FLASH: Fast and Robust Framework for Privacy-Preserving Machine Learning

论文发表在PoPETs20，下载链接FLASH。

1. 设计思想

之前介绍的有关PPML的论文，大多数还是围绕半诚实（semi-honest）模型展开的工作。核心的任务聚焦于在保证隐私的情况下尽可能的提升系统性能。除了ABY3保证了正确性（correct with abort）和ASTRA进一步保证了公平性（Fairness）。FLASH进一步实现了输出可达性（Guaranteed Output Delivery，GOD），即无论恶意敌手进行何种攻击诚实参与方都可以得到计算结果。FLASH采用了4方计算架构，在诚实大多数（最多1方是静态恶意敌手）情况下可以满足GOD安全。其核心的设计思想在于，当检测到恶意行为时可以定位到恶意方在两方之间（但是不能确定具体是哪一方），但是可以确定剩余的两方是诚实参与方。如此，则可以令诚实参与方获得数据明文，从安全计算转化为明文计算（不对诚实参与方保护隐私）。

2. 主要工作

1）FLASH首先基于Additive Shairing（$[\cdot]$-sharing）设计了4方下的Mirrored Sharing（$[\![\cdot]\!]$-sharing）方案；
2）进一步构造了Bi-Convey原语用来在4方下将S_1 和 S_2共有的输入x在T的辅助下传送给R，该过程或者成功传送（S_1和S_2没有恶意行为），或者R和T能确定敌手在（S_1，S_2）之间（之后R和T交换各自的share恢复明文，进行明文计算）；
3）最终，关于乘法和比较等操作的构造，则是让每一次交互都可以抽象成一次Bi-Convey过程，从而使得整个系统能够满足GOD安全性要求；
4）进一步，FLASH构造了面向机器学习的计算模块，包括向量乘法、激活函数计算、截断、比特转化等，并对这些模块做了进一步优化。例如，向量乘法的通信量与向量大小无关、Sigmod函数的近似等。并和ABY3、ASTRA进行了比较。性能的理论提升如下表。

Protocol	Equation	ABY3		ASTRA		FLASH	
		Rounds	Comm.	Rounds	Comm.	Rounds	Comm.
Multiplication	$[\![x]\!].[\![y]\!] \to [\![x.y]\!]$	5	21ℓ	7	25ℓ	5	12ℓ
Dot Product	$[\![\vec{x} \odot \vec{y}]\!] = [\![\sum_{i=1}^{d} x_i y_i]\!]$	5	$21m\ell$	7	$23m\ell + 2\ell$	5	12ℓ
MSB Extraction	$[\![x]\!] \to [\![\mathsf{msb}(x)]\!]^{\mathbf{B}}$	$\log \ell + 4$	42ℓ	10	$52\ell + 4$	6	$16\ell + 4$
Truncation	$[\![x]\!].[\![y]\!] \to [\![(xy)^t]\!]$	$2\ell - 1$	$\approx 108\ell$	–	–	5	14ℓ
Bit Conversion	$[\![b]\!]^{\mathbf{B}} \to [\![b]\!]$	6	42ℓ	–	–	5	14ℓ
Bit Insertion	$[\![b]\!]^{\mathbf{B}}[\![x]\!] \to [\![bx]\!]$	7	63ℓ	–	–	5	18ℓ

Table 1. Comparison of FLASH framework with ABY3 and ASTRA; ℓ and m denote the ring size and number of features respectively.

3. Mirrored Sharing Semantics

- Additive Sharing（$[\cdot]$-sharing）：对于x，在2方下可以被分享为 x^1和x^2，满足$x = x^1 + x^2$。其中每一个参与方持有一份。
- Mirrored Sharing（$[\![\cdot]\!]$-sharing）：对于x在4方下：
  - 存在σ_x和μ_x满足：$\mu_x = x + \sigma_x$；
  - σ_x被$[\cdot]$-sharing分享在参与方$\mathrm{E} = \{\mathrm{E}_1, \mathrm{E}_2\}$之间，即$[\sigma_x]_{\mathrm{E}1} = \sigma_x^1$，$[\sigma_x]_{\mathrm{E}2} = \sigma_x^2$。而参与方$\mathrm{V} = \{\mathrm{V}_1, \mathrm{V}_2\}$则都持有$\sigma_x^1$和$\sigma_x^2$；
  - 类似的，μ_x 被$[\cdot]$-sharing 分享在$\mathrm{V} = \{\mathrm{V}_1, \mathrm{V}_2\}$之间，即$[\mu_x]_{\mathrm{V}_1} = \mu_x^1$，$[\mu_x]_{\mathrm{V}_2} = \mu_x^2$。而$\mathrm{E} = \{\mathrm{E}_1, \mathrm{E}_2\}$则都持有$\mu_x^1$和$\mu_x^2$。整体的语义分享如下：

E_1：$[\![x]\!]_{\mathrm{E}_1} = (\sigma_x^1, \mu_x^1, \mu_x^2)$	V_1：$[\![x]\!]_{\mathrm{V}_1} = (\sigma_x^1, \sigma_x^2, \mu_x^1)$
E_2：$[\![x]\!]_{\mathrm{E}_2} = (\sigma_x^2, \mu_x^1, \mu_x^2)$	V_2：$[\![x]\!]_{\mathrm{V}_2} = (\sigma_x^1, \sigma_x^2, \mu_x^2)$

很显然，从$[\cdot]$-sharing 的线性性质可以很容易的推导出$[\![]\!]$-sharing 的线性，即支持非交互计算加法和明文-密文乘法。

4. Robust 4PC

4.1 Bi-Convey

如前所属，Bi-Convey $\Pi_{bic}(S_1, S_2, x, R, T)$在4方下将$S_1$ 和 S_2共有的输入x在T的辅助下传送给R，该过程或者成功传送（S_1和S_2没有恶意行为），或者R和T能确定敌手在（S_1，S_2）之间（之后R和T交换各自的share恢复明文，进行明文计算）。具体来说：

1. S_1和S_2各自将x发送给R。同时，S_1和S_2将关于x的承诺$com(x)$发送给T。当然，关于承诺使用的随机数是相同的；
2. 如果R收到的x是相等的，那么S_1和S_2均没有作恶。R接受x，令$msg_R = continue$，并丢弃来自T的任何信息；否则，令$msg_R = I_R$，其中I_R表示R的share和随机数种子；
3. 对于T，如果收到的承诺相同，则令$msg_T = com(x)$；否则令$msg_T = I_R$；

4. R和T互相交换msg；
5. 如果$msg_R = I_R$且$msg_T = com(x)$，则R接受和msg_T匹配的x。否则，R和T可以在本地恢复明文进行明文计算。

4.2 Input Sharing

在Input Sharing阶段，Dealer可能是恶意的敌手，那么其可能分发给不同的参与方的share不匹配。为了确认一致性，需要在share之后利用承诺进行验证。例如，如果Dealer是V_1，在预计算阶段，所有的参与方根据共享的随机数种子生成（$\sigma_x^1, \sigma_x^2, \mu_x^1$）。在线计算阶段，$\mathrm{V}_1$计算$\mu_2 = x + \sigma_x^1 + \sigma_x^2 - \mu_x^1$并把$\mu_x^2$发送给$\mathrm{E}$和$\mathrm{V}_2$。最后，$\mathrm{E}_1, \mathrm{E}_2, \mathrm{V}_2$利用承诺$com(\mu_x^2)$来进行验证，取majority为最终分享结果。Dealer是其他参与方的情况类似。具体协议如下。

- **Input:** Party D inputs value x while others input $\perp$.
- **Output:** Parties obtain $[\![x]\!]$ as the output.

– **If** $\mathsf{D} = \mathsf{E}_1$**:** Parties in $\mathbf{V}$ and E_1 locally sample σ_x^1, while all the parties in $\mathcal{P}$ locally sample σ_x^2. Parties in $\mathbf{V}$ and E_1 locally compute $\sigma_x = \sigma_x^1 + \sigma_x^2$. Similar steps are done for $D = \mathsf{E}_2$.

– **If** $\mathsf{D} = \mathsf{V}_i$ **for** $i \in \{1, 2\}$**:** Parties in $\mathbf{V}$ and E_1 locally sample σ_x^1, while parties in $\mathbf{V}$ and E_2 locally sample σ_x^2. Parties in $\mathbf{V}$ locally compute $\sigma_x = \sigma_x^1 + \sigma_x^2$.

– **If** $\mathsf{D} = \mathsf{V}_1$**:** Party V_1 computes $\mu_x = x + \sigma_x$. Parties in $\mathbf{E}$ and V_1 locally sample μ_x^1. Party V_1 computes and sends $\mu_x^2 = \mu_x - \mu_x^1$ to parties in $\mathbf{E}$ and V_2. Parties in $\mathbf{E}$ and V_2 exchange the received copy of μ_x^2. If there exists no majority, then they identify V_1 to be corrupt and engage in semi-honest 3PC excluding V_1 (with default input for V_1). Else, they set μ_x^2 to the computed majority. Similar steps are done for $D = \mathsf{V}_2$.

– **If** $\mathsf{D} = \mathsf{E}_i$ **for** $i \in \{1, 2\}$**:** Party E_i computes $\mu_x = x + \sigma_x$. Parties in $\mathbf{E}$ and V_1 locally sample μ_x^1. Party E_i computes and sends $\mu_x^2 = \mu_x - \mu_x^1$ to V_2 and the co-evaluator. E_i sends $\mathsf{com}(\mu_x^2)$ to V_1. Parties other than the dealer exchange the commitment of μ_x^2 to compute majority (the co-evaluator and V_2 also exchange their copies of μ_x^2). If no majority exists, then they identify E_i to be corrupt and engage in semi-honest 3PC excluding E_i (with default input for E_i). Else, they set μ_x^2 to the computed majority.

Fig. 3. $\Pi_{\mathsf{sh}}(\mathsf{D}, x)$: Protocol to generate $[\![x]\!]$ by dealer D.

4.3 Circuit Evaluation

加法和明文-密文乘法比较容易，难点在于密文-密文乘法。对于乘法$z = xy$，E中两方的目标是计算

$$
\begin{aligned}
\mu_z &= xy + \sigma_z \\
&= (\mu_x - \sigma_x)(\mu_y - \sigma_y) + \sigma_z \\
&= \mu_x\mu_y - \mu_x\sigma_y - \mu_y\sigma_x + \sigma_x\sigma_y + \sigma_z
\end{aligned}
$$

令 $A = -\mu_x^1\sigma_y - \mu_y^1\sigma_x + \delta_{xy} + \sigma_z + \Delta$，$B = -\mu_x^2\sigma_y - \mu_y^2\sigma_x - \Delta$，其中$\delta_{xy} = \sigma_x\sigma_y$。那么根据$[\![\,]\!]$-sharing的语义，在预计算阶段的随机数生成基础上，V_1可以在本地计算A，V_2可以在本地计算B。但是这并不能保证A和B能够成功的被E获得。为了解决这个问题，进一步将A和B分解为$A = A_1 + A_2$，$B = B_1 + B_2$。其中，$A_j = -u_x^1\sigma_y^j - \mu_y^1\sigma_x^j + \delta_{xy}^j + \Delta_j$被$\mathrm{V}_1$和$\mathrm{E}_j$共有，$B_j = -u_x^2\sigma_y^j - \mu_y^2\sigma_x^j - \Delta_j$被$\mathrm{V}_2$和$\mathrm{E}_j$共有，$j \in \{1, 2\}$。如此一来，$A_j$，$B_j$则可以利用$\Pi_{bic}$成功发送给$\mathrm{E}$中两方。其中$\delta_{xy}^j$，$\Delta_j$可以在预计算也利用共享随机数种子和原语$\Pi_{bic}$生成。最终，$\mathrm{E}$在计算$\mu_z$之后便可以计算$\mu_z^2 = \mu_z - \mu_z^1$，并利用$\Pi_{bic}(\mathrm{E}_1, \mathrm{E}_2, \mu_z^2, \mathrm{V}_2, \mathrm{V}_1)$将$\mu_z^2$发送给$\mathrm{V}_2$。具体协议$\Pi_{mult}$如下。

- **Input:** Parties input their $[\![\mathsf{x}]\!]$ and $[\![\mathsf{y}]\!]$ shares.
- **Output:** Parties obtain $[\![\mathsf{z}]\!]$ as the output, where $\mathsf{z} = \mathsf{xy}$.

– Parties in $\mathbf{V}$ and E_1 collectively sample σ_{z}^1 and δ_{xy}^1, while parties in $\mathbf{V}$ and E_2 together sample σ_{z}^2.

– Verifiers $\mathsf{V}_1, \mathsf{V}_2$ compute $\delta_{\mathsf{xy}} = \sigma_{\mathsf{x}}\sigma_{\mathsf{y}}$, set $\delta_{\mathsf{xy}}^2 = \delta_{\mathsf{xy}} - \delta_{\mathsf{xy}}^1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{V}_2, \delta_{\mathsf{xy}}^2, \mathsf{E}_2, \mathsf{E}_1)$, which makes sure that E_2 receives δ_{xy}^2.

– Parties in $\mathbf{V}$ and E_1 collectively sample Δ_1. Parties V_1 and E_1 compute $\mathsf{A}_1 = -\mu_{\mathsf{x}}^1\sigma_{\mathsf{y}}^1 - \mu_{\mathsf{y}}^1\sigma_{\mathsf{x}}^1 + \delta_{\mathsf{xy}}^1 + \sigma_{\mathsf{z}}^1 + \Delta_1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_1, \mathsf{A}_1, \mathsf{E}_2, \mathsf{V}_2)$, such that E_2 receives A_1.

– Similarly, parties in $\mathbf{V}$ and E_2 collectively sample Δ_2. Parties V_1 and E_2 compute $\mathsf{A}_2 = -\mu_{\mathsf{x}}^1\sigma_{\mathsf{y}}^2 - \mu_{\mathsf{y}}^1\sigma_{\mathsf{x}}^2 + \delta_{\mathsf{xy}}^2 + \sigma_{\mathsf{z}}^2 + \Delta_2$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_2, \mathsf{A}_2, \mathsf{E}_1, \mathsf{V}_2)$, such that E_1 receives A_2.

– Parties V_2 and E_1 compute $\mathsf{B}_1 = -\mu_{\mathsf{x}}^2\sigma_{\mathsf{y}}^1 - \mu_{\mathsf{y}}^2\sigma_{\mathsf{x}}^1 - \Delta_1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_1, \mathsf{B}_1, \mathsf{E}_2, \mathsf{V}_1)$. Similarly, V_2 and E_2 compute $\mathsf{B}_2 = -\mu_{\mathsf{x}}^2\sigma_{\mathsf{y}}^2 - \mu_{\mathsf{y}}^2\sigma_{\mathsf{x}}^2 - \Delta_2$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_2, \mathsf{B}_2, \mathsf{E}_1, \mathsf{V}_1)$.

– Evaluators compute $\mu_z = \mathsf{A}_1 + \mathsf{A}_2 + \mathsf{B}_1 + \mathsf{B}_2 + \mu_x\mu_y$ locally. Parties in $\mathbf{E}$ and V_1 collectively sample μ_z^1 followed by evaluators setting $\mu_z^2 = \mu_z - \mu_z^1$ and invoking $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu_z^2, \mathsf{V}_2, \mathsf{V}_1)$for V_2 to receive μ_z^2.

Fig. 4. $\Pi_{\mathsf{mult}}(\mathsf{x}, \mathsf{y}, \mathsf{z})$: Multiplication Protocol

4.4 Output Computation

恢复算法比较直观，因为每一方缺失的份额都被其他三方持有，所以可以令其他三方中两方发送缺失的份额，而剩下的一方直接发送对应的哈希值，最终结果取majority。具体协议Π_{oc}如下。

- **Input:** Parties input their $[\![\mathsf{z}]\!]$ shares.
- **Output:** Parties obtain z as the output.

– For $i, j \in \{1, 2\}$ and $i \neq j$, E_i receives σ_{z}^{j} from parties in $\mathbf{V}$ and $\mathsf{H}(\sigma_{\mathsf{z}}^{j})$ from E_j .
– V_2 receives μ_{z}^{1} from parties in $\mathbf{E}$ and $\mathsf{H}(\mu_{\mathsf{z}}^{1})$ from V_1.
– V_1 receives μ_{z}^{2} from parties in $\mathbf{E}$ and $\mathsf{H}(\mu_{\mathsf{z}}^{2})$ from V_2.
– Each party sets the missing share as the majority among the received values and outputs $\mathsf{z} = \mu_{\mathsf{z}}^{1} + \mu_{\mathsf{z}}^{2} - \sigma_{\mathsf{z}}^{1} - \sigma_{\mathsf{z}}^{2}$.

Fig. 5. Π_{oc}: Protocol for Robust Reconstruction

5. ML Building Blocks

进一步构造面向ML计算的安全计算模块。

5.1 Arithmetic/Boolean Couple Sharing Primitive

在之后的计算中，会出现秘密值x只被E或者V中两方持有，持有双方试图生成$[\![x]\!]$的情况。对于这种情况的sharing，可以令被另外两方完全持有的分享为0，从而减少开销。具体来说，如果x被E中两方持有，则$\sigma_x^1 = \sigma_x^2 = 0$；如果$x$被$\mathrm{V}$中两方持有，则$\mu_x^1 = \mu_x^2 = 0$。具体协议$\Pi_{cSh}$如下。

Case 1: ($\mathbf{S} = \mathbf{E}$)

- **Input:** E_1 and E_2 input x while others input $\perp$.
- **Output:** Parties obtain $[\![x]\!]$ as the output.

– Parties set $\sigma_x^1 = 0$ and $\sigma_x^2 = 0$. Parties in $\mathbf{E}$ and V_1 collectively sample random $\mu_x^1 \in \mathbb{Z}_{2^\ell}$.
– E_1 and E_2 set $\mu_x^2 = x - \mu_x^1$. Parties then execute $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu_x^2, \mathsf{V}_2, \mathsf{V}_1)$, such that V_2 receives μ_x^2.

Case 2: ($\mathbf{S} = \mathbf{V}$)

- **Input:** V_1 and V_2 input x while others input $\perp$.
- **Output:** Parties obtain $[\![x]\!]$ as the output.

– Parties set $\mu_x^1 = 0$ and $\mu_x^2 = 0$. Parties in $\mathbf{V}$ and E_1 collectively sample random $\sigma_x^1 \in \mathbb{Z}_{2^\ell}$.
– V_1 and V_2 set $\sigma_x^2 = x - \sigma_x^1$. Parties then execute $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{V}_2, \sigma_x^2 5, \mathsf{E}_2, \mathsf{E}_1)$, such that E_2 receives σ_x^2.

注意，Case 2存在笔误：1)$\sigma_x^2 = -x - \sigma_x^1$； 2)$\Pi_{bic}(\mathrm{V}_1, \mathrm{V}_2, \sigma_x^2, \mathrm{E}_2, \mathrm{E}_1)$。

5.2 Dot Product

对于向量内积 $z = \vec{\mathbf{x}} \odot \vec{\mathbf{y}} = \sum_{i=1}^{d} x_i y_i$，可以简单的调用乘法协议$\Pi_{mult}$完成，但是这会使得通信和向量大小正相关。为了使得通信量独立于向量大小，可以借鉴ABY3中的方法，即现在share上做完

加法求和再对求和结果进行通信。需要改变的则是对于$\delta_{xy}^2 = \sum_{i=1}^{d} \delta_{xi\,yi} - \delta_{xy}^1$的计算，类似的还有关于$A_j, B_j$的计算。其余计算则没有太大改变。具体协议$\Pi_{dp}$如下。

- **Input:** Parties input their $[\![\vec{\mathbf{x}}]\!]$ and $[\![\vec{\mathbf{y}}]\!]$ shares.
- **Output:** Parties obtain $[\![\mathbf{z}]\!]$ as output, where $\mathbf{z} = \vec{\mathbf{x}} \odot \vec{\mathbf{y}}$.

– Parties in $\mathbf{V}$ and E_1 collectively sample σ_{z}^1 and δ_{xy}^1, while parties in $\mathbf{V}$ and E_2 together sample σ_{z}^2.

– Verifiers $\mathsf{V}_1, \mathsf{V}_2$ compute $\delta_{\mathsf{xy}} = \Sigma_{i=1}^{d} \sigma_{\mathsf{x}_i} \sigma_{\mathsf{y}_i}$, set $\delta_{\mathsf{xy}}^2 = \delta_{\mathsf{xy}} - \delta_{\mathsf{xy}}^1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{V}_2, \delta_{\mathsf{xy}}^2, \mathsf{E}_2, \mathsf{E}_1)$, such that E_2 receives δ_{xy}^2.

– Parties in $\mathbf{V}$ and E_1 collectively sample Δ_1. Parties V_1 and E_1 compute $\mathsf{A}_1 = \Sigma_{i=1}^{d}(-\mu_{x_i}^1 \sigma_{y_i}^1 - \mu_{y_i}^1 \sigma_{x_i}^1) + \sigma_{\mathsf{z}}^1 + \delta_{\mathsf{xy}}^1 + \Delta_1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_1, \mathsf{A}_1, \mathsf{E}_2, \mathsf{V}_2)$, such that E_2 receives A_1.

– Similarly, parties in $\mathbf{V}$ and E_2 collectively sample Δ_2. Parties V_1 and E_2 compute $\mathsf{A}_2 = \Sigma_{i=1}^{d}(-\mu_{x_i}^1 \sigma_{y_i}^2 - \mu_{y_i}^1 \sigma_{x_i}^2) + \sigma_{\mathsf{z}}^2 + \delta_{\mathsf{xy}}^2 + \Delta_2$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_2, \mathsf{A}_2, \mathsf{E}_1, \mathsf{V}_2)$, such that E_1 receives A_2.

– V_2 and E_1 compute $\mathsf{B}_1 = \Sigma_{i=1}^{d}(-\mu_{x_i}^2 \sigma_{y_i}^1 - \mu_{y_i}^2 \sigma_{x_i}^1) - \Delta_1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_1, \mathsf{B}_1, \mathsf{E}_2, \mathsf{V}_1)$. Similarly, V_2 and E_2 compute $\mathsf{B}_2 = \Sigma_{i=1}^{d}(-\mu_{x_i}^2 \sigma_{y_i}^2 - \mu_{y_i}^2 \sigma_{x_i}^2) - \Delta_2$ and execute $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_2, \mathsf{B}_2, \mathsf{E}_1, \mathsf{V}_1)$.

– Evaluators compute $\mu_z = \mu_x \mu_y + \mathsf{A}_1 + \mathsf{A}_2 + \mathsf{B}_1 + \mathsf{B}_2$ locally. Parties in $\mathbf{E}$ and V_1 collectively sample μ_z^1 followed by evaluators setting $\mu_z^2 = \mu_z - \mu_z^1$ and execute $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu_z^2, \mathsf{V}_2, \mathsf{V}_1)$ for V_2 to receive μ_z^2.

Fig. 7. $\Pi_{\mathsf{dp}}([\![\vec{\mathbf{x}}]\!], [\![\vec{\mathbf{y}}]\!])$: Dot Product of two vectors

5.3 MSB Extraction

对于比较$u < v$，其等价于提取$a = u - v$的最高有效位$msb(a)$。本文采取和ASTRA相同的设计思想：对于随机数$r \in \mathbb{Z}_{2^\ell}$，有$msb(a) = msb(r) \oplus msb(ra)$。因此，可以令参与方在预计算阶段生成$[\![r]\!]$和$[\![p]\!]^{\mathbf{B}}$，其中$p = msb(r)$。在线计算阶段，则求调用$\Pi_{mult}([\![r]\!], [\![a]\!])$并公开计算结果$ra$从而求得$q = msb(ra)$，然后计算$[\![q]\!]^{\mathbf{B}}$，最后计算$[\![msb(a)]\!]^{\mathbf{B}} = [\![p]\!]^{\mathbf{B}} \oplus [\![q]\!]^{\mathbf{B}}$。但是利用乘法隐藏真实值是有一定的泄露的：例如，如果r是奇数，而公开的ra是偶数，那么a一定是偶数。所以，这种方法的安全性并不如one-time-pad。

- **Input:** Parties input their $[\![\mathsf{a}]\!]$ shares.
- **Output:** Parties obtain $[\![\mathsf{msb(a)}]\!]^{\mathbf{B}}$ as the output.

– Parties in $\mathbf{E}$ sample random $\mathsf{r} \in \mathbb{Z}_{2^\ell}$ and set $\mathsf{p} = \mathsf{msb(r)}$.

– Parties execute $\Pi_{\mathsf{cSh}}(\mathbf{E}, \mathsf{r})$ and $\Pi^{\mathbf{B}}_{\mathsf{cSh}}(\mathbf{E}, \mathsf{p})$ to generate $[\![\mathsf{r}]\!]$ and $[\![\mathsf{p}]\!]^{\mathbf{B}}$ respectively.

– Parties execute $\Pi_{\mathsf{mult}}([\![\mathsf{r}]\!], [\![\mathsf{a}]\!])$ to generate $[\![\mathsf{ra}]\!]$. Parties also execute $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu^2_{\mathsf{ra}}, \mathsf{V}_1, \mathsf{V}_2)$ and $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu^1_{\mathsf{ra}}, \mathsf{V}_2, \mathsf{V}_1)$ to reconstruct ra towards V_1 and V_2 respectively. Verifiers then set $\mathsf{q} = \mathsf{msb(ra)}$.

– Parties execute $\Pi^{\mathbf{B}}_{\mathsf{cSh}}(\mathbf{V}, \mathsf{q})$ to generate $[\![\mathsf{q}]\!]^{\mathbf{B}}$ followed by locally computing $[\![\mathsf{msb(a)}]\!]^{\mathbf{B}} = [\![\mathsf{p}]\!]^{\mathbf{B}} \oplus [\![\mathsf{q}]\!]^{\mathbf{B}}$.

Fig. 8. $\Pi_{\mathsf{msb}}([\![\mathsf{a}]\!])$: Extraction of MSB from a value

5.4 Truncation

为了防止多次连续乘法造成溢出，本文截断方案采取类似ABY3中的截断方法：首先生成(r, r^t)的秘密分享，其中$r^t = r/2^d$。在线计算计算，乘法计算完成时公开$z - r$，并截断$z - r$获得$(z-r)^t$。进一步，利用协议$\Pi_{cSh}(\mathrm{E}, (z-r)^t)$生成$[\![(z-r)^t]\!]$，并计算最后结果$[\![z^t]\!] = [\![(z-r)^t]\!] + [\![r^t]\!]$。具体协议$\Pi^A_{mulTr}$如下。

- **Input:** Parties input their $[\![\mathsf{x}]\!]$ and $[\![\mathsf{y}]\!]$ shares.
- **Output:** Parties obtain $[\![\mathsf{z^t}]\!]$ as output, where $\mathsf{z^t} = (\mathsf{xy})^{\mathsf{t}}$.

– Parties in $\mathbf{V}$ and E_1 collectively sample σ^1_{z} and r_1, while parties in $\mathbf{V}$ and E_2 together sample σ^2_{z} and r_2.

– Verifiers set $\mathsf{r} = \mathsf{r}_1 + \mathsf{r}_2$ and truncate r by d bits to obtain $\mathsf{r^t}$. Parties execute $\Pi_{\mathsf{cSh}}(\mathbf{V}, \mathsf{r^t})$ to generate $[\![\mathsf{r^t}]\!]$ sharing.

– Verifiers locally set $\delta_{xy} = \sigma_x \cdot \sigma_y$ and compute $\delta^2_{xy} = \delta_{xy} - \delta^1_{xy}$, where δ^1_{xy} is collectively sampled by parties in $\mathbf{V}$ and E_1. Parties then execute $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{V}_2, \delta^2_{\mathsf{xy}}, \mathsf{E}_2, \mathsf{E}_1)$, such that E_2 receives δ^2_{xy}.

– Parties in $\mathbf{V}$ and E_1 collectively sample Δ_1. Parties V_1 and E_1 compute $\mathsf{A}_1 = -\mu^1_{\mathsf{x}}\sigma^1_{\mathsf{y}} - \mu^1_{\mathsf{y}}\sigma^1_{\mathsf{x}} + \delta^1_{\mathsf{xy}} - \mathsf{r}_1 + \Delta_1$ and execute $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_1, \mathsf{A}_1, \mathsf{E}_2, \mathsf{V}_2)$, such that E_2 receives A_1.

– Similarly, parties in $\mathbf{V}$ and E_2 collectively sample Δ_2. Parties V_1 and E_2 compute $\mathsf{A}_2 = -\mu^1_{\mathsf{x}}\sigma^2_{\mathsf{y}} - \mu^1_{\mathsf{y}}\sigma^2_{\mathsf{x}} + \delta^2_{\mathsf{xy}} - \mathsf{r}_2 + \Delta_2$ and execute $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_2, \mathsf{A}_2, \mathsf{E}_1, \mathsf{V}_2)$, such that E_1 receives A_2.

– Parties V_2 and E_1 compute $\mathsf{B}_1 = -\mu^2_{\mathsf{x}}\sigma^1_{\mathsf{y}} - \mu^2_{\mathsf{y}}\sigma^1_{\mathsf{x}} - \Delta_1$ and execute $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_1, \mathsf{B}_1, \mathsf{E}_2, \mathsf{V}_1)$. Similarly, V_2 and E_2 compute $\mathsf{B}_2 = -\mu^2_{\mathsf{x}}\sigma^2_{\mathsf{y}} - \mu^2_{\mathsf{y}}\sigma^2_{\mathsf{x}} - \Delta_2$ and execute $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_2, \mathsf{B}_2, \mathsf{E}_1, \mathsf{V}_1)$.

– Evaluators compute $\mathsf{z} - \mathsf{r} = \mu_x\mu_y + \mathsf{A}_1 + \mathsf{A}_2 + \mathsf{B}_1 + \mathsf{B}_2$ and truncate it by d bits to obtain $(\mathsf{z}-\mathsf{r})^{\mathsf{t}}$.

– Parties execute $\Pi_{\mathsf{cSh}}(\mathbf{E}, (\mathsf{z}-\mathsf{r})^{\mathsf{t}})$ to generate $[\![(\mathsf{z}-\mathsf{r})^{\mathsf{t}}]\!]$ sharing and locally add to obtain $[\![\mathsf{z^t}]\!] = [\![(\mathsf{z}-\mathsf{r})^{\mathsf{t}}]\!] + [\![\mathsf{r^t}]\!]$

Fig. 9. $\Pi^{\mathsf{A}}_{\mathsf{mulTr}}(\mathsf{x}, \mathsf{y})$: Truncation Protocol

5.5 Bit Conversioni_

在5.3节中，协议Π_{msb}求得的是Boolean shares。但是计算得到Boolean shares之后，ML中后续的任务往往又涉及到乘法等算术计算。因此，需要将Bit的Boolean shares转化为等价的Arithmetic shares。对于比特 b，有：

$$b = \mu_b \oplus \sigma_b = \mu_b' + \sigma_b' - 2\mu_b'\sigma_b'$$

其中，μ_b'和σ_b'是对应比特值的算术表示，明文持有他们的参与者（E中两方持有μ_b，V中两方持有σ_b）可以在本地进行转化。因此，难点在于计算最后一个乘法。利用前文设计的Π_{cSh}协议和Π_{mult}，可以完成Bit Conversion。具体协议Π_{btr}如下。

- **Input:** Parties input their $[\![b]\!]^{\mathbf{B}}$ shares.
- **Output:** Parties obtain $[\![b]\!]$ as the output.

– Parties execute $\Pi_{\mathsf{cSh}}(\mathbf{V}, \sigma_{b'})$ and $\Pi_{\mathsf{cSh}}(\mathbf{E}, \mu_{b'})$ to generate $[\![\sigma_{b'}]\!]$ and $[\![\mu_{b'}]\!]$ respectively.

– Parties execute $\Pi_{\mathsf{mult}}([\![\mu_{b'}]\!], [\![\sigma_{b'}]\!])$ to generate $[\![\mu_{b'}\sigma_{b'}]\!]$, followed by locally computing $[\![b]\!] = [\![\mu_{b'}]\!] + [\![\sigma_{b'}]\!] - 2\,[\![\mu_{b'}\sigma_{b'}]\!]$.

Fig. 10. $\Pi_{\mathsf{btr}}([\![\mathsf{b}]\!]^{\mathbf{B}})$: Conversion of a bit to arithmetic equivalent

5.6 Bit Insertion

给定Boolean shared的比特 $[\![b]\!]^{\mathbf{B}}$和Arithmetic shared的$[\![x]\!]$，求$[\![bx]\!]$在ML计算中是很常见的一种操作，比如ReLU。一种直接的方法可以首先调用$\Pi_{btr}([\![b]\!]^{\mathbf{B}})$实现Bit Conversion然后再调用协议$\Pi_{mult}$实现乘法。进一步，本文提出了如下方法减少通信量和通信轮数。具体来说，对于

ParseError: KaTeX parse error: Too many tab characters: & at position 287: … + \sigma_{bx} &= \gamma_{b'x} …

其中，$\gamma_{b'x} = \mu_{b'}\mu_x, \delta_{b'x} = \sigma_{b'}\sigma_x$。如此，参与方可以首先对于$\mathrm{V}$生成关于$\mu_{b'}$和$\gamma_{b'x}$的$[\cdot]$-shares，对于$\mathrm{E}$生成关于$\sigma_{b'}$和$\delta_{b'x}$的$[\cdot]$-shares，如此参与方可以本地计算$A_1, A_2, B_1, B_2$（每一项都被2个参与方共有）。最后，调用$\Pi_{bic}$实现传输并计算最终的$\mu_{bx}$。具体协议$\Pi_{bin}$如下。

- **Input:** Parties input their $[\![b]\!]^{\mathbf{B}}$ and $[\![x]\!]$ shares.
- **Output:** Parties obtain $[\![bx]\!]$ as the output.

– Parties in $\mathbf{V}$ and E_1 collectively sample random $\sigma^1_{bx} \in \mathbb{Z}_{2^\ell}$, while parties in $\mathbf{V}$ and E_2 together sample random σ^2_{bx}.

– Parties in $\mathbf{V}$ and E_1 collectively sample random $\sigma^1_{b'}$ followed by V_1 and V_2 setting $\sigma^2_{b'} = \sigma_{b'} - \sigma^1_{b'}$. Parties then execute $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{V}_2, \sigma^2_{b'}, \mathsf{E}_2, \mathsf{E}_1)$, such that E_2 receives $\sigma^2_{b'}$. The same procedure is used for E_2 to receive $\delta^2_{b'x}$.

– Parties in $\mathbf{E}$ and V_1 collectively sample random $\mu^1_{b'}$ followed by E_1 and E_2 setting $\mu^2_{b'} = \mu_{b'} - \mu^1_{b'}$. Parties then execute $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu^2_{b'}, \mathsf{V}_2, \mathsf{V}_1)$, such that V_2 receives $\mu^2_{b'}$. The same procedure is used for V_2 to receive $\gamma^2_{b'x}$.

– Parties in $\mathbf{V}$ and E_1 collectively sample Δ_1. Parties V_1 and E_1 compute $\mathsf{A}_1 = -\mu^1_{b'}\sigma^1_x + (\mu^1_x - 2\gamma^1_{b'x})\sigma^1_{b'} + (2\mu^1_{b'} - 1)\delta^1_{b'x} + \sigma^1_{bx} + \Delta_1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_1, \mathsf{A}_1, \mathsf{E}_2, \mathsf{V}_2)$.

– Similarly, parties in $\mathbf{V}$ and E_2 collectively sample Δ_2. Parties V_1 and E_2 compute $\mathsf{A}_2 = -\mu^1_{b'}\sigma^2_x + (\mu^1_x - 2\gamma^1_{b'x})\sigma^2_{b'} + (2\mu^1_{b'} - 1)\delta^2_{b'x} + \sigma^2_{bx} + \Delta_2$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_1, \mathsf{E}_2, \mathsf{A}_2, \mathsf{E}_1, \mathsf{V}_2)$.

– Parties V_2 and E_1 compute $\mathsf{B}_1 = -\mu^2_{b'}\sigma^1_x + (\mu^2_x - 2\gamma^2_{b'x})\sigma^1_{b'} + (2\mu^2_{b'} - 1)\delta^1_{b'x} - \Delta_1$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_1, \mathsf{B}_1, \mathsf{E}_2, \mathsf{V}_1)$. Similarly, V_2 and E_2 compute $\mathsf{B}_2 = -\mu^2_{b'}\sigma^2_x + (\mu^2_x - 2\gamma^2_{b'x})\sigma^2_{b'} + (2\mu^2_{b'} - 1)\delta^2_{b'x} - \Delta_2$ and invoke $\Pi_{\mathsf{bic}}(\mathsf{V}_2, \mathsf{E}_2, \mathsf{B}_2, \mathsf{E}_1, \mathsf{V}_1)$.

– Evaluators compute $\mu_{b'x} = \mathsf{A}_1 + \mathsf{A}_2 + \mathsf{B}_1 + \mathsf{B}_2 + \gamma_{b'x}$ locally. Parties in $\mathbf{E}$ and V_1 collectively sample $\mu^1_{b'x}$ followed by evaluators setting $\mu^2_{b'x} = \mu_{b'x} - \mu^1_{b'x}$ and invoking $\Pi_{\mathsf{bic}}(\mathsf{E}_1, \mathsf{E}_2, \mu^2_{b'x}, \mathsf{V}_2, \mathsf{V}_1)$.

Fig. 11. $\Pi_{\mathsf{bin}}([\![\mathsf{b}]\!]^{\mathbf{B}}, [\![x]\!])$: Insertion of bit b in a value

6. Evaluation

本文做了关于关键模块的性能测试，进一步进行了ML模型的性能实验。对于ML中的算子，矩阵乘法、卷积可以归约到向量乘法，$\mathrm{Sigmoid}$可以用分段函数计算（分段方法和SecureML一样），而ReLU则使用可以先做再求乘积。实验结果和ABY3进行比较。

6.1 模块实验

1）Dot Product：

首先在固定向量长度下比较FLASH和ABY3的Latency；

Work	LAN Latency (ms)	WAN Latency (s)
ABY3	3.55	1.10
FLASH	1.51	1.08

Table 5. Latency of 1 dot product computation for 784 features

其次，比较性能在特征增加时带来的Latency变化。

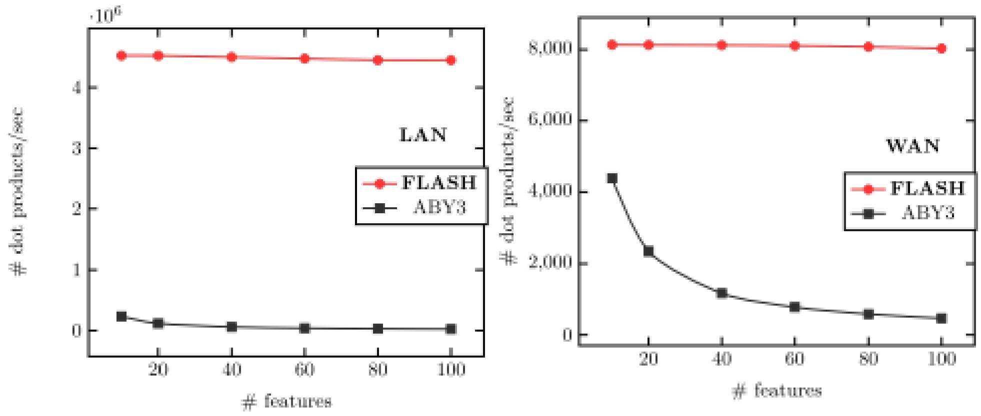

Fig. 12. # of dot product computations with increasing features.

2) MSB Extraction

首先比较一次协议调用的Latency：

Work	LAN Latency (ms)	WAN Latency (s)
ABY3	3.53	2.29
FLASH	1.77	1.31

Table 6. Latency for single execution of MSB Extraction protocol

进一步，比较多次调用的Latency增长趋势：

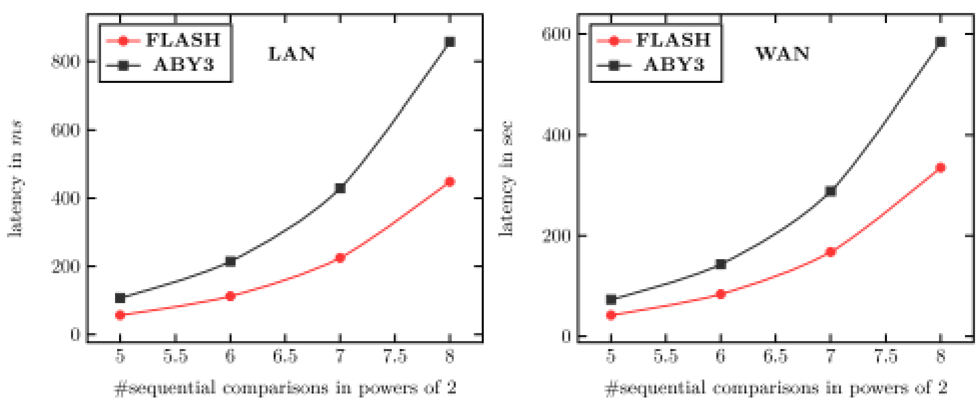

Fig. 13. Latency with increasing sequential comparisons

3) Truncation

Latency 和 Throughput 比较如下：

Work	LAN Latency (ms)	WAN Latency (s)
ABY3	1.52	1.11
FLASH	1.51	1.07

Table 7. Latency for a single execution of Truncation protocol

Work	LAN		WAN	
	#mult/sec	Improv.	#mult/min	Improv.
ABY3	0.45M	$8.8\times$	4.76M	$8.81\times$
FLASH	3.97M		0.54M	

Table 8. Throughput Comparison wrt # multiplications with truncation

4）ML Evaluation

首先比较关于线性回归和Logistic回归的Latency。

Setting	# Features	Ref.	Linear Reg.	Logistic Reg.
LAN (ms)	10	ABY3	1.68	5.59
		FLASH	1.51	3.26
	100	ABY3	2.03	5.94
		FLASH	1.51	3.26
	1000	ABY3	3.63	7.54
		FLASH	1.52	3.27
WAN (sec)	10/100/1000	ABY3	1.11	3.78
		FLASH	1.08	2.46

Table 9. Latency of frameworks for Linear and Logistic Reg.

接下来，比较两种回归的Throghput。

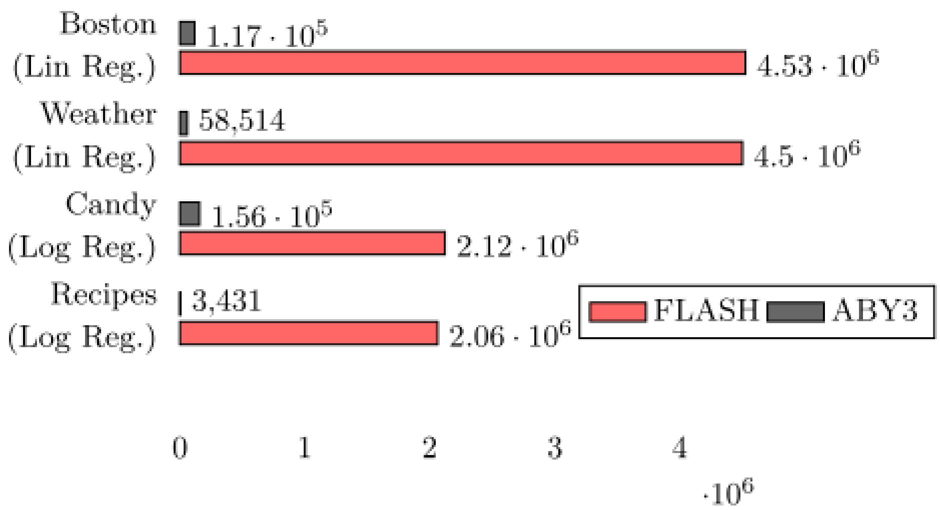

Fig. 14. Throughput Comparison (# queries/sec) for Linear and Logistic Regression in LAN setting

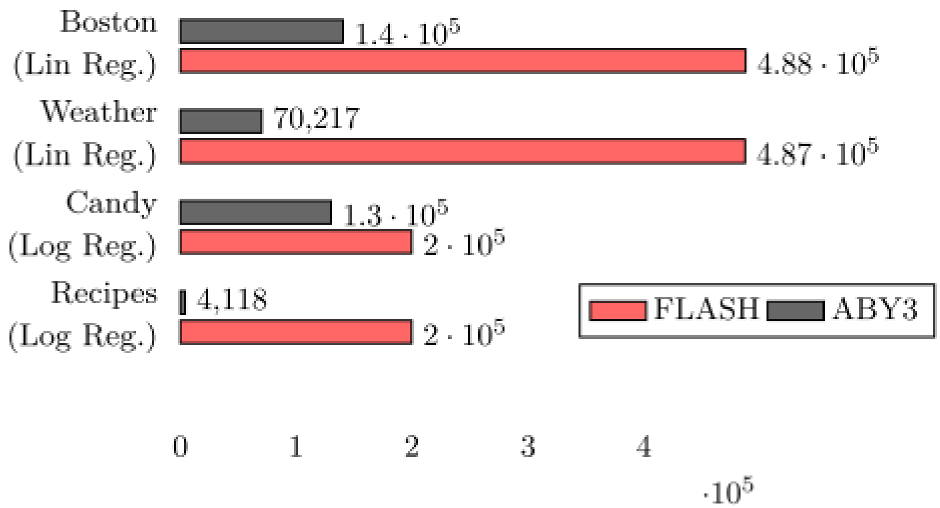

Fig. 15. Throughput Comparison (# queries/min) for Linear and Logistic Regression in WAN setting

最后比较NN模型的Latency和Throghtput。

Setting	# Features	Ref.	DNN	BNN
LAN (ms)	10	ABY3	59.71	59.73
		FLASH	18.65	23.37
	100	ABY3	67.78	67.77
		FLASH	18.74	23.69
	1000	ABY3	146.37	147.36
		FLASH	19.06	23.80
WAN (*sec*)	10/100/1000	ABY3	13.56	13.56
		FLASH	11.24	13.68

Table 10. Latency of frameworks for DNN and BNN

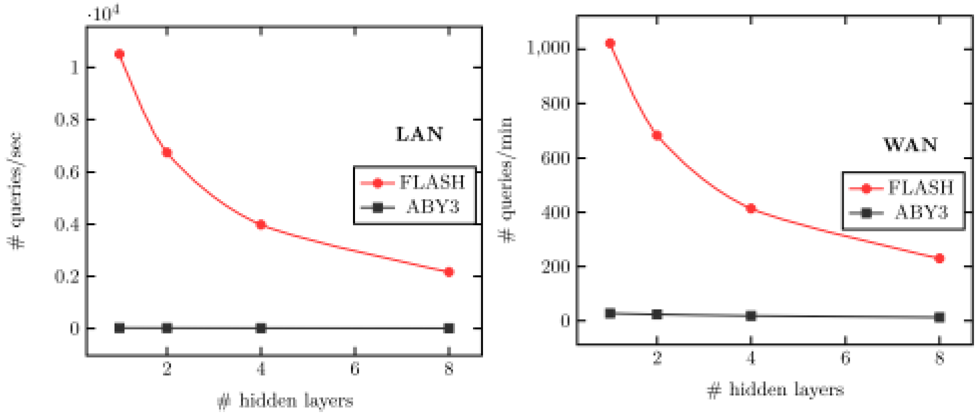

Fig. 16. Throughput Comparison for DNN with increasing number of hidden layers.

7. 结论

本文是比较早的一项实现GOD安全性的安全ML的工作，而且不再需要用广播。对后来的工作有很多借鉴意义。但是，也有许多需要改进之处，例如如何在实现GOD的情况下实现Privacy尚未得到很好的解决。