少パーティの秘密分散ベース秘密計算のための $O(\ell)$ ビット通信ビット分解および $O(|p'|)$ ビット通信 Modulus 変換法

五十嵐　大 †　　濱田　浩気 †　　菊池　亮 †　　千田　浩司 †

† 日本電信電話株式会社 NTT セキュアプラットフォーム研究所
180-8585 東京都武蔵野市緑町 3-9-11

ikarashi.dai@lab.ntt.co.jp

あらまし 秘密計算における最も重要な演算の一つとして, ビット分解がある. ビット分解は, ℓ ビットの数値が格納された 1 つの暗号文 (もしくは秘密分散値) を, 1 ビットの真偽値の暗号文/秘密分散値から成る長さ ℓ の列に変換する操作である. また Modulus 変換は秘密分散値の法をある p から別の p' に変更する操作である.

本稿ではパーティ数の小さい (k, n)-秘密分散ベース秘密計算において, 非常に高速なビット分解および Modulus 変換を提案する. 通信量はシェアのビット長 $|p|, |p'|$(すなわち $\ell \leq |p|$ である) として $O(|p|^2)$ ビット, $O(|p|^2 + |p'|^2)$ ビットが従来最良であったのに対して, 提案手法は $O(\ell)$ ビット, $O(|p'|)$ となる. 体要素単位でなくビット単位であることに注意されたい. またビット分解の提案手法の出力は論理回路の実行に必要十分な mod 2 であり, 不可能な結果ではないことにも注意されたい.

$O(\ell)$ Bits Communication Bit Decomposition and $O(|p'|)$ Bits Communication Modulus Conversion for Small k Secret-sharing-based Secure Computation

Dai IKARASHI†　　Koki HAMADA†　　Ryo KIKUCHI†　　Koji CHIDA†

†NTT Secure Platform Laboratories, NTT Corporation
3-9-11, Midoricho, Musashino-shi, Tokyo 180-8585, Japan

ikarashi.dai@lab.ntt.co.jp

Abstract As one of the most significant operations in the field of secure computation, bit decomposition is known. Bit decomposition is an operation that converts one encrypted or shared value whose plaintext is a ℓ-bit numerical value into ℓ encrypted / shared values whose plaintexts are 1-bit truth-values. Modulus conversion is a conversion which change mod p shares into mod p' shares.

The paper proposes a highly efficient bit decomposition and modulus conversion for secure computations based on (k, n) secret sharing schemes with small k. The communication complexity of proposed protocols are $O(\ell)$ bits and $O(|p'|)$ bits, while $O(|p|^2)$ and $O(|p|^2+|p'|^2)$ were the best in existing protocols, where $|p|$ and $|p'|$ are the bit length of the underlying secret sharing scheme (i.e., $\ell \leq |p|$).

1　はじめに

近年, 暗号化/秘密分散されたデータに対して処理を行う, 秘密計算の研究が盛んである. 中でも特に秘密分散ベースの秘密計算は処理効率が比較的高く, sharemind[1, 2] など実用的な性能を持つ実装が現れ始めている. 秘密分散ベースの秘密計算は加法準同型性を持ち, 論理回路により任意の処理を実行できる汎用性と, 加算・乗算がネイティブ演算であり高速であるという特長を併せ持つ. 以降本稿で秘密計算と言えば秘密分散ベースのこととする.

1.1 通信量の表記について

従来通信量は体要素の単位で評価されることが多かったが, 本稿ではビットを単位として評価する. なぜなら筆者らは体を必要に応じて変化させながら処理を行うようなアーキテクチャを想定しているため, 変動する体要素の数では通信量を正確に測れないからである. このために各既存手法の通信量は本稿では各々における表記よりも, 体のビット長分次数が上がっている.

1.2 ビット分解に関する背景と関連研究

準同型性を持つ秘密計算において**ビット分解**は効率の観点で非常に重要な意味を持ち, 盛んに研究されている [5, 7, 11]. 加算・乗算を行うと平文は $\{0,1\}$ の元すなわちビットではなくなり, 任意の処理を実現する論理回路との互換性を失う. このとき再度任意の処理を行いたい場合に用いられるのがビット分解である. 操作としては, ℓ ビットの平文が格納された一つの秘密分散値を, ビットを格納した ℓ 個の秘密分散値に変換する操作である.

ビット分解は処理が重く秘密計算において大きな課題であった. 近年秘密計算の性能の指標の一つであるラウンド数については定数ラウンドの手法が多数提案 [5, 7, 11] されている. しかし通信量に関しては秘密分散の空間のビット数を $|p|$ として $O(|p|^2)$ ビットが最良である. 近年注目される大規模データ処理ではラウンド数よりも通信量の方が支配的であり, 依然としてビット分解は重要な課題である.

1.3 Modulus 変換に関する背景と関連研究

Modulus 変換はビット分解ほど盛んには研究されてはいないが, 秘密計算の実用の上では非常に重要な演算である. Modulus 変換は法を p とする秘密分散値を別の法 p' による秘密分散値に変換する操作である. 通常の計算では, 例えば 32bit 整数から 64bit 整数への型変換などに相当する. 秘密計算においては処理量とデータ量が法 p のビット長に比例するため, データを格納しうる最小の法の形式で扱うのが理想である. しかし通常の計算と同様, 他の値と組み合わせた処理等では形式を合わせる必要がある. また, 乗算等の際にはデータ長が入力よりも大きくなるため p を超えることがあり, このような際にもより大きな法 p' への変換は有効である. Modulus 変換において加藤ら [8] は定数ラウンドを実現した. しかしやはり通信量は二次である $O(|p|^2 + |p'|^2)$ ビットである.

1.4 本稿の貢献

本稿では**商転移**と呼ぶ新しい方法を用いて, 通信量 $O(\ell)$ ビットのビット分解および $O(|p'|)$ ビットの Modulus 変換を提案する. なお従来の体要素ベースの表記ではそれぞれ何と $O(0), O(1)$ となる. (ビット分解では $|p|$ に比例する通信を一切行わない. ℓ はシェア空間でなく平文空間のビット長である.) また実装実験によりそれぞれの semi-honest 版と malicious 版の実性能を示す.

2 準備

ここではまず (k,n)-線形秘密分散, 複製秘密分散, ビット分解の基本を紹介する.

2.1 (k,n)-線形秘密分散

ここでは (k,n)-線形秘密分散は以下のように定義する.

定義 2.1 ((k,n)-線形秘密分散)
関数 $\mathtt{SHARE} : \mathcal{R} \to \mathcal{R}'^n$ が (k,n)-線形秘密分散であるとは, 任意の単射 $\sigma : \{0,\cdots,k-1\} \to \{0,\cdots,n-1\}$ に対して以下の復元用の係数列が存在することとする. σ は n 個のシェアから k 個のシェアを任意に選択することを表現している. ただし $\mathcal{R}, \mathcal{R}'$ は環, C は $\mathcal{R}'$ との積が定義される集合とする.

(復元)
ある係数列 $(\lambda_0,\cdots\lambda_{k-1}) \in C^k$ が存在して, 任意の入力 a に対して $\sum_{i<k} \lambda_i \mathtt{SHARE}(a)_{\sigma(i)} = a.$

$\mathtt{SHARE}(a)$ を分散値と呼び, $[\![a]\!]$ と表記する. また各 $i \in \{0,\cdots n-1\}$ に対して $\mathtt{SHARE}(a)_i$ を $[\![a]\!]_i$ と表記し, i 番目の, またはパーティ i のシェアと呼ぶ.

Shamir の秘密分散は代表的な (k,n)-線形秘密分散である. Shamir の秘密分散では係数列は Lagrange 補完法における Lagrange 係数である.

2.2 複製秘密分散

複製秘密分散とは次のような秘密分散である. まず $m = {}_nC_{k-1}$ 個の可換群 $\mathcal{G}$ の元 $a_0,\cdots,a_{m-1}$ をもって平文 a を $a = \sum_{i<m} a_i$ とする. そして各

$k-1$ パーティの組 (i 番目の $k-1$ パーティ組 $\mathcal{P}_i$ とする) について, $\mathcal{P}_i$ に属**さない**パーティがみな a_i を持つ. $\mathcal{P}_i$ に属するパーティはみな a_i を持たない. 以上のようにすると, 任意の $k-1$ パーティ組に対して, 持たない a_i が存在するから $k-1$ パーティまでの結託に対して安全である. また k パーティ集まると, どの a_i も必ず誰かが持っているため, 復元できる. よって (k,n)-秘密分散となっている. 各 a_i を**サブシェア**と呼ぶ.

例えば $(2,3)$-複製秘密分散は, $a = a_0 + a_1 + a_2$ とし, 各パーティのシェアは (a_0, a_1), (a_1, a_2), (a_2, a_0) となる.

複製秘密分散は (k,n)-線形秘密分散の定義において, $\mathcal{R}' = C = \mathcal{R}^d$ であり, λ_i が 0 と 1 だけから成る d 次ベクトルと見なすことで, (k,n)-線形秘密分散であると見なすことができる.

本稿の semi-honest プロトコルでは (k,k)-複製秘密分散を用いる. (k,k)-複製秘密分散は, k 人だけでシェアを持つため効率的であるという利点がある.

また, (k,k)-複製秘密分散は任意の (k,n)-線形秘密分散からオフラインで簡単に変換できるという利点もある. すなわち, 任意に k パーティを選択し (簡略化のため本稿では $0, \cdots, k-1$ の k パーティとする.), 各 $i < k$ に対して以下のように復元時の係数を乗ずればよい.

$$\{a\}_i = \lambda_i [\![a]\!]_i$$

また Malicious 対応プロトコルでは (k,n)-複製秘密分散を用いる (Scheme 1).

(k,n)-複製秘密分散は, (k,k)-複製秘密分散とは逆に, 任意の (k,n)-線形秘密分散にオフラインで変換できる [4] という利点がある.

2.3 共通的な記法

- p, p': 素数.
- $|p|$, $(|p'|)$: 素数のビット長. すなわち $p < 2^e$ となる最小の $e \in \mathbb{N}$.
- ℓ: 平文空間のビット長. $\ell \leq |p|$ である.
- m: 複製秘密分散におけるサブシェアの数.

2.4 秘密分散値の表記の区別

ビット分解では数値を格納した 1 つのシェアを, ビットを格納した ℓ 個のシェアに変換する. これらを明確に区別して表記するため, 以下の記法を用いる.

- 数値 a の分散値: $[\![a]\!]^{\mathbb{Z}_p}$
- ビット b の分散値: $[\![b]\!]^{\mathbb{Z}_2}$
- ビット長 ℓ の数値 a のビット表現の ℓ 個の分散値列: $[\![a]\!]^{\mathbb{Z}_2^\ell}$
- 同じ法の任意の線形秘密分散と, 複製秘密分散が同時に現れることがあるが, この際は複製秘密分散を $\{a\}^{\mathbb{Z}_p}$ のように書いて区別する.

またプロトコル Scheme 2, Scheme 3 上では, 算術演算がどの代数構造上であるか明示する. 例えば $\mathbb{Z}_p$ 上乗算であれば $\times_{\mathbb{Z}_p}$ のようである.

2.5 ビット分解の基本

ビット分解の目的は, ℓ ビット数値 a を格納した分散値 $[\![a]\!]^{\mathbb{Z}_p}$ を, ビット表現の分散値列 $[\![a]\!]^{\mathbb{Z}_2^\ell}$ に変換することである.

ビット分解の最も基本的なアイデアは以下のようなアプローチである.

ビット分解の基本的処理

1. 各パーティ $i < k$ が,
   (a) シェアのビット毎分散: 自分のシェア $[\![a]\!]_i^{\mathbb{Z}_p}$ に, 復元用の係数 λ_i を乗じて $\lambda_i [\![a]\!]_i^{\mathbb{Z}_p}$ とする.
   (b) $\lambda_i [\![a]\!]_i^{\mathbb{Z}_p}$ のビット列表現の各ビットを秘密分散して $[\![\lambda_i [\![a]\!]_i^{\mathbb{Z}_p}]\!]^{\mathbb{Z}_2^\ell}$ とする
2. 加算: 加算回路の秘密計算により
   $$[\![\sum_{i<k} \lambda_i [\![a]\!]_i^{\mathbb{Z}_p}]\!]^{\mathbb{Z}_2^{|p|}} \left(= \sum_{i<k} [\![\lambda_i [\![a]\!]_i^{\mathbb{Z}_p}]\!]^{\mathbb{Z}_2^{|p|}} \right)$$
   を計算, すなわち復元を行う.
3. mod p: 加算回路だけでは mod p の処理が抜けているため, p との大小比較回路および p を減ずる減算回路の秘密計算の $k-1$ 回の反復により,
   $[\![a]\!]^{\mathbb{Z}_2^\ell} = [\![\sum_{i<k} \lambda_i [\![a]\!]_i^{\mathbb{Z}_p} \mod p]\!]^{\mathbb{Z}_2^\ell}$ を得る.

上記のようなアプローチにおける効率の観点での問題は, 以下の 3 点である.

1. step 3. で, 大小比較回路・減算回路の実行回数が $k-1$ 回もある

Scheme 1 [プロトコル] 任意の線形秘密分散から複製秘密分散への変換
入力: 線形秘密分散値 $[\![a]\!]^{\mathbb{Z}_p}$
出力: 複製秘密分散値 $\{a\}^{\mathbb{Z}_p}$

1: 複製秘密分散の乱数 $\{r\}^{\mathbb{Z}_p}$ を生成する.
2: $\{r\}^{\mathbb{Z}_p}$ を所望の線形秘密分散 $[\![r]\!]^{\mathbb{Z}_p}$ に変換する.
3: $[\![a-r]\!]^{\mathbb{Z}_p}$ を計算して, malicious 対応の方式で公開し $a-r$ を得る.
4: $(a-r)+\{r\}^{\mathbb{Z}_p}=\{a\}^{\mathbb{Z}_p}$ を得る.

2. $[\![\cdot]\!]^{\mathbb{Z}_2^{|p|}}$ の形式の秘密分散値が $|p|^2$ ビットのサイズを要すること

3. step 2., step3. で, a のビット数に関わらず $|p|$ ビットの回路を実行する必要があること

Damgård らは文献 [5] において, $[\![a]\!]$ を乱数のシェア $[\![r]\!]$ と公開値 $a-r$ の $(2,2)$-秘密分散に変換することで 1 番目の課題を解決した.

本稿では残りの 2., 3. の課題を解決する.

3 mod p シェアの商とその計算法 : 商転移

本節ではビット分解, Modulus 変換で共通に用いる, **商転移**と呼ぶ方法を提案する. 簡略化のため p は, 最も効率的に処理できる, メルセンヌ素数 (すなわち "2 べき −1 の素数") とする. アイデア自体はメルセンヌ素数以外でも利用可能である.

定理 3.1 (商転移)
k を自然数, $u=\lceil \log k \rceil$ とし, p をメルセンヌ素数とする. $a\in\mathbb{N}$ を, $a<p$ かつ $a \mod 2^u = 0$ を満たす任意の平文とする. また $(x_0,\cdots,x_{k-1})\in\mathbb{Z}_p^k$ を, $\sum_{i<k} x_i \underset{p}{\equiv} a$ を満たす任意の値の列とする. すなわち $\mathbb{Z}$ 上では, ある $q<k$ なる商 $q\in\mathbb{N}$ が存在して $\sum_{i<k} x_i = a+qp$ である.

この q に対して, 以下の等式が成り立つ.

$$q = -\sum_{i<k} x_i \mod 2^u \tag{1}$$

証明 式 (*1*) の右辺を考えると,

$$-\sum_{i<k} x_i \mod 2^u = -a-qp \mod 2^u$$

である. この右辺で mod を分配すると,

$$-(a \mod 2^u) + q(-(p \mod 2^u)) \mod 2^u \tag{2}$$

となる. 仮定より $a \mod 2^u = 0$ であることと, p がメルセンヌ素数であり $p \underset{2^u}{\equiv} -1$ となることから, 式 (*2*) はさらに

$$0 + q(-(-1)) \mod 2^u = q \mod 2^u$$

と変形される. $q<k<2^u$ であるから $q \mod 2^u = q$ であり, よって $q = -\sum_{i<k} x_i \mod 2^u$ である.

□ *(*定理 3.1*)*

定理の $a+qp$ のうち, a は p による剰余, q は商である. 通常であればシェアの和 $\sum_{i<k} x_i$(すなわち (k,n)-線形秘密分散の復元中の $\sum_{i<k}\lambda_i \mathrm{SHARE}(a)_{\sigma(i)}$ を計算してから除算により商 q を求める. これに対して定理 3.1 は, この値がシェアの和 $\sum_{i<k} x_i$ の下位 u ビットと等価となること, すなわち商が上位ビットから下位ビットに転移することを言っている. 例えば, $k=2$ のとき, 最下位ビットの 1 ビット加算だけで q の計算が行えることを示している.

4 商転移を用いたビット分解

商転移を利用すると効率的なビット分解を構成できる. 以下の定理はビット分解を導く公式である. 式 (3) は既に mod p 演算を含まず, mod 2^u や mod 2^ℓ という, シェアのビット表現から計算しやすい形式となっている. より単純な記号で書けば, 系 4.1 のようになる. これらをプロトコルとしたのが Scheme 2 である. Scheme **??**は, Scheme 2 で $k=2$ とした場合である. $k=2$ の場合特に, $q_u + [r_u \neq 0]$ が OR ゲート 1 回で求められ非常に効率的である.

定理 4.1 (ビット分解公式) □ 定理 4.1

$k, u, p, a, (x_0, \cdots, x_{k-1})$ を, 定理 3.1 と同様とする.

すると ℓ を $\ell + u \le |p|$ として以下の等式が成り立つ.

$$\frac{a}{2^u} \underset{2^\ell}{\equiv} \sum_{i<k} x_i \div 2^u + \left[\sum_{i<k} x_i \underset{2^u}{\not\equiv} 0\right] \tag{3}$$

ただし記法 $x \div y$ は x の y による商を表し, 論理式 X に対して $[X]$ は, X が真ならば 1, 偽ならば 0 を表すとする.

証明 q を $\sum_{i<k} x_i$ の p による商とする. すると $\sum_{i<k} x_i = a + qp$ であるから, $a = \sum_{i<k} x_i - qp$ であり, よって

$$a \underset{2^{\ell+u}}{\equiv} \sum_{i<k} x_i - qp \tag{4}$$

でもある.

p がメルセンヌ素数であることと $\ell + u \le |p|$ より, $p \underset{2^{\ell+u}}{\equiv} -1$ である. よって式 (4) に定理 3.1 を適用すれば,

$$a \underset{2^{\ell+u}}{\equiv} \sum_{i<k} x_i + \left(-\sum_{i<k} x_i \mod 2^u\right) \tag{5}$$

と変形される. $\sum_{i<k} x_i \mod 2^u = r_u$ とおけば,

$$-\sum_{i<k} x_i \mod 2^u = \begin{cases} 0 & r_u = 0 \text{ のとき} \\ 2^u - r_u & \text{そうでないとき} \end{cases} \tag{6}$$

である. よって $-\sum_{i<k} x_i \mod 2^u$ は, $2^u[r_u \neq 0] - r_u$ と表現できる. $\sum_{i<k} x_i$ の 2^u による商を q_u とおけば $\sum_{i<k} x_i = 2^u q_u + r_u$ であり, 従って式 (7) は以下のように変形される.

$$\begin{aligned} a &\underset{2^{\ell+u}}{\equiv} 2^u q_u + r_u + (2^u[r_u \neq 0] - r_u) \\ &= 2^u(q_u + [r_u \neq 0]) \end{aligned}$$

a の仮定より両辺とも 2^u で割り切れ, 結局

$$\begin{aligned} \frac{a}{2^u} &\underset{2^{\ell+u}}{\equiv} q_u + [r_u \neq 0] \\ &= \sum_{i<k} x_i \div 2^u + \left[\sum_{i<k} x_i \underset{2^u}{\not\equiv} 0\right] \end{aligned}$$

となる.

系 4.1 $k, u, p, a, (x_0, \cdots, x_{k-1})$ を, 定理 3.1 と同様とする. さらに各 q_i, r_i をそれぞれ x_i の u ビット目以降および $u-1$ ビット以前の表現する数値とし, q_u, r_u を $\sum_{i<k} r_i$ の u ビット目以降および $u-1$ ビット以前の表現する数値とする. そして入力 a' を, $a = 2^u a'$ なる数とする. すると ℓ を $\ell + u \le |p|$ として以下の等式が成り立つ.

$$a' \underset{2^\ell}{\equiv} \sum_{i<k} q_i + q_u + [r_u \neq 0] \tag{7}$$

4.1 ビット分解プロトコル

ここでビット分解プロトコルを示す (Scheme 2). 系 4.1 の公式に従って処理している. 入力は複製秘密分散となっており, 特に semi-honest では (k, k), malicious では (k, n) を想定している. 2.2 節に書いたようにどちらの場合も任意の線形秘密分散値から変換できるため, 適用が限定されるものではない. 特に (k, k)-複製秘密分散への変換はオフラインのため高速である.

4.2 秘匿性

Scheme 2 は $[\![\cdot]\!]^{\mathbb{Z}_2}$ 上の秘密分散, 加算回路および零判定回路から成った. よってこれら 4 つが無条件秘匿性 (unconditional privacy) をもつならば Scheme 2 はやはり無条件秘匿性をもつ.

4.3 効率

4.4 通信量

通信量は簡略化して書けば $k(\ell+u)\mathtt{SHARE}_{\mathbb{Z}_2} + (k-1)\mathtt{add}_{\ell+u} + \mathtt{zero\_test}_u$ 程度である. $\mathtt{SHARE}_{\mathbb{Z}_2}$ はビット分散値の分散処理の通信量, $\mathtt{add}_{\ell+u}$ は $\ell+u$ ビットの加算回路の通信量, $\mathtt{zero\_test}_u$ は u ビットの零判定回路の通信量である. 加算回路, 零判定回路は最も基本的な, 回路サイズ最適の回路として計算した. なお $|p|$ に依存した通信は一切発生しない. k, n を小さい数として定数と見ると, オーダーは $O(\ell)$ となる.

$k = 2, n = 3$ で $[\![\cdot]\!]^{\mathbb{Z}_2}$ を mod 2 の複製秘密分散としてより具体的に計算すると, 総通信量 $10\ell+4$ ビットである. ただし mod 2 上では XOR がオフラインで処理でき, 1 ビット全加算器が 1 乗算で実現されることに注意 [6]. mod p 上乗算が $6|p|$

Scheme 2 [プロトコル] 複製秘密分散のビット分解
入力: mod p 複製秘密分散値 $[\![a]\!]^{\mathbb{Z}_p}$, ただし $2^u a < p$, $u = \lceil \log m \rceil$
出力: ビットの秘密分散値列 $[\![a \mod 2^\ell]\!]^{\mathbb{Z}_2^\ell}$

1: 公開値倍の秘密計算により $[\![a']\!]^{\mathbb{Z}_p} := 2^u \times_{\mathbb{Z}_p} [\![a]\!]^{\mathbb{Z}_p}$ とする.
2: **for** $i < m$ **do** (in parallel)
3: $\quad [\![a']\!]_i^{\mathbb{Z}_p}$ の 0 ビット目から u ビットをビットごとに分散して $[\![r_i]\!]^{\mathbb{Z}_2^u}$ とおく.
4: $\quad [\![a']\!]_i^{\mathbb{Z}_p}$ の u ビット目から ℓ ビットをビットごとに分散して $[\![q_i]\!]^{\mathbb{Z}_2^\ell}$ とおく.
5: 加算回路の秘密計算により $\sum_{i<k}^{\mathbb{Z}_{2^{2u}}} [\![r_i]\!]^{\mathbb{Z}_2^u}$ を計算し, 結果のうち下位 u ビットを $[\![r_u]\!]^{\mathbb{Z}_2^u}$, 上位 u ビットを $[\![q_u]\!]^{\mathbb{Z}_2^u}$ と見なす.
6: 零判定回路の秘密計算により $[\![[r_u \neq 0]]\!]^{\mathbb{Z}_2}$ を計算する.
7: 加算回路の秘密計算により $\sum_{i<k}^{\mathbb{Z}_{2^\ell}} [\![q_i]\!]^{\mathbb{Z}_2^\ell} +_{\mathbb{Z}_{2^\ell}} [\![q_u]\!]^{\mathbb{Z}_2^u} +_{\mathbb{Z}_{2^\ell}} [\![[r_u \neq 0]]\!]^{\mathbb{Z}_2}$ を計算して出力する.

ビット通信量であることを考えると, これは例えば ℓ が $|p|/2$ 程度のとき, 遅い演算として研究が重ねられてきたビット分解がなんと mod p 上乗算 1 回よりも軽量であることを意味している!

4.5 ラウンド数

ラウンド数は, 通信量同様回路サイズ最適として計算した場合で, $\ell + u$ である. $k = 2, n = 3$ で, $\ell + 1$ ラウンドである. 定数ラウンドのビット分解が近年数多く提案されているが, 例えば $\ell = 32$ のとき提案手法は 33 であり, 最良のラウンド数である文献 [7] の 25 ラウンドと大きな遜色はない.

5 Modulus 変換

Modulus 変換は, mod p で分散した分散値を, 平文を秘匿したまま法を別の値 p' として mod p' の分散値に変換する処理である. Modulus 変換もビット分解同様, 商の算出が重要である. なぜなら加法的秘密分散で $\sum_{i<k} x^i = a + qp$ と表せば, mod p' では qp が消えないためこの秘密分散値の平文は $a + qp \mod p'$ である. よって $q(p \mod p')$ を計算して減算すればよいこととなる. p と p' が公開のため $(p \mod p')$ は平文で計算できることに注意.

Scheme 3 に Modulus 変換の提案手法を示す. mod 2, mod p, mod p' の 3 つの mod が現れるので混同しないよう注意されたい. ビット分解同様, 入力は複製秘密分散であるが 2.2 節のように任意の線形秘密分散から変換できるため適用は限定されない.

また mod 2 → mod p' 変換は, mod 2 の乱数の (k, n)-複製秘密分散値を生成して m 個の mod p' 上 (k, n)-複製秘密分散値に変換 (Scheme 5) し, これら m 個の XOR を秘密計算で得れば良い.

5.1 秘匿性

Scheme 3 はビット分解と同様, 無条件秘匿のプロトコルとオフライン処理のみから成る. そのため Scheme 3 と同様, 無条件秘匿である.

5.2 効率

5.2.1 通信量

通信は step 3: の分散, step 4: の加算回路, step 5: の mod 2 → mod p' 変換で行われる.

step 3: は m 回の分散であり通信量 $O(nm^2)$ ビット, step 4: は mu 回の $\mathbb{Z}_2$ 上ビット AND であり, $O(m^2 u)$ ビット, step 5: は n 回の p' 上ビット XOR であり $O(nmp')$ ビットである. k, n を小さい数として定数と見れば, 合計で $O(p')$ である.

特に $k = 2, n = 3$ では step 3: は通信量 4 ビット, step 4: は $k = 2$ では XOR のみのため 0 ビット, step 5: は $3+9p'$ ビットである. よって $9p'+7$ ビットである. これは乗算 1 回分の $6p'$ ビットと比較して高々1.5 倍程度である.

5.3 ラウンド数

ラウンド数は step 3: は $O(1)$, step 4: は $O(k + u) = O(k)$, step 5: は $O(\log n)$ であり, 合計 $O(k + \log n)$ である. k, n を定数と見れば $O(1)$ となる.

Scheme 3 [プロトコル] 複製秘密分散値の Modulus 変換
入力: mod p 複製秘密分散値 $[\![a]\!]^{\mathbb{Z}_p}$, ただし $2^u a < p$, $u = \lceil \log m \rceil$
出力: mod p' 複製秘密分散値 $[\![a]\!]^{\mathbb{Z}_{p'}}$

1: $[\![a']\!]^{\mathbb{Z}_p} := 2^u \times_{\mathbb{Z}_p} [\![a]\!]^{\mathbb{Z}_p}$ とする.
2: **for** $i < m$ **do** (in parallel)
3: $\quad -_{\mathbb{Z}_{2^u}} [\![a']\!]_i^{\mathbb{Z}_p} \mod 2^u$ の 0 ビット目から u ビットを分散し, $[\![r_i]\!]^{\mathbb{Z}_2^u}$ を得る. マイナスが $\mathbb{Z}_p$ 上ではなく $\mathbb{Z}_{2^u}$ 上となっていることに注意.
4: 加算回路の秘密計算により $\sum_{i<k}^{\mathbb{Z}_{2^u}} [\![r_i]\!]^{\mathbb{Z}_2^u}$ を計算し, $[\![q]\!]^{\mathbb{Z}_2^u}$ とする.
5: mod 2 → mod p' 変換等, 所定の変換処理により $[\![q]\!]^{\mathbb{Z}_2^u}$ を $[\![q]\!]^{\mathbb{Z}_{p'}}$ に変換する.
6: **for** $i < m$ **do** (in parallel)
7: $\quad [\![a']\!]_i^{\mathbb{Z}_p}$ を mod p' のシェアと見なして $[\![a'_{p'}]\!]_i^{\mathbb{Z}_{p'}}$ とおく. 定理 3.1 より, $a'_{p'} \underset{p'}{\equiv} 2^u a + qp$ となっている.
8: 加算および公開値倍の秘密計算を用いて $2^{-u} \times_{\mathbb{Z}_{p'}} ([\![a'_{p'}]\!]^{\mathbb{Z}_{p'}} -_{\mathbb{Z}_{p'}} p[\![q]\!]^{\mathbb{Z}_{p'}})$ を計算して出力する.

Scheme 4 [プロトコル (オフライン)] 複製秘密分散のサブシェアに任意の処理 f を行ってまた複製秘密分散値とする
入力: $n-k+1$ パーティが複製秘密分散値のサブシェア $x_i^{\mathbb{Z}_p}$ を持つ
出力: 複製秘密分散値 $\{f(x_i)\}^{\mathbb{Z}_{p'}}$ (入力と異なる法でもよい)

1: $n-k+1$ パーティは $f(x_i)$ を計算する.
2: この $n-k+1$ パーティのサブシェアを $f(x_i)$ とする.
3: 他のサブシェアは全て 0 とする.

6 Malicious 対応

Malicious 対応のプロトコルは普通, 演算ごとに改ざん検知を行うことで実現される. 例えば Scheme 2, Scheme 3 における加算回路などの論理回路は, [10, 9] などにより改ざん検知可能である. しかし各パーティのシェアをビットごとに分散する部分の改ざん検知はそのようなアプローチではうまくいかない. なぜならビットごとの分散は代数構造を無視した操作であり, 代数を用いた改ざん検知が適用できないからである.

本稿ではこの解決に, (k, n)-複製秘密分散を用いる. Modulus 変換における mod 2 → mod p' でも用いた Scheme 5 を用いれば, サブシェアの各ビットの秘密分散がオフラインで行うことができる.

後は残りの部分は乗算と公開で構成されるから, [10, 9] 等の malicious 対応乗算および [10] に記した malicious 対応公開等を用いればよい.

7 実装実験

表 1 に提案手法の実機性能を示す. 乗算の malicious 対応には [9] を用いた. 入力は Shamir の秘密分散値である. ビット分解に関しては semi-honest でも malicious でも, sharemind の直近の実装 [3] の, 25,000 回/s 程度を遙かに上回っている. 特に $\ell = 2$ の際のパフォーマンスは圧倒的である.

Modulus 変換に関しては実装レベルの比較対象は無い.

全体的に malicious 対応が semi-honest よりも 10 倍以上遅くなっているが, 入力に対して回路が小さいため [9] の malicious 対応のオーバーヘッドのうち回路サイズに依存しない部分が大きく影響していると考えられる.

8 おわりに

本稿ではパーティ数が小さい場合に適用できる, 効率的なビット分解および Modulus 変換の方法を提案した.

ビット分解の通信量は従来 $O(|p|^2)$ に対して $O(\ell)$ であり非常に効率的である. ただし $|p|$ はシェア空間のビット長, ℓ は平文空間のビット長であり一般に $\ell \leq p$ である.

Modulus 変換は従来 $O(|p|^2 + |p'|^2))$ に対して $O(|p'|)$ でありやはり非常に効率的である.

Scheme 5 [プロトコル] 任意の線形秘密分散に対する malicious モデル上のビット分解および Modulus 変換

入力: 線形秘密分散値 $[\![a]\!]^{\mathbb{Z}_p}$

出力: $[\![a]\!]^{\mathbb{Z}_p}$ に対するビット分解または Modulus 変換の結果

1: Scheme 1 を用いて $[\![a]\!]^{\mathbb{Z}_p}$ を (k, n)-複製秘密分散 $\{a\}^{\mathbb{Z}_p}$ に変換する.

2: $\{a\}^{\mathbb{Z}_p}$ のサブシェアの各ビットの分散に Scheme 5 を用いて, ビット分解または Modulus 変換を行う. それらの中の乗算および公開は文献 [10, 9] 等任意の方法により malicious 対応とする.

表 1: 提案手法の実機上の性能 $((k, n) = (2, 3), p = 2^{31} - 1, p' = 2^{61} - 1)$

	処理の所要時間 [ms]			
適用手法	10 万件	100 万件	1,000 万件	1,000 万件の速度 [M 件/s]
ビット分解 ($\ell = 2$, semi-honest)	16	34	152	65.789
ビット分解 ($\ell = 2$, malicious)	212	391	3,945	2.535
ビット分解 ($\ell = 29$, semi-honest)	426	513	1,718	5.821
ビット分解 ($\ell = 29$, malicious)	1,351	1,690	25,103	0.398
Modulus 変換 (semi-honest)	38	156	2,069	4.833
Modulus 変換 (malicious)	329	2,455	28,099	0.356

[環境] CPU: 2.5GHz x 2 core, memory: 4GB, Network: 600Mbps

参考文献

[1] sharemind news blog. `http://sharemind.cyber.ee/`.

[2] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In S. Jajodia and J. López eds., *ESORICS*, Vol. 5283 of *Lecture Notes in Computer Science*, pp. 192–206. Springer, 2008.

[3] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.

[4] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In J. Kilian ed., *TCC*, Vol. 3378 of *Lecture Notes in Computer Science*, pp. 342–362. Springer, 2005.

[5] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In S. Halevi and T. Rabin eds., *TCC*, Vol. 3876 of *Lecture Notes in Computer Science*, pp. 285–304. Springer, 2006.

[6] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In J. A. Garay, A. Miyaji, and A. Otsuka eds., *CANS*, Vol. 5888 of *Lecture Notes in Computer Science*, pp. 1–20. Springer, 2009.

[7] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In T. Okamoto and X. Wang eds., *Public Key Cryptography*, Vol. 4450 of *Lecture Notes in Computer Science*, pp. 343–360. Springer, 2007.

[8] 加藤遼, 桐淵直人, 西脇雄高, 吉浦裕. 定数ラウンド (k, n) 秘匿モジュロ変換プロトコルの提案. In *CSS2011*, 2011.

[9] 五十嵐大, 菊池亮, 濱田浩気, 千田浩司. 少パーティ数の秘密分散ベース秘密計算における効率的な malicious モデル上 simd 計算の構成法. In *CSS2013*, 2013.

[10] 五十嵐大, 千田浩司, 濱田浩気, 菊池亮. 非常に高効率な $n \geq 2k - 1$ malicious モデル上秘密分散ベースマルチパーティ計算の構成法. In *SCIS2013*, 2013.

[11] 大原一真, 鈴木幸太郎, 米山一樹, 太田和夫. マルチパーティ計算による定数ラウンドかつほぼ線形な通信量のビット加算プロトコル. In *SCIS2013*, 2013.