Security Proof of FLUTE

카이스트 전산학부 한승우

2025년 3월 26일

Notations

- Let $\mathcal{Q}$ be a nonempty set of $\langle\cdot\rangle$-shared values (wires). We use the following notation:

$$\mathsf{m}_{\mathcal{Q}} \triangleq \bigwedge_{u\in\mathcal{Q}} \mathsf{m}_u \quad \text{and} \quad \lambda_{\mathcal{Q}} \triangleq \bigwedge_{u\in\mathcal{Q}} \lambda_u.$$

Lookup Table

LUT (Lookup Table)

δ-to-σ lookup table T is a function T: $\{0,1\}^{\delta} \rightarrow \{0,1\}^{\sigma}$.

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

Example 3-to-1 LUT T

$$(\overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}) \\ \oplus(\overline{x_1} \wedge x_2 \wedge x_3) \\ \oplus(x_1 \wedge \overline{x_2} \wedge x_3)$$

DNF Representation of T

$$y = \begin{pmatrix} \overline{x_1} \\ \overline{x_1} \\ x_1 \end{pmatrix} \odot \begin{pmatrix} \overline{x_2} \\ x_2 \\ \overline{x_2} \end{pmatrix} \odot \begin{pmatrix} \overline{x_3} \\ x_3 \\ x_3 \end{pmatrix}$$

"Multi-Fan-In Inner Product" representation

$[\cdot]$ Secret Sharing

P_0, P_1: parties

$[\cdot]$ Secret Sharing

A bit $v \in \mathbb{Z}_2$ is said to be $[\cdot]$-*shared* between P_0 and P_1 if P_i holds $[v]_i$ such that

$$v = [v]_0 \oplus [v]_1.$$

Multiplication of $[\cdot]$-shared Values

Beaver's Multiplication Triple

Suppose P_0 and P_1 have $[\cdot]$-shares of $u, v \in \mathbb{Z}_2$. Suppose additionally that they have $[\cdot]$-shares of a, b, c where $c = ab$.

1. P_i computes $[u \oplus a]_i$ and $[v \oplus b]_i$, and sends them to P_{1-i}.
2. P_0 and P_1 now know $d := u \oplus a$ and $e := v \oplus b$.
3. P_i computes $[z]_i = i \cdot de \oplus d[b]_i \oplus e[a]_i \oplus [c]_i$.

Then, $[z]_0 \oplus [z]_1 = z = uv$.

$$\begin{aligned} z = uv &= (d \oplus a)(e \oplus b) \\ &= \underbrace{de}_{\text{public}} \oplus db \oplus ea \oplus \underbrace{ab}_{=c} \end{aligned}$$

$\langle\cdot\rangle$ Secret Sharing

$\langle\cdot\rangle$ Secret Sharing

A bit $v \in \mathbb{Z}_2$ is said to be $\langle\cdot\rangle$-*shared* if:

1. A value $\lambda_v \in \mathbb{Z}_2$ is $[\cdot]$-shared between P_0 and P_1.
2. A value $\mathrm{m}_v \in \mathbb{Z}_2$ is public.
3. $v = \mathrm{m}_v \oplus \lambda_v$

The $\langle\cdot\rangle$-share of v is denoted $\langle v\rangle_i = (\mathrm{m}_v, [\lambda_v]_i)$.

Multiplication of $[\cdot]$-Shared Values

We will use the following functionality as black-box. In other words, we assume that there is some protocol that securely realizes this functionality.

$\mathcal{F}_{\mathsf{AND}}$: Multiplication of $[\cdot]$-Shared Values (Ideal)

Input $[\cdot]$-shares of $u, v \in \mathbb{Z}_2$ from P_0 and P_1.

Output $[\cdot]$-shares of $uv \in \mathbb{Z}_2$ to each P_i

1. Recover $u, v \in \mathbb{Z}_2$ from the shares and calculate $z = uv$.
2. Sample random $[z]_0 \xleftarrow{\$} \mathbb{Z}_2$.
3. Send $[z]_0$ to P_0 and $[z]_0 \oplus z$ to P_1.

Note that both $[z]_0$ and $[z]_1$ are independent from inputs and are uniformly distributed in $\mathbb{Z}_2$.

Simulation Based Notion of Security

Definition

Let $X = \{X(a,n)\}_{a \in \{0,1\}^*, n \in \mathbb{N}}$ and $Y = \{Y(a,n)\}_{a \in \{0,1\}^*, n \in \mathbb{N}}$ be two probability ensembles. Then, we say that X and Y are *computationally indistinguishable* if, for any PPT algorithm D, and for any $a \in \{0,1\}^*$, we have

$$\Pr[D(X(a,n),a) = 1] - \Pr[D(Y(a,n),a) = 1] = \mathrm{negl}(n).$$

We write $X \stackrel{c}{\equiv} Y$ if they are computationally indistinguishable.

Simulation Based Notion of Security

Definition

Let $f = (f_1, f_2)$ be a two-party functionality. A (poly-time) protocol π *securely computes* f *against static semi-honest adversaries* if there are PPT algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\left\{\left(\mathcal{S}_1(1^n, x, f_1(x,y)), f(x,y)\right)\right\}_{x,y,n} \stackrel{c}{\equiv} \left\{\left(\mathsf{view}_1^\pi(x,y,n), \mathsf{output}^\pi(x,y,n)\right)\right\}_{x,y,n}$$

$$\left\{\left(\mathcal{S}_2(1^n, y, f_1(x,y)), f(x,y)\right)\right\}_{x,y,n} \stackrel{c}{\equiv} \left\{\left(\mathsf{view}_2^\pi(x,y,n), \mathsf{output}^\pi(x,y,n)\right)\right\}_{x,y,n}$$

where

- x and y are inputs of P_0 and P_1, respectively,
- n is the security parameter,
- $\mathsf{view}_i^\pi(x,y,n)$ is the tuple of P_i's input, incoming messages, and internal random tape, and
- $\mathsf{output}^\pi(x,y,n)$ is the output of π (of both parties).

The LUT Functionality

$\mathcal{F}_{\text{LUT}}$: LUT (Ideal)

Input $\langle\cdot\rangle$-shares of $x^1, \cdots, x^\delta \in \mathbb{Z}_2$

Output $\langle\cdot\rangle$-shares of $\mathbf{z} := \mathsf{T}(x^1, \cdots, x^\delta)$ to each user.

1. Reconstruct $x^1, \cdots, x^\delta \in \mathbb{Z}_2$ from the shares and calculate $z = \mathsf{T}(x^1, \cdots, x^\delta)$.
2. Sample random $[\lambda_z]_i \xleftarrow{\$} \mathbb{Z}_2$ for $i \in \{0, 1\}$ and set $\mathsf{m}_z := z \oplus \lambda_z$.
3. Return $(\mathsf{m}_z, [\lambda_z]_i)$ to P_i.

If $(\mathsf{m}_z, [\lambda_z]_0)$ and $(\mathsf{m}_z, [\lambda_z]_1)$ denote outputs of P_0 and P_1, respectively, then m_z, $[\lambda_z]_0$, and $[\lambda_z]_1$ are independent and unifromly distributed in $\mathbb{Z}_2$.

FLUTE Protocol

Π_{LUT}: LUT (Real)

Input $\langle\cdot\rangle$-shares of $x^1, \cdots, x^\delta \in \mathbb{Z}_2$

Output $\langle\cdot\rangle$-shares of $\mathbf{z} := \mathsf{T}(x^1, \cdots, x^\delta)$ to each user.

Setup Phase:

1. Each user samples $[\lambda_{\mathbf{z}_w}]_i \xleftarrow{\$} \mathbb{Z}_2$ for $w \in [\sigma]$ and $i \in \{0, 1\}$.
2. Each user let $\mathcal{I} := \{x^1, \cdots, x^\delta\}$ and use $\mathcal{F}_{\mathsf{AND}}$ to get shares $[\lambda_{\mathcal{Q}}]_i$ for $\varnothing \neq \mathcal{Q} \subseteq \mathcal{I}$.

Online Phase:

1. P_i computes its share of $\mathbf{u}_k^j = \overline{x^j} \oplus \mathbf{e}_k^j$ for $j \in [\delta]$ and $k \in [2^\delta]$.
2. P_i computes, for each $w \in [\sigma]$,
$$[\mathbf{z}_w]_i = i \cdot \bigoplus_{k=1}^{2^\delta} (\mathbf{y}_k^w \cdot \mathsf{m}_{\mathcal{I}_k}) \oplus \bigoplus_{k=1}^{2^\delta} \left[\mathbf{y}_k^w \cdot \bigoplus_{\mathcal{Q} \subsetneq \mathcal{I}_k} \left(\mathsf{m}_{\mathcal{Q}} \cdot [\lambda_{\mathcal{I}_k \setminus \mathcal{Q}}]_i \right) \right].$$
3. P_i computes $[\mathsf{m}_{\mathbf{z}_w}]_i = [\mathbf{z}_w]_i \oplus [\lambda_{\mathbf{z}_w}]_i$ for $w \in [\sigma]$ and sends them to P_{1-i}.

Building the Simulator

Building the simulator $\mathcal{S}_0$ when P_0 is corrupt

Input 1^n, $\langle x^j \rangle_0$ for $j \in [\delta]$, and $(\mathsf{m}_{\mathbf{z}_w}, [\lambda_{\mathbf{z}_w}]_0)$ for $w \in [\sigma]$.

Output Simulation of P_0's view. In other words, $(\langle \mathbf{x} \rangle_0, r; (\text{transcripts from } \mathcal{F}_{\mathsf{AND}}), [\mathsf{m}_{\mathbf{z}}]_1)$ mimicking P_0's view. P_0's incoming messages

1. Randomly fix a random tape r for P_0.
2. Imagine P_1 with arbitrary inputs in the head. For instance, inputs of P_1 are all zero.
3. Execute Π_{LUT} with the imaginary party P_1 except:
4. When P_1 is about to send its $[\mathsf{m}_{\mathbf{z}_w}]_1$ values, hand P_0 the (possibly flipped) values so that P_0's output matches $(\mathsf{m}_{\mathbf{z}_w}, [\lambda_{\mathbf{z}_w}]_0)$.
5. Output the view from the imaginary execution of Π_{LUT}.

Building the Simulator

When P_1 is about to send its $[\mathrm{m}_{\mathbf{z}_w}]_1$ values, hand P_0 the (possibly flipped) values so that P_0's output matches $(\mathrm{m}_{\mathbf{z}_w}, [\lambda_{\mathbf{z}_w}]_0)$.

This is possible since P_1 can always *fake* the sampled value of $[\lambda_{\mathbf{z}_w}]_1$. ($P_1$ randomly samples $[\lambda_{\mathbf{z}_w}]_1$ in the setup phase.)

Building the Simulator

Proof? that $\mathcal{S}_0$ works

The only differences between the real and the ideal world are:

1. Possible lying about the choice of $[\lambda_{\mathbf{z}_w}]_1$.
2. Inputs of P_1 are arbitrarily chosen.

Hence, any good distinguisher D for the real and the ideal world must distinguish one of them.

1. If D distinguishes $[\lambda_{\mathbf{z}_w}]_1$, then it successfully attacks the PRF primitive.
2. If D detects that the simulators decided the choice, then it successfully attacks the $\mathcal{F}_{\mathsf{AND}}$ functionality. $\mathcal{F}_{\mathsf{AND}}$ is called a constant number ($< 2^{\delta}$) of times.

References

- Brüggemann, A., Hundt, R., Schneider, T., Suresh, A., & Yalame, H. (2023). Flute: Fast and secure lookup table evaluations. *2023 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/sp46215.2023.10179345
- Yehuda L. (2016). How To Simulate It - A Tutorial on the Simulation Proof Technique. *Cryptology ePrint Archive*. https://eprint.iacr.org/2016/046