# Privacy Preserving Neural Network

A study of SecureNN: a 3PC framework

Capstone Project

Kuber Shahi, ASP'22

December 19, 2021

**Advisors:** Dr. Mahavir Jhawar and Dr. Debayan Gupta

## Project Overview

## Project Overview

- In this work, we studied privacy preserving neural network (PPNN) frameworks in 2-server (2PC) and 3-server (3PC) settings.
- We looked at SecureML [3] and SecureNN [5], PPNN frameworks in 2PC and 3PC setting respectively.
- First, to understand the building blocks of a neural network (NN), we studied and implemented [4] an NN with ReLU in the hidden layer and Softmax in the output layer, solving a multi-class classification problem.
- Second, we analysed and implemented (some of them) the building blocks of SecureNN, identifying secure comparison a key component of these PPNN frameworks.

## Problem at hand

## Why Privacy Preserving Machine Learning (PPML)?

- Machine Learning (ML) algorithms needs to provide high level of robustness and accuracy which is contingent upon availability of high volume of data.
- But accumulation and sharing of data is not always possible for all companies because of:
    1. data regulations and guidelines which discourages sharing of data
    2. data being shared is proprietary information or sensitive in nature.
    3. high cost of accumulating and maintaining data for ML models' training.
- Hence, there is a need for mechanisms for sharing of data and training ML models between different parties without revealing anything about the data being shared.

## Problem Statement

**Problem Statement:**

$M$ data owners want to collaboratively and securely train an ML model while preserving the privacy of their input data.

- From the viewpoint of cryptography, this is a secure multiparty computation (MPC) problem.
- Given $N$ parties $(p_1, p_2, \ldots p_N)$, each having their private data, $d_1, d_2, \ldots, d_N$, MPC provides a mechanism to securely evaluate a public function (say $F$) on private data, $F(d_1, d_2, \ldots, d_N)$ while keeping the input data private/secret.
- Considering ML training is a series of function evaluation, MPC primitives could be used to train an ML model securely.

## MPC Approach Challenges

- ML model training is already compute-intensive task and the added requirement of preserving the privacy makes it more intensive in terms of computational requirement.
- Hence, it is better to outsource such task to computationally powerful servers. Such MPC-employed scheme of securely training the model using servers is called Secure Outsource Computation (SOC) setting.

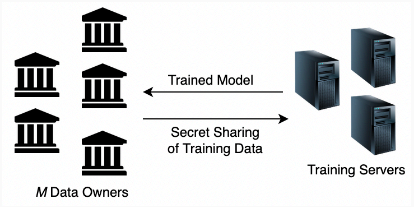


Figure 1: SOC setting

- All state-of-art PPML frameworks use SOC setting typically comprising of 2-4 servers.

## SecureNN in SOC

- SecureML uses 2 servers and SecureNN uses 3 servers, hence called 2PC and 3PC framework respectively.
- For this talk, we are going to focus on SecureNN PPNN framework.

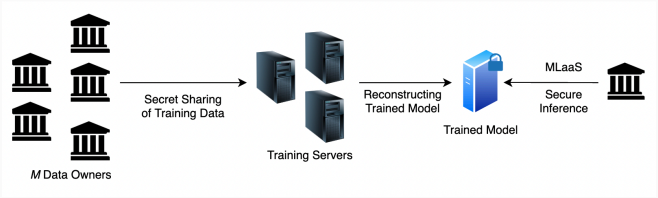


Figure 2: SecureNN SOC Architecture

- In the entire process of training, the privacy of data being shared and final trained model must be preserved.

## Preliminaries

## Mapping from $\mathbb{R}^n$ to $\mathbb{Z}_{2^{64}}$ i

- To preserve privacy during training, MPC primitives are used which work with integer rings whereas real world data are real numbers and hence, ML algorithms preform computation on real number.
- Consequently, we need to come up with a way to map real numbers to integer rings specifically $\mathbb{Z}_{2^l}$.
- SecureML uses a mapping that embeds fixed-point decimal numbers onto integer ring $\mathbb{Z}_{2^l}$

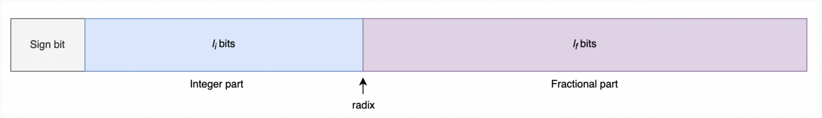


Figure 3: Fixed Point Representation

## Mapping from $\mathbb{R}^n$ to $\mathbb{Z}_{2^{64}}$ ii

**Algorithm 1** Mapping $\prod_{Map}(x)$

```
Input: $x \in \mathbb{R}$
Output: $x' \in \mathbb{Z}_{2^{64}}$ is the mapping of $x$
1: if $x \geq 0$ then
2:     $x' \leftarrow (2^{l_f} \times x)$
3: else
4:     $x' = 2^{64} - |2^{l_f} \times x|$
5: end if
```

- So, positive numbers are mapped to $x' = 2^{l_f}x$ and negative numbers are mapped to $y' = 2^{64} - |2^{l_f}y|$.
- For instance, with precision of $l_f = 13$, 4 will be mapped to $2^{15}$, and $-1$ will be mapped to $2^{64} - 2^{13}$

## Mapping from $\mathbb{R}^n$ to $\mathbb{Z}_{2^{64}}$ iii

- After mapping, the range of positive number that is supported in the ring is from 0 to $2^{63} - 1$ and the negative number is supported from $2^{63}$ to $2^{64} - 1$

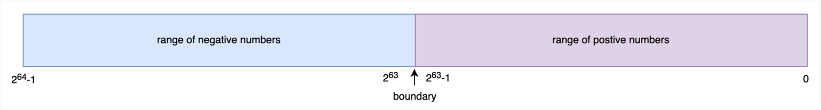


Figure 4: Integer ring $\mathbb{Z}_{2^{64}}$

## Arithmetic Operations on Decimal Numbers

- The addition of two fixed point numbers is straightforward map them and add them.
- But while multiplying two fixed point numbers, the fractional part doubles, and hence the last $l_f$ bits needs to be truncated.

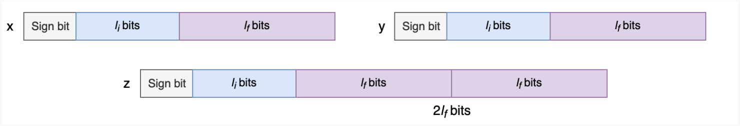


Figure 5: Multiplication doubles the fractional part

- To truncate a number $z \in \mathbb{Z}_{2^{64}}$, we right shift it by $l_f$ bits i.e $z \leftarrow \frac{z}{2^{l_f}}$

## Secret Sharing i

- To preserve the privacy of input data, we split the data into multiple shares, and distribute the shares among the parties, so that only when certain number of parties come together the secret can be reconstructed.
- **2-out-of-2 additive secret sharing in $\mathbb{Z}_L$**
  Two shares of $x$ over $\mathbb{Z}_L$ are denoted as $(\langle x\rangle_0^L, \langle x\rangle_1^L)$
  - Input: $x \in \mathbb{Z}_L$
  - $\mathtt{Share}^L(x)$:
    1. Pick a common random $r \in \mathbb{Z}_L$
    2. Set first share as $\langle x\rangle_0^L$ = r
    3. Set second share as $\langle x\rangle_1^L$ = $(x - r)$ mod $L$
  - Output: $(\langle x\rangle_0^L, \langle x\rangle_1^L)$
  - $\mathtt{Rec}^L(\langle x\rangle_0^L, \langle x\rangle_1^L)$: $x = \langle x\rangle_0^L + \langle x\rangle_1^L \bmod L$

## Secret Sharing ii

- **2-out-of-2 additive secret sharing in $\mathbb{Z}_{L-1}$:** Two shares of $x$ over $\mathbb{Z}_{L-1}$ are denoted as $(\langle x\rangle_0^{L-1}, \langle x\rangle_1^{L-1})$
- **2-out-of-2 additive secret sharing in $\mathbb{Z}_p$:**
  Two shares of $x$ over $\mathbb{Z}_p$ are denoted as $(\{\langle x[i]\rangle_0^p\}_{i\in[l]}, \{\langle x[i]\rangle_1^p\}_{i\in[l]})$ where $l = 64$
  - Input: $x \in \mathbb{Z}_L$ is a 64-bit integer
  - $\mathtt{Share}^p(x)$:

```
1: for $i = \{0, \dots, l\}$ do
2:     Pick a common random $r \in \mathbb{Z}_p$
3:     Set $\langle x[i]\rangle_0^p = r$
4:     Set $\langle x[i]\rangle_1^p = (x[i] - r) \bmod p$
5: end for
```

  - Output: $(\{\langle x[i]\rangle_0^p\}_{i\in[l]}, \{\langle x[i]\rangle_1^p\}_{i\in[l]})$

## Secret Sharing iii

- $Rec^p(\{\langle x[i]\rangle_0^p\}_{i\in[l]}, \{\langle x[i]\rangle_1^p\}_{i\in[l]})$:

```
1: for $i = \{0, \ldots, l\}$ do
2:     $x[i] = (\langle x[i]\rangle_0^p + \langle x[i]\rangle_1^p) \bmod p$
3: end for
```

- So, the data is first mapped and then secret shared in $\mathbb{Z}_{2^{64}}$ between two servers in SecureNN.

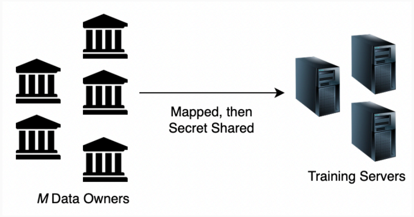


Figure 6: Mapping and Secret Sharing in SecureNN

- Next, the 3 servers collaboratively train on the secretly shared data.

## Neural Network

## NN building blocks i

- SecureNN considers a neural network (NN) with three layers, two hidden layers each consisting of 128 neurons with ReLU applied on them, and an output layer consisting of 10 neurons with Softmax applied on them.
- I implemented a similar NN which has two layers: one hidden layer consisting of 256 neurons with ReLU activation function applied on them, and an output layer consisting of 10 neurons with Softmax applied on them.

## NN building blocks ii

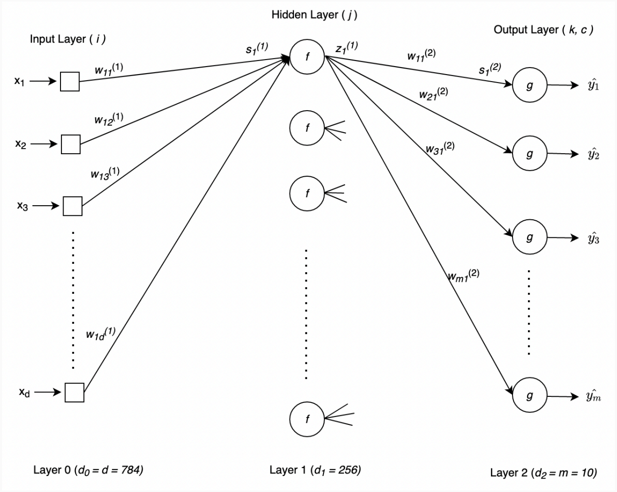


Figure 7: Implemented NN Architecture

## NN building blocks iii

- **ReLU:**

$$y = f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{1}$$

- **Derivative of ReLU** ($ReLU'(a)$)**:**

$$\frac{\delta y}{\delta x} = f'(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2}$$

- In short, **ReLU** can be written as:

$$ReLU(x) = (x > 0) \cdot x = ReLU'(x) \cdot x \tag{3}$$

## NN building blocks iv

- **Softmax:** Given an input vector $x$, softmax activation is a way to normalize the components of the vector. Each component of the vector, $x_i$, is normalized by applying:

$$g(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{4}$$

- The loss function used was cross entropy loss function.
- **Update function:**
  - *Output layer:*

$$w_{kj}(new) := w_{kj}(old) - \alpha[z_j(\hat{y}_k - y_k)] \tag{5}$$

  - *Hidden Layer:*

$$w_{ji}(new) := w_{ji}(old) - \alpha\left(\sum_{k=1}^{m}(\hat{y}_k - y_k) \times w_{kj}\right) \times f'(s_j) \times x_i \tag{6}$$

## NN building blocks v

- **Results:** The NN was trained on MNIST [2]. After 15 epochs of training, the final training loss and accuracy were 0.607362 and 92.0233 respectively %. The testing loss and accuracy were 0.705534 and 91.23 respectively. %.
- **NN building blocks:** In conclusion, for any neural network that solves a multi-class classification problem, the main computation of the NN apart from addition/subtraction involves:
  1. Multiplication
  2. Computation of non-linear activation function such as ReLU and Softmax.
  3. Division

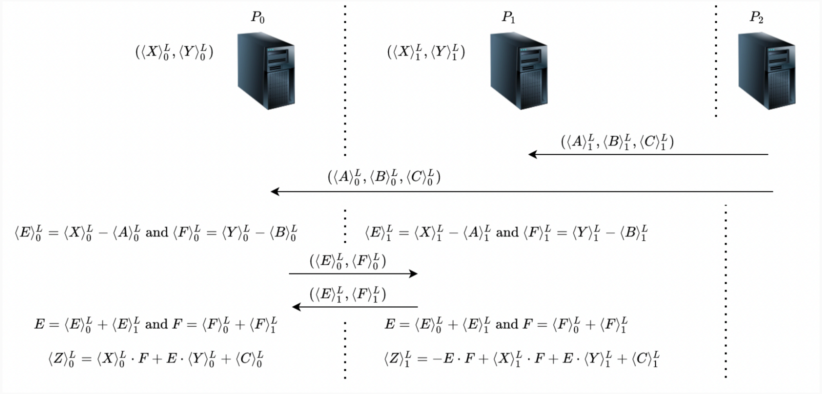
  For privacy preserving neural network (PPNN), these three operation should be computed securely in the SOC setting.

## SecureNN

## Matrix Multiplication ($\mathcal{F}_{\text{MatMult}}$) i

- **Input:** $P_0$ and $P_1$ hold $(\langle X \rangle_0^L, \langle Y \rangle_0^L)$ and $(\langle X \rangle_1^L, \langle Y \rangle_1^L)$ respectively.
  **Output:** $P_0$ gets $(\langle X.Y \rangle_0^L)$ and $P_1$ gets $(\langle X.Y \rangle_1^L)$
- provides the functionality of securely multiplying two numbers (say $x$ and $y$) between two parties $P_0$ and $P_1$ using beaver multiplication triples.
- Beaver Multiplication triples [1] are a tuple of secret-shared values $(\langle a \rangle_j^L, \langle b \rangle_j^L, \rangle c \langle_j^L)$ such that $c = a \times b$, and $a$ and $b$ are uniformly chosen at random.

## Matrix Multiplication ($\mathcal{F}_{\mathsf{MatMult}}$) ii



$P_0$
$P_1$
$P_2$
$(\langle X\rangle_0^L, \langle Y\rangle_0^L)$
$(\langle X\rangle_1^L, \langle Y\rangle_1^L)$
$(\langle A\rangle_1^L, \langle B\rangle_1^L, \langle C\rangle_1^L)$
$(\langle A\rangle_0^L, \langle B\rangle_0^L, \langle C\rangle_0^L)$
$\langle E\rangle_0^L = \langle X\rangle_0^L - \langle A\rangle_0^L$ and $\langle F\rangle_0^L = \langle Y\rangle_0^L - \langle B\rangle_0^L$
$\langle E\rangle_1^L = \langle X\rangle_1^L - \langle A\rangle_1^L$ and $\langle F\rangle_1^L = \langle Y\rangle_1^L - \langle B\rangle_1^L$
$(\langle E\rangle_0^L, \langle F\rangle_0^L)$
$(\langle E\rangle_1^L, \langle F\rangle_1^L)$
$E = \langle E\rangle_0^L + \langle E\rangle_1^L$ and $F = \langle F\rangle_0^L + \langle F\rangle_1^L$
$E = \langle E\rangle_0^L + \langle E\rangle_1^L$ and $F = \langle F\rangle_0^L + \langle F\rangle_1^L$
$\langle Z\rangle_0^L = \langle X\rangle_0^L \cdot F + E \cdot \langle Y\rangle_0^L + \langle C\rangle_0^L$
$\langle Z\rangle_1^L = -E \cdot F + \langle X\rangle_1^L \cdot F + E \cdot \langle Y\rangle_1^L + \langle C\rangle_1^L$

Figure 8: Secure Matrix Multiplication functionality

## Matrix Multiplication ($\mathcal{F}_{\text{MatMult}}$) ii

- If we add both the output shares we get:

$$
\begin{aligned}
&= (\langle Z\rangle_1)^L + (\langle Z\rangle_1)^L \\
&= \langle X\rangle_0^L \cdot F + E \cdot \langle Y\rangle_0^L + \langle C\rangle_0^L - E \cdot F + \langle X\rangle_1^L \cdot F + E \cdot \langle Y\rangle_1^L + \langle C\rangle_1^L \\
&= (\langle X\rangle_0^L + \langle X\rangle_1^L) \cdot F + E \cdot (\langle Y\rangle_0^L + \langle Y\rangle_0^L) + C - E \cdot F \\
&= X \cdot F + E \cdot Y + C - E \cdot F \\
&= X \cdot F + (X - A) \cdot Y + A \cdot B - E \cdot F \\
&= X \cdot F + X \cdot Y - A \cdot Y + A \cdot B - E \cdot F \\
&= X \cdot F + X \cdot Y - A \cdot (Y - B) - E \cdot F \\
&= X \cdot F + X \cdot Y - A \cdot F - E \cdot F \\
&= (X - A) \cdot F + X \cdot Y - E \cdot F \\
&= E \cdot F + X \cdot Y - E \cdot F \\
&= X \cdot Y \\
&= Z
\end{aligned}
$$

## Computation of ReLU i

- **ReLU** can be written as: $ReLU(x) = (x > 0) \cdot x = ReLU'(x) \cdot x$
- **Input:** $P_0$ and $P_1$ hold $\langle a \rangle_0^L$ and $\langle a \rangle_1^L$ respectively.
  **Output:** $P_0$ gets $(\langle ReLU'(a) \rangle_0^L)$ and $P_1$ gets $(\langle ReLU'(a) \rangle_1^L)$
- Computing $ReLU'(a)$ is related to computing $MSB(a)$ in the $Z_{2^{64}}$ as:

$$ReLU'(a) = \begin{cases} 1 & a > 0, MSB(a) = 0 \\ 0 & a \le 0, MSB(a) = 1 \end{cases} \tag{7}$$

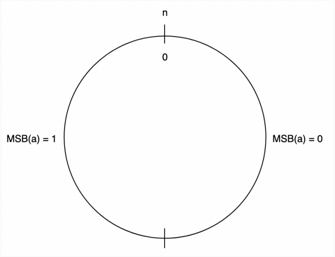


Figure 9: MSB in a ring

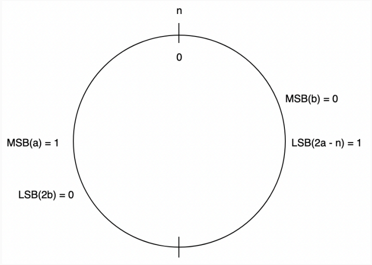


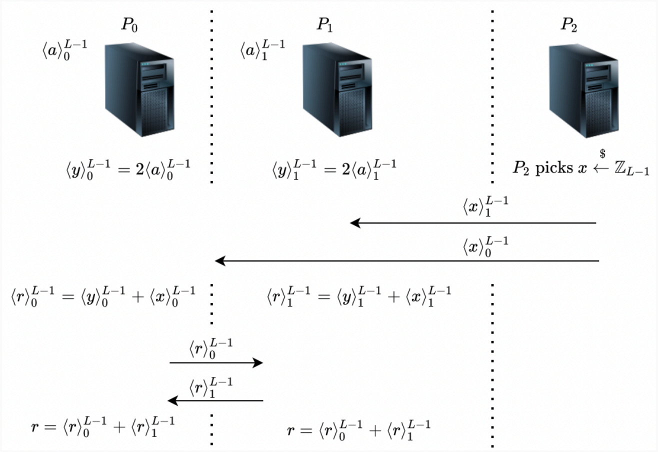
Figure 10: MSB to LSB

## Computation of ReLU ii

- Again, in an odd ring (having odd order n), computation of $MSB(a)$ can be converted to computation $LSB(2a - n)$.
- SecureNN provides a $\mathcal{F}_{ShareConvert}$ functionality to convert the shares of $a$ over $\mathbb{Z}^L$ to over $\mathbb{Z}^{L-1}$
- Using $\mathcal{F}_{ShareConvert}$, the input now becomes:
- **Input:** $P_0$ and $P_1$ hold $\langle a \rangle_0^{L-1}$ and $\langle a \rangle_1^{L-1}$ respectively.

$$ReLU'(a) = \begin{cases} 1 & a > 0, LSB(y = 2a) = 0 \\ 0 & a \leq 0, LSB(y = 2a) = 1 \end{cases} \tag{8}$$

## Compute $LSB(y = 2a)$ i



$P_0$
$P_1$
$P_2$
$\langle a\rangle_0^{L-1}$
$\langle a\rangle_1^{L-1}$
$\langle y\rangle_0^{L-1} = 2\langle a\rangle_0^{L-1}$
$\langle y\rangle_1^{L-1} = 2\langle a\rangle_1^{L-1}$
$P_2$ picks $x \overset{\$}{\leftarrow} \mathbb{Z}_{L-1}$
$\langle x\rangle_1^{L-1}$
$\langle x\rangle_0^{L-1}$
$\langle r\rangle_0^{L-1} = \langle y\rangle_0^{L-1} + \langle x\rangle_0^{L-1}$
$\langle r\rangle_1^{L-1} = \langle y\rangle_1^{L-1} + \langle x\rangle_1^{L-1}$
$\langle r\rangle_0^{L-1}$
$\langle r\rangle_1^{L-1}$
$r = \langle r\rangle_0^{L-1} + \langle r\rangle_1^{L-1}$
$r = \langle r\rangle_0^{L-1} + \langle r\rangle_1^{L-1}$

Figure 11: Computing $LSB(y)$

## Compute $LSB(y = 2a)$ ii

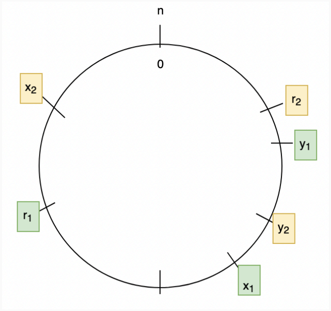


Figure 12: The sum may or may not wrap around the ring

- The sum $r$ either wraps around the ring or it does not.

$$r[0] = \begin{cases} y[0] \oplus x[0] & \text{if } r \text{ doesn't wrap} \\ y[0] \oplus x[0] \oplus 1 & \text{if } r \text{ wraps} \end{cases} \tag{9}$$

## Compute $LSB(y = 2a)$ iii

- To check whether $r$ wraps around the ring or not, it suffices to check whether the final sum is less than one of the individual value - that is (x>r).
- if $\gamma = (x > r)$ denotes the wrap bit, then we can calculate $LSB(y) = y[0]$ as:

$$y[0] = r[0] \oplus x[0] \oplus \gamma \tag{10}$$

- So, at the end,

$$ReLU'(a) = \begin{cases} 1 & a > 0, y[0] = 0 \\ 0 & a \leq 0, y[0] = 1 \end{cases} \tag{11}$$

- SecureML uses Garbled Circuits to extract the $\texttt{MSB}$ whereas using this approach SecureNN gains significant improvement in communication. SecureNN is $7\times$ faster than SecureML in LAN setting.

## Private Compare ($\mathcal{F}_{\text{PC}}$) i

- provides the functionality of comparing two $l$-bit numbers in the integer ring $\mathbb{Z}_L$.
- **Input:** $P_0$ and $P_1$ hold $\{\langle x[i]\rangle_0^p\}_{i\in[l]}, \langle x[i]\rangle_1^p\}_{i\in[l]}$ respectively, and a common $l-bit$ integer $r$. **Output:** $P_2$ gets $\gamma = (x > r)$.
- **Bit-by-bit Comparison:**
  1. Find the ***XOR*** of $x$ and $r$ as $w = x \oplus r$
  2. Compute $c[i] = r[i] - x[i] + 1 + \sum_{k=i+1}^{l} w[i]$
  3. Check for a 0 in $c$. $(x > r)$ iff $\exists\ i \in [l]$, such that $c[i] = 0$.
- Let's take two numbers $x = 141 = 10001101$ and $r = 135 = 10000110$ such that $(x > r)$.

$$\begin{array}{r} 10001101 \\ \oplus 10000110 \\ \hline w = 000001011 \end{array}$$

## Private Compare ($\mathcal{F}_{\text{PC}}$) ii

- Compute $c$

$$\begin{aligned}
c[8] &= (1-1)+1+0 = 1 \\
c[7] &= (0-0)+1+0 = 1 \\
c[6] &= (0-0)+1+0 = 1 \\
c[5] &= (0-0)+1+0 = 1 \\
c[4] &= (0-1)+1+0 = 0 && \text{(point where they first differ)} \\
c[3] &= (1-1)+1+1 = 2 \\
c[2] &= (1-0)+1+1 = 3 \\
c[1] &= (1-0)+1+2 = 3
\end{aligned}$$

## Private Compare ($\mathcal{F}_{\mathrm{PC}}$) iii

- Now, let's take another example where $x < r$, $x = 49 = 110001$ and $r = 53 = 110101$.

$$\begin{array}{r} 110001 \\ \oplus 110101 \\ \hline w = 000100 \end{array}$$

- compute c

$$\begin{aligned}
c[6] &= (1-1)+1+0 = 1 \\
c[5] &= (1-1)+1+0 = 1 \\
c[4] &= (0-0)+1+0 = 1 \\
c[3] &= (1-0)+1+0 = 2 \qquad \text{(point where they first differ)} \\
c[2] &= (0-0)+1+1 = 2 \\
c[1] &= (1-1)+1+1 = 2
\end{aligned}$$

## Private Compare ($\mathcal{F}_{\mathsf{PC}}$) iv

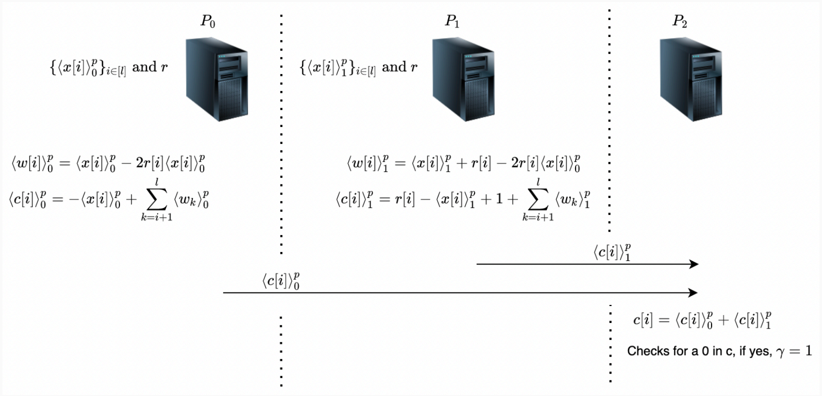

$P_0$
$P_1$
$P_2$
$\{\langle x[i]\rangle_0^p\}_{i\in[l]}$ and $r$
$\{\langle x[i]\rangle_1^p\}_{i\in[l]}$ and $r$
$\langle w[i]\rangle_0^p = \langle x[i]\rangle_0^p - 2r[i]\langle x[i]\rangle_0^p$
$\langle c[i]\rangle_0^p = -\langle x[i]\rangle_0^p + \sum_{k=i+1}^{l} \langle w_k\rangle_0^p$
$\langle w[i]\rangle_1^p = \langle x[i]\rangle_1^p + r[i] - 2r[i]\langle x[i]\rangle_0^p$
$\langle c[i]\rangle_1^p = r[i] - \langle x[i]\rangle_1^p + 1 + \sum_{k=i+1}^{l} \langle w_k\rangle_1^p$
$\langle c[i]\rangle_1^p$
$\langle c[i]\rangle_0^p$
$c[i] = \langle c[i]\rangle_0^p + \langle c[i]\rangle_1^p$
Checks for a 0 in c, if yes, $\gamma = 1$

Figure 13: Private Compare functionality

## Dependency between Protocols

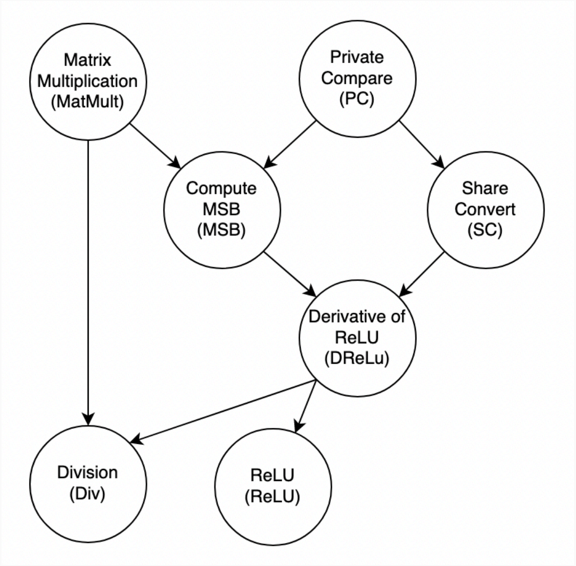


Figure 14: Dependency between protocols

## References i

D. Beaver.
Efficient multiparty protocols using circuit randomization.
In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.
https://doi.org/10.1007/3-540-46766-1_34.

L. Deng.
The mnist database of handwritten digit images for machine learning research [best of the web].
*IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
https://doi.org/10.1109/MSP.2012.2211477.

## References ii

P. Mohassel and Y. Zhang.
SecureML: A system for scalable privacy-preserving machine learning.
In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, 2017.
*https://doi.org/10.1109/SP.2017.12*.

K. Shahi.
Implementation Link.
*https://github.com/kubershahi/ashoka-capstone*.

## References iii

S. Wagh, D. Gupta, and N. Chandran.
SecureNN: 3-party secure computation for neural network training.
In *Proceedings on Privacy Enhancing Technologies*, volume 3, pages 26–49, 2019.
*https://doi.org/10.2478/popets-2019-0035*.