Vol.2015-CSEC-70 No.26
Vol.2015-SPT-14 No.26
2015/7/3

SHSS: オブジェクトストレージ向けの超高速秘密分散ライブラリ

五十嵐 大[1,a)]　露崎 浩太[2]　川原 祐人[1]

概要：オンラインストレージにおけるセキュリティ対策や激甚災害対策として近年, 秘密分散技術の研究が活発化している。一方, ストレージ分野では近年, データ消失への対策として消失訂正符号が活発に研究・実用化されてきている. 本稿では秘密分散に, 近年普及が進むオブジェクトストレージシステムにおいて消失訂正符号の上位互換としてセキュリティ面のメリットがあることを指摘し, オブジェクトストレージ向けの高速秘密分散法を提案する. さらに, SHSS(Super High-speed / Sugoku Hayai Secret Sharing) と呼ぶライブラリを実装し, オブジェクトストレージに適用可能な従来技術の約 50 倍速, また標準的なオブジェクトストレージ OpenStack Swift 組み込み時, 現在標準的な消失訂正符号ライブラリと同等の性能である, 分散/復元 10Gbps 程度の性能を実現したことを報告する.

キーワード：秘密分散, 消失訂正符号, オブジェクトストレージ

SHSS: "Super High-speed (or, Sugoku Hayai) Secret Sharing" Library for Object Storage Systems

Dai Ikarashi[1,a)]　Kota Tsuyuzaki[2]　Yuto Kawahara[1]

Abstract: Recently, as a measure for the information security and the disaster recovery regarding on-line storage systems, the research of *secret sharing* technology has become quite active. On the other hand, in the research field of storages, *erasure codes* has been widely studied and quickly spread over practical storage systems recently. In this paper, we point out that secret sharing has a merit from the aspect of information security as an upward compatible function of erasure codes when it is applied for object storage systems, which are becoming popular today, and propose an efficient secret sharing scheme suitable for object storage systems. Furthermore, we implemented a secret sharing library called SHSS (Super High-speed / Sugoku Hayai Secret Sharing), and report it's performance. It is about 50 times faster itself than that in the existing report for object storage systems, and combined with OpenStack Swift, it performs about 10 Gbps, which is as the same level as the standard erasure code library without security.

Keywords: secret sharing, erasure code, object storage

1. はじめに

1.1 (k, n)-秘密分散

オンラインストレージにおけるセキュリティ対策や激甚災害対策として近年, 秘密分散技術の研究が活発化している [11], [19], [22], [23], [24], [25]. (k, n)-秘密分散とは, データから n 個の断片を生成し, そのうち任意の k 個の断片が揃えば復元できるという可用性と, 不正者が $k-1$ 個以下の断片を手に入れても一切の情報を得られないという秘匿性 [*1] を併せ持つ技術である. 可用性に関しては, n 個の断片をそれぞれ遠隔拠点に配置することで, $m = n - k$ 以下の拠点が激甚災害により動作を停止してもデータの保護およびサービスの継続が可能となる. 秘匿性に関しても, 互

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories, Midoricho 3-9-11, Musashino, Tokyo 180–8585, Japan
[2] NTT ソフトウェアイノベーションセンタ
NTT Software Innovation Center, Midoricho 3-9-11, Musashino, Tokyo 180–8585, Japan
[a)] ikarashi.dai@lab.ntt.co.jp

[*1] 改ざんを防止する完全性を併せ持つ方式もある. [12], [20]

いに管理者の異なる拠点に断片を配置することで内部不正やマルウェア感染による情報漏えいを防ぐことができる.

最も代表的な手法は Shamir の秘密分散である [18].

1.2 消失訂正符号

一方ストレージ分野では近年, データ消失への対策として消失訂正符号が活発に研究・実用化されてきている. [3], [4], [5], [8], [9], [14] 消失訂正符号は (k, n)-秘密分散に似ているが, データを k 個の断片 (本稿ではデータシェアと呼ぶ) に分割し, m 個のパリティを生成して計 n 個の断片とする. データシェアとパリティのうち合わせて k 個以上が揃えば復元できる. ただし秘匿性は考慮されていない. 例えばデータシェアは普通, 入力データを単純に k 個に分割しただけのものである.

消失訂正符号は可用性と, **堅牢性**を高める技術として注目されている. 堅牢性とはストレージ分野の信頼性指標のひとつで, 多数のディスクに大量のデータを保管してディスクの破損が無視できない頻度で発生する環境で, “データが失われない確からしさ” を示す. (可用性と異なり, サービスが一時停止することは許容することに注意.) 従来は堅牢性を高めるために複製や RAID などで対策がされてきた. しかし, 複製では n 個に複製した場合データ量が n 倍に増大してしまう, RAID では十分な堅牢性が得られないという課題があった. 対して消失訂正符号では,

(1) データ量は n/k 倍で済む
(2) 任意に高い堅牢性を設定できる

という特長がある. これまで消失訂正符号は計算コストのためストレージ分野での幅広い普及はされてこなかったが, 近年 1CPU あたり 10Gbps 級の実装 [3], [4], [14] が出始めたことにより, 急速に普及し始めている.

1.3 単一拠点における秘密分散のメリット

1.1 節の通り, これまで秘密分散は技術の持つ可用性・秘匿性を最大限活かすため, 複数拠点への分散で議論されることがほとんどだった. しかし秘密分散には, 単一拠点においても下記の 3 つのメリットがある.

(1) 堅牢性: 単一拠点で激甚災害対策とならなくても, 複数ディスクに断片を分散することで消失訂正符号と同等の堅牢性向上効果がある.
(2) ディスク廃棄時の情報漏えい対策: 日々一部のディスクが破損するような, 多数のディスクを扱う状況では通常, ディスクに粉砕などの情報漏えい対策を施して廃棄する必要がある. 暗号化や秘密分散でディスクだけではデータが読めないような対策がしてあれば, このような処理を低減することができる.
(3) 鍵管理不要: 秘密分散では暗号化と異なり暗号鍵が必要ないため, 鍵管理が不要である.

1.4 計算量的秘密分散

計算量的秘密分散はショート秘密分散とも呼ばれ, それまでの秘密分散でデータ容量が n 倍となってしまうことを解決するため, 1993 年に Krawczyk が提案した [10]. 計算量的秘密分散は消失訂正符号と同等のデータ容量と可用性・堅牢性を実現し, さらに秘匿性を持ち, 事実上消失訂正符号の上位互換技術である. しかし消失訂正符号で CPU あたり 10Gbps 級の実装が現れているのに対し, 秘密分散は数 Gbps 程度までであった [22], [23]. 特に**オブジェクトストレージ**分野で求められる 11-nines ($= 1 - 10^{-11}$) といった堅牢性のパラメータでは数百 Mbps が最速であった [16].

1.5 本稿の貢献

本稿では計算量的秘密分散における性能の課題を解決するため, 高速な計算量的秘密分散を提案し, さらに SHSS(Super High-speed / Sugoku Hayai Secret Sharing) と呼ぶライブラリとして提案方式を実装し, 20Gbps 程度の性能を報告する. また標準的なオブジェクトストレージソフトウェアの一つである OpenStack Swift [6] 組み込み時, 現在標準的な消失訂正符号ライブラリと同等の性能である, 分散/復元 10Gbps 程度の性能を実現したことを報告する.

2. 関連研究

2.1 秘密分散

秘密分散の性能向上に関しては, 特に国内で研究が進んでいる. 保坂ら [25] と栗原ら [11] は XOR ベースの効率的な秘密分散を提案した. さらに須賀は k, n が限定されるという, 栗原らの手法の制限を解決した [24]. また松尾らは $k = 5, n = 6$ の XOR ベースの秘密分散で, 分散 1.4Gbps, 復元 1.5Gbps を実現した [23]. さらに五十嵐らは素体および拡大体を用い, $k = 2, n = 3$ の情報理論的秘密分散で分散 4.3 Gbps / 復元 7.4 Gbps [22], 計算量的秘密分散で分散 / 復元とも 4.7 Gbps を実現した.

しかしこれらの性能は堅牢性を考慮しない k, n のもとであり, オブジェクトストレージ分野で望まれる堅牢性を確保した場合には性能が低下すると考えられる. なぜならオブジェクトストレージ分野で多く採用される 99.999999999%の堅牢性の場合, $m(= n - k)$ は実際 3 以上となるが, 一方秘密分散の計算量は m に比例してしまうため, $m = 1$ となる $k = 5, n = 6$ や $k = 2, n = 3$ よりも大幅に性能が低下すると考えられるからである.

2.2 消失訂正符号

消失訂正符号に関しては Plank らが 10 Gbps を超える高速な実装を報告しており [14], また実用面でもオープンソースで, Jerasure [4], ISA-L [3] などのやはり CPU あたり 10 Gbps 以上の性能を実現するようなライブラリが現れてきている.

3. 準備

3.1 記法

(1) n: 全断片数.

(2) k: 復元に必要な断片数. 消失訂正符号においてはデータシェア数と一致.
(3) m: $n-k$. 消失訂正符号においてはパリティ数と一致.
(4) f: GF(2^{64}) における既約多項式 (3.2 節)

3.2 GF(2^{64})

本稿では 2 の拡大体を利用する. 特に既約多項式を $f[X] = X^{64} + X^4 + X^3 + X + 1$ とする拡大体 GF(2^{64}) である.

GF(2^{64}) は mod 2 の係数をとる多項式を既約多項式 f で多項式除算した余りの集合であり, 四則演算を行うことができるという特長がある. 特殊な四則演算をもつ, ビットの 64 次ベクトルと思ってもよい. 64bit 整数で表現でき, 項 x^i を 2^i で表現する. 例えば $1+x+x^3$ は $2^0+2^1+2^3 = 11$ と表現できる.

2 の拡大体において, 加減算は XOR だけで済み高速である. 乗算・除算に関して様々な高速化テクニックがある.

3.3 Reed-Solomon 符号と Vandermonde 行列

Reed-Solomon 符号 [15] は消失訂正符号におけるパリティの生成方式の一つである. i 番目のパリティ b_i を生成するために座標 x_i を設定し, $m-1$ 次多項式の係数に入力 $a_0, \cdots, a_{k-1}$ を埋め込む.

$$b_i = a_0 + a_1 x_i + \cdots + a_{k-1} x_i^{k-1}$$

復号には Peterson 法 [13] などがある.

Vandermonde 行列 [17] は符号の分野で利用される行列で, 各要素 A_{ij} が以下で表される.

$$A_{ij} = x^{ij}$$

消失訂正符号の符号化はパリティ生成の処理を線形写像と見なして, 行列として表現できるが, この行列の一例として Vandermonde 行列を利用することができる.

3.4 Shamir の秘密分散

Shamir の秘密分散 [18] は, Reed-Solomon 符号に似ているが, 平文を a とすると, i 番目の断片 b_i を以下の計算で生成する.

$$b_i = a_0 + r_1 x_i + \cdots + r_{k-1} x_i^{k-1}$$

ただし x_i は断片に設定された座標, $r_1, \cdots, r_{k-1}$ は乱数である. 情報理論的安全性という, k 個の断片が揃わない限りいかなる計算資源をもってしても決して解読できないという最強の安全性強度を持つが, データ量は入力と比べ n 倍に増大する.

3.5 計算量的秘密分散

計算量的秘密分散は秘密分散で安全性が計算量理論に基づく方式である. Shamir の秘密分散をはじめとする情報理論的秘密分散と比べ, データ量が小さくて済む. Krawczyk は 1. 暗号鍵を情報理論的秘密分散によって (k,n)-秘密分散し (鍵の断片), 2. 暗号化したデータを同じ k, n をもつ消失訂正符号により分散し (データの断片), n 箇所のサーバがこの鍵の断片とデータの断片をそれぞれ 1 個ずつ持てば秘密分散となることを証明した [10].

鍵の容量を十分小さいとして無視すれば入力データと比較した総データ量は消失訂正符号と同じ n/k 倍となる.

4. GF(2^{64}) 乗算の高速化テクニック

後述する提案アルゴリズムは, 概念的なアルゴリズムとしてはシンプルである. しかしながら, シンプルに見える提案アルゴリズムが速度面で精密に設計されていることを理解するためには, GF(2^64) の乗算・除算の高速化テクニックが前提となる. しかし本稿では紙面の関係で除算を利用する部分まで記載できないため, 乗算についてのみ説明する.

4.0.1 乗算

$a, b \in$ GF(2^{64}) の乗算は, 2 つの 63 次多項式 $a = \sum_{i<64} a_i x^i$, $b = \sum_{i<64} b_i x^i$ を掛けてから f で割る操作である. $\sum_{i<64}\sum_{j<64} a_i b_j x^{i+j} \mod f$ である. ℓ 次の項の係数は $\bigoplus_{i+j=\ell} a_i b_j$ となる.

4.0.1.1 キャリー無し乗算 MUL

126 次多項式 $\sum_{i<64}\sum_{j<64} a_i b_j x^{i+j}$ の整数表現を得る操作をキャリー無し乗算と呼ぶ. ちょうど乗算を繰り上がり (キャリー) 無しにしたような状態になっている. Intel Sandybridge 以降, AMD Bulldozer 以降では PCLMUL 1 命令で処理できる [7].

4.0.1.2 リダクション (または, mod f 処理)

126 次多項式を mod f して 63 次多項式にする処理をリダクションと呼ぶ. 以下の同値関係を使って処理する.

$$f = x^{64} + x^4 + x^3 + x + 1 = 0 \mod f$$

変形すると以下のように, 64 次項を 4 次式に落とす関係となる.

$$x^{64} = x^4 + x^3 + x + 1 \mod f$$

64 次だけでなく 64 次以上の項も全て 60 次次数を下げられる.

$$x^{64+n} = x^n(x^4 + x^3 + x + 1) \mod f$$

126 次多項式を, 63 次多項式 ℓ と 62 次多項式 h で $\ell + x^{64}h$ と表せば,

$$\ell + x^{64}h = \ell + (x^4 + x^3 + x + 1)h \mod f$$

である. ある任意の要素 a と $x+1$ との乗算 $(x+1)a$ とは, $xa + a = xa \oplus a$ である. また $x^n a$ は a の各項が n 次高い項となるで, 整数表現における 2^n 倍, もしくは n ビット左シフトと等価である. よって, 以下が言える.

$$(x^4 + x^3 + x + 1)h = (h << 4) \oplus (h << 3) \oplus (h << 1) \oplus h$$

となる. ただし, 右シフトを >>, 左シフトを << と表す. h は 62 次多項式なので, $(h << 4) \oplus (h << 3)$ は 64 次以上の多項式となり, 再度次数を下げる必要がある. 64 次以上の部分は以下のようになる.

$$x^4(h_{62}x^{62} + h_{61}x^{61} + h_{60}x^{60}) + x^3(h_{62}x^{62} + h_{61}x^{61})$$
$$= x^{64}((h >> 60) \oplus (h >> 61))$$

64 ビット整数内で 64 ビットを超えたビットは切り捨てられることを考慮すると, 以下を計算すればよい.

$$x^{64}(h \oplus (h >> 60) \oplus (h >> 61))$$
$$= (x^4 + x^3 + x + 1)(h \oplus (h >> 60) \oplus (h >> 61))$$
$$= (x^3 + 1)(x + 1)(h \oplus (h >> 60) \oplus (h >> 61))$$

総合すると Scheme 2 のようになる.

Scheme 1 [GF(2^{64}) のリダクション RED]
入力: 62 次多項式 h,
出力: リダクション $(x^4 + x^3 + x + 1)h$

1: $h' := h \oplus (h >> 60) \oplus (h >> 61)$
2: $h'' := h' \oplus (h' + h')$
3: **output** $h'' \oplus (h'' << 3)$

4.0.1.3 ビットとの乗算 BMUL

乗算の際, 片方が単項式 (1 ビットだけ立っている) とき, キャリー無し乗算は以下のように高速に処理することができる.

$$ax^b = x^{64}(a >> (64 - b)) + (a << b)$$

特に b がコンパイル時定数の時には, $64 - b$ の計算時間がかからないこと, シフトが定数量シフトの方が速いことから高速である.

4.0.1.4 61 ビット数までとの乗算のリダクション

乗算の際, 片方が 61 ビット以内のとき, より正確には両方のビット数を足して 125 以下のとき, リダクションも効率化できる. なぜなら, $(h >> 60) \oplus (h >> 61) = 0$ だからである.

Scheme 2 [61 ビット数との乗算後の GF(2^{64}) のリダクション BRED]
入力: 59 次多項式 h,
出力: リダクション $(x^4 + x^3 + x + 1)h$

1: $h' := h \oplus (h + h)$
2: **output** $h'' \oplus (h' << 3)$

61 ビット数で 1 ビットだけ立っている数, つまり $0 \leq i \leq 60$ との範囲での 2^i との (リダクション含む) 乗算は最も高速である.

4.0.1.5 線形結合

線形結合 $\sum_{i<n} c_i a_i$ は, リダクションを最後の 1 回のみにまとめられる. なぜなら, 加算が XOR のため幾つ加算してもビット数は増加しないからである. もちろん, c_i が全て 61 ビット数ならば, BRED1 回でよい.

5. 提案手法: 分散アルゴリズム

提案手法は Krawzcyk の計算量的秘密分散に属する. そのため, おおまかに以下のような処理となる.
(1) 暗号鍵を生成し, 情報理論的秘密分散により分散する
(2) 入力データを暗号化し, 消失訂正符号により分散する
消失訂正符号化, 情報理論型秘密分散は, 以下のように平文の入力ベクトル a に線形変換 (つまり行列)A を掛けてシェアの出力ベクトル b を得る処理として表現できる.

$$b = Aa$$

例えば RAID5 は加算を XOR として以下のように表される.

$$b_0 = a_0$$
$$b_1 = a_1$$
$$b_2 = a_0 + a_1$$

これを行列で書けば, 以下のようになる. ただし乗算は AND である.

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$$

A の i 番目の行は, i 番目のシェアを生成する際に入力 a の各要素に掛ける係数を表す.

5.1 消失訂正符号化

5.1.1 入力 a

$a_0, \cdots, a_{k-1}$ を平文とすると以下のベクトル.

$$a = \begin{pmatrix} a_0 \\ \vdots \\ a_{k-1} \end{pmatrix}$$

5.1.2 出力 b

$b_0, \cdots, b_{n-1}$ を出力とすると以下のベクトル.

$$b = \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

入力ベクトルの要素と一致する要素を, データシェアと呼ぶ. そうでない要素を, パリティシェアと呼ぶ.

5.1.3 分散行列 A

消失訂正符号部分は, パリティ生成の行列を Vandermonde 行列とする Reed-Solomon 符号とする. k 行目までは単位行列なので, k 個目までのシェアは平文そのままである. 4.0.1.3 節の通り x の 60 乗までとの乗算および線形結合が高速なため, 要素が全て x のべきである Vandermonde 行列との乗算が高速となる.

$$A_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \text{ and } i < k \\ x^{(i-k)j} & \text{if } i \geq k \end{cases}$$

つまり,

$$\underbrace{\left(\begin{array}{ccccc} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \\ \hline 1 & 1 & 1 & \cdots & 1 \\ 1 & x & x^2 & \cdots & x^{k-1} \\ 1 & x^2 & x^4 & \cdots & x^{2(k-1)} \\ \vdots & & & \ddots & \\ 1 & x^{m-1} & x^{2(m-1)} & \cdots & x^{(m-1)(k-1)} \end{array}\right)}_{k \text{ 列}} \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\end{array}}\right\} k \text{ 行} \\ \\ \\ \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\end{array}}\right\} m \text{ 行} \end{array}$$

5.2 情報理論型秘密分散の分散

5.2.1 入力 a

$r_0, \cdots, r_{k-2}$ を乱数とし, s を平文とすると以下のベクトル.

$$a = \begin{pmatrix} r_0 \\ \vdots \\ r_{k-2} \\ \hline s \end{pmatrix}$$

5.2.2 出力 b

消失訂正符号化の場合 (5.1.2 節) と同じ.

5.3 分散行列 A

情報理論的秘密分散においても, 分散処理を線形写像と見たとき, 可能な限り演算が効率的な 1 や 0 を行列要素として増やしたい. そのため, 消失訂正符号化と同じような行列の構成ができないか考える. しかし消失訂正符号化の分散行列と情報理論型秘密分散の入力 a を掛けると $b_{k-1} = s$ となってしまう. 秘密分散では秘匿性のため, 平文をそのままシェアとすることは禁じられる. そのため, 消失訂正符号化と異なり k 行目から Vandermonde 行列とすれば, 行列に 0 と 1 が多く効率的で, かつ安全な情報理論的秘密分散とすることができる.

$$A_{ij} = \begin{cases} 1 & \text{if } i = j \text{ and } i < k-1 \\ 0 & \text{if } i \neq j \text{ and } i < k-1 \\ x^{(i-k+1)j} & \text{if } i \geq k-1 \end{cases}$$

つまり,

$$\underbrace{\left(\begin{array}{ccccc} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \\ \hline 1 & 1 & 1 & \cdots & 1 \\ 1 & x & x^2 & \cdots & x^{k-1} \\ 1 & x^2 & x^4 & \cdots & x^{2(k-1)} \\ \vdots & & & \ddots & \\ 1 & x^m & x^{2m} & \cdots & x^{m(k-1)} \end{array}\right)}_{k \text{ 列}} \begin{array}{l} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\end{array}}\right\} k-1 \text{ 行} \\ \\ \\ \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\end{array}}\right\} m+1 \text{ 行} \end{array}$$

6. 提案方式: 復元

復元の操作も線形変換と見ることができる. A', b' を, A, b のうち利用する k 個のシェアに対応する行だけを抜き出した行列/ベクトルとすると,

$$b' = A'a$$

なので, A' に逆行列が存在すれば以下のように復元できる.

$$a = A'^{-1}b'$$

6.1 消失訂正符号の復号

まず, データシェアが全て利用可能な場合, データシェアは入力そのものなので, 入力をそのまま出力すればよい. 何もする必要はない. 以降, 少なくとも 1 つ利用不能なデータシェアがある場合を考える. この場合, 利用可能なデータシェアとパリティシェアから, 平文を復元する必要がある. 利用不能なデータシェアが f 個存在するとする.

概念的には, 逆行列のうち f 行を抜き出してシェアと掛ければよい. しかし, そのまま逆行列を掛けるということはしない. 逆行列の要素は一般的に, x の 60 までのべきとはならない. そのため逆行列のうち f 行と掛けた場合の処理量は,

$$f(k\text{MUL} + \text{RED}) \tag{1}$$

となる.

6.1.1 利用可能なデータシェア成分の除去

逆行列をそのまま掛けるかわりに, データシェアが 1 つの入力要素の成分しかもたないことを利用して, 最初にパリティシェアからデータシェアに対応する平文の要素を除去する. 例えばデータシェア b_0 とあるパリティシェア b_p (ただし p はいずれかのパリティシェアに対応する番号) について, 以下のように 0 番目の平文の要素を除去できる.

$$b_0 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix} a,$$

$$b_p = \begin{pmatrix} 1 & x^{p-k} & x^2p-k & \cdots & x^{(p-k)(k-1)} \end{pmatrix} a \text{ より},$$

$$b_p - b_0 = \begin{pmatrix} 0 & x^{p-k} & x^2p-k & \cdots & x^{(p-k)(k-1)} \end{pmatrix} a$$

このような変形を使うと, $d_0, \cdots, p_{k-f-1}$ を復元に用いるパリティシェアの番号, $p_0, \cdots, p_{f-1}$ を復元に用いるパリティシェアの番号, $e_0, \cdots e_{f-1}$ を利用不能なデータシェアの番号とすれば, 以下のように復元すべき $a_{e_0}, \cdots, a_{e_{f-1}}$ と f 個の要素を, k 次よりも低次の f 次正方行列を用いて表現することができる.

$$\begin{pmatrix} b_{p_0} - \sum_{i<f} x^{p_0 d_i} b_{d_i} \\ b_{p_1} - \sum_{i<f} x^{p_1 d_i} b_{d_i} \\ \vdots \\ b_{p_{f-1}} - \sum_{i<f} x^{p_{f-1} d_i} b_{d_i} \end{pmatrix} = \underbrace{\begin{pmatrix} x^{p_0 e_0} & \cdots & x^{p_0 e_{f-1}} \\ x^{p_1 e_0} & \cdots & x^{p_1 e_{f-1}} \\ \vdots & \ddots & \\ x^{p_{f-1} e_0} & \cdots & x^{p_{f-1} e_{f-1}} \end{pmatrix}}_{f \text{ 列}} \left.\begin{pmatrix} a_{e_0} \\ a_{e_1} \\ \vdots \\ a_{e_{f-1}} \end{pmatrix}\right\} f \text{ 行}$$

変形後の左辺 b' にこの f 次正方行列の逆行列を掛ければ復元すべき平文が計算される.

$$\begin{pmatrix} x^{p_0 e_0} \cdots & x^{p_0 e_{f-1}} \\ x^{p_1 e_0} \cdots & x^{p_1 e_{f-1}} \\ \vdots & \ddots \\ x^{p_{f-1} e_0} & \cdots \quad x^{p_{f-1} e_{f-1}} \end{pmatrix}^{-1} \begin{pmatrix} b'_0 \\ b'_1 \\ \vdots \\ b'_{f-1} \end{pmatrix} = \begin{pmatrix} a_{e_0} \\ a_{e_1} \\ \vdots \\ a_{e_{f-1}} \end{pmatrix} \quad (2)$$

ただし

$$b' = \begin{pmatrix} b'_0 \\ b'_1 \\ \vdots \\ b'_{f-1} \end{pmatrix} = \begin{pmatrix} b_{p_0} - \sum_{i<f} x^{p_0 d_i} b_{d_i} \\ b_{p_1} - \sum_{i<f} x^{p_1 d_i} b_{d_i} \\ \vdots \\ b_{p_{f-1}} - \sum_{i<f} x^{p_{f-1} d_i} b_{d_i} \end{pmatrix}$$

b をこのように b' へと変形するのに必要な処理量は,

$$(k-f)f\text{BMUL} + f\text{BRED}$$

であり, f 次の逆行列との乗算は

$$f^2\text{MUL} + f\text{RED}$$

だから, 合計すると

$$f((k-f)\text{BMUL} + f\text{MUL} + \text{BRED} + \text{RED}) \quad (3)$$

となる.

6.2 情報理論型秘密分散の復元

情報理論型秘密分散の復元は消失訂正符号とは状況がかなり異なる. 消失訂正符号では, 平文の線形結合で表されるシェアを平文へと加工するためには, f 回の k 次元の線形結合を行うため, fk に比例する乗算が必要である.

一方情報理論型分散の復元では, 1 個の平文を復元するだけなので, 1 回の k 次元線形結合で復元できるはずである. よって, 消失訂正符号の復号で用いた, f 個のパリティから BMUL で $k-f$ 個のデータシェアの成分を除去していく処理量 $f(k-f)$ の処理よりも, 低速な MUL でよいから k 次元線形結合 1 回で復元する方を選択する方が結果的に高速である.

7. 提案手法: 復旧

消失訂正符号や秘密分散には, 分散 (符号化) と復元 (復号) のほか, 復旧という操作が存在する. 復旧とは, 一部の断片が破損した際, 他の k 個以上の断片から破損した断片を再度生成する操作である.

断片の破損はほとんどの場合 1 個のため, 本稿では 1 断片の復旧を想定し, 情報理論的秘密分散の復元と同じく, 線形結合 1 回で復旧する方法をとる.

8. 性能測定

本節では提案手法を実装したライブラリである SHSS の性能を報告する. まず 8.1 節でライブラリ単体の性能について報告し, 次いで 8.2 節で, 標準的なオブジェクトストレージソフトウェアの一つである OpenStack Swift から呼んだ場合の性能を報告する. 共通の環境については以下である.

(パラメータ) SHSS では現在, 4 組の k, n の組み合わせが実装されている. $(6, 9), (10, 14), (11, 18), (20, 24)$ の 4 組である. これらはいずれも堅牢性において, Amazon AWS S3 [1] や NTT コミュニケーションズ Cloud^n [2] の堅牢性である nine elevens (99.999999999%) 以上となるパラメータである. パラメータ (k, n) の算出においては Markov 吸収モデルを用いる信頼性モデル [21] を利用し, 算出に必要な値は以下とした.

- 論理容量: 500 TB および 100 TB
- 1 ディスクのサイズ: 3 TB
- 1 ノードあたりのディスク数: 35 および 12
- RepairRate (per Node, Network): 2.0 (Gbps)
- Partition 数 (per disk): 100
- Node Failure Rate: 0.02
- AFR (Disk Failure Rate): 0.0073
- 故障判定までの時間: 0.1 (hours)

(ソフトウェア環境) ソフトウェア環境はどちらの測定でも以下である.

- OS: Ubuntu 14.04.1 Server
- 言語: C++
- コンパイラ: gcc 4.7.3

表 1　SHSS の各演算の性能 (128 bit AES)[Gbps]

演算	(6,3)	(10, 4)	(11, 7)	(20, 4)
encode	21.8	21.8	15.4	21.6
decode	23.2	18.7	17.3	19.4
reconstruct	10.7	6.7	5.9	3.7

表 2　SHSS の各演算の性能 (256 bit AES) [Gbps]

演算	(6,3)	(10, 4)	(11, 7)	(20, 4)
encode	17.5	18.0	13.7	17.6
decode	18.0	15.6	14.5	16.3
reconstruct	11.6	7.0	6.1	3.8

(測定データ) 測定に使用したデータの条件は以下の通りである.

- データサイズ: オブジェクトストレージを想定し, 1 MB のオブジェクトに分割されたバイナリが連続して到来すると想定する. 実データで期待できないはずであるキャッシュメモリの効果を最小化するため 1 MB のランダムデータを十分な数 (1 万個) 用意し, 連続に順に処理を行う.
- データ格納場所: 入力先はメインメモリからであり, 出力先もメインメモリである.
- 復元時の断片の状況: 提案手法の処理からして, データシェアが多く残っているほど高速であるが, 測定ではデータシェアが 2 個破損した状態とする.
- 復旧時の断片の状況: 破損したシェアがパリティシェアの場合は, データシェアから高速に復旧できるが, 測定ではデータシェアの方が 1 個破損した状態とする.

8.1　ライブラリ単体の比較

SHSS 単体で性能測定した結果を 表 1 (鍵長 128 bit), 表 2 (鍵長 256 bit) に記す. 秘匿性については以下の 2 バリエーションを測定した.

(秘匿性) 計算量的秘密分散に必要な暗号化は AES とし, 鍵長は以下の 2 種とした.

(1) 128 bit AES
(2) 256 bit AES

(ハードウェア環境) ハードウェア環境は以下である.

- CPU: Intel Core i7 4710MQ (Haswell Refresh) 2.5GHz x 4 core (ただし利用は 1 スレッド)
- Memory: 16GB dual channel DDR3L 1600MHz

8.1.1　既存研究との比較

文献中では秘密分散とはされていないが, Resch らによる消失訂正符号と暗号化の組み合わせである AONT – RS_{secure} [16] と比較する. 論文中のグラフから, (20, 4) のパラメータで 0.46 Gbps であることが読み取れる. SHSS は鍵長 128 bit では 21.6 Gbps で, 約 47 倍高速である. また鍵長 256 bit では 17.6 Gbps で, 約 38 倍高速である.

純粋な計算量的秘密分散との比較に関しては, 同じパラメータの計算量的秘密分散の実装報告が無いことから直接の比較は難しい. しかし, 従来最速であった $k = 2, n = 3$ で

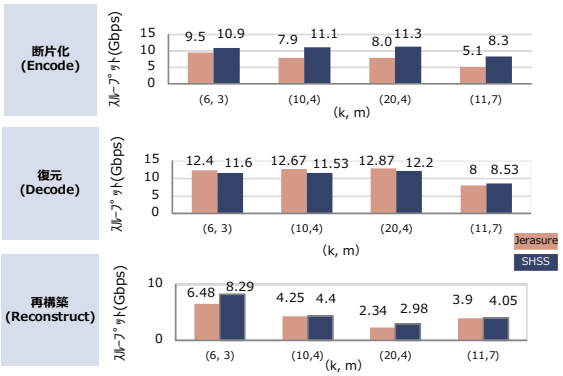

図 1　OpenStack Swift の消失訂正符号 API 部分である liberasure から呼び出した場合のスループット比較 (SHSS vs Jerasure)

の分散 3.2 Gbps, 復元 3.3 Gbps (最も条件の近い 1 スレッド Core i7 2.4GHz の場合, 256 bit AES) [22] を SHSS が 4 パラメータとも超えており最大では 5.5 倍程度の速度であること, さらに一般に秘密分散の計算量は k および m に比例するため SHSS の 4 パラメータが $k = 2, n = 3(m = 1)$ に対してかなり不利なことも併せれば, SHSS は従来の計算量的秘密分散より遙かに高速であると言えるであろう.

8.2　OpenStack Swift 上での比較

OpenStack Swift の消失訂正符号 API である liberasure から 標準ライブラリである Jerasure [4] を呼んだ場合と, SHSS を呼んだ場合の比較を 図 8.2 に記す. この測定におけるハードウェア環境は, SHSS, Jerasure とも同じで, 以下である.

(ハードウェア環境)

- CPU: Intel Xeon E5-2630 v3 (Haswell) 2.4GHz x 8 core (ただし利用は 1 スレッド)
- Memory: 32GB quad channel DDR3 1600MHz

グラフから, 秘匿性機能を持つ SHSS が秘匿性機能を持たない Jerasure とほぼ同等の性能を持つことが示された.

9.　おわりに

本稿ではオブジェクトストレージ向けに, 効率的な秘密分散の方式を提案し, この方式を実装した非常に高速な秘密分散ライブラリ SHSS の性能を報告した.

SHSS は同じパラメータの既存研究より鍵長 128 bit で約 47 倍, 256 bit で約 38 倍高速であった. また堅牢性でなく高速性最重視の $k = 2, n = 3$ のパラメータの既存研究と比較しても 約 5.5 倍高速であった.

さらに OpenStack Swift から呼び出した場合のスループットにおいて OpenStack Swift 標準の消失訂正符号エンジンである Jerasure と同等の性能を持つことを示した. 秘密分散は消失訂正符号に対して秘匿性を追加した上位互換機能であるため, この結果は OpenStack Swift に対して, 性能面でのデメリットなく秘匿性機能を追加可能であるという効果を意味する.

参考文献

[1] Amazon s3. http://aws.amazon.com/jp/s3/.
[2] cloud^{n}. http://www.ntt.com/cloudn/.
[3] Intel Storage Acceleration Library (ISA-L). https://01.org/intel%C2%AE-storage-acceleration-library-open-source-version.
[4] Jerasure. http://github.com/tsuraan/Jerasure.
[5] OpenStack Swift Erasure Code Support. http://docs.openstack.org/developer/swift/overview_erasure_code.html.
[6] Welcome to Swift's Documentation. http://docs.openstack.org/developer/swift/.
[7] A. Fog. Instruction tables lists of instruction latencies, throughputs and micro-operation break-downs for intel, amd and via cpus. *http://www.agner.org/optimize/instruction_tables.pdf*.
[8] S. Ghemawat, H. Gobioff, and S. Leung. The google file system. In M. L. Scott and L. L. Peterson eds., *Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, SOSP 2003, Bolton Landing, NY, USA, October 19-22, 2003*, pp. 29–43. ACM, 2003.
[9] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In G. Heiser and W. C. Hsieh eds., *2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012*, pp. 15–26. USENIX Association, 2012.
[10] H. Krawczyk. Secret sharing made short. In D. R. Stinson ed., *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, Vol. 773 of *Lecture Notes in Computer Science*, pp. 136–146. Springer, 1993.
[11] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka. On a fast (*k, n*)-threshold secret sharing scheme. *IEICE Transactions*, 91-A(9):2365–2378, 2008.
[12] S. Obana and T. Araki. Almost optimum secret sharing schemes secure against cheating for arbitrary secret distribution. In X. Lai and K. Chen eds., *ASIACRYPT*, Vol. 4284 of *Lecture Notes in Computer Science*, pp. 364–379. Springer, 2006.
[13] W. W. Peterson. Encoding and error-correction procedures for the bose-chaudhuri codes. *IRE Transactions on Information Theory*, 6(4):459–470, 1960.
[14] J. S. Plank, M. Blaum, and J. L. Hafner. SD codes: erasure codes designed for how storage systems really fail. In K. A. Smith and Y. Zhou eds., *Proceedings of the 11th USENIX conference on File and Storage Technologies, FAST 2013, San Jose, CA, USA, February 12-15, 2013*, pp. 95–104. USENIX, 2013.
[15] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.
[16] J. K. Resch and J. S. Plank. AONT-RS: blending security and performance in dispersed systems. In *9th USENIX Conference on File and Storage Technologies, CA, USA, February 15-17, 2011*, pp. 191–202, 2011.
[17] G. A. F. Seber. *A Matrix Handbook for Statisticians*. John Wiley & Sons, inc.
[18] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
[19] S. Takahashi and K. Iwamura. Secret sharing scheme suitable for cloud computing. In L. Barolli, F. Xhafa, M. Takizawa, T. Enokido, and H. Hsu eds., *27th IEEE International Conference on Advanced Information Networking and Applications, AINA 2013, Barcelona, Spain, March 25-28, 2013*, pp. 530–537. IEEE Computer Society, 2013.
[20] M. Tompa and H. Woll. How to share a secret with cheaters. In A. M. Odlyzko ed., *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, Vol. 263 of *Lecture Notes in Computer Science*, pp. 261–265. Springer, 1986.
[21] Q. Xin, E. L. Miller, T. J. E. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin. Reliability mechanisms for very large storage systems. In *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, MSS 2003, San Diego, California, USA, April 7-10, 2003*, pp. 146–156. IEEE Computer Society, 2003.
[22] 五十嵐大, 菊池亮, 濱田浩気, 千田浩司. AES-NI を用いた 2 の拡大体演算高速化と, 10GEther/Infiniband を見越した秘密分散の高速実装. In *SCIS2014*, 2014.
[23] 松尾正克, 武藤浩二. 排他的論理和を用いた (k, n)-しきい値秘密分散法. *Panasonic Technical Journal*, 59(2):115–120, 2013.
[24] 須賀祐治. 排他的論理和を用いた (k,n) 閾値秘密分散法の新しい構成とその優位性について. In *CSS2012*, 2012.
[25] 保坂範和, 多田美奈子, 加藤岳久. 秘密分散法とその応用. 東芝レビュー, 62(7):23–26, 2007.