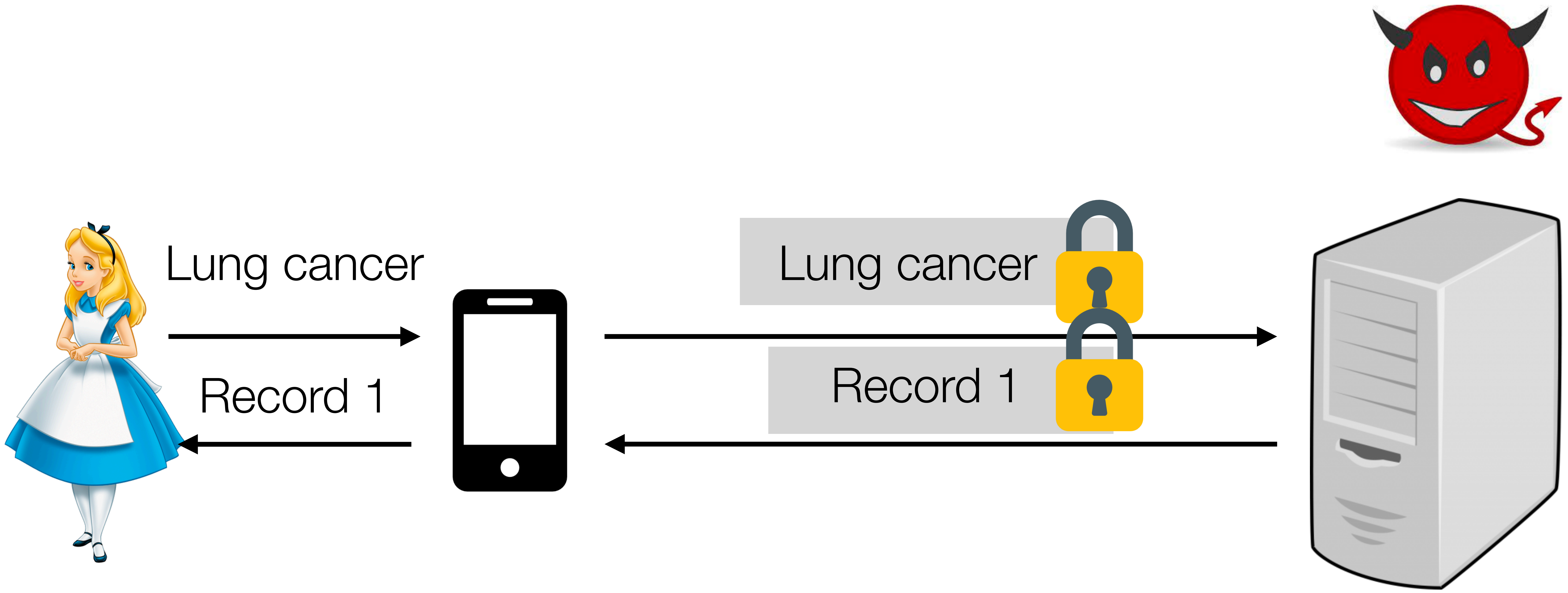
CS 350S: Privacy-Preserving Systems

Private Information Retrieval II

Recap: Private information retrieval

Goal: Attacker that controls the server learns nothing about a user's query

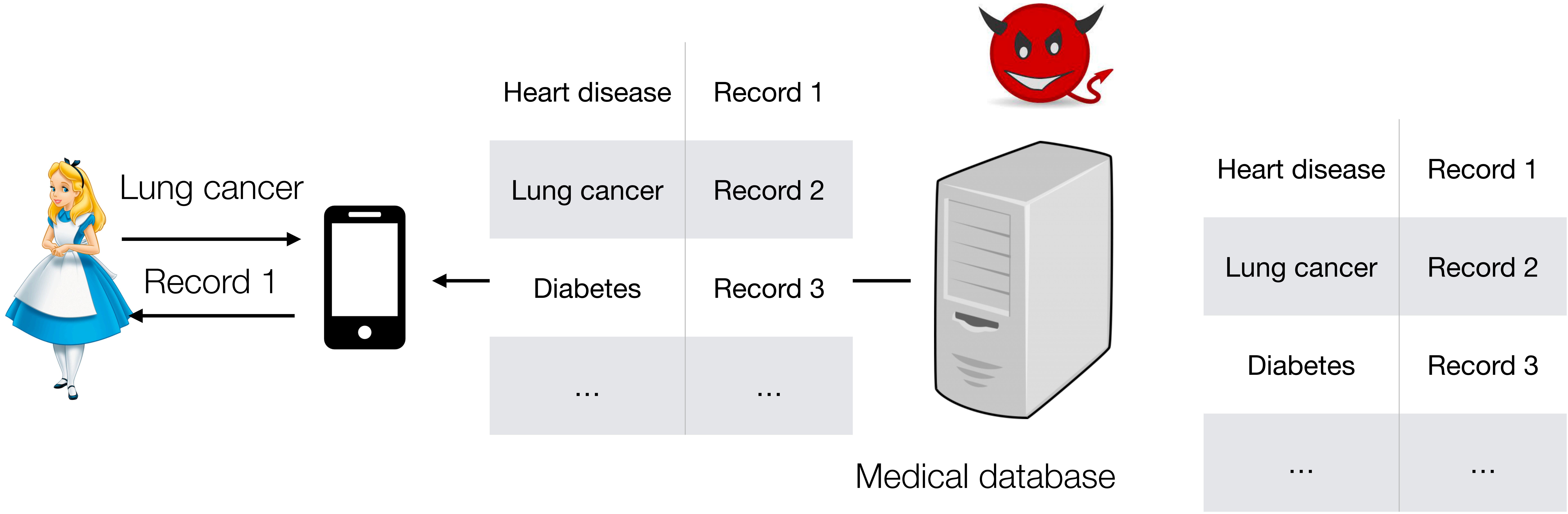


Medical database

Heart disease	Record 1
Lung cancer	Record 2
Diabetes	Record 3
…	…

Can also use to fetch articles, images, podcasts, movies, etc. from a remote server

Trivial solution: download the whole database

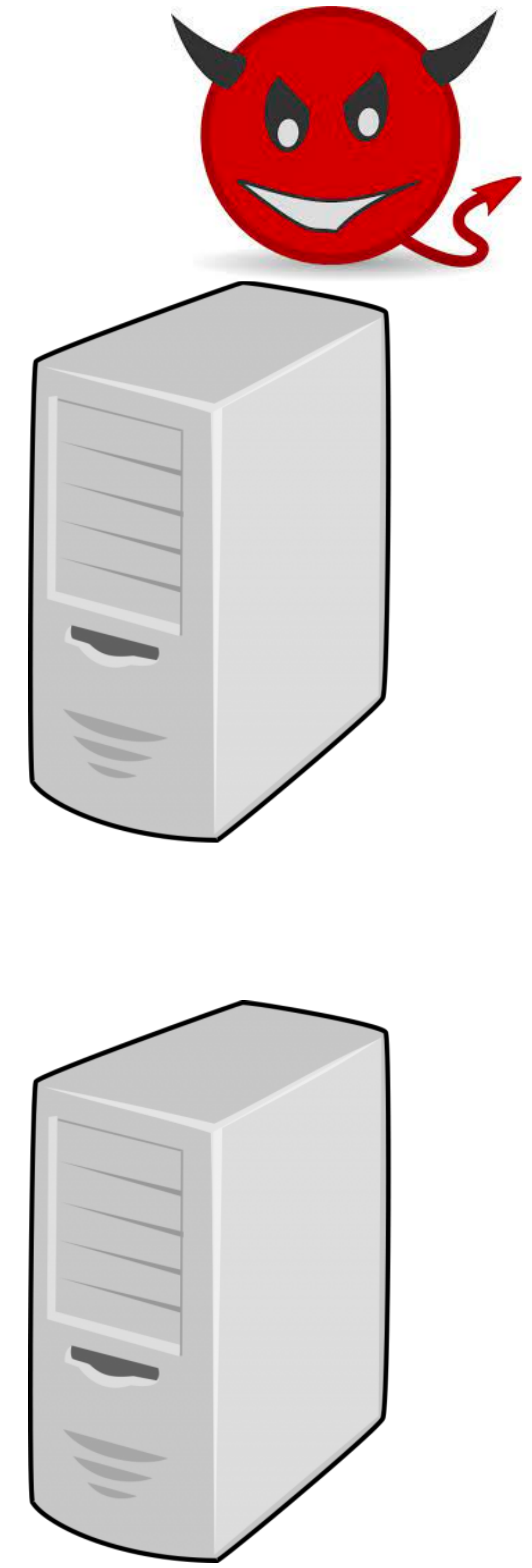


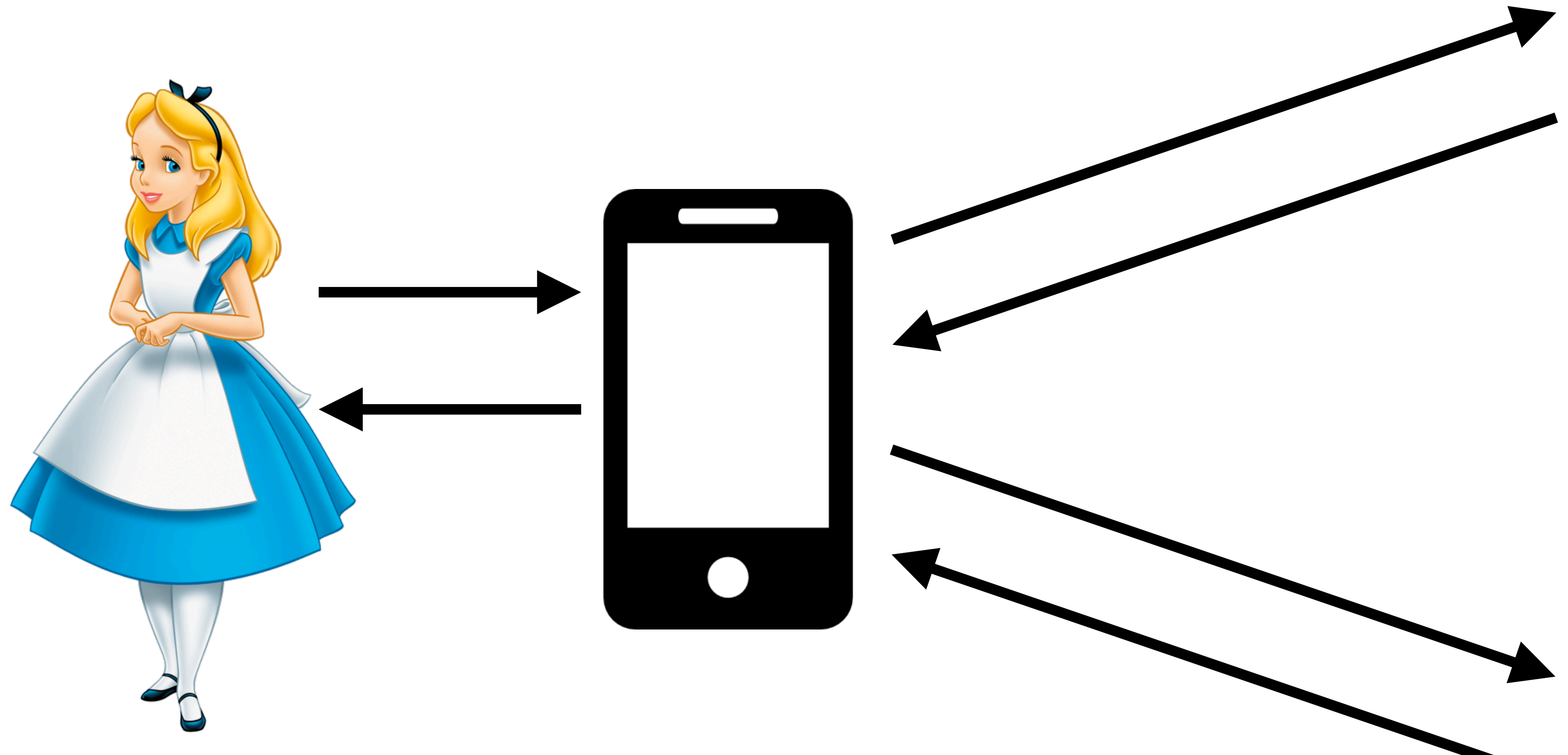
Want to privately query a database with communication *sublinear* in database size

Two models

Multi-server PIR [CGKS’95]

Protects privacy if at least one server is honest



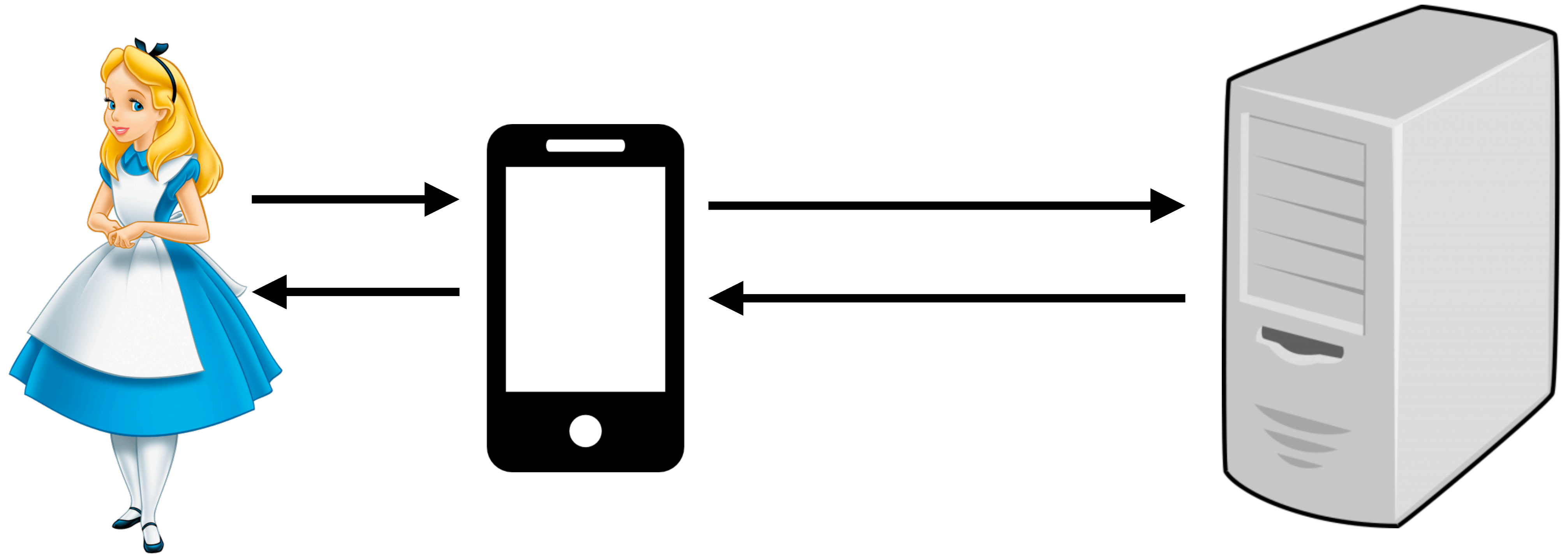


Can be information-theoretic

TODAY

Single-server PIR [KO’97]

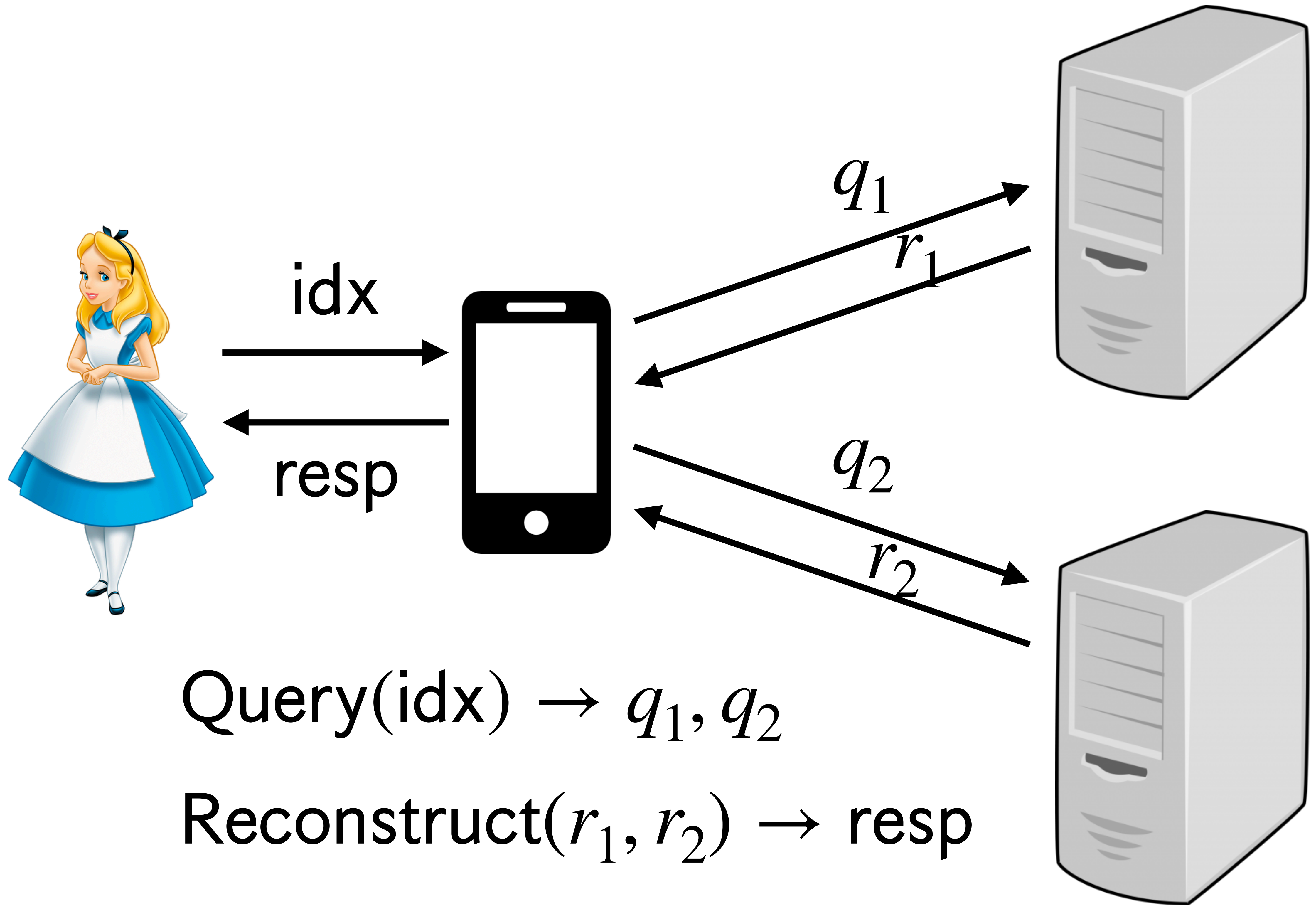
Protects privacy if the server is compromised



Requires a cryptographic assumption

Two-server PIR

Answer(db, q_1) $\rightarrow r_1$



Query(idx) $\rightarrow q_1, q_2$

Reconstruct(r_1, r_2) $\rightarrow$ resp

Answer(db, q_2) $\rightarrow r_2$

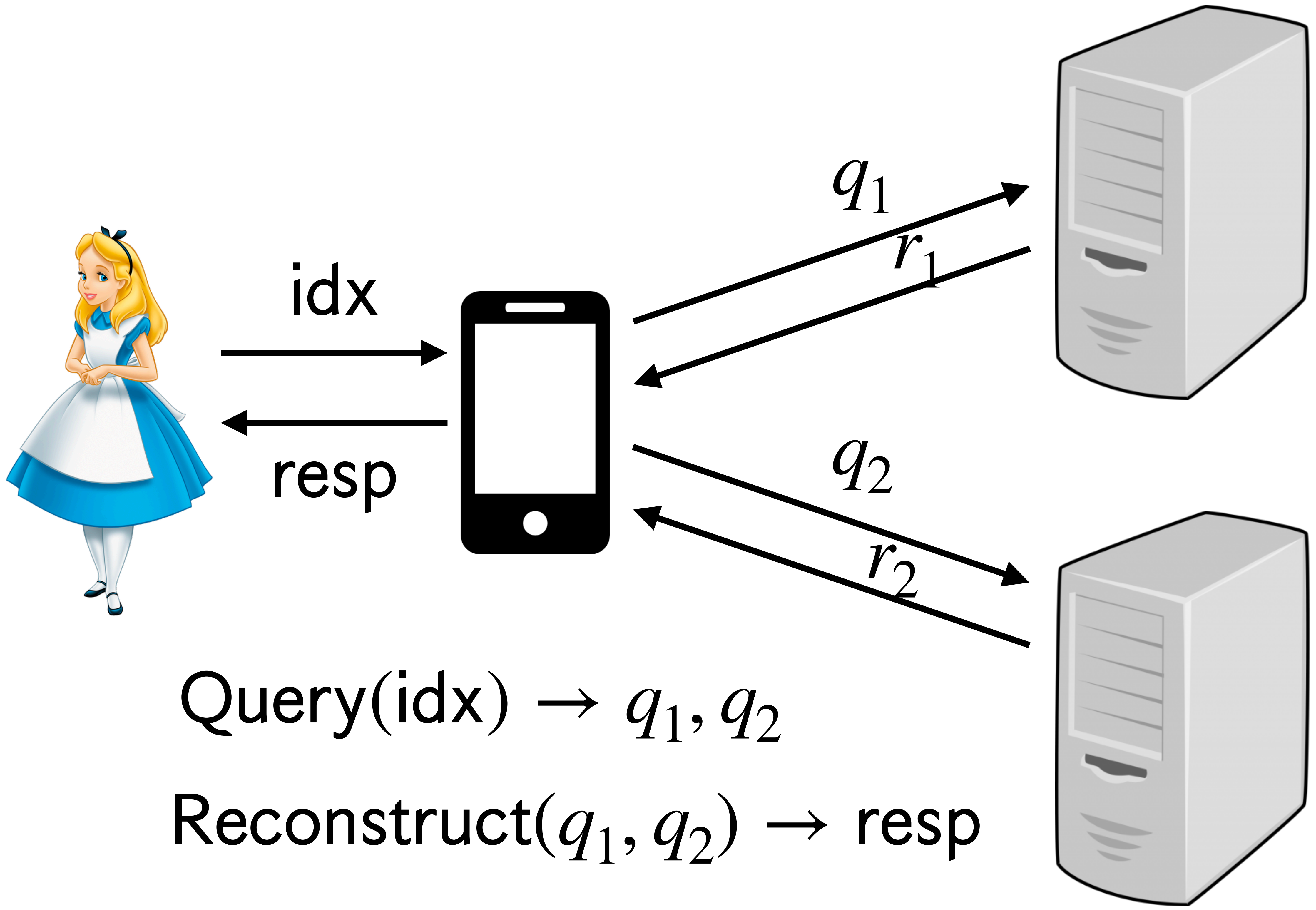
For database with n elements, query space Q, values in $\{0,1\}$, and response space R

Query : $[n] \rightarrow Q^2$
Answer : $\{0,1\}^n \times Q \rightarrow R$
Reconstruct : $R^2 \rightarrow \{0,1\}$

Note: **Query** is randomized and takes security parameter as implicit input

Two-server PIR definitions

Answer(db, q_1) $\rightarrow r_1$



Query(idx) $\rightarrow q_1, q_2$

Reconstruct(q_1, q_2) $\rightarrow$ resp

Answer(db, q_2) $\rightarrow r_2$

***Correctness*: For all $n \in \mathbb{N}$, db $\in \{0,1\}^n$, idx $\in [n]$, the probability that:**

Query(idx) $\rightarrow q_1, q_2$

Answer(db, q_i) $\rightarrow r_i$ **for** $i \in \{1,2\}$

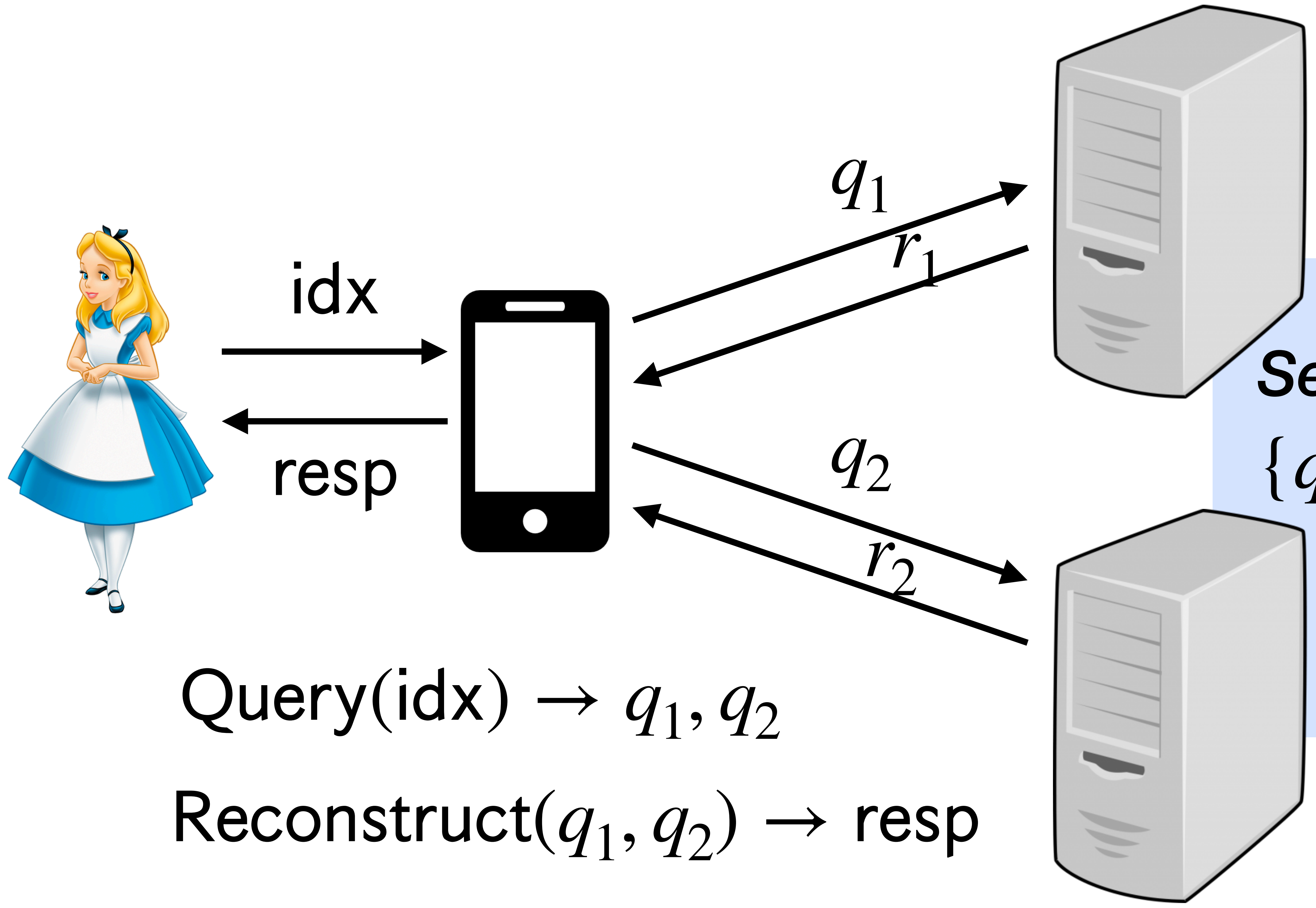
Reconstruct(q_1, q_2) $=$ db_{idx}

is 1.

Informally, the client gets the requested record

Two-server PIR definitions

$\text{Answer}(\text{db}, q_1) \rightarrow r_1$



$\text{Query}(\text{idx}) \rightarrow q_1, q_2$

$\text{Reconstruct}(q_1, q_2) \rightarrow \text{resp}$

$\text{Answer}(\text{db}, q_2) \rightarrow r_2$

Security: **For all** $i, j \in [n], b \in \{1,2\}$

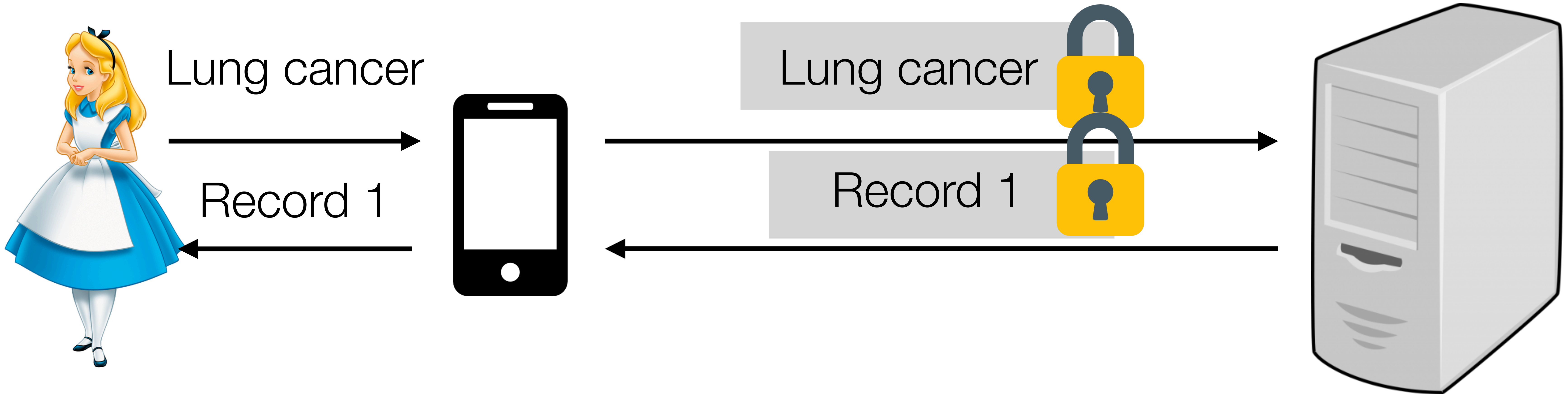
$\{q_b : \text{Query}(i) \rightarrow q_1, q_2\} \approx \{q'_b : \text{Query}(j) \rightarrow q'_1, q'_2\}$

Informally, no one server can learn the client's query

What we can hope for with PIR

Is it possible to build a PIR scheme that only touches part of the database?

No! If query execution only touches part of the database, then an attacker can learn which part of the database is *not* accessed by the query



Heart disease	Record 1
Lung cancer	Record 2
Diabetes	Record 3
…	…

The server must touch every bit of the database to respond to a client's query (preprocessing is permitted) [BIM'00]

Outline

1. **Function secret sharing**
2. Splinter
3. Searching on encrypted data

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)

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

Function secret sharing

[Boyle, Gilboa, Ishai]

Security parameter λ, function f

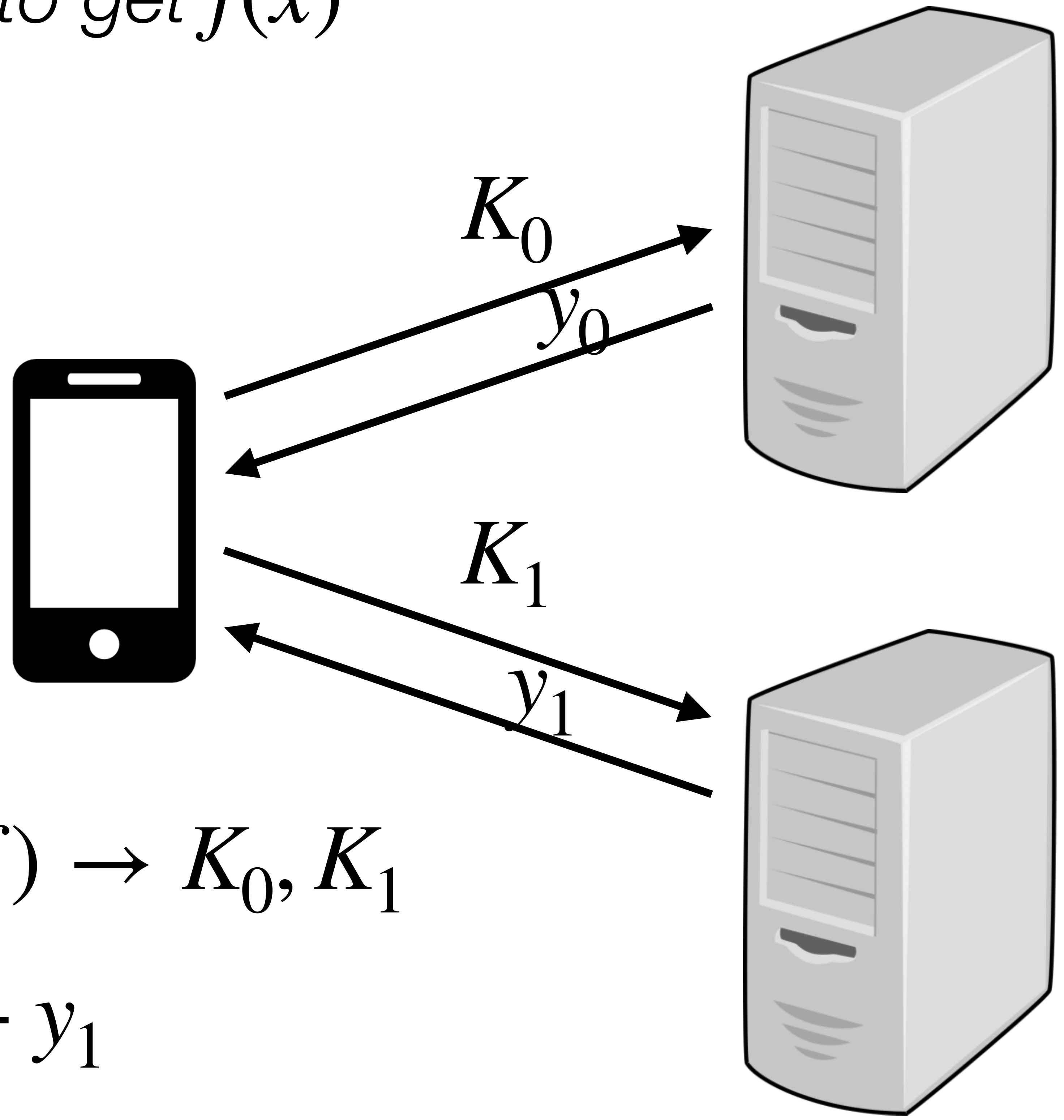
Considering just two parties

Two algorithms:

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

Client has secret f, wants to get $f(x)$

Servers have x



Eval$(K_0, x) \to y_0$

Eval$(K_1, x) \to y_1$

Gen$(1^\lambda, f) \to K_0, K_1$

$y \leftarrow y_0 + y_1$

FSS definitions

FSS is defined over a function class $\mathscr{F}$ with security parameter λ has the following syntax:

- $\mathsf{Gen}(1^\lambda, f) \rightarrow K_0, K_1$: On input security parameter and $f \in \mathscr{F}$, output keys K_0, K_1
- $\mathsf{Eval}(K_i, x) \rightarrow y_i$: On input key K_i (output from Gen) and input string $x \in \mathscr{D}_f$, where $\mathscr{D}_f$ is the input domain of function f, output a share of $f(x)$

Correctness: For all $f \in \mathscr{F}, x \in \mathscr{D}_f$,

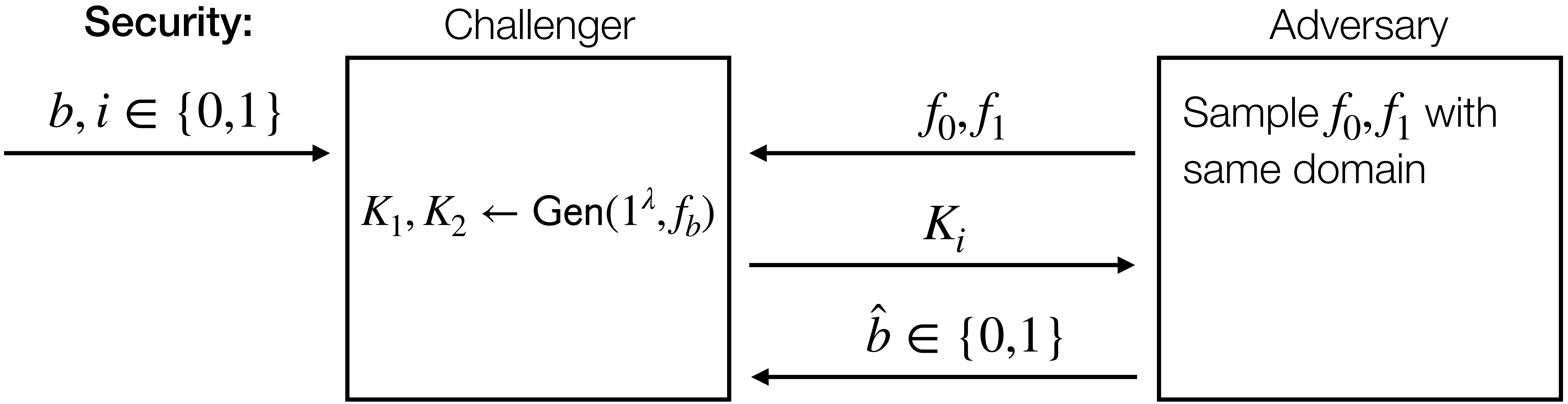
$$\Pr[K_1, K_2 \leftarrow \mathsf{Gen}(1^\lambda, f) : \mathsf{Eval}(K_0, x) + \mathsf{Eval}(K_1, x) = f(x)] = 1$$

FSS definitions

FSS is defined over a function class $\mathcal{F}$ with security parameter λ has the following syntax:

- $\mathsf{Gen}(1^\lambda, f) \rightarrow K_0, K_1$: On input security parameter and $f \in \mathcal{F}$, output keys K_0, K_1
- $\mathsf{Eval}(K_i, x) \rightarrow y_i$: On input key K_i (output from Gen) and input string $x \in \mathcal{D}_f$, where $\mathcal{D}_f$ is the input domain of function f, output a share of $f(x)$

Security:



Similar to semantic security

Functions with efficient FSS constructions

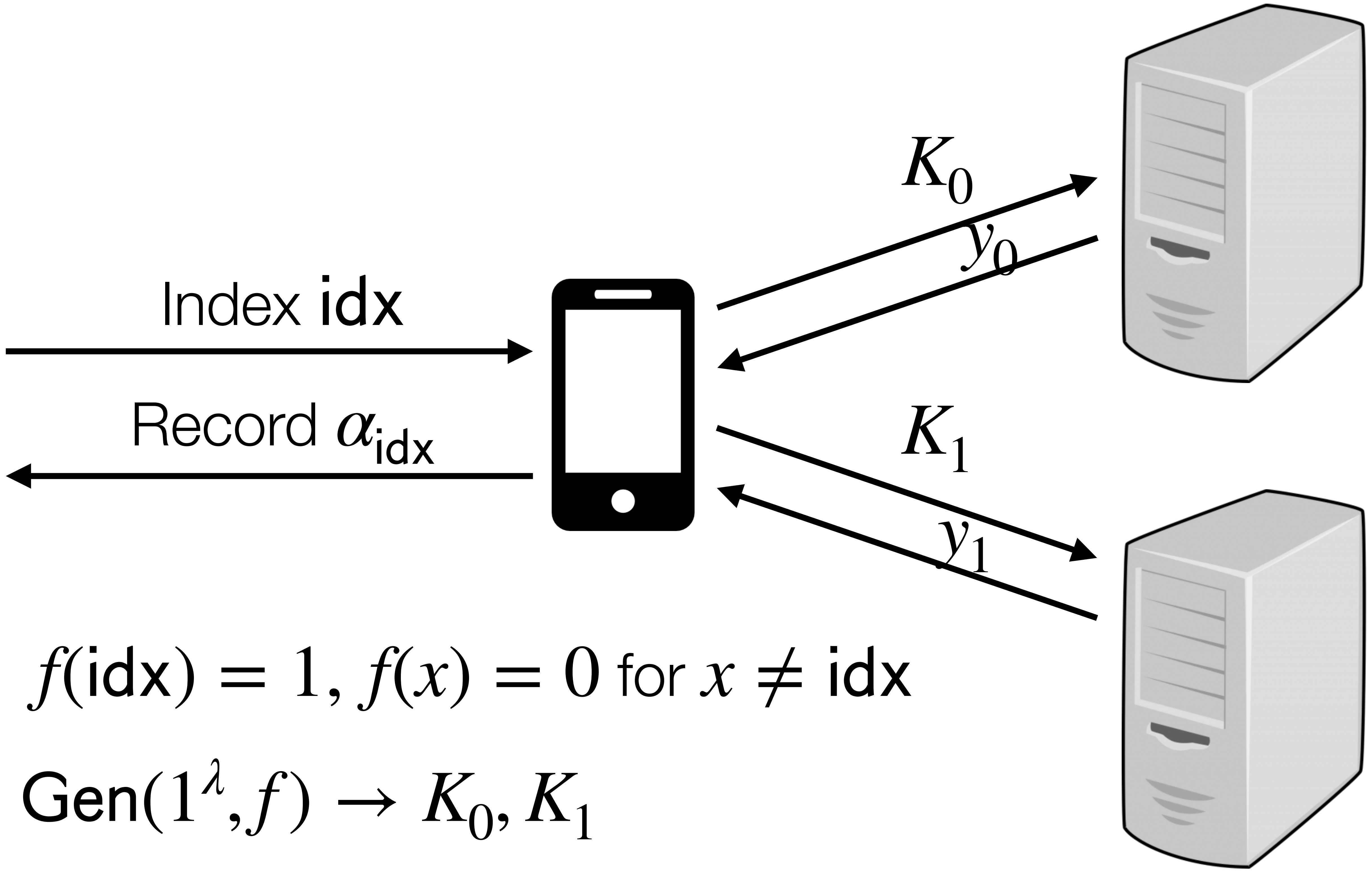
Point functions: For a, b, the corresponding point function f is defined as $f(a) = b$ and, for all $x \neq a, f(x) = 0$

- Corresponding FSS construction: distributed point function (DPF)

Comparison functions: For a, b, the corresponding comparison function f is defined as $f(x) = b$ for $x < a$ and $f(x) = 0$ for $x \geq a$

- Corresponding FSS construction: distributed comparison function (DCF)
- Can generalize to > and intervals

PIR from distributed point functions



$1 : \alpha_1$ $\quad \mathsf{Eval}(K_0, 1) \cdot \alpha_1 +$

$2 : \alpha_2$ $\quad \mathsf{Eval}(K_0, 2) \cdot \alpha_2 +$ $\quad = y_0$

$\dots$ $\quad \dots$

$n : \alpha_n$ $\quad \mathsf{Eval}(K_0, n) \cdot \alpha_n$

$1 : \alpha_1$ $\quad \mathsf{Eval}(K_1, 1) \cdot \alpha_1 +$

$2 : \alpha_2$ $\quad \mathsf{Eval}(K_1, 2) \cdot \alpha_2 +$ $\quad = y_1$

$\dots$ $\quad \dots$

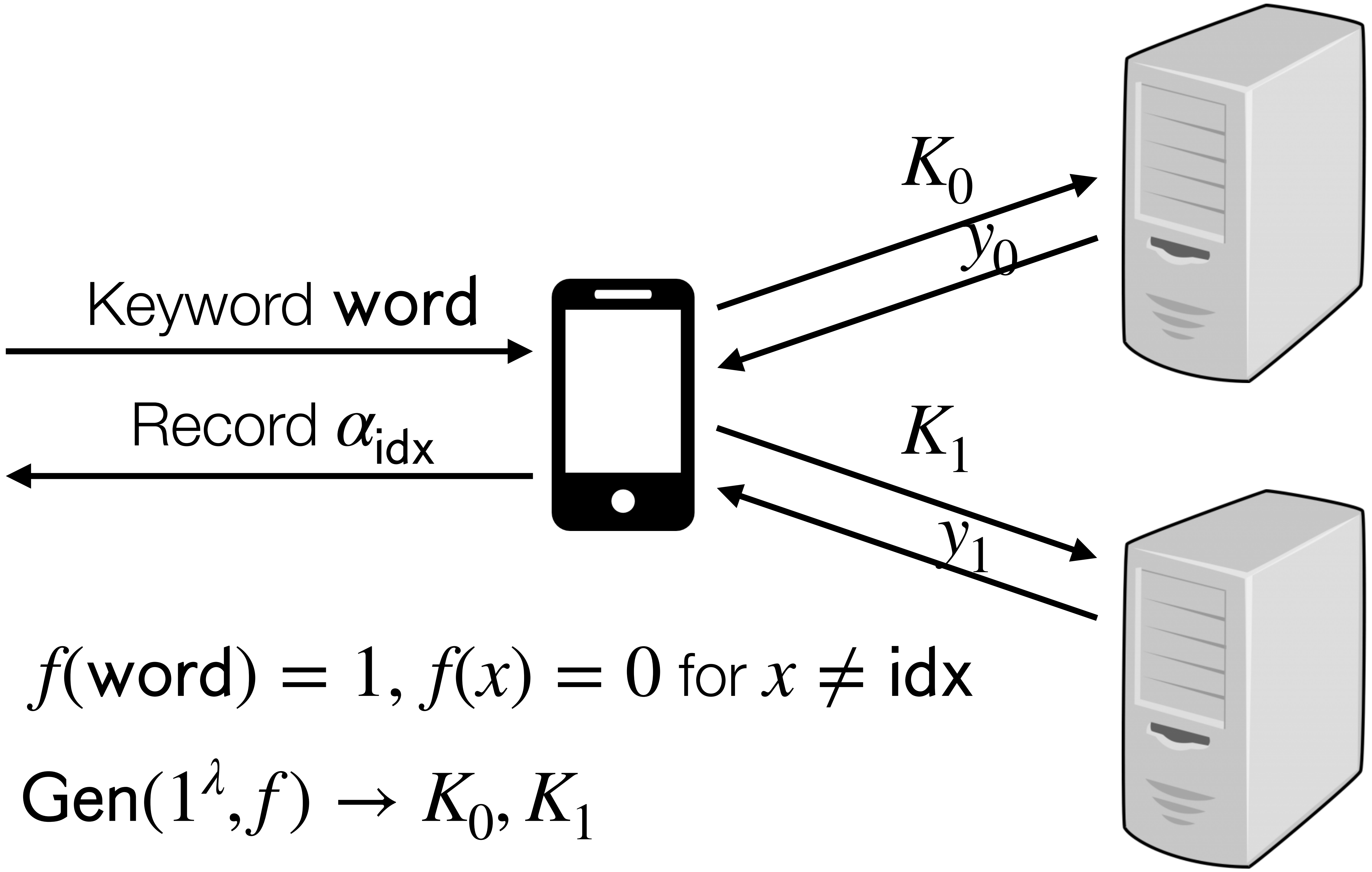
$n : \alpha_n$ $\quad \mathsf{Eval}(K_1, n) \cdot \alpha_n$

$f(\mathsf{idx}) = 1, f(x) = 0$ for $x \neq \mathsf{idx}$

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

$y \leftarrow y_0 + y_1$

PIR by keywords from DPFs



$w_1 : \alpha_1$ $\quad$ $\mathsf{Eval}(K_0, w_1) \cdot \alpha_1 +$

$w_2 : \alpha_2$ $\quad$ $\mathsf{Eval}(K_0, w_2) \cdot \alpha_2 +$ $\quad$ $= y_0$

$\cdots$ $\quad$ $\cdots$

$w_n : \alpha_n$ $\quad$ $\mathsf{Eval}(K_0, w_n) \cdot \alpha_n$

$w_1 : \alpha_1$ $\quad$ $\mathsf{Eval}(K_1, w_1) \cdot \alpha_1 +$

$w_2 : \alpha_2$ $\quad$ $\mathsf{Eval}(K_1, w_2) \cdot \alpha_2 +$ $\quad$ $= y_1$

$\cdots$ $\quad$ $\cdots$

$w_n : \alpha_n$ $\quad$ $\mathsf{Eval}(K_1, w_n) \cdot \alpha_n$

$f(\mathsf{word}) = 1$, $f(x) = 0$ for $x \neq \mathsf{idx}$

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

$y \leftarrow y_0 + y_1$

Costs for DPFs and DCFs

FSS is defined over a function class $\mathscr{F}$ with security parameter λ has the following syntax:

- $\mathsf{Gen}(1^\lambda, f) \rightarrow K_0, K_1$: On input security parameter and $f \in \mathscr{F}$, output keys K_0, K_1

- $\mathsf{Eval}(K_i, x) \rightarrow y_i$: On input key K_i (output from Gen) and input string $x \in \mathscr{D}_f$, where $\mathscr{D}_f$ is the input domain of function f, output a share of $f(x)$

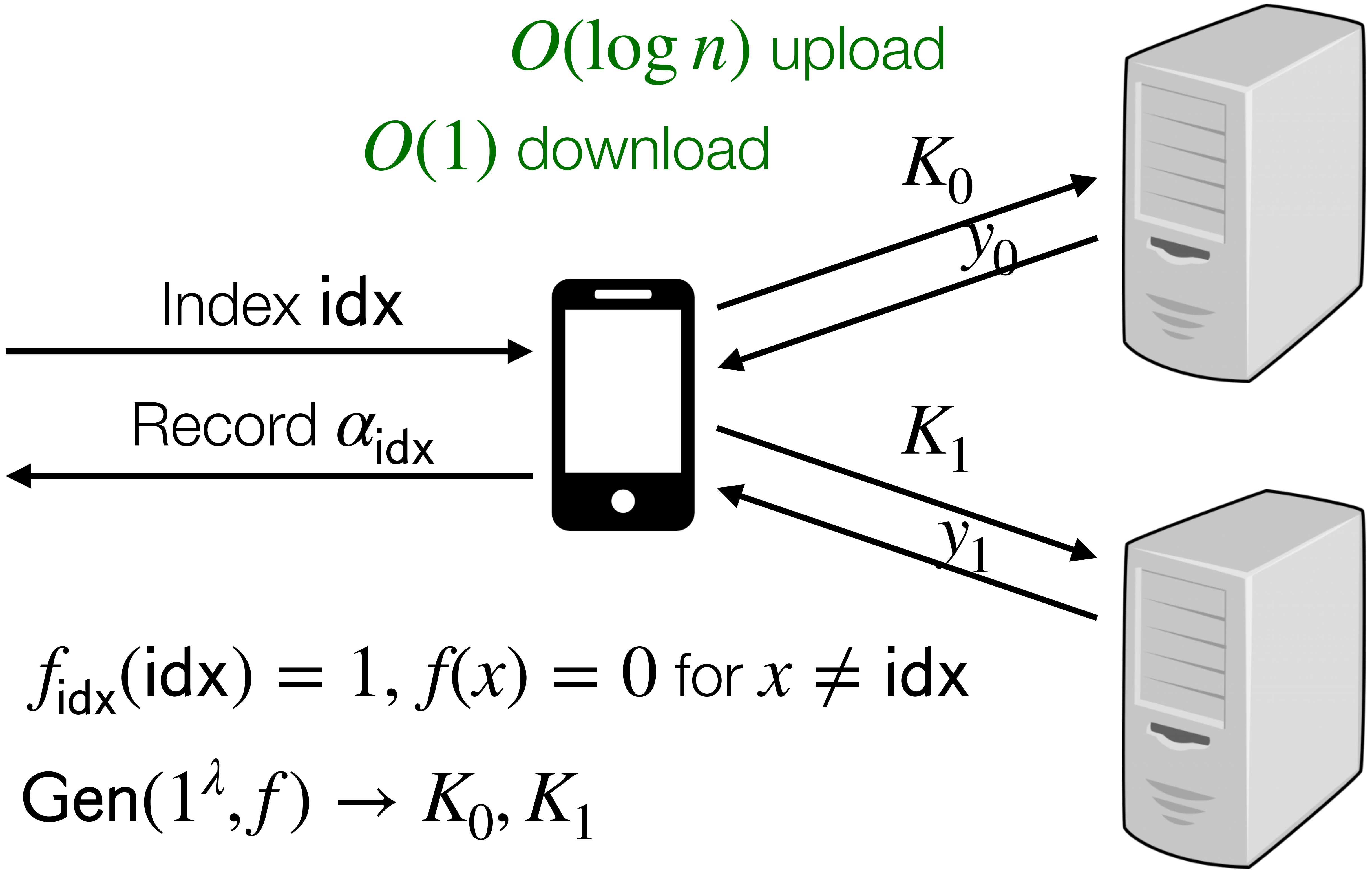
Only AES operations

For $\mathscr{D} = \{0,1\}^m$, key size is $O(m)$

Costs for PIR from DPFs

$O(\log n)$ upload

$O(1)$ download



$f_{\mathsf{idx}}(\mathsf{idx}) = 1,\ f(x) = 0$ for $x \neq \mathsf{idx}$

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

$y \leftarrow y_0 + y_1$

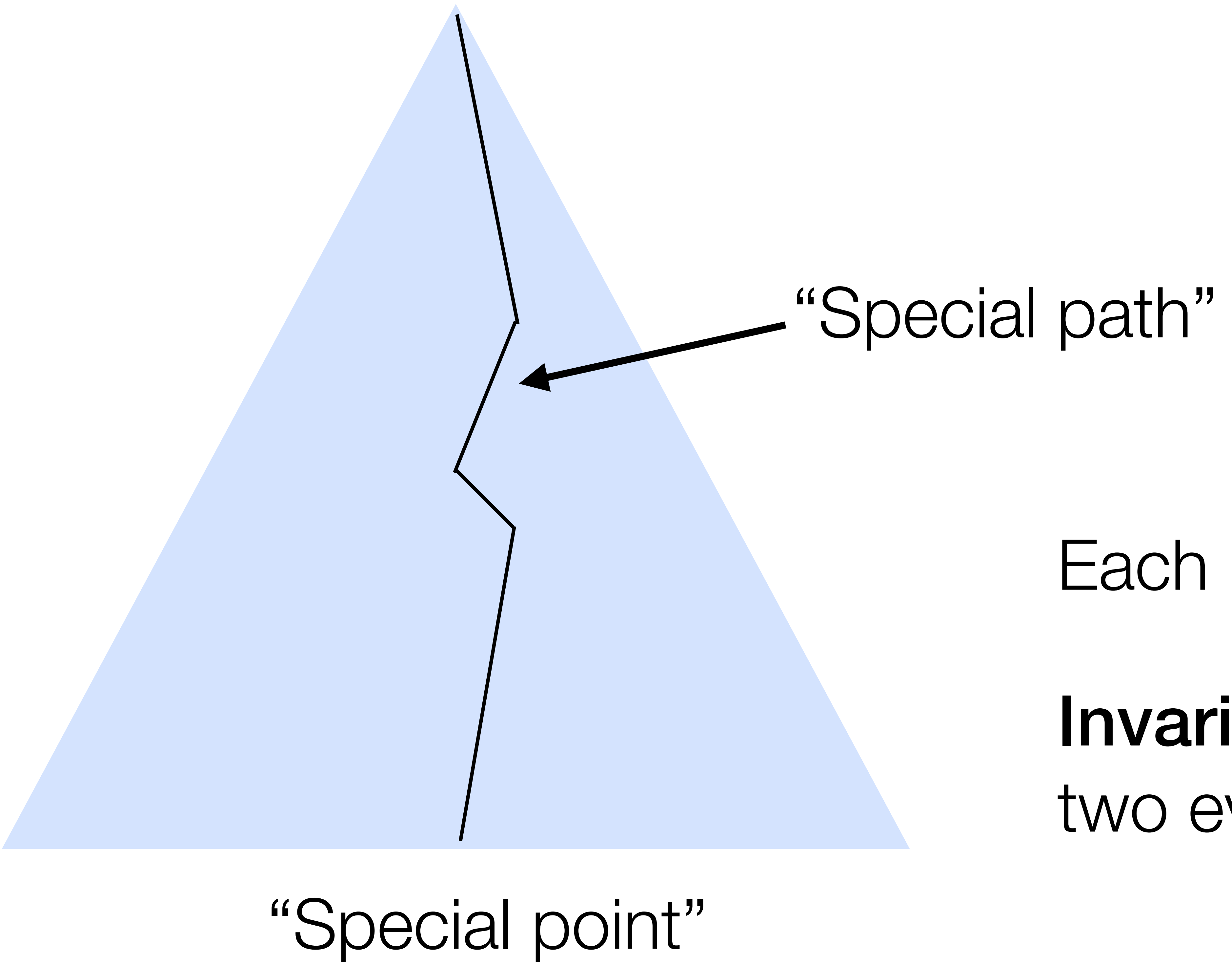
Linear scan over data

$1 : \alpha_1$	$\mathsf{Eval}(K_0,1) \cdot \alpha_1 +$	
$2 : \alpha_2$	$\mathsf{Eval}(K_0,2) \cdot \alpha_2 +$	$= y_0$
$\ldots$		
$n : \alpha_n$	$\mathsf{Eval}(K_0, n) \cdot \alpha_n$	
$1 : \alpha_1$	$\mathsf{Eval}(K_1,1) \cdot \alpha_1 +$	
$2 : \alpha_2$	$\mathsf{Eval}(K_1,2) \cdot \alpha_2 +$	$= y_1$
$\ldots$		
$n : \alpha_n$	$\mathsf{Eval}(K_1, n) \cdot \alpha_n$	

DPF construction intuition

Observation: If K_0 outputs y_0, K_1 outputs y_1, and $y_0 = y_1$, then if we assemble shares as $y_0 - y_1$, (y_0, y_1) are shares of 0.

Construction idea: Each server can use its key to construct a tree that is identical to the other server's tree (secret shares of 0) *except* along a path from the root to the non-zero ("special") point.



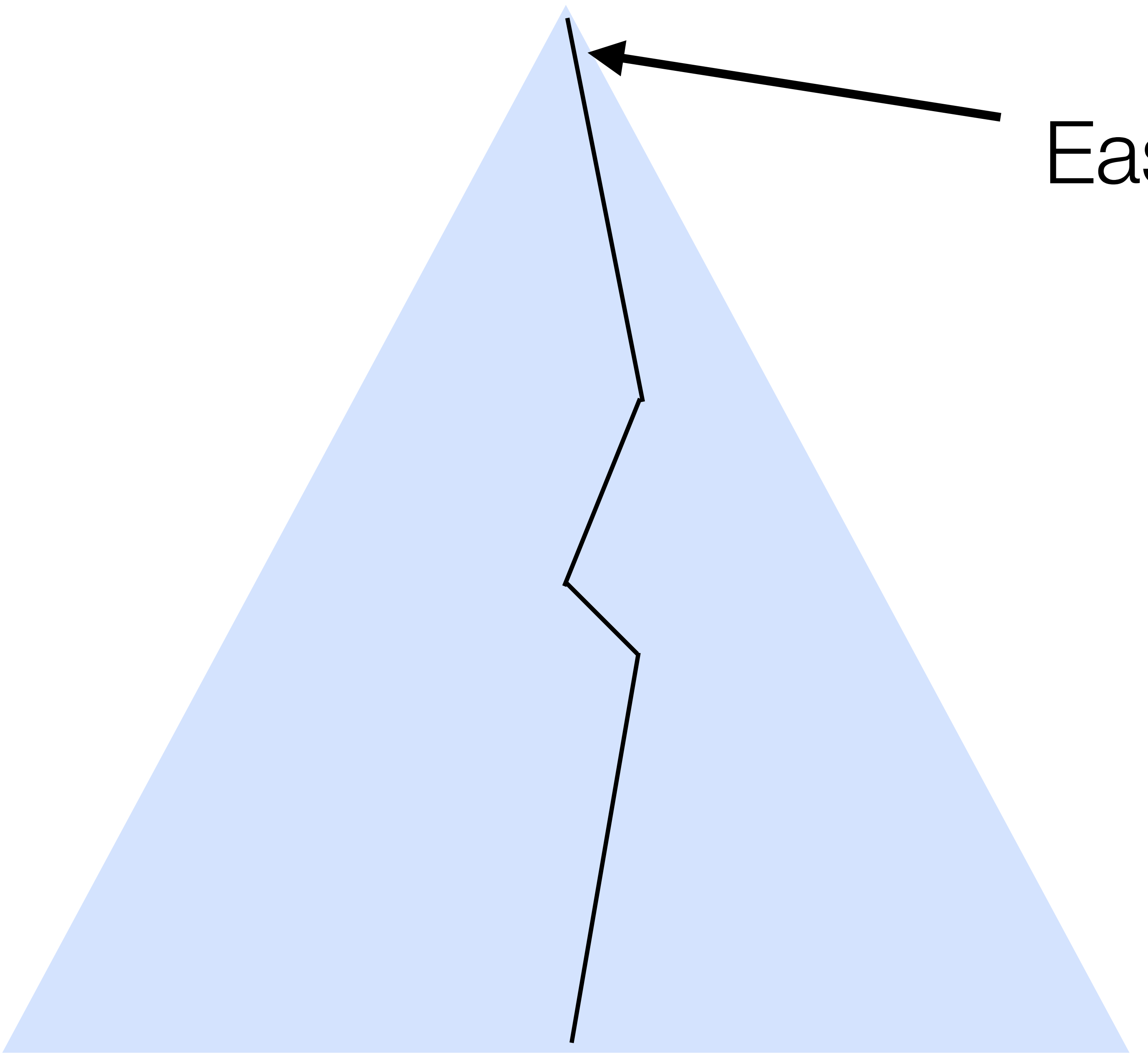
Each node is associated with a seed and a control bit

Invariant: Seed and control bit are identical across the two evaluations, except along the special path

DPF construction intuition

Construction idea: Each server can use its key to construct a tree that is identical to the other server's tree (secret shares of 0) *except* along a path from the root to the non-zero ("special") point.

Invariant: Seed and control bit are identical across the two evaluations, except along the special path



Easy to maintain invariant at root (always on special path)

What about when we leave the special path?

Idea: use a "correction word" to set the node values to the same value when leaving the special path

- $[s] \oplus [t] \cdot \mathsf{CW}$ where s is the seed, t is the control bit, and CW is the correction word — correction conditioned on t

Applications of FSS beyond PIR

Privately writing to a shared database

- Want to hide which element the client is writing to
- Applications to metadata-hiding messaging (see later in class) [Riposte]

Private aggregate statistics

- Privately collect histograms
- Compute most popular set of strings without revealing all strings [Poplar]

Multiparty computation

- Execute some computation across parties, but without each party revealing its inputs to the other parties [BGI19]

Outline

1. Function secret sharing
2. **Splinter**
3. Searching on encrypted data

Splinter

[Wang, Yun, Goldwasser, Vaikuntanathan, Zaharia]

Gap between PIR and what applications need:

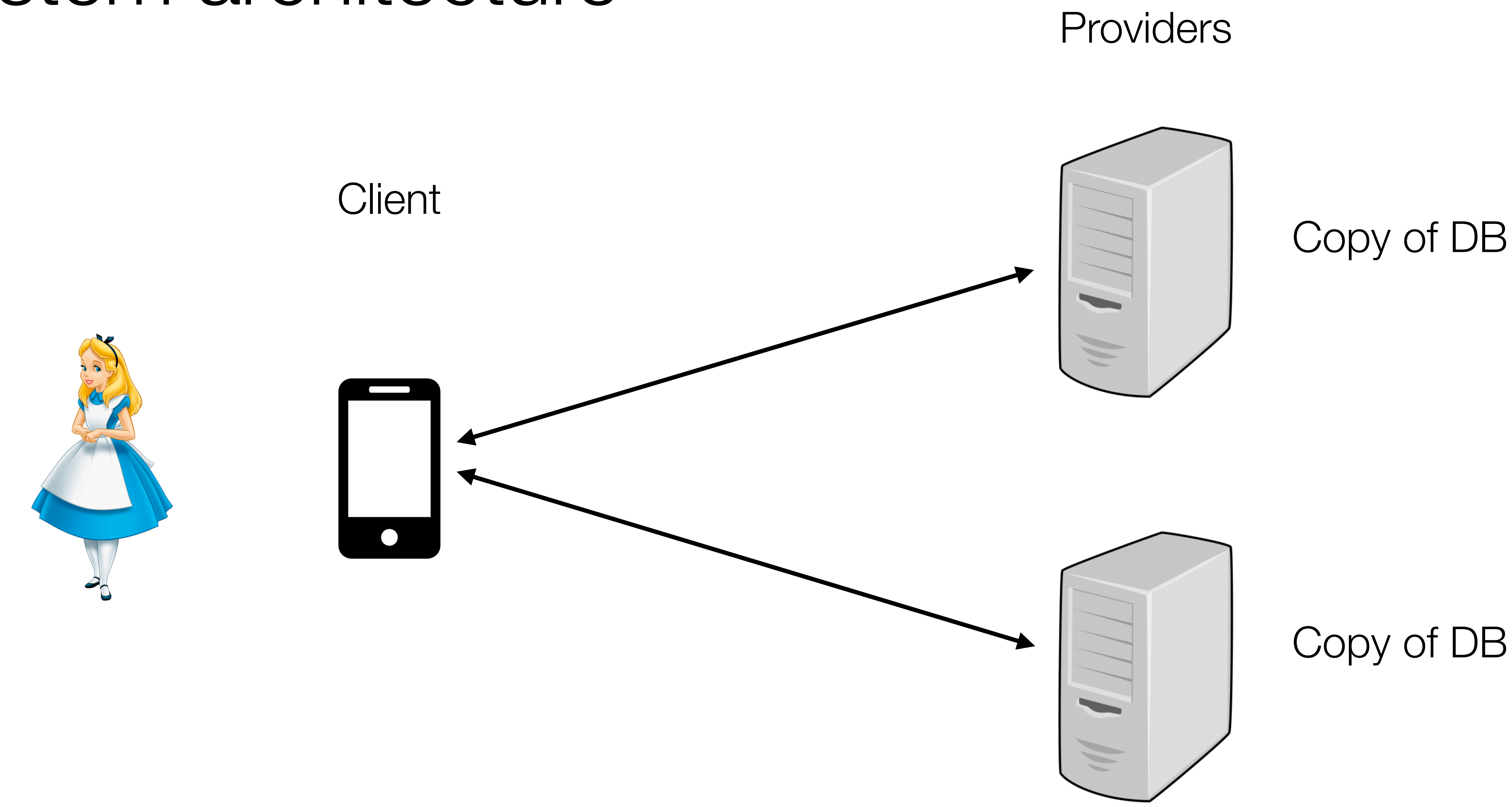
- Private information retrieval: Privately look up a record in a database by some index
- Applications need: Private SQL-like queries based on filters

Splinter helps to bridge this gap

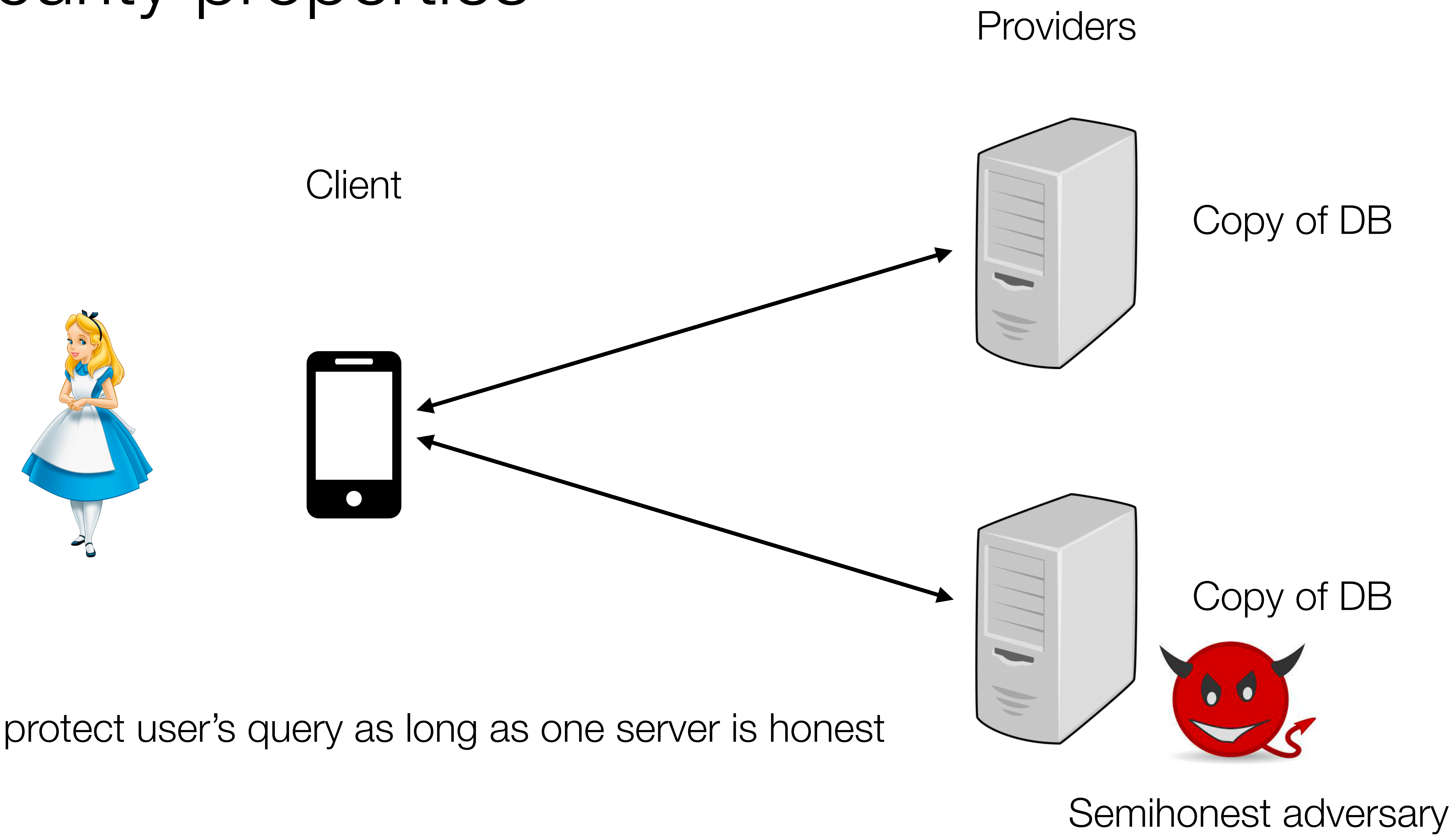
```
SELECT TOP 10 flights from flights
WHERE source = SFO AND dest = EWR
ORDER BY price
```

```
SELECT AVG(price) WHERE month=3
AND origin=SFO AND dest=EWR
```

System architecture



Security properties



Goal: protect user's query as long as one server is honest

Security properties

Splinter hides some query parameters, but not the query structure

Column names, aggregation function, and other features of query structure are not hidden from the server (only the “?” are hidden)

```
SELECT TOP 10 flights from flights
WHERE source = ? AND dest = ?
ORDER BY price
```

Queries supported

Query format:

SELECT $aggregate_1$, $aggregate_2$, ...
FROM *table*
WHERE *condition*
[GROUP BY $expr_1$, $expr_2$, ...]

aggregate:

- COUNT | SUM | AVG | STDEV (*expr*)
- MAX | MIN (*expr*)
- TOPK (*expr*, *k*, *sort_expr*)
- HISTOGRAM (*expr*, *bins*)

condition:

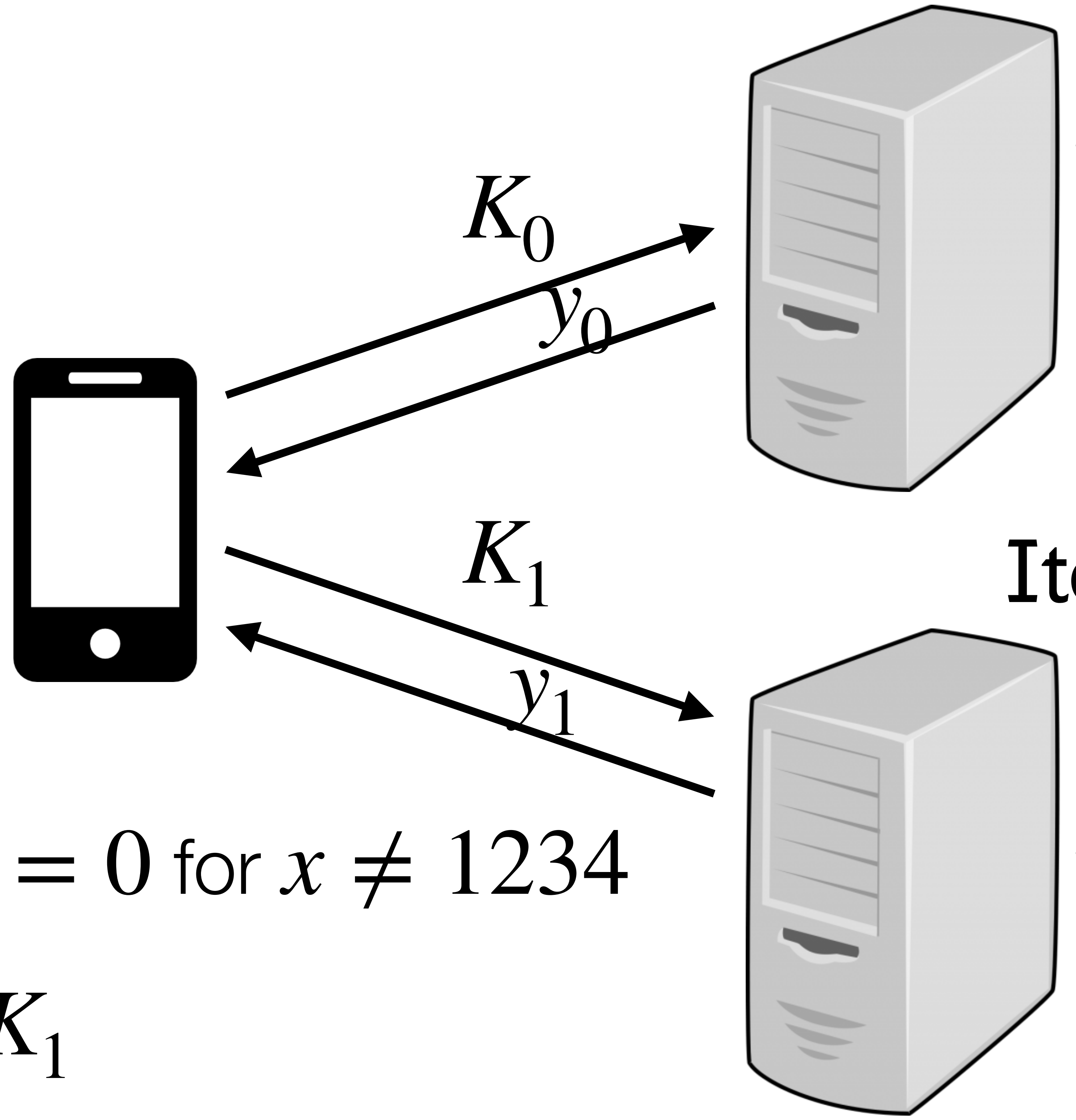
- *expr* = *secret*
- $secret_1 \leq expr \leq secret_2$
- AND of '=' conditions and up to one interval
- OR of multiple disjoint conditions (e.g., `country="UK" OR country="USA"`)

expr: any public function of the fields in a table row (e.g., `ItemId + 1` or `Price * Tax`)

Count queries

Can extend to interval conditions with interval FSS

```
SELECT COUNT(*)
FROM Items
WHERE ItemID = 1234
```



ItemID : Price

$1234 : \$10$ $\mathsf{Eval}(K_0, 1234) +$

$5678 : \$20$ $\mathsf{Eval}(K_0, 5678) +$ $= y_0$

$\ldots$ $\ldots$

$1111 : \$50$ $\mathsf{Eval}(K_0, 1111)$

ItemID : Price

$1234 : \$10$ $\mathsf{Eval}(K_1, 1234) +$

$5678 : \$20$ $\mathsf{Eval}(K_1, 5678) +$ $= y_1$

$\ldots$ $\ldots$

$1111 : \$50$ $\mathsf{Eval}(K_1, 1111)$

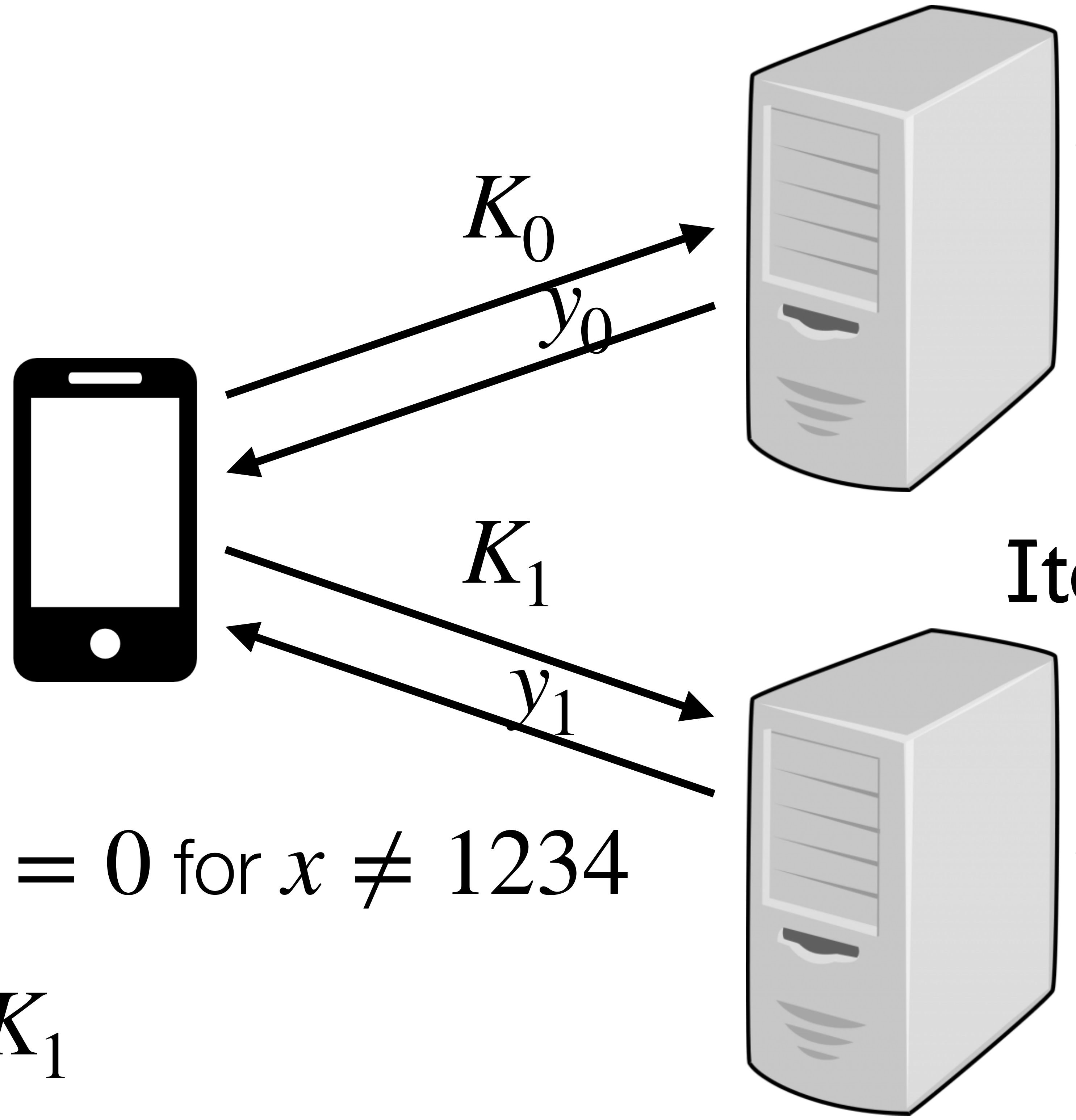
$f(1234) = 1, f(x) = 0$ for $x \neq 1234$

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

$y \leftarrow y_0 + y_1$

Sum queries

Can extend to interval conditions with interval FSS




```
SELECT SUM(Price)
FROM Items
WHERE ItemID = 1234
```

$f(1234) = 1$, $f(x) = 0$ for $x \neq 1234$

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

$y \leftarrow y_0 + y_1$

ItemID : Price

$1234 : \$10$
$5678 : \$20$
$\cdots$
$1111 : \$50$

$\mathsf{Eval}(K_0, 1234) \cdot 10 +$
$\mathsf{Eval}(K_0, 5678) \cdot 20 +$
$\cdots$
$\mathsf{Eval}(K_0, 1111) \cdot 50$
$= y_0$

ItemID : Price

$1234 : \$10$
$5678 : \$20$
$\cdots$
$1111 : \$50$

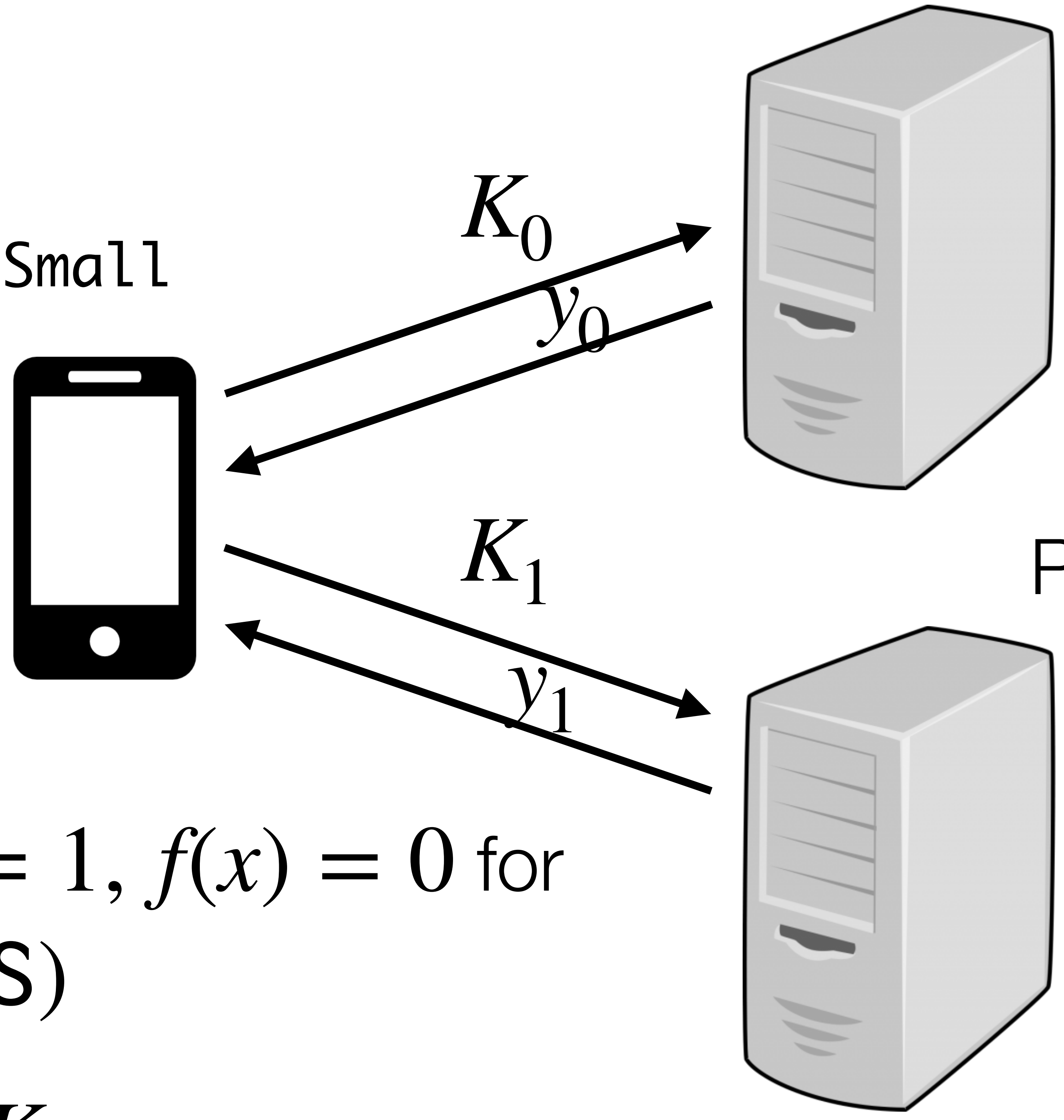
$\mathsf{Eval}(K_1, 1234) \cdot 10 +$
$\mathsf{Eval}(K_1, 5678) \cdot 20 +$
$\cdots$
$\mathsf{Eval}(K_1, 1111) \cdot 50$
$= y_1$

Extending to other linear aggregation functions

- AVG(x): Fetch SUM(x) and COUNT(x) and use to compute the mean
- STDEV(x): Fetch SUM(x) and COUNT(x), along with SUM(x^2) and COUNT(x^2)

More complex conditions: AND

```
SELECT COUNT(*)
FROM Items
WHERE Price=50 AND
Color=Blue AND Size=Small
```



Price, Color, Size

\$10, Green, M
\$20, Blue, S
…
\$50, Yellow, L

Eval$(K_0, 10||$Green$||$M$)$ +
Eval$(K_0, 20||$Blue$||$S$)$ +
…
Eval$(K_0, 50||$Yellow$||$L$)$

$= y_0$

Price, Color, Size

\$10, Green, M
\$20, Blue, S
…
\$50, Yellow, L

Eval$(K_0, 10||$Green$||$M$)$ +
Eval$(K_0, 20||$Blue$||$S$)$ +
…
Eval$(K_0, 50||$Yellow$||$L$)$

$= y_1$

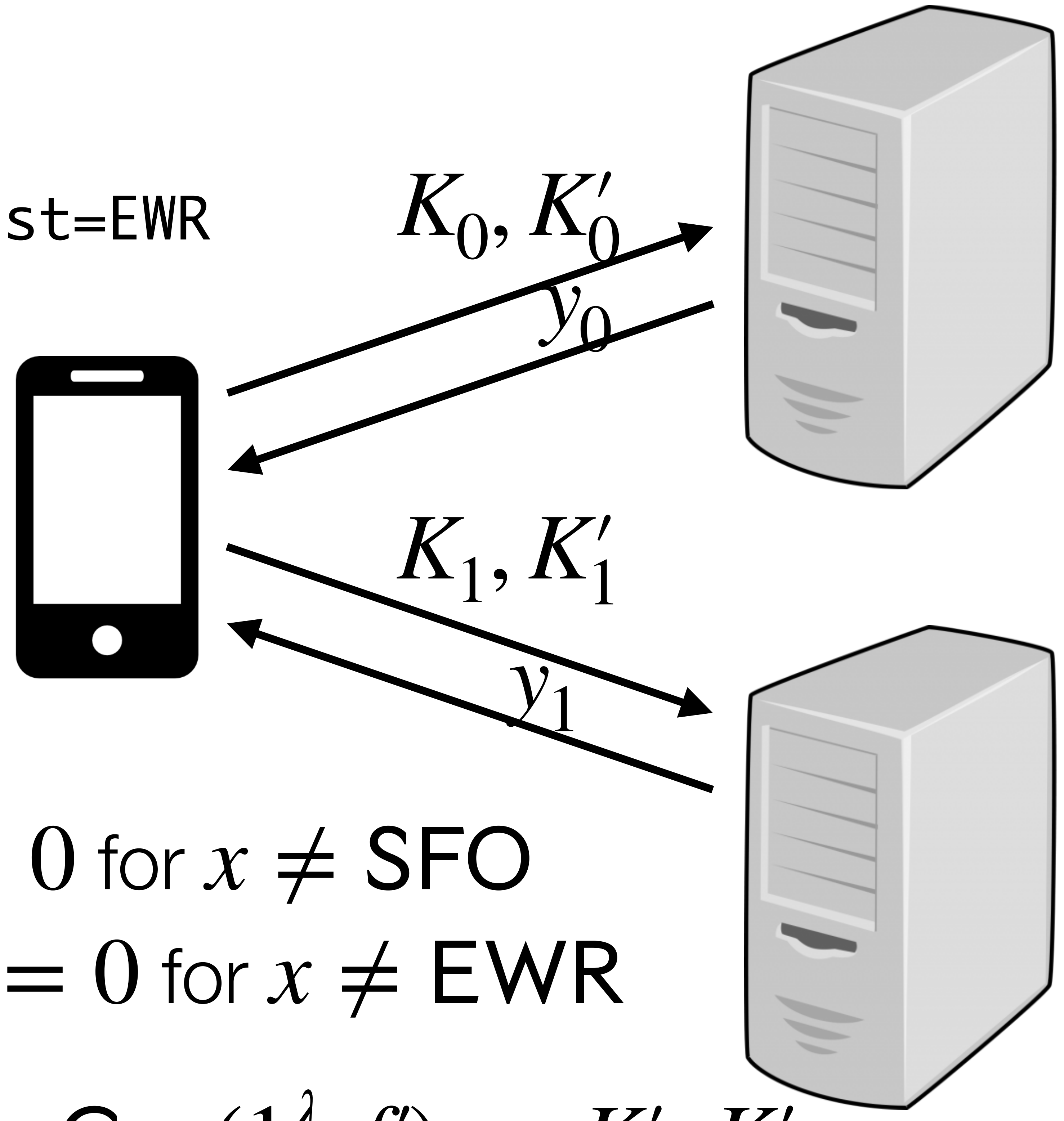
$f(50||\text{Blue}||\text{S}) = 1$, $f(x) = 0$ for $x \neq (50||\text{Blue}||\text{S})$

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

$y \leftarrow y_0 + y_1$

More complex conditions: Disjoint OR

```
SELECT COUNT(*)
FROM Trips
WHERE Dest=SFO OR Dest=EWR
```



Dest	
SFO	$\mathsf{Eval}(K_0, \mathsf{SFO}) + \mathsf{Eval}(K_0', \mathsf{SFO})$
JFK	$\mathsf{Eval}(K_0, \mathsf{JFK}) + \mathsf{Eval}(K_0', \mathsf{JFK})$
$\dots$	$\dots$
LAX	$\mathsf{Eval}(K_0, \mathsf{LAX}) + \mathsf{Eval}(K_0', \mathsf{LAX})$

$= y_0$

Dest	
SFO	$\mathsf{Eval}(K_1, \mathsf{SFO}) + \mathsf{Eval}(K_1', \mathsf{SFO})$
JFK	$\mathsf{Eval}(K_1, \mathsf{JFK}) + \mathsf{Eval}(K_1', \mathsf{JFK})$
$\dots$	$\dots$
LAX	$\mathsf{Eval}(K_1, \mathsf{LAX}) + \mathsf{Eval}(K_1', \mathsf{LAX})$

$= y_1$

$f(\mathsf{SFO}) = 1$, $f(x) = 0$ for $x \neq \mathsf{SFO}$

$f'(\mathsf{EWR}) = 1$, $f'(x) = 0$ for $x \neq \mathsf{EWR}$

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

$y \leftarrow y_0 + y_1$

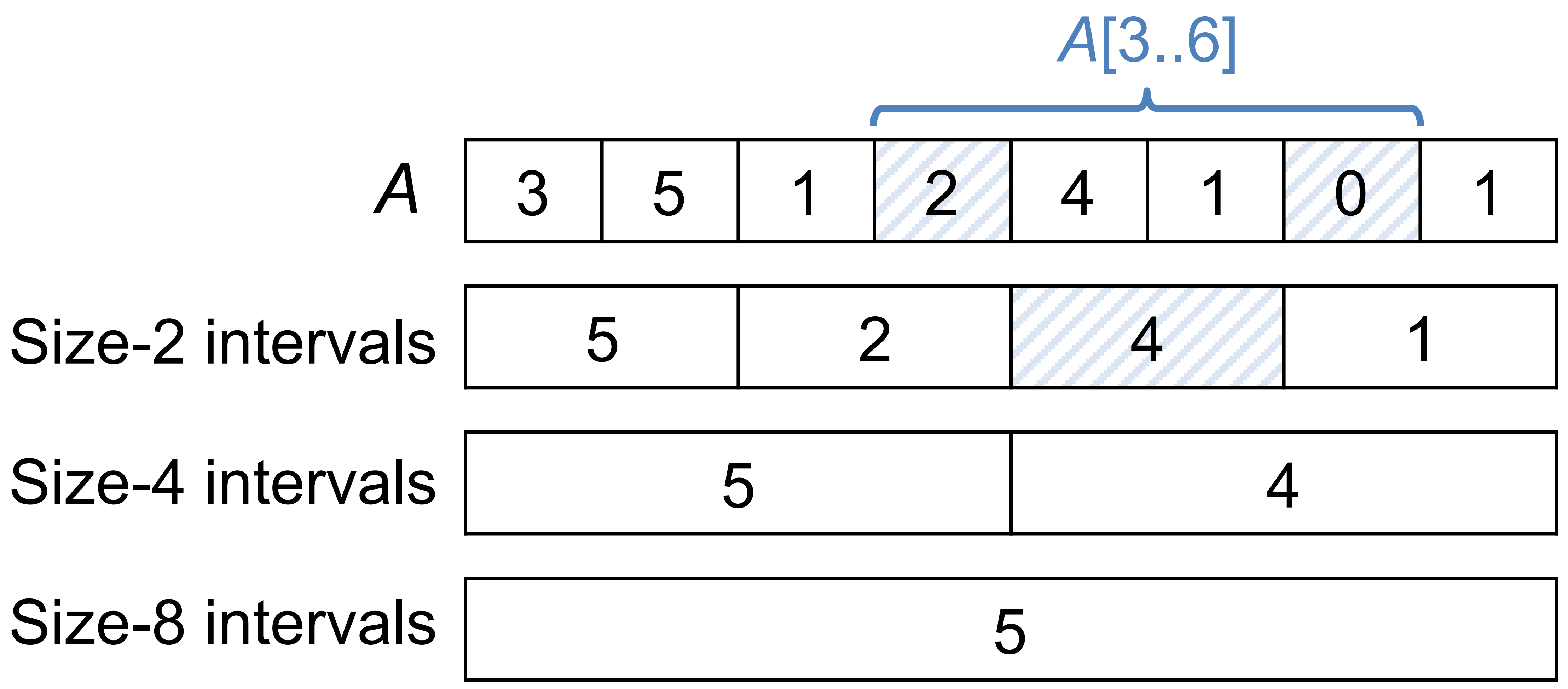
Max/min for interval conditions

```
SELECT MAX(Price) FROM Items WHERE 9 <= Rating <= 10
```

Setup: build an array A ordered by ratings

Round 1: Find indices i, j in A where all **rating** $\in [9,10]$ using keys for interval function for **rating** $\in [0,8]$, **rating** $\in [0,9]$

Round 2: Select maxes from power-of-2 sized intervals that cover exactly records i through j in A



See paper for max for single conditions

Max/min for disjoint OR

```
SELECT MIN(Price) FROM Flights WHERE Dest=SFO OR Dest=EWR
```

Setup: build an array A ordered by ratings

Binary search over A in decreasing power-of-2 sizes to find the smallest index where Dest=SFO or Dest=EWR

- `SELECT COUNT(*) FROM A WHERE (Dest=SFO OR Dest=EWR) AND index ∈ [start, end]`
- To search over interval, take advantage of fact that intervals are power-of-2 aligned (match on first x bits)

Use a DPF to retrieve the price for the entry with the smallest index satisfying Dest=SFO OR Dest=EWR

[For small number of conditions, can also run one query for each clause and combine at the client]

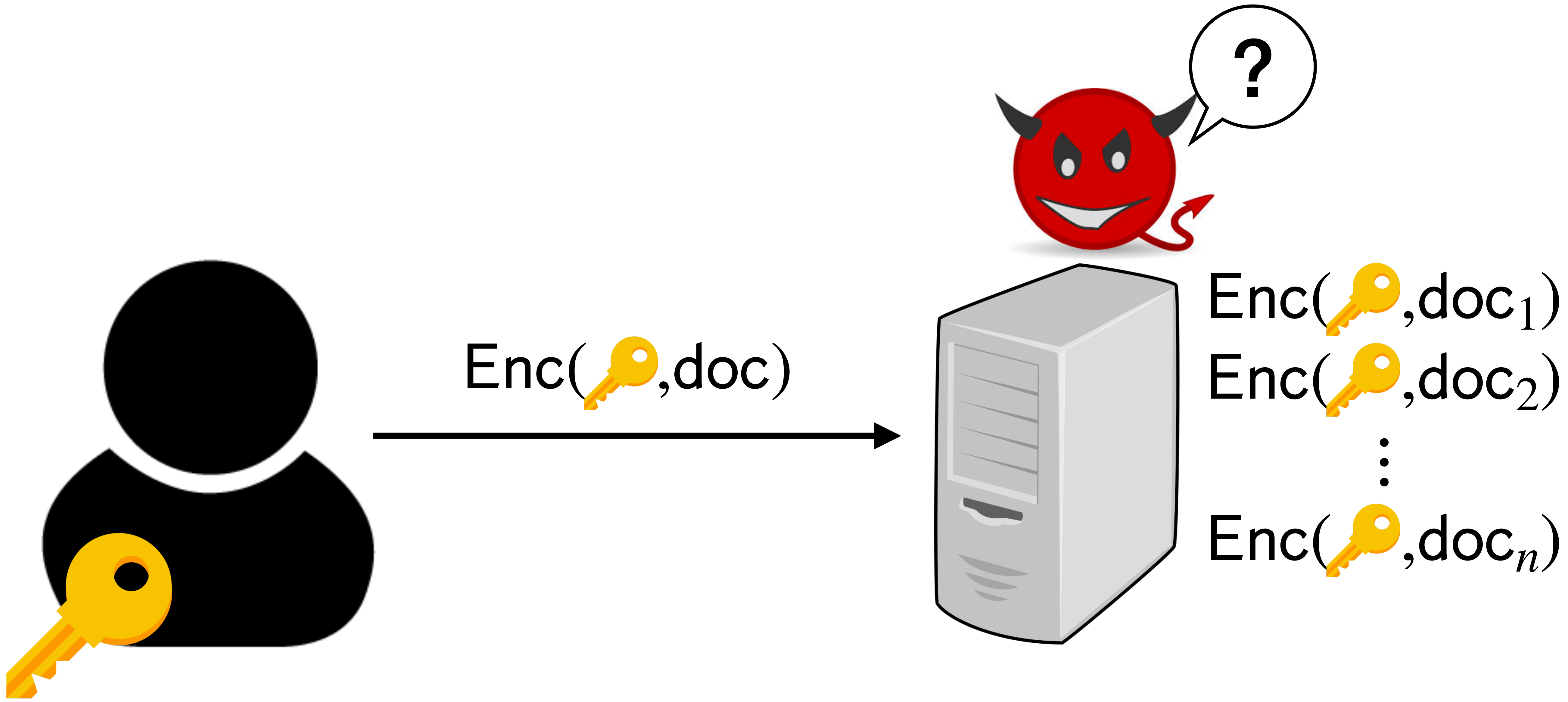
TOPK naturally extends MAX/MIN

Outline

1. Function secret sharing
2. Splinter
3. **Searching on encrypted data**

End-to-end encrypted filesystems

Provide strong security guarantees if attacker compromises server.

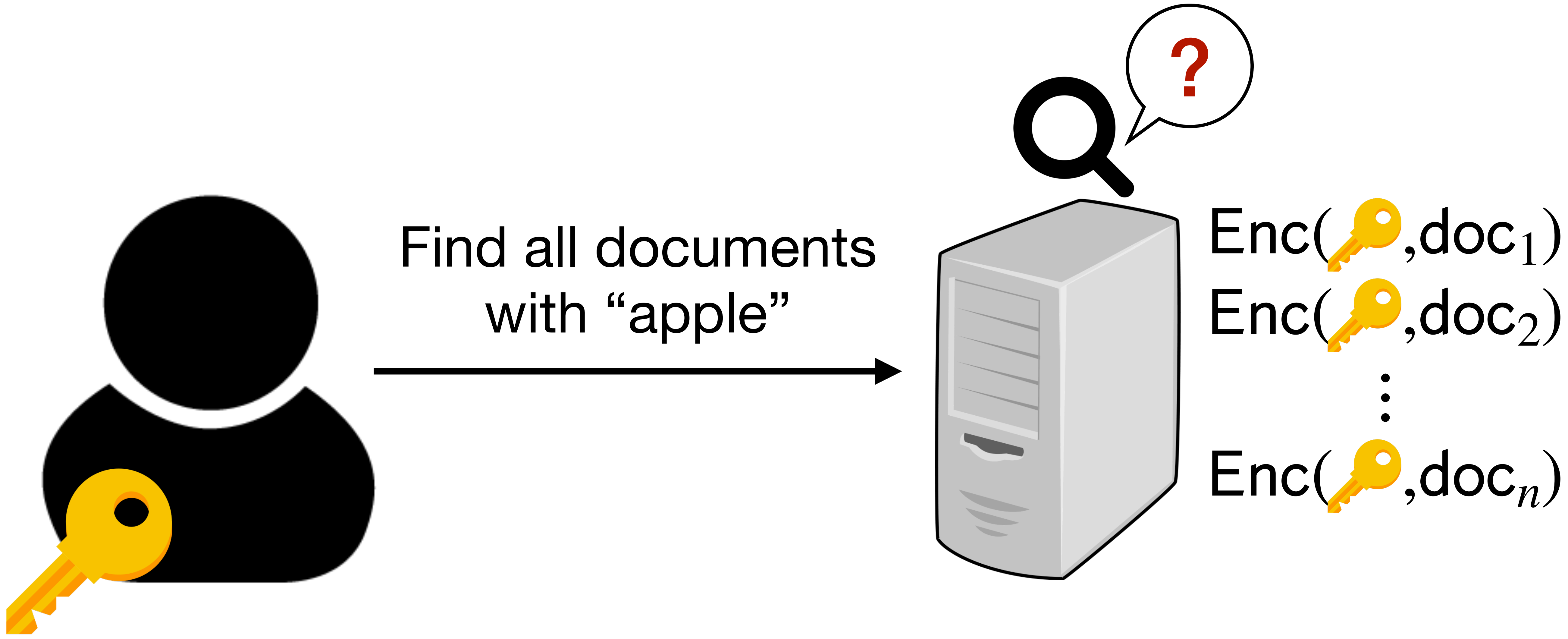


Users expect the ability to search

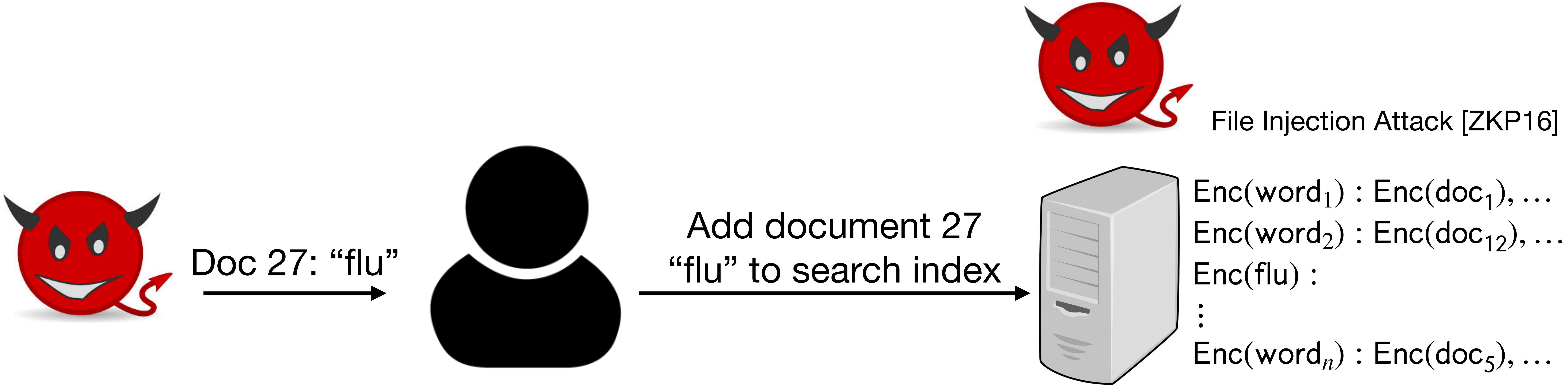
Doc 1
Doc 7
Doc 21
Doc 53

Search for end-to-end encrypted filesystems

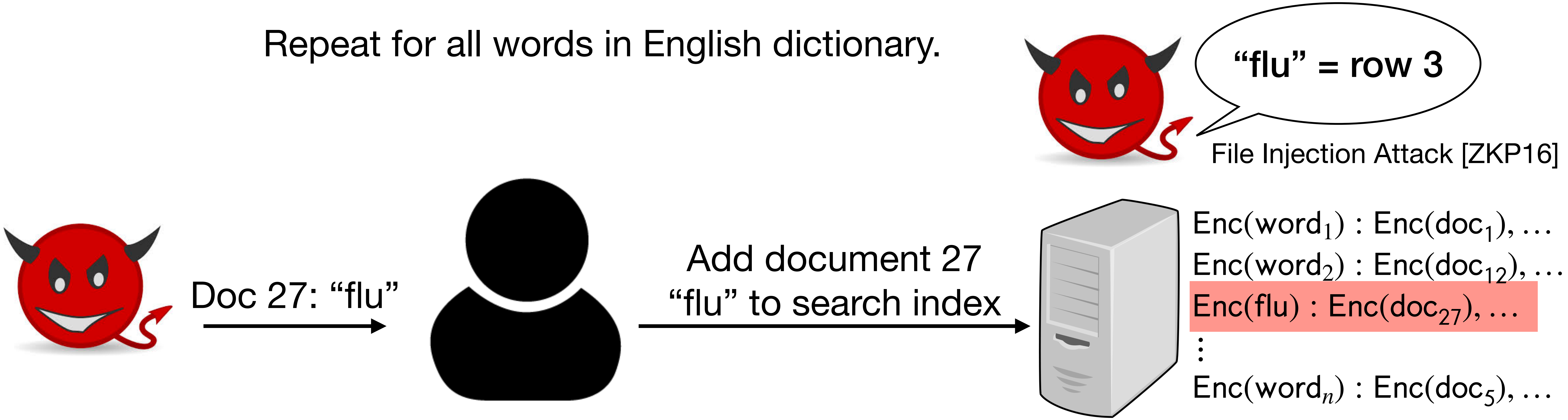
Challenge: server cannot decrypt data to search.



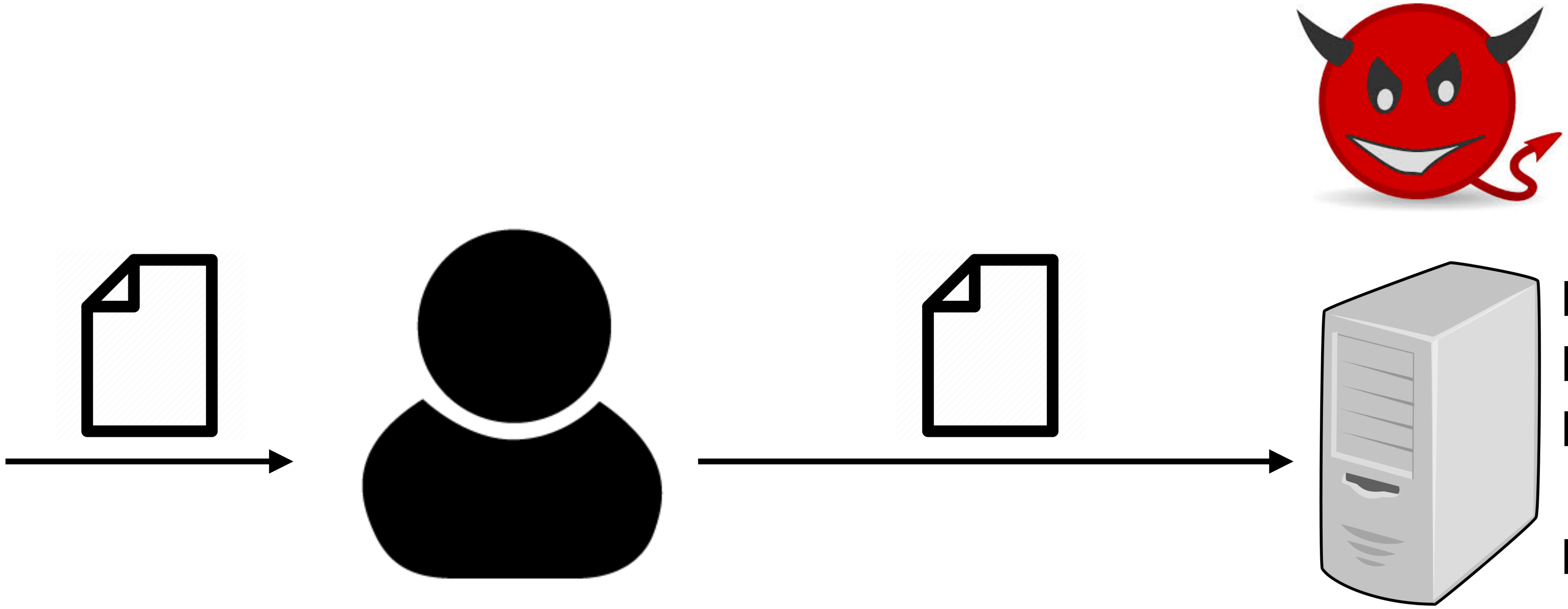
How the server accesses an encrypted index can leak information



How the server accesses an encrypted index can leak information



How the server accesses an encrypted index can leak information



File Injection Attack [ZKP16]

$\mathsf{Enc}(\mathsf{word}_1) : \mathsf{Enc}(\mathsf{doc}_1), \ldots$

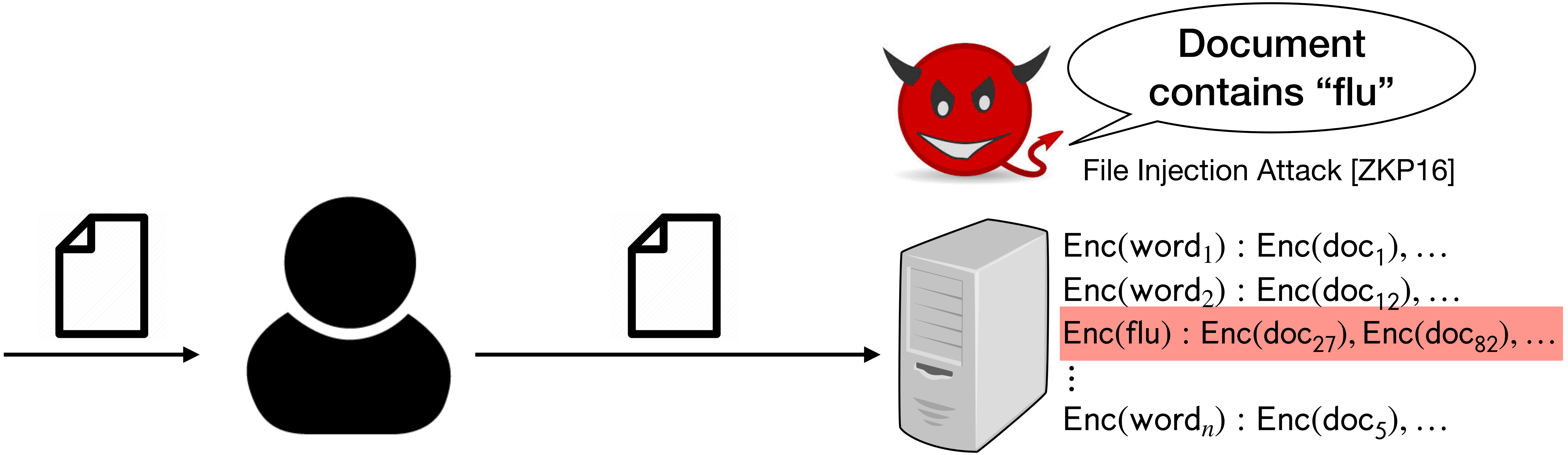
$\mathsf{Enc}(\mathsf{word}_2) : \mathsf{Enc}(\mathsf{doc}_{12}), \ldots$

$\mathsf{Enc}(\mathsf{flu}) : \mathsf{Enc}(\mathsf{doc}_{27}), \ldots$

$\vdots$

$\mathsf{Enc}(\mathsf{word}_n) : \mathsf{Enc}(\mathsf{doc}_5), \ldots$

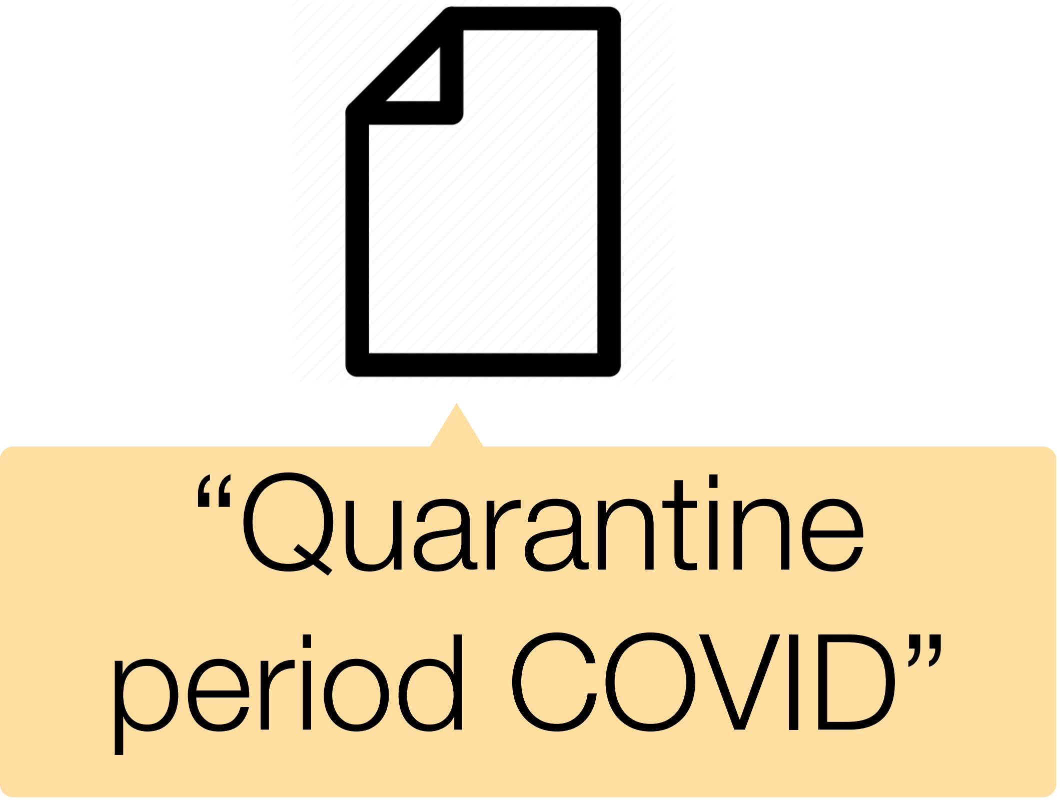
How the server accesses an encrypted index can leak information

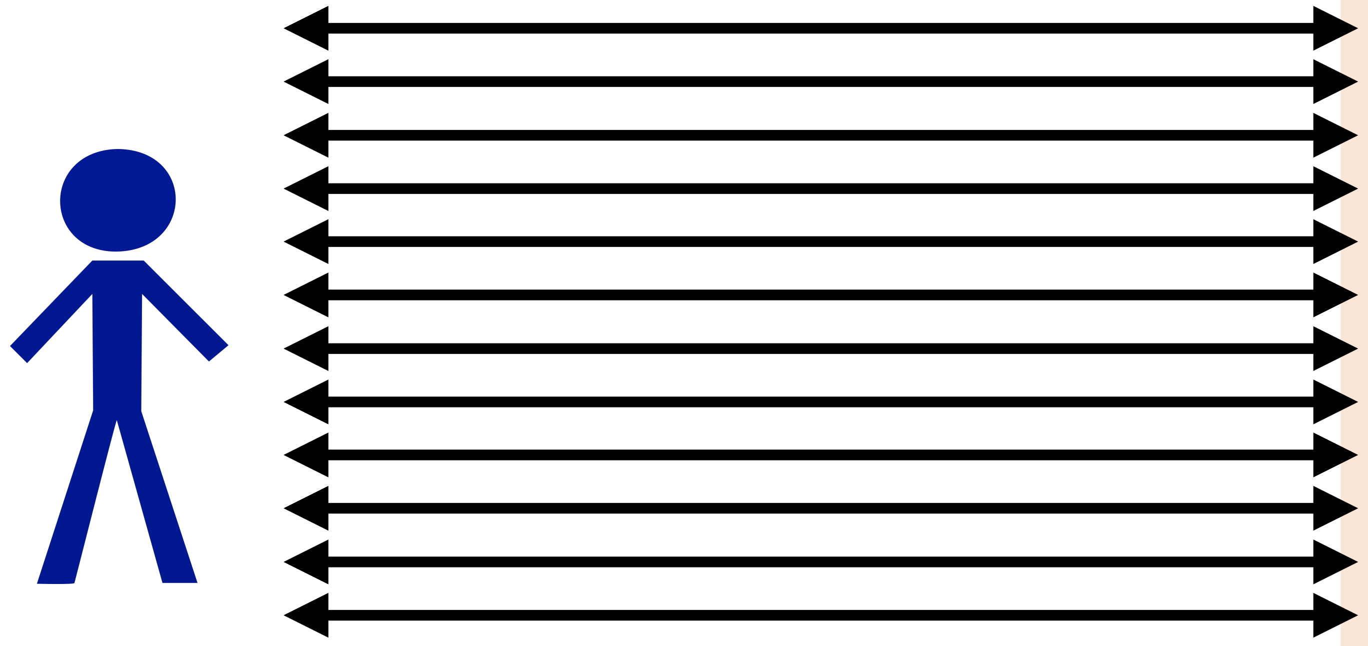


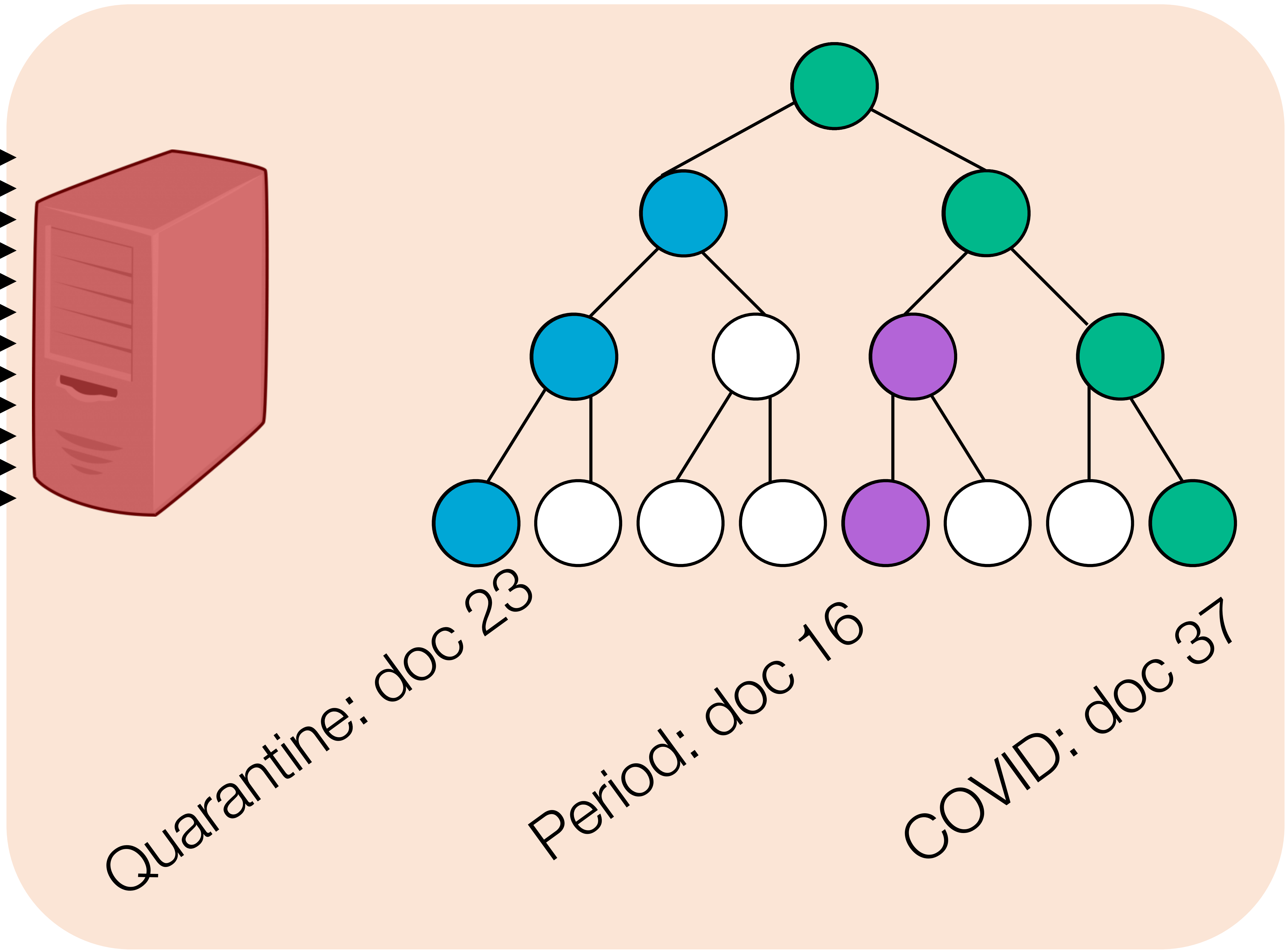
… and many more attacks

Oblivious RAM solutions are slow for encrypted search

Client can read/write server data without revealing data location [GO96, PathORAM]

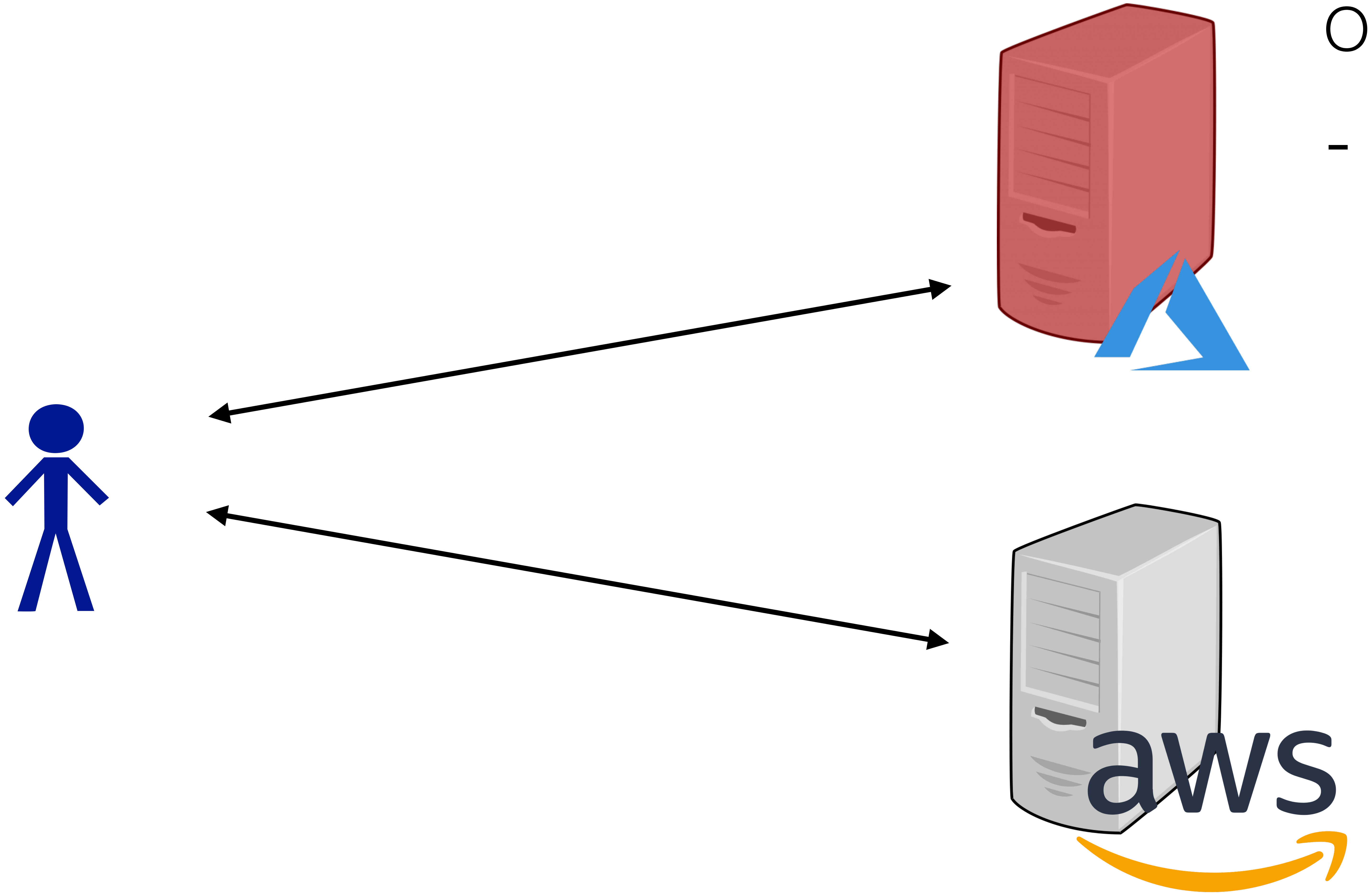






+ Logarithmic overhead
- High concrete costs

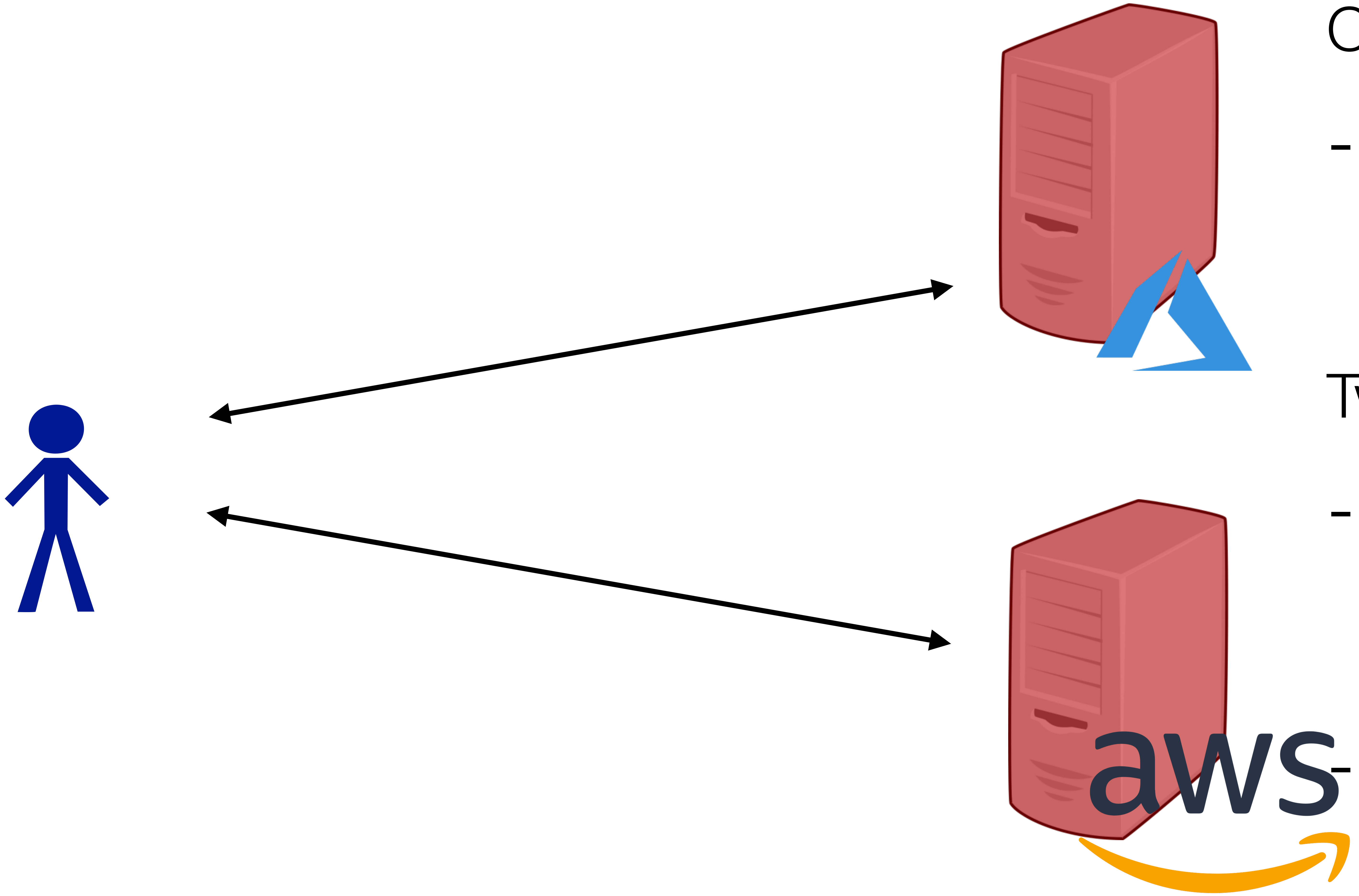
Security properties



One malicious server:

- Attacker can't learn anything about queries or document contents from memory accesses

Security properties



One malicious server:

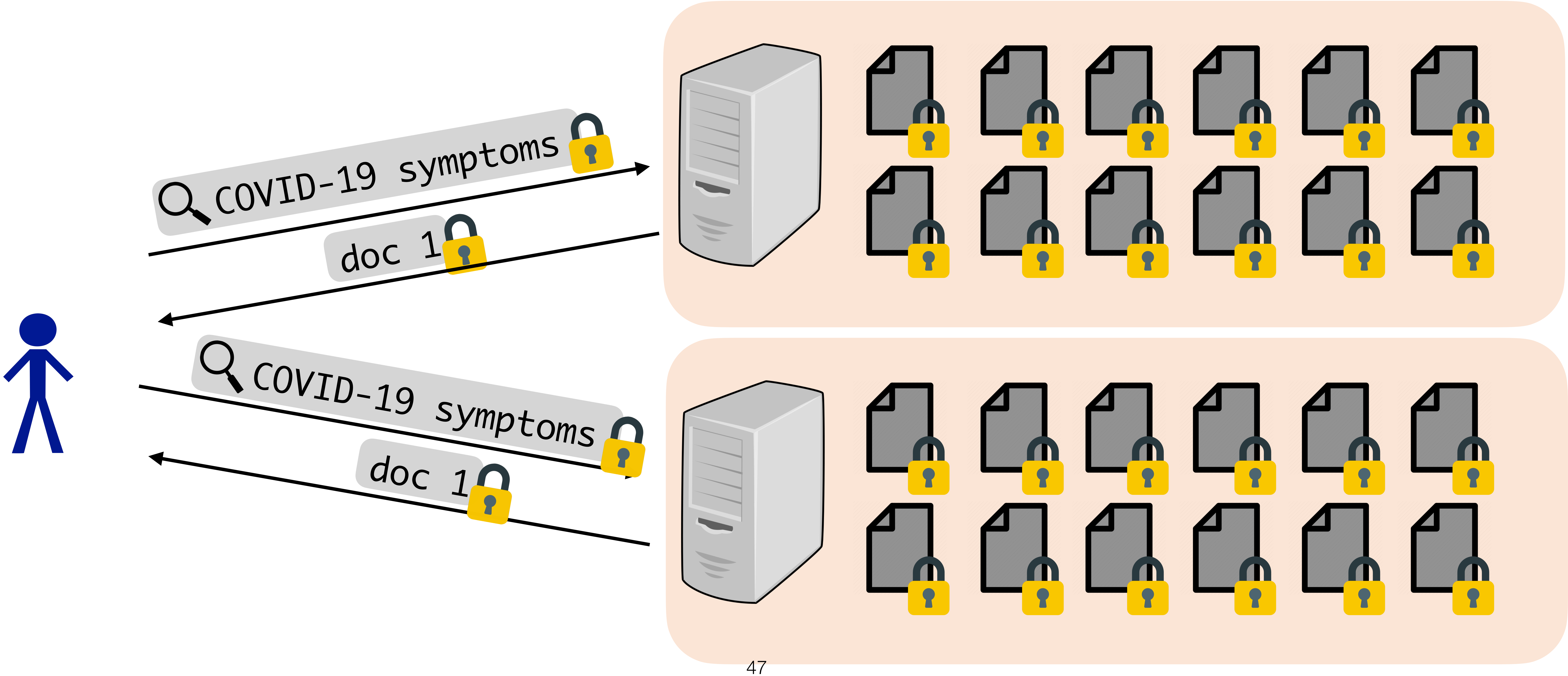
- Attacker can't learn anything about queries or document contents from memory accesses

Two malicious servers:

- Attacker can't immediately learn document contents (but can infer info from memory accesses)

DORY: Efficient search without search-access-pattern leakage

[Dauterman, Feng, Luo, Popa, Stoica]



High-level approach

Searchable encryption schemes

[Song, Wagner, Perrig], …

Queries require accessing
part of search index

Leak info about query and data

DORY

Queries require accessing
entire search index
$\Longrightarrow$ ok if concretely fast

Leak nothing about query or data

Building the search index

Bloom filter containing keywords in doc 1

doc 1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$\cdots$	$v_{1,m}$
doc 2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$\cdots$	$v_{2,m}$
doc 3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$\cdots$	$v_{3,m}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
doc n	$v_{n,1}$	$v_{n,2}$	$v_{n,3}$	$\cdots$	$v_{n,m}$

Building the search index

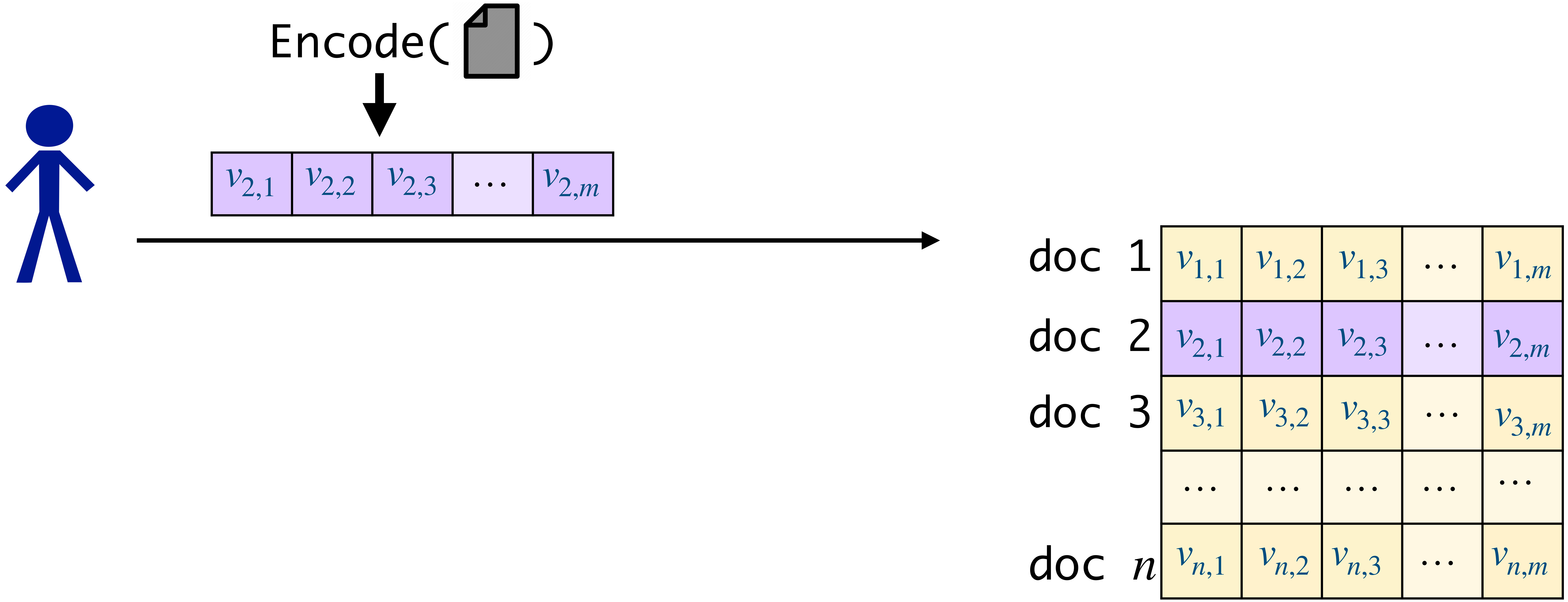
Bloom filter containing keywords in `doc 1`

`doc 1`	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$\cdots$	$v_{1,m}$
`doc 2`	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$\cdots$	$v_{2,m}$
`doc 3`	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$\cdots$	$v_{3,m}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
`doc` n	$v_{n,1}$	$v_{n,2}$	$v_{n,3}$	$\cdots$	$v_{n,m}$

Can check for a keyword with few point queries

High-level approach

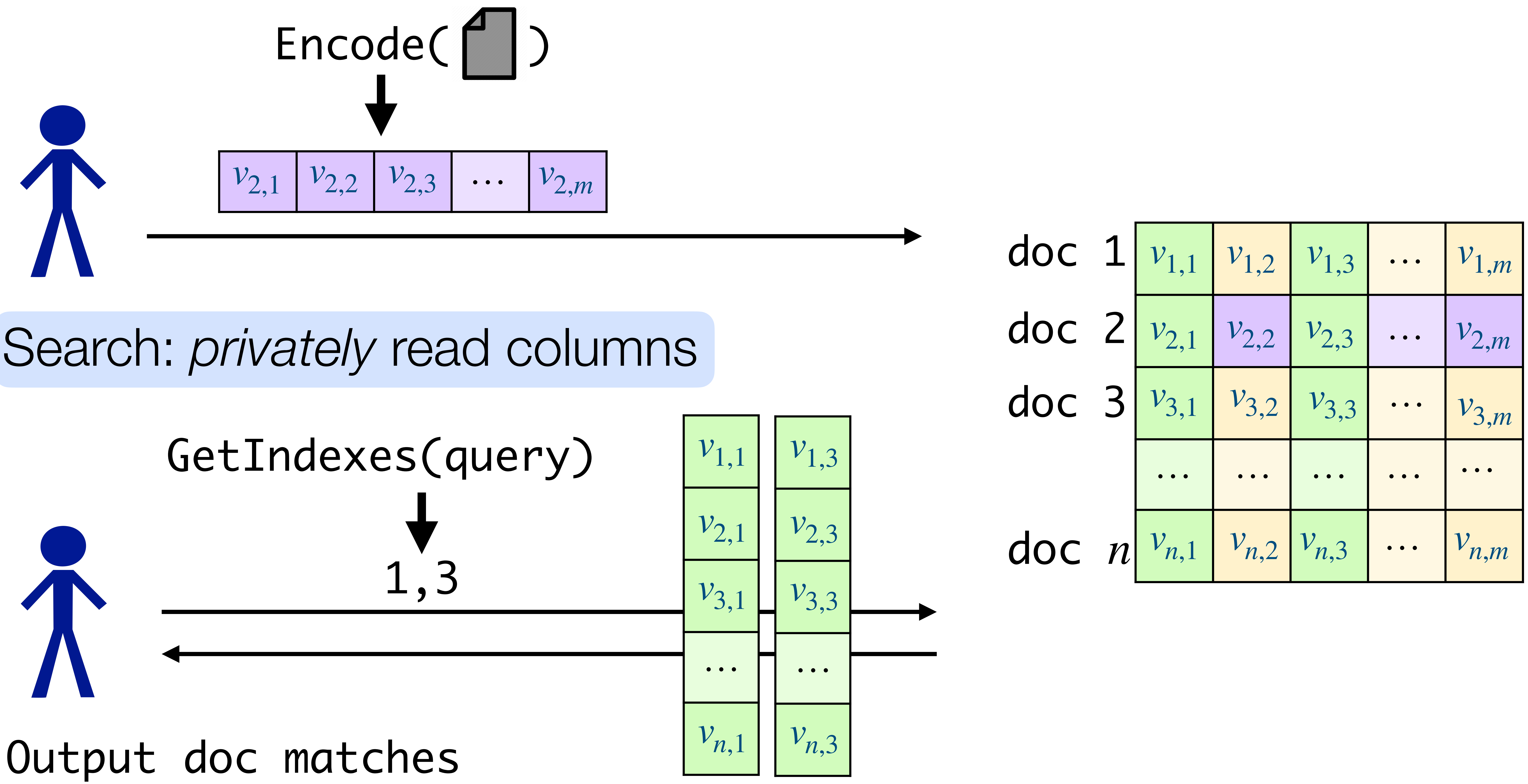
Update: write a row



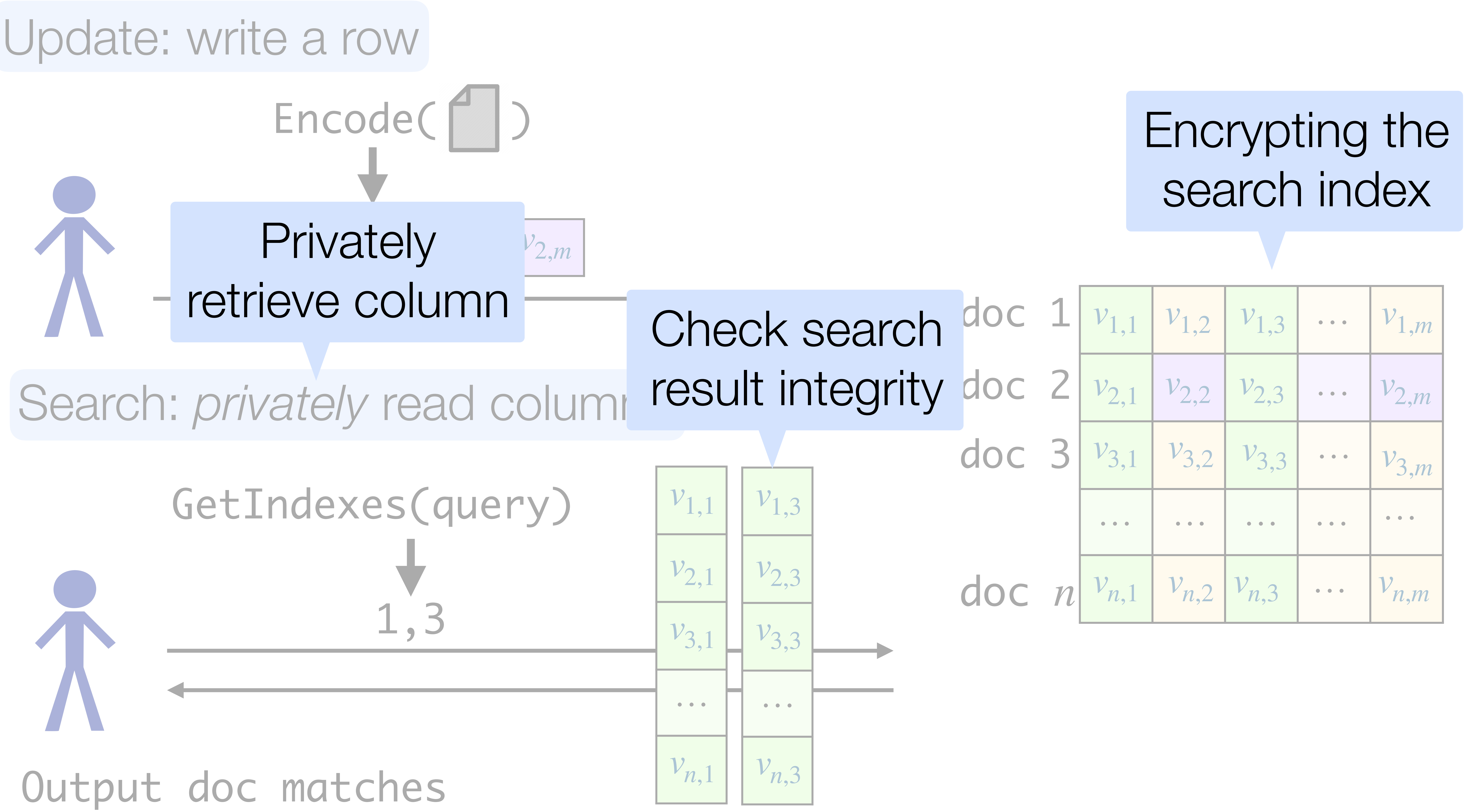
High-level approach

Update: write a row

Search: *privately* read columns



High-level approach



Privately reading a column to search

Privately reading a column to search

Privacy: If at least one server is honest, the attacker learns no information about the query

Privately reading a column to search

Efficiency: Requires linear scan, but concretely fast

Privately reading a column to search

Reduce communication: compress secret-shared one-hot vector with distributed point functions [GI14, BGI16]

Encrypting the search index

Challenge: Client uploads *rows* of bits, but retrieves *columns* of bits

- Don't want to increase communication by a factor of $\lambda \approx 128$.

Idea: Generate a unique mask using document version number

$$
\begin{array}{r}
\text{doc } 1 \\ \text{doc } 2 \\ \text{doc } 3 \\ \\ \text{doc } n
\end{array}
\begin{array}{|c|c|c|c|c|}
\hline
v_{1,1} & v_{1,2} & v_{1,3} & \cdots & v_{1,m} \\ \hline
v_{2,1} & v_{2,2} & v_{2,3} & \cdots & v_{2,m} \\ \hline
v_{3,1} & v_{3,2} & v_{3,3} & \cdots & v_{3,m} \\ \hline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hline
v_{n,1} & v_{n,2} & v_{n,3} & \cdots & v_{n,m} \\ \hline
\end{array}
\oplus
\begin{array}{l}
\mathrm{PRF}_k(0 \,||\, \mathrm{version}_0) \\
\mathrm{PRF}_k(1 \,||\, \mathrm{version}_1) \\
\mathrm{PRF}_k(2 \,||\, \mathrm{version}_2) \\
\quad \vdots \\
\mathrm{PRF}_k(n \,||\, \mathrm{version}_n)
\end{array}
\longrightarrow
\begin{array}{r}
\text{doc } 1 \\ \text{doc } 2 \\ \text{doc } 3 \\ \\ \text{doc } n
\end{array}
\begin{array}{|c|c|c|c|c|}
\hline
v'_{1,1} & v'_{1,2} & v'_{1,3} & \cdots & v'_{1,m} \\ \hline
v'_{2,1} & v'_{2,2} & v'_{2,3} & \cdots & v'_{2,m} \\ \hline
v'_{3,1} & v'_{3,2} & v'_{3,3} & \cdots & v'_{3,m} \\ \hline
\cdots & \cdots & \cdots & \cdots & \cdots \\ \hline
v'_{n,1} & v'_{n,2} & v'_{n,3} & \cdots & v'_{n,m} \\ \hline
\end{array}
$$

Search result integrity

MAC tags allow client to verify result corresponds to valid update

Aggregate MACs compress all MAC tags in a column into a single tag [KL08]

- XOR individual MAC tags into single aggregate tag

Aggregate MACs	t_1	t_2	t_3	$\cdots$	t_m
doc 1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$\cdots$	$v_{1,m}$
doc 2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$\cdots$	$v_{2,m}$
doc 3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$\cdots$	$v_{3,m}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
doc n	$v_{n,1}$	$v_{n,2}$	$v_{n,3}$	$\cdots$	$v_{n,m}$

Logistics

Signups for meetings for feedback on project proposals

- Required for all groups

Course feedback form on Ed

References

Beimel, Amos, Yuval Ishai, and Tal Malkin. "Reducing the servers computation in private information retrieval: PIR with preprocessing." In *Annual International Cryptology Conference*, pp. 55-73. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

Boneh, Dan, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Lightweight techniques for private heavy hitters." In *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 762-776. IEEE, 2021.

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.

Boyle, Elette, Niv Gilboa, and Yuval Ishai. "Secure computation with preprocessing via function secret sharing." In *Theory of Cryptography Conference*, pp. 341-371. Cham: Springer International Publishing, 2019.

Chor, Benny, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private information retrieval." *Journal of the ACM (JACM)* 45, no. 6 (1998): 965-981.

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.

Goldreich, Oded, and Rafail Ostrovsky. "Software protection and simulation on oblivious RAMs." *Journal of the ACM (JACM)* 43, no. 3 (1996): 431-473.

Katz, Jonathan, and Andrew Y. Lindell. "Aggregate message authentication codes." In *Cryptographers' Track at the RSA Conference*, pp. 155-169. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

Stefanov, Emil, Marten van Dijk, Elaine Shi, T-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. "Path ORAM: an extremely simple oblivious RAM protocol." *Journal of the ACM (JACM)* 65, no. 4 (2018): 1-26.

Song, Dawn Xiaoding, David Wagner, and Adrian Perrig. "Practical techniques for searches on encrypted data." In *Proceeding 2000 IEEE symposium on security and privacy. S&P 2000*, pp. 44-55. IEEE, 2000.

Zhang, Yupeng, Jonathan Katz, and Charalampos Papamanthou. "All your queries are belong to us: the power of {File-Injection} attacks on searchable encryption." In *25th USENIX Security Symposium (USENIX Security 16)*, pp. 707-720. 2016.