CS 65500
Advanced Cryptography

Lecture 24: Homomorphic Secret Sharing

Instructor: Aarushi Goel

Spring 2025

Agenda

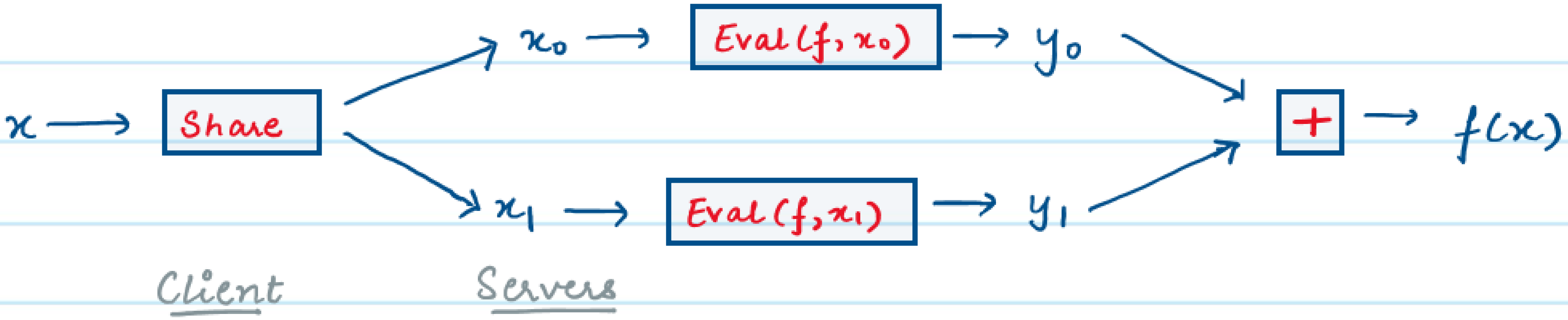
→ Homomorphic Secret Sharing

→ Motivation

→ Construction

→ Applications

Homomorphic Secret Sharing



→ Client computes and sends *shares* of x to servers, such that the servers can *non-interactively* compute additive shares of any function evaluated on x.

* Correctness: $y_0 + y_1 = f(x)$

* Security: x_0 and x_1 individually hide x

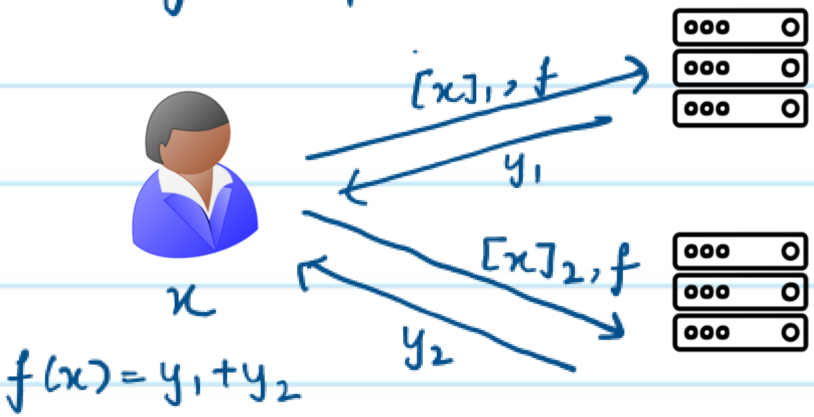
* Efficiency: - size of x_0, x_1 should be independent of the size of f.

- y_0, y_1 should be the same size as $f(x)$.

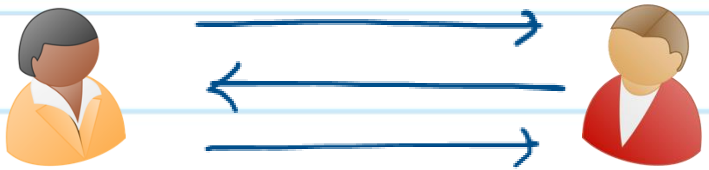
next week

Applications

→ Private delegation of computation to cloud servers (alternative to FHE)



→ Minimizing communication in secure multiparty computation.



total communication sublinear in the size of f.

→ Function secret sharing → PIR

Known Constructions of HSS

Today

- → The first construction was proposed by Elette Boyle, Niv Gilboa & Yuval Ishai in 2016 for all functions in NC^1 from DDH.
- → We have other constructions for NC^1 based on DCR, LWE, Class groups
- → We have some constructions for lower complexity classes based on variants of LPN
- → We have constructions for P/poly based on FHE or Obfuscation.

RMS Programs

↓

Restricted-Multiplication Straightline Programs.

→ The class of RMS programs consists of a magnitude bound M and arbitrary sequence of the following four instructions:

1 Load input into memory: $v_j \leftarrow x_i$

2. Add values in memory: $v_K \leftarrow v_i + v_j$

3. Multiply value in memory by an input value: $v_K \leftarrow x_i \cdot v_j$

4. Output value from memory: $\text{Out} \leftarrow v_i$

* If at any step of execution, the size of a memory value exceeds the bound M, the program outputs $\perp$.

RMS programs capture functions in NC_1 and logspace computations.

HSS for RMS Programs. (from DDH)

→ Let G be a DDH group of size q with generator g.

→ Let's consider the following types of distributed encodings of $\mathbb{Z}_q$ elements:

	Server 1	Server 2		
$[u]$	$g^u \in G$	$g^u \in G$		encryption
$\langle v \rangle$	$v_1 \in \mathbb{Z}_q$	$v_2 \in \mathbb{Z}_q$	$(v_1 = v + v_2)$	additive
$\{w\}$	$w_1 \in G$	$w_2 \in G$	$(w_1 = w_2 \cdot g^w)$	multiplicative

→ Let us for simplicity assume that g^u is a secure encryption of u (even though we know that it's not)

HSS for RMS Programs – Attempt I

* Share $(\vec{x})$: Let $\vec{x} = x_1, \ldots, x_n$

for each x_i : Encrypt $[x_i]$

Additively secret share $\langle x_i \rangle$

each server is given the encryption and an additive share of x_i.

* Eval $\left([\vec{x}], \langle \vec{x} \rangle, \overset{\text{RMS program}}{P} \right)$: The invariant we are going to maintain is that the servers should be able to compute additive shares of all memory values.

→ $v_j \leftarrow x_i$: $\langle v_j \rangle \longleftarrow \langle x_i \rangle$

→ $v_k \leftarrow v_i + v_j$: $\langle v_k \rangle \longleftarrow \langle v_i \rangle + \langle v_j \rangle$

→ $Out \leftarrow v_j$: $out \longleftarrow \langle v_j \rangle$

→ $v_k \leftarrow x_i \cdot v_j$: The servers have $[x_i], \langle x_i \rangle, \langle v_j \rangle$
how can they use these to non-interactively compute $\langle v_k \rangle$?

Idea 1:

$$\begin{tikzcd}
\underline{\text{Server 1}} & \underline{\text{Server 2}} \\
g^x,\ x_1, v_1 \arrow[d] & g^x, x_2, v_2 \arrow[d] \\
(g^x)^{v_1} = g^{xv_1} & g^{xv_2}
\end{tikzcd}$$

$x_1 = x + x_2$
$v_1 = v + v_2$

$$g^{xv_1} = g^x \cdot g^{xv_2}$$

They can compute multiplicative shares of $x \cdot v$

Q Multiplicative → Additive shares.?

Converting Multiplicative to Additive Shares

Non-interactively.

* **Distributed DLog:**

Let $\phi : G \to \{0,1\}^*$ be a PRF

Given a group element $h \in G$, distributed DLOG is a deterministic algorithm that computes the smallest value of i, such that $\phi(h \cdot g^i) = 0$.

Algorithm (g, h, ϕ):

```
    $h' \leftarrow h$ , $i \leftarrow 0$
while $\phi(h') \neq 0$ :
    $h' \leftarrow h' \times g$ , $i = i+1$
end while
output $i$.
```

Server 1	Server 2
g_1	g_2
$\mathrm{DistDLOG}(g, g_1, \phi)$	$\mathrm{DistDLOG}(g, g_2, \phi)$
$\hookrightarrow i \in \mathbb{Z}_q$	$\hookrightarrow i + vx \in \mathbb{Z}_q$

s.t. $g_1 = g^{vx} \cdot g_2$

→ Observe that i & $i+vx$ form additive secret shares of vx, which is exactly what we wanted the servers to compute.

Some Issues with this construction

Issue 1: The DistDLOG algorithm is such that the output obtained by the servers may not always be a *correct* additive sharing of vx. The probability of error is at most inverse polynomial in sec parameter which is not negligibly small.

→ In fact the exact probability of error depends on vx.

→ This yields an HSS for RMS programs with very high correctness error

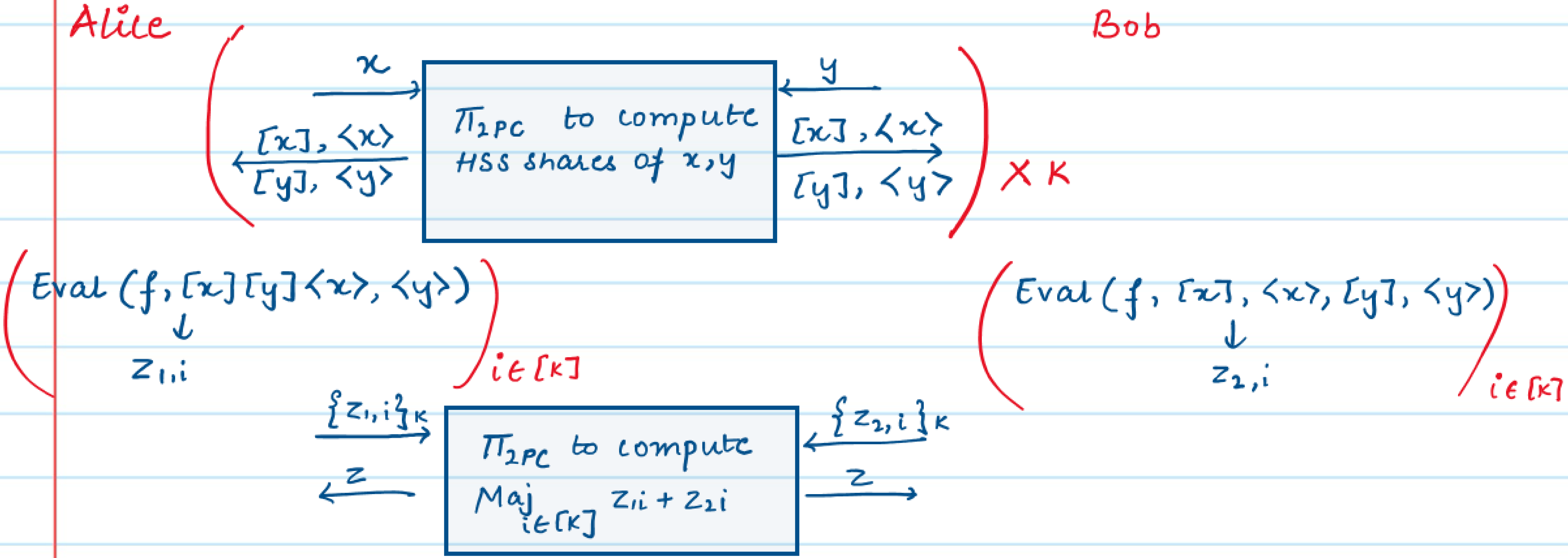
Issue 2: In our discussion so far, we assumed that g^u is a secure encryption of u. However, these ideas can be easily extended to work with El Gamal encryption instead of g^u.

(Read the [BGI16] paper to see how)

Application of HSS to Sublinear MPC

total communication sublinear in size of func.

→ We can use this HSS with low correctness to design a sublinear secure two-party computation for computing any function in NC^1



Application of HSS to FSS

- → This HSS scheme with low correctness can also be used to design an FSS with low correctness for any function in NC^1.
- → Let U_x denot a *universal* function that takes a function f as input and outputs $f(x)$.
- → Let (HSS.Share, HSS.Eval) be an HSS scheme.
- → The FSS scheme can be constructed as follows:

FSS.Share (f): Run HSS.Share$(f) \rightarrow [f], \langle f \rangle$

FSS.Eval $(U_x, [f], \langle f \rangle)$: Run HSS.Eval $(U_x, [f], \langle f \rangle)$