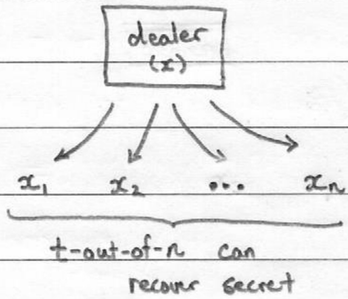
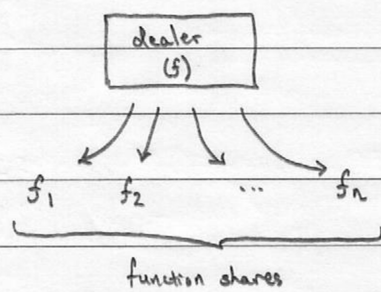
Function Secret Sharing and PIR

Shamir secret sharing allows a dealer to split a value across many parties



- possible to perform arbitrary computation on secret-shared data [Ben-Or, Goldwasser, Wigderson, '88]
- each user secret-shares its input
  - addition is local
  - multiplication requires communication (degree-reduction)
- very efficient: information-theoretic (honest-majority for semi-honest security)

Function secret sharing [Boyle, Gilboa, Ishai '15]: allows a dealer to split a function



guarantee: for any x:

$$\sum_{i=1}^{n} f_i(x) = f(x)$$

requirements: function shares should be
- succinct (otherwise, can have trivial construction where truth table is secret-shared)
- not reveal anything about the function f (information-theoretically: secret share truth table, but more efficient construction possible with computational-hiding properties)

This lecture: consider one special case of function secret-sharing (for two-party distributed point functions (DPFs))

- Introduced by Gilboa and Ishai (Eurocrypt 2014) – surprisingly powerful and useful primitive
- Point function: $f_y(x) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise.} \end{cases}$
- Distributed point function consists of two algorithms (Gen and Eval):

  $\text{Gen}(y) \to (k_0, k_1)$ generates keys for point function at y

  $\text{Eval}(k, x') \to y'$ evaluates point function at x'

- Correctness: for all points x, y, if $(k_0, k_1) \leftarrow \text{Gen}(y)$;

$$\text{Eval}(k_0, x) \oplus \text{Eval}(k_1, x) = f_y(x)$$

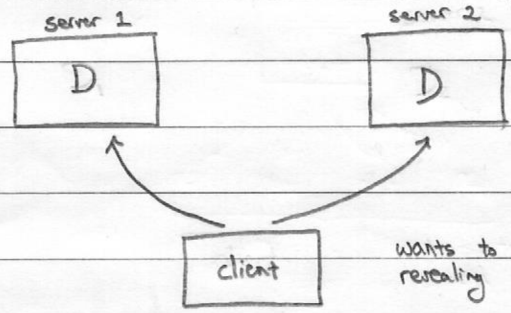
- Security: for all points y: $(k_0, k_1) \leftarrow \text{Gen}(y)$:

$$\{k_b\}_{b \in \{0,1\}} \overset{c}{\approx} \text{Sim}(b, |x|, |y|)$$

Intuitively: key k_b reveals nothing about point y, other than size of domain and range

Why DPFs?

Gives an immediate solution for multi-server PIR (private information retrieval)



for reading: databases are replicated on multiple servers

wants to read record i without revealing index i to the servers

Applications: perform queries to a database without revealing query to server

(e.g. [private flight lookups to prevent discriminatory pricing, private navigation to ensure location privacy, private lookups in Tor hidden services]

Splinter system [NSDI 2017]

Can also consider reverse problem: writing to a database without revealing which position was updated

- very useful for anonymous messaging: Riposte system [Oakland 2015]

Closely related to oblivious transfer (no requirement for sender privacy, only receiver privacy)

- goal in PIR is to minimize communication (in OT, usually it is to minimize computation, but can combine OT with PIR to reduce communication — "strong PIR")

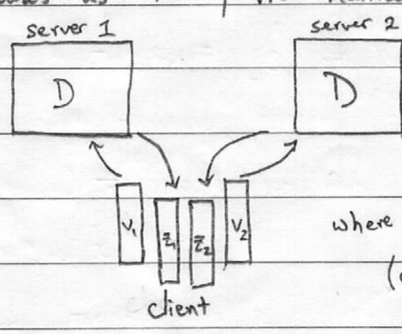
trivial PIR is to send entire database: $O(n)$ communication

best-possible in single-server information-theoretic setting

Information-Theoretic PIR

Two-server PIR with $O(\sqrt{n})$ communication:

- View databases as $\sqrt{n}$-by-$\sqrt{n}$ matrices:



$$z_1 = D \cdot v_1$$

$$z_2 = D \cdot v_2$$

$$z_1 + z_2 = D(v_1 + v_2) = De_i = D_i$$

where $v_1 + v_2 = e_i$

(client's desired element in column i)

i^{th} column of D

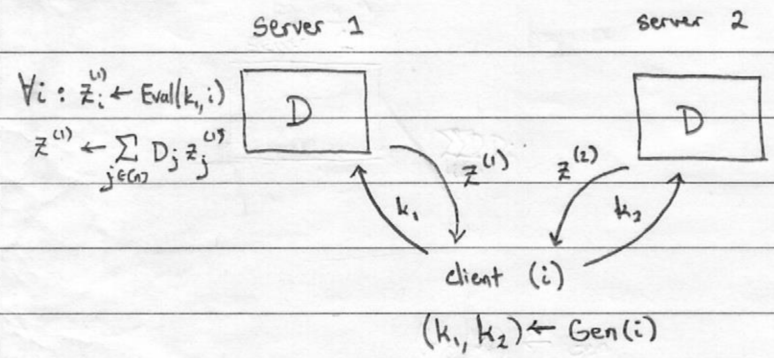
- Can reduce communication to $O(n^{1/3})$ [CGKS98]
- Best known lower bound is $5 \log n$ (trivial lower bound is $\log n$ – need to communicate bits of index)
- Conjectured lower bound $\Omega(n^{1/3})$ in [CGKS98]
- Breakthrough work by Dvir, Gopi (2015): 2-server PIR with communication $n^{O\left(\sqrt{\log\log n / \log n}\right)}$

Distributed Point Functions (DPFs)

Relies on computational assumptions and gives 2-server PIR with polylog communication

- Note: with computational assumptions, can have <u>single-server</u> PIR with polylog communication but this requires algebra (in fact, single-server PIR with sublinear communication implies OT, so algebra is probably <u>necessary</u>)
- DPFs give very <u>efficient</u> construction in 2-party setting (relying only on one-way functions) (AES)

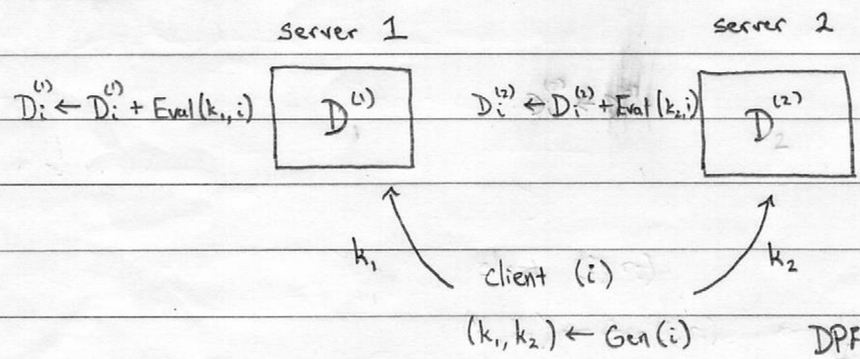
2-server PIR from DPF:



$$z^{(1)} + z^{(2)} = \sum_{j\in[n]} D_j \underbrace{\left(z_j^{(1)} + z_j^{(2)}\right)}_{=0 \text{ except if } i=j} = D_i$$

Total communication : $\underbrace{|k_1| + |k_2|}_{O(\log n)} + \underbrace{|z^{(1)}| + |z^{(2)}|}_{O(1) \text{ for bits}}$

2-server Writable PIR from DPF:



database is secret shared:

$$D = D^{(1)} + D^{(2)}$$

(initially all zeroes)

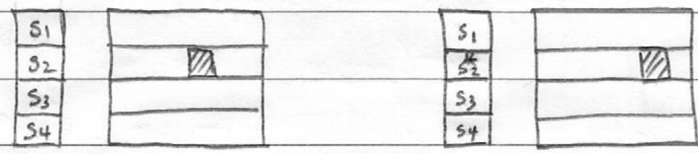
DPF guarantees that update only flips bit at position i

Take-away : Reading obliviously from a database : secret-share query (same database)

Writing obliviously to a database : secret-share query (secret-share database)

Distributed Point Functions

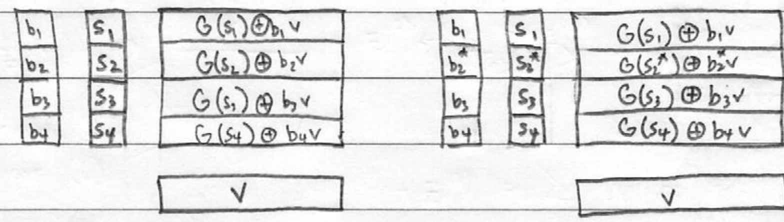
- Start by constructing a DPF with $\sqrt{n}$-size keys (where the domain size is n)
- View output of DPF as $\sqrt{n}$-by-$\sqrt{n}$ grid – compress using PRG



two functions differ only at shaded index

① associate random PRG seed with each column $\Rightarrow$ use PRG to derive pseudorandom string for each row $\Rightarrow$ different PRG for the target row → but now every element in the row differs

② introduce a correction factor for the columns



one correction factor so that $G(s_2) \oplus G(s_2^*) \oplus v = e_j$ (if desired entry is in column j)

→ problem: when do you xor with v (cannot reveal the special point)

Solution: introduce vector of control bits

$$b_2^* = 1 - b_2$$

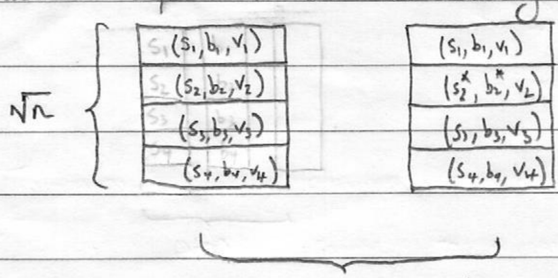
Correctness: rows other than i are identical
row j xors to e_j by construction

Security: seeds and control bits uniformly random, correction factor blinded by $G(s_2^*)$, so keys computationally indistinguishable from random

Efficiency: $\sqrt{n}$-size keys

Towards (Poly)-Logarithmic-Size Keys:

Observation keys in $\sqrt{n}$-DPF have following structure



can be viewed as shares of a point function over domain of size $\sqrt{n}$ (for ...) $\Rightarrow$ compress using another DPF on $\sqrt{n}$ elements $\Rightarrow$ recursively apply construction to obtain polylog-key size

- Can also build iteratively using tree-based construction
- Lots of applications (very practical!)