CrypTFlow2: Practical 2-Party Secure Inference

Part 0 Background and Motivation

```
The prediction of neural network inference under 2PC is always a difficult research
problem. The first difficulty is the huge cost of nonlinear computation, such as
comparison and division. The other is the loss of accuracy.

CrypTFlow2 proposes a new protocol for security comparison based on Oblivious
Transfer (OT) and deeply optimizes the protocol. Furthermore, several operator
protocols for neural networks are designed based on the comparison protocol, such as
ReLU, Truncation, faithful Division (divisor is public), Avgpool, and Maxpool.

In addition, two versions of the ring$\ \mathbb{Z}_L(L = 2^l)$ and $\mathbb{Z}_n$ (n
is any large integer) are designed in this paper, so as to adapt the linear layer
computation for OT and Homomorphic Encryption (HE).

Finally, the code for this article has been open sourced, linked to:
[https://github.com/mpc-msri/EzPC/tree/master/SCI]()
```

Part 1 Premliminaries

```
This paper is aimed at the semi-honest adversaries in two-party computation, mainly
using the additive secret sharing under two parties, OT, OT constructed AND triples,
Multiplexer, and B2A transformation, and HE, where secret sharing is already
familiar, and HE is mainly used in the linear layer of neural networks. The core
construction of this article is based on OT, and we mainly review the knowledge of
OT here.
```

Notation :

- $$w \leftarrow^s W$$

  For a set W, w is an element randomly selected from W.

- $$[l]$$

  denotes the set of integers $0, .., l-1$

- $$1\{b\}$$

  denote the indicator function that is 1 when ? is true and 0 when ? is false.

1.1 Oblivious Transfer

```
OT is very clear in terms of its function. The standard definition of 1-out-of-2 OT
involves two parties: the sender S holds two bits of information, and the receiver R
holds a selection bit. After the OT protocol is executed,R can get the information
of the index corresponding to the selection bit, but cannot get other information,
and S does not know which information R selects. It can be generalized to 1-out-of-n
OT, that is, the receiver needs to secretly obtain a certain information from the n
messages of the sender. Similarly, it is further extended to k-out-of-n OT, that is,
out of n messages from the sender, the receiver secretly obtains k messages.
```

1730708346728

```
In the OT protocol, the sender has all the data rights, the receiver has the option
of a single data, the data exchange is completed inadvertently, while ensuring the
privacy of the private data of both sides.
```

1.2 OT-based MUX and B2A

F_{MUX}:

- input: $\langle a\rangle^n$ and $\langle c\rangle^B$
- output: if c = 1, output $\langle a\rangle^n$; otherwise, output shares over 0

Based on $\binom{2}{1}-OT_\eta$, $x_0 = -r_0 + c \cdot \langle a\rangle_0^n,\ \ x_1 = -r_1 + c \cdot \langle a\rangle_1^n$. Therefore, $z = z_0 + z_1 = c \cdot a$. Since OT is used twice, the communication expense is $2(\lambda + 2\eta)$.

Algorithm 6 Multiplexer, Π^{n}_{MUX}:

Input: For $b \in \{0,1\}$, P_b holds $\langle a\rangle^n_b$ and $\langle c\rangle^B_b$.
Output: For $b \in \{0,1\}$, P_b learns $\langle z\rangle^n_b$ s.t. $z = a$ if $c = 1$, else $z = 0$.

1: For $b \in \{0,1\}$, P_b picks $r_b \xleftarrow{\$} \mathbb{Z}_n$.
2: P_0 sets s_0, s_1 as follows: If $\langle c\rangle^B_0 = 0$, $(s_0, s_1) = (-r_0, -r_0 + \langle a\rangle^n_0)$. Else, $(s_0, s_1) = (-r_0 + \langle a\rangle^n_0, -r_0)$.
3: P_0 & P_1 invoke an instance of $\binom{2}{1}$-OT_η where P_0 is the sender with inputs (s_0, s_1) and P_1 is the receiver with input $\langle c\rangle^B_1$. Let P_1's output be x_1.
4: P_1 sets t_0, t_1 as follows: If $\langle c\rangle^B_1 = 0$, $(t_0, t_1) = (-r_1, -r_1 + \langle a\rangle^n_1)$. Else, $(t_0, t_1) = (-r_1 + \langle a\rangle^n_1, -r_1)$.
5: P_0 & P_1 invoke an instance of $\binom{2}{1}$-OT_η where P_1 is the sender with inputs (t_0, t_1) and P_0 is the receiver with input $\langle c\rangle^B_0$. Let P_0's output be x_0.
6: For $b \in \{0,1\}$, P_b outputs $\langle z\rangle^n_b = r_b + x_b$.

F_{B2A}:

- Input: $\langle c\rangle^B$;
- output: $\langle d\rangle^n$, which satisfies $d = c$.

Since $d = \langle c\rangle^B_0 + \langle c\rangle^B_1 - 2\langle c\rangle^B_0 \cdot \langle c\rangle^B_1$, the key is on the product term. Based on $\binom{2}{1} - OT_\eta$, $y_1 = x + \langle c\rangle^B_0 \cdot \langle c\rangle^B_1$. Therefore, $\langle d\rangle^n_0 = \langle c\rangle^B_0 + 2x$. Since OT is used twice, the communication expense is $\lambda + \eta$.

Algorithm 7 Boolean to Arithmetic, Π^n_{B2A}:

Input: P_0, P_1 hold $\langle c\rangle^B_0$ and $\langle c\rangle^B_1$, respectively, where $c \in \{0,1\}$.
Output: P_0, P_1 learn $\langle d\rangle^n_0$ and $\langle d\rangle^n_1$, respectively, s.t. $d = c$.

1: P_0 & P_1 invoke an instance of $\binom{2}{1}$-COT$_\eta$ where P_0 is the sender with correlation function $f(x) = x + \langle c\rangle^B_0$ and P_1 is the receiver with input $\langle c\rangle^B_1$. Party P_0 learns x and sets $y_0 = n - x$ and P_1 learns y_1.
2: For $b \in \{0,1\}$, P_b computes $\langle d\rangle^n_b = \langle c\rangle^B_b - 2 \cdot y_b$.

Part 3 Millionaries' and DReLu Protocols (non-linear)

Millionaires' protocols

General idea

```
Input :$ P_0$ has $x\in \{0,1\}^l$ and $ P_1$ has $y\in \{0,1\}^l$

output :$P_0, P_1$ learn shares of $\{x>y\}$
```

for $x = x_1 \mid\mid x_0, \quad y = y_1 \mid\mid y_0$:

$$\{x>y\} = \{x_1 < y_1\} \oplus (\{x_1 = y_1\} \wedge \{x_0 < y_0\})$$

$$\{x=y\} = (\{x_1 = y_1\} \wedge \{x_0 = y_0\})$$

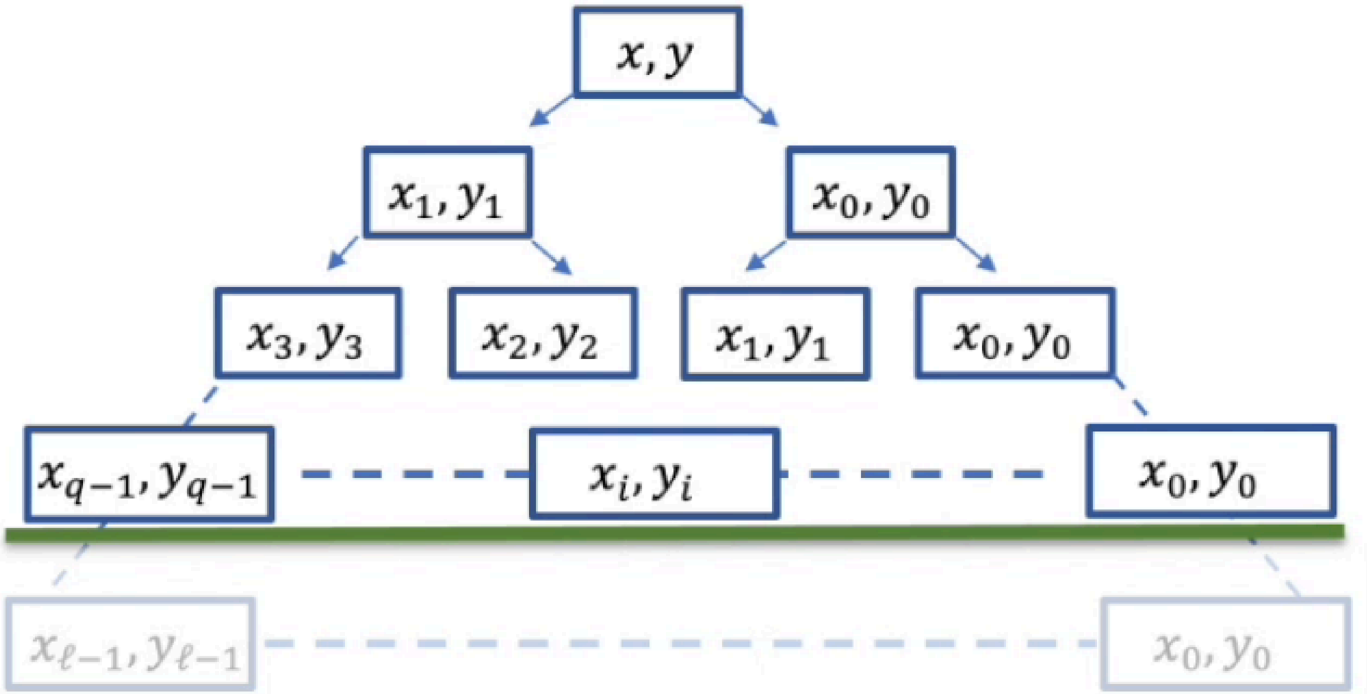
> In GSV07, we conduct this comparison until the single bit-leaves

In CRTF2, however, we stop at $q = l/m$ leaves; $x_i, y_i \in \{0,1\}^m$ and conduct$2 \times \binom{M}{1} - OT_1$ on each leaf; $M = 2^m$

```
This change is efficient only for small bitlengths, such as$\ m=4$
```

Communications for $m = 4$: $6\lambda l \rightarrow 2\lambda l$

After more optimizations: $2\lambda l \rightarrow \lambda l$



Protocol

```
As shown in the picture below. Let$\ M = 2^m$ and consider a simple case $q = l/m$,
where q is to the power of 2. By recursing $log_2^q$ times, we obtain a tree with q
leaves. Each leaf has m bits, i.e. $x = x_{q-1}||...||x_0$ and $y = y_{q-
1}||...||y_0$, where $x_i,y_i\in\{0,1\}^m$. By doing so, both parties can take
advantage of $\begin{pmatrix}M\\1\end{pmatrix}-OT$ and use its comparision function
and equivalence function. After calculating those leaves, we can calculate AND and
XOR recursively to the root node in order to get the final output.
```

1730122971317

��þ��Э������£���������Э��ľ�������ڦ� ���䘬P_0, P_1 ����������x,y,�������������lλ�Ķ�����y�

�����P_0, P_1 ���õ���Ꮭ��������������h��������λ��

��h�����P_0�������x���q����ǐ����P_1 ��������y���q����ǘ����ÿ����ǘ���Ĵ�C���m��

�ڳ��������M=2^m �������������j=0���q-1�����㓥����� ���Ĳ����P_0 ���w������������λ�������������λ������J̌õ��������� ������ġ�

���崇������k=0��M-1㶊��. ��������P_0���÷��,�$s_{j,k}$ ��s���J̌õ���`$x_j < k$��..

��□□���P_0���÷��,�$t_{j,k}$ ��t���J̌õ���`$x_j = k$��..

�گ¸�������k��㶊�� �ھŲ���P_0, P_1����1-out-of-M OTЭ�飬 ����M������$s_{j,k}$��w��h����P_0 ��Ï���и����������M������$s_{j,k}$��P_1 ��Ï���ƨ����������y_j

��ɥ����P_0, P_1����1-out-of-M OTЭ�飬����M������$t_{j,k}$ ��w��h����P_0��Ï���и����������M������$t_{j,k}$��P_1 ��Ï���ƨ����������y_j

��ɥh��������j��㶊�� ��ɥ������������i����1��l $log_2 q$ ��ɥ�����������j����0��$\frac{q}{2^l} - 1$ ��ɥ�IJ��������`�T���`���� ��ɥ�崇 �����ڤ�����������J����'���Ï����ɱ����й���������܇ ����Ŀƴ��ҹ����Ա��ڮ��龍 ��ɥ������������`�,�`���룬 ������h���`���ж�ӯ���ʱ����λ�Í���������DZŁЄĽ���� ����x��y������λ�� ��ɥ�ͅ�������j��㶊�� ��ɥ�¸�������i��㶊�� ��ɥ�Ų����������Ľ����

See the appendix for a detailed code-reading

Security Proof

The author simply deduced that the resulting output could equal the result of the comparison. Security follows OT

and F_{AND}'s Security of hybrid protocols

Generalization

1. In the case that m does not divide l, for l mod m=r, which is the remainder of l divided by m, run it separately
2. If q is not a power of 2, consider $2^a < q \leq 2_{a+1}$, So let's take 2^a part as a fragment to build a perfect binary tree. Finally, compare the value of the perfect binary tree with the rest of the comparison.

Optimization

1. In steps 9 and 10, OT is used to combine two 1-bit $\binom{M}{1}-OT$ into a 2-bit $\binom{M}{1}-OT$. Because P_1's input is still y_j
2. In steps 14 AND 16, we use AND. Use related AND, that is, the AND protocol with the same input, to save expenses.
3. For the entire binary tree, the least significant fork of the equation is never used, so it can be omitted.

Communication

Layer	Protocol	Comm. (bits)	Rounds
Millionaires' on $\{0,1\}^\ell$	GC [62, 63]	$4\lambda\ell$	2
	GMW[4]/GSV [29, 32]	$\approx 6\lambda\ell$	$\log\ell+3$
	SC3[5][21]	$> 3\lambda\ell$	$\approx 4\log^*\lambda$
	This work ($m=4$)	$< \lambda\ell + 14\ell$	$\log\ell$
Millionaires' example $\ell=32$	GC [62, 63]	16384	2
	GMW/GSV [29, 32]	23140	8
	SC3 [21]	13016	15
	This work ($m=7$)	2930	5
	This work ($m=4$)	3844	5

Protocol for DReLU

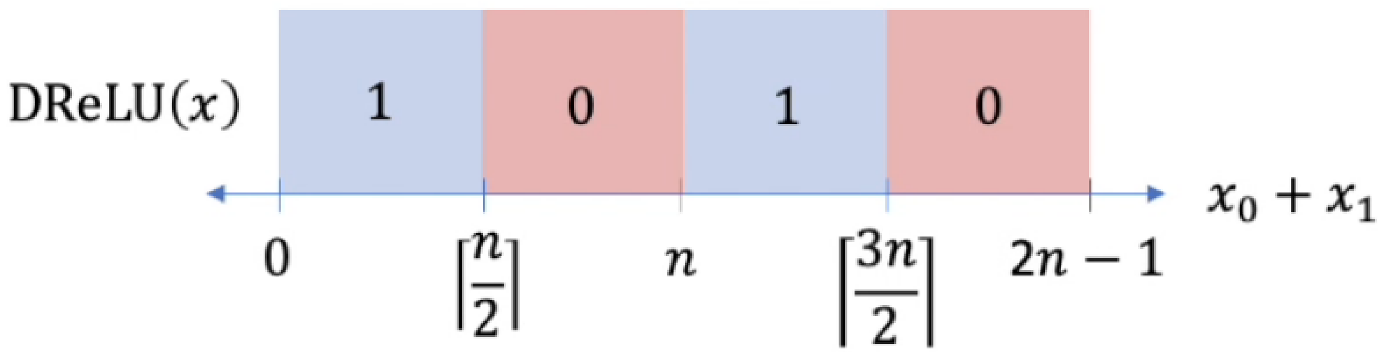
General idea

```
Over$\ Z$, $DReLU(x) = \begin{cases}0,\ x<0\\1,\ x\geq0\end{cases}$

Over$\ Z_n$, $DReLU(x) = \begin{cases}0,\ x\in[\lceil\frac{n}{2}\rceil,n)\\1,\
x\in[0,\lceil\frac{n}{2}\rceil)\end{cases}$
```

Input: P_0 has $x_0 \in [0,n)$ and P_1 has $x_1 \in [0,n)$; s.t. $x_0+x_1 = x \bmod n$

Output: P_0, P_1 learn shares of $DReLu(x)$



Computing DReLU(x):

- Simple: 3 calls to Millionaires'
- Optimized: 2 comparisons
- In particular: for $n = 2^l$, only needs 1 comparison

$$lt : x_0 + x_1 \geq \lceil \frac{n}{2} \rceil \Leftrightarrow x_0 + \lfloor \frac{n}{2} \rfloor > n - 1 - x_1$$

$$wrap : x_0 + x_1 \geq n \Leftrightarrow x_0 > n - 1 - x_1$$

$$rt : x_0 + x_1 \geq \lceil \frac{3n}{2} \rceil \Leftrightarrow x_0 + \lfloor \frac{n}{2} \rfloor > 2n - 1 - x_1$$

Protocol

Over Z, $x_0, x_1 \in \{2^l\}$

We can use 2's complement to encode a_0, a_1, where $a = a_0 \oplus a_1$

Algorithm 2 ℓ-bit integer DReLU, $\Pi_{\mathsf{DReLU}}^{\mathsf{int},\ell}$:

Input: P_0, P_1 hold $\langle a \rangle_0^L$ and $\langle a \rangle_1^L$, respectively.
Output: P_0, P_1 get $\langle \mathsf{DReLU}(a) \rangle_0^B$ and $\langle \mathsf{DReLU}(a) \rangle_1^B$.

1: P_0 parses its input as $\langle a \rangle_0^L = \mathsf{msb}_0 || x_0$ and P_1 parses its input as $\langle a \rangle_1^L = \mathsf{msb}_1 || x_1$, s.t. $b \in \{0,1\}$, $\mathsf{msb}_b \in \{0,1\}$, $x_b \in \{0,1\}^{\ell-1}$.
2: P_0 & P_1 invoke an instance of $\mathcal{F}_{\mathsf{MILL}}^{\ell-1}$, where P_0's input is $2^{\ell-1} - 1 - x_0$ and P_1's input is x_1. For $b \in \{0,1\}$, P_b learns $\langle \mathsf{carry} \rangle_b^B$.
3: For $b \in \{0,1\}$, P_b sets $\langle \mathsf{DReLU} \rangle_b^B = \mathsf{msb}_b \oplus \langle \mathsf{carry} \rangle_b^B \oplus b$.

https://blog.csdn.net/m0_3790841

It is obvious that the result of DReLU can be deduced by the MSB of a. (the result is the inversion of the MSB for a)

```
Step1:  we can divide a0 and a1 into 2 parts

Step2: online, we only compare$2^{l-1} - 1 - x0$ and $x_1$ by Millionares'

Step3: offline, we generate$[DReLU]_b = msb_b\oplus[carry]_b\oplus b$
```

The last b is used as an inversion of the MSB for a.

Over Z_n,

Algorithm 3 Simple Integer ring DReLU, $\Pi^{\mathrm{ring},n}_{\mathrm{DReLU}^{\mathrm{simple}}}$:

Input: P_0, P_1 hold $\langle a\rangle^n_0$ and $\langle a\rangle^n_1$, respectively, where $a \in \mathbb{Z}_n$.
Output: P_0, P_1 get $\langle \mathrm{DReLU}(a)\rangle^B_0$ and $\langle \mathrm{DReLU}(a)\rangle^B_1$.

1: P_0 & P_1 invoke an instance of $\mathcal{F}^{\eta}_{\mathrm{MILL}}$ with $\eta = \lceil \log n \rceil$, where P_0's input is $(n-1-\langle a\rangle^n_0)$ and P_1's input is $\langle a\rangle^n_1$. For $b \in \{0,1\}$, P_b learns $\langle \mathrm{wrap}\rangle^B_b$ as output.
2: P_0 & P_1 invoke an instance of $\mathcal{F}^{\eta+1}_{\mathrm{MILL}}$, where P_0's input is $(n-1-\langle a\rangle^n_0)$ and P_1's input is $((n-1)/2+\langle a\rangle^n_1)$. For $b \in \{0,1\}$, P_b learns $\langle \mathrm{lt}\rangle^B_b$ as output.
3: P_0 & P_1 invoke an instance of $\mathcal{F}^{\eta+1}_{\mathrm{MILL}}$, where P_0's input is $(n+(n-1)/2-\langle a\rangle^n_0)$ and P_1's input is $\langle a\rangle^n_1$. For $b \in \{0,1\}$, P_b learns $\langle \mathrm{rt}\rangle^B_b$ as output.
4: For $b \in \{0,1\}$, P_b invokes $\mathcal{F}^2_{\mathrm{MUX}}$ with input $(\langle \mathrm{lt}\rangle^B_b \oplus \langle \mathrm{rt}\rangle^B_b)$ and choice $\langle \mathrm{wrap}\rangle^B_b$ to learn $\langle z\rangle^B_b$.
5: For $b \in \{0,1\}$, P_b outputs $\langle z\rangle^B_b \oplus \langle \mathrm{lt}\rangle^B_b \oplus b$.

https://blog.csdn.net/m0_379084

In the first 3 steps, we obtain the share of lt, wrap, and rt

```
Step 4: if wrap = 1, z = lt XOR rt; else, z = 0, we can interpret as, [0,0,1,0]

Step 5: Pb = z XOR lt XOR b , we can interpret as, [1,0,1,0]
```

1730120881231

Optimization

1. First let P_1 adjust the input of step 1, 2, 3 to be consistent, so that the leaf nodes in F_{Mill} can be calculated together (step 9 & 10 algorithm 1). The specific method is to add $\frac{\eta-1}{2}$, to P_1 's input of step 1&3 in algorithm 3;
2. Further, the P_0 reduces the execution of step 2 or step 3 according to its input. That is, if $\langle a\rangle_0^\eta > \frac{\eta-1}{2}$, then $it = 1$ in step 2; Otherwise, $rt = 0$ in step 3.

   Correctness can be easily deduced if we take the input ofP_1 in step14 into the OT protocol from step 5 to step 12.

   Since$\langle z\rangle_0^B$ is completely random, the safety can be proved by $(F_{Mill}^{\eta+1}, \binom{4}{1} - OT)$-hybird model.

   Algorithm 4 calls $F_{Mill}^{\eta+1}$ twice and $\binom{4}{1} - OT$ once. Total communication < $\frac{3}{2}\lambda(\eta+1) + 28(\eta+1) + 2\lambda + 4$, which is better than algorithm 3×.

Algorithm 4 Optimized Integer ring DReLU, $\Pi_{\mathsf{DReLU}}^{\mathsf{ring},n}$:

Input: P_0, P_1 hold $\langle a\rangle_0^n$ and $\langle a\rangle_1^n$, respectively, where $a \in \mathbb{Z}_n$. Let $\eta = \lceil \log n \rceil$.

Output: P_0, P_1 get $\langle \mathsf{DReLU}(a)\rangle_0^B$ and $\langle \mathsf{DReLU}(a)\rangle_1^B$.

P_0 & P_1 invoke an instance of $\mathcal{F}_{\mathsf{MILL}}^{\eta+1}$, where P_0's input is $(3(n-1)/2 - \langle a\rangle_0^n)$ and P_1's input is $(n-1)/2 + \langle a\rangle_1^n$. For $b \in \{0, 1\}$, P_b learns $\langle \mathsf{wrap}\rangle_b^B$ as output.
P_0 sets $x = (2n - 1 - \langle a\rangle_0^n)$ if $\langle a\rangle_0^n > (n-1)/2$, else $x = (n - 1 - \langle a\rangle_0^n)$.
P_0 & P_1 invoke an instance of $\mathcal{F}_{\mathsf{MILL}}^{\eta+1}$, where P_0's input is x and P_1's input is $((n-1)/2 + \langle a\rangle_1^n)$. For $b \in \{0, 1\}$, P_b learns $\langle \mathsf{xt}\rangle_b^B$ as output.
P_0 samples $\langle z\rangle_0^B \xleftarrow{\$} \{0, 1\}$.
for $j = \{00, 01, 10, 11\}$ **do**
    P_0 parses j as $j_0||j_1$ and sets $t_j = 1 \oplus \langle \mathsf{xt}\rangle_0^B \oplus j_0$.
    if $\langle a\rangle_0^n > (n-1)/2$ **then**
        P_0 sets $s'_j = t_j \wedge (\langle \mathsf{wrap}\rangle_0^B \oplus j_1)$.
    else
        P_0 sets $s'_j = t_j \oplus ((1 \oplus t_j) \wedge (\langle \mathsf{wrap}\rangle_0^B \oplus j_1))$
    end if
    P_0 sets $s_j = s'_j \oplus \langle z\rangle_0^B$
end for
P_0 & P_1 invoke an instance of $\binom{4}{1}$-OT_1 where P_0 is the sender with inputs $\{s_j\}_j$ and P_1 is the receiver with input $\langle \mathsf{xt}\rangle_1^B || \langle \mathsf{wrap}\rangle_1^B$. P_1 sets its output as $\langle z\rangle_1^B$.
For $b \in \{0, 1\}$, P_b outputs $\langle z\rangle_b^B$.

https://blog.csdn.net/m0_37908414

ReLU

Relu = DRelu * a. We can use DRelu's and Mux's protocol to caculate Relu

Algorithm 8 ℓ-bit integer ReLU, $\Pi_{\text{ReLU}}^{\text{int},\ell}$:

Input: P_0, P_1 hold $\langle a\rangle_0^L$ and $\langle a\rangle_1^L$, respectively.
Output: P_0, P_1 get $\langle \text{ReLU}(a)\rangle_0^L$ and $\langle \text{ReLU}(a)\rangle_1^L$.

1: For $b \in \{0,1\}$, P_b invokes $\mathcal{F}_{\text{DReLU}}^{\text{int},\ell}$ with input $\langle a\rangle_b^L$ to learn output $\langle y\rangle_b^B$.
2: For $b \in \{0,1\}$, P_b invokes $\mathcal{F}_{\text{MUX}}^{L}$ with inputs $\langle a\rangle_b^L$ and $\langle y\rangle_b^B$ to learn $\langle z\rangle_b^L$ and sets $\langle \text{ReLU}(a)\rangle_b^L = \langle z\rangle_b^L$.

https://blog.csdn.net/m0_37908 41

Maxpool layer

A pairwise comparison is used to select the largest method. Choose the two largest numbers first, and then compare them in turn.

For x,y are both at P0,P1 as the number of secret shares, first locally calculate the respective secret share x-y=w. Input the two secret shared w into the DRelu protocol to see if w is greater than 0, and the result is set to v. Using a data selector, t=wv. Finally each output z=y+t.

Consider that when v=0, t=0, the output z=y.

Consider v=1, t=w=x-y The output z=y+x-y=x.

Part 4 Division and Truncation (linear)

4.1 Expressing general division and truncation using arithmetic over secret shares

```
Define$\ idiv:\mathbb{Z}\times\mathbb{Z}\rightarrow\mathbb{Z}$ as signed integer
division, the quotient is prone to $-\infty$, the remainder have the same sign as
divisor. Furthermore, define:
```

$$rdiv(a,d) = idiv(a_u - 1\{a_u \geq \lceil n/2 \rceil\} \cdot n, d)\ mod\ n,$$

```
Here$\ a_u\in\{0,...,n-1\}$ is the unsigned exhibition for $a\in\mathbb{Z}_n$,
$0<d<n$.
```

Theorem 4.1.1

```
Let$\ a\in\mathbb{Z}_n$ has its share as $\lang a\rang^n_0,\lang a\rang^n_1\in
\mathbb{Z}_n$, $n = n^1\cdot d+n^0$. ($n^0,n^1,d\in\mathbb{Z}_n$ and $0\leq
a_0^0,a_1^0<d)$.  Denote $n' = \lceil \frac{n}{2}\rceil\in\mathbb{Z}$. Define
$corr,\ A,\ B,\ C$ as follows:
```

$$corr = \begin{cases} -1, & (a_u \geq n') \wedge (a_0 < n') \wedge (a_1 < n') \\ 1, & (a_u < n') \wedge (a_0 \geq n') \wedge (a_1 \geq n') \\ 0, & otherwise \end{cases}$$

$$A = a_0^0 + a_1^0 - (1\{a_0 \geq n'\} + 1\{a_1 \geq n'\}) \cdot n^0$$

$$B = idiv(a_0^0 - 1\{a_0 \geq n'\} \cdot n', d) + idiv(a_1^0 - 1\{a_1 \geq n'\} \cdot n', d)$$

$$C = 1\{A < d\} + 1\{A < 0\} + 1\{A < -d\}.$$

Therefore we have:

$$rdiv(\langle a\rangle_0^n, d) + rdiv(\langle a\rangle_1^n, d) + (corr \cdot n^1 + 1 - C - B) =_n rdiv(a, d)$$

Proof

- decompose $rdiv(\langle a\rangle_i^n, d)$;
$$\begin{aligned} \mathsf{rdiv}(\langle a\rangle_i^n, d) &=_n \mathsf{idiv}(a_i - 1\{a_i \geq n'\} \cdot n, d) \\ &=_n \mathsf{idiv}(a_i^1 \cdot + a_i^0 - 1\{a_i \geq n'\} \cdot (n^1 \cdot d + n^0), d) \\ &=_n a_i^1 - 1\{a_i \geq n'\} \cdot n^1 + \mathsf{idiv}(a_i^0 - 1\{a_i \geq n'\} \cdot n^0, d). \end{aligned}$$

- $a_u = a_0 + a_1 - w \cdot n$, where $w = 1\{a_0 + a_1 \geq n\}$;
$$\begin{aligned} a_u &= a_0 + a_1 - w \cdot n \\ &= (a_0^1 + a_1^1 - w \cdot n^1) \cdot d(a_0^0 + a_1^0 - w \cdot n^0) \\ &= (a_0^1 + a_1^1 - w \cdot n^1 + k) \cdot d(a_0^0 + a_1^0 - w \cdot n^0 - k \cdot d), \end{aligned}$$

- Since $\leq a_0^0 + a_1^0 - w \cdot n^0 - k \cdot d < d$��

$$\begin{aligned}\mathsf{rdiv}(a,d) &=_n a_0^1 + a_1^1 - w \cdot n^1 + k - 1\{a \geq n'\} \cdot n^1 \\ &+ \mathsf{idiv}(a_0^0 + a_1^0 - w \cdot n^0 - k \cdot d - 1\{a \geq n'\} \cdot n^0, d) \\ &=_n a_0^1 + a_1^1 - w \cdot n^1 - 1\{a \geq n'\} \cdot n^1 \\ &+ \mathsf{idiv}(a_0^0 + a_1^0 - w \cdot n^0 - 1\{a \geq n'\} \cdot n^0, d).\end{aligned}$$

- Then we have:

$$\begin{aligned}c &=_n \mathsf{rdiv}(a,d) - \mathsf{rdiv}(\langle a\rangle_0^n, d) - \mathsf{rdiv}(\langle a\rangle_1^n, d) \\ &= (1\{a_0 \geq n'\} + 1\{a_1 \geq n'\} - w - 1\{a \geq n'\}) \cdot n^1 \\ &+ \mathsf{idiv}(a_0^0 + a_1^0 - w \cdot n^0 - 1\{a \geq n'\} \cdot n^0, d) \\ &- (\mathsf{idiv}(a_0^0 - 1\{a_0 \geq n'\} \cdot n^0, d) + \mathsf{idiv}(a_1^0 - 1\{a_1 \geq n'\} \cdot n^0, d)) \\ &=_n c^1 \cdot n^1 + c^0 - B.\end{aligned}$$

CSDN @

- Let $A_i' = idiv(a_0^0, a_1^0 - i \cdot n^0, d)$;

#	$1\{a_0 \geq n'\}$	$1\{a_1 \geq n'\}$	$1\{a_u \geq n'\}$	w	c^1	c^0
1	0	0	0	0	0	A_0'
2	0	0	1	0	-1	A_1'
3	0	1	0	1	0	A_1'
4	0	1	1	0	0	A_1'
5	1	0	0	1	0	A_1'
6	1	0	1	0	0	A_1'
7	1	1	0	1	1	A_1'
8	1	1	1	1	0	A_2'

Table 8: Truth table for the correction terms c^0 and c^1 in the proof of division theorem (Appendix C).

From this table we know that $c^1 = corr$. Therefore $c =_n corr \cdot n^1 + c^0 - B$.

Let $C_0 = 1\{A < d\}$, $C_1 = 1\{A < 0\}$, $C_0 = 1\{A < -d\}$, then $C = C_0 + C_1 + C_2$

Row 1 corresponds to$A = a_0^0 + a_1^0$;

Row 8 corresponds to$A = a_0^0 + a_1^0 - 2 \cdot n^0$;

Other rows correspond to$c^0 = idiv(A, d)$;

It is obvious that $-2 \cdot d + 2 \leq A \leq 2 \cdot d - 2$, therefore $c^0 \in \{-2, -1, 0, 1\}$

Therefore, $c =_n corr \cdot n^1 + (1 - C) - B$

Corollary 4.2: truncation for $l-bit$ integers can be simplified to:

$$(a_0 >> s) + (a_1 >> s) + corr \cdot 2^{l-s} + 1\{a_0^0 + a_1^0 \geq 2^s\} =_L (a >> s)$$

4.2 Truncation of $l-bit$ Integer

```
$F_{Trunc}^{int,l,s}$ is a function that performs on$\ l-bit$ integers and truncate
its lower s bits. The result is exactly the same as cleartext.

In the algorithm, step 1-15 calculate$\ corr$. Step 16 calculate $1\{a_0^0 +
a_1^0\geq2^s\}$, and conduct $F_{B2A}$ at step 17.

The complicated part is the verification of$\ corr$, especially the construction of
$OT$ in step 15. By taking $P_1$'s input $\lang m\rang_1^m||x_1$ into step 15, we
can verify its correctness. Since the calculation of $\ corr $ is completely random,
the saftey can be proved by $(F_{DReLU}^{int,l}, \begin{pmatrix}4\\1\end{pmatrix}-
OT, F_{Mill}^{s}, F_{B2A}^L)$-hybird model.

Since$\ (F_{DReLU}^{int,l}, \begin{pmatrix}4\\1\end{pmatrix}-OT, F_{Mill}^{s},
F_{B2A}^L)$ are each conducted once, the communicationis less than $\lambda l +
2\lambda + 19l + communication\ for\ F_{Mill}^s$, which is mainly based on s.
```

Algorithm 5 Truncation, $\Pi_{\mathsf{Trunc}}^{\mathsf{int},\ell,s}$:

Input: For $b \in \{0,1\}$, P_b holds $\langle a \rangle_b^L$, where $a \in \mathbb{Z}_L$.

Output: For $b \in \{0,1\}$, P_b learns $\langle z \rangle_b^L$ s.t. $z = a \gg s$.

For $b \in \{0,1\}$, let $a_b, a_b^0, a_b^1 \in \mathbb{Z}$ be as defined in Corollary 4.2.

For $b \in \{0,1\}$, P_b invokes $\mathcal{F}_{\mathsf{DReLU}}^{\mathsf{int},\ell}$ with input $\langle a \rangle_b^L$ to learn output $\langle \alpha \rangle_b^B$. Party P_b sets $\langle m \rangle_b^B = \langle \alpha \rangle_b^B \oplus b$.

For $b \in \{0,1\}$, P_b sets $x_b = \mathsf{MSB}(\langle a \rangle_b^L)$.

P_0 samples $\langle \mathrm{corr} \rangle_0^L \xleftarrow{\$} \mathbb{Z}_{2^\ell}$.

for $j = \{00, 01, 10, 11\}$ **do**

  P_0 computes $t_j = (\langle m \rangle_0^B \oplus j_0 \oplus x_0) \wedge (\langle m \rangle_0^B \oplus j_0 \oplus j_1)$ s.t. $j = (j_0 || j_1)$.

  if $t_j \wedge \mathbf{1}\{x_0 = 0\}$ **then**

    P_0 sets $s_j =_L -\langle \mathrm{corr} \rangle_0^L - 1$.

  else if $t_j \wedge \mathbf{1}\{x_0 = 1\}$ **then**

    P_0 sets $s_j =_L -\langle \mathrm{corr} \rangle_0^L + 1$.

  else

    P_0 sets $s_j =_L -\langle \mathrm{corr} \rangle_0^L$.

  end if

end for

P_0 & P_1 invoke an instance of $\binom{4}{1}$-OT_ℓ, where P_0 is the sender with inputs $\{s_j\}_j$ and P_1 is the receiver with input $\langle m \rangle_1^B || x_1$ and learns $\langle \mathrm{corr} \rangle_1^L$.

P_0 & P_1 invoke an instance of $\mathcal{F}_{\mathsf{MILL}}^{s}$ with P_0's input as $2^s - 1 - a_0^0$ and P_1's input as a_1^0. For $b \in \{0,1\}$, P_b learns $\langle c \rangle_b^B$.

For $b \in \{0,1\}$, P_b invokes an instance of $\mathcal{F}_{\mathsf{B2A}}^{L}$ ($L = 2^\ell$) with input $\langle c \rangle_b^B$ and learns $\langle d \rangle_b^L$.

P_b outputs $\langle z \rangle_b^L = (\langle a \rangle_b^L \gg s) + \langle \mathrm{corr} \rangle_b^L \cdot 2^{\ell - s} + \langle d \rangle_b^L$, $b \in \{0,1\}$.

4.3 Division

```
$F_{Div}^{ring,n,d}$  stands for division on general ring. This protocol is similar
to truncation protocol.  Since $-3d+2\leq A-d,A,A+d\leq3d-2$,  $C=(DReLU(A-
d)\oplus1) + (DReLU(A+d)\oplus1)$ can be calculated in terms of $\delta = \lceil
log_2^{6d}\rceil-bit$. Before calculate C, A needs to be calculated first.
Therefore, calculate A on $\mathbb{Z}_n$ and $\mathbb{Z}_\Delta(\Delta = 2^\delta)$
simultaneously.
```

Algorithm 9 Integer ring division, $\Pi_{\mathsf{DIV}}^{\mathsf{ring},n,d}$:

Input: For $b \in \{0,1\}$, P_b holds $\langle a \rangle_b^n$, where $a \in \mathbb{Z}_n$.
Output: For $b \in \{0,1\}$, P_b learns $\langle z \rangle_b^n$ s.t. $z = \mathsf{rdiv}(a,d)$.

For $b \in \{0,1\}$, let $a_b, a_b^0, a_b^1 \in \mathbb{Z}$ and $n^0, n^1, n' \in \mathbb{Z}$ be as defined in Theorem 4.1. Let $\eta = \lceil \log(n) \rceil$, $\delta = \lceil \log 6d \rceil$, and $\Delta = 2^\delta$.
For $b \in \{0,1\}$, P_b invokes $\mathcal{F}_{\mathsf{DReLU}}^{\mathsf{ring},n}$ with input $\langle a \rangle_b^n$ to learn output $\langle \alpha \rangle_b^B$. Party P_b sets $\langle m \rangle_b^B = \langle \alpha \rangle_b^B \oplus b$.
For $b \in \{0,1\}$, P_b sets $x_b = \mathbf{1}\{\langle a \rangle_b^n \geq n'\}$.
P_0 samples $\langle \mathsf{corr} \rangle_0^n \xleftarrow{\$} \mathbb{Z}_n$ and $\langle \mathsf{corr} \rangle_0^\Delta \xleftarrow{\$} \mathbb{Z}_\Delta$.
for $j = \{00, 01, 10, 11\}$ **do**
    P_0 computes $t_j = (\langle m \rangle_0^B \oplus j_0 \oplus x_0) \wedge (\langle m \rangle_0^B \oplus j_0 \oplus j_1)$ s.t. $j = (j_0 || j_1)$.
    if $t_j \wedge \mathbf{1}\{x_0 = 0\}$ **then**
        P_0 sets $s_j =_n -\langle \mathsf{corr} \rangle_0^n - 1$ and $r_j =_\Delta -\langle \mathsf{corr} \rangle_0^\Delta - 1$.
    else if $t_j \wedge \mathbf{1}\{x_0 = 1\}$ **then**
        P_0 sets $s_j =_n -\langle \mathsf{corr} \rangle_0^n + 1$ and $r_j =_\Delta -\langle \mathsf{corr} \rangle_0^\Delta + 1$.
    else
        P_0 sets $s_j =_n -\langle \mathsf{corr} \rangle_0^n$ and $r_j =_\Delta -\langle \mathsf{corr} \rangle_0^\Delta$.
    end if
end for
P_0 & P_1 invoke an instance of $\binom{4}{1}$-$\mathsf{OT}_{\eta+\delta}$ where P_0 is the sender with inputs $\{s_j || r_j\}_j$ and P_1 is the receiver with input $\langle m \rangle_1^B || x_1$. P_1 sets its output as $\langle \mathsf{corr} \rangle_1^n || \langle \mathsf{corr} \rangle_1^\Delta$.
For $b \in \{0,1\}$, P_b sets $\langle A \rangle_b^\Delta =_\Delta a_b^0 - (x_b - \langle \mathsf{corr} \rangle_b^\Delta) \cdot n^0$.
For $b \in \{0,1\}$, P_b sets $\langle A_0 \rangle_b^\Delta =_\Delta \langle A \rangle_b^\Delta - b \cdot d$, $\langle A_1 \rangle_b^\Delta = \langle A \rangle_b^\Delta$, and $\langle A_2 \rangle_b^\Delta =_\Delta \langle A \rangle_b^\Delta + b \cdot d$.
for $j = \{0, 1, 2\}$ **do**
    For $b \in \{0,1\}$, P_b invokes $\mathcal{F}_{\mathsf{DReLU}}^{\mathsf{int},\delta}$ with input $\langle A_j \rangle_b^\Delta$ to learn output $\langle \gamma_j \rangle_b^B$. Party P_b sets $\langle C_j' \rangle_b^B = \langle \gamma_j \rangle_b^B \oplus b$.
    For $b \in \{0,1\}$, P_b invokes an instance of $\mathcal{F}_{\mathsf{B2A}}^n$ with input $\langle C_j' \rangle_b^B$ and learns $\langle C_j \rangle_b^n$.
end for

For $b \in \{0, 1\}$, P_b sets $\langle C \rangle_b^n = \langle C_0 \rangle_b^n + \langle C_1 \rangle_b^n + \langle C_2 \rangle_b^n$.
For $b \in \{0, 1\}$, P_b sets $B_b = \mathsf{idiv}(a_b^0 - x_b \cdot n^0, d)$.
P_b sets $\langle z \rangle_b^n =_n \mathsf{rdiv}(\langle a \rangle_b^n, d) + \langle \mathsf{corr} \rangle_b^n \cdot n^1 + b - \langle C \rangle_b^n - B_b$, for $b \in \{0, 1\}$.

```
Correctness verification is the same to truncation's. Safety check is based on$(\begin{pmatrix}4\\1\end{pmatrix}-OT_{\eta + \delta}, F_{DReLU}^{ring,n}, F_{B2A}^n)$-hybird model.

The protocol calls$\begin{pmatrix}4\\1\end{pmatrix}-OT_{\eta + \delta}\ and\ F_{DReLU}^{ring,n}$ once, $F_{DReLU}^{\delta},\ and\  F_{B2A}^n$ 3 times. Total communication is less than $(\frac{3}{4}\lambda + 34)\cdot(\eta + 2\delta)$ bit.

Improvement on Avgpool is listed below:
```

Layer	Protocol	Comm. (bits)	Rounds
Avgpool_d $\mathbb{Z}_{2^\ell}$	GC [62, 63]	$2\lambda(\ell^2 + 5\ell - 3)$	2
	This work	$< (\lambda + 21) \cdot (\ell + 3\delta)$	$\log(\ell\delta) + 4$
Avgpool_d $\mathbb{Z}_n$	GC [62, 63]	$2\lambda(\eta^2 + 9\eta - 3)$	2
	This work	$< (\frac{3}{2}\lambda + 34) \cdot (\eta + 2\delta)$	$\log(\eta\delta) + 6$
Avgpool_{49} $\mathbb{Z}_{2^\ell}, \ell = 32$	GC [62, 63]	302336	2
	This work	5570	10
Avgpool_{49} $\mathbb{Z}_n, \eta = 32$	GC [62, 63]	335104	2
	This work	7796	14

Table 3: Comparison of communication with garbled circuits for Avgpool_d. We define $\eta = \lceil \log n \rceil$ and $\delta = \lceil \log(6 \cdot d) \rceil$. For concrete bits of communication we use $\lambda = 128$. Choice of $d = 49$ corresponds to average pool filter of size 7×7.

4.4 Truncation Optimization

```
For scenarios where$\ 2\cdot n^0\leq d = 2^s$ is satisfied, $\ A\geq -d$ always stands. Therefore $\ A\geq -d$ in the calculation of Ccan be omitted. Further decrease by $\delta = \lceil log_2(4d)\rceil$.
```

Part 5 Secure Inference

The inference process of neural network model is carried out as follows

1. Linear layer: Call multiplication based on OT or multiplication based on HE, adjust according to the scene;
2. ReLU: invokes the ReLU protocol
3. Avgpool: Call division protocol
4. Truncation: Since the output of ReLU is non-negative, we can reduce calculation expense
5. Maxpool and Argmax: Millionaire's and F_{MUX} in order

Part 6 Evaluation

```
In this paper, they implement OT based on EMP and chooses efficient AES. The HE is
SEAL. They integrate those into CrypTFlow system. Firstly, the ReLU is calculated bu
comparing with the GC-based method, raising it by 2-25$\times$.
```

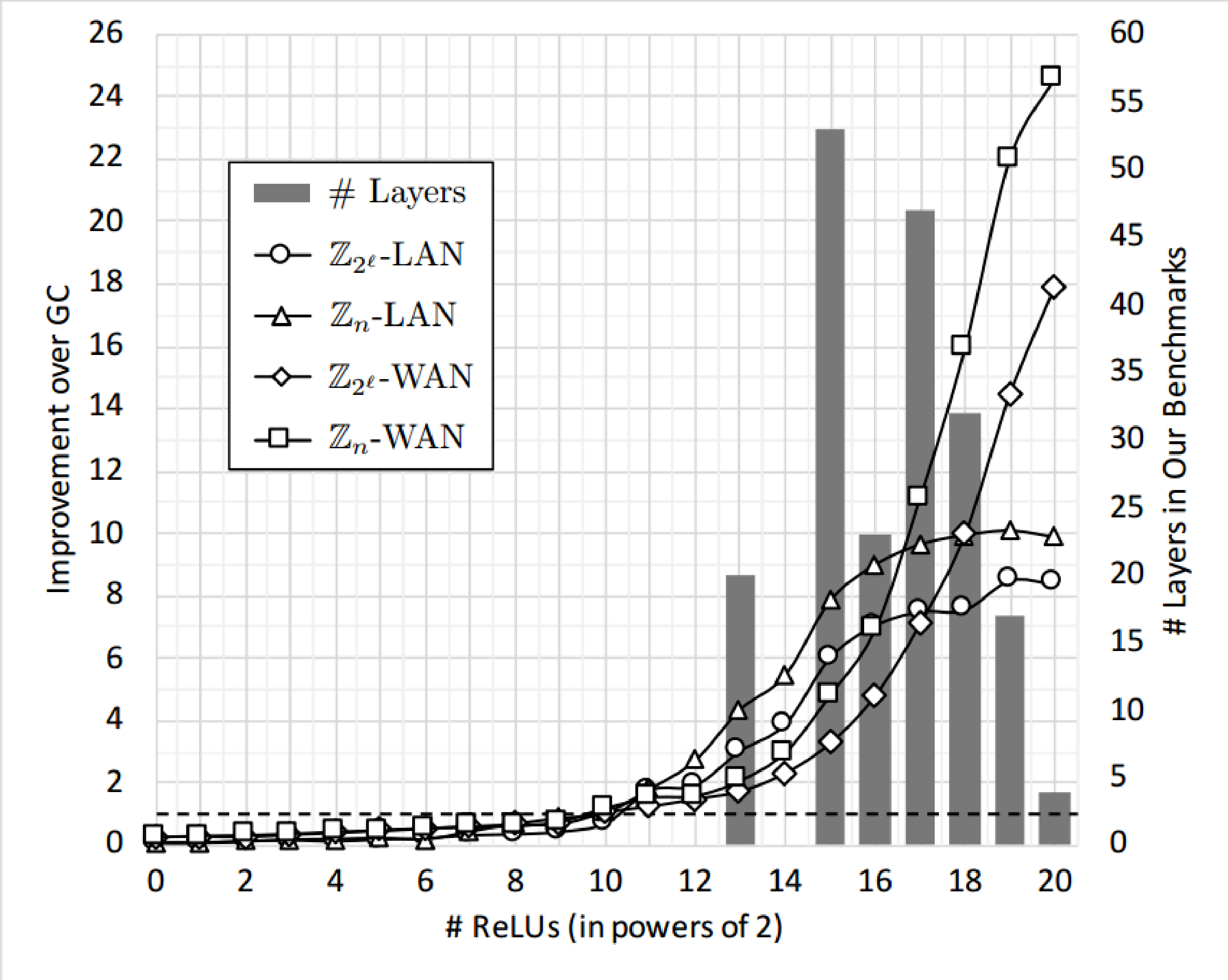


Figure 1: The left y-axis shows ($\frac{\text{GC Time}}{\text{Our Time}}$). The right y-axis shows the total number of ReLU layers corresponding to each layer size in our benchmark set. The legend entries denote the input domain and the network setting.

```
Next, the ReLU in the real network is promoted as follows. In the local area
network, improve the calculation 8-9$\times$, in the wide area network to improve
the time 18-21$\times$, improve the communication 7-9$\times$.
```

Benchmark	Garbled Circuits			Our Protocols		
	LAN	WAN	Comm	LAN	WAN	Comm
SqueezeNet	26.4	265.6	7.63	3.5	33.3	1.15
ResNet50	136.5	1285.2	39.19	16.4	69.4	5.23
DenseNet121	199.6	1849.3	56.57	24.8	118.7	8.21

(a) over $\mathbb{Z}_{2^\ell}$

Benchmark	Garbled Circuits			Our Protocols		
	LAN	WAN	Comm	LAN	WAN	Comm
SqueezeNet	51.7	525.8	16.06	5.6	50.4	1.77
ResNet50	267.5	2589.7	84.02	28.0	124.0	8.55
DenseNet121	383.5	3686.2	118.98	41.9	256.0	12.64

(b) over $\mathbb{Z}_n$

Table 4: Performance comparison with Garbled Circuits for ReLU layers. Runtimes are in seconds and comm. in GiB.

```
For Avgpool, time is improved 51$\times$.
```

Benchmark	Garbled Circuits			Our Protocol		
	LAN	WAN	Comm	LAN	WAN	Comm
SqueezeNet	0.2	2.0	36.02	0.1	0.8	1.84
ResNet50	0.4	3.9	96.97	0.1	0.8	2.35
DenseNet121	17.2	179.4	6017.94	0.5	3.5	158.83

(a) over $\mathbb{Z}_{2^\ell}$

Benchmark	Garbled Circuits			Our Protocol		
	LAN	WAN	Comm	LAN	WAN	Comm
SqueezeNet	0.2	2.2	39.93	0.1	0.9	1.92
ResNet50	0.4	4.2	106.22	0.1	1.0	3.82
DenseNet121	19.2	198.2	6707.94	0.6	4.4	214.94

(b) over $\mathbb{Z}_n$

Table 9: Performance comparison of Garbled Circuits with our protocols for computing Avgpool **layers. Runtimes are in seconds and communication numbers are in MiB.**

```
Compared with Delphi, it is still greatly improved in both nonlinear layer and
online calculation.
```

Benchmark	Metric	Linear	Non-linear		
			Delphi	Ours	Improvement
MiniONN	Time	10.7	30.2	1.0	30.2×
	Comm.	0.02	3.15	0.28	12.3×
ResNet32	Time	15.9	52.9	2.4	22.0×
	Comm.	0.07	5.51	0.59	9.3×

Table 5: Performance comparison with Delphi [49] for non-linear layers. Runtimes are in seconds and comm. in GiB.

Benchmark	Linear	Non-linear		
		Delphi	Ours	Improvement
MiniONN	< 0.1	3.97	0.32	12.40×
ResNet32	< 0.1	6.99	0.63	11.09×

Table 6: Performance comparison with Delphi [49] for online runtime in seconds.

```
Finally, for large networks, predictions can be made in 10min (LAN) and 20min (WAN).
```

Benchmark	Protocol	LAN	WAN	Comm
SqueezeNet	SCI_{OT}	44.3	293.6	26.07
	SCI_{HE}	59.2	156.6	5.27
ResNet50	SCI_{OT}	619.4	3611.6	370.84
	SCI_{HE}	545.8	936.0	32.43
DenseNet121	SCI_{OT}	371.4	2257.7	217.19
	SCI_{HE}	463.2	1124.7	35.56

Table 7: Performance of CrypTFlow2 on ImageNet-scale benchmarks. Runtimes are in seconds and comm. in GiB.

Part 7 Conclusion

```
This paper is one of the best two-party computational comparison protocols and is
the basis of SIRNN. In this paper, an efficient calculation method of accurate
```

```
truncation and division is presented. This ideas are very enlightening for later
work.
```

Appendix I: Code-reading

millionaire_with_equality.h:

`MillionaireWithEquality` ��h��ģ���ʾ��ģ������ `IO`����������ÇĿ����˽�̡����������ıŁ�Э�養�����ͬ�������Ļ�顲��Ĺ������Ա��������£�

- `IO *io`: h���̿� IO ������̿룬���ڷ�����������������
- `sci::OTPack<IO> *otpack`: h���̿� `sci::OTPack<IO>` ������̿룬���ڷ���� Oblivious Transfer��OT��Э�顲
- `TripleGenerator<IO> *triple_gen`: h���̿� `TripleGenerator<IO>` ������̿룬���������ДК�養��Щ��ДК����Э���������ڷ�ö����龍
- `MillionaireProtocol<IO> *mill`: h���̿� `MillionaireProtocol<IO>` ������̿룬�����̠����Ł�Э�顲
- `bool del_mill`: h��������ͅ�Ƿ���Ç��������������ɾ�� `mill` ����
- `int party`: ��ʾ���븘�ı�ţ̌���������������Э���elJ�ʹ������Δ�
- `int l, r, log_alpha, beta, beta_pow`: ��Щ�����������ڷ浉Э���eк�̤�������м����������磬`l` ��ʾ���سּ��Ȣ�`beta` ��ʾ�������`beta_pow` ��ʾ `beta` ���ڪ�
- `int num_digits, num_triples, log_num_digits`: ��Щ�������������ڷ浉Э���eк�������̡����磬`num_digits` ��ʾ���ĸ�����`num_triples` ��ʾ��ДК����������
- `uint8_t mask_beta, mask_r`: ��Щ 8 λ�ڷ����������ϲ浉�������̡�����������Э���н���λ������

��Щ��Ա������һ����ʾ������ `MillionaireWithEquality` ��Ļ�����������ıŁ�Э��������Ļ�顲

`MillionaireWithEquality` ��Ĺ��캯

�������ڷ�ʹ������ˈ������������������������

- `party`: ��ʾ���븘�ıš�
- `io`: h���̿� IO ������̿룬���ڷ����������������

- `otpack`: h����� `sci::OTPack<IO>` ������런���گ��� Oblivious Transfer��OT��Э��顯
- `mill_in`: h����w���� `MillionaireProtocol<IO>` ������런 �������ş�����Ĭ��Ĭ `nullptr`��
- `bitlength`: h����������̓���س��Ծ�Ĭ��Ĭ 32��
- `radix_base`: h����������̓�������Ĭ��Ĭ `MILL_PARAM`��

�ڽ��캯���У����Ł������IJ����������ij�Ա���������� `mill_in` Ĭ `nullptr`����h����µ� `MillionaireProtocol<IO>` ������� `del_mill` ����Ĭ `true`���Ա���������������ŕ����ö�������� `mill_in` ���� `mill` ��Ա�������

���������� `mill` ����� `triple_gen` ��Ա������������ `triple_gen` ��Ա������������ `configure` ������ʹ��ô����� `bitlength` �� `radix_base` ������������á�

������캯��ȷ���� `MillionaireWithEquality` ���ʵ���گ���ĵ��ȷ�ʹ�����б�Ç�ij�Ա���������������Ç�������ʹ������e� `MillionaireProtocol<IO>` ����

`bitlen_lt_beta` ��h�����ڷȶɛ������Ł��������붙 ����ر������ڷϽ������ú����ʹ���ȭȍ�ӧ����ǳMPC���e� Oblivious Transfer��OT��Э����Ļ����������ıŁ������IJ�������£�

- `res_cmp`: h����� `uint8_t` ����������런���گ泙�ŁϽ�����
- `res_eq`: h����� `uint8_t` ����������런���گ泙 ����Ł������
- `data`: h����� `uint64_t` ����������런����Ç�Łɛ�����ڥ�
- `num_cmps`: h����������̓Ç����ϬŁɛĴ�����
- `bitlength`: h����������̓ÿ������ڍı��س���ȡ�
- `greater_than`: h���������̓�ŁɛĶ���Ĭ��Ĭ `true`��
- `radix_base`: h����������̓�������Ĭ��Ĭ `MILL_PARAM`��

��������¢��� N �� mask������ N �� 2 �� bitlength ���ڷ�mask �� N - 1����Щ����������������ȷ�����������ڍ���ȍ���ı��ف�����س�

����������������ځ��붚
�Ľ�ƚ��party�����в�ʳ�Ĳ��������� party ��
sci::ALICE������������������ res_cmp �� res_eq ���顢
Ȼ��Ïÿ���ŁӨ���h�� leaf_messages ���養�������
set_leaf_ot_messages ��������Ч�J̌�� OT ��Ш������� bitlength
��.��w��ʳ�� OT Э�鷹����Щ��Щ�������и� leaf_messages
������ڨ档

��� party �� sci::BOB���□�h�� choice ���養�浬
ÿ�����ۍ����������.����� bitlength ��.��w��ʳ�� OT
Э����ծŁϽ����Ȼ�□��� res_eq �� res_cmp ���養�и� choice
������ڨ档

��������̒��ʹ�� OT
Э���������������ɣ�ȷ����‚ŁC��İ�ȍ�ԁ���‚�ҧ�

compare_with_eq �������ڰŁ��������붚
�����ر������ڨŁϽ���������Ⱡ��������ú���ʹ�ð�ȍ�□���
䶖MPC����e� Oblivious
Transfer��OT��Э�����ˡ�.����������ıŁ)�������Ĳ�������£�

- res_cmp: h��ˌ�� uint8_t ����������ˌ�룬����ڨ浬�ŁϽ����
- res_eq: h��ˌ�� uint8_t ����������ˌ�룬����ڨ浬
  ����Ⱡ������
- data: h��ˌ�� uint64_t ����������ˌ�룬����Ç�Łє����ݐ�
- num_cmps: h����������̀Ç����ɓŁєĴ�����
- bitlength: h����������̀ÿ������ۍı��س��ɖ�
- greater_than: h�������.ˌ�̀�Łєķ���Ĭ��Ï true��
- radix_base: h����������̀������Ĭ��Ï MILL_PARAM��

��������ɳ��� configure ���������ȉ���� bitlength ��
radix_base ������������á���� bitlength C�ڽ����
beta������� bitlen_lt_beta �������ɓŁш��������ۂ����

�������������ر�� num_cmps �� 8
�ı�������������ǣ���������輯
Ȼ�□���Ïÿ���ŁӨ���h����ʅ��������
data_ext������ӵʹ���ʳ̡��Ƶ���������У���鵖���� 0 ���론

�������ŷ����ڱ捲���ڱ泙

����������ȡ�ļ�������¸�digits�����ŁϽ����leaf_res_cmp�
�������Ϸ������leaf_res_eq����Ȼ�ͬ��������������ȡ�
�������¸����泙�� digits �����C�

���ڸ��붼�Ľ�ł��party�����������в�Ĳ�������� party
�� sci::ALICE������������������ leaf_res_cmp ��
leaf_res_eq ���養������ set_leaf_ot_messages
��������Ҷ�Ĵ�� OT ��Ш��Ȼ�ͬ���� bitlength �� r
�������w����� OT Э�鷢����Щ��Ш�������и� leaf_ot_messages
������ڵ档

��� party �� sci::BOB������� bitlength �� r ����ޤw����� OT
Э����ðŁϽ����Ȼ�ͬ��� leaf_res_eq �� leaf_res_cmp ���養�и�
choice ������ڵ档

��������̤�ʹ�� OT
Э��������������ɣ�ȷ����˛ŁC��İ�ő�ժ����¸�ђ�