Computer Security Symposium 2019
21 - 24 October 2019

秘密計算 AIの実装に向けた秘密実数演算群の設計と実装 -$O(|p|)$ビット通信量 $O(1)$ラウンドの実数向け右シフト-

五十嵐 大[1,a)

概要：本稿では機械学習など, 数値計算の性質を持つ演算の実現に向け, 高速な右シフト/除数公開除算, 逆数, 除数秘匿除算, 平方根とその逆数, 指数関数を設計・実装し, 性能と精度を報告する. 右シフト/除数公開除算に関しては $O(|p|)$ ビット通信量, $O(1)$ ラウンドを両立する初の方式である.

キーワード：秘密計算, 機械学習, 実数演算

Secure Real Number Operations for Secure AI -$O(|p|)$-Bit Communication and $O(1)$-Round Right Shift Protocol-

Dai IKARASHI[1,a)]

Abstract: We design and implement fast right shift/public divisor division, reciprocal, private divisor division, square root and its reciprocal and exponential function for realizing numerical computations such as machine learning, and report their performances and precisions. Especially, the right shift/public divisor division is the first algorithm which achieves $O(|p|)$ bits communication and $O(1)$ rounds.

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.

特に三品らは秘密一括写像等の手法を用いて, 機械学習の代表的なアルゴリズムの一つであるロジスティック回帰において, 平文のツールである R に比肩する性能を実現した. 秘密一括写像は, 一定の条件下で任意の関数を計算できるが, 下記の課題がある.

(1) 計算精度に対して指数的に処理コストが増大するため, 現実的な精度は 16 ビット程度が限界であり, 精度設計がシビアである
(2) 並列処理向けのため, ミニバッチ学習のような, 並列数が小さいケースでは性能が低い

本稿ではこれらの課題を解決するため, 下記の特徴を目指した実数演算群を設計し, 秘密計算ライブラリ MEVAL に実装する.

(1) 単精度実数精度 (23 ビット) 以上の固定小数点演算
(2) 秘密一括写像より高速

筆者はこれまでの経験で, 浮動小数点でなければ実装できない演算に出会っていないため, 速度に優れる固定小数点で設計・実装する. 浮動小数点は汎用的で利用側は最も容易だが, 演算ごとにビット位置の調整が入るため, 秘密分散ベース秘密計算のアドバンテージであったはずの加減算が急激に重くなり, 効率的ではない. 秘密計算はまだ浮動小数点が適切である性能の段階には来ていないと考える.

後者に関して, 普通はソートを含む秘密一括写像は遅いのだが, MEVAL はソートの高速化に大変力を入れている [1] ため, これは意外と簡単ではない.

演算として具体的には, 下記を設計・実装する.

(1) 高速な右シフト/公開除数除算
(2) 逆数, 除数秘匿除算
(3) 平方根とその逆数

[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

(4) 指数関数

右シフト/公開除数除算に関しては, $O(|p|)$ ビット通信量, $O(1)$ ラウンドを両立する初のアルゴリズムであり, 微小な誤差が許容される数値計算の性質を利用しつつ, 誤差が理論的に最小であると言える興味深い手法である.

1.1 関連研究

実数演算のアルゴリズムおよび実装としては, アルゴリズムでは Bogdanov ら [2](2016 年版), 実装では Jaak ら [3] が最良である. 右シフトに関しては, 適用する代数に差はあるが, [2] が通信量 $O(|p|\log|p|)$, ラウンド数 $O(\log|p|)$ であり, 三品ら [4] が通信量 $O(|p|)$, ラウンド数 $O(|p|)$ であり, 通信量とラウンド数のトレードオフの関係である. これは加算回路の選択に依存している.

本稿の右シフトは通信量 $O(|p|)$, ラウンド数 $O(1)$ であり, このトレードオフを解決する.

1.2 記法

- $[\![x]\!]$: mod p 要素 x を線形秘密分散したシェア.
- $\{x\}$: mod 2 要素 x を加法的または複製秘密分散したシェア.
- $\langle\!\langle x \rangle\!\rangle$: mod p 要素 x を加法的または複製秘密分散したシェア.
- p: 素数. 主にメルセンヌ素数を想定する.
- $|p|$: p のビット数.
- k: 秘密分散の閾値.
- m: 秘密分散のシェアの数. 複製秘密分散ではサブシェアの数.
- $[\![]\!]^a, \{\}^a, \langle\!\langle \rangle\!\rangle^a$: 小数点位置が a のシェア.
- $[\![]\!]_i, \{\}_i, \langle\!\langle \rangle\!\rangle^i$: i 番目のシェアまたはサブシェア.
- $+_a, -_a, \times_a$: 出力の小数点位置が a の演算. 加減乗算に右シフト, 左シフト (2 べき数との乗算) を組み合わせる.
- a/d: 小数点以下切り捨ての整数除算.
- $\frac{a}{d}$: 実数除算.

加法的秘密分散と複製秘密分散を統一的に使っているが, この 2 つは復元が加法で成るという性質が一致しており, 本稿で論じる下位演算で統一的に扱えるからである. 具体的には, passive 安全でよい場合は (k,k)-加法的秘密分散が効率的 (シェア数 m は k) であり, active 安全にしたい場合は (k,n)-複製秘密分散 (m は k に対して指数的, 少パーティ数向け) を用いる.

1.3 前提

mod p シェアとして格納する値は, $\sqrt{p}$ 未満程度を想定する. なぜなら, そうでなければ乗算でオーバーフローするからである. この前提は, 本稿では以下の観点で重要である.

(1) 商転移 [5][6] を用いるとビット分解の際, 空いている上位ビットを処理しなくてよいので半分のビットが空いていれば演算量が半分になる

(2) 2.1 節での提案手法で, 空いているビット数が演算精度に直結している

2. 提案手法

2.1 高効率な数値計算向け右シフトと公開除数除算

右シフト演算は数値計算においては最も重要な演算の一つである. なぜなら, 乗算を行うごとにデータのビット数は倍々になり, 所定のデータ長に収まらなくなるからである. 乗算のたび, 右シフトを行って一定のデータ長を保つ必要がある. 固定小数点数を利用する上では, シフト量は公開でよい. 筆者は [4] の中で, メルセンヌ素体上の効率的な右シフトを提案した. 本稿では数値計算向けの, さらに効率的な右シフトを提案する. なお数値計算向けと言っているのは, 提案手法は微少な確率的誤差を含むという意味合いである. 完全一致を求める計算には向かないが, 数値計算は元来誤差を含む計算であるため, 誤差が所要精度未満になれば問題がない. 提案手法では, 以下を評価軸として設計する.

(1) 誤差の動く範囲が微小となること

(2) 出力の期待値が真の除算結果に近いこと (右シフトは 2 べきによる除算と見なす)

右シフトは 2 べきによる除算と等しいので, 公開値による除算を考える. 入力を $\langle\!\langle a \rangle\!\rangle$, 公開の除数を d とし, $a = \sum_{j<m} \langle\!\langle a \rangle\!\rangle_j \mod p$ とする. すると $\sum_{j<m} \langle\!\langle a \rangle\!\rangle_j = qp + a$ と表せる. このときに, 各シェアを単純に d で割ることを考える. そこで, $\langle\!\langle a \rangle\!\rangle_j$, a, p をそれぞれ, $\langle\!\langle a \rangle\!\rangle_j = \alpha_j d + r_j$, $a = \alpha d + r$, $p = p'd + r'$ と商と余りで表現する. なお定義より求めたい答えは α となる. すると, 各シェア $\langle\!\langle a \rangle\!\rangle_j$ を単純に d で割ると α_j なので, これらをシェアとみた時にどのような平文に対応するか観察する. 復元はシェアの加算なので対応する平文は, 下記となる.

$$\sum_{j<m}\alpha_j = \sum_{j<m}\frac{\langle\!\langle a \rangle\!\rangle_j - r_j}{d} = \frac{\sum_{j<m}\langle\!\langle a_j \rangle\!\rangle}{d} - \frac{\sum_{j<m} r_j}{d}$$

$$= \frac{qp+a}{d} - \frac{\sum_{j<m} r_j}{d} = \frac{q(p'd+r') + \alpha d + r}{d} - \frac{\sum_{j<m} r_j}{d}$$

$$= \alpha + qp' + \frac{r + qr' - \sum_{j<m} r_j}{d}$$

求めたい値である α もちゃんと含まれているが, qp' や $\frac{r + qr' - \sum_{j<m} r_j}{d}$ が邪魔である. ここで商転移 [5][6] を用いると, a が 2^u(ただし u はシェア数 m に対して $m \le 2^u$ を満たす数) の倍数ならば $\langle\!\langle q \rangle\!\rangle$ は非常に効率的に求めることができる. これは一見限定された場合に見えるが, 実際には先に $a' = 2^u a$, $d' = 2^u d$ と被除数も除数も 2^u 倍してお

けば常に適用できる. よってここから, a, d ともに 2^u の倍数とおく. すると $\langle\!\langle q \rangle\!\rangle$ は効率的に求まり, p' は p の d による商であり公開値であるから qp' は効率的にキャンセルできそうである.

2.1.1 誤差の最小化

$$\frac{r + qr' - \sum_{j<m} r_j}{d}$$

残るは である. これを e とおく. 各 r, r', r_j は $d-1$ 以下なので, この値はたかだか $-m$〜m 程度の基本的に小さい値で, 誤差と言うのがちょうどよい大きさである. 筆者が当面ターゲットとしたいのは $m = 2$ か 3, つまり $(2,2)$-加法的秘密分散や $(2,3)$-複製秘密分散のケースであるからなおさらである.

まず, Scheme 1, Scheme 2 に $m = 2$ の場合の除数公開除算アルゴリズムを記す. Scheme 1 がコアなアルゴリズムであり, $(2,2)$-加法的秘密分散かつ a, d が偶数限定である. step 2 で, $\bmod p$ でない加減算があることに注意. これ

$$\frac{r + qr' - \sum_{j<m} r_j}{d}$$

は, の分子を調整する作用であり, $\langle\!\langle a \rangle\!\rangle_0$ が大きかった場合に $\bmod p$ により小さい値になっては困るのである. Scheme 1 は, Scheme 2 に示したように, $(2,3)$-線形秘密分散かつ一般の a, d に容易に拡張できる. また, Scheme 3 に示したように, 公開の大きな d の倍数を加えて正の数に線形変換することで, $\bmod p$ の値を符号有り数と見なしている場合にも適用可能である.

Scheme 1 除数公開除算 ($(2,2)$-加法的秘密分散入出力/かつ平文が偶数限定)

入力: $\langle\!\langle a \rangle\!\rangle, d$, ただし $\langle\!\langle a \rangle\!\rangle, d$ とも偶数
出力: $\langle\!\langle a/d \rangle\!\rangle$, ただし / は小数点以下切り捨て除算

1: 商転移 [5][6] などにより $\langle\!\langle a \rangle\!\rangle$ の商 $\langle\!\langle q \rangle\!\rangle$ を求める.
2: 法 p の d による商と剰余 p', r' を求める
3: $\langle\!\langle b \rangle\!\rangle_j := \begin{cases} (\langle\!\langle a \rangle\!\rangle_j + d - 1 - r')/d \text{ if } j = 0 \\ \langle\!\langle a \rangle\!\rangle_j / d \text{ otherwise} \end{cases}$
この加減算は $\bmod p$ ではなく $\mathbb{N}$ 上加減算であることに注意!
4: $\langle\!\langle a \rangle\!\rangle - (p'+1)\langle\!\langle a \rangle\!\rangle + 1$ を出力する.

Scheme 2 除数公開除算 ($(2,3)$-線形秘密分散入出力/平文は一般の符号無し整数)

入力: $[\![a]\!], d$
出力: $[\![a/d]\!]$, ただし / は小数点以下切り捨て除算

1: $[\![a]\!]$ を $(2,2)$-加法的秘密分散に変換し, $\langle\!\langle a \rangle\!\rangle$ を得る.
2: $\langle\!\langle a' \rangle\!\rangle := \langle\!\langle 2a \rangle\!\rangle$, $d' := 2d$ を計算する.
3: Scheme 1 により $\langle\!\langle a'/d' \rangle\!\rangle (= \langle\!\langle a/d \rangle\!\rangle)$ を得る.
4: $\langle\!\langle a/d \rangle\!\rangle$ を $[\![a/d]\!]$ に変換する.

Scheme 3 除数公開除算 ($(2,3)$-線形秘密分散入出力/平文は一般の符号有り整数)

入力: $[\![a]\!], d$
出力: $[\![a/d]\!]$, ただし / は小数点以下切り捨て除算

1: $2^{(|p|-1)}$ を d で割って切り上げた数 ω を得る.
2: Scheme 2 により $[\![(\omega d + a)/d]\!] (= [\![\omega + a/d]\!])$ を得る.
3: ω を引いて $[\![a/d]\!]$ を出力する.

さて, Scheme 1 の誤差を考える.

定理 2.1 a, d を $O(\sqrt{p})$ なる自然数とする. このとき, Scheme 1 は以下の 3 つの性質を満たす. これは確率的出力をもつ数値計算向けの固定小数点数向け整数除算として理想的である.

(1) [出力の期待値の小数部分 $\sim \frac{a}{d}$ の小数部分] が成り立ち, その近似誤差は $O(\sqrt{p^{-1}})$ である.
(2) 出力は $1 - O(\sqrt{p^{-1}})$ の確率で a/d もしくは $a/d + 1$ であり, 整数出力で期待値を $\frac{a}{d}$ に一致させるための最小誤差幅である. 最悪値に関しては, $O(\sqrt{p^{-1}})$ の確率で $a/d + 2$ となる.
(3) 特に a が d で割り切れるとき, 誤差は $1 - O(\sqrt{p^{-1}})$ の確率で 0 である.

a, d が $O(\sqrt{p})$ であることは, 乗算を繰り返す数値計算では妥当である. 近似誤差として $\sqrt{p^{-1}}$ が全ての性質で出てくるが, 例えば単精度実数 (精度 23 ビット) を実現するのに適する $p = 2^{61} - 1$ では $2^{-30.5}$, 倍精度実数 (精度 53 ビット) を実現するのに適する $p = 2^{127} - 1$ では $2^{-63.5}$ と, 所要精度を上回る精度となるため問題とはならない.

証明 はじめに, 全てに関係する事項を整理する. まず, $m = 2$ のとき, $e = \frac{r + qr' - \sum_{j<m} r_j}{d} = \frac{r + qr' - r_0 - r_1}{d}$ である. $\alpha_0 + \alpha_1$ が整数なので e は整数であり, 分子 ed は d の倍数である.

次に, q の値について考える. $m = 2$ ではシェアは 2 個なので, 復元時の加算による p 以上への繰り上がりの回数である q は 0 か 1 である. そして加法的秘密分散の分散を考えれば, $\{a\}_0$ を一様乱数とし $\{a\}_1 := a - \{a\}_0 \bmod p$ であり, $q = 1 \Leftrightarrow a < \{a\}_0$ となる. $\{a\}_0$ は一様であるから, $q = 0$ の確率は $O(\frac{a}{p})$, $q = 1$ の確率は $1 - O(\frac{a}{p})$ である. すなわち, a が $\sqrt{p}$ 程度未満なので, q はほとんど 1 である.

そして, Scheme 1 の step 3 における加算は, $\alpha_0 + \alpha_1$ を $\alpha + qp' + \frac{r + qr' - r_0 - r_1}{d}$ ではなく, $\alpha + qp' + \frac{r + (q-1)r' + d - 1 - r_0 - r_1}{d}$ に変化させる作用を持っている.

では, (2) から示す. step 4 で, qp' をキャンセルする以外に, $-q + 1$ を加算している. そのため, 確率 $O(\sqrt{p}^{-1})$ で発生する $q = 0$ のときは $e = \{-1, 0, 1\}$, $q = 1$ のときは $e = \{0, 1\}$ となることを示せばよい.

(i)$q = 0$ のとき, $ed = r + d - 1 - r' - r_0 - r_1$ である.

$1 \le r' \le d-1$(p が素数なので d で割り切れることはない), $0 \le r, r_0, r_1 \le d-1$ より, $-2d+2 \le ed \le 2d-3$ である. ed は d の倍数だから, $ed \in \{-d, 0, d\}$ である. つまり $e \in \{-1, 0, 1\}$ である.
(ii)$q=1$ のとき, $ed = r+d-1-r_0-r_1$ である. $q=0$ のときと同様に考えると, $-d+1 \le ed \le 2d-2$ である. つまり $e \in \{0, 1\}$ である.
(i), (ii) より, (2) が示された.
次に (3) を示す. 仮定より $r=0$ とする. $1-O(\sqrt{p^{-1}})$ なので, $q=1$ について示せばよい. $r=0$ かつ $q=1$ のとき, $ed = d-1-r_0-r_1$ であり, $-d+1 \le ed \le d-1$ である. つまり, $e=0$ であり, (3) が示された.
最後に (1) を示す. $q=0$ のときについては, 確率 $O(\sqrt{p^{-1}})$ で e が定数で抑えられるため, (1) の命題に影響が無い. $q=1$ の場合について, $e=1$ となる確率が r/d であることを示せばよい.
(i)$r=0$ のとき, (3) で示した.
(ii)$r>0$ のとき, $ed = r+d-1-r_0-r_1$ である. $r>0$ なので, $1 \le r \le r+d-1-r_0 \le r+d-1 \ge d$ である. ed が d の倍数より $r_1 = r+d-1-r_0 \mod d$ だから, $r+d-1-r_0$ が d 未満のとき $r_1 = r+d-1-r_0$, d 以上のときは $r_1 = r-1-r_0$ となる. すなわち $e=1 \Leftrightarrow r+d-1-r_0 \ge d \Leftrightarrow r_0 \le r-1$ である. そして, d が $O(\sqrt{p})$ なことから, $\mathbb{Z}/p\mathbb{Z}$ 上一様乱数である $\{a\}_0$ の p による剰余である r_0 は, $O(\frac{d}{p})$ の近似精度で一様乱数である. よって $r_0 \le r-1$ が成り立つ確率は $\frac{r}{d}$ で近似され, $q=1$ のときの小数部分の期待値は $(1-O(\sqrt{p^{-1}}))\frac{r}{d}$ となる. $q=0$ のときも考慮しても, 期待値は $(1-O(\sqrt{p^{-1}}))(1-O(\sqrt{p^{-1}}))\frac{r}{d} = (1-O(\sqrt{p^{-1}}))\frac{r}{d}$ とオーダーは変わらない.
□ 定理

$m \ge 3$ の場合については, 誤差の理論検討, および実機実験を重ねてみたものの, $m=2$ のように理想的な誤差にはならないようである. Scheme 4 に, $m=3$ で幾つかの入力で最も誤差の小さかったパラメータを入れた手法を記載しておく. 調整ポイントは下記である.

(1) シェアを d で割るとき, $d/2-1$ 捨 $d/2$ 入するのか切り捨てるのか
(2) p' に何を加算するのか
(3) 最後に加算する定数が幾つなのか

Scheme 4 除数公開除算 (シェア数 3 以上の複製秘密分散/加法的秘密分散)

入力: $\langle\!\langle a \rangle\!\rangle, d$, ただし $\langle\!\langle a \rangle\!\rangle, d$ とも 2^u の倍数, 2^u はシェア数 m 以上
出力: $\langle\!\langle a/d \rangle\!\rangle$, ただし / は小数点以下切り捨て除算

1: 商転移 [5][6] などにより $\langle\!\langle a \rangle\!\rangle$ の商 $\langle\!\langle q \rangle\!\rangle$ を求める.
2: 法 p の d による商と剰余 p', r' を求める
3: z を, $r' \ge d/2$ ならば 1, そうでなければ 0 とする
4: $\langle\!\langle b \rangle\!\rangle_j := \begin{cases} \langle\!\langle a \rangle\!\rangle_j + (d-r') + (d-r')/2 \text{ if } j=0 \\ \langle\!\langle a \rangle\!\rangle_j \text{ otherwise} \end{cases}$
この加減算は $\mod p$ ではなく $\mathbb{N}$ 上加減算であることに注意!
5: $\langle\!\langle b' \rangle\!\rangle_j$ を $\langle\!\langle b \rangle\!\rangle_j$ を d で割って $d/2-1$ 捨 $d/2$ 入した数とする
6: $\langle\!\langle b' \rangle\!\rangle - (p'+z)\langle\!\langle q \rangle\!\rangle - 1$ を出力する.

2.1.2 処理効率

Scheme 2, Scheme 3 の 3 パーティ秘密計算における処理効率は下記となる. Scheme 4 の効率は割愛する. 商 q の導出は [6] の商転移, 加法的秘密分散から線形秘密分散への変換は [7] の手法を前提とした.
(1) 通信量: $5/3(|p|+1)$ ビット
(2) ラウンド数: 2 ラウンド
通信量 $O(|p|)$, ラウンド数 $O(1)$ の両立は右シフト/除数公開除算として初めてである. また, 見ての通り定数係数も 2 以下とごく小さく, 機械学習などの, 演算を反復するために右シフトを大量に処理するアプリケーションでは非常に有効である!

2.2 逆数と除数秘匿除算

除算は加減乗算の処理が得意な秘密計算にとって, 四則演算の一つにも関わらず容易でないことでよく知られる演算である. 除算は除数の逆数と被除数の乗算で表現できるため, 逆数の計算が本質である. そのため, 除数公開の除算 (すなわち, 逆数を平文で計算できる) と除数秘匿の除算は構成も処理効率も本質的に異なる. (もちろん, 除数公開除算の一種である右シフトの利用頻度が圧倒的なため, 除数公開除算を極限まで効率化することは除数秘匿除算とは異なる意義を持つ)

高速な (一般的にビット長の) 除算や逆数の計算としては, Newton-Raphson 法, Goldschmidt 法が一般的である. しかしこの 2 つの方法はどちらも, 逆数関数の Taylor 展開を効率的に計算しているものであり, 本質的に等しい. 本稿では, Taylor 展開を素直に計算する方法をとる. 入力 $x \in [\frac{1}{2}, 1)$ に対して, $1-x$ の逆数 $\frac{1}{1-x}$ の Taylor 展開は下記で与えられる.

$$\frac{1}{1-x} = \sum_{0 \le i \to \infty} x^i = 1 + x + x^2 + \cdots \qquad (1)$$

n 次まで計算した時の剰余項 (すなわち誤差) は下記で表

せる.

$$x^{n+1}+\cdots = x^{n+1}(1+x+x^2+\cdots) = x^{n+1}\frac{1-x^\infty}{1-x}$$
$$= \frac{x^{n+1}}{1-x} \le \frac{1}{2^{n+1}}$$

$n$bit 精度のためには n 回程度乗算しなければいけなくなるように見えるが, 式 (1) は以下のように少ない乗算回数で計算できる形に変形される.

$$\prod_{0\le j\to\infty}(1+x^{2^j}) = (1+x)(1+x^2)(1+x^4)\cdots$$

これだと $2\log n$ 回で n ビット程度の精度となる.

入力が $[\frac{1}{2},1)$ という制限があったが, e を $\lfloor\log_2 d\rfloor - 1$(言い換えると左シフトで d を変換して $[\frac{1}{2},1)$ に収めるためのシフト量. 負なら右シフト) とすると, $2^e d \in [\frac{1}{2},1)$ となり下記の関係式により回避できる.

$$\frac{1}{d} = 2^e\frac{1}{2^e d} \tag{2}$$

ただしこれには msb 合わせの処理 (Scheme 5) を要する. 本稿では簡単のため, この msb 合わせは左シフトで済むものとする. e が負なら右シフトも必要そうだが, 実際には式 (2) の内部計算では値が 2 以下であることが分かっているため, msb 合わせの出力は小数点以下の精度を高めた, 小数点位置が可能な限り高ビット位置にある固定小数点表現と見なすのが適切である. そのため, 実際右シフトが必要となることは稀である. ビット結合に関しては, 大原らのビット結合 [8] を素体に適用可能とした Scheme 6 を提案する.

Scheme 5 msb 合わせ

入力: $[\![a]\!]$

出力: $[\![b]\!], [\![c]\!]$, ただし b は a の msb が $\ell-1$ ビット目に移動するよう $\log_2 c$ ビット左シフトした値

1: ビット分解により $[\![a]\!]$ のビット表現 $\{a_0\},\cdots,\{a_{\ell-1}\}$ を得る.
2: $0\le i<\ell-1$ で, $\{f_i\} := f_{i+1}\vee a_i$ とする
3: $\{f_{\ell-1}\} := \{a_{\ell-1}\}$
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
4: $0\le i<\ell-1$ で, $\{x_i\} := f_i \oplus f_{i+1}$ とする.
5: $\{x_{\ell-1}\} := \{a_{\ell-1}\}$
ここまでで, $0,0,0,1,0,\cdots,0$ のように, msb 位置のみ 1 となっている
6: ビット結合により $\{x_{\ell-1}\},\cdots,\{x_0\}$ を結合し $[\![c]\!]$ を得る.
逆順に結合することに注意. $c = \sum_{i<\ell} 2^i x_{\ell-1-i}$ であり, $2^{\ell-1-\lfloor\log_2 a\rfloor}$ と等しくなっている.
7: $[\![a]\!][\![c]\!]$ を出力する.

Scheme 6 ビット結合

入力: $\{a_0\},\cdots,\{a_\ell\}$, $\ell\le|p|$

出力: $[\![\sum_{i<\ell} 2^i a_i]\!]$

1: $\{b_0\} := \{0\}$
2: $\{a'_0\} := \{a_0\}$
3: $0\le i<\ell$ で, i ビット目の差 $\{a'_i\}$, 桁借り $\{b_i\}$, 商 $\{q'_i\}$ を以下のように帰納的に計算する.
4: $\{a'_i\}, \{b_i\}$ をそれぞれ, $a_i - q'_{i-1} - b_{i-1}$ を減算回路で計算した差と桁借りとする
5: $\{a'_i\}$ の商を計算して $\{q'_i\}$ とする
6: $\{b_{\ell-1}\}, \{q'_{\ell-1}\}$ を mod p 変換し, $[\![b_{\ell-1}]\!], [\![q'_{\ell-1}]\!]$ を得る.
7: $\langle\!\langle a'\rangle\!\rangle$ を, $\langle\!\langle a'\rangle\!\rangle_j := \sum_{i\le\ell} 2^i\{a'_i\}_j$ なるシェアとする.
8: $\langle\!\langle a'\rangle\!\rangle$ を線形秘密分散に変換して $[\![a']\!]$ を得る.
9: $[\![a']\!] - 2^\ell([\![b_{\ell-1}]\!] + [\![q'_{\ell-1}]\!])$ を出力する.

逆数アルゴリズムを Scheme 7 に記す. msb を合わせて Taylor 展開を計算し, 最後に msb 合わせでシフトした分をキャンセルしている. 除数秘匿除算はもはや乗算するのみである (Scheme 8).

Scheme 7 逆数プロトコル

入力: $[\![a]\!]^{e_i}$

出力: $[\![\frac{1}{a}]\!]^{e_o}$

パラメータ: 内部計算用の小数点位置 e_z, 反復回数 I(目安として $\lceil\log e_z\rceil$)

1: msb 合わせにより a を $[\frac{1}{2})$ にシフトした値 $[\![b]\!]^{e_z}$, シフトのために乗じた値 $[\![c]\!]^0$ を得る.
2: $[\![x_0]\!]^{e_z} := 1 -_{e_z} [\![b]\!]^{e_z}$
3: $[\![y_0]\!]^{e_z} := 2 -_{e_z} [\![b]\!]^{e_z}$
4: $1\le i\le I$ で以下を繰り返す
5: $[\![x_i]\!]^{e_z} := [\![x_{i-1}]\!]^{e_z} \times_{e_z} [\![x_{i-1}]\!]^{e_z}$
6: $[\![y_i]\!]^{e_z} := [\![y_{i-1}]\!]^{e_z} \times_{e_z} [\![x_i]\!]^{e_z}$
7: $[\![y_{I-1}]\!]^{e_z} \times_{e_o} [\![c]\!]^0$ を出力する

Scheme 8 除数秘匿除算プロトコル

入力: $[\![a]\!]^{e_i}, [\![d]\!]^{e'_i}$

出力: $[\![\frac{a}{d}]\!]^{e_o}$

1: 逆数プロトコルにより $[\![\frac{1}{d}]\!]^{e_{-i'}}$ を計算する
2: $[\![\frac{1}{d}]\!]^{e_{-i'}}]\!] \times_{e_o} [\![a]\!]^{e_i}$ を出力する

2.3 平方根・平方根の逆数

深層学習で用いられる学習最適化アルゴリズムである Adam では, 平方根の逆数が必要である [9]. また, 平方根も平方根の逆数を経由して $\sqrt{x} = x\frac{1}{\sqrt{x}}$ として計算した方が効率的に計算できることが知られており, 平方根の逆数が計算できれば一挙両得である.

平方根の逆数は Taylor 展開だと収束が遅いので, 入力を

a とするとき, 以下の Newton 法で計算する.

$$x_i = x_{i-1} * (3 - a x_{i-1}^2)/2, \text{ ただし } x_0 = 1$$

逆数と同様, $a \in [\frac{1}{2}, 1)$ を要求するので, msb 合わせが必要である. しかし複雑なのは, 式 (2) と異なり, 下記のように最後に掛ける値が 2^e の平方根である点である.

$$\frac{1}{\sqrt{a}} = \sqrt{2^e}\frac{1}{\sqrt{2^e a}}$$

平方根の逆数の秘密計算においては, Newton 法部分は平文の定石と同じだが, この $\sqrt{2^e}$ の計算を効率的に行うことが必要である. Scheme 9 では入力のビット列の中でビットが立っているのが msb だけなことを利用して, ほとんどの計算をオフライン処理である mod 2 の XOR で済ますことができている. ここで一気に $\sqrt{c}(=\sqrt{2^e})$ を計算してしまわないのは, $\sqrt{2}$ は無理数のため $|p|/2$ ビットに近い高精度での表現が必要で, 2 べきである $[\![c']\!]$ と安易に乗じておくと $|p|/2$ ビットを超えてしまうからである.

Scheme 9 平方根の逆数用: msb 合わせの追加処理 (シフト用乗数 c の平方根を求める)

入力: $\{x_0\}, \cdots \{x_\ell - 1\}$

出力: $[\![r]\!], [\![c']\!]$, ただし r は Newton 法後に $\sqrt{2}$ を掛ける必要があるかどうかを表す真理値, c' は Newton 法後に掛けるべき 2 べきの値

1: $\ell' := \lceil \frac{\ell}{2} \rceil$ とする.
2: 各 $i < \ell$ で, $\{x'_i\} := \{x_{\ell-1-i}\}$ とする
3: 各 $i < \ell'$ で, $\{y_i\} := \{x'_{2i}\} \oplus \{x'_{2i+1}\}$ とする. ただし ℓ が奇数のとき, $\{y_{\ell'-1}\} := \{x'_{2i}\}$ とする.
これは, $2^{\lfloor \log_2 \sqrt{c} \rfloor}$ のビット表現である.
4: $\{r\} := \{x'_1\} \oplus \{x'_3\} \oplus \{x'_5\} \oplus \cdots$ とする.
これは, $\sqrt{2}$ を掛ける必要があるかどうかを表す.
5: mod p 変換で $\{r\}$ を $[\![r]\!]$ とする.
6: ビット結合で $\{y_0\}, \cdots, \{y_{\ell'-1}\}$ を $[\![c']\!]$ として出力する.

Scheme 9 を使った平方根の逆数プロトコルを Scheme 10 に示す. msb 合わせを行い, Newton 法, そして msb 合わせでシフトした分を補正している. 平方根の逆数ができれば, 平方根は後は乗算だけである (Scheme 11).

Scheme 10 平方根の逆数プロトコル

入力: $[\![a]\!]^{e_i}$

出力: $[\![\frac{1}{\sqrt{a}}]\!]^{e_o}$

パラメータ: 内部計算用の小数点位置 e_z, 反復回数 I(目安として $\lceil \log e_z \rceil$)

1: Scheme 5, Scheme 9 より a を $[\frac{1}{2})$ にシフトした値 $[\![b]\!]^{e_z}$, 合わせてシフトのために乗じた値の平方根を表す値 $[\![c']\!], [\![r]\!]$ を得る.
2: $[\![x_0]\!]^{e_z} := 3 -_{e_z} [\![b]\!]^{e_z}$
3: $[\![y_0]\!]^{e_z} := [\![x_0]\!]^{e_z} /2$
4: $1 \le i \le I$ で以下を繰り返す
5: $\quad [\![x_i]\!]^{e_z} := 3 -_{e_z} [\![y_{i-1}]\!]^{e_z} \times_{e_z} [\![y_{i-1}]\!]^{e_z} \times_{e_z} [\![b]\!]^{e_z}$
6: $\quad [\![y_i]\!]^{e_z} := [\![x_{i-1}]\!]^{e_z} \times_{e_z+1} [\![y_{i-1}]\!]^{e_z}$
7: $[\![y_{I-1}]\!]^{e_z} \times_{e_z} ([\![r]\!]^0 \times_{e_z} \sqrt{2}) \times_{e_o} [\![c']\!]^0$ を出力する

Scheme 11 平方根プロトコル

入力: $[\![a]\!]^{e_i}$

出力: $[\![\sqrt{a}]\!]^{e_o}$

1: Scheme 10 により, $[\![\frac{1}{\sqrt{a}}]\!]^{e_{-i/2}}$ を計算する.
2: $[\![a]\!]^{e_i} \times_{e_o} [\![\frac{1}{\sqrt{a}}]\!]^{e_{-i/2}}$ を出力する.

2.4 指数関数

指数関数はロジスティック回帰や深層学習などで, シグモイド関数やソフトマックス関数といった関数の中で用いられる. 指数関数は Taylor 展開の収束が速いため, Taylor 展開の計算が適している.

$$\exp x = \sum_{0 \le i \to \infty} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} \cdots$$

ただし, やはり入力そのままでは適用できない. x が大きいと上式の収束が遅いのは明らかである. よって x を下記のように加法的に分割して計算する.

(1) 想定される入力の最小値 μ
(2) $x - \mu$ の, 小数点以下 t ビット以上の上位 u ビット $x_0, \cdots, x_{u-1}$
(3) $x - \mu$ の, x_0 よりも下位ビット全体が表す数 ρ

$$\exp x = \exp \mu \exp 2^{-t} x_0 \cdots \exp 2^{u-t-1} x_{u-1} \exp x_\sigma$$

μ は想定される最小値だから公開値, $\exp 2^i x_j$ の形の数は, x_j がビットなので, x_j が 0 なら 1, 1 なら $\exp 2^i$ という if-then-else ゲート (具体的には $x_j(\exp 2^i - 1) + 1$) で計算できる. (平文上のアルゴリズムではもっと大きな表を使うが, 秘密計算は大きい表の参照が苦手であるから 2 値の表を使う.) ここで, 指数関数は急速に増大するため, 固定小数点数で扱うべき入力範囲は非常に小さく, 大きい u を想定する必要はなくこの部分の処理コストは大きくないことに注意する. 例えば入力が 20 程度でも, 出力は 30 ビット程度の大きな数となる. また, $\exp x_\sigma$ は, 上位ビットが抜け

落ちていて 2^{-t} 未満の小さい数であることが保証されているので, Taylor 展開で計算する. ビット数に関する収束速度は t に比例に近い関係で速くなる.

指数関数プロトコルを Scheme 12 に記す. 入力の差異による値の変化が大きすぎるため, 浮動小数点のように指数部と仮数部に分けて計算することに注意.

Scheme 12 指数関数プロトコル

入力: $[\![a]\!]^{e_i}$

出力: $[\![\frac{1}{\sqrt{a}}]\!]^{e_o}$

パラメータ: 内部計算用の小数点位置 e_z, Taylor 展開の所要次数 I

1: $[\![a']\!]^{e_i} := [\![a]\!]^{e_i} - \mu$ を計算する.
2: 小数点以下 t ビットより上位のビットをビット分解で取り出し mod p 変換し, $[\![a'_0]\!]^0, \cdots, [\![a'_{u-1}]\!]^0$ を得る.
3: 各 i で, if-then-else ゲートにより $[\![f'_i]\!]^{e_z} :=$ if $[\![a'_i]\!]$ then f_i else 1 を計算する. なお f_i は $\exp 2^{i-t}$ の仮数部である.
4: 各 i で, if-then-else ゲートにより $[\![\varepsilon'_i]\!]^0 :=$ if $[\![a'_i]\!]$ then 2^{ε_i} else 1 を計算する. なお ε_i は $\exp 2^{i-t}$ の指数部である.
5: 各 i の $[\![f'_i]\!]^{e_z}$ の積 $[\![f']\!]^{e_z}$ を計算する. 上位ビット部分の仮数部である.
6: 各 i の $[\![\varepsilon'_i]\!]^0$ の積 $[\![\varepsilon']\!]^0$ を計算する. 上位ビット部分の指数部の 2 べきである.
7: $[\![a_\sigma]\!]^{e_i} := [\![a']\!]^{e_i} - e_i \sum_{i<u} 2^{i-t} [\![a'_i]\!]^0$ を計算する.
8: 実数乗算と除数公開除算により, $[\![a'_\sigma]\!]^{e_i} := \sum_{0 \le i < I} \frac{[\![a^i_\sigma]\!]^{e_i}}{i!}$ を計算する.
9: $[\![f'_i]\!]^{e_z} \times_{e_z} [\![a'_\sigma]\!]^{e_i} \times_{e_z} [\![\varepsilon']\!]^0 \times_{e_o} \exp \mu$ を出力する.

2.5 実装について

MEVAL では本稿で紹介したよりも細部においてもう少し性能と精度に最適化され, かつあらゆる入出力の小数点位置の関係に対応したアルゴリズムを実装した. これらは複雑すぎるため本稿の中で紹介することは文量と理解のしやすさを両立する意味で妥当でないと判断したが, 本稿に記載したアルゴリズムが基本となっている.

実装では, 逆数プロトコルに関しては内部計算精度 e_z は 29, 反復回数 4, 平方根の逆数プロトコルに関しては $e_z = 28$, 反復回数 6, 指数関数プロトコルに関しては $e_z = 25$, Taylor 展開の次数 4, 上位ビットの閾値 t は 4 を採用した. 精度に関して, いずれも単精度実数の 23 ビットを超えている. 反復回数に関しては理論的な目安を元に, 実験において十分な精度となった値を採用した. 指数関数の Taylor 展開の次数と上位ビットの閾値 t は, 一方を大きくすれば他方は小さくてよいという性質のものであり, 詳細な計算は割愛するが処理コストを比較した結果 4, 4 とした.

2.6 処理効率について

右シフト/除数公開除算以外に関しては, もはや複雑すぎる, 入出力の指数のパターンにより細部の処理が異なるなど, 詳細な理論的処理効率を述べるのはそれほど有益ではなく, 実機実験を参照いただきたい. なお, おおよそは下記程度である.

(1) 逆数・除数秘匿除算: 通信量 $30|p|$ ビット, ラウンド数 90 程度
(2) 平方根の逆数・平方根: 通信量 $70|p|$ ビット, ラウンド数 110 程度
(3) 指数関数: 通信量 $50|p|$ ビット, ラウンド数 50 程度

3. 実機実験

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

3.1 右シフト/除数公開除算の精度

表 1 は, 右シフトおよび除数公開除算の絶対 L1 誤差 (整数表現としての誤差) である. 小数点位置 t の場合, 誤差はこの $\frac{1}{2^t}$ という意味となる. 10000 データに対して処理した平均と最悪を記載した. passive 安全 ($m = 2$), active 安全 ($m = 3$) とも, 想定通りの誤差であると言える.

表 1 右シフト/除数公開除算の絶対 L1 誤差 [無次元量]

安全性	passive	passive	active	active
項目	平均	最悪	平均	最悪
右シフト	0.3304	1.000	0.483	1.875
除数公開除算	0.335	1.059	0.495	2.000

3.2 逆数, 除数秘匿除算, 平方根とその逆数, 指数関数の精度

表 2 は, その他の提案手法のビット表記での精度である. 10000 データに対して, double 型上の c の標準関数の結果を正として算出した. いずれも最悪値でも単精度実数精度である 23bit を超えていることが分かる.

表 2 逆数, 除数秘匿除算, 平方根とその逆数, 指数関数の精度 [bits]

項目	平均	最悪
逆数	28.84	26.25
除数秘匿除算	30.89	27.41
平方根	28.92	25.64
平方根の逆数	29.34	27.06
指数関数	25.77	24.10

3.3 性能

3.3.1 性能検証: スループット

表 3 は各演算のスループットである. 文献の記載の無いものは, MEVAL 最新版の値である. 一括写像は 16 ビット, それ以外のデータ長は 29 ビットとした. [3] の 1000 万件時の値は, 記載がないため 100 万件時の値を代替として記載した. スループットであるため大きくは変わらないと

考えられる．また, [3] の実装では逆数, 平方根, 指数関数に関しては固定小数点版よりも浮動小数点版の方が速いため, 単精度浮動小数点版を記載した．MEVAL, Sharemind とも, 基本性能の参考値として整数乗算 (MEVAL は 61bit, Sharemind は 32bit) を記載した．基本性能としては, マシン差で説明可能性のある程度の差のようである．

表 3 性能検証: 並列実行時のスループット [M op/s]

件数	1,000	1000 万
右シフト	0.612	14.6
逆数	0.0219	1.244
平方根	0.0147	0.541
指数関数	0.0355	0.792
以下, 比較用		
乗算	1.06	35.8
乗算 ([3])	5.79	23.7
右シフト ([4])	0.07	8.01
右シフト ([3])	0.833	1.82
逆数 ([3])	0.0547	0.0567
平方根 ([4] の右シフト利用)	0.00520	0.285
平方根 ([3])	0.0559	0.0574
指数関数 ([3])	0.0350	0.0391
一括写像	0.0101	0.523

提案した右シフトのスループットが, MEVAL 乗算の半分弱, Sharemind の右シフトの 8 倍程度あることが分かり, 高速性が確認できた．また逆数, 平方根, 指数関数に関しても, Shareind の実装より 1 桁以上高速なことが確認された．固定小数点の実装同士では差はより広がる．右シフトの差よりも大きいため, 各演算のアルゴリズム自体も提案手法が有利であると推測される．また平方根の中の右シフトを [4] に変更すると性能が右シフト自身の比と同程度劣化することから, 右シフトの影響が大きいことが分かる．さらに, 16 ビットの一括写像より逆数, 平方根, 指数関数とも同程度か速いことが確認できた．精度は提案手法が上であるから, 良い結果であるといえる．

3.3.2 性能検証: ラウンド数

表 3 は, ラウンド数の影響を検証するため, 各演算の応答時間をデータ数 1, NW 遅延 100ms として測定したものである．データ数 1 では処理自体の時間は無視できるため, この応答時間から実装上のラウンド数が推定できる．

表 4 性能検証: データ数 1, 遅延 100ms の場合の応答時間 [s]

演算	応答時間	推定されるラウンド数
右シフト	0.210	2
逆数	8.88	89
平方根	11.2	112
指数関数	4.47	45
以下, 比較用		
右シフト ([4])	3.48	35
平方根 ([4] の右シフト利用)	62.52	625
一括写像	4.28	43

いずれの提案手法とも, 見積もりと符合している．右シフトのラウンド数が大きく改善されていること, 右シフトのラウンド数が上位演算全体のラウンド数に影響することが分かる．また, ラウンド数に関しては逆数, 平方根は一括写像より不利であることが分かった．

4. おわりに

本稿では秘密計算において, 機械学習等の実装に重要となる実数演算を設計・実装した．特に, 実際に行われる計算で頻繁に行われる右シフト/除数公開除算において, 通信量 $O(|p|)$, ラウンド数 $O(1)$ を両立する初の方式を提案した．これを用いて逆数, 除数秘匿除算, 平方根とその逆数, 指数関数を実装して性能を報告し, いずれも既存実装の 1 桁程度高速なこと, 精度が単精度実数の精度を超えていることを確認した．

参考文献

[1] 五十嵐大，濱田浩気， 菊池亮，千田浩司: 超高速秘密計算ソートの設計と実装：秘密計算がスクリプト言語に並ぶ日，*CSS2017* (2017).

[2] Bogdanov, D., Niitsoo, M., Toft, T. and Willemson, J.: High-performance secure multi-party computation for data mining applications, *Int. J. Inf. Sec.*, Vol. 11, No. 6, pp. 403–418 (2012).

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

[4] 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).

[5] 五十嵐大，濱田浩気， 菊池亮，千田浩司: 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，*CSS2013* (2013).

[6] 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).

[7] 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).

[8] 大原一真，荒木敏則， 土田光， 古川潤: 異なるサイズの環が混在する不正検知可能なマルチパーティ計算，*SCIS2018* (2018).

[9] 三品気吹，濱田浩気，五十嵐大: 実用的な秘密計算ディープラーニングの実現，*CSS2019* (2019).