Vol.2021-CSEC-94 No.15
Vol.2021-SPT-43 No.15
2021/7/19

n<2k-1 において malicious な攻撃者に対しても安全な秘密分散を用いた秘匿計算の高速化

工藤凌也 [1]　岩村惠市 [1]　稲村勝樹 [2]

概要：一般に，(k,n)閾値秘密分散法を用いた秘匿計算では，n<2k-1 において情報理論的安全性を持つ秘匿計算は不可能とされている．そのため，著者等のグループでは n<2k-1 において安全に秘匿計算が実行できる条件を探索するというアプローチをとり，malicious な攻撃者に対しても安全な秘匿計算法を提案した．ただし，この手法は全ての処理を秘匿計算として実行するため効率的でない．本論文では，その手法における演算処理を事前計算と秘匿計算に分け，演算のうち通信が必要な処理を事前計算に集中させることで従来法より高速な秘匿計算が実現できることを示す．

キーワード：秘密分散，秘匿計算，マルチパーティ計算，malicious adversary，$n < 2k - 1$

Fast Secret Computation using Secret Sharing for n<2k-1 secure against malicious adversaries

RYOYA KUDO[1]　KEIICHI IWAMURA[1]
MASAKI INAMURA[2]

Abstract: In general, the (k, n)-threshold secret sharing scheme cannot be used to perform an information-theoretically secure secret computation for $n < 2k - 1$. Therefore, our research team has been focusing on conditions under which the secret computation can be performed securely for $n < 2k - 1$, and proposed a secret computation method that is safe against malicious adversaries. However, this method is not efficient because it performs all the operations as secret computation. In this paper, we divide the arithmetic operations into precomputation and secret computation and show that we can achieve faster secret computation than the conventional method by concentrating the operations that require communication in precomputation.

Keywords: Secret Sharing Scheme, Secure Computation, MPC, malicious adversary, $n < 2k - 1$

1. はじめに

近年，IoT（Internet of Things）の発達により，様々な情報が収集され，クラウド上で動作するデータ分析技術を用いて統計情報に処理されるなどして利活用されている[1]．これらのデータには，何らかの個人情報が含まれている場合がある．そのようなデータを処理したり利活用したりする際は，個人を特定できないような形にするなど，取り扱いには十分注意しないと，プライバシーに影響を与える可能性がある．このように，個人のプライバシーを守りながらデータ分析を行う手法の一つとして，個人情報を秘匿したまま演算を行うことができる秘匿計算に関する研究が行われている．秘匿計算を実現する主な手法としては，準同型暗号を用いる方式[2][3]と，秘密分散を用いる方式がよく知られている．準同型暗号は暗号文のまま加算および乗算が可能であるが，計算量が多く，演算に時間がかかるという問題がある．そのため，著者らのグループでは，計算量が軽い秘密分散法を用いた秘匿計算法に関する研究を行っている．

秘密分散法は，1979 年に Shamir によって提案された(k, n)閾値秘密分散法（Shamir's Secret Sharing，以降 SSS）[4]が有名である．これは，1 つの秘密情報からn個のシェアを生成し，n台のサーバに分散しておき，そのシェアのうちk個以上を集めてくることで秘密情報を復元できるという手法である．このとき，k個未満のシェアからは一切秘密情報を復元することができない．一方で，SSS では秘密情報sおよびシェア$a_1, \ldots, a_{k-1}$から多項式$f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$を生成しているため，秘匿乗算を行うと多項式の最大次数に変化が起こるという問題がある．具体的には，1 回の乗算では閾値がkから$2k - 1$へ増加する（最大次数が$k - 1$から$2k - 2$に変化する）ため，$n < 2k - 1$において乗算結果が復元できないということである．これに対し，著者らのグループでは，秘密情報に乱数をかけることで秘匿し，スカラー量として乗算を行うというアプローチのもと，神宮らが TUS1 方式[5]を，Aminuddin らが TUS2 方式[6]を，鴇田らが TUS3 方式[7]を，岩村らが TUS4 方式[8]を，落合らが TUS5 方式[9]をそれぞれ提案している．以降，全体を総称して TUS 方式と呼び，個別の場合は TUS5

1 東京理科大学
Tokyo University of Science
2 広島市立大学
Hiroshima City University

Vol.2021-CSEC-94 No.15
Vol.2021-SPT-43 No.15
2021/7/19

方式のように番号まで表記する．これらの方式により，以下の 3 つの条件の下，次数変化のない秘匿乗算が実現されている．

(1) 乱数に 0 を含まない．
(2) 攻撃者が知らない乱数とそれを用いた乱数のシェア集合がある．
(3) 演算の連続において各サーバが扱うシェア集合内のシェアの位置は固定される．

上記の方式のうち，TUS5 方式は TUS 方式のアプローチを採用することで秘匿乗算においても次数変化が発生せず，$n < 2k-1$においても実行可能であり，計算結果を検証することで malicious な攻撃者に対しても安全性を持つ方式として提案された．また，Dishonest majority を想定し，秘密情報の機密性について情報理論的安全性を確保するなど，高い安全性を持つ．一方で，TUS5 方式は TUS3 方式に検証処理を追加したものとなっているが，TUS3 方式は処理量が多く，TUS5 方式においても性能の向上が課題であった．

そこで，本論文では TUS5 方式に対し，演算処理を事前計算と秘匿計算に分け，演算のうち通信が必要な処理を事前計算に集中させ，秘匿計算から通信を排除することで従来法より高速な秘匿計算が実現できることを示す．

本論文の構成は以下のようになっている．まず 2 章で関連研究について説明し，3 章では提案方式を，4 章ではその安全性を示す．また，5 章では性能について考察する．

2. 従来方式

2.1 (k,n)閾値秘密分散法

(k,n)閾値秘密分散法は，以下の 2 つの特徴を持つ秘密分散法である．

(1) k個未満のシェアからは，秘密情報に関する情報は一切得られない．
(2) 任意のk個以上のシェアからは，元の秘密情報を一意に復元できる．

代表的な(k,n)閾値秘密分散法として，SSS（多項式を用いる方式），栗原らの XOR を用いる方式[10][11]（以降 XOR 法），加法的秘密分散法[12][13][14][15]などがある．これらの方法は情報セキュリティの 3 要素の 1 つである機密性の確保を目的としている．また，n や k を柔軟に変更できれば可用性を確保できるほか，検証可能秘密分散（Verifiable Secret Sharing，以降 VSS）のように完全性を確保できる方式も存在する．これ以降，特に断らない限り SSS を用いるものとする．

2.2 malicious な攻撃者に対応可能な方式

malicious な攻撃者とは，サーバを乗っ取り，管理下においてプロトコルから逸脱させる攻撃者のことであり，データの改ざんなどが行われる可能性を考慮する必要がある．秘密分散時にシェア以外に検証用の値も配布することで，各参加者が受け取ったシェアや復元値を検証する方法が一般的であり，VSS と呼ばれる．代表的な方法として，Damgård らによる SPDZ 系列（SPDZ-1[12]，SPDZ-2[13]，MASCOT[14]），荒木らによる方式[15]，千田らによる方式[16] などがある．

SPDZ 系列は，$n = k$のときにのみ適用可能であり，攻撃者が過半数である Dishonest majority を想定している．なお，加法的秘密分散を用いており，Beaver の multiplication triple に対してシェアを生成し，検証には情報理論的 MAC やコミットメントスキームを用いている．

荒木らの方式は，$k = 2, n = 3$に限定し，加法的秘密分散をもとにした Replicated 秘密分散を用いており，ユーザがn個のシェアのうちある 1 つのシェア以外を保持するように秘密分散を行う方式である．SPDZ 系列と同様に Beaver の multiplication triple を用いているが，$k = 2, n = 3$という条件より正直者が過半数である Honest majority である．

千田らの方式は，Honest majority を想定しており，大きな有限体を仮定することで偶然に検証を通過する確率を限りなく小さくすることで安全性を実現している．なお，TUS5 方式にもこの方法が採用されている．

2.3 コミットメントスキーム

コミットメントスキーム[17]は malicious な攻撃者に対して安全な VSS としてよく用いられ，コミットメントCはメッセージmと結び付けられる（束縛性）が，メッセージは明らかにしない（秘匿性）という性質を持つ．なお，両者の性質を共に無条件（完全または統計的）に実現することは不可能であり，一方が無条件かつもう一方が計算量的である 2 種類のスキームに大別できる．一般的に，秘匿性を重視するシナリオが多いことから，無条件な秘匿性かつ計算量的な束縛性のタイプが良く用いられる．なお，TUS5 方式でこのタイプを用いていることから，提案方式でもこちらを用いることとする．

情報をコミットする committer と検証する verifier の 2 者間のプロトコルであり，[17]の表記を用いて以下のような手順からなる．

(1) コミットフェーズ
メッセージmが送られると，committer は$[C,(m,d)] = \mathrm{Commit}(m)$を計算し，$C$をコミットメントとして公開する．なお，コミットメントをオープンするための値dは公開することができる．
(2) オープンフェーズ
committer は verifier に(m,d)を公開することで，コミットメントCをオープンする．verifier は，メッセージがコミットメントと整合しているかを検証する．

2.4 TUS 方式

SSS では，1 回の乗算では閾値がkから$2k-1$へ増加するため，$n < 2k-1$において乗算結果が復元できないという問題があった．これに対し，神宮らは TUS1 方式として，一時的に片方の秘密情報を復元し，スカラー量と多項式を

Vol.2021-CSEC-94 No.15
Vol.2021-SPT-43 No.15
2021/7/19

掛けるようにすることで次数変化のない秘匿乗算を実現した[5]．しかし，この方式では，安全性の観点から加減算と乗算を組み合わせることができない．そこで，Aminuddin らは，TUS2 方式として，“1” のシェア集合を用いることで解決した[6]．この方式は，秘匿計算結果および乱数に 0 を含まないなどの 3 つの条件があり，情報理論的安全性を持つものの，計算量が多く効率的ではなかった．そのため，鴇田らは，TUS3 方式として，秘匿計算に XOR 法を用いることで，最速ではないものの効率的に秘匿計算を実現した[7].

この TUS3 方式をもとに，TUS4 方式[8]と TUS5 方式[9]が提案されている．TUS4 方式では，演算を事前処理と秘匿計算に分け，通信が必要な処理を事前処理に集中させることで秘匿計算から通信を削減して高速化を行った．この方式は，1 章で述べた TUS 方式の 3 つの条件のうち，応用に応じて(2)以外を解決している．TUS5 方式では，malicious な攻撃者を想定したほか，Dishonest majority を想定し，$n < 2k-1$においても実行可能で，秘密情報の機密性を情報理論的に確保している．また，TUS 方式に合わせて 2 通りでの秘匿計算による計算結果を検証し，コミットメントスキームによって秘密情報の正当性を検証したほか，全入力検証により検証処理を削減している．さらに，計算を実行する環境に応じてコストや安全性を変更できるという特長も持つ．

3. 提案方式

TUS5 方式に対し，TUS4 方式と同様に演算を事前処理と秘匿計算に分け，通信が必要な処理を事前処理に集中させることで，秘匿計算から通信を削減して高速化を実現する方法を示す．なお，3.1 節の分散処理および 3.2 節の復元処理は TUS5 方式と同様のため，概要のみを示す．

【表記】

S_i：全サーバのうち，i番目のサーバ．$(i = 0,1,\dots,n-1)$

S_j：演算に関与するサーバのうち，j番目のサーバ．$(i = 0,1,\dots,k-1)$

$[x]_i$：サーバS_iが保持する SSS によるxのシェア．

$[x]_i^{XOR}$：サーバS_iが保持する XOR 法によるxのシェア．

$\varepsilon_h, \varepsilon_{h,l}$：変換用乱数 $(h = 1,2,\dots,$必要個数$)$．対応するシェアが信頼できる第三者により事前にn台のサーバに秘密分散され，各サーバがローカルに保持する．

$\alpha_{y,j}, \beta_{y,j}, \gamma_{y,j}$：秘密情報$a, b, c$に対して，サーバ$S_j$が生成してローカルに保持する乱数．$(y = 0,\dots,5)$

$\delta_{z,j}$：計算結果dに対して，サーバS_jが生成してローカルに保持する乱数．$(z = 0,2,3,5)$

3.1 分散処理

【分散のための事前処理】

① サーバS_jは乱数 $A_{1,j}, A_{2,j}, \alpha_{0,j}, \dots, \alpha_{5,j}$ を生成し，$\alpha_{1,j}, \alpha_{4,j}$をコミットする．また，$A_{1,j}, A_{2,j}, \alpha_{0,j}, \dots, \alpha_{5,j}$を$n$台のサーバに XOR 法で分散する．

② サーバS_jは変換用乱数を復元し，以下を計算して，n台のサーバに送信する．

$$\alpha_{0,j}\alpha_{1,j},\ \alpha_{2,j}\alpha_{1,j},\ \alpha_{3,j}\alpha_{4,j},\ \alpha_{5,j}\alpha_{4,j},$$
$$\frac{\alpha_{2,j}}{\varepsilon_{1,j}},\ \frac{\alpha_{2,j}A_{1,j}}{\varepsilon_{2,j}},\ \frac{\alpha_{5,j}}{\varepsilon_{3,j}},\ \frac{\alpha_{5,j}A_{2,j}}{\varepsilon_{4,j}},\frac{1}{\alpha_{2,j}\varepsilon_{5,j}},\frac{A_{2,j}}{\varepsilon_{6,j}},\frac{1}{\alpha_{5,j}\varepsilon_{7,j}},\frac{A_{1,j}}{\varepsilon_{8,j}}$$

③ 全サーバは各k個の値の積を計算し，分母の変換用乱数を打ち消すように対応する変換用乱数のシェアを掛け，1 回の分散処理用に以下を保持する．

$$\alpha_0\alpha_1,\ \alpha_2\alpha_1,\ \alpha_3\alpha_4,\ \alpha_5\alpha_4, \left[A_{1,l}\right]_i^{XOR}, \left[A_{2,l}\right]_i^{XOR}$$

$$\left[\alpha_{0,l}\right]_i^{XOR}, \left[\alpha_{1,l}\right]_i^{XOR}, \left[\alpha_{2,l}\right]_i^{XOR}, \left[\alpha_{3,l}\right]_i^{XOR}, \left[\alpha_{4,l}\right]_i^{XOR}, \left[\alpha_{5,l}\right]_i^{XOR}$$

$$[\alpha_2]_i, [\alpha_2A_1]_i, [\alpha_5]_i, [\alpha_5A_2]_i, \left[\frac{1}{\alpha_2}\right]_i, [A_2]_i, \left[\frac{1}{\alpha_5}\right]_i, [A_1]_i$$

【分散処理】

① サーバS_jは$\left[A_{1,l}\right]_j^{XOR}, \left[A_{2,l}\right]_j^{XOR}$を入力者に送信する．

② 入力者は$A_{1,l}, A_{2,l}$を復元する．さらに，$a + A_1$, $a + A_2$を計算して，n 台のサーバに送信する．

③ サーバS_jは以下を計算し，n台のサーバへ送信する．

$$[\alpha_2(a+\alpha_1)]_j = (a + A_1) \times [\alpha_2]_j + \alpha_2\alpha_1 - [\alpha_2A_1]_j$$
$$[\alpha_5(a+\alpha_4)]_j = (a + A_2) \times [\alpha_5]_j + \alpha_5\alpha_4 - [\alpha_5A_2]_j$$

④ 全サーバは，受け取った各々k個のシェアより，$\alpha_2(a+\alpha_1), \alpha_5(a+\alpha_4)$ を復元し，以下を計算する．

$$[\alpha_1]_i = \alpha_2(a+\alpha_1) \times \left[\frac{1}{\alpha_2}\right]_i + [A_2]_i - (a + A_2)$$

$$[\alpha_4]_i = \alpha_5(a+\alpha_4) \times \left[\frac{1}{\alpha_5}\right]_i + [A_1]_i - (a + A_1)$$

⑤ 各サーバS_iは，秘密情報aに対するシェアとして，以下を保持する．

$$\alpha_0\alpha_1,\ \alpha_2(a+\alpha_1),\ \alpha_3\alpha_4,\ \alpha_5(a+\alpha_4),\ [\alpha_1]_i,\ [\alpha_4]_i$$
$$\left[\alpha_{0,l}\right]_i^{XOR}, \left[\alpha_{1,l}\right]_i^{XOR}, \left[\alpha_{2,l}\right]_i^{XOR}, \left[\alpha_{3,l}\right]_i^{XOR}, \left[\alpha_{4,l}\right]_i^{XOR}, \left[\alpha_{5,l}\right]_i^{XOR}$$

3.2 復元処理

① 復元に参加するk台のサーバS_jは，以下を復元者に送信する．

$$\alpha_2(a+\alpha_1), \left[\alpha_{2,0}\right]_j^{XOR}, \dots, \left[\alpha_{2,k-1}\right]_j^{XOR}, [\alpha_1]_j,$$

$$\alpha_5(a+\alpha_4), \left[\alpha_{5,0}\right]_j^{XOR}, \dots, \left[\alpha_{5,k-1}\right]_j^{XOR}, [\alpha_4]_j$$

② 復元者はシェアから α_1, α_4を復元し，$\alpha_{1,j}, \alpha_{4,j}$に対するコミットメントをオープンして，それぞれ$k$個掛け合わせて$\alpha_1, \alpha_4$を計算し，これら 2 種類の手順で得られる$\alpha_1, \alpha_4$が一致するか検証する．

③ 復元者はシェアから $\alpha_{2,j}, \alpha_{5,j}$を復元し，以下を計算する．そして，ここで計算した値と，手順②で得たα_1, α_4から計算される$(\alpha_1 - \alpha_4)$を比較して一致するか

Vol.2021-CSEC-94 No.15
Vol.2021-SPT-43 No.15
2021/7/19

検証する．

$$\frac{\alpha_2(a+\alpha_1)}{\prod_{j=0}^{k-1}\alpha_{2,j}}-\frac{\alpha_5(a+\alpha_4)}{\prod_{j=0}^{k-1}\alpha_{5,j}}$$

④ 全ての検証が問題なければ，復元者は以下の計算で復元結果を得る．

$$a=\frac{\alpha_2(a+\alpha_1)}{\prod_{j=0}^{k-1}\alpha_{2,j}}-\alpha_1=\frac{\alpha_5(a+\alpha_4)}{\prod_{j=0}^{k-1}\alpha_{5,j}}-\alpha_4$$

3.3 秘匿積和演算

本節では，3 人の入力者がそれぞれ 1 つずつ保持する秘密情報a,b,cの合計 3 入力から 1 出力$d=ab+c$を求める秘匿積和演算について述べる．なお，同様の処理で和$a+b$や積$a\times b$が求められ，繰り返し行うことで拡張積和演算$\sum_{i=1}^{l}(a_{1,i}a_{2,i}\cdots a_{m_i,i})$も計算可能である．しかし，演算を繰り返す場合は，復元結果が 0 であると，途中結果が 0 であることが漏洩してしまう．そこで，本節では 0 が復元されない場合の方法を示す．0 が復元される場合は，3.4 節で述べる補正処理を追加で行う必要がある．ここで，3 入力a,b,cに対するサーバS_iが持つシェアを，TUS5 方式同様以下のように定義する．

$$\alpha_0\alpha_1,\alpha_2(a+\alpha_1),\alpha_3\alpha_4,\alpha_5(a+\alpha_4),$$
$$[\alpha_{0,l}]_i^{XOR},[\alpha_{2,l}]_i^{XOR},[\alpha_{3,l}]_i^{XOR},[\alpha_{5,l}]_i^{XOR}$$
$$\beta_0\beta_1,\beta_2(b+\beta_1),\beta_3\beta_4,\beta_5(b+\beta_4),$$
$$[\beta_{0,l}]_i^{XOR},[\beta_{2,l}]_i^{XOR},[\beta_{3,l}]_i^{XOR},[\beta_{5,l}]_i^{XOR}$$
$$\gamma_0\gamma_1,\gamma_2(c+\gamma_1),\gamma_3\gamma_4,\gamma_5(c+\gamma_4),$$
$$[\gamma_{0,l}]_i^{XOR},[\gamma_{2,l}]_i^{XOR},[\gamma_{3,l}]_i^{XOR},[\gamma_{5,l}]_i^{XOR}$$

また，出力dに対するサーバS_iが保持するシェアを，TUS5 方式同様以下のように定義する．ただし，$d_x=\gamma_x-\alpha_x\beta_x$，$x=1,4$である．

$$\delta_0d_1,\delta_2(d+d_1),\delta_3d_4,\delta_5(d+d_4),$$
$$[\delta_{0,l}]_i^{XOR},[\delta_{2,l}]_i^{XOR},[\delta_{3,l}]_i^{XOR},[\delta_{5,l}]_i^{XOR}$$

なお，事前計算を行うタイミングによっては，乱数に関するシェアのみで秘密情報に関するシェアは保持していない可能性もあるが，事前計算では使用しないため問題はない．

【事前計算】

① 演算に参加するk台のサーバS_jは，$\varepsilon_{1,j},\varepsilon_{2,j},\dots,\varepsilon_{12,j}$に対するシェアを収集し，復元する．

② サーバS_jは乱数$\delta_{0,j},\delta_{2,j},\delta_{3,j},\delta_{5,j}$を生成して，$n$台のサーバに XOR 法で分散する．

③ サーバS_jは以下を計算し，n台のサーバに送信する．

$$\frac{\delta_{0,j}}{\gamma_{0,j}\varepsilon_{1,j}},\frac{\delta_{0,j}}{\alpha_{0,j}\beta_{0,j}\varepsilon_{2,j}},\frac{\delta_{2,j}}{\alpha_{2,j}\beta_{2,j}\varepsilon_{3,j}},\frac{\delta_{2,j}}{\alpha_{2,j}\beta_{0,j}\varepsilon_{4,j}},\frac{\delta_{2,j}}{\alpha_{0,j}\beta_{2,j}\varepsilon_{5,j}},\frac{\delta_{2,j}}{\gamma_{2,j}\varepsilon_{6,j}}$$
$$\frac{\delta_{3,j}}{\gamma_{3,j}\varepsilon_{7,j}},\frac{\delta_{3,j}}{\alpha_{3,j}\beta_{3,j}\varepsilon_{8,j}},\frac{\delta_{5,j}}{\alpha_{5,j}\beta_{5,j}\varepsilon_{9,j}},\frac{\delta_{5,j}}{\alpha_{5,j}\beta_{3,j}\varepsilon_{10,j}},\frac{\delta_{5,j}}{\alpha_{3,j}\beta_{5,j}\varepsilon_{11,j}},\frac{\delta_{5,j}}{\gamma_{5,j}\varepsilon_{12,j}}$$

④ 全サーバは，手順③で受け取った各k個の値の積を計算し，以下を得る．

$$\frac{\delta_0}{\gamma_0\varepsilon_1},\frac{\delta_0}{\alpha_0\beta_0\varepsilon_2},\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3},\frac{\delta_2}{\alpha_2\beta_0\varepsilon_4},\frac{\delta_2}{\alpha_0\beta_2\varepsilon_5},\frac{\delta_2}{\gamma_2\varepsilon_6}$$
$$\frac{\delta_3}{\gamma_3\varepsilon_7},\frac{\delta_3}{\alpha_3\beta_3\varepsilon_8},\frac{\delta_5}{\alpha_5\beta_5\varepsilon_9},\frac{\delta_5}{\alpha_5\beta_3\varepsilon_{10}},\frac{\delta_5}{\alpha_3\beta_5\varepsilon_{11}},\frac{\delta_5}{\gamma_5\varepsilon_{12}}$$

⑤ サーバS_jは以下を計算して保持する．

$$\left[\frac{\delta_0}{\gamma_0}\right]_j=\frac{\delta_0}{\gamma_0\varepsilon_1}\times[\varepsilon_1]_j,\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_j=\frac{\delta_0}{\alpha_0\beta_0\varepsilon_2}\times[\varepsilon_2]_j,$$
$$\left[\frac{\delta_2}{\alpha_2\beta_2}\right]_j=\frac{\delta_2}{\alpha_2\beta_2\varepsilon_3}\times[\varepsilon_3]_j,\left[\frac{\delta_2}{\alpha_2\beta_0}\right]_j=\frac{\delta_2}{\alpha_2\beta_0\varepsilon_4}\times[\varepsilon_4]_j,$$
$$\left[\frac{\delta_2}{\alpha_0\beta_2}\right]_j=\frac{\delta_2}{\alpha_0\beta_2\varepsilon_5}\times[\varepsilon_5]_j,\left[\frac{\delta_2}{\gamma_2}\right]_j=\frac{\delta_2}{\gamma_2\varepsilon_6}\times[\varepsilon_6]_j,$$
$$\left[\frac{\delta_3}{\gamma_3}\right]_j=\frac{\delta_3}{\gamma_3\varepsilon_7}\times[\varepsilon_7]_j,\left[\frac{\delta_3}{\alpha_3\beta_3}\right]_j=\frac{\delta_3}{\alpha_3\beta_3\varepsilon_8}\times[\varepsilon_8]_j,$$
$$\left[\frac{\delta_5}{\alpha_5\beta_5}\right]_j=\frac{\delta_5}{\alpha_5\beta_5\varepsilon_9}\times[\varepsilon_9]_j,\left[\frac{\delta_5}{\alpha_5\beta_3}\right]_j=\frac{\delta_5}{\alpha_5\beta_3\varepsilon_{10}}\times[\varepsilon_{10}]_j,$$
$$\left[\frac{\delta_5}{\alpha_3\beta_5}\right]_j=\frac{\delta_5}{\alpha_3\beta_5\varepsilon_{11}}\times[\varepsilon_{11}]_j,\left[\frac{\delta_5}{\gamma_5}\right]_j=\frac{\delta_5}{\gamma_5\varepsilon_{12}}\times[\varepsilon_{12}]_j$$

⑥ サーバS_jは以下を計算する．

$$[\delta_0d_1]_j=[\delta_0(\gamma_1-\alpha_1\beta_1)]_j$$
$$=\gamma_0\gamma_1\times\left[\frac{\delta_0}{\gamma_0}\right]_j-\alpha_0\alpha_1\times\beta_0\beta_1\times\left[\frac{\delta_0}{\alpha_0\beta_0}\right]_j$$
$$[\delta_3d_4]_j=[\delta_3(\gamma_4-\alpha_4\beta_4)]_j$$
$$=\gamma_3\gamma_4\times\left[\frac{\delta_3}{\gamma_3}\right]_j-\alpha_3\alpha_4\times\beta_3\beta_4\times\left[\frac{\delta_3}{\alpha_3\beta_3}\right]_j$$

【秘匿計算】

① サーバS_jは以下を計算する．

$$[\delta_2(d+d_1)]_j=[\delta_2\{(ab+c)+(\gamma_1-\alpha_1\beta_1)\}]_j$$
$$=\alpha_2(a+\alpha_1)\times\beta_2(b+\beta_1)\times\left[\frac{\delta_2}{\alpha_2\beta_2}\right]_j$$
$$-\alpha_2(a+\alpha_1)\times\beta_0\beta_1\times\left[\frac{\delta_2}{\alpha_2\beta_0}\right]_j$$
$$-\alpha_0\alpha_1\times\beta_2(b+\beta_1)\times\left[\frac{\delta_2}{\alpha_0\beta_2}\right]_j$$
$$+\gamma_2(c+\gamma_1)\times\left[\frac{\delta_2}{\gamma_2}\right]_j$$
$$[\delta_5(d+d_4)]_j=[\delta_5\{(ab+c)+(\gamma_4-\alpha_4\beta_4)\}]_j$$
$$=\alpha_5(a+\alpha_4)\times\beta_5(b+\beta_4)\times\left[\frac{\delta_5}{\alpha_5\beta_5}\right]_j$$
$$-\alpha_5(a+\alpha_4)\times\beta_3\beta_4\times\left[\frac{\delta_5}{\alpha_5\beta_3}\right]_j$$
$$-\alpha_3\alpha_4\times\beta_5(b+\beta_4)\times\left[\frac{\delta_5}{\alpha_3\beta_5}\right]_j$$
$$+\gamma_5(c+\gamma_4)\times\left[\frac{\delta_5}{\gamma_5}\right]_j$$

② 全サーバは，サーバS_jから$[\delta_0d_1]_j,[\delta_3d_4]_j$を集めてきて，$\delta_0d_1,\delta_3d_4$を復元する．

③ 全サーバは，サーバS_jから$[\delta_2(d+d_1)]_j,[\delta_5(d+d_4)]_j$

Vol.2021-CSEC-94 No.15
Vol.2021-SPT-43 No.15
2021/7/19

を集めてきて，$\delta_2(d+d_1), \delta_5(d+d_4)$を復元する．

④ 各サーバS_iは，計算結果$d=ab+c$に対するシェアとして，以下を保持する．

$$\delta_0 d_1, \delta_2(d+d_1), \delta_3 d_4, \delta_5(d+d_4),$$
$$[\delta_{0,l}]_i^{XOR}, [\delta_{2,l}]_i^{XOR}, [\delta_{3,l}]_i^{XOR}, [\delta_{5,l}]_i^{XOR}$$

3.4 補正処理

演算中に 0 が復元された場合，TUS5 方式同様，以下の対策を講じる必要がある．

(1) 0 が復元された場合, その情報を全サーバが保持する．

(2) シェアの計算前に, 演算に参加するサーバはその時点で漏洩している情報を参照する. 次に計算しようとしているシェアの復元結果が 0 となる場合に, 個々の乱数や秘密情報や計算結果が漏洩するならば, 新たな乱数w_1を用いて，$d_1' = d_1 + w_1$とする．

ここで，(2)の処理を行うかどうかを判断するタイミングは，3.3 節【秘匿計算】の手順②および【秘匿計算】の手順③である．以下に(2)の処理を行う場合の処理を示す．なお，3.3 節との変更部分のみを示す．

【3.3 節【秘匿計算】の手順②で$[\delta_0 d_1]_i$に対して補正処理が実行される場合】

各サーバは，3.3 節【事前計算】手順⑥までの処理が済んでいるとする．なお，以下の処理のうち，手順①～④，⑦の分岐後～⑨，および⑤⑥⑩のうち以下の計算は，3.1 節【分散のための事前処理】または 3.3 節【事前計算】の類似の処理と同時に実行しておくとここでの通信が削減され，より高速化できる．

$$[\delta_0 w_1]_j = \frac{\delta_0 w_1}{\varepsilon_{13}} \times [\varepsilon_{13}]_j, [\delta_2 w_1]_j = \frac{\delta_2 w_1}{\varepsilon_{14}} \times [\varepsilon_{14}]_j$$

① サーバS_jは$\varepsilon_{13,j}$に対するシェアを収集し，復元する．

② サーバS_jは乱数$w_{1,j}$を生成し，n台のサーバへ XOR 法で分散する．また，$w_{1,j}$をコミットする．

③ サーバS_jは $\delta_{0,j} w_{1,j}/\varepsilon_{13,j}$ を計算し，n台のサーバに送信する．

④ 全サーバは，手順③で受け取った各k個の値の積を計算し， $\delta_0 w_1/\varepsilon_{13}$ を得る．

⑤ サーバS_jは以下を計算し，$[\delta_0 d_1']_j$をn台のサーバに送信する．

$$[\delta_0 d_1']_j = [\delta_0(w_1 + \gamma_1 - \alpha_1\beta_1)]_j$$
$$= \frac{\delta_0 w_1}{\varepsilon_{13}} \times [\varepsilon_{13}]_i + \gamma_0\gamma_1 \times \left[\frac{\delta_0}{\gamma_0}\right]_j$$
$$- \alpha_0\alpha_1 \times \beta_0\beta_1 \times \left[\frac{\delta_0}{\alpha_0\beta_0}\right]_j$$

⑥ 全サーバは, $\delta_0 d_1'$を復元する. 復元結果が 0 の場合は，以下を計算してn台のサーバに送信し，全サーバは，$\delta_0 d_1 \neq 0$を復元して保持する． ここでの$\delta_0 w_1/\varepsilon_{13} \times [\varepsilon_{13}]_i$は，手順⑤と同じで良い．

$$[\delta_0 d_1]_i = \delta_0 d_1' - \frac{\delta_0 w_1}{\varepsilon_{13}} \times [\varepsilon_{13}]_i$$

⑦ $\delta_0 d_1 \neq 0$ を保持した場合は 3.3 節と同じであるため，3.3 節【秘匿計算】の手順①，手順③，手順④を行う．$\delta_0 d_1' \neq 0$を保持した場合，サーバS_jは$\varepsilon_{14,j}$に対するシェアを収集し，復元する．

⑧ サーバS_jは $\delta_{2,j} w_{1,j}/\varepsilon_{14,j}$ を計算し，n台のサーバに送信する．

⑨ 全サーバは，手順⑧で受け取った各k個の値の積を計算し， $\delta_2 w_1/\varepsilon_{14}$ を得る．

⑩ サーバS_jは以下を計算し，$[\delta_2(d+d_1')]_j$をn台のサーバに送信する．

$$[\delta_2(d+d_1')]_j = [\delta_2\{(ab+c) + (w_1 + \gamma_1 - \alpha_1\beta_1)\}]_j$$
$$= \frac{\delta_2 w_1}{\varepsilon_{14}} \times [\varepsilon_{14}]_j$$
$$+ \alpha_2(a+\alpha_1) \times \beta_2(b+\beta_1) \times \left[\frac{\delta_2}{\alpha_2\beta_2}\right]_j$$
$$- \alpha_2(a+\alpha_1) \times \beta_0\beta_1 \times \left[\frac{\delta_2}{\alpha_2\beta_0}\right]_j$$
$$- \alpha_0\alpha_1 \times \beta_2(b+\beta_1) \times \left[\frac{\delta_2}{\alpha_0\beta_2}\right]_j$$
$$+ \gamma_2(c+\gamma_1) \times \left[\frac{\delta_2}{\gamma_2}\right]_j$$

⑪ 全サーバは，$\delta_2(d+d_1')$を復元し，保持する．

⑫ サーバS_iは，以下を保持する．（3.3 節との変更分及び追加分のみ示す.）

$$\delta_0 d_1', \delta_2(d+d_1'), [w_{1,l}]_i^{XOR}$$

【3.3 節【秘匿計算】の手順③で補正処理が実行される場合】

補正処理の手順①から再び実行する．なお，$\delta_0 d_1 \neq 0$を保持しても無意味なため，$\delta_0 d_1' \neq 0$となるまで乱数w_1を変えて計算を繰り返す．

4. 安全性

安全性要件および攻撃者設定を TUS5 方式と同様，以下のように設定する．

【安全性要件】

情報セキュリティの 3 要素を，それぞれ以下のように設定する．

(1) 機密性 : honest な入力者が入力した秘密情報は攻撃者に漏洩しない．

(2) 完全性：偶然すべての攻撃が成功する確率を除き，honest な復元者は，入力者が実際に分散した値もしくはそれらから計算できる値を復元処理により得る．

(3) 可用性：$n \geq k$において利用可能である．

【攻撃者設定】

攻撃者の要件を以下のように設定する．

(1) 変換用乱数組$[\varepsilon_h]_i, [\varepsilon_{h,l}]_i^{XOR}$により定義される$\varepsilon_h$を知らず，不正に知ろうとする．

(2) t入力 1 出力の演算に対し，攻撃者との結託は最大$t-1$人である．
(3) 最大で$k-1$台のサーバを corrupt でき，サーバが持つ情報を知る．
(4) 攻撃者は, corrupt したサーバをプロトコルから逸脱させることができる．
(5) 機密性については計算能力が無限な攻撃者を，完全性については計算能力が有限な攻撃者を仮定する．

4.1 機密性

攻撃者は，corrupt する$k-1$台のサーバ$S_0,\dots,S_{k-2}$がローカルに保持する値や，演算中に全サーバへ送信される値を組み合わせて，honest なサーバS_{k-1}がローカルに持つ情報や honest な入力者の秘密情報を知ろうとする．ここで，定理 1 より, 機密性の証明においては基本的に$n=k$のプロトコルについて議論を行う．

定理 1: 機密性において, $n=k$のプロトコルと$n>k$のプロトコルの安全性に関して, $n>k$の場合のプロトコルで採用する XOR 法が情報理論的安全性を有するなら，両者の安全性は同等である．

証明：TUS5 方式[9]と同じため省略する．[証明終]

次に，各サーバが保持する乱数の積を n 台のサーバに送信する処理や，受け取った各k個の値の積を計算する処理について，定理 2 が成り立つ．

定理 2：演算中に全サーバが受け取る 2 つ以上の乱数の積および商で構成された値を組み合わせても，個々の乱数を得ることはできない．さらに，これらのような値を構成する乱数の一部は入力者や復元者が攻撃者と結託した場合に漏洩するが，変換用乱数を打ち消せず残りの乱数は漏洩しない．また，入力者や復元者との結託によって漏洩する乱数は，直接的な機密性への影響はない．

証明：TUS5 方式[9]と同じため省略する．[証明終]

4.1.1 秘匿積和演算に対する機密性

攻撃者は，最大 2 人の入力者と結託し，残りの入力者の秘密情報を知ろうとする．aの入力者が結託したとすると，攻撃者は$a, A_1, A_2, A_{1,k-1}, A_{2,k-1}$を知る．これに関連して$(a+A_1),(a+A_2),\alpha_2(a+\alpha_1),\alpha_5(a+\alpha_4)$を知るが, 個別の乱数は$A_1, A_2$以外 1 つも漏洩しない．なお，ここまでの機密性に関する議論は TUS5 方式と同じである．

【【事前計算】に対する機密性】

定理 2 より手順①から手順④で honest なサーバのみがローカルに保持するいかなる乱数も漏洩しないことが言える．ここで生成される新たな乱数$\delta_0,\delta_2,\delta_3,\delta_5$についても同様である．また，手順⑤はローカルな演算であり，ここで生成されるシェアは【攻撃者設定】(1)に示した変換用乱数組を用いて生成されているため，各サーバが持つシェアは安全に生成できる．手順⑥はローカルな演算であり，乱数は漏洩しない．

【【秘匿計算】に対する機密性】

手順①はローカルな計算であり，乱数は漏洩しない．また，手順②および手順③について考えると，ここまでの処理において，定理 2 により honest なサーバのみがローカルに保持する乱数は漏洩していない．ゆえに，復元される値から$\delta_0,\delta_2,\delta_3,\delta_5$を取り除くことはできず，本処理では入力者の秘密情報は攻撃者に漏洩しない．これは，2 人の入力者が攻撃者と結託した場合でも同様に言える．

4.1.2 補正処理に対する機密性

通信削減のための処理では，対応する処理での証明がここでも成り立ち，個々の乱数は一切漏洩しない．この処理を事前に行わない場合でも，TUS5 方式[9]と同様の議論により，個々の乱数および新たにコミットした乱数w_1が攻撃者に漏洩しないため，有益な情報は得られない．

以上より，提案方式における機密性が言える．

4.2 完全性

定理 3 より, 完全性の証明においても$n=k$のプロトコルについて議論を行う．

定理 3: 完全性において, $n=k$のプロトコルと$n>k$のプロトコルの安全性に関して, $n>k$の場合のプロトコルで採用する XOR 法が情報理論的安全性を有するなら，両者の安全性は同等である．

証明：TUS5 方式[9]と同じため省略する．[証明終]

また，秘密情報の復元に関して，定理 4 および定理 5 が成り立つ．

定理 4: 秘密情報の復元に関して，$[\textit{乱数}]\times([\textit{秘密情報}]+[\textit{コミットされた乱数}])$という値を構成する$[\textit{秘密情報}]$と$[\textit{コミットされた乱数}]$に発生する差分が等しくなければ, 偶然の場合を除いてコミットメントスキームによる検証は通過できない．

定理 5：秘密情報aの復元に関して，復元者が得るα_2,α_5への改ざん攻撃は，$\alpha_2(a+\alpha_1),\alpha_5(a+\alpha_4)$のような $[\textit{乱数}]\times([\textit{秘密情報}]+[\textit{コミットされた乱数}])$という値に関して，$[\textit{秘密情報}]$と$[\textit{コミットされた乱数}]$に同じ差分を発生させるのみであるため，定理 4 と併せてこれは無意味な攻撃である．

証明: いずれも TUS5 方式[9]と同じため省略する. [証明終]

4.2.1 秘匿計算の結果の復元に関する完全性

提案方式において，秘匿計算結果は，定理 4 や定理 5 と同型の$[\textit{乱数}]\times([\textit{計算結果}]+[\textit{コミットされた乱数}])$という形で表現される．また，$[\textit{計算結果}]$に$a$が含まれれば，$[\textit{コミットされた乱数}]$には$\alpha_1,\alpha_4$の項が含まれるというように, $[\textit{計算結果}]$と$[\textit{コミットされた乱数}]$には対応した項が含まれる．よって，秘匿計算結果の復元に関して，$[\textit{計算結果}]$と$[\textit{コミットされた乱数}]$を独立に議論し, それぞれに発生する差分が等しくなることを言うことで，完全性を証明する．なお，乱数の対応する 2 通りの値は独立であるため，片方についてのみ着目して議論する．

定理 4 より，$[\textit{計算結果}]$の秘密情報単体の項について，

Vol.2021-CSEC-94 No.15
Vol.2021-SPT-43 No.15
2021/7/19

コミットメントスキームによる検証を全て通過した場合，[*秘密情報*]と[*コミットされた乱数*]に発生する差分は等しいと言えることから，完全性が言える．本節のここまでの議論は TUS5 方式[9]と同じである．

[*計算結果*]に含まれる秘密情報同士の積の項について議論する．これらの項は秘匿乗算の実行によって発生する．計算結果のある項がabであったとすると，3.3 節【秘匿計算】手順①より，積に関する部分は次のように書ける．

$$[\delta_2\{(ab)-(\alpha_1\beta_1)\}]_j$$
$$=\alpha_2(a+\alpha_1)\times\beta_2(b+\beta_1)\times\left[\frac{\delta_2}{\alpha_2\beta_2}\right]_j$$
$$-\alpha_2(a+\alpha_1)\times\beta_0\beta_1\times\left[\frac{\delta_2}{\alpha_2\beta_0}\right]_j$$
$$-\alpha_0\alpha_1\times\beta_2(b+\beta_1)\times\left[\frac{\delta_2}{\alpha_0\beta_2}\right]_j$$

ここで，TUS5 方式と同様に攻撃者が各項の乱数比$\left(\frac{\delta_2}{\alpha_2\beta_2}\right)$などを調整した場合，このシェアから復元される値は以下となる．

$$\delta_2\{(ab)-(\alpha_1\beta_1)\}'$$
$$=r_1\delta_2(a+\alpha_1)(b+\beta_1)$$
$$-r_2\delta_2(a+\alpha_1)\beta_1-r_3\delta_2\alpha_1(b+\beta_1)$$
$$=r_1\delta_2(ab+a\beta_1+b\alpha_1+\alpha_1\beta_1)$$
$$-r_2\delta_2(a\beta_1+\alpha_1\beta_1)$$
$$-r_3\delta_2(b\alpha_1+\alpha_1\beta_1)$$
$$=\delta_2\{r_1ab+(r_1-r_2)a\beta_1$$
$$+(r_1-r_3)b\alpha_1+(r_1-r_2-r_3)\alpha_1\beta_1\}$$

つまり，提案方式と TUS5 方式とでは乱数に違いがあるものの，乱数比を調整した結果であると考えると，TUS5 方式と同様の議論が成り立つと考えられる．ゆえに，対応する[*秘密情報*]と[*コミットされた乱数*]に発生する差分は$r_1-r_2-r_3=r_1$となることから，対応する[*秘密情報*]と[*コミットされた乱数*]に発生する差分は等しいと言え，完全性が成り立つ．なお，ここまで定理 4 を基に議論してきたが，定理 5 も同様に成り立つことが言える．

また，3.4 節の補正処理についても，TUS5 方式と同様の議論が成り立つ．

以上より，秘匿計算の結果の復元において，[*計算結果*]と[*コミットされた乱数*]を秘密情報に関して分解したとき，対応する項同士に発生する差分は等しくなることから，偶然の場合を除いて不正は必ず検出される．なお，処理中の各検証が偶然通過する確率は$1/p$であり，十分大きな素数pを法とした大きな有限体を仮定している提案方式では，$1/p$ は十分小さいと言える．ゆえに，提案方式における完全性が言える．

5. 性能に関する考察

本章で用いる表記を以下のように定義する．

【表記】

I：入力（秘密情報）の数

C'：コミットメントに関する通信量

D：秘密情報のデータサイズ

本章では，3.3 節【秘匿計算】の処理を中心に議論を行う．以下，特に断りなく手順を記述した場合，3.3 節【秘匿計算】における手順を指す．

演算を繰り返さない場合，手順②以降は復元処理時にまとめて行うことが可能である．そのため，秘匿計算時には手順①のみを実行すれば良く，これはローカルな演算であるため通信を 0 にすることが可能である．ここで，表 1 に提案方式と TUS5 方式，SPDZ-2（malicious な攻撃者に対応可能な方式の中で条件が TUS5 方式に最も近い）の比較を示す．なお，【事前計算】における処理は，任意のタイミングで行うことができるため，【秘匿計算】における処理には直接影響しない．ゆえに，他方式との比較は行わないが，参考として示す．

表 1 より，提案方式と TUS5 方式を比較すると，秘匿計算では通信を一切行わないため，通信量とラウンド数が削減できていることが分かる．また，SPDZ-2 と比較すると，秘匿計算では提案方式が優位であることが分かる．また，分散と復元では提案方式の通信量が多くなっているものの，ラウンド数は提案方式の方が少なくなっていることも分かる．

演算を繰り返す場合は，繰り返す演算が決まっており，乱数の組み合わせが定まっているならば，あらかじめ【事前計算】でその分の乱数も計算しておくことで通信回数を削減できる．また，演算結果が 0 でなければ，手順②および手順③をまとめて行うことができ，通信はこの復元に関するもののみで済む．演算結果が 0 のときは，補正処理が必要となるため復元の回数が増えるが，数回程度であると考えられる．

6. まとめ

TUS5 方式に対し，通信が必要な演算を事前計算で行うことで，高速化を図った．その結果，演算を繰り返す必要がない場合は秘匿計算時の通信を完全に排除することができた．なお，安全性についても検討を行い，TUS5 方式と同等の安全性が言えることが分かった．

今後は，処理全体を実装することでより具体的に実行速度の比較を行う．

表 1　他方式との比較

Table 1　Comparison to other methods

		提案方式	TUS5 方式	SPDZ-2
振舞い		malicious	malicious	malicious
機密性		Information-theoretic	Information-theoretic	Computational
完全性		Computational	Computational	Computational
可用性		$n \geq k$	$n \geq k$	$n = k$
【秘匿計算】における処理				
通信量	分散	$2\{k^2 + (k+1)n\}D$	$2\{k^2 + (k+1)n\}D$	$2nD$
	復元	$2\{(k^2+1) + kI\}D + 2kIC'$	$2\{(k^2+1) + kI\}D + 2kIC'$	$8C'$
	和	0	$(8k^2 + 16kn)D$	0
	積	0	$(8k^2 + 16kn)D$	$4nD$
	積和	0	$(12k^2 + 20kn)D$	$4nD$
ラウンド数	分散	3	3	1
	復元	1	1	8
	和	0	4	0
	積	0	4	1
	積和	0	4	1
【事前計算】における処理				
通信量	和	$(8k^2 + 12kn)D$	0	0
	積	$(8k^2 + 12kn)D$	0	0
	積和	$(12k^2 + 16kn)D$	0	0
ラウンド数	和	3	0	0
	積	3	0	0
	積和	3	0	0

参考文献

[1] 鈴木一哉，森本昌治，岩井孝法：IoT 技術の最新動向，電子情報通信学会通信ソサイエティマガジン，Vol.12，No.1，pp. 12-20 (2018)

[2] 林卓也：準同型暗号を用いた秘密計算とその応用，システム制御情報学会誌システム/制御/情報，Vol.63，No.2，pp.64-70 (2019)

[3] Brakerski, Z., Gentry, C. and Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption without Bootstrapping, ACM TOCT, O'Donnell, R. (eds), Vol.6, No.3, pp.309-325, ACM (2009).

[4] Shamir, A.: How to Share a Secret, Comm. ACM, Vol.22, No.11, pp.612-613 (1979).

[5] 神宮武志，青井健，ムハンマド　カマル　アフマド　アクマル　アミヌディン，岩村惠市：秘密分散法を用いた次数変化のない秘匿計算手法，情報処理学会論文誌，Vol.59, No.3, pp.1038-1049 (2018).

[6] ムハンマド　カマル　アフマド　アクマル　アミヌディン，岩村惠市：秘密分散を用いた四則演算の組み合わせに対して安全な次数変化のない秘匿計算，情報処理学会論文誌，Vol.59, No.9, pp.1581-1595 (2018).

[7] 鴇田恭平，岩村惠市：高速かつ n<2k-1 において秘密情報に 0 を含んでも実行可能な秘密分散による秘匿計算，電気学会論文誌 C，Vol.138, No.12, pp.1634-1645 (2018).

[8] Iwamura, K. and Mohd Kamal, A.A.A.: Secure computation by secret sharing using input encrypted with random number, SECRYPT2021, (to appear).

[9] 落合将吾，岩村惠市：n<2k-1 において malicious な攻撃者に対しても安全な秘密分散を用いた秘匿計算とその拡張，情報処理学会論文誌（採録予定）

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

[11] Kurihara, J., Kiyomoto, S., Fukushima, K. and Tanaka, T.: A new (k,n)-threshold secret sharing scheme and its extension, ISC 2008 Conference, (2008).

[12] Damgård, I., Pastro, V., Smart, N. and Zakarias, S.: Multiparty Computation from Somewhat Homomorphic Encryption, CRYPTO 2012, pp.643-662 (2012).

[13] Damgård, I., Keller, M., Larraria, E., et al.: Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits, ESORICS 2013, LNCS 8134, pp.1-18, Springer (2013).

[14] Keller, M., Orsiniy, E. and Schollz, P.: MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer:, Proc. the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp.830-842 (2016).

[15] Araki, T., Barak, A., Furukawa, J. et al.: Optimized Honest-Majority MPC for Malicious Adversaries — Breaking the 1 Billion-Gate Per Second Barrier, 2017 IEEE Symposium on Security and Privacy, pp.843-862 (2017).

[16] Chida, K., Genkin, D., Hamada, K., et al.: Fast Large-Scale Honest-Majority MPC for Malicious Adversaries, CRYPTO 2018, Shacham, H. and Boldyreva, A. (eds), LNCS, Vol.10993, pp.34-64, Springer (2018).

[17] Backes, M., Kate, A., Patra, A.: Computational Verifiable Secret Sharing Revisited, ASIACRYPT 2011, Lee D.H., Wang X. (eds), LNCS, Vol.7073, pp.590-609, Springer (2011).