Computer Security Symposium 2020
26 - 29 October 2020

# 高効率なシフト量秘匿シフトプロトコルの構成による，速度と精度を両立する秘密計算上の浮動小数点数の実現

五十嵐 大[1,a)]

**概要**：本稿では 3 パーティの秘密分散ベース秘密計算における高効率なシフト量秘匿シフトプロトコルを提案し，これをキーとして高速な秘密計算上の浮動小数点数のアルゴリズム群を設計・実装する．本稿では性能確保のため固定小数点とのハイブリッドを前提として負荷の高い MSB 合わせ処理を減らしていることも相まって，浮動小数点で問題となる加算において，Sharemind の実装より 200 倍程度高スループットであった．

**キーワード**：秘密計算，ビットシフト，浮動小数点

# Realizing Secure Floating-Point Numbers via Constructing An Efficient Private Shift-Amount Shift Protocol

Dai IKARASHI[1,a)]

***Abstract:*** In this paper, we propose an efficient private shift-amount shift protocol on secret-sharing-based secure computation, and using it, design and implement efficient algorithms for floating-point number operations on secure computation. Especially, its addition, which was problematic in floating-point number operations, was around 200 times faster than the implementation of Sharemind, due to the efficient shift and the reduce of MSB-adjustment assuming a hybrid of fixed-point and floating-point.

## 1. はじめに

秘密計算は暗号化したまま処理が可能な暗号技術であり，複数者のデータを秘匿したまま集約して処理をする統合データ分析などの有望な応用が考えられている．秘密計算は性能が課題であったが近年目覚ましい性能向上を遂げており，AI などの先端的な処理も現実的となりつつある．

筆者らはこれまでに，固定小数点を用いた四則演算や初等関数実数を設計・実装してきた．しかし，固定小数点は浮動小数点と同ビット数の精度でも実際の処理精度に問題が発生する場合がある．例えば，指数関数の出力など値の範囲が極端に広いデータでは，値が大きい場合にオーバーフローしないようにすると，小数点以下の領域が減り，値が小さいときの精度が落ちてしまう．またユーザの入力したデータが想定より小さい場合なども，小数点以下の領域が小さいために精度が不足する．

そこで本稿では，実数演算における精度と秘密計算による浮動小数点演算を構成し，固定小数点とのハイブリッドにより性能と演算精度の両立を目指す．すなわち，精度に問題のない箇所では固定小数点で処理を行い，固定小数点では精度に問題のある場合に必要に応じて浮動小数点を利用する．このようなモチベーションのため，本稿では厳密な浮動小数点は目指さない．すなわち，浮動小数点は $2^\rho x$ のように仮数部 $x$ と指数部 $\rho$ から成り，精度を最大に保つため処理の度に仮数部の MSB(Most Significant Bit) 位置を一定に保つことで $x$ を $1 \sim 2$ に収めておくことが多い．しかし MSB を合わせる処理は秘密計算ではそれなりに重く，かつ固定小数点に変換する前提では不要な処理のため，連続する浮動小数点処理の中でさらに高い精度が必要な場合のみ行うことにする．

構成の中で興味深いのは，数値の指数を入力して左シフトを行う，乗法的ローテーションである．指数 $\rho$ を数値で入力して，$2^\rho$ を計算するのは秘密計算においてはべき乗で

[1] NTT セキュアプラットフォーム研究所, NTT Secure Platform Laboratories
[a)] dai.ikarashi.rd@hco.ntt.co.jp

あり, 効率的に計算することはできなかった. 本稿では, ランダム置換の考え方を導入して, 乗算 2 回未満の通信量と 2 ラウンドでこれを実現した.

結果として, 入力とシフト量 (左右どちらなのかも含め) を秘匿するビットシフト演算であるシフト量秘匿左右シフトの効率的な方式を構成し, これを使って，浮動小数点加算・乗算および, 固定/浮動小数点の効率的かつ高精度なベクトル和・積和の方式を示す.

### 1.1 関連研究

浮動小数点のアルゴリズムおよび実装としては, アルゴリズムにおいては Kamm ら [1], 天田ら [2] などがある. 前者は効率の記載がなく, 比較は難しい. 後者については記載がある. しかし本稿で扱う浮動小数点は高速性を求めて固定小数点とのハイブリッドを前提としており, 1 演算ごとの MSB 合わせは行わない. 一方天田らの方式は MSB 合わせまで行う厳密な浮動小数点であり, テクニックの意味ではこれと比較することはフェアではない. ただしそのような, MSB 合わせを毎回は行わないアーキテクチャの提案自体も本稿の貢献の一つであるため, 簡単に比較を行う.

実装に関しては, Jaak ら [3] が最良である.

また, シフト量秘匿シフトプロトコルに注目して効率化している研究はなく, 比較対象がない.

### 1.2 記法

- $k$: 秘密分散の閾値. 特に 2 を想定する.
- $n$: 秘密分散の分散数/秘密計算のパーティ数. 特に 3 を想定する.
- $P$: 素数. 特にメルセンヌ素数 $2^61-1$ を想定する.
- $p$: $P$ のビット数. $P$ がメルセンヌ素数のとき, やはり素数である. 61 を想定する.
- $Q$: 剰余環の位数. $P, p$ や, 浮動小数点の指数部に使う位数を含む一般的な位数を意味する. 指数部のシェアに用いた時は特に $2^{13}-1$ を想定する.
- $\ell$: 格納されるデータの最大ビット長. $p$ より小さいことを想定する.
- $\lambda$: 格納される指数部の最大ビット長. 10 以下を想定する.
- $[\![x]\!]^y$: mod $y$ 要素 $x$ を $(k,n)$-秘密分散したシェア.
- $\langle x\rangle^y$: mod $y$ 要素 $x$ を $(k,k)$-加法的秘密分散したシェア.
- $\langle\!\langle x\rangle\!\rangle^y$: mod $y$ 要素 $x$ を $(k,n)$-複製秘密分散したシェア. $(k,n)$-秘密分散であるため, $[\![x]\!]^y$ の形式のシェアに適用できるプロトコルはこのシェアにも適用できることに注意. この記法の場合, 特に複製秘密分散の性質を利用していることを意味する.
- $[\![x]\!]^{2^m}$: $[\![x]\!]^2$ の形式のシェアを $m$ 個並べたシェア. 数値のビット表現と見なすこともある.
- $\rho\circ\vec{a}$: ベクトル $\vec{a}$ にローテーション $\rho$ を施したベクトル. ローテーションは数でもあり置換でもあるため, 要素ごとの乗算 $\rho\vec{a}$ と区別する.
- $x\simeq y$: $x$ と $y$ は計算機上の実数として等しい. すなわち, 差が一定の誤差範囲内である.
- $a/d$: 小数点以下切り捨ての整数除算. 特に 2 べき数での整数除算は右シフトと等しい.
- $\dfrac{a}{d}$: 実数除算.
- [ 命題 ]: 命題が成り立てば 1, 成り立たなければ 0

### 1.3 準備：既存のプロトコル

乗法的ローテーションの理解のため, ローテーションプロトコル (シェア出力および公開値出力) を Scheme 1, Scheme 2 に示した. これらはランダム置換 [4] に基づく.

**Scheme 1** ローテーションプロトコル

**入力:** シェアのベクトル $[\![\vec{a}]\!]^P$, ローテーション量のシェア $\langle\!\langle\rho\rangle\!\rangle^Q$

**出力:** $[\![\rho\circ\vec{a}]\!]^P=[\![a_\rho]\!]^P,\cdots,[\![a_{n-1+\rho\bmod n}]\!]^P$

1: **ラウンド 1**
2: $[\![\vec{a}]\!]^2$ を, パーティ 0, 1 がシェアを持つ $(k,k)$-加法的秘密分散に変換し $\langle\vec{a}_0\rangle^2$ を得る.
3: パーティ 0, 1 は乱数 $\vec{r}_{01}$, パーティ 1, 2 は $\vec{r}_{12}$ を共有しておく.
4: パーティ 0 は $\vec{b}_0:=\langle\!\langle\rho\rangle\!\rangle^Q_{01}\circ\langle\vec{a}\rangle^P_0-\vec{r}_{01}$ を計算してパーティ 2 に送る.
5: パーティ 1 は $\vec{b}_1:=\langle\!\langle\rho\rangle\!\rangle^Q_{12}\circ(\langle\!\langle\rho\rangle\!\rangle^Q_{01}\circ\langle a\rangle^P_1+\vec{r}_{01})-\vec{r}_{12}$ を計算してパーティ 0 に送る.
6: **ラウンド 2**
7: パーティ 0 は $\langle\vec{c}\rangle^P_0:=\langle\!\langle\rho\rangle\!\rangle^Q_{20}\circ\vec{b}_1$ を計算する.
8: パーティ 2 は $\langle\vec{c}\rangle^P_2:=\langle\!\langle\rho\rangle\!\rangle^Q_{20}\circ(\langle\!\langle\rho\rangle\!\rangle^Q_{12}\circ\vec{b}_0+\vec{r}_{12})$ を計算する.
9: **ラウンド 3**
10: $\langle\vec{c}\rangle^P$ を $(k,n)$-秘密分散のシェア $[\![\vec{c}]\!]^P$ に変換して出力する. $\vec{c}=\rho\circ\vec{a}$ が成り立っている.

**Scheme 2** 公開値出力ローテーションプロトコル

**入力:** シェアのベクトル $[\![\vec{a}]\!]^p$, ローテーション量のシェア $\langle\!\langle\rho\rangle\!\rangle^Q$

**出力:** $\rho\circ\vec{a}=a_\rho,\cdots,a_{Q-1+\rho\bmod Q}$

1: **ラウンド 1**
2: $[\![\vec{a}]\!]^2$ を, パーティ 0, 1 がシェアを持つ $(k,k)$-加法的秘密分散に変換し $\langle\vec{a}_0\rangle^2$ を得る.
3: パーティ 0, 1 は乱数 $\vec{r}_{01}$, パーティ 1, 2 は $\vec{r}_{12}$ を共有しておく.
4: パーティ 0 は $\vec{b}_0:=\langle\!\langle\rho\rangle\!\rangle^Q_{01}\circ\langle\vec{a}\rangle^P_0-\vec{r}_{01}$ を計算してパーティ 2 に送る.
5: パーティ 1 は $\vec{b}_1:=\langle\!\langle\rho\rangle\!\rangle^Q_{01}\circ\langle a\rangle^P_1+\vec{r}_{01}$ を計算してパーティ 2 に送る.
6: **ラウンド 2**
7: パーティ 2 は $\vec{c}:=\langle\!\langle\rho\rangle\!\rangle^Q_{20}\circ(\langle\!\langle\rho\rangle\!\rangle^Q_{12}\circ(\vec{b}_0+\vec{b}_1)$ を計算してパーティ 0, 1 に送る. $\vec{c}=\rho\circ\vec{a}$ が成り立っている.

ほかに本稿では, 下記のプロトコルを部品として使う.

( 1 ) $(k,n)$-秘密分散から $(k,k)$-加法的秘密分散への変換: [5]

(2) $(k,k)$-加法的秘密分散から $(k,n)$-秘密分散への変換: [5]
(3) mod 2 → mod $Q$ 変換 [5]
(4) シフト量公開右シフト [6]
(5) 一括シフト量公開右シフト [7]
(6) モジュラス変換 (入力がメルセンヌ素数) [5]
(7) ビット分解 [5]

商転移 [5] の効果で, これらの部品の通信量には体のビット数より小さい, 格納される平文の上限のビット数 $\ell$ や $\lambda$ が登場することに注意.

## 2. 提案手法

### 2.1 要素演算

要素演算として, 乗法的ローテーション (Scheme 3), フラグ列 → 数値シェア変換 Scheme 4, 非商転移モジュラス変換 Scheme 5 を示す.

重要なのは前者 2 つであり, ランダム置換の考え方を利用する. メルセンヌ素数 $P$ を位数とする体では, 2 べき数を掛けるのは位数 $p$ のローテーションに相当する. ローテーションは置換であるから, ランダム置換と同じ考え方で, mod $p$ 数をべきとして 2 べき数を掛けることができる！

**Scheme 3** 乗法的ローテーションプロトコル (シフト量秘匿左シフトプロトコル)

**入力:** 数値シェア $\llbracket a \rrbracket^P$, ローテーション量のシェア $\langle\!\langle \rho \rangle\!\rangle^p$
**出力:** $\llbracket 2^\rho a \rrbracket^P$

**ラウンド 1**
$\llbracket a \rrbracket^P$ を, パーティ 0, 1 がシェアを持つ $(k,k)$-加法的秘密分散 $\langle a \rangle^P$ に変換する.
パーティ 0, 1 は乱数 $r_{01}$, パーティ 1, 2 は $r_{12}$ を共有しておく.
パーティ 0 は $b_0 := 2^{\langle\!\langle \rho \rangle\!\rangle^p_{01}} \langle a \rangle^P_0 - r_{01}$ を計算してパーティ 2 に送る.
パーティ 1 は $b_1 := 2^{\langle\!\langle \rho \rangle\!\rangle^p_{12}} (2^{\langle\!\langle \rho \rangle\!\rangle^p_{01}} \langle a \rangle^P_1 + r_{01}) - r_{12}$ を計算してパーティ 0 に送る.
**ラウンド 2**
パーティ 0 は $\langle c \rangle^P_0 := 2^{\langle\!\langle \rho \rangle\!\rangle^p_{20}} b_1$ を計算する.
パーティ 2 は $\langle c \rangle^P_2 := 2^{\langle\!\langle \rho \rangle\!\rangle^p_{20}} (2^{\langle\!\langle \rho \rangle\!\rangle^p_{12}} b_0 + r_{12})$ を計算する.
**ラウンド 3**
$\langle c \rangle^P$ を $(k,n)$-秘密分散のシェア $\llbracket c \rrbracket^P$ に変換して出力する. $c = 2^\rho a$ が成り立っている.

Scheme 4 では, 論理回路で計算した $p$ 個の "そのビット位置が MSB であるかどうか" を表すフラグ列を, 数値のシェアに変換する.

**Scheme 4** フラグ列 → 数値シェア変換プロトコル

**入力:** 長さ $|p|$ のビットシェアベクトル $\llbracket \vec{f} \rrbracket^2$, ただし $\vec{f}$ の中にはただ一つだけ 1 が存在する
**パラメータ:** $p$ のビット数 $|p|$
**出力:** $\vec{f}$ の中に存在する 1 の位置の mod $p$ シェア $\llbracket b \rrbracket^p$

mod $p$ 複製秘密分散上の一様乱数 $\langle\!\langle \rho \rangle\!\rangle^p$ を生成する.
公開値出力ローテーション (Scheme 2) により, 公開値 $\rho \circ \vec{f}$ を計算する.
$\rho$ が一様乱数で, $\vec{f}$ は 1 カ所だけ 1 と決まっているため, $\rho \circ \vec{f}$ は数値をビット位置として表現したローテーション上の一様乱数であり, 公開しても安全である.
$\rho \circ \vec{f}$ の 1 の位置を得て $b'$ とおく. $b'$ は元の 1 の位置 $b$ に対して, $b' = b + \rho$ となっている.
$\langle\!\langle b \rangle\!\rangle^p = b' - \langle\!\langle \rho \rangle\!\rangle^p$ を計算して出力する.

Scheme 5 は, 商転移の条件である, 空きビットが所定のビット数ある, という条件を満たさない場合の, 効率的な素体上のモジュラス変換である. プロトコル中で, $a'_0 + a_1 = a + qQ + 2^{|Q|} - Q = a + 2^{|Q|} - (1-q)Q$. $q = 0$ ならばこれは $2^{|Q|} - (Q - a)$ となり, $a < Q$ より $2^{|Q|}$ より小さい. $q = 1$ ならば $2^{|Q|} + a$ となり, $a \geq 0$ より $2^{|Q|}$ 以上である. すなわち $q = 1 \Leftrightarrow a'_0 + a_1 \geq 2^{|Q|}$ である. 故に $a'_0 + a_1$ の最上位である $|Q|$ 番目のビットは $q$ に等しい.

**Scheme 5** 非商転移モジュラス変換プロトコル

**入力:** 数値シェア $\llbracket a \rrbracket^p$.
**パラメータ:** $p$ のビット数 $|p|$
**出力:** 異なる法 $Q$ で分散されたシェア $\llbracket a \rrbracket^Q$

$\llbracket a \rrbracket^p$ を, パーティ 0, 1 がシェアを持つ $(k,k)$-加法的秘密分散 $\langle a \rangle^p$ に変換する.
パーティ 0 は $a'_0 := \langle a \rangle^p_0 + (2^{|p|} - p)$ を $\mathbb{Z}$ **上加算により** mod $p$ **せずに**計算し, その各ビットを秘密分散し $\llbracket a'_0 \rrbracket^{2^p}$ を得る.
パーティ 1 は $\langle a \rangle^Q_1$ の各ビットを秘密分散し, $\llbracket a_1 \rrbracket^{2^p}$ を得る.
加算回路により $a'_0 + a_1$ のビット表現のシェア $\llbracket a'_0 + a_1 \rrbracket^{2^{p+1}}$ を得る. 加算回路計算後はビット長が 1 増えることに注意.
$\llbracket a'_0 + a_1 \rrbracket^{2^{p+1}}$ の最上位ビットを $\llbracket q \rrbracket^2$ とおく. $q$ はシェア $\langle a \rangle^p$ の商, すなわち $\langle a \rangle_0 + \langle a \rangle_1 = a + qp$ と表したときの $q$ である.
mod 2 → mod $p$ 変換により, $\llbracket q \rrbracket^2$ から $\llbracket q \rrbracket^Q$ を得る.
パーティ 0, 1 はそれぞれ $\langle a \rangle^p_0, \langle a \rangle^p_1$ を mod Q シェアだと見なし, $\langle a' \rangle^Q$ を得る. $a' = a + qp$ が成り立っている.
$\langle a' \rangle^Q$ を $(k,n)$-秘密分散に変換し, $\llbracket a' \rrbracket^Q$ を得る.
$\llbracket a \rrbracket^Q = \llbracket a' \rrbracket^Q - p \llbracket q \rrbracket^Q$ を計算して出力する.

### 2.2 シフト量秘匿シフトプロトコル

乗法的ローテーションはローテーション量を秘匿したローテーションであり, これを用いてシフトを構成していく.

**Scheme 6** シフト量秘匿右シフトプロトコル

**入力:** 数値シェア $\llbracket a \rrbracket^P$, シフト量のシェア $\langle\!\langle \rho \rangle\!\rangle^p$, ただし $\rho \leq M$ とし, $a$ は $2^M a$ がオーバーフローでない範囲であるとする.

**パラメータ:** シフト量の上限 $M$

**出力:** $\rho$ ビット右シフトした値 $\llbracket a/2^\rho \rrbracket^P$

1: $\langle\!\langle M - \rho \rangle\!\rangle^p$ を計算する.
2: 乗法的ローテーション (Scheme 3) により左シフト $\llbracket 2^{M-\rho} a \rrbracket^P$ を計算する .
3: シフト量公開右シフトにより $M$ ビット右シフトして $\llbracket a/2^\rho \rrbracket = \llbracket 2^{M-\rho} a/2^M \rrbracket$ を得て出力する.

$a$ と $\rho$ の条件は, 右シフトの最大量 $M$ に対して, 左に $M$ ビットシフトしてもオーバーフローしないという条件である. 典型的には乗算に備えてシェアの有効ビット数の半分以下で秘密計算を行うため, 固定小数点の典型的状況ではそれほど苦しい条件ではない. しかし浮動小数点では指数部が著しく大きくなることもあり, このプロトコルだけでは苦しい.

**Scheme 7** シフト量秘匿左右シフトプロトコル-その 1:基本-

**入力:** 数値シェア $\llbracket a \rrbracket^P$, 正負ありうる左シフト量のシェア $\langle\!\langle \rho \rangle\!\rangle^Q$,

**パラメータ:** 入力の MSB 位置のとりうる上限 $M_{max}$

**出力:** $\rho$ ビットシフトした値 $\llbracket s \rrbracket^P$

1: 商転移を使うモジュラス変換により $\langle\!\langle \rho \rangle\!\rangle^p$ を計算する.
2: 大小比較により, $\llbracket f_L \rrbracket^2 := \llbracket [\rho \geq 0] \rrbracket^2$ を計算する.
3: mod 2 → mod $p$ 変換により $\langle\!\langle f_L \rangle\!\rangle^p$ を計算する.
4: $\langle\!\langle \rho' \rangle\!\rangle^p := \langle\!\langle \rho \rangle\!\rangle^p + M_{max} - M_{max} \langle\!\langle f_L \rangle\!\rangle$ を計算する.
5: 乗法的ローテーションにより $\llbracket b \rrbracket^P := \llbracket 2^{\rho'} a \rrbracket^P$ を計算する.
6: シフト量公開右シフトにより $\llbracket c \rrbracket^P := \llbracket 2^{\rho'} a/2^{M_{max}} \rrbracket^P$ を計算する.
7: mod 2 → mod $P$ 変換により $\llbracket f_L \rrbracket^P$ を計算する.
8: 積和により $\llbracket s \rrbracket := \llbracket c \rrbracket + (\llbracket b \rrbracket^P - \llbracket c \rrbracket^P) \llbracket f_L \rrbracket^P$ を計算する. この式は選択ゲートになっており, $s$ は $\rho < 0$ ならば $c$, $\rho \geq 0$ ならば $b$ となっていることに注意.

Scheme 7 は $\rho$ が正負いずれでも, すなわち右シフトか左シフトか分からなくても適用可能である. しかし, 右シフトにおいて Scheme 6 の性質から, 右シフトできる量に制限がある. 左シフトによるオーバーフローについては, 平文でも異常値となる (大きい 2 べきを掛けるという意味なのでより大きい値の出力が期待されるが, 実際には 0 となる) ため, ケアしないこととする. 右シフト量の範囲が大きい場合でも適用できるのが, Scheme 8 である. このプロトコルは, 数値と指数を入力として数値を指数分シフトするため, 浮動小数点から固定小数点への変換処理でもある.

**Scheme 8** シフト量秘匿左右シフトプロトコル-その 2:右シフト量の範囲が大きい-

**入力:** 数値シェア $\llbracket a \rrbracket^P$, 正負ありうる左シフト量のシェア $\langle\!\langle \rho \rangle\!\rangle^Q$,

**パラメータ:** 入力の MSB 位置のとりうる上限 $M_{max}$, シェアが許容する最大の MSB 位置 $M_{lim}$

**出力:** $\rho$ ビットシフトした値 $\llbracket s \rrbracket^P$

1: $u := M_{lim} - M_{max}$ とおき, $d := \lceil \frac{M_{max}}{u} \rceil$ とおく. $u$ は, 一回のシフト量秘匿右シフトでできる右シフト量であり, $d$ は $M_{max}$ ビット右シフトするのに必要なシフト量秘匿右シフトの回数である.
2: 商転移を使うモジュラス変換により $\langle\!\langle \rho \rangle\!\rangle^p$ を計算する.
3: 大小比較により, $\llbracket f_0 \rrbracket^2 := \llbracket [\rho \geq -M_{max}] \rrbracket^2$, $\llbracket f_1 \rrbracket^2 := \llbracket [\rho \geq -M_{max} + u] \rrbracket^2, \cdots$, $\llbracket f_{d-1} \rrbracket^2 := \llbracket [\rho \geq -M_{max} + (d-1)u] \rrbracket$, $\llbracket f_L \rrbracket^2 := \llbracket [\rho \geq 0] \rrbracket^2$ を計算する. $f_L$ ならば $f_{d-1}$, $f_{d-1}$ ならば $f_{d-2}, \cdots$ が成り立つ, 推移的なフラグであることに注意.
4: mod 2 → mod $p$ 変換により, $\langle\!\langle f_1 \rangle\!\rangle^p, \langle\!\langle f_2 \rangle\!\rangle^p, \cdots, \langle\!\langle f_{d-1} \rangle\!\rangle^p, \langle\!\langle f_L \rangle\!\rangle^p$ を計算する. $\langle\!\langle f_0 \rangle\!\rangle^p$ は不要である.
5: $\langle\!\langle \rho' \rangle\!\rangle^p :=$ $\langle\!\langle \rho \rangle\!\rangle^p + M_{max} - u(\sum_{1 \leq i < d} \langle\!\langle f_i \rangle\!\rangle^p) + ((d-1)u - M_{max}) \langle\!\langle f_L \rangle\!\rangle^p$ を計算する.
6: 乗法的ローテーションにより $\llbracket b \rrbracket^P := \llbracket 2^{\rho'} a \rrbracket^P$ を計算する.
7: 一括シフト量公開右シフトにより, $\llbracket c_0 \rrbracket^P := \llbracket 2^{\rho'} a/2^{M_{max}} \rrbracket^P$, $\llbracket c_1 \rrbracket^P := \llbracket 2^{\rho'} a/(2^{M_{max}} - u) \rrbracket^P, \cdots$ $\llbracket c_{d-1} \rrbracket^P := \llbracket 2^{\rho'} a/(2^{M_{max}} - u(d-1)) \rrbracket^P$ を計算する.
8: mod 2 → mod $P$ 変換により $\llbracket f_0 \rrbracket^P, \llbracket f_1 \rrbracket^P, \cdots, \llbracket f_{d-1} \rrbracket^P, \llbracket f_L \rrbracket^P$ を計算する. 今度は $\llbracket f_0 \rrbracket^P$ が必要である.
9: 積和により $\llbracket s \rrbracket :=$ $\llbracket c_0 \rrbracket^P \llbracket f_0 \rrbracket^P + (\llbracket c_1 \rrbracket - \llbracket c_0 \rrbracket)^P \llbracket f_1 \rrbracket^P + \cdots$ $+ (\llbracket c_{d-1} \rrbracket - \llbracket c_{d-2} \rrbracket)^P \llbracket f_{d-1} \rrbracket^P + (\llbracket b \rrbracket^P - \llbracket c_{d-1} \rrbracket^P) \llbracket f_L \rrbracket^P$ を計算して出力する. この式は推移的なフラグに対する選択ゲートになっており, $s$ は, $f_*$ が全て 0 ならば 0, $f_0$ ならば $c_0$, $f_1$ ならば $c_1, \cdots$, $f_L$ ならば $b$ となっていることに注意.

### 2.3 MSB 合わせ

ここでは MSB(Most Significant Bit) を固定位置に移動する, MSB 合わせを提案する (Scheme 9). MSB 合わせは実数演算の構成でも用いてきた ([7] など) が, 本稿では固定小数点を浮動小数点に変換するという意味をもつ. 乗法的ローテーションとフラグ列 → 数値変換を用いた, 新しいプロトコルを提案する. [7] では効率のため, 直接乗算に利用できる mod $P$ の 2 べき値を, ビット演算主体で構成していた. しかし, その場合, $p$ を超えるような指数を扱うことができず, $p = 61$ では ±30 程度までであり, 浮動小数点への

拡張には向かなかった．本稿の方式は, 1 回の指数部の出力は mod $p$ だが, モジュラス変換により大きいモジュラスに自由に変換して演算可能である．

---

**Scheme 9** MSB 合わせ (新)

---

**入力:** 数値シェア $[\![ a ]\!]^P$
**パラメータ:** 入力の最大ビット数 $\ell$
**出力:** $[\![ 2^\rho a ]\!]^P, \langle\!\langle \rho \rangle\!\rangle^p$, ただし $2^\rho a$ の MSB は $\ell - 1$ ビット目

---

1: ビット分解により $[\![ a ]\!]^P$ のビット表現 $[\![ a ]\!]^{2^\ell}$ を得る．
2: $0 \le i < \ell - 1$ で帰納的に, $[\![ f_i ]\!]^2 := [\![ f_{i+1} \vee a_i ]\!]^2$ とする．
ただし $[\![ f_{\ell-1} ]\!]^2 := [\![ a_{\ell-1} ]\!]^2$.
ここまでで, $0, 0, 0, 1, 1, \cdots, 1$ のような, msb を境に 01 が並ぶ形になっている．
3: $0 \le i < \ell - 1$ で, $[\![ x_i ]\!]^2 := [\![ f_i \oplus f_{i+1} ]\!]^2$ とする．
ただし $[\![ x_{\ell-1} ]\!]^2 := [\![ a_{\ell-1} ]\!]^2$.
ここまでで, $0, 0, 0, 1, 0, \cdots, 0$ のように, msb 位置のみ 1 となるフラグになっている．
4: (入力に 0 があり得る場合) 入力が 0 の場合にも 1 が入るように, $[\![ x_\ell ]\!]^2 = [\![ 1 ]\!]^2$ とする．
5: フラグ列 → 数値シェア変換 (Scheme 4) により $[\![ x_\ell ]\!]^2, [\![ x_{\ell-1} ]\!]^2, \cdots [\![ x_0 ]\!]^2$(降順であることに注意) を変換して $\langle\!\langle \rho \rangle\!\rangle^p$ とする．
6: 乗法的ローテーション (Scheme 3) により $[\![ 2^\rho a ]\!]^P$ を計算して出力する．

---

### 2.4　浮動小数点加算・乗算

いよいよ浮動小数点演算 (加算・乗算) を構成する．これらは要素演算から, 素直に構成する．

本稿では浮動小数点は $([\![ a ]\!]^P, [\![ \rho_a ]\!]^Q)$ というペアを想定する．すなわち, 実数 $x$ を, $x = 2^{\rho_a} a$ のように表す．

ふつう浮動小数点では $a$ に相当する仮数部は, 常に MSB が固定の位置に揃うようになっている．しかし本稿では, 処理効率のためその部分にはこだわらず, 上記のペアの形であれば, $a$ の MSB 位置に関わらず浮動小数点と呼ぶ．なお, 符号は $a$ 内部で $P/2$ 以上を負として表現する．

---

**Scheme 10** 浮動小数点数加算

---

**入力:** 浮動小数点数 $([\![ a ]\!]^P, [\![ \rho_a ]\!]^Q)$, $([\![ b ]\!]^P, [\![ \rho_b ]\!]^Q)$
**出力:** $([\![ c ]\!]^P, \langle\!\langle \rho_c \rangle\!\rangle^Q)$, ただし $2^{\rho_a} a + 2^{\rho_b} b \simeq 2^{\rho_c} c$

---

1: $[\![ \rho_a ]\!]^Q$, $[\![ \rho_b ]\!]^Q$ を大小比較し, $[\![ c ]\!]^2 := [\![ [\rho_a \ge \rho_b] ]\!]^2$ を得る．
2: mod 2 → mod $P$ 変換, mod 2 → mod $Q$ 変換により, $[\![ c ]\!]^P$, $[\![ c ]\!]^Q$ を得る．
3: $[\![ \rho_g ]\!]^Q := [\![ c?\rho_a : \rho_b ]\!]^Q (= [\![ ca + (1-c)b ]\!]^Q)$ を計算する．
$[\![ \rho_a ]\!]^Q$ と $[\![ \rho_b ]\!]^Q$ のうち大きい方である．
4: $[\![ \rho_s ]\!]^Q := [\![ c?\rho_b : \rho_a ]\!]^Q$ を計算する．
$[\![ \rho_a ]\!]^Q$ と $[\![ \rho_b ]\!]^Q$ のうち小さい方である．
5: $[\![ g ]\!]^P := [\![ c?a : b ]\!]^P$ を計算する．
$[\![ a ]\!]$ と $[\![ b ]\!]$ のうち指数部の大きい方である．
6: $[\![ s ]\!]^P := [\![ c?b : a ]\!]^P$ を計算する．
$[\![ a ]\!]$ と $[\![ b ]\!]$ のうち指数部の小さい方である．
7: $[\![ \rho_{dif} ]\!] := [\![ \rho_g - \rho_s ]\!]^Q$ を計算する．
8: mod $Q$ → mod $p$ 変換により, $\langle\!\langle \rho_{dif} \rangle\!\rangle^Q$ を計算する．
9: Scheme 8 により $[\![ s ]\!]^P$ を $\langle\!\langle -\rho_{dif} \rangle\!\rangle^Q$ でシフトし, $[\![ s' ]\!]^P$ とする．
ただし $\rho_{dif}$ が非負なので右シフトとなるため, 左シフトの分岐は省いてよい．
$(s', \rho_g) \simeq (s, \rho_s)$ となっている．
10: $([\![ g + s' ]\!]^P, \langle\!\langle \rho_g \rangle\!\rangle)$ を出力する．

---

**Scheme 11** 浮動小数点数乗算

---

**入力:** 浮動小数点数 $([\![ a ]\!]^P, [\![ \rho_a ]\!]^Q)$, $([\![ b ]\!]^P, [\![ \rho_b ]\!]^Q)$
**出力:** $([\![ c ]\!]^P, \langle\!\langle \rho_c \rangle\!\rangle^Q)$, ただし $2^{\rho_a} a 2^{\rho_b} b \simeq 2^{\rho_c} c$

---

1: $[\![ ab ]\!]^P$ を計算する．
2: 必要に応じて $[\![ ab ]\!]^P$ をビットシフトし, $[\![ c ]\!]^P$ とする．
(入力のビット数がある程度既知であったり, MSB が調整してあるなどの理由で $a, b$ のビット数がある程度高いことが既知である場合) シフト量公開右シフトにより所定のビット数 $\sigma$ で右シフトする．
(ビット数が不明な場合)Scheme 9 により MSB を固定位置に合わせた後, シフト量公開右シフトにより適切なビット位置に右シフトする．右シフト量を $[\![ \sigma ]\!]^Q$ とする．
3: $([\![ c ]\!]^P, [\![ \rho_a + \rho_b + \sigma ]\!]^Q)$ を計算して出力する．

---

### 2.5　和・積和について

平文において, 積和は加算の繰り返しであるが, ラウンド数を気にする秘密分散ベースの秘密計算では, 並列に処理できる必要がある．さらに, 和や積和は微小な誤差でもベクトルの要素数分積もるという性質があり, 精度についても単なる積和では不足する場合がある．そこで和・積和については特別に演算を構成していく．

浮動小数点の和では, 指数部を等しくする必要があるため, まずは指数部統一のプロトコルが必要である (Scheme 12).

そして積和は特に, 浮動小数点版 (高速版: Scheme 15, 高精度版: Scheme 18) だけでなく, 固定小数点についても高精度版を提案する (Scheme 18). 積和は, 出力が高ビットになるため入力時点で MSB を一定に抑えたいが, 単純に右シフトしてしまうと, 小さい入力が桁落ちして精度が落ち

る．そのため, ベクトル全体を, 最も絶対値の大きいデータの MSB(ベクトル MSB と呼ぶ) を固定位置に合わせるようなシフトで一斉にシフトする (Scheme 16)．一斉のシフトなのは, やはり指数が等しくなければならないためである．

---

**Scheme 12** 浮動小数点ベクトルの指数部統一

---

**入力:** 浮動小数点数ベクトル $(⟦\vec{a}⟧^P, ⟦\vec{\rho_a}⟧^Q)$

**出力:** $(⟦\vec{b}⟧^P, ⟦\rho_{max}⟧^Q)$, ただし各 $i$ 番目の要素に対して $2^{(\rho_a)_i} a_i \simeq 2^{\rho_{max}} b_i$

---

1: 最大値計算 (例えば Scheme 13) により, $⟦\vec{\rho_a}⟧$ から $⟦\rho_{max}⟧^Q$ を得る．

2: $⟦\vec{\rho}_{dif}⟧^Q := ⟦\vec{\rho}⟧^Q - ⟦\rho_{max}⟧^Q$ を計算する．

3: Scheme 8 により, $⟦\vec{a}⟧$ の各要素を $⟦-\vec{\rho}_{dif}⟧$ の各要素でシフトし, $⟦\vec{b}⟧^P$ とする．
ただし $\vec{\rho}_{dif}$ の各要素が非負なので右シフトとなるため, 左シフトの分岐は省いてよい．

4: $(⟦\vec{b}⟧^P, ⟦\rho_{max}⟧^Q)$ を出力する．

---

**Scheme 13** 最大値回路

---

**入力:** ビット表現した整数の列 $\vec{a}$

**出力:** ビット表現した最大値 $a_{max}$

---

1: $\vec{a}_{\ell-1}$ の全ビット OR をとり, $(a_{max})_{\ell-1}$ とする．
このビットベクトルの最大値であり, 出力の最左ビットでもある．

2: $\vec{e}_{\ell-1} := \vec{a}_{\ell-1} \oplus (a_{max})_{\ell-1}$ を計算する．
このレコードが現在の最大値候補であるかどうかを表す．

3: **for** $i = \ell - 2$ **to** $0$ **do**

4: 　$\vec{a}_i \wedge e_{i+1}$ を計算し, $\vec{b}_i$ とする．

5: 　$\vec{b}_i$ の全ビット OR をとり, $(a_{max})_i$ とする．
　最大値候補となっているレコードの中の最大値であり, 出力の $i$ ビット目でもあり, このビット位置の最大値が 1 の場合に, そのレコードが最大値候補であるかどうかを表す意味もある．

6: 　$\vec{e'}_i := \vec{e}_{i+1} \wedge \neg(a_{max})_i$ を計算する．
このビット位置の最大値が 0 の場合に, そのレコードが最大値候補であるかどうかを表す．

7: 　$\vec{b}_i \oplus \vec{e'}_i$ を計算し, $\vec{e}_i$ とする．
このレコードが現在の最大値候補であるかどうかを表す．

8: $a_{max}$ を出力する．

---

**Scheme 14** 浮動小数点ベクトル和

---

**入力:** 浮動小数点数ベクトル $(⟦\vec{v}⟧^P, ⟦\vec{\rho_v}⟧^Q)$

**パラメータ:** ベクトル長 $m$

**出力:** $(⟦\vec{b}⟧^P, ⟪\vec{\rho_b}⟫^Q)$, ただし $\displaystyle\sum_{0 \le i < m} 2^{(\rho_a)_i} a_i \simeq 2^{\rho_b} b$

---

1: Scheme 12 により, $(⟦\vec{v}⟧^P, ⟦\vec{\rho_v}⟧^Q)$ の指数を揃えたベクトル $(⟦\vec{v'}⟧^P, ⟦\rho_{v'}⟧^Q)$ を得る．

2: $(⟦\displaystyle\sum_{0 \le i < m} v'_i⟧^P, ⟦\rho_{v'}⟧^Q)$ を計算して出力する．

---

**Scheme 15** 浮動小数点ベクトル積和

---

**入力:** 浮動小数点数ベクトル $(⟦\vec{a}⟧^P, ⟦\vec{\rho_a}⟧^Q)$, $(⟦\vec{b}⟧^P, ⟦\vec{\rho_b}⟧^Q)$

**パラメータ:** ベクトル長 $m$

**出力:** $(⟦c⟧^P, ⟪\rho_c⟫^Q)$, ただし $\displaystyle\sum_{0 \le i < m} 2^{(\rho_a)_i (\rho_b)_i} a_i b_i \simeq 2^{\rho_c} c$

---

1: Scheme 12 により, $(⟦\vec{a}⟧^P, ⟦\vec{\rho_a}⟧^Q)$, $(⟦\vec{b}⟧^P, ⟦\vec{\rho_b}⟧^Q)$ の指数を揃えたベクトル $(⟦\vec{a'}⟧^P, ⟦\rho_{a'}⟧^Q)$, $(⟦\vec{b'}⟧^P, ⟦\rho_{b'}⟧^Q)$ を得る．

2: $(⟦\displaystyle\sum_{0 \le i < m} a'_i b'_i⟧^P, ⟦\rho_{a'} + \rho_{b'}⟧^Q)$ を計算して出力する．

---

**Scheme 16** ベクトル MSB 合わせ

---

**入力:** シェアのベクトル $⟦\vec{v}⟧^P$

**パラメータ:** 入力の最大ビット数 $\ell$, ベクトル長 $m$

**出力:** $⟦2^\rho \vec{v}⟧^P, ⟪\rho⟫^p$, ただし $2^\rho \vec{a}$ のベクトル MSB は $\ell - 1$ ビット目

---

1: ビット分解により $⟦\vec{a}⟧^P$ のビット表現 $⟦\vec{v}⟧^{2^\ell}$ を得る．

2: 各ビット位置 $0 \le i < \ell$ のベクトル $⟦\vec{v_i}⟧$ に対して, 全要素の OR をとる．
すなわち $⟦a_i⟧^2 := ⟦(v_i)_0⟧^2$ OR $\cdots$ OR $⟦(v_i)_{m-1}⟧^2$．

3: $⟦a⟧^{2^\ell} := ⟦A_0⟧^2, \cdots ⟦A_{\ell-1}⟧^2$ に対して, Scheme 9 の step. 2〜step.5 を行い, $⟪\rho⟫^p$ を得る．
$a$ の MSB が $\vec{v}$ のベクトル MSB と等しいことに注意．

4: 乗法的ローテーション (Scheme 3) により $⟦2^\rho \vec{v}⟧^P$ を計算して出力する．

---

**Scheme 17** 精度を高めた固定小数点ベクトル積和

---

**入力:** 固定小数点数ベクトル $⟦\vec{a}⟧^P$, $⟦\vec{b}⟧^P$

**パラメータ:** ベクトル長 $m$

**出力:** $⟦c⟧^P$, ただし $\displaystyle\sum_{0 \le i < m} a_i b_i \simeq c$

---

1: Scheme 16 により MSB 位置を調整したベクトルとシフト量, $(⟦\vec{a'}⟧, ⟪\rho'_a⟫^p)$, $(⟦\vec{b'}⟧, ⟪\rho'_b⟫^p)$ を得る．

2: mod $p \to$ mod $Q$ 変換により, $⟦\rho'_a⟧^Q$, $⟦\rho'_b⟧^Q$ を得る．

3: $⟦c⟧^P := ⟦\displaystyle\sum_{0 \le i < m} a'_i b'_i⟧^P$ を計算する．

4: Scheme 8 により, $⟦c⟧^P$ を $⟦-\rho'_a - \rho'_b⟧^Q$ でシフトして出力する．

---

**Scheme 18** 精度を高めた浮動小数点ベクトル積和

**入力:** 浮動小数点数ベクトル $([\![\vec{a}]\!]^P, [\![\vec{\rho}_a]\!]^Q)$, $([\![\vec{b}]\!]^P, [\![\vec{\rho}_b]\!]^Q)$

**パラメータ:** ベクトル長 $m$

**出力:** $([\![c]\!]^P, \langle\!\langle \rho_b \rangle\!\rangle^Q)$, ただし $\sum_{0 \le i < m} 2^{(\rho_a)_i (\rho_b)_i} a_i b_i \simeq 2^{\rho_b} b$

1: Scheme 16 により $[\![\vec{a}]\!]^P$, $[\![\vec{b}]\!]^P$ の MSB 位置を調整したベクトルとシフト量, $([\![\vec{a'}]\!], \langle\!\langle \rho'_a \rangle\!\rangle^p)$, $([\![\vec{b'}]\!], \langle\!\langle \rho'_b \rangle\!\rangle^p)$ を得る.
2: mod $p \to$ mod $Q$ 変換により $[\![\rho_{a'}]\!]^Q$, $[\![\rho_{b'}]\!]^Q$ を得る.
3: Scheme 12 により, $([\![\vec{a'}]\!]^P, [\![\vec{\rho}_a - \rho_{a'}]\!]^Q)$, $([\![\vec{b'}]\!]^P, [\![\vec{\rho}_b - \rho_{b'}]\!]^Q)$ の指数部を統一したベクトルと指数部, $([\![\vec{a''}]\!], [\![\rho_{a''}]\!]^Q)$, $([\![\vec{b''}]\!], [\![\rho_{b''}]\!]^Q$ を得る.
4: $[\![c]\!]^P := [\![\sum_{0 \le i < m} a''_i b''_i]\!]^P$ を計算し, $([\![c]\!]^P, [\![\rho_{a''} + \rho_{b''}]\!]^Q)$ を得る.
Scheme 11 と同様のポリシーで, 必要に応じてさらにシフトを行って出力する.

### 2.6 処理効率

アルゴリズムの処理効率に関して, 要素演算である乗法的ローテーション (Scheme 3), フラグ列 $\to$ 数値変換 (Scheme 4), 非商転移モジュラス変換 (Scheme 5), シフト量秘匿左右シフトプロトコル-その 2-(Scheme 8) および, 比較用に浮動小数点加算・乗算について評価する. 複雑かつ比較不能なものは割愛する. また, 通信量において定数項は省略する.

(1) 乗法的ローテーション: 通信量 $\frac{4}{3}|P|$ ビット, 2 ラウンド
(2) フラグ列 $\to$ 数値変換: 通信量 $\frac{4}{3}|\ell|$ ビット, 2 ラウンド
(3) 非商転移モジュラス変換: 通信量 $|Q| + |p|$ ビット, $|p|$ ラウンド
(4) シフト量秘匿左右シフトプロトコル-その 2-: 通信量 $(\frac{5}{3}d + \frac{10}{3})|P| + (2d+1)|p|$, ラウンド数 $\lambda + 4$
(5) 浮動小数点加算: 通信量 $(\frac{5}{3}d + \frac{19}{3})|P| + 3|Q| + 2\lambda + (4d+1)|p|$, ラウンド数 $2\lambda + 7$
(6) 浮動小数点乗算: 通信量 $\frac{8}{3}|P|$, 3 ラウンド

$d$ は Scheme 8 中の分割数 $d$ である.

比較対象として, 天田らの方式 [2] は, 加算については 2 つ方式があり, 通信量は対数以下のコストを丸めれば, $22|P| + 5|Q| + O(\log|P| + \log|Q|)$ と表せる. ラウンド数については良い方で, 定数の 42 である. 乗算に関しては通信量 $12|P| + O(1)$, ラウンド数は定数の 23 である. 本稿で目指す浮動小数点は, 性能を目指した固定小数点とのハイブリッドで, 毎度の MSB 調整はしない方針のため, そもそも性能に有利である. その上で, そのようなアーキテクチャの提案も含めての比較と考えてほしい. 本稿の加算は, $d$ が典型的には 1 であることを考えると, $6|P| + 3|Q| + 2\lambda + 5|p|$ である. $|P| = 61, |Q| = 13, \lambda = 10, |p| = 6$ であることを考えると, 3 倍程度本稿の方式が効率的である. 加算は複雑なため, 要素であるシフトを高速化してもまだ桁違いにはならないようである. 乗算に関しては 5 倍程度高速である.

## 3. 実機性能評価

本節では実機実験の結果を報告する. 下記のマシン 3 台の, マルチパーティ計算である.

- CPU: Xeon Gold 6144 3.5GHz, 6 cores x 2 sockets
- memory: 768GB
- NW: 10Gbps リングトポロジ
- OS: CentOS 7.3

表 1 は各演算の性能である. 太字が提案手法であり, その他は参考/比較用である. 浮動小数点加算・乗算については Sharemind の数値 [3] を記した. その他は我々の固定小数点演算との比較である.

パラメータとして MSB 位置の上限が重要になるが, これは 28bit(0 スタート表記のため, 29bit 数) とした. 符号付きで mod $P$ で商転移を用いることができる最大の MSB 位置が 57 であり, その半分以内という条件である. 28bit は単精度を超えており, 多くのアプリケーションで十分と考える.

積和演算としては実際には, 行列乗算を選択した. 行列乗算は積和から成り, かつ機械学習などで極めて重要だからである. 具体的には, 左の行列を行数 100 とし, 行数 $\times$ 列数 $=$“件数” となるようにし, 右の行列は左の列数を長さとするベクトルとした. 処理量は, サイズを列数とする積和を, 行数分だけ繰り返す処理量に等しい.

スケールとして, 1000 件, 100 万件, 1000 万件の 3 つと, 遅延を 100ms と極大にすることで実ラウンド数を計測したものを記した. また, passive モデルの他に active モデルの場合の性能も示した (プロトコルは割愛, [8][9] により passive 版から拡張). active モデルのセキュリティパラメータは 8bit であり, 攻撃検知率は約 99%である. 計算量的安全性と異なりオフライン攻撃が不可能なため, この確率は攻撃を抑止するには十分である.

目立つのは, 乗法的ローテーションの性能が最も基本的な演算である, 整数乗算に近いこと, そして浮動小数点加算の性能であろう.

前者に関して, 数値シェアをべきとするべき乗を乗算するというこのプロトコルが乗算に匹敵することから, 乗法的ローテーションプロトコルの画期的さが分かるであろう.

後者に関しては, 固定小数点加算, 浮動小数点加算 (本稿), 同 (Sharemind) で, 前 2 者間で 50 倍, 後 2 者間でさらに 200 倍性能が低下して推移する. 固定小数点加算と Sharemind の加算では実に 10,000 倍の差である. 浮動小数点が何故避けられるか理解できると共に, 本稿の加算が非常に高速なこと, そしてそれでも固定小数点には及ばず, 精度を求める際でも固定・浮動小数点のハイブリッド構成をとることの妥当性が垣間見える.

ラウンド数では乗法的ローテーションで理論値の 2 でなく 3 であるなどの乖離があるが, 実装が原因と考えられる.

表 1　性能検証 (スループットは [M op/s], ラウンド数は無次元)

| | passive | | | | active | | | |
|---|---|---|---|---|---|---|---|---|
| 件数 | 1,000 | 100 万 | 1000 万 | ラウンド数 | 1,000 | 100 万 | 1000 万 | ラウンド数 |
| 整数乗算 (参考) | 1.19 | 20.20 | 21.19 | 1 | 0.93 | 15.27 | 14.83 | 5 |
| **乗法的ローテーション** | 1.18 | 16.74 | 16.83 | 3 | 0.69 | 11.28 | 11.63 | 6 |
| **フラグ列 $\rightarrow$ 数値変換** | 0.94 | 18.77 | 18.38 | 3 | 0.39 | 1.44 | 1.46 | 7 |
| mod 61 $\rightarrow$ mod $2^{13}-1$ 変換 | 0.36 | 28.47 | 28.39 | 2 | 0.07 | 8.13 | 8.07 | 12 |
| **シフト量秘匿左右シフト-その 2-** | 0.21 | 5.94 | 5.36 | 8 | 0.04 | 1.81 | 1.91 | 20 |
| **MSB 合わせ (新)** | 0.45 | 10.87 | 10.45 | 14 | 0.10 | 1.26 | 1.26 | 24 |
| **ベクトル MSB 合わせ** | 0.23 | 19.03 | 17.53 | 14 | 0.07 | 5.56 | 5.41 | 24 |
| 固定小数点加算 | 11.11 | 337.27 | 308.95 | 0 | 16.95 | 430.11 | 308.74 | 0 |
| **浮動小数点加算** | 0.27 | 6.88 | 6.82 | 15 | 0.06 | 2.77 | 2.69 | 21 |
| 浮動小数点加算 [3] | 0.03 | 0.03 | | | | | | |
| 固定小数点乗算 | 1.18 | 14.33 | 14.76 | 3 | 0.22 | 6.59 | 5.69 | 8 |
| **浮動小数点乗算** | 1.03 | 13.72 | 13.59 | 3 | 0.19 | 6.38 | 5.63 | 8 |
| 浮動小数点乗算 [3] | 0.14 | 0.20 | | | | | | |
| 固定小数点ベクトル和 | 17.54 | 933.71 | 980.87 | 0 | 18.87 | 994.04 | 952.83 | 0 |
| **浮動小数点ベクトル和** | 0.09 | 10.73 | 10.83 | 34 | 0.03 | 3.31 | 3.69 | 34 |
| **固定小数点行列乗算 (高精度)** | 0.03 | 9.58 | 10.33 | 102 | 0.002 | 0.93 | 1.59 | 123 |
| **浮動小数点行列乗算 (高精度)** | 0.03 | 7.01 | 7.05 | 123 | 0.003 | 0.63 | 1.35 | 152 |

## 4. おわりに

本稿では 3 パーティの秘密分散ベース秘密計算において, 浮動小数点演算を構成した. キーとなる演算はシフト量秘匿シフトプロトコルで, その構成の中で, 技法として乗法的ローテーションプロトコルと呼ぶ非常に効率的なビットローテーションプロトコルを提案した. 結果として, 浮動小数点において課題となる加算について, 既存実装の 200 倍程度のスループットを得た. また, MSB を毎回調整することをしないアーキテクチャをとるため, 乗算は固定小数点の場合とほぼ等しい性能である.

本稿の結果により, 固定小数点演算の中に精度の要請で必要に応じて浮動小数点を交える, ハイブリッド構成が非常に現実的になったと言える.

### 参考文献


[1] Kamm, L. and Willemson, J.: Secure floating point arithmetic and private satellite collision analysis, *Int. J. Inf. Sec.*, Vol. 14, No. 6, pp. 531–548 (online), DOI: 10.1007/s10207-014-0271-8 (2015).

[2] 西出隆志天田拓磨: 通信量を削減した浮動小数点演算のためのマルチパーティ計算, 情報処理学会論文誌, Vol. Vol.60 No.9, pp. 1433–1447 (2019).

[3] Randmets, J.: Programming Languages for Secure Multi-party Computation Application Development, *PhD thesis. University of Tartu* (2017).

[4] 濱田浩気, 五十嵐大, 千田浩司, 高橋克巳: 3 パーティ秘匿関数計算のランダム置換プロトコル, *CSS2010* (2010).

[5] Kikuchi, R., Ikarashi, D., Matsuda, T., Hamada, K. and Chida, K.: Efficient Bit-Decomposition and Modulus-Conversion Protocols with an Honest Majority, *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings* (Susilo, W. and Yang, G., eds.), Lecture Notes in Computer Science, Vol. 10946, Springer, pp. 64–82 (online), DOI: 10.1007/978-3-319-93638-3_5 (2018).

[6] 三品気吹, 五十嵐大, 濱田浩気, 菊池亮: 高精度かつ高効率な秘密ロジスティック回帰の設計と実装, *CSS2018* (2018).

[7] 五十嵐大: M op/s を超える秘密計算上の初等関数群, *SCIS2020* (2020).

[8] Chida, K., Hamada, K., Ikarashi, D., Kikuchi, R., Kiribuchi, N. and Pinkas, B.: An Efficient Secure Three-Party Sorting Protocol with an Honest Majority, *IACR Cryptology ePrint Archive*, Vol. 2019, p. 695 (online), available from ⟨https://eprint.iacr.org/2019/695⟩ (2019).

[9] Ikarashi, D., Kikuchi, R., Hamada, K. and Chida, K.: Actively Private and Correct MPC Scheme in t<n/2 from Passively Secure Schemes with Small Overhead, *IACR Cryptology ePrint Archive*, Vol. 2014, p. 304 (online), available from ⟨http://eprint.iacr.org/2014/304⟩ (2014).