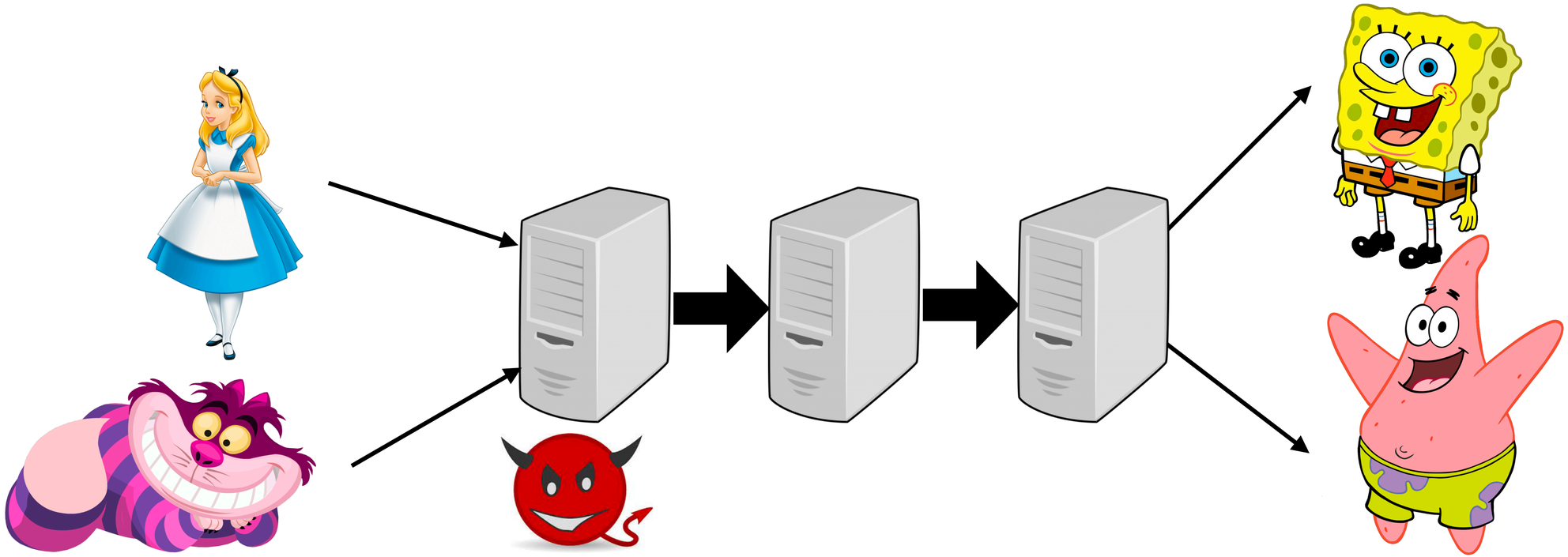
CS 350S: Privacy-Preserving Systems

Anonymous Messaging II

Recap: Mixnet [Chaum81]

- Messages sent through sequence of mixes
  - Can form arbitrary network of mixes (“mixnet”)
- An attacker might control some mixes — even a single good mix guarantees anonymity

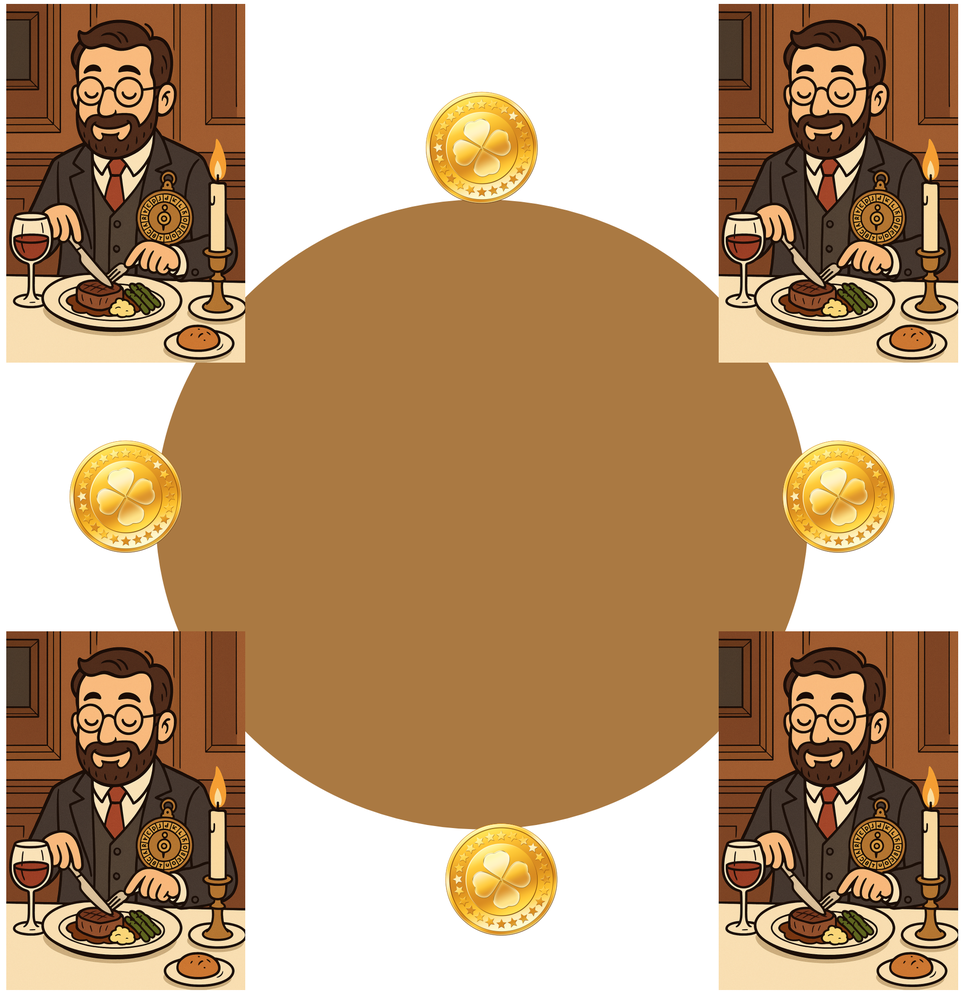


Recap: DC-nets [Chaum88]

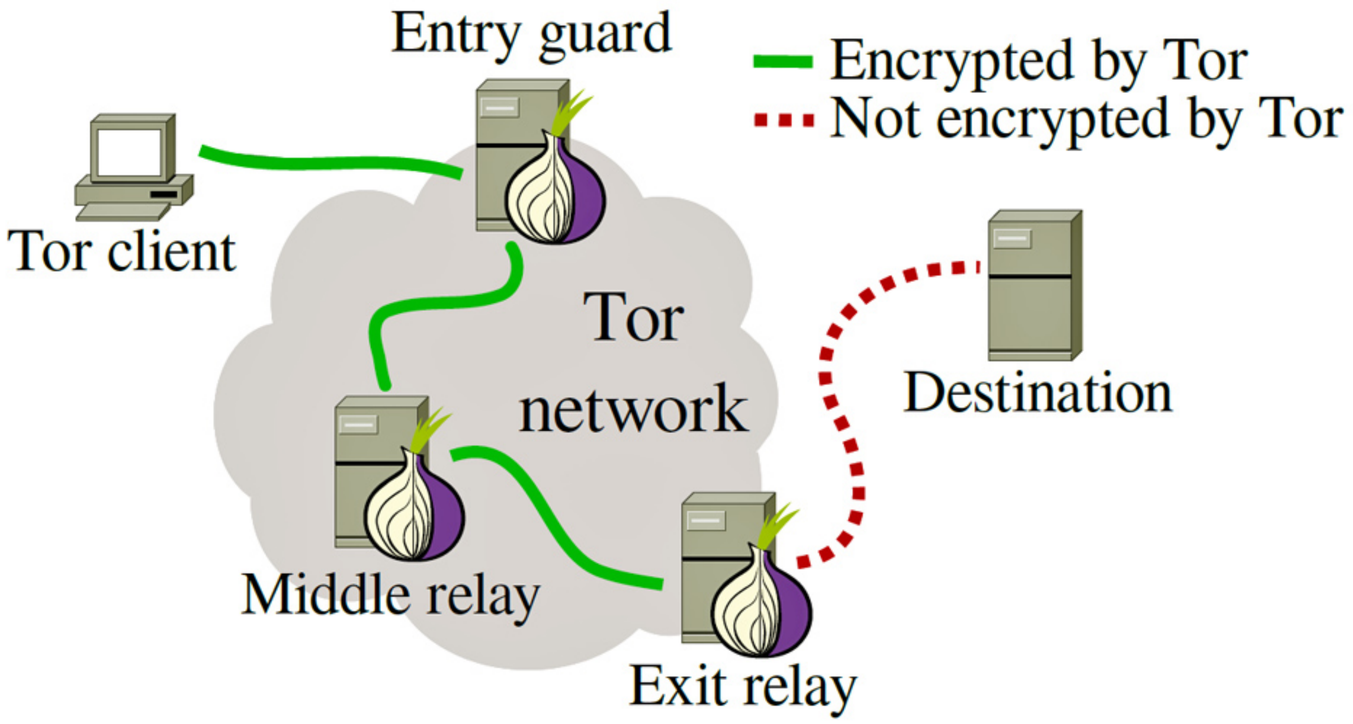
The cryptographers want to know: is the NSA paying, or one of the cryptographers?
… but without learning which cryptographer is paying

Protocol:

1. Each pair of adjacent cryptographers flips a coin that only they can see
2. Each cryptographer says whether or not the two coins are the same or different
   If the cryptographer paid for dinner, he/she states the *opposite* (i.e., if the same, says different)
3. Odd # differences: cryptographer paid
   Even # differences: NSA paid



Recap: Tor



https://arstechnica.com/information-technology/2014/01/scientists-detect-spoiled-onions-trying-to-sabotage-tor-privacy-network/

Recap: anonymous messaging

Mixnets

- Need high latency for good security: need to wait longer for large batches of requests (i.e., large anonymity set)

DC-nets

- Information-theoretic security, but high communication overheads ($O(n)$ communication with n parties for 1 sender to send a single bit)

Tor

- Low latency, but security guarantees are more heuristic and less precise

Outline

1. **Riposte**
2. Logistics
3. Student presentation

Riposte motivation

[Corrigan-Gibbs, Boneh, Mazières]

Prior anonymous messaging systems either

1. Resist traffic analysis attacks (mixnets, DC-nets, …) , or
2. Scale to many users with low latency (Tor, …)

Riposte's goal: resist traffic analysis while scaling to many users

Functionality

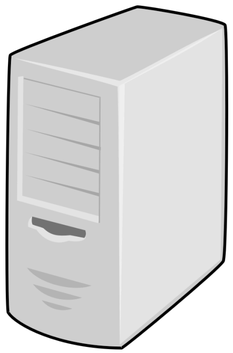
Tor: private point-to-point communication

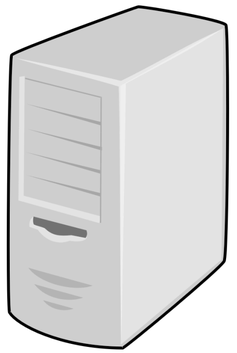
Riposte: posting anonymously to a bulletin board

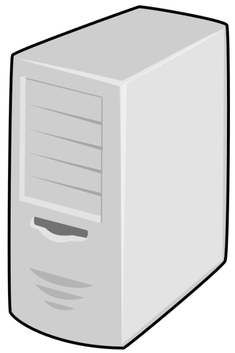
Riposte architecture

Add this server later

Can scale to >2 DB servers





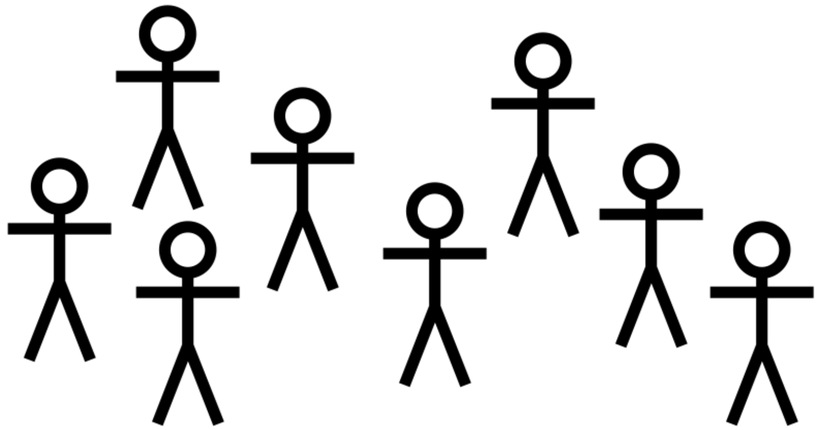


DB server 1

DB server 2

Audit server

Clients



Riposte security goals

Correctness: If all the servers execute the protocol faithfully, the plaintext state of the DB at the end of the protocol matches the result of applying each valid client write request to an empty DB.

(s, t)**-Write Privacy:** An adversary's advantage at guessing which honest client wrote into a particular row of the DB is negligibly better than random guessing, even when the adversary controls all but two clients and t out of s servers.

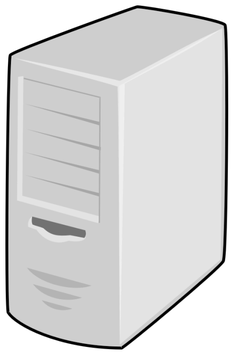
Disruption resistance: An adversary who controls n clients can write into at most n database rows during a single protocol round.

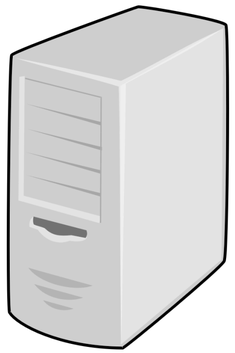
Rely on servers for availability

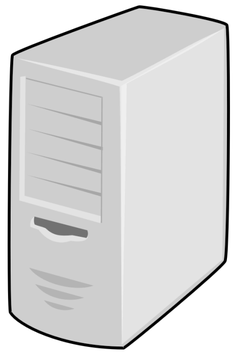
Riposte architecture

Add this server later

Can scale to >2 DB servers







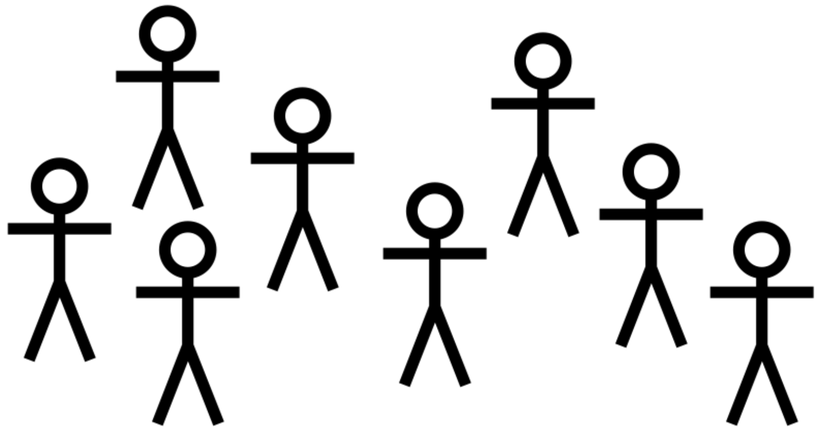
DB server 1

DB server 2

Audit server

Cannot deny service or violate other clients' privacy

Clients



Can deny service, but cannot violate client privacy

Recap: Secret sharing

For some group $\mathbb{G}$, split a value $x \in \mathbb{G}$ into secret shares $x_1, \ldots, x_k \in \mathbb{G}$ such that $\sum_{i=1}^{k} = x$

Information-theoretic privacy: Given just $[x]_b$ for $b \in [k]$, adversary learns no information about x

Operations:

- Given x, generate secret shares by randomly sampling $x_1, \ldots, x_{k-1}$ and setting $x_k = x - \sum_{i=1}^{k-1} x_i$
- Given $x_1, \ldots, x_k$, reconstruct x by computing $x = \sum_{i=1}^{k} x_i$

Computing on secret shares:

- Can add secret shares: $[x] + [y] = [x + y]$
- Can multiply by a constant: $c \cdot [x] = [c \cdot x]$ (by extension)

Recap: Function secret sharing (FSS)

[Boyle, Gilboa, Ishai]

Idea: Secret share a value rather than a function

Informal properties for n-party function secret sharing:

- Split function f into functions $f_1, \ldots, f_n$ where $f(x) = \sum_{i=1}^{n} f_i(x)$
- Describe f_i using a short key K_i
- Key K_i reveals nothing about f

Recap: Function secret sharing

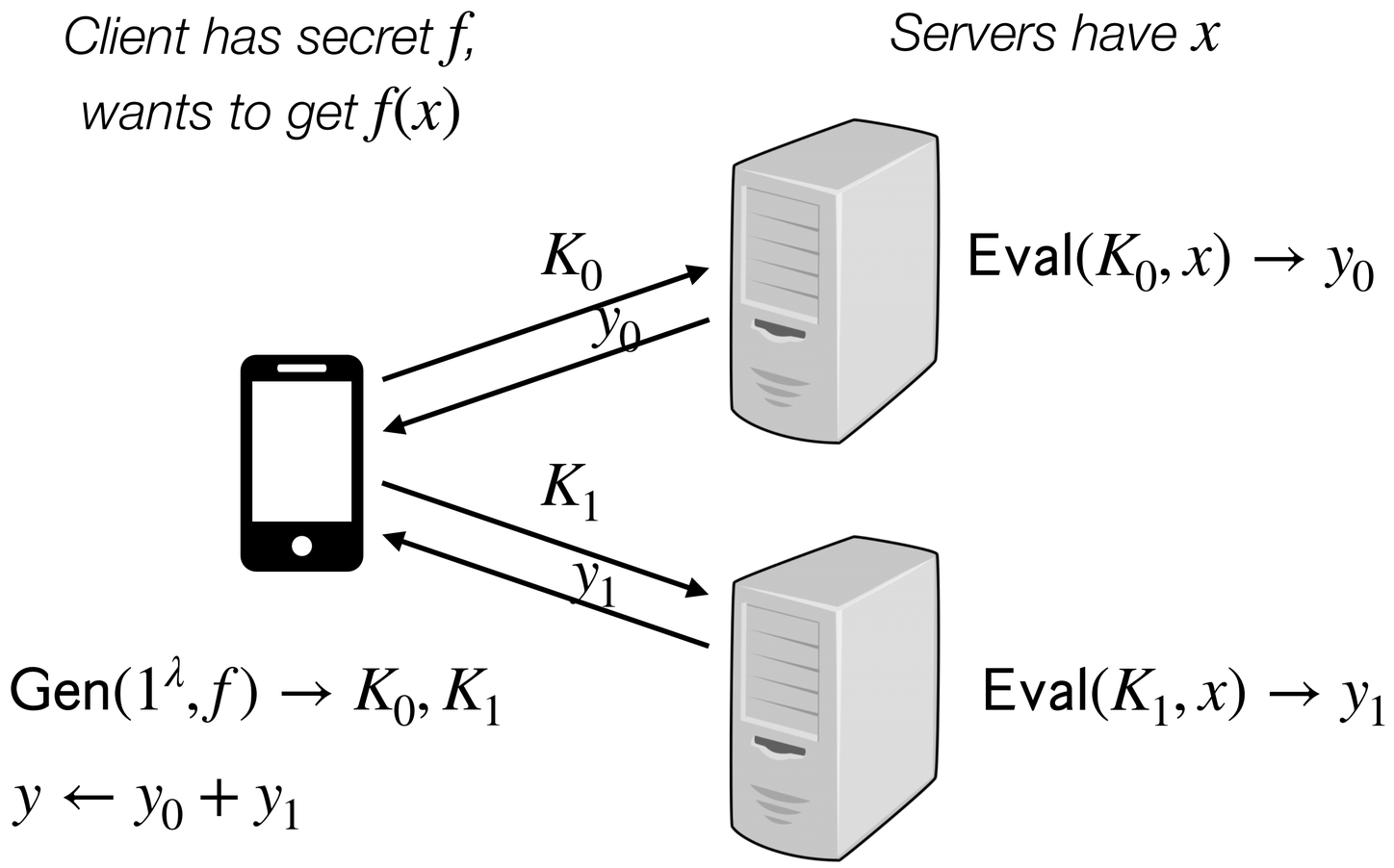
[Boyle, Gilboa, Ishai]

Security parameter λ, function f

Considering just two parties

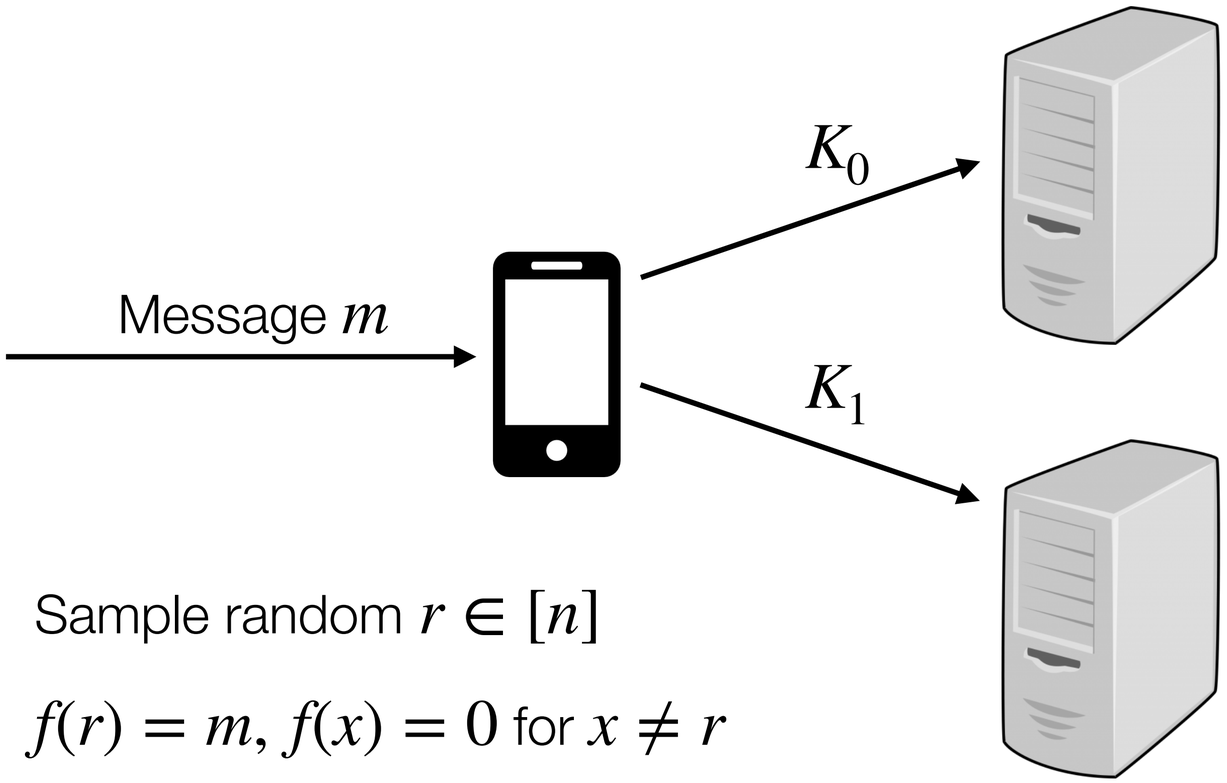
Two algorithms:

- $\mathsf{Gen}(1^\lambda, f) \to K_0, K_1$
- $\mathsf{Eval}(K_i, x) \to y_i$



Riposte toy protocol

During an epoch: collect client submissions



$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_0, 1)$

$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_0, 2)$

$\ldots \qquad \ldots$

$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_0, n)$

$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_1, 1)$

$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_1, 2)$

$\ldots \qquad \ldots$

$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_1, n)$

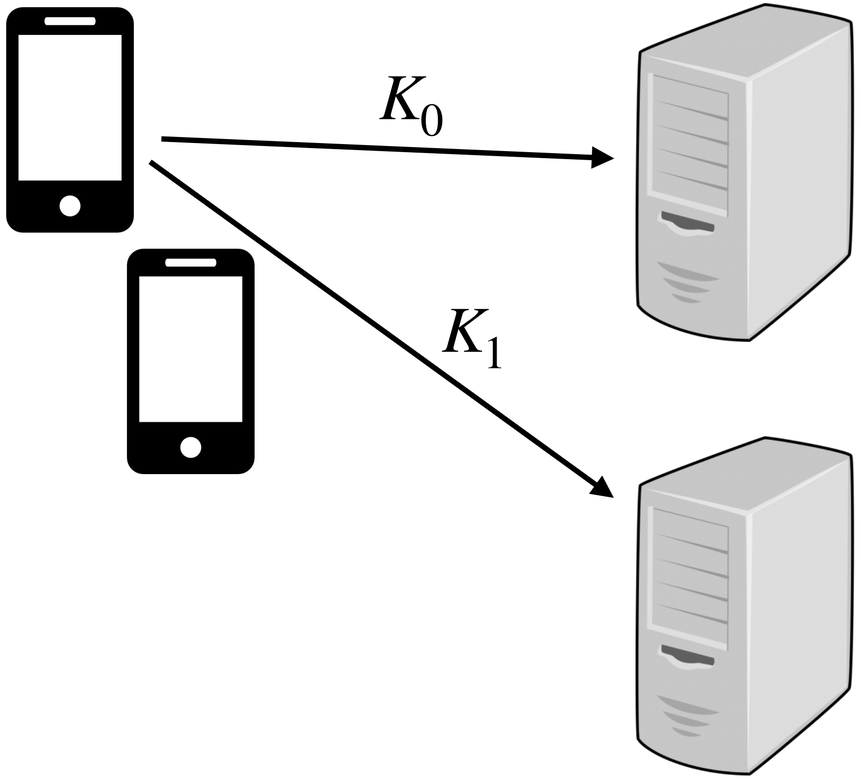
Sample random $r \in [n]$

$f(r) = m, f(x) = 0$ for $x \neq r$

$\mathsf{Gen}(1^\lambda, f) \rightarrow K_0, K_1$

Riposte toy protocol

During an epoch: collect client submissions



$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_0, 1)$

$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_0, 2)$

$\ldots$

$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_0, n)$

$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_1, 1)$

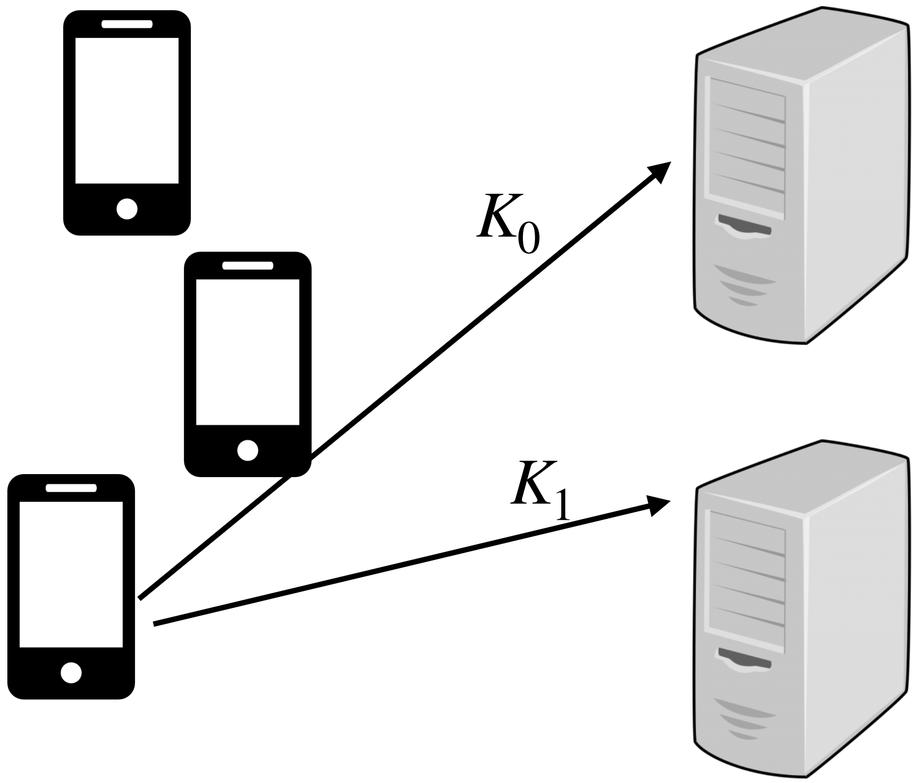
$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_1, 2)$

$\ldots$

$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_1, n)$

Riposte toy protocol

During an epoch: collect client submissions

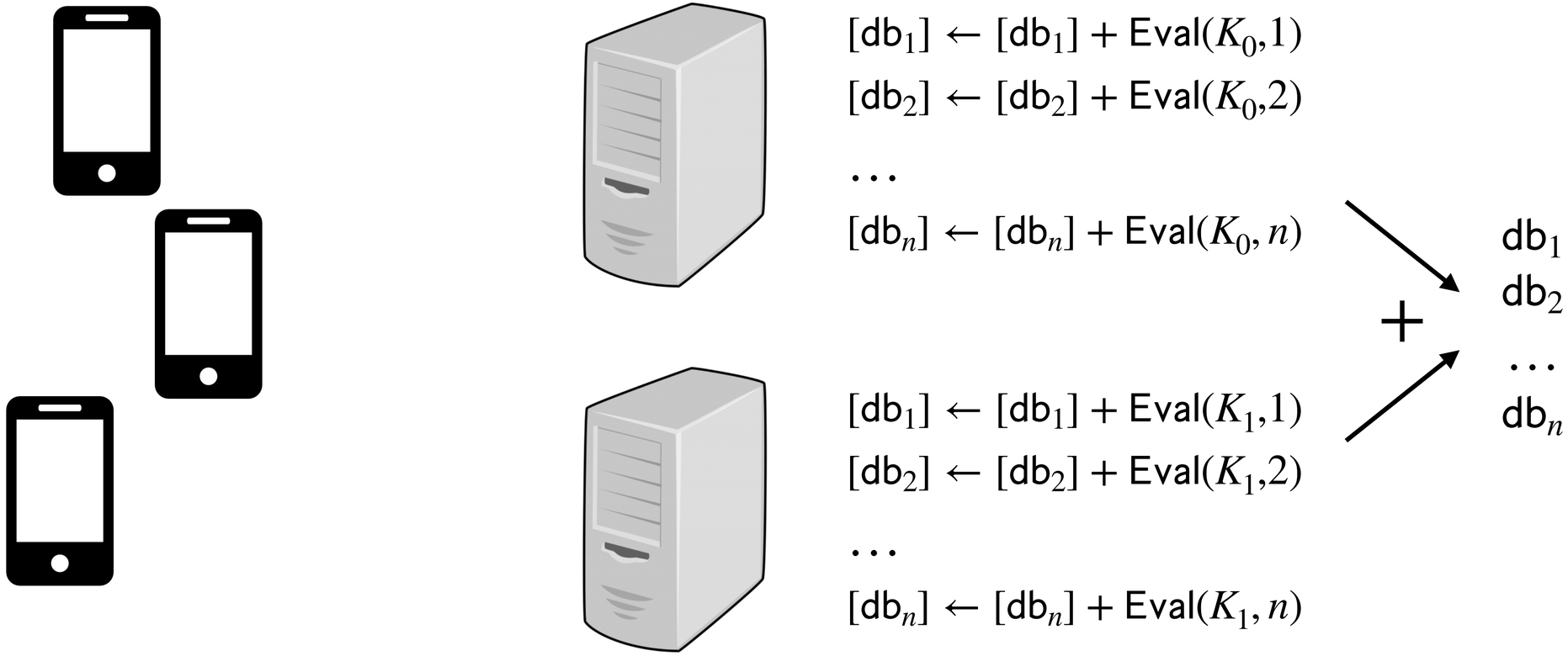


$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_0, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_0, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_0, n)$$

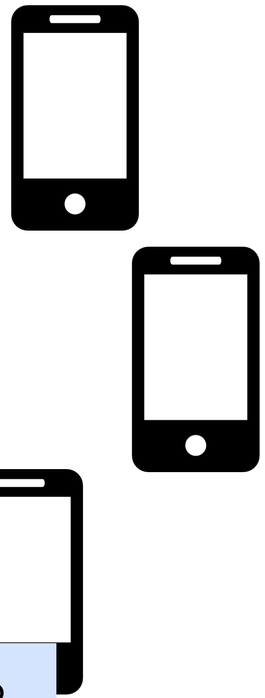
$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_1, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_1, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_1, n)$$

Riposte toy protocol

At the end of the epoch: reconstruct database state



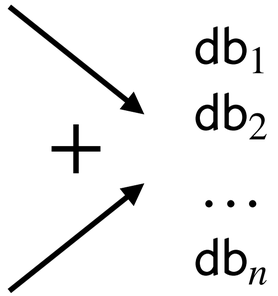
Riposte toy protocol



Why private?

$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_0, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_0, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_0, n)$$

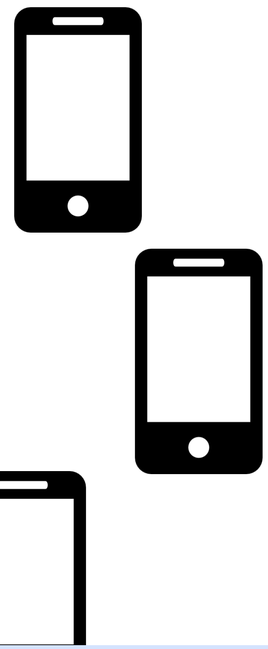
$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_1, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_1, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_1, n)$$

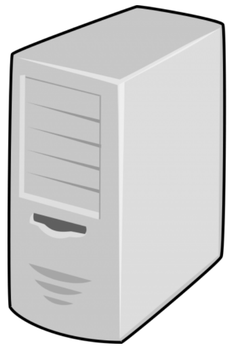


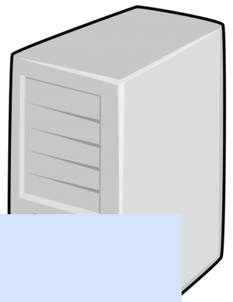
$$+$$

$$\mathsf{db}_1$$
$$\mathsf{db}_2$$
$$\cdots$$
$$\mathsf{db}_n$$

Riposte toy protocol







$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_0, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_0, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_0, n)$$

$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_1, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_1, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_1, n)$$

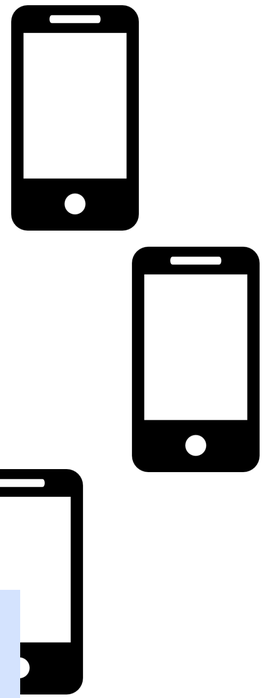
$$+$$

$$\mathsf{db}_1$$
$$\mathsf{db}_2$$
$$\cdots$$
$$\mathsf{db}_n$$

Why private?

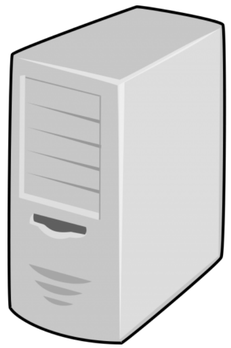
DPF hides location and content of each update

Riposte toy protocol

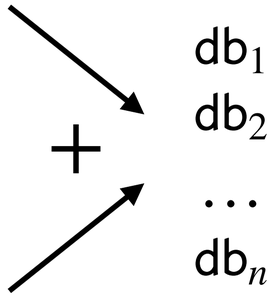


Limitations?

$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_0, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_0, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_0, n)$$

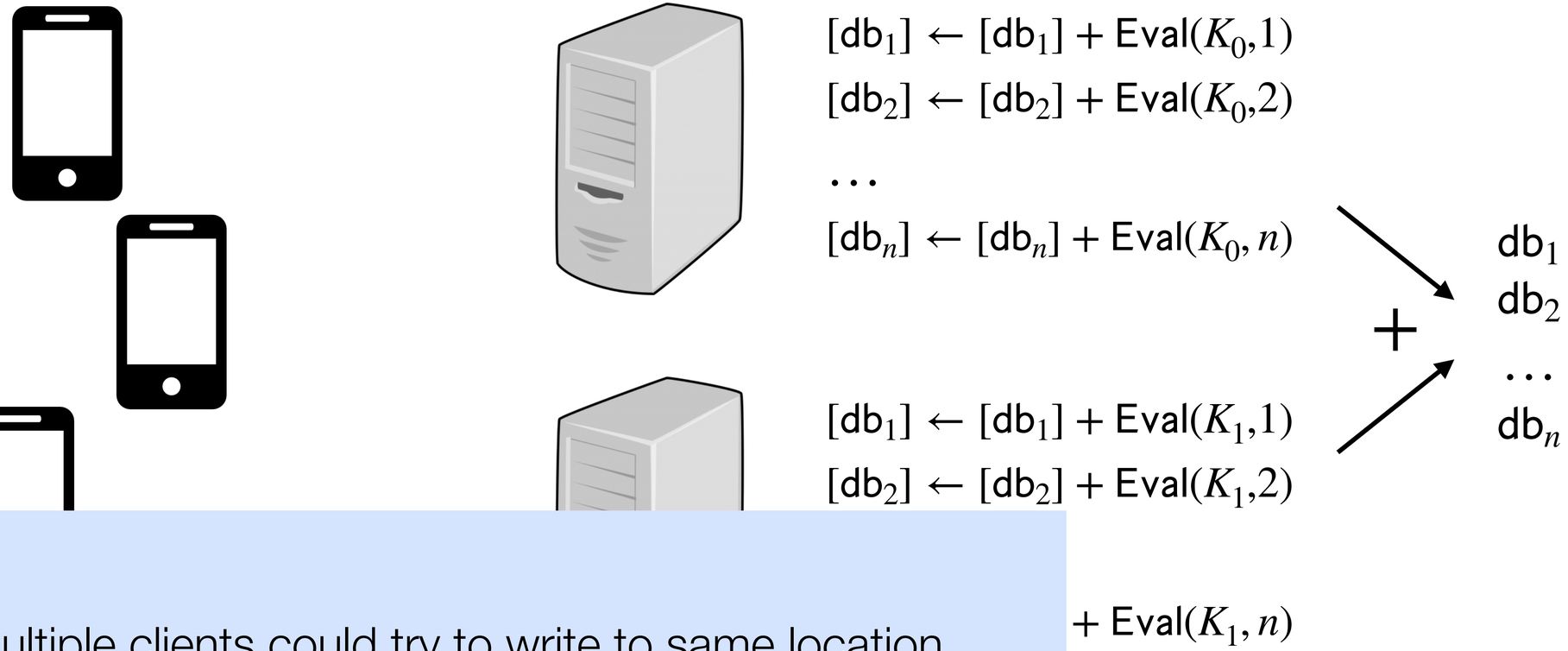


$$[\mathsf{db}_1] \leftarrow [\mathsf{db}_1] + \mathsf{Eval}(K_1, 1)$$
$$[\mathsf{db}_2] \leftarrow [\mathsf{db}_2] + \mathsf{Eval}(K_1, 2)$$
$$\cdots$$
$$[\mathsf{db}_n] \leftarrow [\mathsf{db}_n] + \mathsf{Eval}(K_1, n)$$



$$\mathsf{db}_1$$
$$\mathsf{db}_2$$
$$\cdots$$
$$\mathsf{db}_n$$

Riposte toy protocol



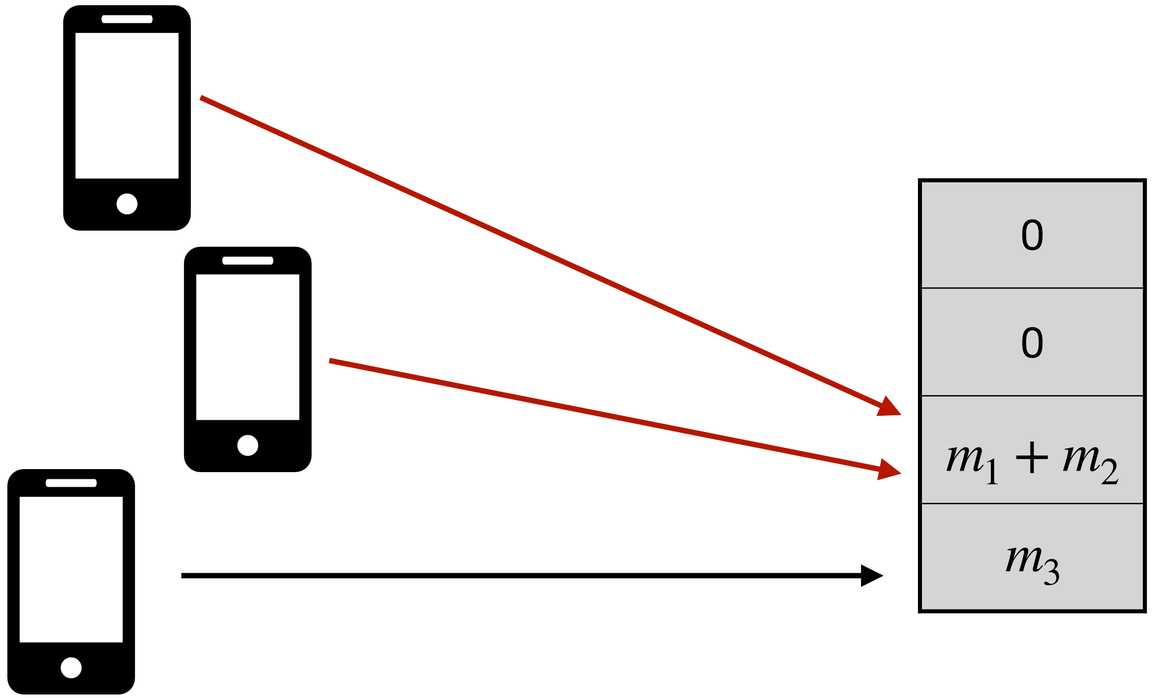
Limitations?

Collisions: multiple clients could try to write to same location

Malicious clients: one malicious client could overwrite many rows

Problem 1: Collisions

Two clients may try to write to the same location

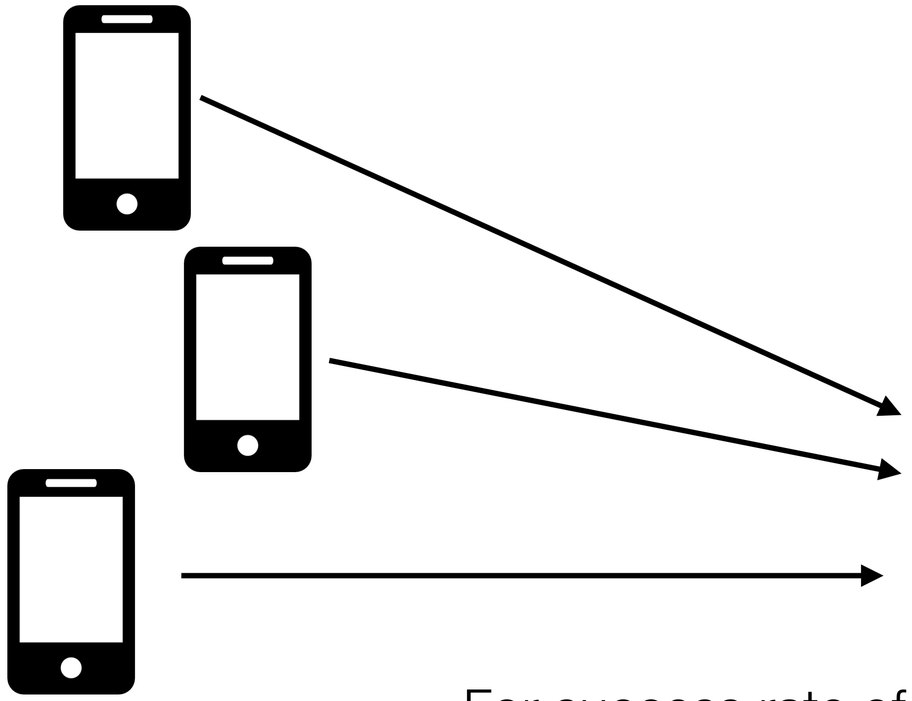


One approach: make hash table large enough to minimize probability of collision

More efficient approach: encode data so can recover even if a collision occurs

Problem 1: Collisions

Idea: write two values such that if a collision occurs, it is still possible to recover both messages



0	0
0	0
$m_1 + m_2$	$m_1^2 + m_2^2$
m_3	m_3^2

$$S_1 = m_1 + m_2$$
$$S_2 = m_1^2 + m_2^2$$
$$2S_2 - S_1^2 = (m_1 - m_2)^2$$

Solve for m_1, m_2

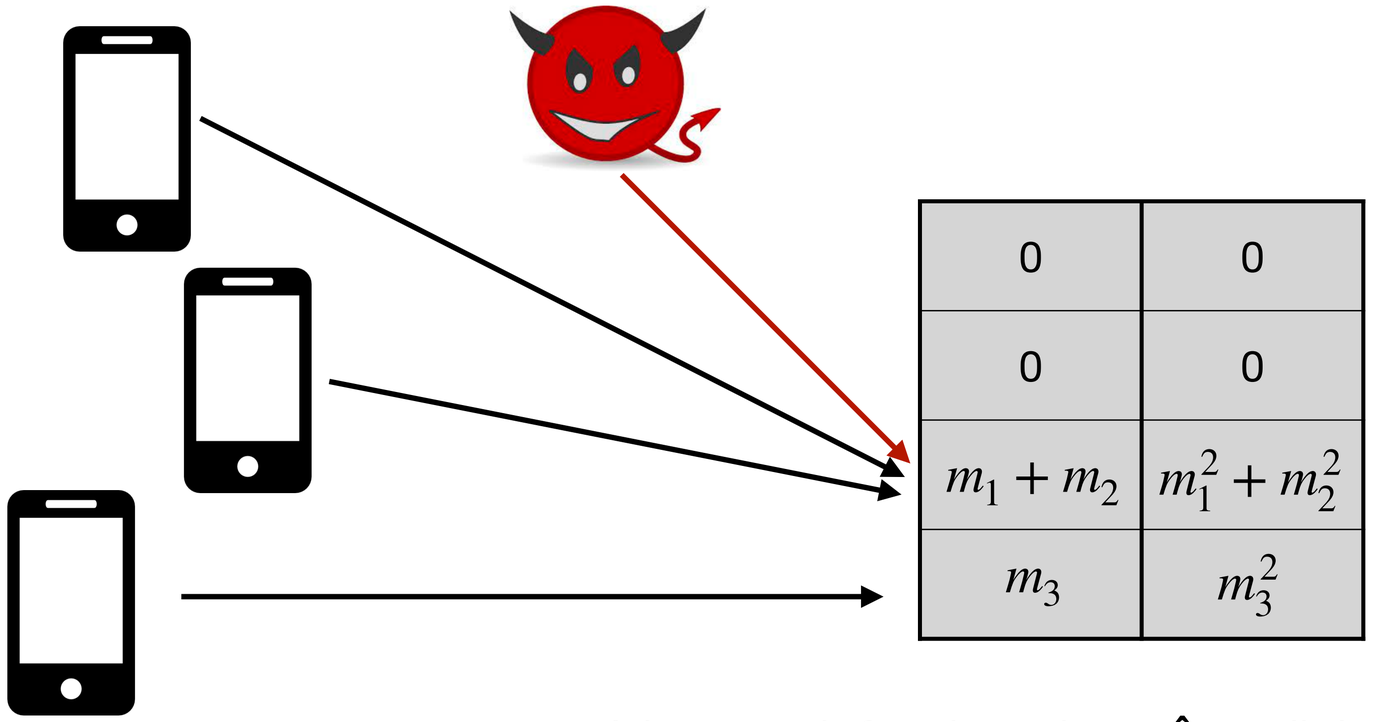
(all arithmetic mod p)

For success rate of 95% and m messages:

- With encoding trick, need a table with $2.7m$ rows
- Without encoding trick, need $19.5m$ rows (each row is smaller)

Problem 1: Collisions

What if a malicious client chooses writing locations not randomly (using some adversarial strategy)?

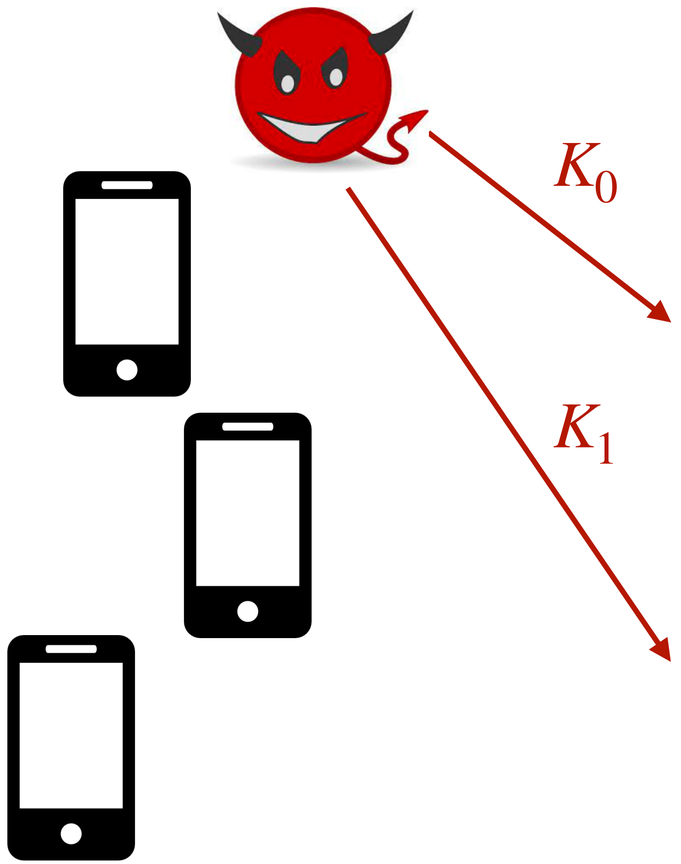


0	0
0	0
$m_1 + m_2$	$m_1^2 + m_2^2$
m_3	m_3^2

Idea: to defend against $\hat{m}$ malicious clients, perform DB analysis assuming $n - \hat{m}$ rows instead of n

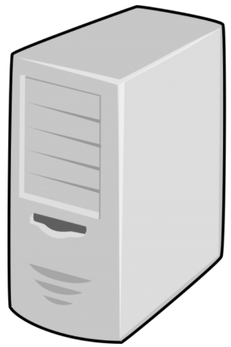
Problem 2: Malicious clients

One malicious client should not be able to corrupt multiple rows of the database



K_0

K_1



$[\text{db}_1] \leftarrow [\text{db}_1] + \text{Eval}(K_0, 1)$

$[\text{db}_2] \leftarrow [\text{db}_2] + \text{Eval}(K_0, 2)$

$\cdots$

$[\text{db}_n] \leftarrow [\text{db}_n] + \text{Eval}(K_0, n)$

$[\text{db}_1] \leftarrow [\text{db}_1] + \text{Eval}(K_1, 1)$

$[\text{db}_2] \leftarrow [\text{db}_2] + \text{Eval}(K_1, 2)$

$\cdots$

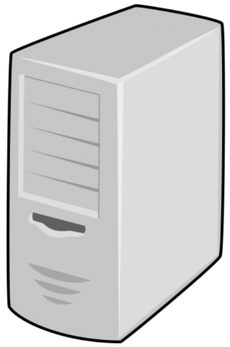
$[\text{db}_n] \leftarrow [\text{db}_n] + \text{Eval}(K_1, n)$

Problem 2: Malicious clients

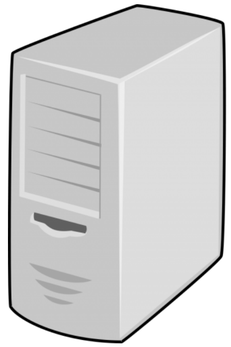
Disruption resistance: An adversary who controls n clients can write into at most n database rows during a single protocol round.

Need to check that update vectors $\mathbf{v}_A, \mathbf{v}_B$ are shares of 0 everywhere except at one index

… but without any server learning anything about the index where the vectors differ



Update vector $\mathbf{v}_A$



Update vector $\mathbf{v}_B$

Problem 2: Malicious clients

Riposte approaches:

- Add a third audit server to check for well-formed updates
- Use zero-knowledge proofs

Another approach: verifiable DPFs [BGI16]

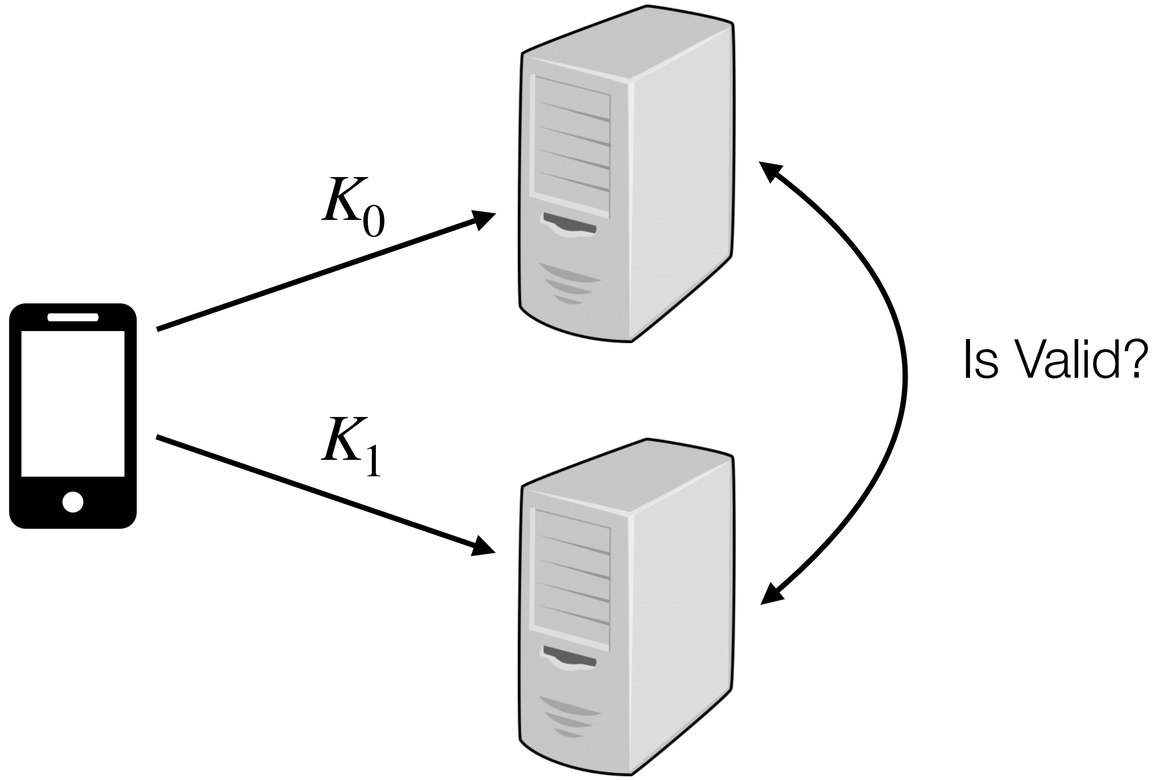
Verifiable FSS [BGI16]

Verifiable FSS problem:

- Client sends FSS keys to servers
- Do the FSS keys satisfy some property?

Servers should only learn that the keys are for a function f that satisfies some property

… but the servers should not learn the function f !



Verifiable FSS [BGI16]

Verifiable FSS problem:

- Client sends FSS keys to servers
- Do the FSS keys satisfy some property?

Two ingredients: Randomized linear sketching + Verification protocol

- Randomized linear sketching: shrinks FSS outputs down to a small number of values that the servers can check to verify some property
- Verification protocol: the servers run the verification check without revealing the inputs and only reveal the result (multi-party computation — next week!)

Verifiable FSS [BGI16]

Verifiable FSS template (parameterized by a finite field $\mathbb{F}$, domain size n, small degree d):

1. Servers pick some random matrix $\mathbf{L} \in \mathbb{F}^{d \times n}$
2. Run FSS Eval to output n values in $\mathbb{F}$
3. Multiply the random matrix by the results of Eval
4. Run a verification protocol to check that the output satisfies some property

Check that FSS key is for function where $f(\alpha) = 1, f(x) = 0$ for $x \neq \alpha$

$$\mathbf{L} = \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_n \\ r_1^2 & r_2^2 & r_3^2 & \dots & r_n^2 \end{bmatrix} \times \begin{bmatrix} \mathsf{Eval}(K,1) \\ \mathsf{Eval}(K,2) \\ \mathsf{Eval}(K,3) \\ \dots \\ \mathsf{Eval}(K,n) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Check: $x^2 = y$

Verifiable FSS [BGI16]

Check that DPF key is for function where $f(\alpha) = 1, f(x) = 0$ for $x \neq \alpha$

$$\mathbf{L} = \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_n \\ r_1^2 & r_2^2 & r_3^2 & \dots & r_n^2 \end{bmatrix} \times \begin{bmatrix} \mathsf{Eval}(K,1) \\ \mathsf{Eval}(K,2) \\ \mathsf{Eval}(K,3) \\ \\ \mathsf{Eval}(K,n) \end{bmatrix} = \begin{bmatrix} x \\ \\ y \end{bmatrix}$$

Check: $x^2 = y$

If f is not a unit vector or the all-zeroes vector, using field $\mathbb{F}$, the probability of passing the verification check is $O(1/\mathbb{F})$

Verifiable FSS [BGI16]

Check that DPF key is for function where $f(\alpha) = 1, f(x) = 0$ for $x \neq \alpha$

$$\mathbf{L} = \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_n \\ r_1^2 & r_2^2 & r_3^2 & \dots & r_n^2 \end{bmatrix} \times \begin{bmatrix} \mathsf{Eval}(K,1) \\ \mathsf{Eval}(K,2) \\ \mathsf{Eval}(K,3) \\ \\ \mathsf{Eval}(K,n) \end{bmatrix} = \begin{bmatrix} x \\ \\ y \end{bmatrix}$$

Check: $x^2 = y$

Limitation: only provides privacy if servers are semi honest (honest-but-curious), but not malicious

Verifiable FSS [BGI16]

Check that DPF key is for function where $f(\alpha) = 1, f(x) = 0$ for $x \neq \alpha$

$$\mathbf{L} = \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_n \\ r_1^2 & r_2^2 & r_3^2 & \dots & r_n^2 \end{bmatrix} \times \begin{bmatrix} \mathsf{Eval}(K,1) \\ \mathsf{Eval}(K,2) \\ \mathsf{Eval}(K,3) - 1 \\ \\ \mathsf{Eval}(K,n) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Check: $x^2 = y$

Limitation: only provides privacy if servers are semi honest (honest-but-curious), but not malicious

Server can alter FSS evaluation at one point:

- If check *does not* succeed → the changed location *was not* the special (non-zero) point
- If the check *does* succeed →the changed location *was* the special point (vector is all zeros)

Verifiable FSS [BGI16]

Check that DPF key is for function where $f(\alpha) = 1, f(x) = 0$ for $x \neq \alpha$

$$\mathbf{L} = \begin{bmatrix} r_1 & r_2 & r_3 & \dots & r_n \\ r_1^2 & r_2^2 & r_3^2 & \dots & r_n^2 \end{bmatrix} \times \begin{bmatrix} \mathsf{Eval}(K,1) \\ \mathsf{Eval}(K,2) \\ \mathsf{Eval}(K,3) \\ \\ \mathsf{Eval}(K,n) \end{bmatrix} = \begin{bmatrix} x \\ \\ y \end{bmatrix}$$

Check: $x^2 = y$

Construction with similar structure for f where $f(\alpha) = \beta, f(x) = 0$ for $x \neq \alpha$ and arbitrary β

Later this class: proofs on secret-shared data with malicious security (Prio)

When would you use Tor vs. Riposte?

Tor

- Low-latency
- Point-to-point connections
- Can use to connect to existing websites
- Vulnerable to traffic-analysis attacks

Riposte

- Only provides a bulletin board abstraction
- Have to wait until the end of an epoch for an update to be posted
- Higher overhead than Tor
- Relies on centralized servers, which could become a target
- Defends against traffic-analysis attacks

Outline

1. Riposte
2. **Logistics**
3. Student presentation

Logistics

Project progress report due 11/11 (1.5 weeks)

- Related work survey, progress made to this point
- Goal (but not requirement): have draft of design to get feedback
- Optional feedback meetings after progress reports due

Next lecture: Guest lecture from Riana Pfefferkorn

Outline

1. Riposte
2. Logistics
3. **Student presentation**

References

Boyle, Elette, Niv Gilboa, and Yuval Ishai. "Function secret sharing." In *Annual international conference on the theory and applications of cryptographic techniques*, pp. 337-367. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

Boyle, Elette, Niv Gilboa, and Yuval Ishai. "Function secret sharing: Improvements and extensions." In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1292-1303. 2016.

Chaum, David. "The dining cryptographers problem: Unconditional sender and recipient untraceability." Journal of cryptology 1, no. 1 (1988): 65-75.

Chaum, David L. "Untraceable electronic mail, return addresses, and digital pseudonyms." Communications of the ACM 24, no. 2 (1981): 84-90.

Corrigan-Gibbs, Henry, Dan Boneh, and David Mazières. "Riposte: An anonymous messaging system handling millions of users." In *2015 IEEE Symposium on Security and Privacy*, pp. 321-338. IEEE, 2015.

Dingledine, Roger, Nick Mathewson, and Paul Syverson. "Tor: The second-generation onion router." (2004).