XOR 法の拡張による効率的な秘匿計算の実現

鴇田 恭平[†1]　岩村 惠市[†1]

概要：Shamir 法における秘匿演算手法は広く知られているが，栗原らが提案した XOR 法のような乱数の加減算による秘密分散手法に対しては限られた場合での秘匿加減算以外実現されていない．そこで，著者らは第 77 回 CSEC 研究発表会において，XOR 法を拡張して秘密情報を多値のデータとして扱う多値化法を提案して、秘匿四則演算を実現できる手法を提案した．これは，神宮らが提案した次数を変化させない四則演算手法（TUS 方式）を用いるが，この方式では、Shamir 法に TUS 方式を適用した場合に比べ、多くの記憶量を要する問題があった．そこで，本論文では方式を改良し，その安全性の証明と，改良方式が効率的であることを示す．

キーワード：秘密分散法，秘匿計算，準同型性

Efficient Realization of Secure Computation by Extension of XOR Scheme

Kyohei Tokita[†1]　Keiichi Iwamura[†1]

Abstract: Secure computation methods for Shamir's Secret Sharing Scheme(SSSS) is well-known. On the other hand, only secure addition and subtraction based on secret sharing schemes, which distributes a secret by adding random numbers, such as the XOR scheme proposed by Kurihara et al. is realized under limited conditions. Therefore, we extended the XOR scheme and proposed the multi-value scheme, which deals with the secret as an e-ary number for realization of secure computation at CSEC77. It realized by using the TUS method proposed by Shingu et al., which realizes secure computation for SSSS without the increase of the number of the polynomial. However, there is a problem in applying the TUS method to the multi-value scheme because the method needs more storage than the TUS method for SSSS. In this paper, we therefore improve it, prove the security of the proposed scheme and show that the proposed scheme can process more efficiently than the existing secure multiplication method.

Keywords: Secret sharing scheme, Secure computation, homomorphisms

1. はじめに

近年，ビッグデータという言葉が注目されている．その大きな特徴は，単純に膨大なデータ量を持つことだけでなく，データの多様性，発生の速さ，正確さも持つことである．ウェブサービス分野では現在既に活用が進んでおり，さらに各データを連携させることでさらなる付加価値の創出も期待されるため，現在の研究における一つのキーワードとなっている[1]．一方でプライバシー情報や秘密情報の取り扱いがビッグデータにおける大きな課題となっている[2]．ビッグデータの活用を推進するには，この問題を解消しつつデータ分析結果を享受できる対策を講じる必要がある．その対策の一つに，プライバシー保護データマイニングがある[3]．プライバシー保護データマイニングは，プライバシーを保護しつつ，データから有用な情報の抽出を試みるための技術の総称であり，データを蓄積し，それを解析した結果のみを表示することで，データの内容を秘匿しながら解析できるようにする秘匿演算によって実現できるが，その秘匿演算を実現する一手法として Shamir によって提案された(k,n)閾値秘密分散法(以降 Shamir 法)[4]が上げられる．しかし，Shamir 法は k が大きい場合の秘密情報の分散・復元時の計算負荷が問題となっていた．

それに対して，栗原らが XOR のみで秘密情報の分散と復元を実現できる高速な XOR 法[5]を提案したが，XOR 法は秘密情報をビット列として扱うため，秘匿演算に対応できない．それを解決するため，XOR 法を拡張して，秘密情報を多値の値として扱い秘匿四則演算に対応できる多値化法を提案し，その安全性を証明する．

また，XOR 法や多値化法のように乱数の加減算のみで秘密情報を分散する手法に対しては秘匿加減算以外実現されておらず，乗算に対する準同型性を持たない場合が一般的である．そこで，乗法準同型性を持たない多値化法に対しても秘匿四則演算を実現できる手法を提案する．これは，神宮らが提案した次数を変化させずに秘匿四則演算を行える手法（以降, TUS 方式）[8]を適用することで実現するが，TUS 方式は，秘密情報に 0 を含むとそれが漏洩するため，秘密情報に 0 を含まないという条件がある．また，多値化法は n が 2 を超えると XOR 法と同様に秘密情報を分割して秘匿する．よって，TUS 方式を多値化法にそのまま適用すると，分割された部分秘密情報に 0 を含むとその部分秘

†1 東京理科大学
Tokyo University of Science

密情報が0であることが漏洩するという問題が発生する．0を含まない情報は，医療データなどには多く存在する．例えば，脈拍や血圧などは全て正の値であり，0 は死亡していることを意味しており，医療的な統計計算には用いられないことが多い．よって，秘密情報自体に0を含まないことはその有用性を大きく損なわない．それに対して，部分的に0を含まない情報はほとんど存在しないと言ってよく，多値化法に TUS 方式をそのまま適用することは大きな問題を生じさせる．

また，XOR 法では前述のように n>2 の場合，秘密情報を分割するが，生成された分散値のサイズも小さくなり，それを連結した分散値は，元の秘密情報のサイズと等価とできた．多値化法も上記の拡張ではそれと同様になるが，著者らが第 77 回 CSEC 研究発表会で発表した，多値化法に TUS 方式を適用した方式[12]で秘匿演算を行うと，分割された秘密情報を秘匿したままで演算処理をされるため，あらかじめ分散値のサイズを復元結果に対応させて大きくとる必要がある．すなわち，TUS 方式を多値化法にそのまま適用すると計算量は小さくなるが，記憶量・通信量が大きく増大するという問題があった．

そこで，上記の問題を解決するように方式を改良し，その安全性を証明する．この改良方式が Shamir 法に TUS 法を適用した秘匿計算法と比較して，記憶量・通信量を大きく増大させずに計算量を大きく削減できることを示す．

よって，本論文では以下を達成する．

①XOR 法を拡張し，秘密情報を多値の値として扱える多値化法を提案し，その安全性を示す．

②乗法準同型性を持たない多値化法に対して，秘匿四則演算が実現できる手法を示す．

③記憶量・通信量の増加を抑え，計算量を大きく削減して高速化を実現する秘匿計算法を提案する．

ここで本論文の構成を示す．2 章に従来方式として，著者らが提案した多値化法とそれによる秘匿演算法を説明し，3 章では，XOR 法を拡張して多値の値を扱う多値化法を示し，その安全性を示す．4 章では TUS 方式を適用して，乗法準同型性をもたない多値化法に対して秘匿四則演算ができる手法を示す．5 章では多値化法に TUS 方式を適用したときに生じる問題点を議論してそれを解決する改良方式を提案する．6 章ではその改良方式が Shamir 法に TUS 方式を適用した秘匿演算に対して，記憶量・通信量を大きく増加させず，計算量を削減して高速化を実現することを示す．

2. 従来手法

2.1 多値化法

2 章で示された XOR 法は秘密情報をビット列として扱うため，秘匿演算を実現できないという問題点を持つ．また，XOR 法における秘密情報を多値の数値に置き換えた手法も提案されている[7]が，k=2 に限定されるなどいくつかの制限があった．また，[7]を用いて秘匿乗算を含む秘匿四則演算法を実行する手法は提案されていない．そこでまず，XOR 法を拡張して n≧k に対して一般的に適用できる手法（以降，多値化法）を提案し，その安全性を示す．

2.1.1 記号の定義

n :分散数

k :閾値

i :ユーザ番号$\left(i\in GF(n_p)\right)$

j :部分分散情報の番号$\left(j\in GF(n_p-1)\right)$

n': $n'\geq n$ を満たす素数

N :自然数の集合

P :n 人のユーザの集合

D :分散情報の計算・配布を行うディーラ

S :秘密情報

S_x :部分秘密情報$(1\leq x\leq n_p-1, S_0\in\{0\}^d)$

r^{α}_{β} :乱数$(0\leq\alpha\leq k-2, 0\leq\beta\leq n_p-1)$

W_i :ユーザ P_i に配布される分散情報

$W_{(i,j)}$:ユーザ P_i に配布される部分分散情報

p : p≥e を満たす素数

$GF(n')$:$GF(n')=\{0,1,\ldots,n'-1\}$

本章の(k,n)閾値秘密分散法のアルゴリズムは，明示しない限り p を法としたものとし，希望する分散数 n が合成数である場合，(k,n')閾値秘密分散法の中から分散情報を n 個用いることで，(k,n)閾値秘密分散法を実現する．以降での提案方式の説明では，簡単化のため n=n'として説明する．

2.1.2 分散アルゴリズム

XOR 法は，秘密情報 S をビット列として扱う．これは S を 2 進数の数値として扱っていると解釈できるが，処理を XOR で行うため分散値を用いた四則演算に対応できない．そこで，XOR 法を一般化して S を e 進数の数値とみなして分割し，XOR を p を法とする加算に置き換えて後述する秘匿演算に対応できるように拡張する．

入力:S

出力:$W_i(i=0,\cdots,n-1)$

(1)D は秘密情報 S を e 進数の数とみなし，n-1 等分して $S_1,S_2,\ldots,S_{n-1}$ を得る．ここでは，簡単のために以下のように表せるとする．ただし$S_0=0$とする．

$$S=S_1\cdot e^{n-2}+S_2\cdot e^{n-3}\cdots+S_{n-1}\cdot e^0=\sum_{i=1}^{n-1}S_i\cdot e^{n-1-i}$$

$$S_{(e)}=S_1\|S_2\|\cdots\|S_{n-1}$$

(2)D は $GF(p)$ 上の乱数r^{α}_{β}を全て独立に(k-1)n-1 個生成する．

$$r^0_0,r^0_1,\cdots,r^0_{n-2},r^1_0,\cdots,r^1_{n-2},r^1_{n-1},\cdots,r^{k-2}_0,\cdots,r^{k-2}_{n-1}$$

(3)D は部分分散情報$W_{(i,j)}$を p を法として以下の式により $0\leq i\leq n-1, 0\leq j\leq n-2$ においてそれぞれ生成する．ただし，ここでの添え字は n'を法として計算される．

$$W_{(i,j)}=S_{j-i}+\left(\sum_{h=0}^{k-2} r_{h,i+j}^{h}\right)$$

(4)D は $0\le i\le n-1$ において各部分分散情報 $W_{(i,0)},W_{(i,1)},\cdots,W_{(i,n-2)}$ を連結して分散情報 W_i を生成し各ユーザに配布する.

$$W_i=(W_{(i,0)},W_{(i,1)},\ldots,W_{(i,n-2)})$$

2.1.3 復元アルゴリズム

入力:$W_i(i=t_0,\cdots,t_{k-1})$

出力:S

(1)k 個の集まった分散情報から全ての部分分散情報を取り出す.

$$W_{t_0}\rightarrow W_{(t_0,0)},W_{(t_0,1)},\cdots,W_{(t_0,n-2)}$$
$$\vdots$$
$$W_{t_{k-1}}\rightarrow W_{(t_{k-1},0)},W_{(t_{k-1},1)},\cdots,W_{(t_{k-1},n-2)}$$

(2)集まった全ての各部分分散情報を以下のように表し, kn-2 元のベクトル $V_{(t_i,j)}$ を生成し, 部分分散情報に含まれている $R_{(k,n)}$ の成分に対応した部分は 1, その他は 0 とする.

部分分散情報 $W_{(t_i,j)}$ の場合

$$W_{(t_i,j)}=V_{(t_i,j)}\cdot R_{(k,n)}$$

$$R_{(k,n)}=\left(r_0^0,\cdots,r_{n-2}^0,r_0^1,\cdots,r_{n-1}^1,\cdots,r_0^{k-2},\cdots,r_{n-1}^{k-2},S_1,\cdots,S_{n-1}\right)^T$$

(3)(2)で集まった $V_{(t_0,0)},\cdots,V_{(t_{k-1},n-2)}$ の k(n-1)個のベクトルから以下の 2 進数の{k(n-1)×(kn-1)}の行列 $M_{(t_0,\cdots,t_{k-1})}^{(k,n)}$ を生成する.

$$M_{(t_0,\cdots,t_{k-1})}^{(k,n)}=\left(V_{(t_0,0)},\cdots,V_{(t_0,n-2)},\cdots,V_{(t_{k-1},0)},\cdots,V_{(t_{k-1},n-2)}\right)^T$$

(4)集まったすべての部分分散情報を k(n-1)元のベクトル $W_{(t_0,\cdots,t_{k-1})}$ と表す.

$$W_{(t_0,\cdots,t_{k-1})}=\left(W_{(t_0,0)},\cdots,W_{(t_0,n-2)},\cdots,W_{(t_{k-1},0)},\cdots,W_{(t_{k-1},n-2)}\right)^T$$

$$W_{(t_0,\cdots,t_{k-1})}=M_{(t_0,\cdots,t_{k-1})}^{(k,n)}\cdot R_{(k,n)}$$

ここで, 行列 $M_{(t_0,\cdots,t_{k-1})}^{(k,n)}$ を Gauss-Jordan の消去法(掃き出し法)を用いて対角化処理を行う. これによって, 全ての部分秘密情報に該当する部分を求めて以下を得る.

$$S_1,S_2,\ldots,S_{n-1}$$

(5)全ての部分秘密情報を以下の式に代入して秘密情報を復元する.

$$S=S_1\cdot e^{n-2}+S_2\cdot e^{n-3}\cdots+S_{n-1}\cdot e^0=\sum_{i=1}^{n-1} S_i\cdot e^{n-1-i}$$

2.2 多値化法を用いた秘匿四則演算

著者らは, 第 77 回 CSEC 研究発表会において, 乗法準同型性を持たない多値化法に, 神宮らが提案した次数を変化させずに秘匿四則演算を行える手法（TUS 方式）を適用することで秘匿四則演算を実現させた. ただし, 秘密情報を部分秘密情報に分割した際に部分秘密情報が 0 となる恐れがあり, TUS 方式を適用する条件である,「秘密情報が 0 でない」に対応できないため, それに対応するための手法を提案した.

ただし, ここでは, n'及び $e^{n'+2}$ より大きな素数 p を法とする. ここでも, TUS 方式と同様に, 分散する秘密情報は 0 でないとする.

2.2.1 記号の定義

ここで新たに示す記号以外は 2.1.1 と同様である。

$\overline{[x]}_j^{(p)}$:値xに対するサーバ P_j の保持する素数 p を法とした分散値. 3.2 における部分分散情報を連結した W_j に相当する.

$\overline{[x]}_j^{(q)}$:値xに対するサーバ P_j の保持する素数 q を法とした分散値. 3.2 における部分分散情報を連結した W_j に相当する.

$[x]_j$:値xに関連してサーバ P_j の保持する分散値集合

以下において $i\in GF(k),j\in GF(n')$ とする.

2.2.2 分散アルゴリズム

入力:S

出力:$[S]_j=\left(\overline{[\alpha S]}_j^{(q)},\overline{[\alpha_0]}_j^{(p)},\ldots,\overline{[\alpha_{k-1}]}_j^{(p)}\right)$

(1)D は秘密情報 S を e 進数で表し, n-1 等分して $S_1,S_2,\ldots,S_{n-1}$ で表す. ここでは簡単のため, 以下のように表せるものとする. また, $S_0=0$ とする.

$$S=S_1\cdot e^{n-2}+S_2\cdot e^{n-3}\cdots+S_{n-1}\cdot e^0=\sum_{i=1}^{n-1} S_i\cdot e^{n-1-i}$$

(2)D は k 個の乱数 α_i を生成し, 3.2 の手順で分散する. すなわち, p を法として $\overline{[\alpha_i]}_j^{(p)}$ が計算される.

(3)q を法として, α_i の積 α を計算する.

(4)q を法として, (1)で生成した部分秘密情報に(3)で得た α を乗じ, $\alpha S_1,\ldots,\alpha S_{n-1}$ を得る.

(4)3.2(2)以降の p を q と置き換えて 3.2 の手順で分散を行う. その際, $S_1,\ldots,S_{n-1}$ の代わりに $\alpha S_1,\ldots,\alpha S_{n-1}$ を用いて, $\overline{[\alpha S]}_j^{(q)}$ が計算される.

2.2.3 復元アルゴリズム

入力:$[S]_j$

出力:S

(1)3.3 の手順に従い, k 個の分散情報 $\overline{[\alpha S]}_j^{(q)}$ から全ての部分分散情報を復元し, $\alpha S_1,\ldots,\alpha S_{n-1}$ を得る.

(2)q を法として, 復元した全ての部分秘密情報を以下の式に代入して秘匿化秘密情報を復元する.

$$\alpha S_1 \cdot e^{n-2} + \alpha S_2 \cdot e^{n-3} + \cdots + \alpha S_{n-1} \cdot e^0 = \sum_{i=1}^{n-1} \alpha S_i \cdot e^{n-1-i} = \alpha S$$

(3)分散されたα_iをすべて復元し，以下を計算する．

$$\alpha S \prod_{i=0}^{k-1} \alpha_i^{-1} = S$$

2.2.4 秘匿乗除算

秘密情報 S と S'に対して 4.2.2 で生成した分散値$[S]_j$及び$[S']_j$をサーバP_jはもつとする．

入力:$[S]_j$,$[S']_j$

出力:$[SS']_j$ $\left([S'/S]_j\right) := (\overline{[\beta(\alpha S)^{\pm 1} S']}_j^{(q)}, \overline{[\alpha_0^{\pm 1}\beta_0]}_j^{(p)}, \ldots, \overline{[\alpha_{k-1}^{\pm 1}\beta_{k-1}]}_j^{(p)})$

(1)サーバ P_0 は分散値$\overline{[\alpha S]}_j^{(q)}$を k 個集めて，3.3(1)(2)の手順に従いαSを復元し，全サーバに送る．

(2)サーバP_jは$\overline{[\beta S']}_j^{(q)}$と$\alpha S^{\pm 1}$を掛け合わせ$\overline{[\beta(\alpha S)^{\pm 1} S']}_j^{(q)}$を生成する．

$$\overline{[\beta(\alpha S)^{\pm 1} S']}_j^{(q)} = (\alpha S)^{\pm 1} \overline{[\beta S']}_j^{(q)}$$

(3)サーバP_i(i=0,1,⋯,k-1)は指定された i に対する$\overline{[\alpha_i]}_j^{(p)}$,$\overline{[\beta_i]}_j^{(p)}$を k 個集め，$\alpha_i$,$\beta_i$を復元し，q を法として$\alpha_i\beta_i\left(\beta_i/\alpha_i\right)$を計算する．

(4)k 台のサーバP_iは，$\alpha_i\beta_i$を 3.2 の手順で n 台のサーバに分散する．

2.2.5 秘匿加減算

秘密情報 S と S'に対して 4.2.2 で生成した分散値集合$[S]_j$及び$[S']_j$をサーバP_jはもつとする．

入力:$[S]_j$，$[S']_j$

出力:$[S\pm S']_j = (\overline{[\gamma(S\pm S')]}_j^{(q)}, \overline{[\gamma_0]}_j^{(p)}, \ldots, \overline{[\gamma_{k-1}]}_j^{(p)})$

(1)サーバP_i(i=0,1,⋯,k-1)は指定された i に対する$\overline{[\alpha_i]}_j^{(p)}$,$\overline{[\beta_i]}_j^{(p)}$を k 個集め，$\alpha_i$,$\beta_i$を復元し，乱数$\gamma_i$を生成して，q を法として$\gamma_i/\alpha_i$,$\gamma_i/\beta_i$を計算し，1 つのサーバに送信する．ここでは P_0 とする．

(2)サーバ P_0 はそれぞれ k 個のγ_i/α_i,γ_i/β_iから，以下の式より q を法としてγ/α,γ/βを計算し，全サーバに送る．

$$\frac{\gamma}{\alpha} = \prod_{i=0}^{k-1} \frac{\gamma_i}{\alpha_i} \qquad \frac{\gamma}{\beta} = \prod_{i=0}^{k-1} \frac{\gamma_i}{\beta_i}$$

(3)サーバP_jは以下の演算をして,$\overline{[\gamma(S\pm S')]}_j^{(q)}$を生成する

$$\overline{[\gamma(S\pm S')]}_j^{(q)} = \frac{\gamma}{\alpha}\overline{[\alpha S]}_j^{(q)} \pm \frac{\gamma}{\beta}\overline{[\beta S']}_j^{(q)}$$

(4)サーバ P_i は，3.2 の手順でγ_iを分散する．

3. 提案方式

3.1 従来方式の問題点とその改良

著者らは，前述したように，第 77 回 CSEC 研究発表会において，乗法準同型性を持たない多値化法を用いて，秘匿四則演算が実現できることが示された．特に，n=k=2 の場合秘密情報が分割されないため，下記問題点 1,2 は発生せず，高速かつ効率的な秘匿四則演算が可能である．また，従来方式の安全性は TUS 方式をそのまま適用しているため，TUS 方式の安全性と等価と考えられる．ただし，多値化法は n=k=2 以外では秘密情報を分割するため，以下の 2 つの問題が発生する．

問題点 1:

2.2.4(1)において 1 度 αS を復元するが，秘密情報が分割されているため，具体的には n-1 個の$\alpha S_1,\ldots,\alpha S_{n-1}$が復元され，それらを合成して αS を計算する．よって，$S_1,\ldots,S_{n-1}$中の S_i が 0 であれば，それに対応する αS_i も 0 になる．よって，その部分秘密情報 S_i の値が漏洩する．

問題点 2:

基本的な多値化法は，秘密情報 S のサイズを Q としたとき，p を法とするサイズ P の$S_1,\ldots,S_{n-1}$に分割して，同じサイズの分散値を得る．すなわち，P(n-1)=Q となる．しかし，秘匿演算を行うため$\alpha S_1,\ldots,\alpha S_{n-1}$を生成して，それを分散するとき q を法として秘密分散する．これによって，連結された分散値のサイズは Q(n-1)となり，元の秘密情報のサイズの n-1 倍になる．そのため必要な記憶量が増大し，それに伴い通信量も増大する．

問題点 1 については，著者らは対処したが，処理が煩雑となり，また，問題点 2 が新たに発生した．これらの問題は秘匿化秘密情報 αS を$\alpha S_1,\ldots,\alpha S_{n-1}$に分割するために生じると考えられる．そこで，αS を分割せずにそのまま用いて，記憶量と通信量の増大を抑えながら，高速処理を実現できるように TUS 法を改良する．記号の定義は下記以外前節と同様である．

$[S]_j^*$：$[S]_j$中の$\overline{[\alpha S]}_j^{(q)}$をαSに変更した分散値集合

3.2 提案方式

3.2.1 分散アルゴリズム

入力:S

出力:$[S]_j^* = \left(\alpha S, \overline{[\alpha_0]}_j^{(p)}, \ldots, \overline{[\alpha_{k-1}]}_j^{(p)}\right)$

(1)D は k 個の乱数α_iを生成し，3.2 の手順で分散する．すなわち，p を法として$\overline{[\alpha_i]}_j^{(p)}$が計算される．

(2)q を法として，α_iの積αと秘密情報 S の積 αS を計算し，全サーバに送信する．

3.2.2 復元アルゴリズム

入力:$[S]_j$*

出力:S

(1)3.3 の手順に従いα_iをすべて復元し，α を計算する．

$$\prod_{i=0}^{k-1}\alpha_i=\alpha$$

(2)秘密情報 S を復元する．

$$S=\alpha S/\alpha$$

3.2.3 秘匿乗除算

秘密情報 S と S'に対して 5.2.1 で生成した分散値$[S]_j$*及び$[S']_j$*をサーバP_jはもつとする．

入力:$[S]_j$*,$[S']_j$*

出力:$*\ [SS']_j^{*}\left([S'/S]_j^{*}\right):=(\beta(\alpha S)^{\pm1}S',\overline{[\alpha_0^{\pm1}\beta_0]}_j^{(p)},\ldots,\overline{[\alpha_{k-1}^{\pm1}\beta_{k-1}]}_j^{(p)})$

(1)サーバ P_j は αS と βS'を用いて q を法とした乗算または除算を行う．

$$\beta(\alpha S)^{\pm1}S'=(\alpha S)^{\pm1}\times\beta S'$$

(2)サーバP_i(i=0,1,⋯,k-1)は指定された i に対する$\overline{[\alpha_i]}_j^{(p)},\overline{[\beta_i]}_j^{(p)}$を k 個集め，$\alpha_i,\beta_i$を復元し，q を法として$\alpha_i\beta_i(\beta_i/\alpha_i)$を計算する．

(3)k 台のサーバP_iは，$\alpha_i\beta_i(\beta_i/\alpha_i)$を 3.2 の手順で n 台のサーバに分散する．

3.2.4 秘匿加減算

秘密情報 S と S'に対して 5.2.1 で生成した分散値集合$[S]_j$*及び$[S']_j$*の他に，1 に対する分散値集合$[1]_j$ と$[1']_j$ が準備されているとする．ただし$[1]_j$と$[1']_j$は以下とする．

$$[1]_j=\left(\overline{[\eta]}_j^{(q)},\overline{[\eta_0]}_j^{(p)},\ldots,\overline{[\eta_{k-1}]}_j^{(p)}\right)$$

$$[1']_j=\left(\overline{[\delta]}_j^{(q)},\overline{[\delta_0]}_j^{(p)},\ldots,\overline{[\delta_{k-1}]}_j^{(p)}\right)$$

ここでは，$[1]_j$ と$[1']_j$ は D が処理に先立って生成するとするため，処理(0)を加える．

入力:$[S]_j$*,$[S']_j$*

出力:$[S\pm S']_j^{*}=(\gamma(S\pm S'),\overline{[\gamma_0]}_j^{(p)},\ldots,\overline{[\gamma_{k-1}]}_j^{(p)})$

(0)D は S=1 として 5.2.1 の分散アルゴリズムで 2 つの 1 に対する分散値$[1]_j$ と$[1']_j$を生成する．ただし，$[1]_j$ と$[1']_j$ は秘密情報が 1 であることを公開できる．

(1)サーバ Pj は αS と$\overline{[\delta]}_j^{(q)}$，βS'と$\overline{[\eta]}_j^{(q)}$の乗算を行う．

$$\overline{[\alpha\delta S]}_j^{(q)}=\alpha S\times\overline{[\delta]}_j^{(q)}$$

$$\overline{[\beta\eta S']}_j^{(q)}=\beta S'\times\overline{[\eta]}_j^{(q)}$$

(2) サーバ P_i(i=0,1,⋯,k-1) は指定された i に対する$\overline{[\alpha_i]}_j^{(p)},\overline{[\beta_i]}_j^{(p)},\overline{[\delta_i]}_j^{(p)},\overline{[\eta_i]}_j^{(p)}$を k 個集め，3.3 の手順に従って$\alpha_i,\beta_i,\delta_i,\eta_i$を復元し，乱数$\gamma_i$を生成して，q を法として$\gamma_i/(\alpha_i\delta_i),\gamma_i/(\beta_i\eta_i)$を計算し，1 つのサーバに送信する．ここでは P_0 とする．

(3)サーバ P_0 はそれぞれ k 個の$\gamma_i/(\alpha_i\delta_i),\gamma_i/(\beta_i\eta_i)$から，以下の式より q を法として$\gamma/(\alpha\delta),\gamma/(\beta\eta)$を計算し，全サーバに送る．

$$\frac{\gamma}{\alpha\delta}=\prod_{i=0}^{k-1}\frac{\gamma_i}{\alpha_i\delta_i}\qquad\frac{\gamma}{\beta\eta}=\prod_{i=0}^{k-1}\frac{\gamma_i}{\beta_i\eta_i}$$

(4)サーバP_jは以下の演算をして,$\overline{[\gamma(S\pm S')]}_j^{(q)}$を生成する

$$\overline{[\gamma(S\pm S')]}_j^{(q)}=\frac{\gamma}{\alpha\delta}\overline{[\alpha\delta S]}_j^{(q)}\pm\frac{\gamma}{\beta\eta}\overline{[\beta\eta S']}_j^{(q)}$$

(5)サーバ P_i は，3.2 の手順でγ_iを分散する．

(6)サーバ P_j は$\overline{[\gamma(S\pm S')]}_j^{(q)}$を P_0 に集めて復元し，復元した γ(S±S')を全サーバに配布する．

3.3 安全性評価

3.3.1 分散・復元における安全性

TUS 方式との違いは，αS を分散せず，そのまま保存する点であるが，秘密情報 S は α がわからなければ漏洩せず，α は全ての α_i(i=0,⋯,k-1)が分からなければ合成できない．よって，αS をそのまま保存し，攻撃者が k-1 台のサーバ情報を知ることができても秘密情報は漏洩しない．

3.3.2 秘匿乗除算における安全性

TUS 法と同様に以下の攻撃者を考える．以下の 3 攻撃者を考慮することで，実用上想定される攻撃に対する耐性を考えるには十分である．ただし，攻撃者 1~3 が知ろうとする情報が得られた場合，攻撃成功となるが，攻撃者はプロトコルに従う passive adversary を想定する．

攻撃者 1:サーバを乗っ取るなどして，k-1 台のサーバが知る情報を知り，それを元に秘匿計算の入力となる秘密情報，または秘匿計算の出力結果を知ろうとする．

攻撃者 2:秘匿計算に関する 1 つの値を入力するものが攻撃者となった場合であり，自らが入力した秘密情報及び秘匿化秘密情報に用いられた乱数を知る．さらに，k-1 台のサーバが知る情報も知ることができ，もう一人の入力者が入力した秘密情報，または秘匿計算出力を知ろうとする．

攻撃者 3:秘匿演算の復元を行うものが攻撃者となった場合であり，k 台のサーバから送られる，秘密情報を復元するために必要な情報を知る．さらに，k-1 台のサーバが知る情報も知ることができ，秘匿計算の入力となった秘密情報を知ろうとする．

ただし，攻撃者 4 として攻撃者が復元者かつ 1 つの値の入力者である場合も考えられるが，2 入力であればどのような秘匿計算法を用いても，攻撃者 4 は秘匿計算結果と自分の入力値からもう一人の入力者の秘密情報を知ることが

できる．ここでは 2 入力の秘匿計算を想定しているため，攻撃者 4 を想定しない．

【攻撃者 1 に対する安全性】

攻撃者 1 は k-1 台のサーバから αS,βS',α_j,β_j(j=0,⋯,k-2)を知ることができる．しかし，下記が成り立つため，攻撃者 1 は各入力 S,S'を知ることはできない．

$$H(\alpha)=H\left(\alpha\middle|\alpha_j(j=0,1,\ldots,k-2)\right)$$
$$H(\beta)=H\left(\beta\middle|\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S)=H\left(S\middle|\alpha S,\beta S',\alpha_j,\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S')=H\left(S'\middle|\alpha S,\beta S',\alpha_j,\beta_j(j=0,1,\ldots,k-2)\right)$$

また，下記が成り立つため秘匿演算結果も知ることはできない．

$$H(SS')=H\left(SS'\middle|\alpha S,\beta S',\alpha_j,\beta_j(j=0,1,\ldots,k-2)\right)$$

【攻撃者 2 に対する安全性】

攻撃者 2 を S の入力者とすると，攻撃者 2 は自分が入力した情報と k-1 台のサーバの情報から α,S,αS,βS',α_j,β_j(j=0,⋯,k-2)を知ることができる．重複を除くと，α,S,βS',β_j(j=0,⋯,k-2)を知る．しかし，α,S,βS'は独立に定められているため下記が成り立ち，攻撃者 2 はもう 1 つの入力 S'を知ることはできない．

$$H(\beta)=H\left(\beta\middle|\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S')=H\left(S'\middle|\alpha,S,\beta S',\beta_j(j=0,1,\ldots,k-2)\right)$$

また，同様に秘匿演算結果も下記から知ることはできない．

$$H(SS')=H\left(SS'\middle|\alpha,S,\beta S',\beta_j(j=0,1,\ldots,k-2)\right)$$

攻撃者が S'の入力者であっても同様である．

【攻撃者 3 に対する安全性】

攻撃者 3 は k-1 台のサーバの情報に加えて，復元に必要な情報を知る．よって，αβ,αS,βS',α_j,β_j(j=0,…,k-2),$\alpha_{k-1}\beta_{k-1}$ を知る．しかし，αβ または $\alpha_{k-1}\beta_{k-1}$ を分離できないため，α と β を個別に知ることができない．よって，以下が成り立ち，攻撃者 3 は各入力 S,S'を知ることはできない．

$$H(S)=H\left(S\middle|\alpha\beta,\alpha S,\beta S',\alpha_j,\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S')=H\left(S'\middle|\alpha\beta,\alpha S,\beta S',\alpha_j,\beta_j(j=0,1,\ldots,k-2)\right)$$

3.3.3 秘匿加減算における安全性

TUS 方式では攻撃者 2,3 に対して乱数 α,β が知られるが，αS,βS'が知られないことが安全性の根拠となっている．それに対して改良方式は αS,βS'が初めから知られているため，そのままでは秘匿加減算において安全ではない．そこで改良方式では，1 に対する分散値集合を用いて安全性を確保している．改良方式では，$[1]_j$ と$[1']_j$ は D が事前に準備するとあるが，D が信頼できる第三者機関であれば問題ない．しかし，必ずしも信頼できる第三者機関が設定できるとは限らないので，$[1]_j$ は$[S]_j$*の入力者が入力し，$[1']_j$ は$[S']_j$*の入力者が入力する場合に対する安全性を，各攻撃者に対して以下に示す．

【攻撃者 1 に対する安全性】

攻撃者 1 は k-1 台のサーバから αS,βS',α_j,β_j,η_j,γ_j,δ_j(j=0,⋯,k-2),γ/(αδ),γ/(βη)を知ることができる．しかし，下記が成り立つため，攻撃者 1 は各入力 S,S'を知ることはできない．

$$H(\alpha)=H\left(\alpha\middle|\alpha_j(j=0,1,\ldots,k-2)\right)$$
$$H(\delta)=H\left(\delta\middle|\delta_j(j=0,1,\ldots,k-2)\right)$$
$$H(\gamma)=H\left(\gamma\middle|\gamma_j(j=0,1,\ldots,k-2)\right)$$
$$H(\eta)=H\left(\eta\middle|\eta_j(j=0,1,\ldots,k-2)\right)$$
$$H(\beta)=H\left(\beta\middle|\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S)=H\left(S\middle|\alpha S,\beta S',\frac{\gamma}{\alpha\delta},\frac{\gamma}{\beta\eta}\alpha_j,\beta_j,\eta_j,\delta_j,\gamma_j(j=0,1,\ldots,k-2)\right)$$
$$H(S')=H\left(S'\middle|\alpha S,\beta S',\frac{\gamma}{\alpha\delta},\frac{\gamma}{\beta\eta}\alpha_j,\beta_j,\eta_j,\delta_j,\gamma_j(j=0,1,\ldots,k-2)\right)$$

また，下記より秘匿演算結果も知ることはできない．

$$H(S\pm S')=H\left(S\pm S'\middle|\alpha S,\beta S',\frac{\gamma}{\alpha\delta},\frac{\gamma}{\beta\eta}\alpha_j,\beta_j,\eta_j,\delta_j,\gamma_j(j=0,1,\ldots,k-2)\right)$$

【攻撃者 2 に対する安全性】

攻撃者 2 を S の入力者とすると，攻撃者 2 は自分が入力した情報と k-1 台のサーバの情報から α,S,αS,βS',η,β_j,δ_j,γ_j(j=0,⋯,k-2),γ/δ,γ/β を知る（重複した情報は省略）,しかし下記が成り立つため，攻撃者 2 はもう 1 つの入力 S'を知ることはできない．

$$H(\delta)=H\left(\delta\middle|\delta_j(j=0,1,\ldots,k-2)\right)$$
$$H(\gamma)=H\left(\gamma\middle|\gamma_j(j=0,1,\ldots,k-2)\right)$$
$$H(\beta)=H\left(\beta\middle|\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S)=H\left(S\middle|\alpha,S,\beta S',\eta,\beta_j,\delta_j,\gamma_j(j=0,1,\ldots,k-2),\gamma/\delta,\gamma/\beta\right)$$
$$H(S')=H\left(S'\middle|\alpha,S,\beta S',\eta,\beta_j,\delta_j,\gamma_j(j=0,1,\ldots,k-2),\gamma/\delta,\gamma/\beta\right)$$

また，下記より秘匿演算結果も知ることはできない．

$$H(S\pm S')=H\left(S\pm S'\middle|\alpha,S,\beta S',\eta,\beta_j,\delta_j,\gamma_j(j=0,1,\ldots,k-2),\gamma/\delta,\gamma/\beta\right)$$

それに対して，$[1']_j(\overline{[\delta]}_j^{(q)})$が導入されない場合，γδ=γ となり，γ/β から β が漏洩し，βS'から秘密情報 S'が漏洩する．これは，S'の入力者が攻撃者 2 でとなっても同様の議論が成り立ち，γη=γ から S が漏洩する．この場合，攻撃者 2 は 2 つの入力を知るために出力も知る．

以上より，$[1]_j$ と$[1']_j$ の導入により，αS,βS'をそのまま保存しても攻撃者 2 に対して安全であることが言える．

【攻撃者 3 に対する安全性】

攻撃者 3 は復元時に得る情報と k-1 台のサーバの情報から，$\alpha S,\beta S',\alpha_j,\beta_j,\eta_j,\delta_j,\gamma_j(j=0,\cdots,k-2),\gamma/(\alpha\delta),\gamma/(\beta\eta),\gamma$ を知る．よって，重複を除くと攻撃者 3 は $\alpha S,\beta S',\gamma,\alpha\delta,\beta\eta,\alpha_j,\beta_j,\eta_j,\delta_j$ (j=0,…,k-2)を知る．しかし，下記が成り立つため，攻撃者 3 は各入力 S,S'を知ることはできない．

$$H(\alpha)=H\left(\alpha\middle|\alpha_j(j=0,1,\ldots,k-2)\right)$$
$$H(\delta)=H\left(\delta\middle|\delta_j(j=0,1,\ldots,k-2)\right)$$
$$H(\eta)=H\left(\eta\middle|\eta_j(j=0,1,\ldots,k-2)\right)$$
$$H(\beta)=H\left(\beta\middle|\beta_j(j=0,1,\ldots,k-2)\right)$$
$$H(S)=H\left(S\middle|\alpha S,\beta S',\gamma,\alpha\delta,\beta\eta,\alpha_j,\beta_j,\eta_j,\delta_j\ (j=0,\ldots,k-2)\right)$$
$$H(S')=H\left(S'\middle|\alpha S,\beta S',\gamma,\alpha\delta,\beta\eta,\alpha_j,\beta_j,\eta_j,\delta_j\ (j=0,\ldots,k-2)\right)$$

それに対して，$[1]_j$,$[1']_j(\overline{[\delta]}_j^{(q)},\overline{[\eta]}_j^{(q)})$が導入されていない場合，αδ から α が漏洩し，βη から β が漏洩するため，秘密情報が漏洩する．

以上より，$[1]_j$ と$[1']_j$の導入により，$\alpha S,\beta S'$をそのまま保存しても攻撃者 3 に対して安全性であることが言える．

4. 評価

以下に，2.2 に示す Shamir 法ベースの秘匿演算，4.2 に示す基本方式，5.2 に示す改良方式の比較を行う．ここでは，法 p で分散された分散値のサイズを P，法 q で分散された分散値のサイズを Q として，記憶量・計算量・通信量に関する性能評価を行い，表 1 に示す．記憶量は，分散情報の数で評価し，計算量については，一般に加減算は乗除算に比べ，十分計算量が小さいため，乗除算の回数で評価する．また，通信量については，ビット数で評価した．また，表 2 において，(n,k)=(2,2)，(5,4)における具体例を示す．ただし，表 2 における評価では，ともに e=10，p=11 とし，(n,k)=(2,2)のとき q=127，(5,4)のとき，q=161053 とした．

表 1 性能比較

		Shamir 法	従来方式	提案方式
記憶量		Q(k+1)n	Q(n-1+k)n	Q(k+1)n
計算量	分散	n(k+1)(k-1)+k/2	0	0
	復元	$(k+1)^2(k-1)$	0	0
	乗除算	k(2R+C+1)+R+n	k+n	k+n
	加減算	k(2R+C+3)	4k	6k
通信量	分散	Q(k+1)n	Q(n-1+k)n	Q(kn+1)
	復元	Q(k+1)k	Q(2k-1)k	$Q(k^2+1)$
	乗除算	kQ(1+2k+n)+Q	kQ(3k+n)+Q	kQ(2k+n)
	加減算	A	A	$A+Q(2kn+3n^2-3n+1)$

表 2 性能比較の例

(n,k)		(2,2)			(5,4)		
		Shamir 法	従来方式	提案方式	Shamir 法	従来方式	提案方式
記憶量		42	42	42	450	720	450
計算量	分散	7	0	0	77	0	0
	復元	9	0	0	75	0	0
	乗除算	23	4	4	244	4	9
	加減算	22	8	12	232	16	24
通信量	分散	42	42	35	450	720	378
	復元	42	42	35	360	504	306
	乗除算	105	119	84	1026	1242	936
	加減算	126	126	231	1116	1116	2934

ただし，秘匿加減算の通信量評価において，A=2kQ(k+1)+Q(kn+2)とおいており，Shamir 法においては秘密分散及び復元を行う度に乗算が発生するので，秘密分散に必要な計算量を C=(2k-3)n，復元に必要な計算量を R=(k+1)(k-1)とする．表中の網掛けは，その評価項目における最良の方式を示す．この結果より，従来方式は n=k=2 のとき最も効率的に秘匿四則演算を実現すると言えるが，それ以外では記憶量と通信量が大きく増加するため，記憶量・計算量・通信量において総合的に優れているのは提案方式といえる．

5. まとめ

本論文では記憶量・通信量の増加を抑え，計算量を大きく削減する秘匿計算法とその安全性を示した．

さらなる効率化が今後の課題である

参考文献

[1]鈴木良介: “ビッグデータビジネスの時代”，翔泳社, pp.14 (2011).

[2] 総務省: “平成 24 年版情報通信白書”, pp.153 (2012).

[3] 岡本栄司: “暗号理論入門[第 2 版]”，共立出版株式会社, pp. 91-95 (2006).

[4] A. Shamir: “How to Share a Secret”, Commun.ACM,vol.22, no.11, pp.612-613(1979).

[5] Jun Kurihara, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka: “A New (k, n)-Threshold Secret Sharing Scheme and Its Extension”, ISC 2008 conference, (2008).

[6] 高橋加寿子, 須賀祐治, 岩村惠市: “XOR を用いる秘密分散法の多値化とそれを用いた秘匿計算法”，第 65 回 CSEC 研究会(2014).

[7] Yuji Suga:”Consideration of the XOR-operation based Secure Multiparty Computation”, The Ninth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2015), 2015-07

[8] 神宮武志, 岩村惠市: “除算を含む四則演算に適用可能な秘密分散法を用いた秘匿計算手法の提案”，信学技報 115(122), pp.51-57(2015).

[9] Joel H. Ferziger, Milovan Perić: “コンピュータによる流体力学(小林敏雄,谷口伸行,坪倉誠訳)”，シュプリンガー・フェアラーク東京, pp.88 (2003).

[10] J. Kurihara, S. Kiyomoto, K. Fukushima, and T. Tanaka: “On a fast (k; n)-threshold secret sharing scheme”, IEICE Transactions on Fundamentals of Electronics, Communications and Computer sciences, vol. E91-A, No. 9, pp.2365-2378(2008).

[11] 青井健, 神宮武志, 岩村惠市: “n<2k-1 における秘匿計算の安全性検討及び非対称秘密分散との応用”，信学技報 116(130), pp.237-243(2016)

[12] 鴇田恭平，岩村惠市: “多値化法による秘匿四則演算及びその秘匿部分一致検索への応用”，信学技報 117(55), pp.107-114(2017)

付録

付録 A.1 符号反転の例

多値化法は XOR を単純に加算で置き換えているが，Gauss-Jordan の消去法を実行する際係数を除算する必要が発生する．それに対して，3.2(3)において適切な範囲で S_{j-i} を-S_{j-i} に置き換え，XOR を加算と減算に分けることで秘密情報の復元を除算のない単純な加減算だけで実行できる．

ただし，3.3(2)における $R_{(k,n)}$の成分に対応した部分を 1 にする際，-S_{j-i} とした部分は-1 とする必要がある．

上記変更によって，除算が不要になる証明は大変難しいが，シミュレーションによって確認することができる．以下に，符号反転させた場合とさせない場合の例を示す．

(例)n=5,k=4 の場合，i=1 かつ j=2,3 または i≥2 かつ j=1 の場合に S_{j-i} の符号を反転して，-S_{j-i} を代入して分散し，ユーザt_0,t_1,t_2,t_3から全ての部分秘密情報を求める．

・符号処理をしない場合

$$S_1=\frac{1}{5}\begin{pmatrix}-2W_{(0,0)}+W_{(0,1)}-W_{(0,2)}-3W_{(0,3)}-W_{(1,0)}-2W_{(1,1)}+2W_{(1,2)}\\+W_{(1,3)}+3W_{(2,0)}+W_{(2,1)}-W_{(2,2)}+2W_{(2,3)}\end{pmatrix}$$

$$S_2=\frac{1}{5}\begin{pmatrix}W_{(0,0)}+2W_{(0,1)}+3W_{(0,2)}-W_{(0,3)}-2W_{(1,0)}-4W_{(1,1)}-W_{(1,2)}\\+2W_{(1,3)}+W_{(2,0)}+2W_{(2,1)}-2W_{(2,2)}-W_{(2,3)}\end{pmatrix}$$

$$S_3=\frac{1}{5}\begin{pmatrix}-W_{(0,0)}+3W_{(0,1)}+2W_{(0,2)}+W_{(0,3)}-3W_{(1,0)}-6W_{(1,1)}-4W_{(1,2)}\\-2W_{(1,3)}+4W_{(2,0)}+3W_{(2,1)}+2W_{(2,2)}+W_{(2,3)}\end{pmatrix}$$

$$S_4=\frac{1}{5}\begin{pmatrix}-3W_{(0,0)}-W_{(0,1)}+W_{(0,2)}-2W_{(0,3)}+W_{(1,0)}-3W_{(1,1)}-2W_{(1,2)}\\-W_{(1,3)}+2W_{(2,0)}+4W_{(2,1)}+W_{(2,2)}+3W_{(2,3)}\end{pmatrix}$$

・符号処理をした場合

$$S_1=-3W_{(0,0)}-2W_{(0,1)}-W_{(0,2)}-2W_{(0,3)}+W_{(1,0)}-2W_{(1,2)}+W_{(2,0)}+3W_{(2,1)}+W_{(2,2)}+W_{(2,3)}+W_{(3,0)}-W_{(3,1)}+2W_{(3,2)}+W_{(3,3)}$$

$$S_2=W_{(0,0)}+W_{(0,1)}+W_{(0,2)}+W_{(0,3)}-W_{(1,0)}-W_{(1,1)}-W_{(2,1)}-W_{(2,2)}-W_{(2,3)}+W_{(3,1)}$$

$$S_3=2W_{(0,0)}+2W_{(0,1)}+W_{(0,2)}+2W_{(0,3)}-W_{(1,0)}-W_{(1,1)}+W_{(1,2)}-2W_{(2,1)}-W_{(2,2)}-W_{(2,3)}-W_{(3,0)}+W_{(3,1)}-W_{(3,2)}-W_{(3,3)}$$

$$S_4=W_{(0,0)}+2W_{(0,1)}+W_{(0,2)}+W_{(0,3)}-W_{(1,0)}-W_{(1,1)}+W_{(1,2)}+W_{(1,3)}-2W_{(2,1)}-2W_{(2,2)}-W_{(2,3)}+W_{(3,1)}-W_{(3,3)}$$

この結果から符号反転させた場合，部分分散情報の単純な加減算のみ（係数部分は，その回数加減算を繰り返す）で復元できることがわかる．ただし，上の例の条件で符号反転した場合，係数が整数となるためには，n≤5，または n=7 かつ k≤4，または n≥11 かつ k=2 である必要がある．それ以外の例についても，k,n を設定した際事前にシミュレーションによって確認し，適切に符号反転させる部分を選択すれば乗除算が不要な復号を実現できる．