Vol.2016-DPS-166 No.5
Vol.2016-CSEC-72 No.5
2016/3/3

# k out of n 秘密計算プロトコルの一考察

滝 雄太郎[1,a)]　藤田 茂[1,b)]　宮西 洋太郎[2,c)]　白鳥 則郎[3,d)]

**概要**：クラウドシステムを利用して情報処理を行なう際に，クラウドシステムを構成するサーバがクラッキングされる，あるいはサーバの管理者が不正行為を行なうという可能性があるので，クラウドを安全に利用するためには秘匿性を担保する必要がある．この課題に対してセキュアマルチパーティ法による秘密分散法や秘密計算法が研究されている．中でも，“軽量 3 パーティ秘匿関数計算” は，1) 入力値の秘匿性が保証される，2) 演算結果が改竄された場合に，その検出効率が良い，とされている．本研究では，実用面から幾つかの制約を置いた上で，“軽量 3 パーティ秘匿関数計算” の拡張を検討したので，これを報告する．

**キーワード**：クラウド，秘密分散，秘密計算，セキュリティ

# A Study for k out of n Secret Computation Protocol

Yutaro Taki[1,a)]　Shigeru Fujita[1,b)]　Yohtaro Miyanishi[2,c)]　Norio Shiratori[3,d)]

***Abstract:*** When performing information processing by using Cloud System, Cracker is able to crack server that make up the cloud system, and an administrator of the server is able to do the fraud on Cloud System. In order to use safely, it is necessary to ensure the confidentiality. To solve this problem Secret sharing method and a secret computation method has been studied by the secure multi-party law. Above all, “A Lightweight Three-party secure function evaluatio” is, 1) confidentiality of the input values is ensured, 2) operation result is falsified, is the detection efficiency is good. In this paper, we report that having examined the expansion of “A Lightweight Three-party Secure Function evaluation” in terms of the practical side with some constraints.

***Keywords:*** Cloud, Secret Sharing, Secure Computation, Security

## 1. はじめに

クラウドシステム (以下，クラウドと表記) 上に存在する計算資源を使って処理を実行することで，資産保有のコストを下げ，計算処理を必要とする時だけ，必要な計算資源やディスク資源を利用することができるので，クラウド上での処理を実行したいという要請は高く，導入事例も多い．また，秘密を守るべきデータを分割して，異なる拠点に分散配置することで，一箇所に重要なデータを保管するよりも安全性が向上することが期待できる [1]，[2].

利用者からみると，手元のデータを複数のデータに分割し，それぞれを複数のクラウド業者に分散して保存することとする．各クラウド業者は，複数のサーバを保有しているが，論理的にひとつの計算実行の主体とみなし，これを

[1] 千葉工業大学情報科学部情報工学科，〒 275-0016 千葉県習志野市津田沼 2-17-1
Department of Computer Science, Faculty of Information and Computer Science, Chiba Institute of Technology, 2-17-1, Tsudanuma, Narashino-shi, Chiba-ken, 275-0016 Japan
[2] 株式会社アイエスイーエム，〒 103-0008 東京都中央区日本橋中州 1-5 1103 号
ISEM, Inc., 1103, 1-5, Nihonbashinakasu, Chuo-ku, Tokyo, 103-0008 Japan
[3] 早稲田大学国際情報通信センター 〒 169-8555 東京都新宿区大久保 3-4-1 早大 55 号館
Waseda University, Global Infromation and Telecommunication Institute, 3-4-1 Okubo, Sinjuku-ku, Tokyo, 169-8555 Japan
[a)] taki@sf.cs.it-chiba.ac.jp
[b)] fujita@cs.it-chiba.ac.jp
[c)] ymiyanishi@isem.co.jp
[d)] norio@shiratori.riec.tohoku.ac.jp

本稿ではパーティと呼ぶことにする．

複数のパーティに対して秘密にしたいデータを分散配置する手法のことを秘密分散法と呼ぶ．また，秘密分散法を用いて分散配置したデータを復元することなく計算することのできる手法のことを秘密計算法と呼ぶ．

そして，秘密計算法の一手法にセキュアマルチパーティ法がある．この手法では秘密にしたいデータを分散配置させる他に配置されたデータの信頼性も両立させる手法である．内容としては，秘密にしたい数値データ $a, b$ を $n$ 個のパーティへと分散させそのうちの $k$ 個のパーティからデータを得て，元の数値 $a, b$ を復元する手法である．（以下 k out of n と称する）

秘密分散法の中で，記憶容量の削減と計算量的安全性を提案する手法 [3]，k out of n を条件として，秘密情報の復元，加算，減算，乗算を可能とする手法の提案 [4]，セキュアマルチパーティ法に，新たな工夫と制限を導入することで乗算と除算を効率的に行なう仕組みの提案 [5]，秘密分散法を利用した乗算が一度だけ可能な秘密計算の方法の提案 [6]，記憶容量の削減を実現する秘密分散方式の提案 [7] が，行なわれており，クラウドを使って，秘密分散，秘密計算を行ないたいという要望は高く，多くの研究が盛んに行われている．

一方で，データを外部の機関が保有する機器上へ委ねることへの不安や，安全性への評価の難しさから，クラウド上での処理を実施できない場合がある．安全性への評価については，文献 [8] にて評価方法を提案している．また安心感を高める軽量秘密計算法の実験が報告されている [9]．

秘匿性を担保するための方法として，“軽量 3 パーティ秘匿関数計算”[10] が提案されている．この手法は，主体間の結託は無いと仮定した上で，パーティ数を $n = 3$ に特化した方式である．

今後のクラウドによる処理の増加の中で，データ量の増加に伴なって，利用するパーティの数を $n = 3$ に限定せずに，$n = 4, 5, ...$ と増やす必要性が生じると考えた．

また，“軽量 3 パーティ秘匿関数計算”[10] では，完全秘匿性を担保するために，各主体が協調して計算する乗算での手続き Mul には，乱数が含まれている．この乱数を用いることで，計算量が増加することから，完全な秘匿性を担保せずに，類推が困難であるという程度の処理によっても計算が行えれば，処理の高速化が行えるのではないかと考えた．

以下，2 章では，“軽量 3 パーティ秘匿関数計算”[10] の分散プロトコルと，乗算プロトコルについて述べる．3 章では，2 out of 3 から 2 out of 4 への拡張の検討，3 out of 4 への拡張の検討，一般的な k out of n への拡張の検討とその結果得られた制約について述べる．4 章では，シェアに乱数を含めない場合の実行速度について，実機で評価実験を行なった例をしめす．5 章で，(2 out of 3) out of n として，信頼性を向上させるために，パーティを増加させた時に取り得る構成についての検討を述べる．6 章で，まとめを述べる．

## 2. 関連研究

本章では，軽量 3 パーティ秘匿関数計算 [10] の分散プロトコルと乗算プロトコルの要約を述べる．

### 2.1 分散プロトコル

あるデータ $a$ に対する分散プロトコルは次の手順でおこなわれる．データ $b$ についても同様である．

(1) $a_0, a_1$ をランダムに設定する．
(2) $a_2 := a - a_0 - a_1$ を計算する．
(3) シェアを $i = 0, 1, 2$ として $[a]_i := (a_i, a_{i+1})$ を作成する．
(4) 作成したシェア $[a]_i$ をパーティ $P_i$ へと送信する．

なお，シェア $[a]_i$ の添字 $i$ は記述の簡略化のために $i \pmod 3$ と余剰をとってあるものとする．

### 2.2 乗算プロトコル

乗算プロトコルでは 2.1 に述べた分散プロトコルを用いてデータ $a, b$ が分散されている状態で説明する．データ $a, b$ は式 (1), (2) のように分割されている．

$$a_2 := a - a_0 - a_1 \tag{1}$$
$$b_2 := b - b_0 - b_1 \tag{2}$$

そして式 (3), (4), (5) のようなシェア，パーティへと分散されている．

$$[a]_0 := (a_0, a_1), [b]_0 := (b_0, b_1), P_0 := ([a]_0, [b]_0) \tag{3}$$
$$[a]_1 := (a_1, a_2), [b]_1 := (b_1, b_2), P_1 := ([a]_1, [b]_1) \tag{4}$$
$$[a]_2 := (a_2, a_0), [b]_2 := (b_2, b_0), P_2 := ([a]_2, [b]_2) \tag{5}$$

以上のような状態で $a \times b := c_0 + c_1 + c_2$ となるようなシェア $[c]_i$ を作成する．

(1) パーティ $P_0$ は次の操作をおこなう．
  (a) $r_1, r_2, c_0$ をランダムに設定する．
  (b) $c_1 := (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 - c_0$ を計算する．
  (c) パーティ $P_1, P_2$ にそれぞれ $(r_1, c_1), (r_2, c_0)$ を送信する．
  (d) $[c]_0 := (c_0, c_1)$ とする．
(2) パーティ $P_1, P_2$ はそれぞれ次の操作をおこなう．
  (a) パーティ $P_1$ は $y := a_1 b_2 + a_2 b_1 + r_1$ を計算して $P_2$ へと送信する．
  (b) パーティ $P_2$ は $z := a_2 b_0 + a_0 b_2 + r_2$ を計算して $P_1$ へと送信する．
  (c) それぞれ送信されてきた $z, y$ を用いて $c_2 := y + z + a_2 b_2$ を計算する．

(d) $[c]_1 := (c_1, c_2), [c]_2 := (c_2, c_0)$ とする．

ここで以上の手順の中に出てきた式を列挙する．

$$c_1 := (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 - c_0 \tag{6}$$
$$y := a_1 b_2 + a_2 b_1 + r_1 \tag{7}$$
$$z := a_2 b_0 + a_0 b_2 + r_2 \tag{8}$$
$$c_2 := y + z + a_2 b_2 \tag{9}$$

この時確かに $a \times b = c_0 + c_1 + c_2$ である．理由を式 (10) に述べる，

$$\begin{aligned} c_0 + c_1 + c_2 &= (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 \\ &\quad + y + z + a_2 b_2 \\ &= (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 + a_1 b_2 + a_2 b_1 \\ &\quad + r_1 + a_2 b_0 + a_0 b_2 + r_2 + a_2 b_2 \\ &= (a_0 + a_1)(b_0 + b_1) + a_1 b_2 + a_2 b_1 \\ &\quad + a_2 b_0 + a_0 b_2 + a_2 b_2 \\ &= (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) \\ &= a \times b \end{aligned} \tag{10}$$

## 3. 提案手法

### 3.1 2 out of 4 の場合

2 out of 4 の場合についての分散・乗算プロトコルの動作について述べる．

#### 3.1.1 2 out of 4 の場合の分散プロトコル

あるデータ $a$ に対する 2 out of 4 の場合の分散プロトコルは次の手順でおこなわれる．

(1) $a_0, a_1, a_2$ をランダムに設定する．
(2) $a_3 := a - a_0 - a_1 - a_2$ を計算する．
(3) シェアを $i = 0, 1, 2, 3$ として $[a]_i := (a_i, a_{i+1}, a_{i+2})$ を作成する．
(4) 作成したシェア $[a]_i$ をパーティ $P_i$ へと送信する．

なお, シェア $[a]_i$ の添字 $i$ は記述の簡略化のために $i \pmod 4$ と余剰をとってあるものとする．

#### 3.1.2 2 out of 4 の場合の乗算プロトコル

乗算プロトコルでは先に述べた分散プロトコルを用いてデータ $a, b$ が分散されている状態で説明する．データ $a, b$ は式 (11), (12) のように分割されている．

$$a_3 := a - a_0 - a_1 - a_2 \tag{11}$$
$$b_3 := b - b_0 - b_1 - b_2 \tag{12}$$

そして式 (13), (14), (15), (16) のようなシェア，パーティへと分散されている．

$$[a]_0 := (a_0, a_1, a_2), [b]_0 := (b_0, b_1, b_2), P_0 := ([a]_0, [b]_0) \tag{13}$$
$$[a]_1 := (a_1, a_2, a_3), [b]_1 := (b_1, b_2, b_3), P_1 := ([a]_1, [b]_1) \tag{14}$$
$$[a]_2 := (a_2, a_3, a_0), [b]_2 := (b_2, b_3, b_0), P_2 := ([a]_2, [b]_2) \tag{15}$$
$$[a]_3 := (a_3, a_0, a_1), [b]_3 := (b_3, b_0, b_1), P_3 := ([a]_3, [b]_3) \tag{16}$$

以上のような状態で $a \times b := c_0 + c_1 + c_2 + c_3$ となるようなシェア $[c]_i$ を作成する．

(1) パーティ $P_0$ は次の操作をおこなう．
  (a) $r_1, r_2, r_3, c_0, c_1$ をランダムに設定する．
  (b) $c_2 := (a_0+a_1+a_2)(b_0+b_1+b_2) - r_1 - r_2 - r_3 - c_0 - c_1$ を計算する．
  (c) パーティ $P_1, P_2, P_3$ にそれぞれ $(r_1, c_1, c_2), (r_2, c_2, c_0), (r_3, c_0, c_1)$ を送信する．
  (d) $[c]_0 := (c_0, c_1, c_2)$ とする．
(2) パーティ $P_1, P_2, P_3$ はそれぞれ次の操作をおこなう．
  (a) パーティ $P_1$ は $x := a_1 b_3 + a_3 b_1 + r_1$ を計算して $P_2, P_3$ へと送信する．
  (b) パーティ $P_2$ は $y := a_2 b_3 + a_3 b_2 + r_2$ を計算して $P_1, P_3$ へと送信する．
  (c) パーティ $P_3$ は $z := a_3 b_0 + a_0 b_3 + r_3$ を計算して $P_1, P_2$ へと送信する．
  (d) それぞれ送信されてきた $x, y, z$ を用いて $c_3 := x + y + z + a_3 b_3$ を計算する．
  (e) $[c]_1 := (c_1, c_2, c_3), [c]_2 := (c_2, c_3, c_0), [c]_3 := (c_3, c_0, c_1)$ とする．

ここで以上の手順の中に出てきた式を列挙する．

$$c_2 := (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) - r_1 - r_2 - r_3 - c_0 - c_1 \tag{17}$$
$$x := a_1 b_3 + a_3 b_1 + r_1 \tag{18}$$
$$y := a_2 b_3 + a_3 b_2 + r_2 \tag{19}$$
$$z := a_3 b_0 + a_0 b_3 + r_3 \tag{20}$$
$$c_3 := x + y + z + a_3 b_3 \tag{21}$$

この時確かに $a \times b = c_0 + c_1 + c_2 + c_3$ である．理由を式 (22) に述べる，

$$
\begin{aligned}
&c_0 + c_1 + c_2 + c_3 \\
&\quad = (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) - r_1 - r_2 - r_3 \\
&\qquad + x + y + z + a_3 b_3 \\
&\quad = (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) - r_1 - r_2 - r_3 \\
&\qquad + a_1 b_3 + a_3 b_1 + r_1 \\
&\qquad + a_2 b_3 + a_3 b_2 + r_2 + a_3 b_0 + a_0 b_3 + r_3 + a_2 b_2 \\
&\quad = (a_0 + a_1 + a_2)(b_0 + b_1 + b_2) \\
&\qquad + a_1 b_3 + a_3 b_1 + a_2 b_3 + a_3 b_2 \\
&\qquad + a_3 b_0 + a_0 b_3 + a_2 b_2 \\
&\quad = (a_0 + a_1 + a_2 + a_3)(b_0 + b_1 + b_2 + b_3) \\
&\quad = a \times b \qquad (22)
\end{aligned}
$$

以上により 2 out of 4 の場合において乗算は可能である．

### 3.2 3 out of 4 の場合

3 out of 4 の場合についての分散・乗算プロトコルの動作について述べる．3 out of 4 の場合では分散は可能であるが，乗算は不可能である．次に述べる．

#### 3.2.1 3 out of 4 の場合の分散プロトコル

あるデータ $a$ に対する 3 out of 4 の場合の分散プロトコルは次の手順でおこなわれる．

(1) $a_0, a_1, a_2$ をランダムに設定する．
(2) $a_3 := a - a_0 - a_1 - a_2$ を計算する．
(3) シェアを $i = 0, 1, 2, 3$ として $[a]_i := (a_i, a_{i+1})$ を作成する．
(4) 作成したシェア $[a]_i$ をパーティ $P_i$ へと送信する．

なお, シェア $[a]_i$ の添字 $i$ は記述の簡略化のために $i \pmod 4$ と余剰をとってあるものとする．

#### 3.2.2 3 out of 4 の場合の乗算プロトコル

乗算プロトコルでは先に述べた分散プロトコルを用いてデータ $a, b$ が分散されている状態で説明する．データ $a, b$ は式 (23), (24) のように分割されている．

$$a_3 := a - a_0 - a_1 - a_2 \qquad (23)$$

$$b_3 := b - b_0 - b_1 - b_2 \qquad (24)$$

そして式 (25), (26), (27), (28) のようなシェア，パーティへと分散されている．

$$[a]_0 := (a_0, a_1), [b]_0 := (b_0, b_1), P_0 := ([a]_0, [b]_0) \qquad (25)$$

$$[a]_1 := (a_1, a_2), [b]_1 := (b_1, b_2), P_1 := ([a]_1, [b]_1) \qquad (26)$$

$$[a]_2 := (a_2, a_3), [b]_2 := (b_2, b_3), P_2 := ([a]_2, [b]_2) \qquad (27)$$

$$[a]_3 := (a_3, a_0), [b]_3 := (b_3, b_0), P_3 := ([a]_3, [b]_3) \qquad (28)$$

以上のような状態で $a \times b := c_0 + c_1 + c_2 + c_3$ となるようなシェア $[c]_i$ を作成しようとする．

(1) パーティ $P_0$ は次の操作をおこなう．
  (a) $r_1, r_2, r_3, c_0, c_1$ をランダムに設定する．
  (b) $c_2 := (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 - r_3 - c_0 - c_1$ を計算する．
  (c) パーティ $P_1, P_2, P_3$ にそれぞれ $(r_1, c_1), (r_2, c_2), (r_3, c_0)$ を送信する．
  (d) $[c]_0 := (c_0, c_1)$ とする．
(2) パーティ $P_1, P_2, P_3$ はそれぞれ次の操作をおこなう．
  (a) パーティ $P_1$ は $x := a_1 b_2 + a_2 b_1 + r_1$ を計算して $P_2, P_3$ へと送信する．
  (b) パーティ $P_2$ は $y := a_2 b_3 + a_3 b_2 + r_2$ を計算して $P_1, P_3$ へと送信する．
  (c) パーティ $P_3$ は $z := a_3 b_0 + a_0 b_3 + r_3$ を計算して $P_1, P_2$ へと送信する．
  (d) それぞれ送信されてきた $x, y, z$ を用いて $c_3 := x + y + z + a_3 b_3$ を計算しようとする．
  (e) $[c]_1 := (c_1, c_2), [c]_2 := (c_2, c_3), [c]_3 := (c_3, c_0)$ としようとする．

ここで以上の手順の中に出てきた式を列挙する．

$$c_2 := (a_0 + a_1)(b_0 + b_1) - r_1 - r_2 - r_3 - c_0 - c_1 \qquad (29)$$

$$x := a_1 b_2 + a_2 b_1 + r_1 \qquad (30)$$

$$y := a_2 b_3 + a_3 b_2 + r_2 \qquad (31)$$

$$z := a_3 b_0 + a_0 b_3 + r_3 \qquad (32)$$

$$c_3 := x + y + z + a_3 b_3 \qquad (33)$$

上記の手順には 2 つの問題点が存在する．

(1) 列挙された式の中に $a_0 b_2 + a_2 b_0, a_1 b_3 + a_3 b_1$ の組が無い．
(2) パーティ $P_1$ がデータ $a_3 b_3$ を持っていないため $c_3$ が計算できない．

この問題点から 3 out of 4 では乗算は不可能である．

### 3.3 k out of n への拡張

2 out of 3 から任意の k out of n への拡張をおこなう．

#### 3.3.1 k out of n

あるデータ $a$ に対する k out of n の場合の分散プロトコルは次の手順でおこなわれる．なお，一つのシェアが持つデータの個数は $n - k + 1$ 個である．

(1) $a_0, \ldots, a_{n-1}$ をランダムに設定する．
(2) $a_n := a - a_0 - \cdots - a_{n-1}$ を計算する．
(3) シェアを $i = 0, \ldots, n-1$ として $[a]_i := (a_i, \ldots a_{n-k+i})$ を作成する．
(4) 作成したシェア $[a]_i$ をパーティ $P_i$ へと送信する．

なお，シェア $[a]_i$ のデータの添字は記述の簡略化のために $i \pmod n$ と余剰をとってあるものとする．

#### 3.3.2 k out of n の場合の乗算プロトコル

乗算プロトコルでは先に述べた分散プロトコルを用いてデータ $a, b$ が分散されている状態で説明する．データ $a, b$

は式 (34), (35) のように分割されている．

$$a_n := a - a_0 - \cdots - a_{n-1} \tag{34}$$
$$b_n := b - b_0 - \cdots - b_{n-1} \tag{35}$$

そして式 (36), (37), (38) のようなシェア，パーティへと分散されている．

$$[a]_i := (a_i, \ldots, a_{n-k+i}) \tag{36}$$
$$[b]_i := (b_i, \ldots, b_{n-k+i}) \tag{37}$$
$$P_i := ([a]_i, [b]_i) \tag{38}$$

以上のような状態で $a \times b := c_0 + \cdots + c_n$ となるようなシェア $[c]_i$ を作成する．

(1) パーティ $P_0$ は次の操作をおこなう．
  (a) $r_1, \ldots, r_n, c_0, \ldots, c_{n-2}$ をランダムに設定する．
  (b) $c_{n-2} := (a_0 + \cdots + a_{n-k+0})(b_0 + \cdots + b_{n-k+0}) - r_1 - \cdots - r_n - c_0 - \cdots - c_{n-3}$ を計算する．
  (c) パーティ $P_1, \ldots, P_{n-1}$ にそれぞれ $(r_1, c_1, \ldots c_{n-k+1}), \ldots, (r_n, c_{n-1}, \ldots, c_{n-k+n})$ を送信する．
  (d) $[c]_0 := (c_0, \ldots, c_{n-k+0})$ とする．

(2) パーティ $P_1, \ldots, P_n$ はそれぞれ自身が持っているパーティ内の要素 $p_i$ を用いて次の操作をおこなう．
  (a) パーティ $P_i$ は $S_i := p_i - r_i$ を計算して $P_1, \ldots, P_n$ へと送信する．
  (b) それぞれ送信されてきた $S_1, \ldots, S_n$ を用いて $c_{n-1} := S_1 + \cdots + S_n + a_k b_k + \cdots + a_{n-1} b_{n-1}$ を計算する．
  (c) $[c]_1 := (c_1, \ldots, c_{n-k+1}), \ldots,$ $[c]_{n-1} := (c_{n-1}, \ldots, c_{n-k+n})$ とする．

ここで以上の手順の中に出てきた式を列挙する．

$$c_{n-1} := (a_0 + \cdots + a_{n-k+0})(b_0 + \cdots + b_{n-k+0}) - r_1 - \cdots - r_n - c_0 - \cdots - c_{n-2} \tag{39}$$
$$S_i := p_i - r_i \tag{40}$$
$$c_{n-1} := S_1 + \cdots + S_n + a_k b_k + \cdots + a_{n-1} b_{n-1} \tag{41}$$

この時確かに $a \times b = c_0 + \cdots + c_{n-1}$ である．理由を式 (42) に述べる，

$$\begin{aligned}
&c_0 + \cdots + c_{n-1} \\
&= (a_0 + \cdots + a_{n-k+0})(b_0 + \cdots + b_{n-k+0}) \\
&\quad - r_1 - \cdots - r_n \\
&\quad + S_1 + \cdots + S_n + a_k b_k + \cdots + a_{n-1} b_{n-1} \\
&= (a_0 + \cdots + a_{n-k+0})(b_0 + \cdots + b_{n-k+0}) \\
&\quad - r_1 - \cdots - r_n \\
&\quad + p_1 + \cdots + p_n + r_1 + \cdots + r_n \\
&\quad + a_k b_k + \cdots + a_{n-1} b_{n-1} \\
&= (a_0 + \cdots + a_{n-k+0})(b_0 + \cdots + b_{n-k+0}) \\
&\quad + p_1 + \cdots + p_n + a_k b_k + \cdots + a_{n-1} b_{n-1} \\
&= (a_0 + \cdots + a_{n-k+0})(b_0 + \cdots + b_{n-k+0}) \\
&\quad + a_1 b_{n-k+1} + a_{n-k+1} b_1 + \ldots \\
&\quad + a_{n-k+1} b_0 + a_0 b_{n-k+1} \\
&\quad + a_k b_k + \cdots + a_{n-1} b_{n-1} \\
&= (a_0 + \cdots + a_{n-1})(b_0 + \cdots + b_{n-1}) \\
&= a \times b
\end{aligned} \tag{42}$$

以上により条件付き k out of n 場合において乗算は可能である．

### 3.4 乗算可能な k out of n の条件について

k out of n において乗算可能な場合について考える．

乗算のシェア $c_0, \ldots c_{n-1}$ を作成する際には $(a_0 + \cdots + a_{n-1})(b_0 + \cdots + b_{n-1})$ が必要となる．これを展開すると式 (43) のようになる．

$$\begin{aligned}
a \times b &= \{a_0 + \cdots + a_{n-1}\}\{b_0 + \cdots + b_{n-1}\} \\
&= \left\{ \begin{array}{l} a_0 b_0 + \cdots + a_0 b_{n-1} \\ \vdots \\ a_{n-1} b_0 + \cdots + a_{n-1} b_{n-1} \end{array} \right\}
\end{aligned} \tag{43}$$

分割されたデータの代表を $a_i, b_j$ と呼ぶことにする．

k out of n 乗算プロトコルにおいて $P_1, \ldots, P_{n-1}$ のパーティは計算の途中結果 $S_1, \ldots S_{n-1}$ を他のパーティへと送信する必要がある．

また，パーティは他のパーティが持っている $a_i, b_j$ を受け取ることは許されない．つまり，乗算プロトコルではパーティ $P_0, \ldots P_{n-1}$ 内にあるシェアのデータのみを用いて式 (43) にて挙げられた $a_0 b_0 + \cdots + a_{n-1} b_{n-1}$ の全ての組み合わせを作成する必要がある．

ここで式 (43) 内の $a_i b_j$ の添字間の距離 $|i - j|$ を取るとその最大値は $n/2$ である．そして，一つのシェアに含まれるデータの個数は $n - k + 1$ 個である．

したがって乗算プロトコルが可能な場合は式 (44) に示すようになる．

Vol.2016-DPS-166 No.5
Vol.2016-CSEC-72 No.5
2016/3/3

$$
\begin{aligned}
n/2 &< n-k+1 \\
n &< 2n-2k+2 \\
-n &< -2k+2 \\
-n &< -2(k-1) \\
n &> 2(k-1) \qquad (44)
\end{aligned}
$$

## 4. シェアに乱数を含めることの評価

文献 [10] で示されている各パーティが協調して計算する乗算での手続き Mul には，乱数が含まれている．これによって，完全秘匿性が保証されている．

一方で，元々の数値とは関係の無い数値を含むため，計算が増えている．データ量が多い場合には，この増加分が原因となって計算速度が若干低下すると思われる．

また，乱数を含めなくてもさほど秘匿性には問題が無いと考える．理由を乗算プロトコル 2.2 の際に各パーティが持っている値の式 (3)，(4)，(5) と，計算の途中に他のパーティへと送信される途中結果 $y, z$ の式 (7)，(8) から説明する．

$$[a]_0 := (a_0, a_1), [b]_0 := (b_0, b_1), P_0 := ([a]_0, [b]_0) \quad (3)$$

$$[a]_1 := (a_1, a_2), [b]_1 := (b_1, b_2), P_1 := ([a]_1, [b]_1) \quad (4)$$

$$[a]_2 := (a_2, a_0), [b]_2 := (b_2, b_0), P_2 := ([a]_2, [b]_2) \quad (5)$$

$$y := a_1 b_2 + a_2 b_1 + r_1 \quad (7)$$

$$z := a_2 b_0 + a_0 b_2 + r_2 \quad (8)$$

ここで，例としてパーティ $P_1$ から $P_2$ へと送信される $y$ に乱数が含まれない式 (45) を考える．

$$y' := a_1 b_2 + a_2 b_1 \quad (45)$$

$P_1$ から $y'$ が送信されてくる場合に $P_2$ がもし元のデータ $a$ を復元しようとすると，式 (5) より $a_1$ の値を知ることができれば $a$ の値を復元できる．

式 (5)，(45) より $P_2$ が知り得るものを用いて $a_1 + \ldots$ という式を作ると式 (46) のようになる．

$$a_1 + \frac{a_2 b_1}{b_2} \quad (46)$$

ここで，式 (46) の第二項の成分 $a_2 b_1 / b_2$ に含まれる値 $a_2, b_1, b_2$ は分散プロトコルにより作成される時に一様乱数となっている．

このためパーティ $P_2$ は $a_1$ を完全に復元できないため元のデータ $a$ も復元することはできない．同様にパーティ $P_1$ が $a, b$ の値を，パーティ $P_2$ が $b$ の値を復元することはできない．

この仮説に基づいて，数値計算による実験をおこなったので，以下に報告する．

数値計算の方法は 2 out of 3 によって分散されたとした $1 \times 10^3, 2 \times 10^3, 5 \times 10^3, \ldots, 1 \times 10^5, 2 \times 10^5$ 個のデータ $a, b$ を $r_n$ を使用した乗算プロトコルと $r_n$ を使用しなかった乗算プロトコルで実時間の速度を 10 回測定し平均値を取った．

**表 1** 測定環境

| | |
|---|---|
| PC | 1 台 |
| CPU | Intel Xeon E3-1290 V2 @ 3.70GHz |
| RAM | 32GB |
| OS | CentOS 7.1.1503 |
| 言語 | GNU bash version 4.2.46(1) |

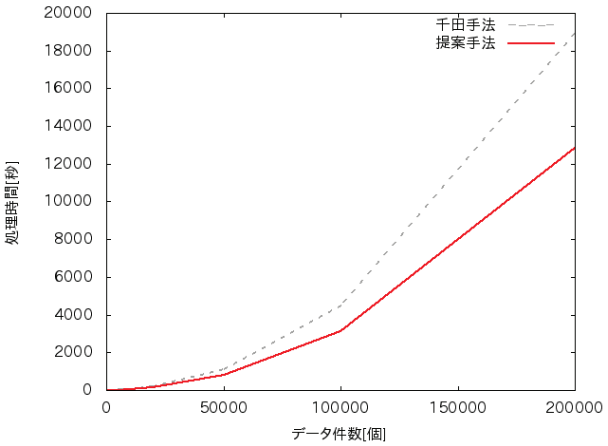


**図 1** 速度比較

測定結果より，データが 5 万個まではあまり速度に開きは無いが，それ以降は目に見える変化があり 20 万個の場合にはおよそ 1.5 倍の速度となった．

## 5. (2 out of 3) out of n について

(2 out of 3) out of n とは「2 out of 3 の秘密乗算をする場合には 3 台のパーティが健在でなければならない」というパーティの稼働率を考慮した設計である．乗算計算の面から見ると，全てのパーティが稼動している必要があり，結果として信頼性が低いと取れる点を防ぐために導入する考えである．

具体的には n 個のパーティのうち利用可能な 3 個のパーティの組み合わせを見つけその後そのパーティの組み合わせで 2 out of 3 の乗算計算をおこなうということである．これにより，乗算が可能な可能性が向上する．つまり信頼性が上がるという考えである．

さて，(2 out of 3) out of n を実現するにはどのようなデータ配置にすれば良いか．ということを考えてみると，まず， 2 out of 3 が基本であるため元のデータは 3 分割される．シェアには分割されたデータが 2 個ずつ配置される．

例えばある 1 つのパーティが利用不能でも 2 out of 3 の乗算が確実にできるようにするには，シェアを 6 つのパー

ティに分散配置する必要がある．

これはあるパーティ $P_i$ を保持するパーティのミラー $P_i'$ を全てのパーティに対して作成しなければならないためである．

## 6. おわりに

本稿では，n=3 に限定されていた手法 [10] を k out of n へ拡張するための検討と，計算処理時間の短縮を狙って，既存手法 [10] で用いられていた乱数を用いない場合の実行速度の比較と検討を述べた．

検討の結果，$n > 2(k-1)$ の条件において，k out of n の乗算プロトコルの設計は可能であることを示した．また，乱数を用いない場合は処理時間が短かいことを示した．

今後，より一般化するための方式の定式化と，実用的範囲での処理時間の軽減を目指して研究を進める予定である．

### 参考文献

[1] Ben-Or, M., Goldwasser, S. and Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation, *Proceedings of the twenti-eth annual ACM symposium on Theory of computing*, ACM, pp. 1–10 (1988).

[2] Bogdanov, D., Laur, S. and Willemson, J.: Sharemind: A framework for fast privacy-preserving computations, *Computer Security-ESORICS 2008*, Springer, pp. 192–206 (2008).

[3] 高橋　慧，小林士郎，岩村恵市：記憶容量削減と計算量的安全性および復元の独立性を実現するクラウドに適した秘密分散法，情報処理学会論文誌，Vol. 54, No. 9, pp. 2146–2155（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009605620/⟩ (2013).

[4] 渡辺泰平，岩村恵市：秘密分散法を用いたサーバ台数変化がない乗算手法，研究報告コンピュータセキュリティ（CSEC），Vol. 2013, No. 3, pp. 1–6（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009634018/⟩ (2013).

[5] 宮西洋太郎，韓　嘯公，北上眞二，金岡　晃，佐藤文明，浦野義頼，白鳥則郎：クラウドサービス利用者の安心感を高める簡易的秘密計算法の提案，ソフトウェアインタプライズモデリング研究会 (SWIM)， Vol. 114, pp. 19–24 (2014).

[6] 中田亮太, 仲地孝之：秘密分散法を利用した乗算が可能な秘密計算の提案 (情報セキュリティ, ライフログ活用技術, ライフインテリジェンス, オフィス情報システム, 一般), 電子情報通信学会技術研究報告. LOIS, ライフインテリジェンスとオフィス情報システム，Vol. 112, No. 306, pp. 15–20（オンライン）, 入手先 ⟨http://ci.nii.ac.jp/naid/110009642190/⟩ (2012).

[7] 高橋　慧，岩村恵市：クラウドコンピューティングに適した秘密分散法，電子情報通信学会技術研究報告. EMM, マルチメディア情報ハイディング・エンリッチメント， Vol. 112, No. 226, pp. 11–16（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009636711/⟩ (2012).

[8] 宮西洋太郎，韓　嘯公，金岡　晃，佐藤文明，北上眞二，浦野義頼，白鳥則郎：セキュアマルチパーティ秘密計算法におけるユーザ安心感定量化の試み～情報システムの「安信性理論」の確立を目指して～，情報処理学会研究報告. マルチメディア通信と分散処理研究会報告， Vol. 2015, No. 41, pp. 1–6（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009884233/⟩ (2015).

[9] 韓　嘯公，宮西洋太郎，北上眞二，浦野義頼，白鳥則郎：クラウドサービス利用者の安心感を高める軽量秘密計算法の実クラウドにおける実験，研究報告コンピュータセキュリティ（CSEC），Vol. 2015, No. 42, pp. 1–7（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/170000093496/⟩ (2015).

[10] 千田浩司，五十嵐大，濱田浩気，高橋克巳：エラー検出可能な軽量 3 パーティ秘匿関数計算の提案と実装評価，情報処理学会論文誌，Vol. 52, No. 9, pp. 2674–2685 (2011).