIFICATE はマイナンバーカードの公開鍵

証明書を示している。SIGNATURE は署名を表している。署名文と公開鍵証明書は Base64 形式とした。

以下に署名を追加したテキストファイルを示す。また、ここでは公開鍵証明書と署名の文字列を載せている。

```
hogehoge
-----BEGIN CERTIFICATE-----
MIIGHzCCBQegAwIBAgIEAJvehTANBgkqhkiG9w0BAQsFADCBgjELMAkGA1UEBhMCSlAxDTALBgNVBAoMBEpQ
S0kxJTAjBgNVBAsMHEpQS0kgZm9yIHVzZXIgYXV0aGVudGljYXRpb24xPTA7BgNVBAsMNEphcGFuIEFnZW5j
eSBmb3IgTG9jYWwgQXV0aG9yaXR5IEluZm9ybWF0aW9uIFN5c3RlbXMwHhcNMTYwNTIwMTUzNzU4WhcNMjAw
ODA5MTQ1OTU5WjAvMQswCQYDVQQGEwJKUDEgMB4GA1UEAwwXNzE0MzQ0RTA1S0FFSEsxMjIxNzAwM0EwggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCzOq4Mh9Xlz5I3zfTiEfqPqSw8I8wv0wCPjXzFVIlEZn5H
+FLnUomWEeqWwqqHB+EuOC4xA7S/1iy+obaxAzkCrGBwgPMp/Bs2wcxl191HTPp0GTeuSLP934wBKhsLeNDu
OEWBlqn6ElsZjSIUmXNp42ueVL09XSCWLFA+0N7V1bRmy/yDXMyGMz8t9ipGgn/NBlr1rqF8EouOQLVLPXo7
KQ1tTkTp1DIyCdWrODk6Y6m6/v+yXeeUzvJey/yEe1SwkaYbfRiUi1jMnOtwIhdZdDGTF3c2CHgOflftvWdz
m4tCJloYWGTOOgsfV89xdJTZW5nauYYOaCAf55OPITr1AgMBAAGjggLtMIIC6TAOBgNVHQ8BAf8EBAMCB4Aw
EwYDVR0lBAwwCgYIKwYBBQUHAwIwSQYDVR0gAQH/BD8wPTA7BgsqgwiMm1UIBQEDHjAsMCoGCCsGAQUFBwIB
Fh5odHRwOi8vd3d3Lmpwa2kuZ28uanAvY3BzLmh0bWwwgbcGA1UdEgSBrzCBrKSBqTCBpjELMAkGA1UEBhMC
SlAxJzAlBgNVBAoMHuWFrOeahOWAi+S6uuiqjeiovOOCteODvOODk+OCuTE5MDcGA1UECwww5YWs55qE5YCL
5Lq66KqN6Ki844K144O844OT44K55Yip55So6ICF6Ki85piO55SoMTMwMQYDVQQLDCrlnLDmlrnlhazlhbHl
m6PkvZPmg4XloLHjgrfjgrnjg4bjg6DmqZ/mp4swga8GA1UdHwSBpzCBpDCBoaCBnqCBm6SBmDCBlTELMAkG
A1UEBhMCSlAxDTALBgNVBAoMBEpQS0kxJTAjBgNVBAsMHEpQS0kgZm9yIHVzZXIgYXV0aGVudGljYXRpb24x
IDAeBgNVBAsMF0NSTCBEaXN0cmlidXRpb24gUG9pbnRzMRIwEAYDVQQLDAlDaGliYS1rZW4xGjAYBgNVBAMM
EUthc2hpd2Etc2hpIENSTERQMDoGCCsGAQUFBwEBBC4wLDAqBggrBgEFBQcwAYYeaHR0cDovL29jc3BhdXRo
bm9ybS5qcGtpLmdvLmpwMIGvBgNVHSMEgacwgaSAFJVnlRtcpw2EoP/x2FqH8aqxNAOFoYGIpIGFMIGCMQsw
CQYDVQQGEwJKUDENMAsGA1UECgwESlBLSTElMCMGA1UECwwcSlBLSSBmb3IgdXNlciBhdXRoZW50aWNhdGlv
bjE9MDsGA1UECww0SmFwYW4gQWdlbmN5IGZvciBMb2NhbCBBdXRob3JpdHkgSW5mb3JtYXRpb24gU3lzdGVt
c4IBATAdBgNVHQ4EFgQUZEVPM29gJQPnbkGlIPj8BkfkbzswDQYJKoZIhvcNAQELBQADggEBADDkFKo0V4WV
d20/dQMfpeRQTyOACH6ZwQx+8+zscCT8pexybown9hciowaR/xQJlHpO/xkDua9gLfa49y8hTQoiaFS7mhJA
prn7DewvF8d6E5EhC8VBK/8ptHqXp09rTyuxRf1gB/suo5BDmT3p0biMB1HJiV09pEPztuokfTatOA9ZsVQb
A/JHmyFoqBWI7NLxG17o/w8i1yzRZd2My6DaA3Joovo3Vfqn3VQvM7VfPmHPA1B+I35leJZ5PjwiIZd0/pbg
KRIT70BDQvmOMyBTXPp5ka/yFOEDOqkU4fGIe3/E5RDrnWkMMnJrmdRW34mLNhYhU5bQ5x+9L6badro=
-----END CERTIFICATE-----
-----BEGIN SIGNATURE-----
NTBjOTJiZDliMjRlZjdlMWE3ZWE5MmE2MDI4NjYyN2Y3Y2RmOTg2ZGQ2ZjNhZjEzNWNmZTM3ZTI1YjhjNjJj
NWY0ZGJiZTA2ZWIzZjUzZDAyNmNkZDg0NjlmNGI3NmVkOWJiNjI3NzQ1ZWVhYTJmNWIxMzUwNDYwZTkyYWM2
NGU3ZmE4MGQzNjMxMWI3OTBlMWFkNjIxZDk2ZWY1Nzc4MmE5MWUzNzM5YTE3YWRkMzg2MjI0OTUyYmI3ZjVl
NWU0ZmJiM2Q5YTIxNTJjNDU4MzExNWM3ZTAwNTY2OTJiYmVhZTg3ZmJjOGMzYWRlMTdiNzkyYzMxZGVhZWM3
ZjVjYTNjNTViN2RkOTVjYmFhNTc1YjQ4MDVkMDQ0MjllNTgwNGE2MzAyYzExZjk1YzhjYzY5YzlhYzllOGFm
NTRiYzhiYzAzMzkyMjFiMjdiY2Q3YjIzNjAwNDQwZWY2Y2RjY2E5MWM0YzY4ZDA4YzhmNTBmNDJhNzBjZjVh
MTdiY2JkNzE0OTA2Y2RlNTRiYjI4OTMyMDA0Y2I4Njc4YTE5NWZkMzQ1MTBjZjAzYjMzZTA0MTE2MDI5YjUx
MDYzZDE4ZTYyZjMxOGU2ZTYxNDlhZTEyNmE3ZjU0ODJlZmQyYmZjOWVhYTU1YjNkMTE2OTA2MTJiZWY3MTYy
MDk4OTRmMDg=
-----END SIGNATURE-----
```

図１１：署名されたテキストファイル

署名の文字列の生成方法について記す。手続きの流れをの図１２に示す。

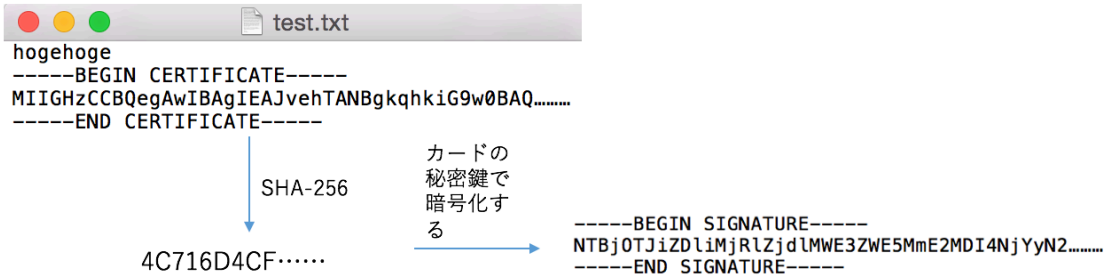


図１２：署名手続きの流れ

署名追加以前にテキストファイルに書かれていた文字列と公開鍵証明書の文字列を結合させたものを SHA-256 にかけてできたハッシュ値を秘密鍵で暗号化して算出された文字列が署名の文字列となっている。

署名の検証について記す。まず署名作成と同様にファイルの選択をする。その次にシステムの起動画面から公開鍵 PK の選択をする必要がある。ここで選択する PK は署名に用いたマイナンバーカードの秘密鍵 SK と対になる PK である。つまり、PK が記されたファイルを所持しておく必要がある。PK が記されたファイルの作成は 2.3.4 実験結果の図ですでに紹介している。公開鍵の選択はテキストファイルの選択同様にダイアログボックスによるもので PC 上のどのファイルでも選択することができる。

PK 選択後に検証ボタンを押すことで署名の検証ができる。署名が正しい場合は「Valid Signature」と記されたメッセージダイアログが、誤っていた場合は「Invalid Signature」と記されたメッセージダイアログが表示される。

検証のシステムについて述べる。手続きの流れを図１３に示す。

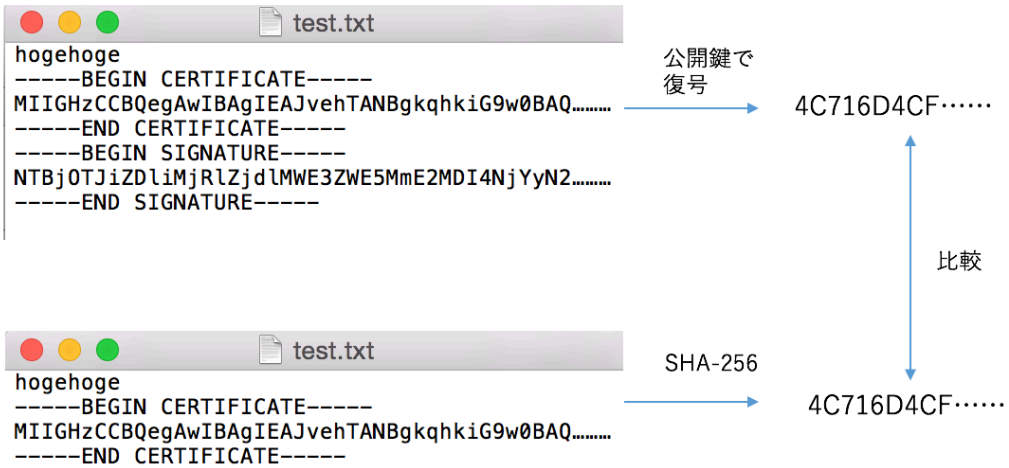


図１３：検証手続きの流れ

検証は署名をマイナンバーカードの PK で復号した値と、テキストファイルの本文および公開鍵証明書の文字列の結合した文字列を SHA-256 にかけて算出された値を比較して行われる。比較して一致した場合が正しい署名、一致しない場合が誤った署名、改ざんを受けた文書、署名者の偽造等を検出したことを表す。

2.2.4 性能評価実験

2.2.4.1 評価実験内容

署名と検証にかかる処理速度をそれぞれ計測した。署名するテキストファイルの文字数 n を 0、10000、20000…50000 文字を 10 回ずつ計測。また、署名するテキストファイルのサイズを 10^{-1} MB, 10^{0}MB, 10^{1} MB, 10^{2} MB と変更し，10 回ずつ計測した。

2.2.4.2 評価実験結果

署名、検証の処理時間は文字数にあまり依存しないことがわかる。また、ハッシュにかかる時間が 10000 文字につき 10ms、署名にかかる時間が 62.5ms であることが算出された。署名と検証にかかる処理速度を以下の図１４、図１５に示す。

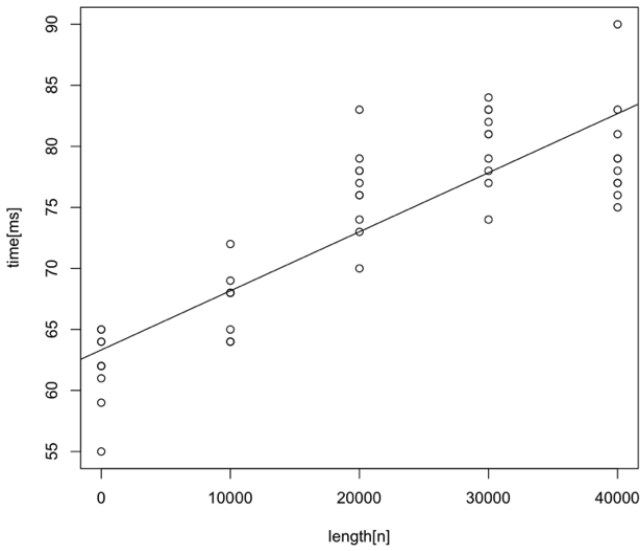


図１４：署名の処理速度

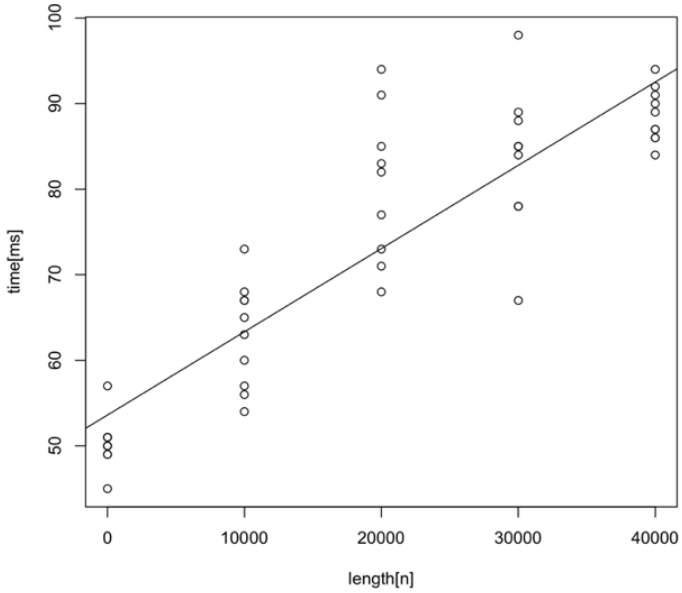


図１５：検証の処理速度

ファイルサイズと処理時間はほぼ比例の関係にあることがわかった。また、10^3MB の計測をしようとしたところ、OutOfMemoryError が発生した。Java では大きなサイズのファイルを開く事ができないことができないことが原因だと考えられる。計測の結果を以下の図１６に示す。

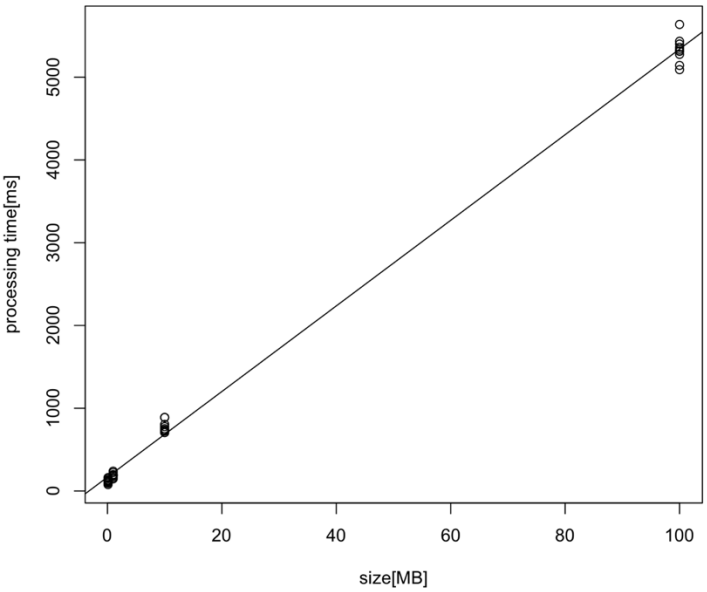


図１６：署名の処理速度

従来の署名ツールとして代表的な Acrobat との性能比較を行った。署名者情報やタイムスタンプ情報、署名できるファイルは従来の Acrobat の方が高性能である。しかし、秘密鍵の保管場所がマイナンバーカードという点で安全性は高い。Acrobat で作成した秘密鍵は、/Users/ユーザ名/Library/Application Support/Adobe/Acrobat/DC/Security/に p12 ファイルとして保管される。また、自治体から認証された本人認証のされた署名となる。性能比較を以下の表にまとめる。

表３：従来システムとの比較

	秘密鍵の保管場所	署名者情報	タイムスタンプ	署名できるファイル
Acrobat	PC 上	有	有	PDF
Captain Signer	マイナンバーカード	有	無	テキストファイル

3 おわりに

本研究ではマイナンバーカードを用いた電子署名システムを試験実装した。従来のファイル上の秘密鍵を利用する場合と比べると、秘密鍵が漏洩する脅威をなくすことに成功した。

しかし、署名者情報やタイムスタンプ等の情報を提示できていないことや署名対象のファイルがテキストファイルのみであるため、マイナンバーカードを用いるメリットを生かしきれていない。これらを今後の課題とする。

付録 A
検索可能秘密分散システムの実装

4 はじめに

4.1 研究背景

2014 年に起きたベネッセコーポレーションによる個人情報流出では、クラウドに保管されたデータがクラウドの管理者によって約 2895 万件の個人情報が漏洩した.このような内部犯行クラウドを利用の大きな脅威である。

4.2 研究目的

この課題に対して、サーバ管理者がサーバに保管したデータを見ることのできないような検索可能秘密分散を提案している論文が存在する。本研究ではに基づき検索可能秘密分散システムの実装を試みる。このシステムが実用されていれば、ベネッセの様な個人情報流出は防止できると期待する。

4.3 秘密分散とは

秘密の情報をシェアと呼ばれるいくつかの分散情報に分割して、秘密に関する情報を守る技術である。そしてこのシェアを指定した閾値以上集めることで秘密を復元することができる。

例として秘密 s = 1 として秘密分散の例を挙げる。

まず f(0) = s = 1 となる k-1 次式の多項式を生成する。ここでの k は秘密を復元するために必要となるシェアの閾値を表す。次に秘密 s やシェア分散数 n よりも大きな素数 p を生成する。次に $1 \leqq i \leqq n$ としてシェア $v_i = f(i) \bmod p$ を計算する。これらの順序を経て分散過程が終了する。

復元過程ではラグランジュの補間公式を用いる。ラグランジュの補完公式とは k-1 次の任意の多項式は(i, f(i))の組みが k 個以上あることで、元の多項式 f(x)を復元できる公式である。公式は

$$f(x) = \lambda_1(x)f(i_1) + \cdots\cdots + \lambda_k(x)f(i_k) \bmod p$$

ただし

$$\lambda j(x) = \frac{(x - i_1) \ldots \overset{j}{\vee} \ldots (x - i_k)}{(i_j - i_1) \ldots \overset{j}{\vee} \ldots (i_j - i_k)} \bmod p$$

多項式を復元した後に f(0) = s を求めることで秘密を復元することができる。

秘密分散のメリットは、秘密のデータを扱う人も分散を行うことで秘密の情報に関する情報を得ることができなくなるため、安全性が高まることにある。またシェアのいくつかを紛失しても閾値以上のシェアが存在していれば秘密を復元することができる。

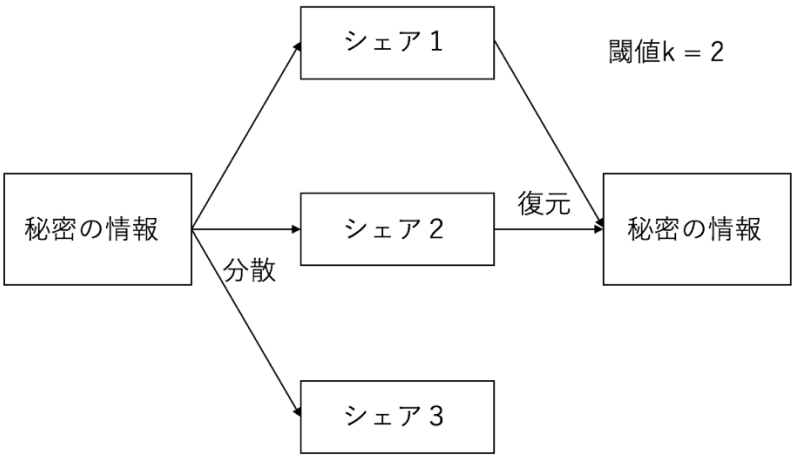


図１７：秘密分散

5 実装

5.1 画像の秘密分散

5.1.1 実装目的

通常秘密分散は文字列に対して使うが、画像のような複雑な情報に対しても秘密分散を適応させることができるかを試行。

5.1.2 実装環境

表 4 ：PC の実装環境

言語	Java
OS	OSX
メモリ	8GB

5.1.3 実装内容

画像の RGB 情報を文字列として読み込み、秘密分散にかけシェアに分割する。また分割したシェアから元の画像を復元する。

2.1.4 実装結果

まず secretRed[y]、secretGreen[y]、secretBlue[y]といった 3 つの配列をあらかじめ作成しておく、この配列に画像の RGB 情報をそれぞれ格納していく。また y は［0 ≦ y ≦ 画像の高さ］である。

画像すべてのピクセルの RGB 情報を 1 つの秘密とするとその桁数は［画像の幅 × 画像の高さ × 3］となり計算にとても時間を要するので、ここでは高さごとに秘密を分割して計算することとした。ただし、この場合でも 1 つの秘密の桁数は［画像の幅 × 3］となり、これは int や long では収まらないので BigInteger クラスを用いた。

実際の画像と秘密の数値例として、例となる図１８を示す。

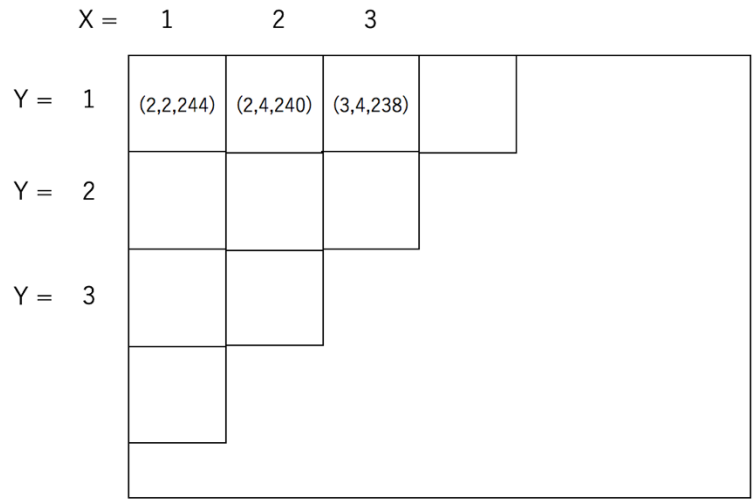


図１８：画像の例

pixcel(1,1)の RGB が(2, 2, 244)、pixcel(2,1)が(2, 4, 240)、pixcel(3, 1)が(3, 4, 238)なので、y=1 の R 情報を文字列結合して m1=secretRed[1] = 002002003…となる。同様に secretGreen[1] = 002004004…、secretBlue[1] = 244240238…である。この 3 つの配列の数値を分散し、それぞれシェアに分けていく。つまり、シェアの数を 5 とすると shareRed_1[] … shareRed_5[]などのように 15 の配列が必要となる。当然、shareRed_1[1]には secretRed[1]の秘密を分散させた 1 つ目のシェアの値を格納する。ここで、分散時に用いられる素数 p は[p の桁数]<[画像の幅×3]である必要があることに注意が必要である。

閾値を 3 として、復元をする場合 shareRed_1[1]、shareRed_2[1]、shareRed_3[1]のように配列の番号が同じ 3 つの配列を用いることで secretRed[1]の値を算出することができる。

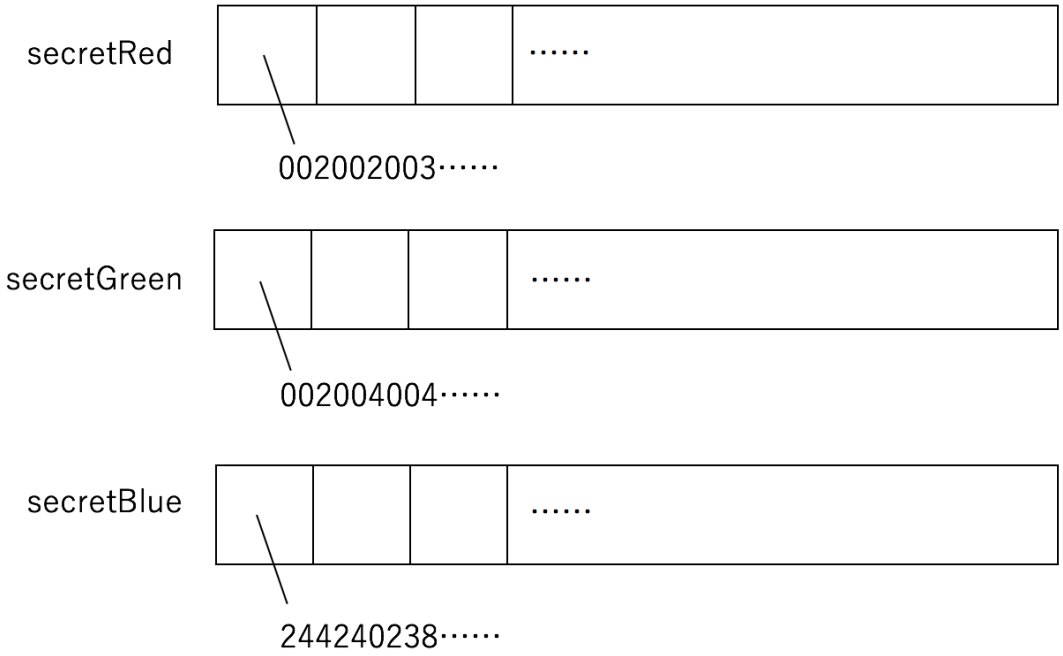


図１９：配列の例

また、どの画像を秘密分散にかけるかはプログラム内で決まっている。画像の変更するには画像の幅と高さが可変となるため、それに合わせた配列を用意する必要がある。これら実装は今後の課題である。

5.2 検索可能秘密分散

5.2.2 実装内容

従来のクラウドは秘密の情報を１つのサーバに保管していたが、本実装は複数のサーバに１つの秘密の情報を保管するシステムとする。データオーナーが秘密の情報を秘密分散を用いてシェアｓに分割し、シェアを複数のサーバに保管するものとする。また、この時、データを取り出す時に必要となるタグｔをシェアとペアにしてサーバに格納する。またタグはキーｋから生成する。

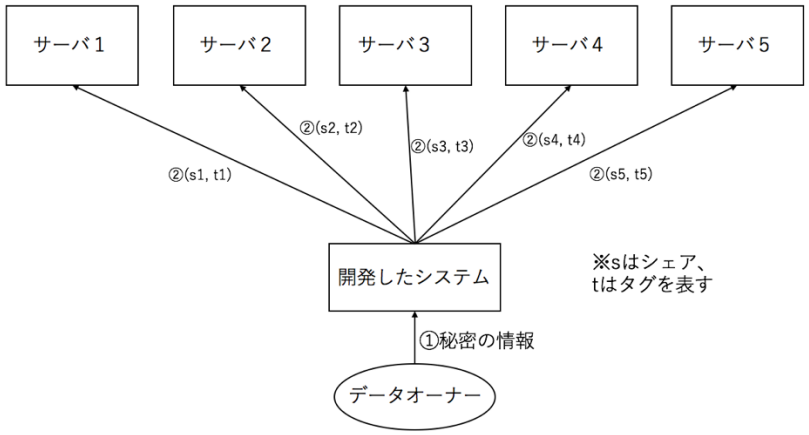


図２０：分散手続きの流れ

また、データユーザはクラウドにタグを用いて検索をかける。格納されたタグとユーザの入力するタグが一致する場合シェアを得ることができ、閾値以上のシェアを得ることで秘密を復号できるものとする。

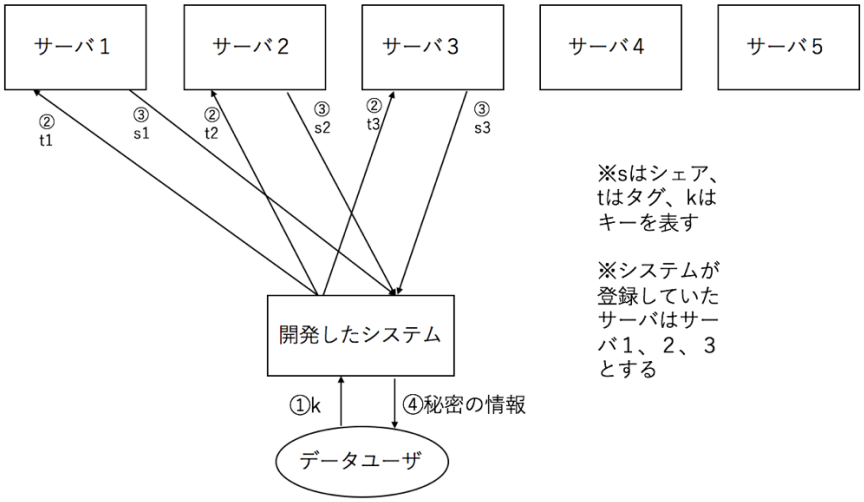


図２１：復元手続きの流れ

システムの詳しいアルゴリズムとして、伊藤らが、サーバ管理者がサーバに保管したデータを見ることのできないような検索可能秘密分散[5]を提案している。これに基づく検索可能秘密分散システムの実装を試みる。

5.2.3 実装結果

システムにはシェア、タグ、キーと役割を持った値を作成した。はじめに、シェアは秘密の情報を秘密分散にかけて算出した値である。次にタグはシェアとペアにしてデータベースに格納する値である。データユーザはこのタグをシステムに検索をかけることでシェアを得ることができるものとする。最後にキーはタグを生成するための値である。キーはデータベースにおけるプラマリーキーのようなもので重複は許されない。本実装では秘密の情報を仮想の個人情報としたので、個人の名前をキーとした。キーはタグを生成することに利用する。キーとアップロードするサーバのユーザ名、ホスト名を文字列結合して SHA-256 にかけたものをタグとした。

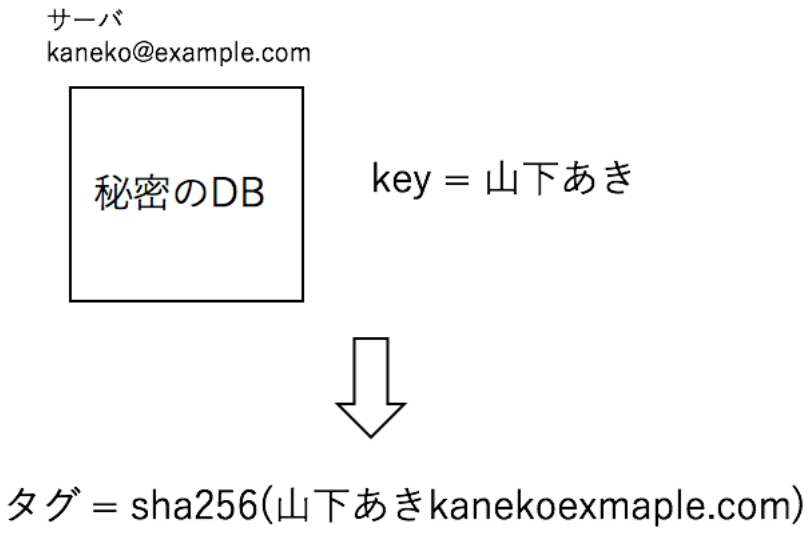


図２２：タグの生成

表５：秘密の情報となるデータベース

	name	phonetic	email	sex	age	birthday	marriage	blood	prefectures
1	山下 あき	やました あき	yamashita_aki@example.com	女	79	1936/8/2	既婚	A型	富山県
2	水口 早紀	みずぐち さき	mizuguchi_saki@example.com	女	75	1940/9/16	既婚	B型	宮城県
3	大泉 努	おおいずみ つとむ	ooizumi_tsutomu@example.com	男	65	1950/9/2	既婚	A型	神奈川県
4	玉田 洋介	たまだ ようすけ	tamada_yousuke@example.com	男	25	1990/2/10	既婚	B型	神奈川県
5	坂本 哲平	さかもと てっぺい	sakamoto_teppei@example.com	男	29	1986/8/7	未婚	B型	栃木県
6	井田 敏也	いだ としや	ida_toshiya@example.com	男	46	1969/5/12	既婚	A型	埼玉県
7	羽田 奈月	はた なつき	hata_natsuki@example.com	女	26	1989/9/11	未婚	B型	広島県
8	柳原 真一	やなぎはら しんいち	yanagihara_shinichi@example.com	男	74	1941/4/17	既婚	O型	兵庫県
9	大貫 夏空	おおぬき そら	oonuki_sora@example.com	女	41	1974/9/22	既婚	A型	大阪府
10	伴 友香	ばん ともか	ban_tomoka@example.com	女	31	1984/7/29	既婚	A型	埼玉県
11	今川 昌代	いまがわ まさよ	imagawa_masayo@example.com	女	52	1963/4/23	既婚	B型	群馬県
12	桑田 広之	くわた ひろゆき	kuwata_hiroyuki@example.com	男	76	1939/1/16	既婚	B型	福岡県
13	笹原 優	ささはら ゆう	sasahara_yuu@example.com	女	25	1989/12/27	未婚	A型	熊本県
14	坂元 法子	さかもと のりこ	sakamoto_noriko@example.com	女	39	1975/12/15	既婚	B型	大阪府
15	瀬戸内 りえ	せとうち りえ	setouchi_rie@example.com	女	78	1936/11/23	既婚	O型	新潟県
16	武藤 美幸	むとう みゆき	mutou_miyuki@example.com	女	35	1979/11/22	既婚	A型	静岡県
17	松井 ひろみ	まつい ひろみ	matsui_hiromi@example.com	女	79	1936/4/16	既婚	O型	岐阜県
18	小田 瞬	おだ しゅん	oda_shun@example.com	男	26	1989/1/23	既婚	O型	山口県
19	奥村 そら	おくむら そら	okumura_sora@example.com	女	37	1978/6/8	既婚	A型	埼玉県
20	河野 彩華	かわの あやか	kawano_ayaka@example.com	女	58	1957/3/4	既婚	B型	奈良県
21	田島 さとみ	たじま さとみ	tajima_satomi@example.com	女	60	1955/7/19	既婚	A型	福島県
22	伊丹 倫子	いたみ のりこ	itami_noriko@example.com	女	52	1963/3/24	既婚	O型	神奈川県

以下システムの概要について説明していく。はじめに秘密のデータベースを秘密分散にかけシェアを生成。分散数を５、復元に必要な閾値を３とする。また、シェアとペアとなるタグを生成。５つのシェアデータベースをサーバにアップロード。アップロードされるデータベースの例を図に示す。列に追加されている tag がタグである。

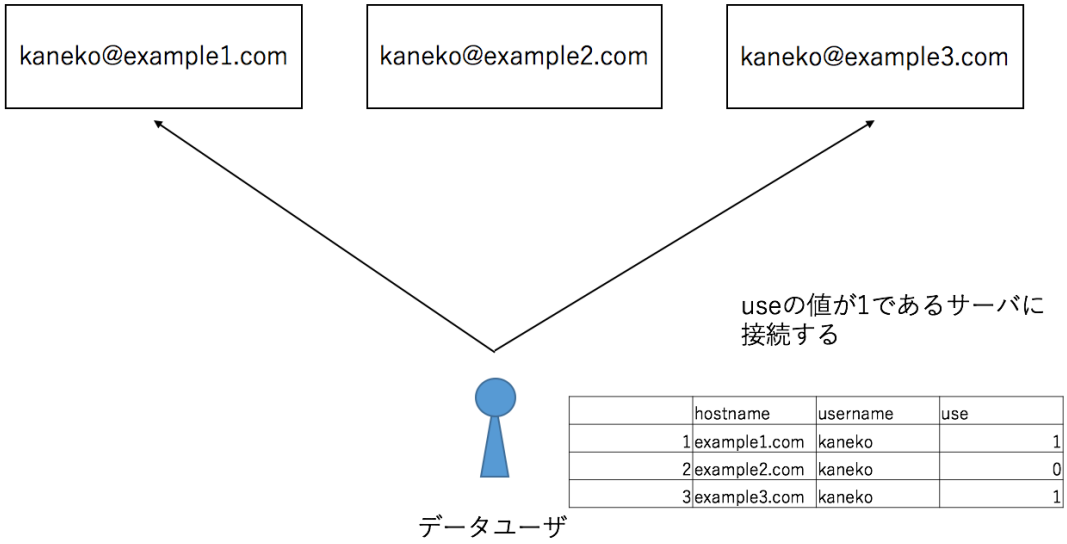
表６：アップロードされるデータベースの例

	tag	name	phonetic	email	sex	age	birthday	marriage	blood	prefecture
1	1481212…	522714778…	8969558…	3442949…	484114…	13268533…	30404…	3863050…	149559…	46323…
2	1204809…	469799474…	3445050…	1583457…	392646…	17552812…	49289…	3584744…	432529…	39086…
3	2207602…	489714945…	3202624…	1057548…	266536…	43499731…	27754…	3863376…	181231…	30022…
4	2354809…	430112638…	1971609…	3557963…	292656…	33123746…	28650…	2383771…	360838…	50930…
5	1233403…	353825486…	4016042…	1817821…	113590…	11673517…	27410…	3803829…	420833…	23342…
6	2452421…	452319373…	5044776…	5938135…	384989…	32692655…	35515…	2263772…	542286…	16267…
7	1342426…	426422857…	3883399…	2628571…	249752…	54625914…	47548…	3748148…	271562…	20906…
8	3701211…	320560807…	4725545…	1210000…	401372…	53829299…	54407…	4880492…	480781…	47961…
9	2292082…	461389733…	2499552…	3209431…	443569…	45103761…	51331…	2715746…	209613…	45259…
10	5406701…	280936543…	2930480…	3758868…	420910…	40511310…	19112…	4254371…	628182…	48195…
11	2001688…	429346240…	3211588…	8741755…	236212…	31107058…	48332…	4662602…	211341…	45986…
12	2232012…	321200785…	4900223…	1919378…	254220…	66038208…	46543…	2159943…	306018…	25667…
13	1655002…	250610852…	2850978…	4072031…	181182…	56840301…	30067…	2978034…	289908…	42470…
14	1301233…	343066630…	1297523…	3343524…	554278…	50725486…	29222…	2931057…	405776…	52783…
15	1721506…	755064426…	3081404…	5078062…	122482…	51211628…	57845…	2598418…	295456…	43679…
16	2431571…	212196381…	1248922…	3785108…	390866…	45194236…	44077…	1272626…	443406…	36489…
17	1104504…	242402928…	1698206…	6345409…	368938…	49292702…	49347…	3412581…	452070…	12093…
18	1413801…	353691888…	2738975…	3599747…	406826…	35278626…	28205…	3105611…	435475…	29884…
19	1051137…	483570667…	3498378…	4806795…	560734…	31495722…	15231…	1623371…	364016…	33036…
20	2061191…	495837114…	1895587…	1919940…	406587…	59045164…	56696…	4156207…	215455…	94432…
21	1082513…	269577929…	1647258…	4154699…	200832…	10098062…	33963…	3352411…	278922…	11536…
22	2451529…	323628860…	3737531…	3240379…	335783…	26312940…	31102…	2945760…	586976…	51004…

秘密分散の方法としては文字列を 1 文字すつに分割して文字コードとして数値化して分散を行っている。「なかの」という文字列であれば「な」は 12394、「か」は 12363、「の」は 12398 であるので 123941236312398 が秘密の情報とされ分散が行われている。また、分散に用いられる素数は十分に大きい定数である。

アップロードしたシェアは、キーをデータユーザが入力することで得ることができる。

またアクセスするサーバはあらかじめ登録しておく必要がある。そのためサーバのアクセス先一覧が格納されているデータベースを作成した。データベースにはサーバのホスト名とユーザ名、使用状態のカラムが存在する。使用状態の型は integer としている。しかし値は 0 か 1 しかとらず、その働きは boolean と同様である。使用状態が 0 のときはサーバにつながず、1 のときはサーバにつなぐ処理を行う。システムの分散過程の時に使用状態が 1 となっているサーバが 4 つあるとするとシェアは 4 つになる。また復号過程の時に使用状態が 1 となっているサーバが 2 つだとすると、その 2 つのサーバのシェアを使用する働きをする。また復号における閾値はプログラム内で決まっている。



	hostname	username	use
1	example1.com	kaneko	1
2	example2.com	kaneko	0
3	example3.com	kaneko	1

図２３：サーバへの接続

入力は SQL の構文をベースとして実装した。以下に実装したコマンドの一覧を示す。

表 7 ：コマンド一覧

1	REGISTRATION SERVER [サーバ登録名]	サーバ登録は ssh の接続に成功した場合にのみローカルのサーバ登録データベースに新規登録される。
2	USE SERVER [サーバ登録名]	サーバ登録データベースに入力したサーバ名が存在する場合のみ実行する。サーバ登録データベースの使用状況を使用状態変更する。
3	DISUSE SERVER [サーバ登録名]	サーバ登録データベースに入力したサーバ名が存在する場合のみ実行する。サーバ登録データベースの使用していない状況を使用状態変更する。
4	DISPERSION DATA [秘密]	秘密をローカルで分散し、ssh を用いて現在使用状況の各サーバに接続。 接続先のデータベースにシェアとタグを格納する。タグは「秘密」「サーバホスト名」「サーバユーザ名」の 3 つを文字列結合し、SHA-256 を用いてハッシュした値とする
5	SELECT KEY = [キー]	キーからタグを生成して、ssh を用いて現在使用状況の各サーバにタグを送信。シェアを格納したデータベースからシェアを得る。 得たシェアから秘密を復号する。

実際にシステムを使用する際の手順を記す。

分散手続き

1, 使用するサーバが未登録の場合、表の 1 . REGISTRATION SERVER [サーバ登録名] を使用しサーバを登録する。

2, 表の 2 . USE SERVER [サーバ登録名] を使用し登録しているサーバからサーバを使用状態へ変更する。

3, 表の 4 . DISPERSION DATA [秘密] を使用し秘密を分散する。

復元手続き

1, 使用するサーバが未登録の場合、表の 1 . REGISTRATION SERVER [サーバ登録名] を使用しサーバを登録する。

2, 表の 2 . USE SERVER [サーバ登録名] を使用し登録しているサーバからサーバを使用状態へ

変更する。

3, 表の５. SELECT KEY = [キー] を使用し秘密を復号する。

これらの手順でシステムを使用する。また秘密の復号はプログラム内で行い、シェアはユーザには見られないようにしている。

5.2.3 評価

復号が正確であるかどうか、全シェアから秘密のデータを復号し、元となる秘密のデータと比較するプログラムを作成したところ、全ての秘密が正しく復号できていることが確認できた。よって正確な分散と復号が行われていることがわかる。

シェアを観測して、シェアから秘密の情報について判別できることがあるか検討する。下図は血液型情報のシェアである。血液型はＡ型、Ｂ型、ＡＢ型、Ｏ型の４種類のみで、文字コードは１文字につき６桁であるので下１２～１８桁が秘密の情報となっている。しかし下図を見る限り全てのシェアの値が異なっているので、シェアから秘密の情報は漏れていないことがわかる。同じ秘密から異なるシェアの値が算出される理由は、分散に用いる多項式の係数が乱数であることによるものだ。よって安全性は高い。

blood
348185454498935146337379445707614880332793770330633972718875527733776569824321020754469
000135531251618557679950515599731102353114123667339661362502682573759170300431348262118
938042486903740138778352071038868669466278041881206745374817925013081647646923475153561
544224609261309926676621106473571787935686326837102843752744766296896321290919332121316
922200955126339081649271300327074306640285694431678840497540399748378804800883363687459
479945149835865402700209017904014024399507748642807238884697798918600079717719415393719
964832435043916821469195872411736539013663630066709761201610237390999912793495227501005
773775910690513866723398798571025713832768038847567789723385958364330298826885328794686
685031100566228218266996897418513822400800303828553178340849946742933645390181632146879
912869067737138302312196274259672535211449139722545686839797064897795912310222370211401
620789147207524042058068583432627678425513026502671543796343040301716073248115792071577
942709875590941694732580277543886507380148814850355272810281440758229620377468503325057
317811768649938312990299190670014097632398952258684848933924486318604198898381169755721
057056375363321784440215047210991574513904963161619843471021805527311708128836527647131
685399862123540908902768818124754130228249687822681479489200155809193506521919340416128
107147307223769081159614705423070328897633712103515777067004264804732726722119174838332
804972756874133612019464231993024639884443021342124897974679308041077445169969830781847
460777594553673842191427047666973179537305134586709422948222146029906131504483876770616
367773258741316941103498113147049851430244122103988111982409873306920946916619100725732
216800086154774009903724481512242444961738056578868606973744124609536780094436978763702
335593794242957847835998711198207666972438492398297730483085240401852626650673589896824
077537612577560943660028947177037182360262308376724356165346637648271861707023378702383

図２４：血液型シェア情報

開発したシステムで１文字あたりの平均復元時間を示す。１文字あたりの平均が0.1163668$\times 10^{-1}$msであるので、１秒あたりに約86000字の復元ができる。また、それぞれのしきい値kを変更し、9000回の処理時間を求め、下に回帰直線を示す。

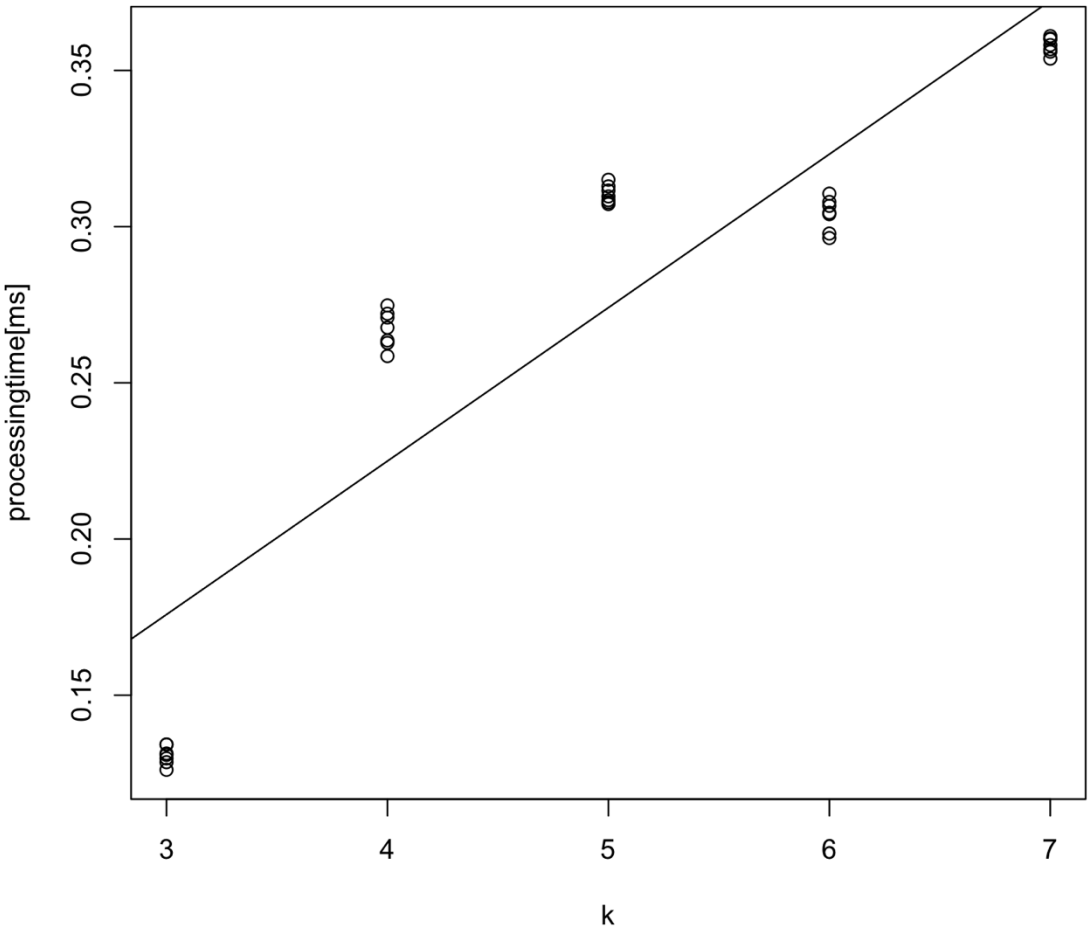


図２５：処理時間の回帰直線

6 まとめ

本研究では、検索可能秘密分散システムを実装した。各シェアから秘密について情報についての情報は漏れることはない。SQL ベースのコマンドであることから慣れ親しみやすいと考えられる。また、その処理時間から実用性が高いと考える。

しかし、開発したシステムの秘密分散するデータの大きさは素数 p に依存する。データの大きさによって p を変更するなど、更なる工夫が必要である。

参考文献

[1]黒澤馨, 尾形わかは ： 現代暗号の基礎数理, pp.106-107, pp116-119, 2004 年

[2] 「マイナンバー詐欺」被害, 読売新聞 2015 年 10 月 07 日

[3] 法務省電子署名法の概要と認定制度について(http://www.moj.go.jp/MINJI/minji32.html)

[4] OpenSC(https://github.com/OpenSC/OpenSC)

[5] 伊藤, 牛田, 山岡, 及川, 菊池 ： 検索可能秘密分散方式の提案, 情報処理学会研究報告, p1-3, 2012 年