Look-Up Table：原理、实现和应用

安全多方计算协议可以分别通过秘密分享（GMW系列）和混乱电路（GC系列）构造，两种方案在通信轮数和通信量方面各有优劣。以布尔电路为例：

- GMW：该系列方案通信量少，但是每一层与门计算需要一轮交互，因此通信轮数和电路深度成正比；
- GC：该系列方案通信轮数是常数轮，但是要传输混乱表，因此通信量较大。
  因而，GMW系列方案适合应用在低延迟网络中以获得较大的吞吐量，而GC系列方案则适合应用在延迟较大的网络中，减少因为网络延迟带来的时间开销。根据GMW v.s. Yao中的介绍，对于深度较浅、尺寸较大的电路来说，GMW系列的方案更能达到更好的整体性能。

Look-Up Table (LUT) 是介于GMW和GC之间的一种技术，既可以有效的降低GMW方案的通信轮数，同时通信量也比GC系列的方案更优。LUT在计算复杂函数时具有广泛的应用。本文接下来的内容集中在半诚实安全下的两方计算。

0. 基础知识

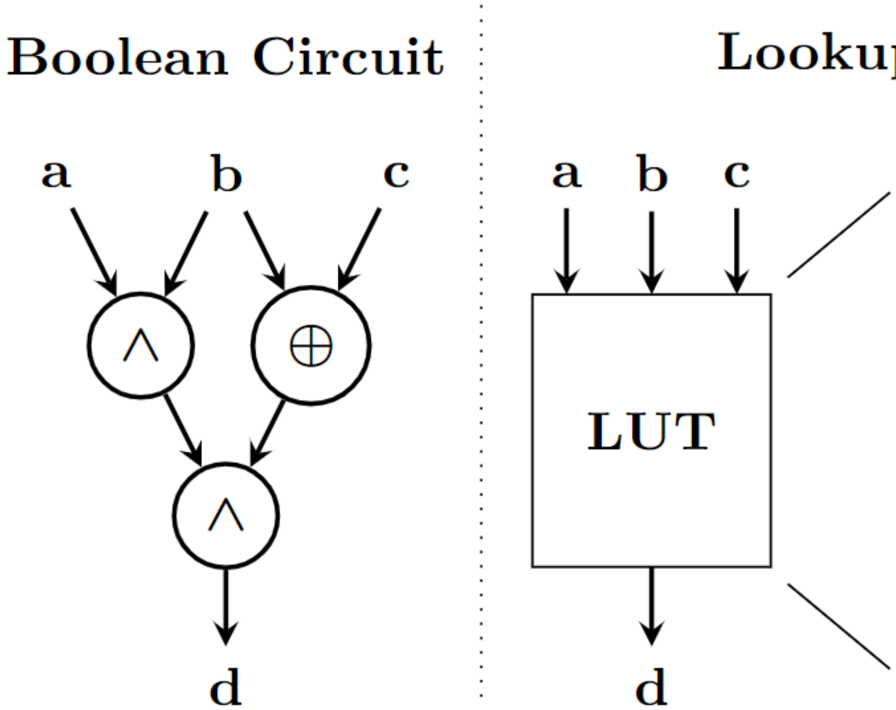
阅读本文需要简单掌握茫然传输（OT，尤其是N选1的OT协议）、秘密分享和ABY2.0的相关知识。感兴趣的读者可以简单读一下之前的博客了解上述相关技术。

1. 基础原理

一个LUT可以看作一个多输入-多输出的布尔电路门，其将$\delta \geq 2$个输入比特映射到σ个输出比特，映射关系可以表示为任意布尔函数

$$f : \{0,1\}^{\delta} \rightarrow \{0,1\}^{\sigma}$$

如下图所示，左侧的布尔电路可以表示为右侧的LUT。如此，复杂的函数可以表示为多个LUT的组合。



a	b	c	d
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

以两方计算为例，S_0和S_1在预计算阶段调用安全计算协议生成LUT，在线阶段根据实际输提取对应的输出表项。

2. 经典实现

本节主要介绍四种经典构造方案。相关论文如下：

- IKM+13
- DKS+18
- BHS+23

2.1 OTTT方案

OTTT方案最早在论文[IKM+13]中提出，该方案的核心思想是在两方之间生成一个由随机分享偏置θ旋转的LUT：T。即给定T，预计算阶段计算之后S_0持有(T^0, r)、S_1持有(T^1, s)，使得对于任意的i有$T[i] = \mathsf{T}^0[i \oplus \theta] \oplus \mathsf{T}^1[i \oplus \theta]$且$\theta = r \oplus s$。

在线阶段，给定秘密值$x = x_0 \oplus x_1$，两方计算如下:

- $S_0 \xrightarrow{x_0 \oplus r} S_1$，$S_1 \xrightarrow{x_1 \oplus s} S_0$；
- S_0和S_1各自在本地恢复$x \oplus \theta$；
- 最后，两方各自提取$\mathsf{T}^i[x \oplus \theta]$，其中$i \in \{0, 1\}$满足$T[x] = \mathsf{T}^0[x \oplus \theta] \oplus \mathsf{T}^1[x \oplus \theta]$。

具体协议如下:

Functionality:

– P_1 has input $x \in X$, P_2 has input $y \in Y$.
– Both parties learn $z = f(x, y)$.

Preprocessing:

1. Sample random $r \in X, s \in Y$ and let A be the permuted truth table; i.e.,
$$A_{x+r, y+s} = f(x, y) \ ;$$
2. Sample a random matrix $M^1 \in Z^{X \times Y}$ and let $M^2 = A - M^1$;
3. Output (M^1, r) to P_1 and (M^2, s) to P_2;

Protocol:

1. P_1 sends $u = x + r$ to P_2;
2. P_2 sends $v = y + s$ and $z_2 = M^2_{u,v}$ to P_1;
3. P_1 sends to P_2 the value $z_1 = M^1_{u,v}$;
4. Both parties output $z = z_1 + z_2$;

Fig. 2. Semi-Honest Secure Protocol using One-Time Truth Table

方案OTTT预计算通信需要$(|MT| + 4) \cdot (\delta - 1) \cdot 2^{\delta}\sigma$比特的通信，在线阶段传输$2\delta$比特。其中，$|MT|$表示两方生成布尔乘法三元组的通信开销。

2.2 OP-LUT方案

论文[DKS+18]提出了OP-LUT和SP-LUT两种方案。在上述OTTT方案的基础上，OP-LUT的主要优化目的在于保持在线通信不增加的情况下，进一步减少预计算的通信开销。具体来说，OP-LUT的预计算协议构造如下：

- S_0在本地选择自己的份额T^0；
- 进一步，对于每一个S_1可能选择的$s \in \{0, 1\}^{\delta}$，S_0都利用T^0和s计算相应的T^1。如此，S_0会计算得到2^{δ}个T^1，每个T^1的大小是$2^{\delta}\sigma$比特（2^{δ}行、σ列）；
- 最后，两方进行一次$\binom{2^{\delta}}{1} - \mathsf{OT}^1_{2^{\delta}\sigma}$即可，其中$S_0$的输入为$2^{\delta}$个$\mathsf{T}^1$，$S_1$的输入为$s$。

在线计算阶段则和OTTT方案一致。除了上述优化之外，[DKS+18]还对OT进行了优化，细节请参阅原文。协议细节见下图:

PROTOCOL 3 (Online-LUT (OP-LUT) - our work)

Inputs and Oracles:

- **Common Input:** Symmetric security parameter κ; number of inputs δ; $N = 2^\delta$; Truth-table $T : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$.
- **Input of P_0:** $x^0 \in \{0,1\}^\delta$.
- **Input of P_1:** $x^1 \in \{0,1\}^\delta$.
- **Oracles:** Both parties have access to a $\binom{N}{1}$ $\mathrm{OT}^1_{\sigma N}$ functionality.

Pre-Computation:

1. P_0 chooses $r \in_R \{0,1\}^\delta$ and $T^0 \in_R (\{0,1\}^\delta \mapsto \{0,1\}^\sigma)$. P_1 chooses $s \in_R \{0,1\}^\delta$.
2. P_0 computes $(X_0, ..., X_{N-1})$, with $X_{s'}[i] = T[r \oplus s' \oplus i] \oplus T^0[i]$, for all $0 \leq i, s' < N$.
3. P_0 and P_1 invoke the $\binom{N}{1}$ $\mathrm{OT}^1_{\sigma N}$ functionality where P_0 plays the sender with inputs $(X_0, ..., X_{N-1})$ and P_1 plays the receiver with input s and output $T^1 = X_s$ s.t. $X_s[i] = T[r \oplus s \oplus i] \oplus T^0[i]$, for all $0 \leq i < N$.
4. **Output:** P_0 outputs (T^0, r); P_1 outputs (T^1, s).

Online Evaluation (same as OTTT in Prot. 2):

1. P_0 sends $u = x^0 \oplus r$ to P_1; P_1 sends $v = x^1 \oplus s$ to P_0.
2. P_0 sets $z^0 = T^0[u \oplus v]$; P_1 sets $z^1 = T^1[u \oplus v]$.
3. **Output:** P_0 outputs z^0; P_1 outputs z^1, s.t. $z^0 \oplus z^1 = T[x^0 \oplus x^1]$.

方案OP-LUT预计算通信需要$|\binom{2^\delta}{1} - \mathrm{OT}^1_{2^\delta \sigma}| - \delta$比特的通信，在线阶段传输$2\delta$比特。

2.3 SP-LUT方案

与方案OTTT和OP-LUT不同的是，SP-LUT方案致力于提升整体通信效率，但代价是增加了在线计算的通信开销。具体来说，SP-LUT在线计算阶段通过N选1的OT协议实现LUT的计算。具体来说，S_0首先计算针对S_1所有可能输入对应的输出（2^δ种结果），然后利用$|\binom{2^\delta}{1} - \mathrm{OT}^1_\sigma$实现计算。该方法可以调用随机OT（rOT）进行优化。具体协议见下图。

PROTOCOL 4 (Setup-LUT (SP-LUT) - our work)

Inputs and Oracles:

- **Common Input:** Symmetric security parameter κ; number of inputs δ; $N = 2^\delta$; Truth-table $T : \{0,1\}^\delta \mapsto \{0,1\}^\sigma$.
- **Input of P_0:** $x^0 \in \{0,1\}^\delta$.
- **Input of P_1:** $x^1 \in \{0,1\}^\delta$.
- **Oracles:** Both parties have access to a $\binom{N}{1}$ R-OT$^1_\sigma$ functionality.

Pre-Computation:

1. P_0 and P_1 invoke the $\binom{N}{1}$ R-OT$^1_\sigma$ functionality where P_0 plays the sender and P_1 plays the receiver. From the OT, P_0 receives random bits $(m_0, ..., m_{N-1})$ and P_1 receives a random choice $s \in \{0,1\}^\delta$ and message m_s.
2. **Output:** P_0 outputs $(m_0, ..., m_{N-1})$; P_1 outputs (m_s, s).

Online Evaluation:

1. P_1 sends $u = s \oplus x^1$ to P_0.
2. P_0 chooses $z^0 \in_R \{0,1\}^\sigma$ and computes and sends $V = (v_0, ..., v_{N-1})$, where $v_i = T[i \oplus x^0] \oplus m_{i \oplus u} \oplus z^0$.
3. P_1 computes $z^1 = v_{x^1} \oplus m_s$.
4. **Output:** P_0 outputs z^0; P_1 outputs z^1, s.t. $z^0 \oplus z^1 = T[x^0 \oplus x^1]$.

方案SP-LUT预计算通信需要$|\binom{2^\delta}{1}-\mathrm{rOT}^1_\sigma|$比特的通信，在线阶段传输$\delta + 2^\delta\sigma$比特。

2.4 FLUTE方案

和之前的方案不同，论文[BHS+23]提出的FLUTE将LUT的计算表示为向量内积计算，并利用ABY2.0的两方掩码秘密分享实现了高效的在线效率。方案构造逻辑如下：

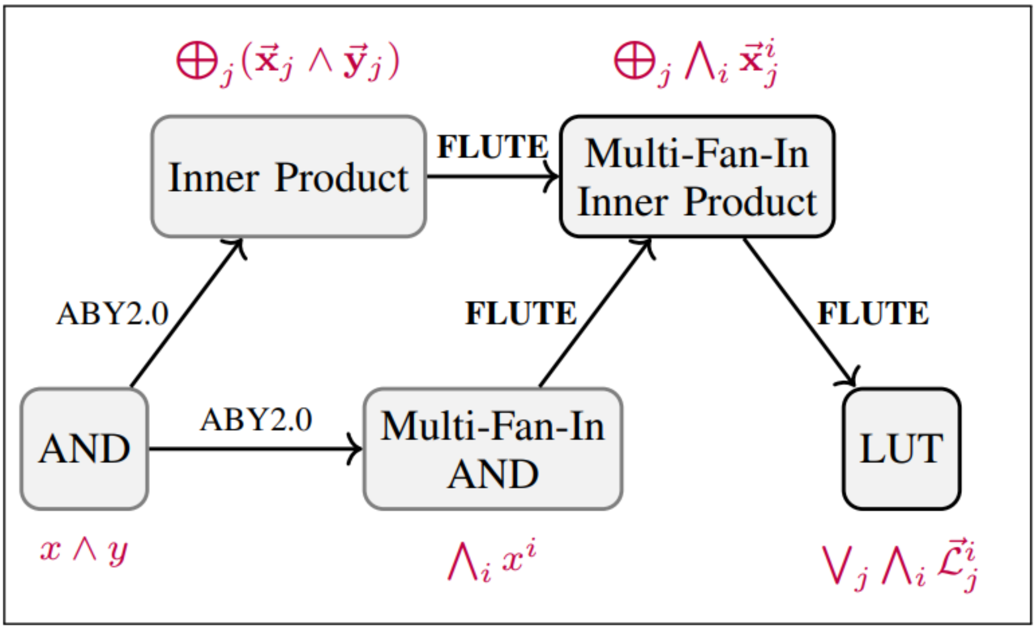


Figure 4: Roadmap of FLUTE protocol design starting with the known primitives in ABY2.0 [68]. Each node's functionality is provided alongside it.

其方案主要设计思想如下图所示，简单分析如下：

- 图(a)种LUT的计算可以表示图(b)的布尔表达式，即输出项为1的向量的OR；
- 进一步，由于(b)中不同的子布尔表达式不可能同时为1，所以OR操作可以简化为XOR，从而将表达式简化为(c)中布尔表达式；
- 最后，(c)中的布尔表达式可以形式化为(d)中的布尔向量内积形式。

x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

(a) LUT T with $\delta = 3$

$$\begin{gathered}\text{LUT output } z \\ = \\ (\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \\ \vee(\overline{x_1} \wedge x_2 \wedge x_3) \\ \vee(x_1 \wedge \overline{x_2} \wedge x_3)\end{gathered}$$

(b) LUT Output Description

$$\begin{aligned}&(\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \\ \oplus&(\overline{x_1} \wedge x_2 \wedge x_3) \\ \oplus&(x_1 \wedge \overline{x_2} \wedge x_3)\end{aligned}$$

(c) OR to XOR Conversion

$$\begin{gathered} y = \vec{\mathcal{L}}^1 \odot \vec{\mathcal{L}}^2 \odot \vec{\mathcal{L}}^3 \\ = \\ \begin{pmatrix}\overline{x_1} \\ \overline{x_1} \\ x_1\end{pmatrix} \cdot \begin{pmatrix}\overline{x_2} \\ x_2 \\ \overline{x_2}\end{pmatrix} \cdot \begin{pmatrix}\overline{x_3} \\ x_3 \\ x_3\end{pmatrix}\end{gathered}$$

(d) Output Computation

除了上述主要思路，具体构造方面FLUTE进行了如下设计：

- 对于向量$\vec{\mathcal{L}}^i$，其第j个元素编码为$\vec{\mathcal{L}}^i_j = x_i \oplus (1 \oplus \vec{\mathcal{E}}^i_j)$，其中$i \in [\delta], j \in [2^\delta]$。而$\mathcal{E}^i$即为$x_i$在表T中的编码。如上图，$\vec{\mathcal{E}}^1 = [0,0,0,0,1,1,1,1]^T$。
- 为了优化多输入向量内积的在线计算通信效率，FLUTE底层采用了ABY2.0的两方掩码秘密分享方案，从而减少了在线计算的开销。
- 对于预计算，ABY2.0需要计算掩码随机数的安全乘积。由于LUT只是针对布尔值进行，因此x_i和$\bar{x}_i$的掩码分享中的随机数部分是相同的，区别只有翻转的公开值部分。因此预计算中不同$\vec{\mathcal{L}}^i$的随机掩码的乘积计算只需要计算其中一个即可，即$\lambda_{\vec{\mathcal{L}}^i_j} = \lambda_{x_i}$，从而使得预计算通信开销比简单直接调用ABY2.0预计算方案减少了2^δ倍。
- 同时，FLUTE使用Silent OT实现预计算的关联随机数生成、两方乘法等操作，比使用IKNP OT进一步提升了实用性。

FLUTE执行LUT的详细协议如下：

Protocol $\Pi_{\mathsf{LUT}}((\langle x_1\rangle, ..., \langle x_\delta\rangle), \mathsf{T})$

Input: LUT T for a function $f : \{0,1\}^\delta \to \{0,1\}^\sigma$, with inputs $\langle x_k\rangle$ and input encoding $\vec{\mathcal{E}}^k \in \{0,1\}^{2^\delta}$, for $k \in [\delta]$, and output encoding $\vec{\mathbf{y}}^w \in \{0,1\}^{2^\delta}$, for $w \in [\sigma]$.

Output: $\langle \vec{\mathbf{z}} \rangle$, where $\vec{\mathbf{z}} = (\vec{\mathbf{z}}_1, ..., \vec{\mathbf{z}}_\sigma) = \mathsf{T}[x_1, ..., x_\delta]$.

Setup Phase:

1. Server S_i, for $i \in \{0,1\}$ samples random $\lambda^i_{\vec{\mathbf{z}}} \in \{0,1\}^\sigma$.
2. Let $\mathcal{I} = \{x_1, \ldots, x_\delta\}$. Then:
   - Execute $\mathcal{F}^{\mathsf{pre}}_{\mathsf{ANDM}}$ to generate $[\lambda_\mathcal{Q}]$ for all $\mathcal{Q} \in 2^\mathcal{I}$.

Online Phase:

1. *Input Preparation:*
   - Locally set $\vec{\mathcal{L}}^i_j = x_i \oplus (1 \oplus \vec{\mathcal{E}}^i_j), \quad \forall i \in [\delta], \forall j \in [2^\delta]$.
   - Locally set $\mathcal{I}_j = \{\vec{\mathcal{L}}^1_j, \ldots, \vec{\mathcal{L}}^\delta_j\}, \quad \forall j \in [2^\delta]$.
2. Server S_i, for $i \in \{0,1\}$ and $w \in [\sigma]$, locally computes
$$[\vec{\mathbf{v}}_w]_i = \bigoplus_{\mathcal{Q} \in 2^\mathcal{I}, \mathcal{Q} \neq \mathcal{I}} \Big((\vec{\mathsf{m}}_\mathcal{Q} \odot \vec{\mathbf{y}}^w) \wedge \lambda_{\mathcal{I} \setminus \mathcal{Q}} \Big) \oplus \lambda^i_{\vec{\mathbf{z}}_w}.$$
3. Servers mutually exchange $[\vec{\mathbf{v}}]$ to obtain $\vec{\mathbf{v}} = [\vec{\mathbf{v}}]_0 \oplus [\vec{\mathbf{v}}]_1$.
4. Locally compute $\mathsf{m}_{\vec{\mathbf{z}}_w} = \vec{\mathbf{v}}_w \oplus (\vec{\mathsf{m}}_\mathcal{I} \odot \vec{\mathbf{y}}^w)$.

Figure 6: FLUTE Lookup Table Evaluation

方案FLUTE预计算通信需要$(|MT| + 4) \cdot (2^\delta - \delta - 1)$比特的通信，在线阶段传输$2\delta$比特。其中，$|MT|$表示两方生成布尔乘法三元组的通信开销。

2.5 方案对比

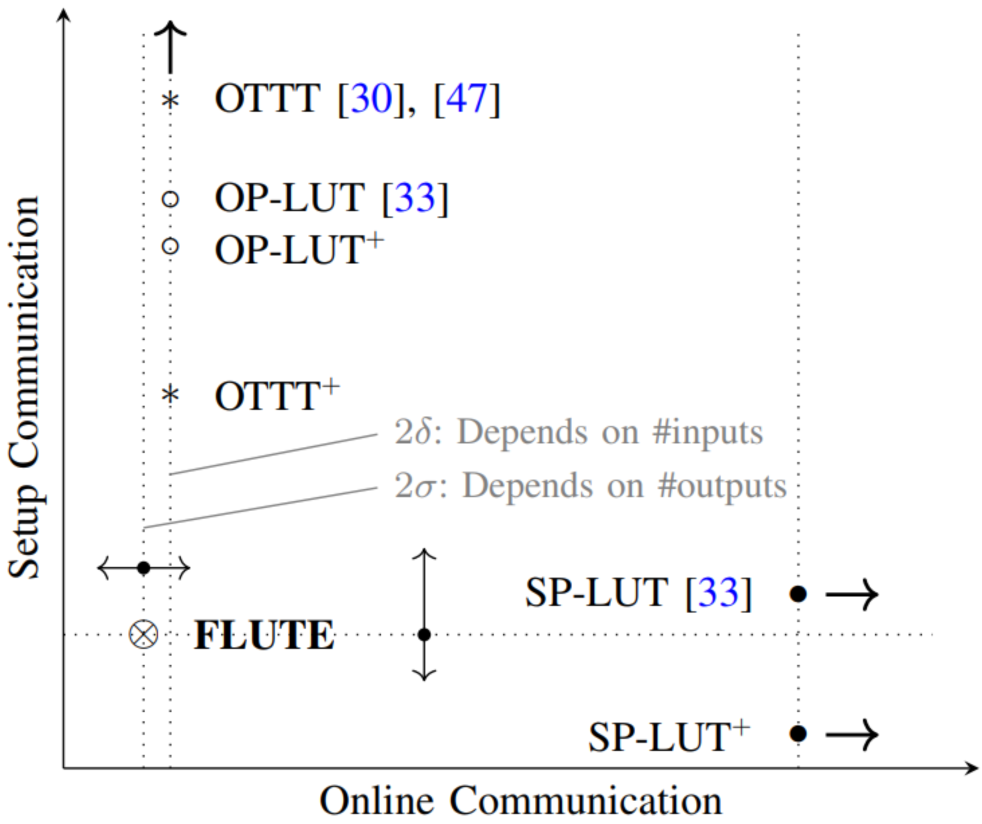


Figure 1: Comparison of setup and online communication between prior LUT protocols and FLUTE. + denotes the modification of prior LUTs that use silent OT [17]. The specific communication of FLUTE relative to other approaches depends on the LUT parameters: number of inputs δ and number of outputs σ. Exact costs depending on the given parameters are consolidated in Table 2.

如上图所示，FLUTE一文中对比了上述不同方案的预计算和在线计算通信开销，可以看到OTTT和OP-LUT在线同核心优于SP-LUT，但SP-LUT整体性能更优。而FLUTE则在预计算和在线计算两部分都达到了目前最好的通信效率。更多的实验对比可以参考FLUTE原文。

3. 应用示例

LUT可以用在多种应用中，且适合计算非线性函数。在隐私保护机器学习中，以SIRNN提出的指数计算协议为例，本文简单介绍一下LUT的应用：给定输入x，将x按照2^d进制进行分解，则有$e^x = e^{\sum_{i=0}^{k-1} 2^{di} x_i}$。如此，对于$x$分解后的每一个$d$比特的分段，可以构造一个LUT表：

$$L_i : \{0,1\}^d \to \mathbb{Z}_{2^{s'+2}}, \text{ s.t. } L_i[j] = \mathsf{Fix}(\mathsf{rExp}(2^{di-s} j), s'+2, s')$$

如此，两方可以计算每一个$L_i[x_i]$得到每个分段的指数计算结果，最后做乘法计算$\mathsf{rExp}(x) = L_{k-1}[x_{k-1}] \cdot \cdots \cdot L_0[x_0]$。最后截断、设置比特位长。具体算法如下：

Functionality $\mathcal{F}_{\mathsf{rExp}}^{m,s,n,s'}(\langle x\rangle^m)$

1) Let $x = x_{k-1}||\ldots||x_0$, $x_i \in \{0,1\}^d$, $i \in [k]$, $dk = m$.
2) For $i \in [k]$, let $L_i : \{0,1\}^d \to \mathbb{Z}_{2^{s'+2}}$ s.t. $L_i(j) = \mathsf{Fix}\left(\mathsf{rExp}(2^{di-s}j), s'+2, s'\right)$.
3) Compute $g = L_{k-1}[x_{k-1}] * \ldots * L_0[x_0]$, g has bitwidth $s'+2$ and scale s'.
4) Return $\langle y\rangle^n$ for $y = \mathsf{SExt}(g, s'+2, n)$.

关于上述协议其他的细节介绍，请参考我们之前的关于SIRNN的博客。