近似多ビット乗算と新しい定数ラウンド基本ツールを用いた省ラウンド秘匿除算プロトコル

樋渡 啓太郎[1,2,a)]　大畑 幸矢[2]　縫田 光司[1,2]

概要：秘匿除算は秘匿計算プロトコルの中でも複雑であり，秘匿機械学習などの応用への高い障壁となっている．Bogdanov ら（Int. J. Inf. Sec. 2012）や Morita ら（ISITA 2018）は，環 $\mathbb{Z}_{2^n}$ 上で計算できるプロトコルを構成した．しかし，これらは通信ラウンド数が多いという問題に加え，プロトコルの途中で入出力よりも大きい型の整数を使用するため，比較的非効率的な多倍長整数が必要であるという実装上の難点もあった．本稿では，二つのアプローチで除算プロトコルの効率化を図る．一つは，入出力と同じサイズの整数で完結するプロトコルの構成であり，もう一つは，大小比較など，内部で使用する基本ツールの改良である．これらの改良により，多倍長整数を呼び出す必要をなくすだけでなく，Morita らの結果と比較して，例えば 64 ビット整数の除算において 64%近くラウンド数を削減することに成功した．

キーワード：秘匿計算, 除算プロトコル

An Efficient Secure Division Protocol Using Approximate Multi-bit Product and New Constant-Round Building Blocks

Keitaro Hiwatashi[1,2,a)]　Satsuya Ohata[2]　Koji Nuida[1,2]

Abstract: Secure division is very complex calculation among the secure multi-party computation (MPC), and it is a barrier to applications of MPC such as secure machine learning. Bogdanov et al. (Int. J. Inf. Sec. 2012) and Morita et al. (ISITA 2018) constructed division protocols working in $\mathbb{Z}_{2^n}$. However, they had difficulties that those protocols needed many communication rounds and they needed to use bigger integers than in/output. In this paper, we construct a more efficient division protocol. Our new protocol uses only the same size integers as in/output, and communication rounds are reduced to about 64% in Comparison with Morita et al.

Keywords: Secure Multi-Party Computation, Division Protocol

1. はじめに

秘匿計算 (Secure Multi-Party Computation，以下 MPC) は第三者への情報漏洩を防止する従来の暗号とは異なり，当事者にも情報を秘匿したまま，データの処理，管理を可能にする技術である．Yao[17] によって初めて提唱されて以降，盛んに研究されてきた． MPC を行う手法としては準同型暗号を用いる方法，秘匿回路を用いる方法，秘密分散を用いる方法などがあるが，今回は近年の研究 [2], [7] で高いスループットが出せることがわかっている秘密分散を用いた手法を採用する．秘密分散を用いた MPC に関しては ABY [9]，SCALE-MAMBA[*1]などの高速なソフトウェアも存在し，実用的な社会実装に近づいている．また，秘密分散を用いた MPC は以下に述べる Client-Aided モデルのもとで情報理論的な安全性を達成できることも一つの利点である．

[1] 東京大学
The University of Tokyo
[2] 産業技術総合研究所
AIST
[a)] keitaro_hiwatashi@mist.i.u-tokyo.ac.jp

[*1] https://homes.esat.kuleuven.be/~nsmart/SCALE/

MPC の社会実装という観点から，Client-Server モデル [4] というモデルが採用されることがある．このモデルはまず，任意の数のクライアントが自身の入力を秘密分散に基づき N 個の計算サーバに送る．そして，計算自体は計算サーバ同士内で行い，計算結果をクライアントに送り返したのち，クライアントが出力を復元する，というモデルである．例えば，クラウドコンピューティングなどの応用においてはこのモデルはマッチしている．本論文ではさらに，計算サーバが用いる補助入力の生成をクライアントが担う，という Client-Aided モデル [11], [12], [13] を採用する．Client-Aided モデルではクライアント側の計算コスト，通信コストが増えてしまうが，計算サーバだけでの生成が非効率的な補助入力の生成が容易であるという利点がある．

また，本論文では計算サーバの個数は 2 つであるものとし，計算サーバ間のネットワーク環境としては現実的な状況である WAN 環境を想定する．WAN 環境においては通信遅延が律速になることが多く，通信ラウンド数がプロトコルの性能の大きな指標となる．

本論文では MPC のうち，秘匿除算を取り扱う．秘匿除算はデータの正規化やソフトマックス関数の計算など，秘匿機械学習において重要な計算を行う上で避けては通れないものであり，機械学習に限らずとも例えば k-means クラスタリングやカイ二乗検定などのデータ分析においても必須の演算である．しかし，秘匿除算は加算や乗算，大小比較などの他の基本演算に比べてはるかに処理が重く，前述のような応用の実用化に際して大きな障壁となっている．

1.1 既存方式における課題

除算を扱う研究は複数存在する．固定小数点数で行うもの [6]，浮動小数点数で行うもの [1]，整数で行うもの [5], [12] などがあげられる．しかし，[1], [6] では，十分大きい素数 p をとってきて，$\mathbb{Z}_p$ 上で数値を表現するため，プロトコルの途中で法 p による剰余の計算を明示的に行う必要がある．また，64 ビット整数の除算を行おうとすると，2^{64} より大きい素数を用いる必要があるため，多倍長整数を用いる必要があった．[5], [12] では，環 $\mathbb{Z}_{2^n}$ 上で数値を扱うため，剰余の計算がビットシフトで済むようになっているが，入出力よりも大きいビットサイズ n を要し，多倍長整数を使用しなければならないことには変わりなかった．しかし，多倍長整数ライブラリを用いた加算，乗算は 64 ビット整数での加算，乗算に比べてはるかに遅く，例えば [11] では 100 倍近く遅いと言及されている．秘密分散を用いた MPC では通常，通信遅延が律速ではあるものの，例えば [15] で Comparison を並列実行した際，10000 並列ほどでローカルの計算時間が通信遅延を上回るという結果があるように，プロトコルの応用によってはローカルの計算が律速になることがある．このような事実を考慮すると，多倍長整数を使わずに入出力と同じビットサイズで完結するプロトコルの構成が望まれる．

1.2 本研究の貢献

本研究では二つのアプローチで除算プロトコルの効率化を図った．一つ目のアプローチはビットサイズの拡張 (以下，ビット拡張と呼ぶ) が不要であるプロトコルの構築である．既存方式においてビット拡張が必要であった部分を，近似的にではあるが拡張なしで計算するプロトコルを作り，生じる誤差を最後に修正する，という方針でプロトコルの構築を行った．

二つ目のアプローチは，大小比較など，内部で使用する基本プロトコルの効率化である．本研究では大小比較の内部でも使用する基本機能である，Overflow と呼ばれる機能を実現する定数ラウンドプロトコルを考案した．[8], [13], [14] などの既存の定数ラウンドプロトコルは，体 $\mathbb{Z}_p$ 上で計算するものであったため，64 ビット整数に適用する際は 64 ビットよりも大きい素数を用意する必要があり，ビット拡張が免れなかった．一方，本研究で提案するプロトコルは環 $\mathbb{Z}_{2^n}$ 上で計算できるものであるため，ビット拡張が不要である．またこの提案プロトコルにより，[13] では 5 ラウンドであった大小比較が 4 ラウンドになっており，加えて通信ラウンドに限らず通信量も改善できている．

これら二つの改善により，ビット拡張を不要にするだけでなく，ラウンド数の削減にも成功した．具体的には，例えば 64 ビット整数除算において必要な通信ラウンド数を [12] と比較して 64%ほど削減することに成功した．

2. 準備

この章では秘密分散を用いた MPC の基本的な知識を振り返る．

2.1 秘密分散

この節では，秘密分散について簡単に記す．本論文で扱う $\mathbb{Z}_{2^n}$ 上の 2-out-of-2 秘密分散は Share, Reconst という二つのアルゴリズムからなる:

Share

input : $x \in \mathbb{Z}_{2^n}$

output : $[\![x]\!] = ([\![x]\!]_1, [\![x]\!]_2)$

$$\text{s.t. } [\![x]\!]_1 \overset{\mathrm{R}}{\in} \mathbb{Z}_{2^n},\ [\![x]\!]_1 + [\![x]\!]_2 = x \mod 2^n$$

Reconst

input : $([\![x]\!]_1, [\![x_2]\!])$

output : $x = [\![x]\!]_1 + [\![x]\!]_2 \mod 2^n$.

$[\![x]\!]$ は x のシェアと呼ばれる．

この秘密分散のもとでは，線形演算は各シェアごとの

線形演算で実現でき，乗算は Beaver triplet [3] を補助入力として使うことで実現できる．Beaver triplet とは，各パーティの知らないランダムな数 a, b と $c = ab$ を満たすようなシェアの組 $([\![a]\!], [\![b]\!], [\![c]\!])$ であり，この補助入力を用いて乗算は以下のように計算される．まず，各パーティは $[\![x-a]\!], [\![y-b]\!]$ を計算し，その後 $\mathsf{Reconst}([\![x-a]\!])$, $\mathsf{Reconst}([\![y-b]\!])$ を実行し，$x' = x - a,\ y' = y - b$ を得る．次に，パーティ 1 が $x'y' + x'[\![b]\!]_1 + y'[\![a]\!]_1 + [\![c]\!]_1$ を，パーティ 2 が $x'[\![b]\!]_2 + y'[\![a]\!]_2 + [\![c]\!]_2$ を計算すると，これは xy のシェアとなっており，乗算が計算できているとわかる．

また，プロトコルによっては論理値を扱うことがあるが，本論文では論理値を $\{0, 1\} \in \mathbb{Z}_{2^n}$ で扱う[*2]．

2.2 攻撃者のモデル

本論文で想定する攻撃者は semi-honest な攻撃者とする．つまり，プロトコルには正しく従うものとする．2.1 節の線形演算，乗算は semi-honest 安全であることが証明されており，これらの組み合わせで得られるプロトコルも Composition Theorem [10] により semi-honest 安全であることがわかるため，本論文ではこれ以降安全性については議論しない．

2.3 基本プロトコル

この節では，特に構成を示すことなく使用するプロトコルの機能を紹介する．プロトコルの詳細は [16] を参照されたい．

- $\mathsf{RightShift}([\![x]\!], i)$：$x$ を i ビット右シフトしたもののシェアを出力する．
- $\mathsf{ExtractBit}([\![x]\!], i)$：$x$ の i ビット目[*3]の値 (0 or 1) のシェアを出力する．
- $\mathsf{Comparison}([\![x]\!], [\![y]\!])$：$x \overset{?}{<} y$ を表す論理値のシェアを出力する．
- $\mathsf{Equal\_zero}([\![x]\!])$：$x \overset{?}{=} 0$ を表す論理値のシェアを出力する．
- $\mathsf{MSNZB}([\![x_1]\!], \ldots, [\![x_m]\!])$：論理値のシェアの配列 $\{[\![x_i]\!]\}$ を入力として以下を満たす論理値のシェアの配列 $\{[\![y_i]\!]\}$ を出力する:

$$y_i = \begin{cases} 1 & x_i = 1,\ x_j = 0\ (\forall j < i) \\ 0 & \text{otherwise.} \end{cases}$$

3. ビット拡張不要の秘匿除算プロトコルの構成

秘匿除算は，シェア $[\![N]\!], [\![D]\!]$ を入力とし，商 $\lfloor N/D \rfloor$ のシェアを出力とするプロトコルである．ただし，ここでは $D \neq 0$ とする．

[*2] 1 が真に対応し，0 が偽に対応する
[*3] 下位から数えて i ビット目

既存方式 [1], [5], [6], [12] はすべて GoldSchmidt 法 [10] に基づいている．Goldschmidt 法とは，

$$\frac{N}{D} = \frac{NY_0Y_1\ldots}{DY_0Y_1\ldots}$$

として，分母を 1 に近づけることで，真の値を分子で近似する，という手法である．特に，D のビット長を d として，

$$Y_0 = 2^{-d},\ \varepsilon = 1 - DY_0,\ Y_i = 1 + \varepsilon^{2^{i-1}}\ (i \geq 1)$$

とすることが多い．ここでは，$Y_{\lceil \log_2 n \rceil}$ まで考慮することとし，

$$\frac{N}{D} \simeq N2^{-d}(1 + \varepsilon + \cdots + \varepsilon^n)$$

で近似する[*4]．

この方式では，内部で小数を扱う必要がある．小数の取り扱いについては，適当な n' をとってきて，$2^{n'}$ 倍して丸めることで整数として取り扱うことが多く，ここでもそのように取り扱う．

[12] はこの Goldschmidt 法を 2 パーティで計算するプロトコルを実現した．しかし，[12] では，小数表現のためのパラメータ n' として，$n' = n + 3 + \lceil \log(\lceil \log n + 2 \rceil - 1) \rceil$ が必要であり，内部で $n + 2n'$ ビットの整数を扱う必要があった．以下，ビット拡張なしで Goldschmidt 法を実現するプロトコルを構成していく．以下では，小数表現のためのパラメータ n' は n に等しいものとする[*5]．

3.1 近似多ビット乗算 –MultBit プロトコル

現状，ビット拡張が必要なのは小数同士の積の部分である．しかし，[12] では，小数同士の積の直後に RightShift を施している．これは，小数同士の積を行うと，整数表現のために掛けていた $2^{n'}$ が 2 回掛かることになってしまうためである．単純に小数同士の積だけを計算する場合は確かにビット拡張しなければオーバーフローを起こしてしまうが，最後に RightShift を施すことを考えると，n ビット内で収まるプロトコルを構成することができる．基本的なアイデアは，$y[i]$ を y の i ビット目として，

$$xy2^{-n} = x2^{-n}\sum_{i=1}^{n} y[n-i+1]2^{n-i}$$
$$= \sum_{i=1}^{n} x2^{-i}y[n-i+1]$$

における，$x2^{-i}$ を x の RightShift で代用することである．これは，確かにビット拡張なしで計算することができる．しかし，ビット拡張ありで計算するときに比べ，丸めによる誤差が大きくなってしまう．この誤差の大きさに関しては 3.4 節において詳細に評価している．

[*4] $Y_1Y_2\ldots Y_{\lceil \log_2 n \rceil} = 1 + \varepsilon + \cdots + \varepsilon^{2^{\lceil \log_2 n \rceil} - 1}$ であるが，ε^n より後の項は打ち切っている．
[*5] ビット拡張を行わないことの必然的な要件として，$n' \leq n$ であるので，ここでは最も丸め誤差の少ない，$n' = n$ を使用することとする．

3.2 多入力 MultBit

前述のプロトコルは 2 入力であるが，自然と多入力に拡張することができる．つまり，

$$xyz2^{-2n} = \sum_{(i,j)\in\{1,\dots,n\}^2} x2^{-i-j}y[n-i+1]z[n-j+1]$$

という変形を使用するのである．(これは 3 入力の例であるが，同様にして，一般の入力数に拡張できる．) これを用いると，Goldschmidt 法におけるループ回数を減らすことができるので，全体のラウンド数を減らすことが可能である．ただし，入力数を増やすと，計算量と通信量と誤差が指数的に増えるため，今回は最大で 4 入力のものを使用する．以下に，N 入力 MultBit のプロトコルを記す．ただし，step 4 の総和は正の整数 $i_1,\dots,i_{N-1}$ に関してとるものとし，また $i=\sum_{j=1}^{N-1} i_j$ とする．

Protocol 1 N_MultBit

Functionality: $\hat{z} \simeq \hat{x}\prod_{i=1}^{N-1}(\hat{y}_i 2^{-n})$
Input: $[\![\hat{x}]\!], [\![\hat{y}_1]\!], \dots, [\![\hat{y}_{N-1}]\!]$
Output: $[\![\hat{z}]\!]$
1: **for** $i \in \{1,2,\dots,n-1\}$ **do**
2: $\quad [\![\hat{x}_i]\!] \leftarrow \mathsf{RightShift}([\![\hat{x}]\!], i)$
3: **end for**
4: $[\![\hat{z}]\!] \leftarrow \sum_{i_1+\cdots+i_{N-1}\le n-1} [\![\hat{x}_i]\!] \prod_{j=1}^{N-1} \mathsf{ExtractBit}([\![\hat{y}_j]\!], n-i_j+1)$

なお，今回は特に $\hat{y}_1=\hat{y}_2=\cdots=\hat{y}_{N-1}$ である場合を扱うので，特に断りがない場合，N_MultBit$([\![\hat{x}]\!],[\![\hat{y}]\!])$ で，N_MultBit$([\![\hat{x}]\!],[\![\hat{y}]\!],\dots,[\![\hat{y}]\!])$ を表すものとする．

3.3 多入力 MultBit を用いた，Goldschmidt 法の構成

上記の N_MultBit を用いて，Goldschmidt 法を構成する．まず，N_MultBit を用いて，入力のべき乗を求める Power プロトコルを構成する．次に Power を用いて，Goldschmidt 法の実装である，QGuess プロトコルを構成する．3.2 節で述べたように，ここでは，N_MultBit の入力数 N は 4 以下としている．また，QGuess の構成に当たって，ReciprocalGuess [12] を使用しているが，これは $[\![D]\!]$ を入力にとり，$2^{n-d}(d:D$ のビット長$)$ を出力とするプロトコルであり，Comparison と同じラウンド数で実現される．

Protocol 2 Power

Functionality: $\hat{\delta}_i \simeq \hat{\varepsilon}^i 2^{-(i-1)n}\ (i=1,\dots,m)$
Input: $[\![\hat{\varepsilon}]\!], m$
Output: $([\![\hat{\delta}_1]\!],\dots,[\![\hat{\delta}_m]\!])$
1: $[\![\hat{\delta}_1]\!] \leftarrow [\![\hat{\varepsilon}]\!]$
2: **for** $i=1,2,\dots,\lceil \log_4 m\rceil$ **do**
3: $\quad$ **for** $j=1,2,3$ **do**
4: $\quad\quad$ **for** $k=1,2,\dots,4^{i-1}$ **do**
5: $\quad\quad\quad$ **if** $4^{i-1}j+k>m$ **then** break
6: $\quad\quad\quad$ $[\![\hat{\delta}_{4^{i-1}j+k}]\!] \leftarrow (j+1)$_MultBit$([\![\hat{\delta}_k]\!],[\![\hat{\delta}_{4^{i-1}}]\!])$
7: $\quad\quad$ **end for**
8: $\quad$ **end for**
9: **end for**

Protocol 3 QGuess

Functionality: $Q' \simeq \lfloor \frac{N}{D} \rfloor$
Input: $[\![N]\!], [\![D]\!]$
Output: $[\![Q']\!]$
1: $[\![\hat{D'}]\!] \leftarrow \mathsf{ReciprocalGuess}([\![D]\!])$
2: $[\![\hat{\varepsilon}]\!] \leftarrow -[\![\hat{D'}]\!] \times [\![D]\!]$
3: $([\![\hat{\delta_1}]\!],\dots,[\![\hat{\delta_n}]\!]) \leftarrow \mathsf{Power}([\![\hat{\varepsilon}]\!], n)$
4: $[\![\hat{\delta}]\!] \leftarrow \sum_{i=1}^{n} [\![\hat{\delta_i}]\!]$
5: $[\![N']\!] \leftarrow 2$_MultBit$([\![N]\!],[\![\hat{D'}]\!])$
6: $[\![Q']\!] \leftarrow [\![N']\!] + 2$_MultBit$([\![\hat{\delta}]\!],[\![N']\!])$

3.4 誤差解析

N_MultBit を使用すると，丸めによる誤差が大きくなってしまうため，QGuess で出力される値は一般には真の値よりも小さくなってしまう．しかし，その誤差の大きさは以下の補題によって見積もることができる．

補題 3.1

x, y を 2 進数表現したとき，非零ビットが $x[i]$ $(l_x \le i \le u_x)$, $y[j]$ $(l_y \le j \le u_y)$ に限られるとする．(この範囲内がすべて非零ビットとは限らない．) この時，$z \leftarrow N$_MultBit(x,y) に関して，

$$x(y2^{-n})^{N-1} - e \le z \le x(y2^{-n})^{N-1},$$
$$e = \sum_{\substack{(I_1,\dots,I_{N-1})\\ \in\{l_y,\dots,u_y\}^{N-1}}} x_0 2^{-s} - (x_0 \gg s),$$
$$s = \sum_{j=1}^{N-1}(n-I_j-1),\ x_0 = 2^{u_x} - 2^{l_y-1}$$

が成立する．(ただし，$\gg$ は右ビットシフトを意味する．) また，z の非零ビットは

$$l_z = l_x + (N-1)l_y - (N-1)(n+1)$$
$$u_z = u_x + (N-1)u_y - n(N-1)$$

として，$z[i](l_z \le i \le u_z)$ に限られる．

証明

まず，N_MultBit には引き算が現れないので，$1-2 = 7 \mod 8$ といったアンダーフローは起こらない．よって，$z = \sum_{i_1+\cdots+i_{N-1}\le n-1} (x \gg i) \prod_{j=1}^{N-1} y[n-i_j+1]$ として，与不等式を示していく[*6]．$f(x,i) = x2^{-i} - (x \gg i)$, $i=\sum_{j=1}^{N-1} i_j$ として，

$$x(y2^{-n})^{N-1} - z = \sum_{\substack{(I_1,\dots,I_{N-1})\\ \in\{l_y,\dots,u_y\}^{N-1}}} f(x,i) \prod_{j=1}^{N-1} y[n-i_j+1]$$

であり，x の非零ビットの位置の仮定から，$f(x,i)$ は $x=x_0$ のとき最大，$x=0$ のとき最小となる．また，y の非零ビッ

[*6] 与不等式が示せれば，N_MultBit においてオーバーフローも起こらないとわかるので，z が出力値と一致する．

トの仮定から，総和を考えるインデックス $i_1, \ldots, i_{N-1}$ はそれぞれ $\{l_y, \ldots, u_y\}$ に限ってよい．これらより，与不等式が導かれる．

次に，z の非零ビットとしてあり得る位置を考える．一番下位の非零ビットは x の一番下位の非零ビットを最も右シフトさせた位置にあり，右シフトのビット数 i は最大で，$(n+1-l_y)(N-1)$ であるので，$l_z = l_x + (N-1)l_y - (N-1)(n+1)$ とできる．また，一番上位の非零ビットに関しては，$x \leq 2^{u_x},\ y \leq 2^{u_y}$ より，$z \leq 2^{u_x+(N-1)(u_y-n)}$ であるため，高々 $u_x + (N-1)u_y - n(N-1)$ ビット目までしか非零になりえない． □

補題 3.2

非負の数 a, b, x, y に対して，

$$N \in \{2,\ 3,\ 4\},$$
$$\max\{0, x-a\} \leq x' \leq x,\ \max\{0, y-b\} \leq y' \leq y,$$
$$z = x(y2^{-n})^{N-1},\ z' = x'(y'2^{-n})^{N-1}$$

のとき，

$$\max\{z-c, 0\} \leq z' \leq z$$
$$c = (N-1)xy^{N-2}2^{-(N-1)n}b + (y2^{-n})^{N-1}a$$

が成立．

証明

$z' \leq z$ と $0 \geq z'$ はよいので，$z - c \leq z'$ を示す．まず，$x - a \geq 0,\ y - b \geq 0$ のとき，

$$\begin{aligned}
x'y'^{N-1} &\geq (x-a)(y-b)^{N-1} \\
&\geq \begin{cases} xy - ay - bx + ab & N=2 \\ xy^2 - 2(x-a)yb + (x-a)b^2 - ay^2 & N=3 \\ xy^3 - 3(x-a)y^2b & \\ \qquad +3(x-a)yb^2 - (x-a)b^3 - ay^3 & N=4 \end{cases} \\
&\geq \begin{cases} xy - ay - bx & N=2 \\ xy^2 - 2xyb - ay^2 & N=3 \\ xy^3 - 3xy^2b - ay^3 & N=4 \end{cases} \\
&= xy^{N-1} - c2^{(N-1)n}
\end{aligned}$$

より，与式は成立する．なお，2 行目から 3 行目の変形には $a \leq x,\ b \leq y$ を用いている．$x - a < 0$ or $y - b < 0$ のときは，$xy^{N-1} - c2^{(N-1)n} \leq 0$ であり，$z' \geq 0$ と合わせると，与式は成立する． □

上記の補題をもとに，QGuess の出力と $\lfloor \frac{N}{D} \rfloor$ の差を数値計算によって見積もることができる．特に，$n = 32,\ 64$ の時は，誤差の大きさが $2n$ で抑えられることを確認できる．

3.5 ErrorCorrect

[12] では小数表現の丸めによって生じる誤差を，$2^{n'-n}\hat{D}'N$ という補正項を足すことで打ち消し，正当性を担保している．しかし，MultBit を用いた場合，丸めによって生じる誤差は [12] のものよりも大きく，誤差を打ち消すような補正項を理論的に導出することは困難である．今回は，3.4 節において見積もった誤差をもとに，真の値を求めるプロトコル ErrorCorrect を構成することで，正当性を担保する．

ErrorCorrect の基本的なアイデアは，真の値が $Q', Q'+1, \ldots, Q'+A$ の範囲にあることがわかっているときに，$Q'D, (Q'+1)D, \ldots, (Q'+A)D$ と N の大小比較を行い，True から False に変化するところを検出するというものである．

しかし，このままでは，$\lfloor \frac{N}{D} \rfloor D \leq N < 2^n \leq (\lfloor \frac{N}{D} \rfloor + 1)D$ の場合に正しく検出できない．ここでは，以下のような仮定をおいて，この問題点を解消する[*7]．

仮定：Q' が 0 のとき，$\lfloor \frac{N}{D} \rfloor = 0$ or 1 である．

この仮定のもとでは $D,\ 2D,\ \ldots,\ AD$ と $N - Q'D$ の大小比較，D と N との大小比較を同時に行い，$Q' = 0$ か否かを表す論理値をかけて出力することで，正しい出力が得られる．

Protocol 4 ErrorCorrect

Functionality: 近似値 Q' と誤差範囲 A をもとに $Q = \lfloor \frac{N}{D} \rfloor$ を出力

Input: $[\![N]\!], [\![D]\!], [\![Q']\!], A$
Output: $[\![Q]\!]$

1: $[\![N']\!] \leftarrow [\![N]\!] - [\![Q']\!] \times [\![D]\!]$
2: $[\![\delta]\!] \leftarrow$ Equal_zero($[\![Q']\!]$)
3: **for** $i = 1, 2, \ldots, A$ **do**
4: $\quad [\![b_i]\!] \leftarrow$ Comparison($[\![N']\!], i \times [\![D]\!]$)
5: **end for**
6: $[\![b]\!] \leftarrow$ MSNZB($[\![b_1]\!], \ldots, [\![b_A]\!]$)
7: $[\![q]\!] \leftarrow \sum_{i=1}^{A} (i-1) \times [\![b[i]]\!]$
8: $[\![Q]\!] \leftarrow [\![\delta]\!] \times ([\![1]\!] - [\![b_1]\!]) + ([\![1]\!] - [\![\delta]\!]) \times ([\![Q']\!] + [\![q]\!])$

3.6 除算プロトコルの構成

QGuess と ErrorCorrect を合わせることにより，整数除算プロトコル Divide が得られる．なお，プロトコル中のパラメータ A は $n = 32,\ 64$ においては $A = 2n$ としてよい．

Protocol 5 Divide

Functionality: $Q = \lfloor \frac{N}{D} \rfloor$ を出力

Input: $[\![N]\!], [\![D]\!]$
Output: $[\![Q]\!]$

1: $[\![Q']\!] \leftarrow$ QGuess($[\![N]\!], [\![D]\!]$)
2: $[\![Q]\!] \leftarrow$ ErrorCorrect($[\![N]\!], [\![D]\!], [\![Q']\!], A$)

4. 基本プロトコル Overflow の効率化

この章では，RightShift や ExtractBit，Comparison の内

[*7] この仮定は QGuess の出力に対して確かに成立する．

部で使用する基本機能である，Overflow の定数ラウンドプロトコルを構成する．Overflow は $([\![x]\!], t)$ を入力とし，$[\![x]\!]_1[t\dots1] + [\![x]\!]_2[t\dots1] \geq 2^t$ を表す論理値のシェアを出力とする．ただし，$[\![x]\!]_i[t\dots1]$ はパーティ i のシェアの t ビット目までの値である．

以下が Overflow を構成するにあたって使用するサブプロトコルの一覧である．これらのプロトコルは，$n < p < \sqrt{2^n}$ を満たす素数 p に対する剰余体 $\mathbb{Z}_p$ 上でのプロトコルである．$p < \sqrt{2^n}$ であるから，これらのプロトコルは n ビット以内の整数だけで実現させることができる．

- Mult$([\![x]\!], t)$：x^t のシェアを出力する．
- Equal_one$([\![x]\!])$：$0 \leq x \leq n$ であるという仮定の下，$x = 1$ かどうかを表す論理値のシェアを出力する．
- assump_Overflow$([\![x]\!])$：$x < \frac{p}{2}$ であるという仮定の下，$[\![x]\!]_1 + [\![x]\!]_2 \geq p$ を表す論理値のシェアを出力する．

4.1 Mult

通常の積を計算する際の補助入力である Beaver triplet は $c = ab$ を満たす $([\![a]\!], [\![b]\!], [\![c]\!])$ の組であった．ここで，補助入力として $([\![a]\!], [\![a^2]\!], \dots, [\![a^t]\!])$ を使用すると，以下のように Mult の計算が 1 ラウンドで実現できる．まず，各パーティが $[\![x]\!] - [\![a]\!]$ を計算し，Reconst により $y = x - a$ を得る．次に，$x^t = (y+a)^t$ を展開した式は，$[\![a]\!], [\![a^2]\!], \dots, [\![a^t]\!]$ に関して線形であるので，$[\![x^t]\!]$ を通信無しで計算することができる．

4.2 Equal_one

$\mathbb{Z}_p$ 上の n 次多項式 f で，

$$f(x) = \begin{cases} 1 & x = 1 \\ 0 & x = 0, 2, 3, \dots, n \end{cases}$$

を満たすものが存在する．$f(x)$ の計算は $x^t, t = 1, 2, \dots, n$ の計算ができればよく，これは Mult$([\![x]\!], t)$ によって 1 ラウンドで計算できる．

4.3 assump_Overflow

$x < \frac{p}{2}$ という仮定のもとでは，$[\![x]\!]_1 + [\![x]\!]_2 < p \Leftrightarrow [\![x]\!]_1 < \frac{p}{2} \wedge [\![x]\!]_2 < \frac{p}{2}$ である．右辺は各パーティがローカルで計算できる論理値の積であるので，1 ラウンドで計算可能である．

4.4 Overflow

前述のプロトコルを使用して，Overflow を構成していく．

まず，各パーティのシェアを 2 進数展開したときの i ビット目の値を $[\![x]\!][i]$ と書く．

ここで，長さ n の配列 y を

$$y[i] = [\![x]\!]_1[i] + [\![x]\!]_2[i]$$

で定める．また，$y[t\dots1]$ で 1 番目から t 番目までの要素を表す．例えば，$n = 3$, $x = 5$, $[\![x]\!]_1 = 6$, $[\![x]\!]_2 = 7$ としたとき，$y = (2, 2, 1)$ となる．(2 進数表記に合わせて，y も一番右を 1 番目としている．)

すると，x のシェアが t ビット目で Overflow している，つまり，$[\![x]\!]_1[t\dots1] + [\![x]\!]_2[t\dots1] \geq 2^t$ となる必要十分条件は，

> $y[t\dots1]$ の中に 2 が含まれ，y を t 番目から 1 番目まで順に見たときに，0 より先に 2 が現れる

と書ける．以下，この条件を計算するプロトコルを構成する．

ここで着目すべきは，y の各要素は 0 or 1 or 2 であり，対応する各パーティのシェア (つまり，$[\![x]\!]$ の各ビット) は 0 or 1 であるため，各 i に対して $([\![x]\!]_1[i], [\![x]\!]_2[i])$ は $y[i]$ の $\mathbb{Z}_p$ 上の秘密分散と自然にみなすことができることである．以下ではこの事実をもとに，$\mathbb{Z}_p$ 上の演算を考える．

4.4.1 第 1 段階

まず，y における 0 と 2 の位置を明らかにするために，1 を 0 に，0 と 2 を 1 に変換する写像を y の各要素に作用させる．つまり，

$$z[i] = (y[i] - 1)^2$$

なる配列 z を計算する．この計算には 1 ラウンドかかる．また，並行して，0 と 1 を 0 に，2 を 1 に変換した配列 z' を計算しておく．つまり，

$$z'[i] = \frac{y[i](y[i] - 1)}{2}$$

である．

4.4.2 第 2 段階

次に，第 1 段階において得られた配列 z の要素の中で 1 であるもののうち，一番左にあるもの (つまり，一番，t 番目に近いもの) を抽出したい．

まず，z の逆順累積和を計算する．つまり，

$$w[i] = \sum_{k=i}^{t} z[k]$$

なる配列 w を計算する．この計算はローカルで可能であるので 0 ラウンドである．

次に，w の各要素に Equal_one を作用させる．つまり，

$$w'[i] \leftarrow \text{Equal\_one}(w[i])$$

とする．ここで，$w[i]$ は 1 以上 t 以下であるため，Equal_one を用いることができる．ここでは，Equal_one の計算にのみ通信が必要であり，通信ラウンド数は 1 ラウンドである．ただし，ここで得られる配列 w' は非零成分が 1 つだけとは限らない．なぜならば，w において，1 に等しい要素が連続して現れることがあるからである．ここで，w' において非零の部分のうち，一番左 (つまり一番 t 番目に近いも

の) 以外の部分に対応する y の要素は 1 であることに注意する．

4.4.3 第 3 段階

最後に，第 2 段階で求めた w' の非零成分が y において 2 に対応しているかどうかを確認すればよい．この確認のために，w' と z' の内積 s を計算する．

$$s = \sum_{i=1}^{t} w'[i] \times z'[i].$$

すると，この s が確かに欲しい出力となっている．

ただし，この s は $\mathbb{Z}_p$ 上のシェアであるので，これを $\mathbb{Z}_{2^n}$ 上のシェアに戻す必要がある．ここで [12] にあるような CastUp を行う必要がある．CastUp において，通信が必要な計算は全体の桁あふれのみである．今回 s は 0 or 1 であるため，assump_Overflow を適用することができ，1 ラウンドで計算することができる．

4.4.4 プロトコルのまとめ

以上をまとめたプロトコルは以下のようになる．y は本章冒頭で定めたものとする．

Protocol 6 Overflow

Functionality: $s = [\![x]\!]_1[t\dots1] + [\![x]\!]_2[t\dots1] \overset{?}{\geq} 2^t$
Input: $[\![x]\!], t$
Output: $[\![s]\!]$
1: $[\![z[i]]\!] \leftarrow ([\![y[i]]\!] - 1)^2$
2: $[\![z'[i]]\!] \leftarrow \frac{[\![y[i]]\!]([\![y[i]-1]\!])}{2}$
3: $[\![w[i]]\!] \leftarrow \sum_{k=i}^{t} [\![z[k]]\!]$
4: $[\![w'[i]]\!] \leftarrow$ Equal_one($[\![w[i]]\!]$)
5: $[\![s']\!] \leftarrow \sum_{i=1}^{t} [\![w'[i]]\!] \times [\![z'[i]]\!]$
6: $[\![s]\!] \leftarrow$ CastUp($[\![s']\!]$)

この Overflow にかかるラウンド数は，step 1，2 を並列に行う部分で 1 ラウンド，step 4-6 で各 1 ラウンドの合計 4 ラウンドである．ここで，さらに補助入力として，$([\![a]\!], \dots, [\![a^t]\!], [\![b]\!], [\![ba]\!], \dots, [\![ba^t]\!])$ を使用すると，4.1 節と同様に計算することで step 4-5 をまとめて 1 ラウンドで計算することができる．よって，3 ラウンドで Overflow を計算できるとわかる．

この Overflow の定数ラウンドプロトコルにより，RightShift，ExtractBit，Equal_zero が 3 ラウンドで，Comparison が 4 ラウンドで実現できる．また，MSNZB も Overflow と似たような構成で 3 ラウンドで実現することが可能である．

5. 性能評価

この章では，3，4 章を組み合わせた除算プロトコルの性能をラウンド数の観点から評価する．

N_MultBit は RightShift(並行して ExtractBit) に加え，N 個の数の積が出てくるので，合計で，$3 + \lceil \log_2 N \rceil$ ラウンドかかる．Power は $\lceil \log_4 n \rceil$ 回分 4_MultBit を呼び出すので，$(3+2) \times \lceil \log_4 n \rceil = 5\lceil \log_4 n \rceil$ ラウンドかかる．また，QGuess における step 5 は step 3 と並行して行えるため，実質的にラウンド数はかからない．以上をまとめると，QGuess のラウンド数は $4+1+5\lceil \log_4 n \rceil + 3 + \lceil \log_2 2 \rceil = 9 + 5\lceil \log_4 n \rceil$ ラウンドとなる．また，ErrorCorrect は step 2 と step 3-6 が並列化できることに注意すると，$1 + $Comparison$ + $MSNZB$ + 1 = 9$ ラウンドとなる．さらに，Overflow の最後のステップに行った CastUp を行わず，Comparison の結果と MSNZB に渡す値を $\mathbb{Z}_p$ 上のシェアとすると，ErrorCorrect は 7 ラウンドで計算できる．これらより，秘匿除算プロトコル Divide のラウンド数は 32 ビット整数，64 ビット整数どちらにおいても，31 ラウンドであるとわかる．

一方 [12] においては，基本プロトコルの実装として [16] を参照しており，[16] におけるラウンド数で計算すると，除算プロトコルにかかる通信ラウンド数は 32 ビット整数で 69 ラウンド，64 ビット整数で 87 ラウンドである．

よって，プロトコル全体の改良，基本プロトコルの改良という二つの改良により，[12] と比較して，32 ビット整数で 55%，64 ビット整数で 64%のラウンド数削減に成功しているとわかる．

謝辞 本研究開発は JST CREST JPMJCR19F6 の補助，および総務省の委託を受けて実施したものである．

参考文献

[1] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure computation on floating point numbers. In *NDSS*, 2013.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 805–817. ACM, 2016.

[3] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Annual International Cryptology Conference*, pp. 420–432. Springer, 1991.

[4] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pp. 192–206. Springer, 2008.

[5] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, Vol. 11, No. 6, pp. 403–418, 2012.

[6] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *International Conference on Financial Cryptography and Data Security*, pp. 35–50. Springer, 2010.

[7] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority mpc for malicious adversaries. In *Annual International Cryptology Conference*, pp. 34–64. Springer, 2018.

[8] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equal-

ity, comparison, bits and exponentiation. In *Theory of Cryptography Conference*, pp. 285–304. Springer, 2006.

[9] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[10] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[11] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38. IEEE, 2017.

[12] Hiraku Morita, Nuttapong Attrapadung, Satsuya Ohata, Koji Nuida, Shota Yamada, Kana Shimizu, Goichiro Hanaoka, and Kiyoshi Asai. Secure division protocol and applications to privacy-preserving chi-squared tests. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pp. 530–534. IEEE, 2018.

[13] Hiraku Morita, Nuttapong Attrapadung, Tadanori Teruya, Satsuya Ohata, Koji Nuida, and Goichiro Hanaoka. Constant-round client-aided secure comparison protocol. In *European Symposium on Research in Computer Security*, pp. 395–415. Springer, 2018.

[14] Takashi Nishide and Kazuo Ohta. Constant-round multiparty computation for interval test, equality test, and comparison. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 90, No. 5, pp. 960–968, 2007.

[15] Satsuya Ohata and Koji Nuida. Towards high-throughput secure mpc over the internet: Communication-efficient two-party protocols and its application. *arXiv preprint arXiv:1907.03415*, 2019.

[16] Sander Siim. A comprehensive protocol suite for secure two-party computation. *Master's Thesis*, 2016.

[17] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 162–167. IEEE, 1986.