HSS schemes are very useful for realizing a broad range of privacy-preserving applications

↳ Here, we will focus on one application that has good <u>concrete efficiency</u>

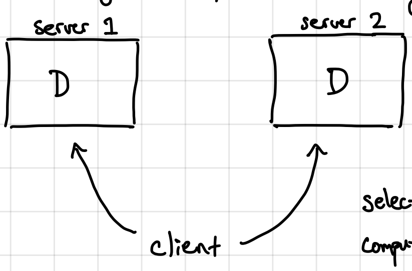
Technically, we will consider the dual notion of function secret sharing (FSS)

$$f \begin{cases} \nearrow f_1 \to f_1(x) \\ \searrow f_2 \to f_2(x) \end{cases} \Big\} \; f_1(x) + f_2(x) = f(x)$$

↳ common input, function shared

(HSS: common function, input shared) ← equivalent if we use universal circuits (both parties evaluate U_x on shares of f)

Private database queries: imagine multiple servers hosting replicas of a single database



select "top 10 restaurants" where "category = $\boxed{\text{Mediterranean}}$"

compute "average price" where "date = $\boxed{\text{tomorrow}}$" and "origin = $\boxed{\text{Austin}}$" and "destination = $\boxed{\text{NYC}}$"

query parameters should be hidden

<u>Goal</u>: Hide query attributes from servers (but revealing structure is fine)

Approach for handling statistical queries (e.g., count, sum, variance): leverage <u>linearity</u>

Consider a query of the form

"COUNT (column) WHERE $x_1 = v_1,\ x_2 = v_2, \dots, x_n = v_n$"

We can define a predicate

$$f(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } x_1 = v_1 \wedge \cdots \wedge x_n = v_n \\ 0 & \text{otherwise} \end{cases}$$

Secret share $f \to f_1, f_2$ and given f_1, f_2 to servers

For each record in the database

x_1	x_2	$\cdots$	x_n	
1	0	$\cdots$	0	$\to f_1(x_1^{(1)}, \dots, x_n^{(1)})$
0	1	$\cdots$	0	$\to f_1(x_1^{(2)}, \dots, x_n^{(2)})$
1	0	$\cdots$	1	$\vdots$
1	0	$\cdots$	0	

$$\hookrightarrow \sum_{i \in [N]} f_1(x_1^{(i)}, \dots, x_n^{(i)})$$

<u>Invariant</u>: if $x_1 = v_1, \dots, x_n = v_n$, then $f_1(x_1, \dots, x_n) + f_2(x_1, \dots, x_n) = 1 \pmod p$

else $f_1(x_1, \dots, x_n) + f_2(x_1, \dots, x_n) = 0 \pmod p$

$$\therefore \sum_{i \in [N]} f_1(x_1^{(i)}, \dots, x_n^{(i)}) + f_2(x_1^{(i)}, \dots, x_n^{(i)}) = \mathrm{COUNT}(x_1 = v_1, \dots, x_n = v_n).$$

Directly generalizes to computing linear functions of elements (important that HSS/FSS outputs linear shares)

For queries like select MAX(rating) where $x_1 = v_1, x_2 = v_2, \dots, x_n = v_n$, servers first preprocess the database by computing MAX(rating) for each combination of values $(x_1, \dots, x_n)$ — "group by" query

↳ Reduces now to select row corresponding to $(v_1, \dots, v_n)$ as described above

Key primitive: function secret sharing for a point function:

$$f_{(v_1,\dots,v_n)}(x_1,\dots,x_n) = \begin{cases} 1 & \text{if } x_1 = v_1, \dots, x_n = v_n \\ 0 & \text{otherwise} \end{cases}$$

To simplify notation, we will write

$$f_y(x) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

FSS for point functions = distributed point functions (DPFs)

We can build DPFs from OWFs (in the 2-party setting)!
↳ Very practical → See Splinter system.

We will start by describing a $\sqrt{N}$ construction where N is the size of the domain $(f_y : [N] \to \{0,1\})$

1) Let $\ell = \sqrt{N}$. Represent domain elements as (i,j) where $i, j \in [\sqrt{N}]$
2) Suppose we want to secret share f_{i^*, j^*}:

Sample PRG seeds, where output of PRG is $\ell = \sqrt{N}$ bits

Share f_1 will consist of all the seeds $s_1, \dots, s_\ell$ $\xrightarrow[\text{behavior on row } i^*]{\text{need to change}}$ Share f_2 will consist of same seeds except s_{i^*} is replaced with independent seed s_{i^*}'

To evaluate at (i,j): compute $\mathrm{PRG}(s_i)$ and output bit j

Observe: For $i \neq i^*$, shares are equal: $\mathrm{PRG}(s_i) \oplus \mathrm{PRG}(s_i) = 0^\ell$ (secret share of 0)

Problem: All entries in row i^* are corrupted
↳ Need to add a correction word

$$w = \boxed{\mathrm{PRG}(s_{i^*}) \oplus \mathrm{PRG}(s_{i^*}') \oplus e_{j^*}}$$

$\mathrm{PRG}(s_{i^*}) \oplus \mathrm{PRG}(s_{i^*}') \oplus w = e_{j^*}$ (0 everywhere except position j^*)

Problem: Should only xor with w in row i^*; otherwise all other rows are corrupted
Also need to hide i^*

Approach: Add "correction bits" $b_1, \dots, b_{i^*}, \dots, b_n \xleftarrow{R} \{0,1\}$
Include $b_1, \dots, b_n$ with both shares, except flip bit b_{i^*} in one of the shares:

Share 1	Share 2
(s_1, b_1)	(s_1, b_1)
$\vdots$	$\vdots$
(s_{i^*}, b_{i^*})	$(s'_{i^*}, 1-b_{i^*})$
$\vdots$	$\vdots$
(s_n, b_n)	(s_n, b_n)

$$w = PRG(s_{i^*}) \oplus PRG(s'_{i^*}) \oplus e_{j^*}$$

To evaluate at (i, j):

$$PRG(s_i) \oplus b_i \cdot w$$

When $i \neq i^*$: $PRG(s_i) \oplus b_i \cdot w \oplus PRG(s_i) \oplus b_i \cdot w = 0^\ell$

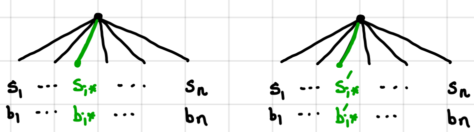
When $i = i^*$: $PRG(s_{i^*}) \oplus b_i w \oplus PRG(s'_{i^*}) \oplus (1-b_i) w = PRG(s_{i^*}) \oplus PRG(s'_{i^*}) \oplus w = e_{j^*}$

} Correct!

Security : w is blinded by $PRG(s_i)$ or $PRG(s_{i^*})$

all other components in any single share are uniform (independent of i^*)

To get shorter keys (size $O(\log N)$): use a tree-based construction



"control bit"

Off-path: secret share of 0^ℓ (in 2-party setting: parties have identical shares)

↳ Any subsequent computation will yield identical outputs [secret shares of 0 ⟶ secret shares of 0]

On-path: control bit is a secret share of 1

↳ Can be used to xor in a correction word — can program output to secret share of arbitrary value

To get $(\log N)$-size keys, use binary tree of depth $\log N$:

Associate a PRG seed with root node

↳ Each PRG seed generates seed for child nodes (GGM style)

Control bit at root is secret share of 1

↳ Allows programming of two output values (for left and right nodes)

Off-path: Program value to secret share of 0 (control bit also 0)

On-path: Program value so control bit is still a secret share of 1

Secret shares consist of share of root PRG seed, $\underbrace{\text{correction factors for each level}}_{\text{each of size } poly(\lambda)}$

↳ Overall seed size: $poly(\lambda) \cdot \log N$

Techniques extends naturally to intervals

exponential-size keys since typically domain is $\{0,1\}^n$ $(N = 2^n)$.

Beyond 2-parties: Best construction from OWFs has share size $2^k \sqrt{N} \cdot poly(\lambda)$ (k = # parties, N = domain size)

Open problem: Share size smaller than $\sqrt{N}$ from OWFs (e.g., can we get $\sqrt[3]{N} \cdot poly(\lambda)$ for 3 parties)

Primitive has many applications: private writes to a database ⇝ anonymous messaging

generating correlated randomness for MPC

(in some settings, HSS/FSS give the fastest OT extension protocol in practice)