Single-Server Private Information Retrieval

Ling Ren

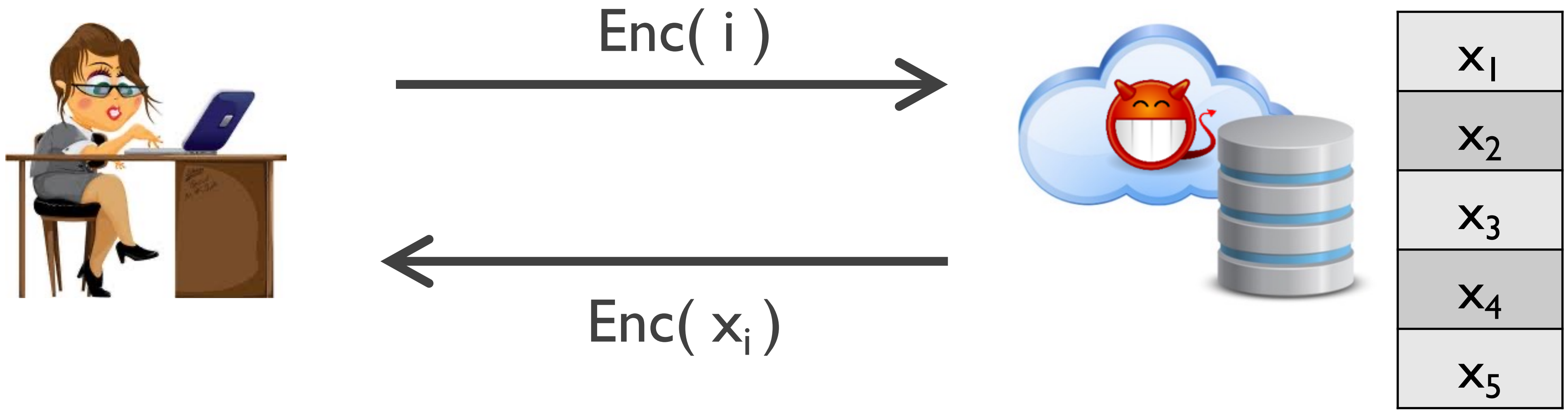
April 23, 2024

Private Information Retrieval (PIR) [CGKS'95]

- Let a client fetch a record privately from a database on server(s)

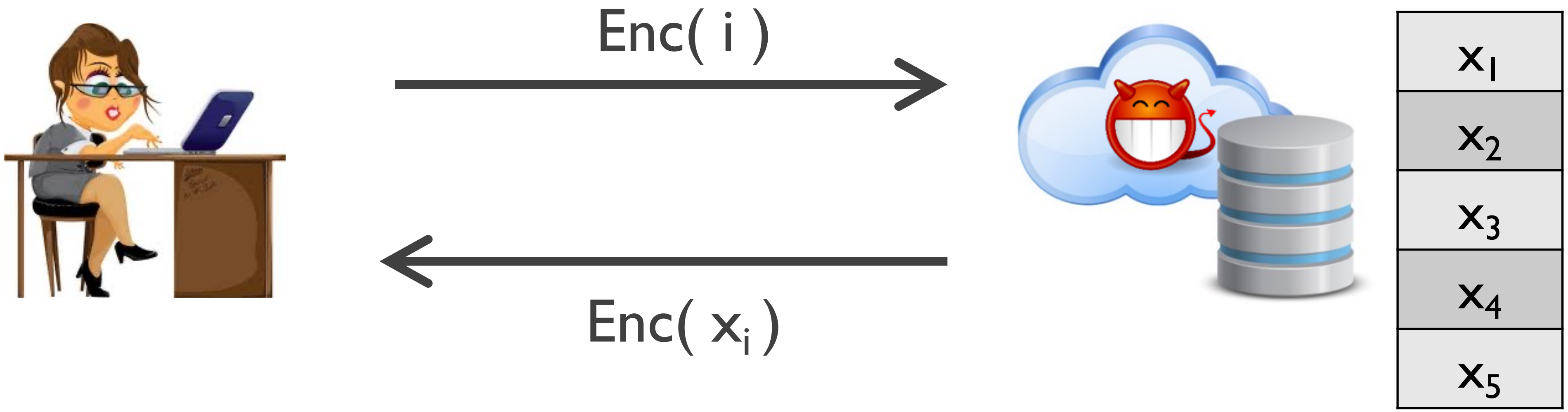
without revealing (to server)

any information about *which* record



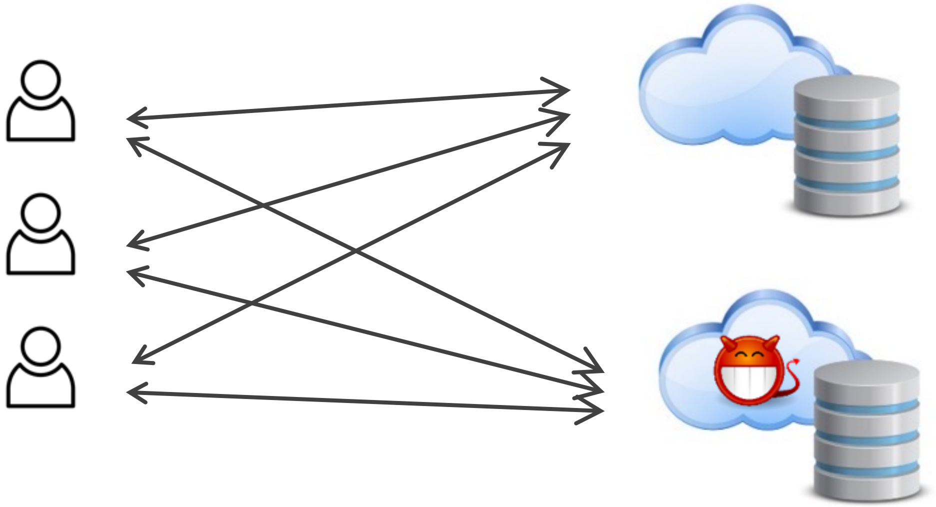
Private Information Retrieval (PIR) [CGKS'95]

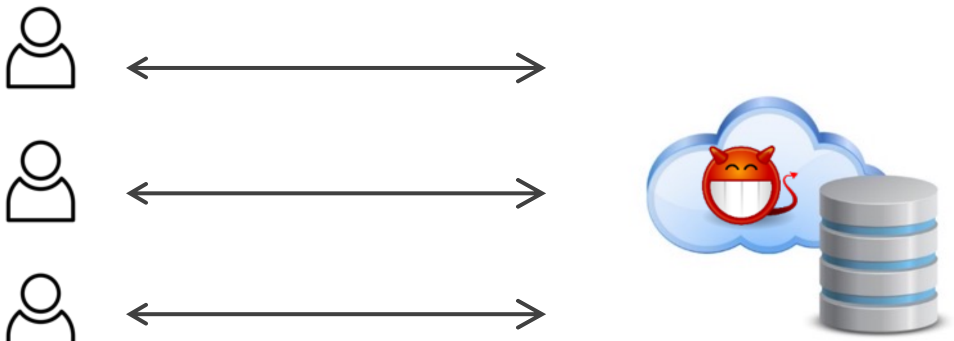
- Let a client fetch a record <u>privately</u> from a database on server(s)
- Applications
  - Anonymous messaging
  - Private media streaming
  - Private look-ups of domain name, public key, passwords, …



Private Information Retrieval (PIR) [CGKS'95]

- Multi-server PIR vs. **Single-server PIR**





PIR Efficiency Metrics

Request size

Server computation

Enc(i)

x_1
x_2
x_3
x_4
x_5

Enc(x_i)

Client storage

Response size

Server (extra) storage

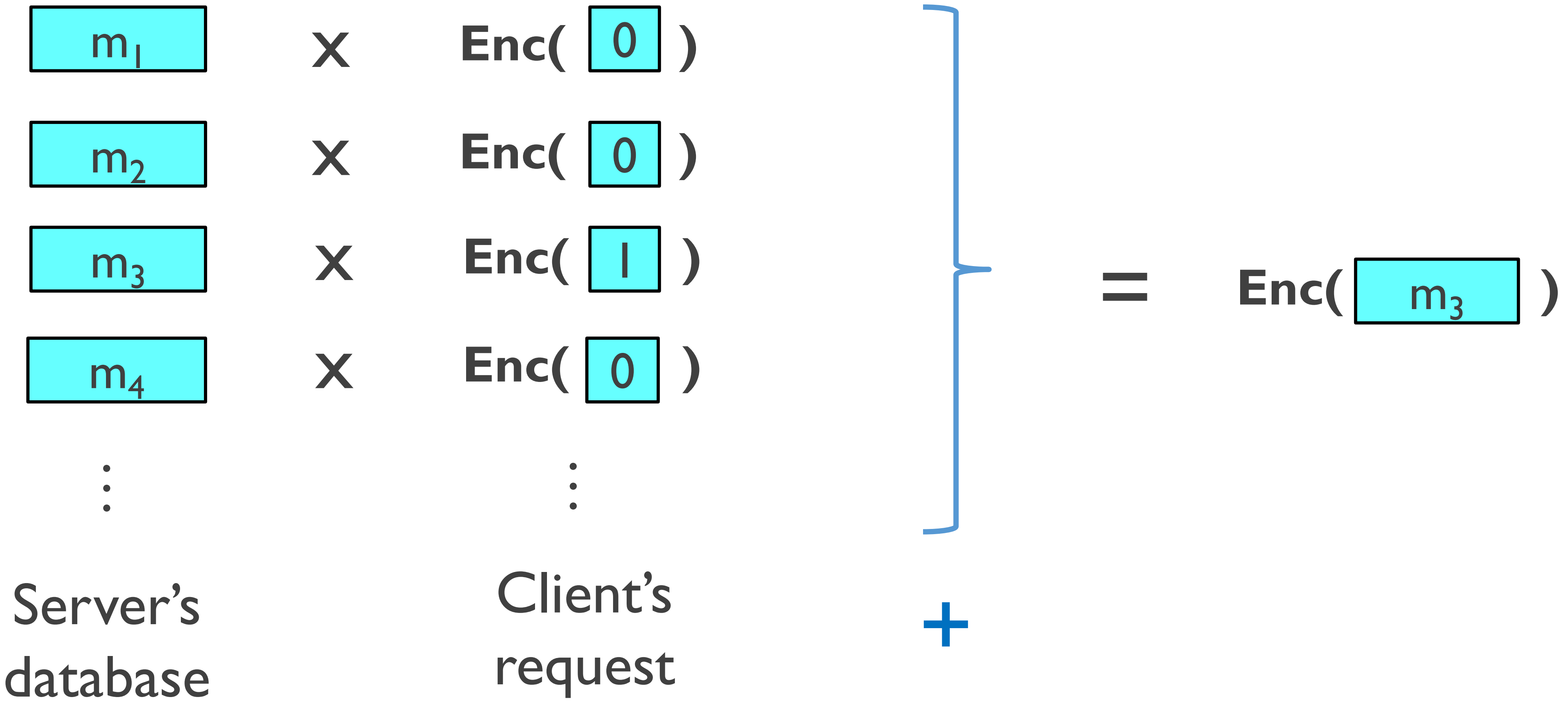
Outline

- Single-server PIR using homomorphic encryption
- Limits of single-server PIR in the standard model
- Batch PIR
- Amortized sublinear stateful PIR

Background: Additively Homomorphic Encryption

- Enc(x) + Enc(y) = Enc(x+y)
- m * Enc(x) = Enc(x) + Enc(x) + … + Enc(x) = Enc(mx)

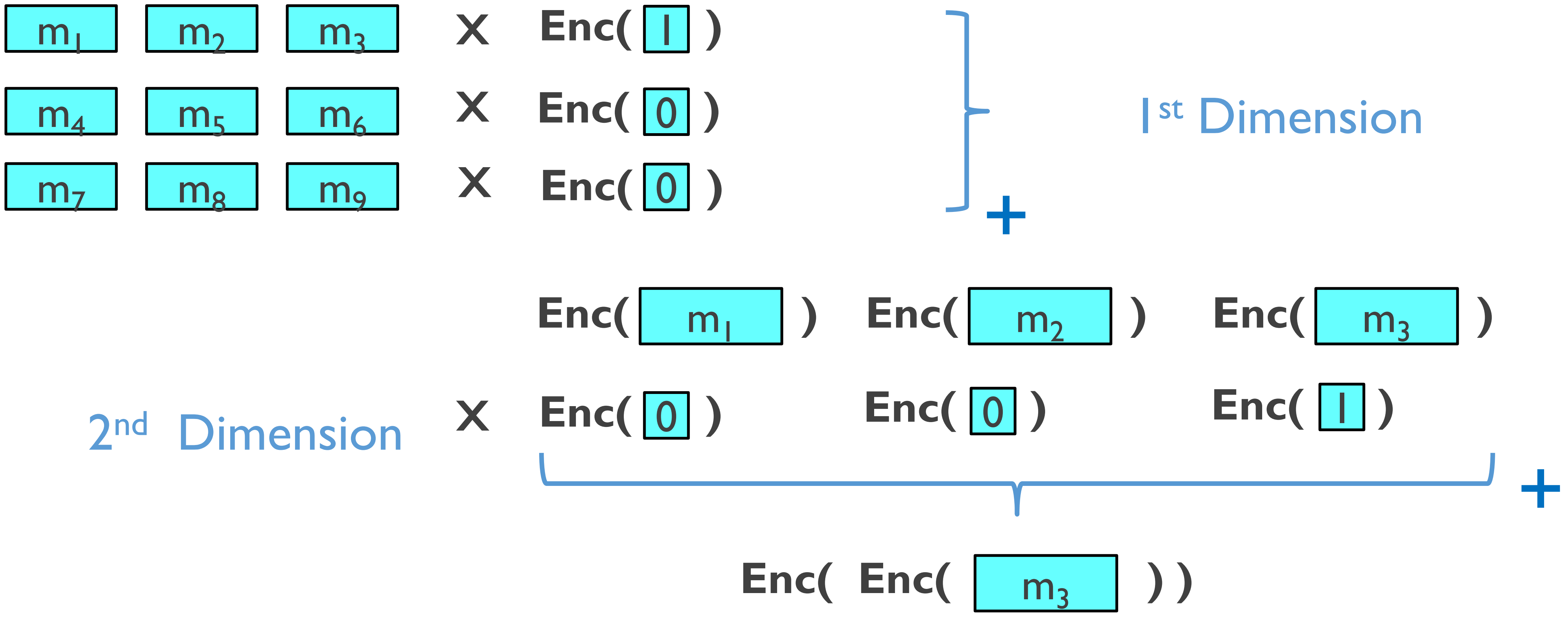
A Strawman PIR using AHE



Client's request too large (linear in database size)

Hierarchical PIR

- Organizing the database in 2D reduces request size to $2\sqrt[2]{n}$



Hierarchical PIR

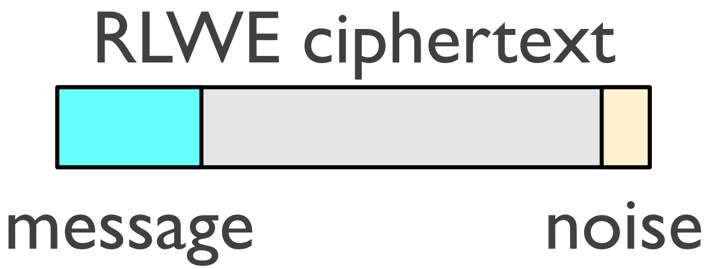
- Organizing the database in 2D reduces request size to $2\sqrt[2]{n}$
- d-dimensional hyper cube reduces request size to d$\sqrt[d]{n}$
- d = log *n* → request size = 2 log *n* (can be improved to log *n*)

- Remaining problem: extremely expensive computation
  - Need "additive" ciphertext blowup, Damgard-Jurik is only candidate

Background: Somewhat Homomorphic Encryption

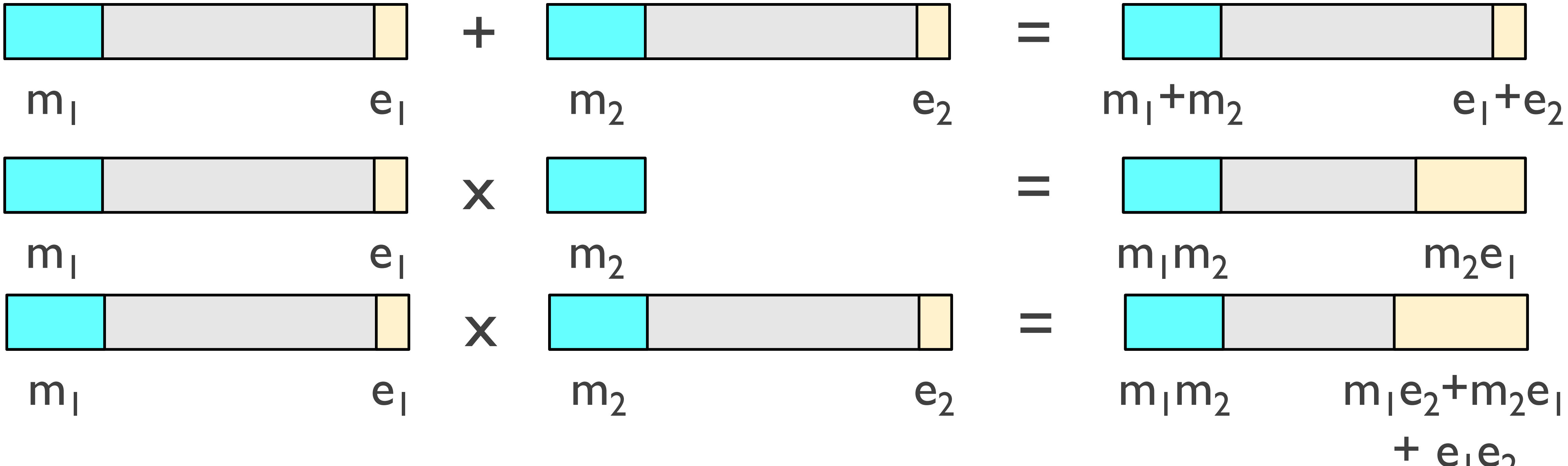
- SHE: supports a limited number of homomorphic addition & multiplication operations on ciphertexts

- Based on Ring Learning with Errors (RLWE) assumption

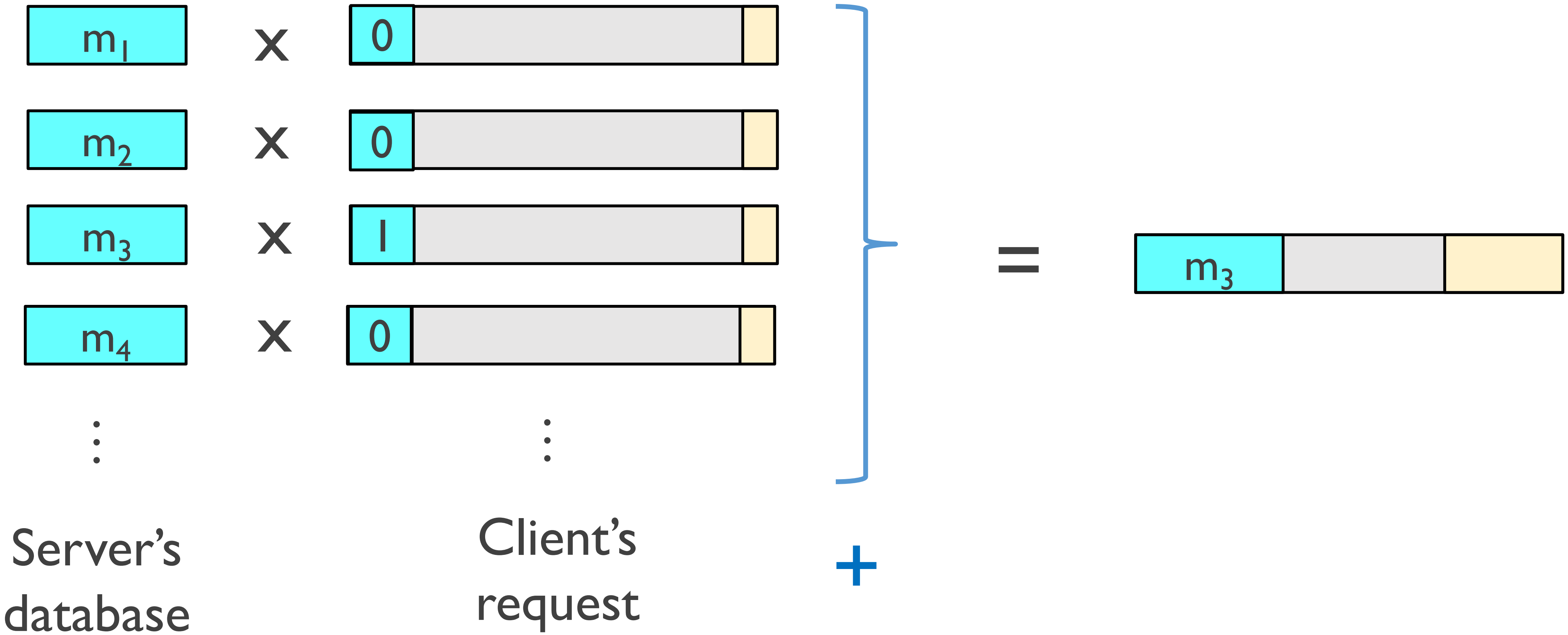


Background on SHE

- Homomorphic operations increase noise
- Multiplication adds a lot more noise than addition



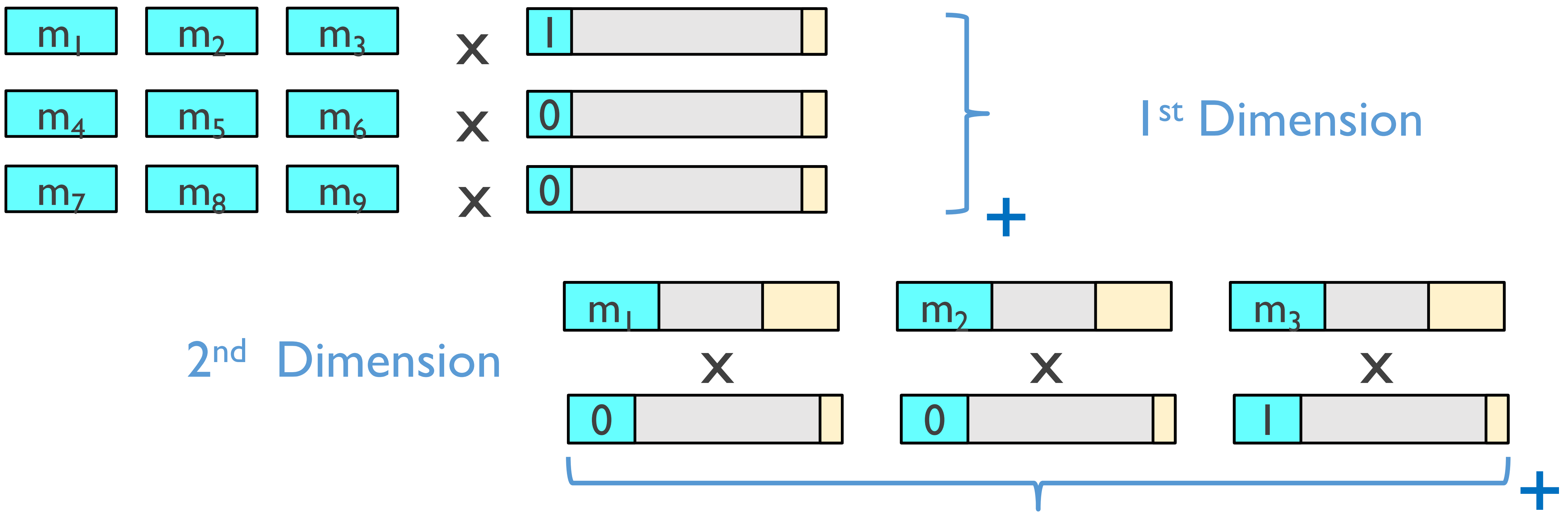
A Strawman PIR using SHE



Client's request too large (linear in database size)

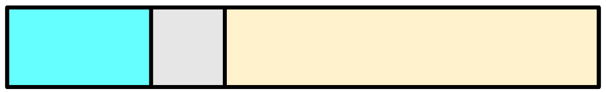
Hierarchical PIR using SHE

- d-dimensional hyper cube reduces request size to $d\sqrt[d]{n}$



Hierarchical PIR using SHE

- d-dimensional hyper cube reduces request size to d$\sqrt[d]{n}$
- Homomorphic multiplication blows up noise quickly
  - d = 2 or 3 in practice → $O(\sqrt{n})$ request size
  - Higher response and computation costs



- Solved in a series of recent works (beyond this lecture)
  - For a database of one million entries each of 12 KB, Onion PIR v2 achieves request = 36 KB, response = 36 KB (3x), computation = 24s

Summary of single-server PIR

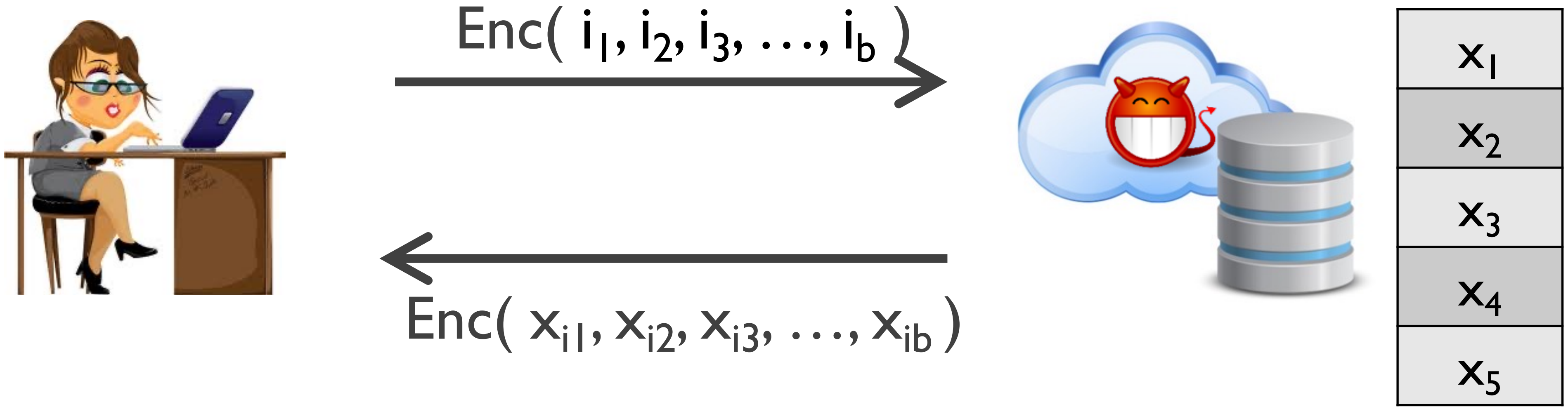
- Reasonable request size and response blowup
- Computation still heavy; only efficient for moderately large entries

- Both issues are somewhat inherent!
- Computation must involve every entry for security
- RLWE ciphertexts are big (e.g., ~ 36 KB)

- Can we do better?
- Amortization! Assume client wants to fetch multiple entries.

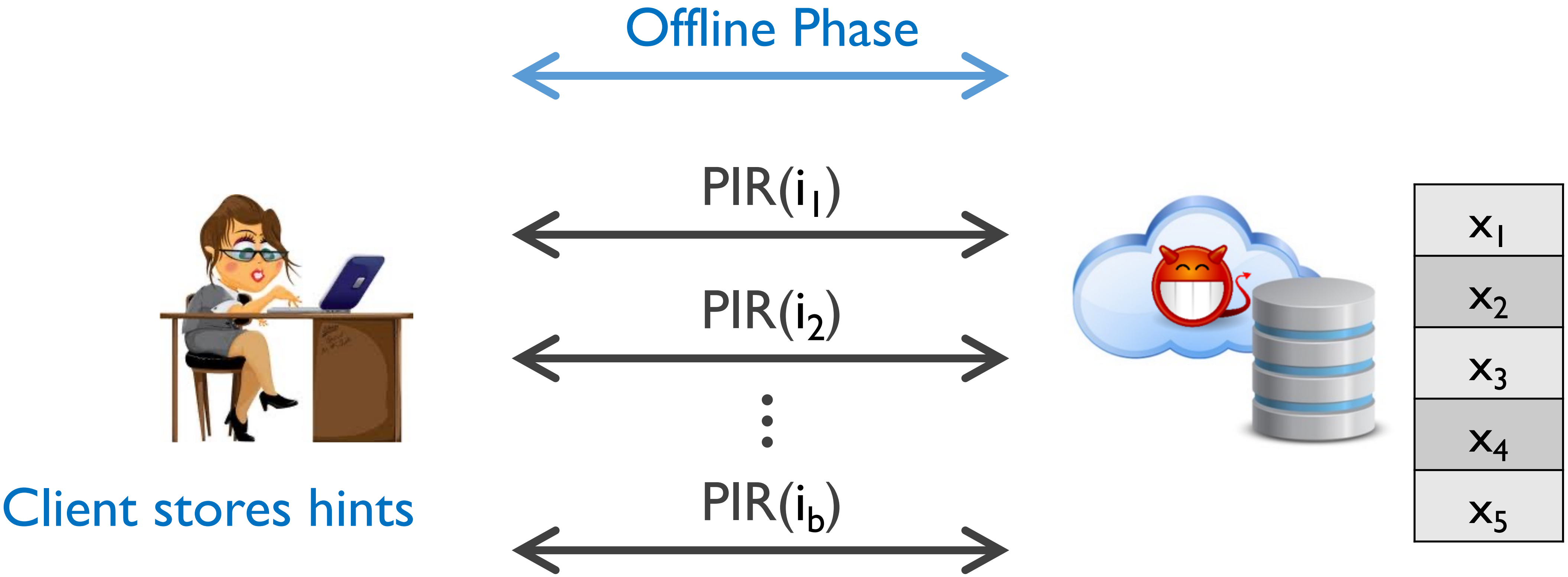
Batch PIR [IKOS'04, ACLS'18]

- Client wants to fetch multiple entries in one go



Stateful PIR [PPY'18, CK'20]

- Client wants to fetch multiple entries, but one at a time

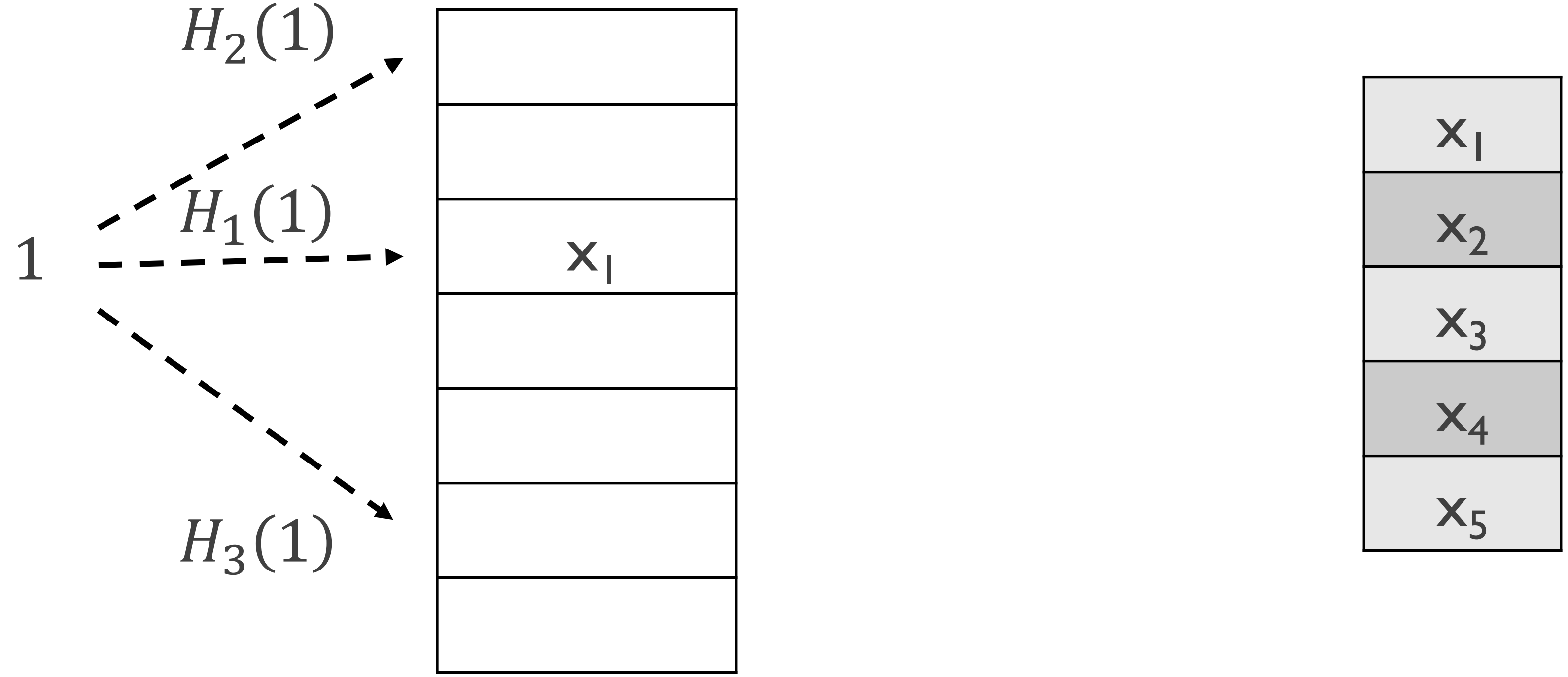


Outline

- Single-server PIR using homomorphic encryption
- Limits of single-server PIR in the standard model
- **Batch PIR**
- Amortized sublinear stateful PIR

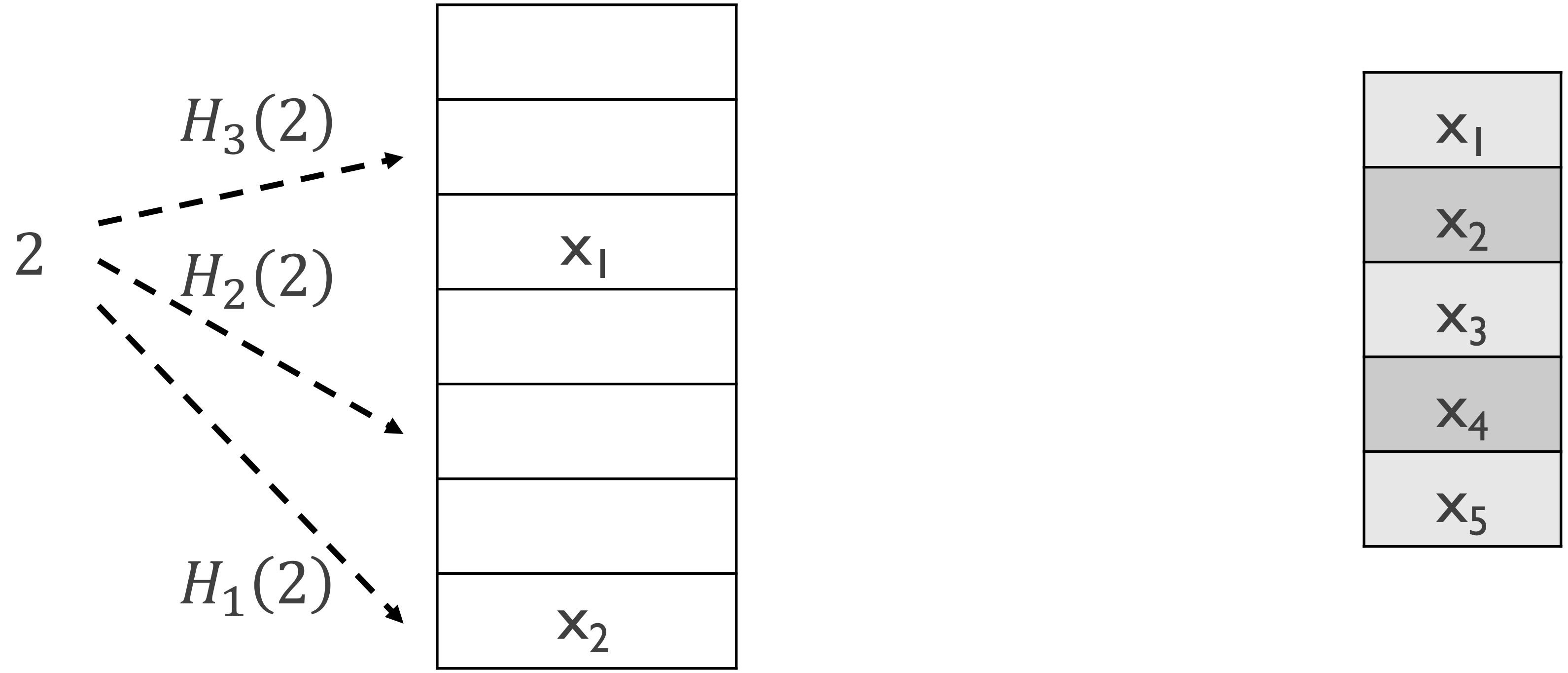
Background: Cuckoo Hashing

- A technique to build a collision-free hash table
- Each entry has multiple (e.g., 3) candidate locations



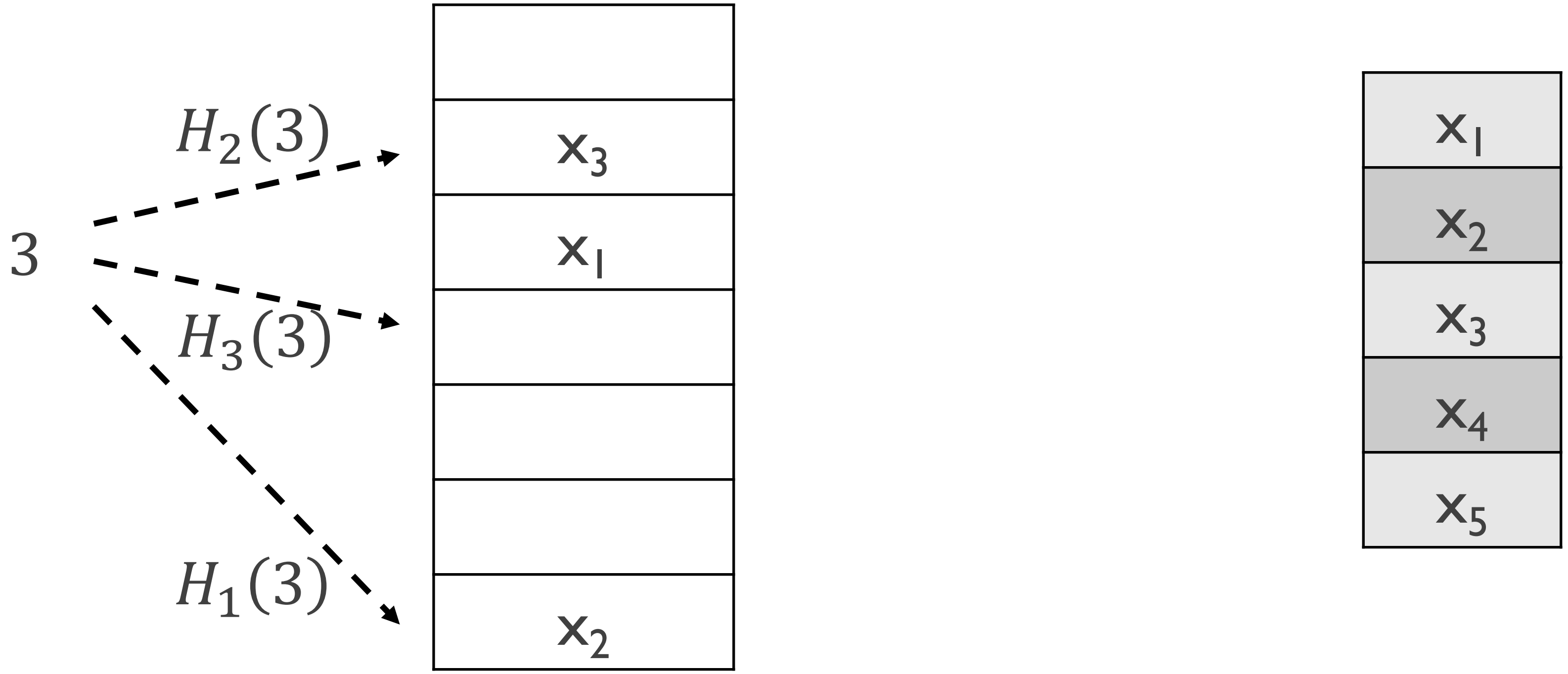
Background: Cuckoo Hashing

- A technique to build a collision-free hash table
- Each entry has multiple (e.g., 3) candidate locations



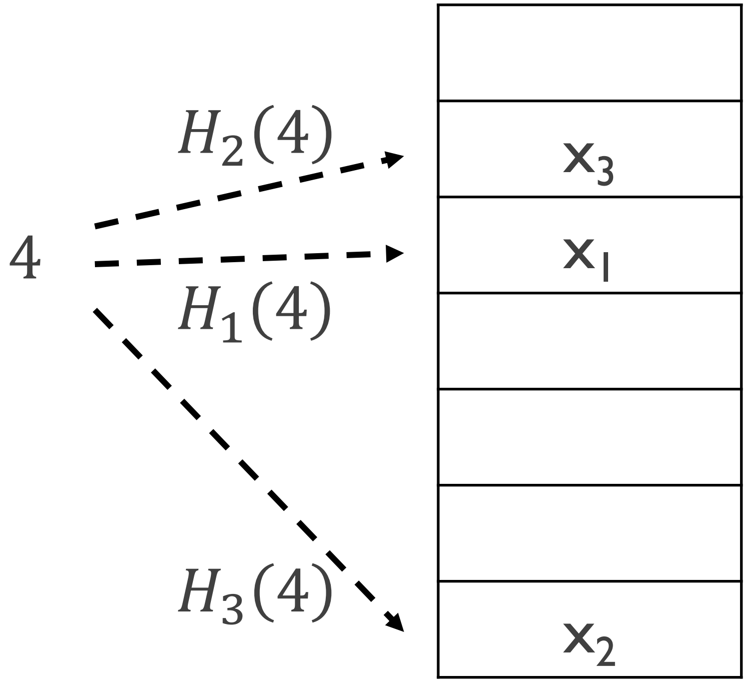
Background: Cuckoo Hashing

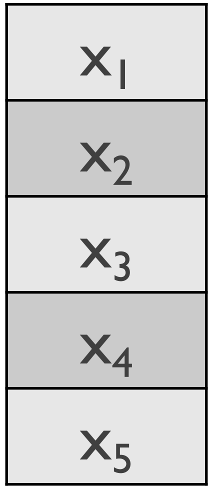
- A technique to build a collision-free hash table
- Each entry has multiple (e.g., 3) candidate locations



Background: Cuckoo Hashing

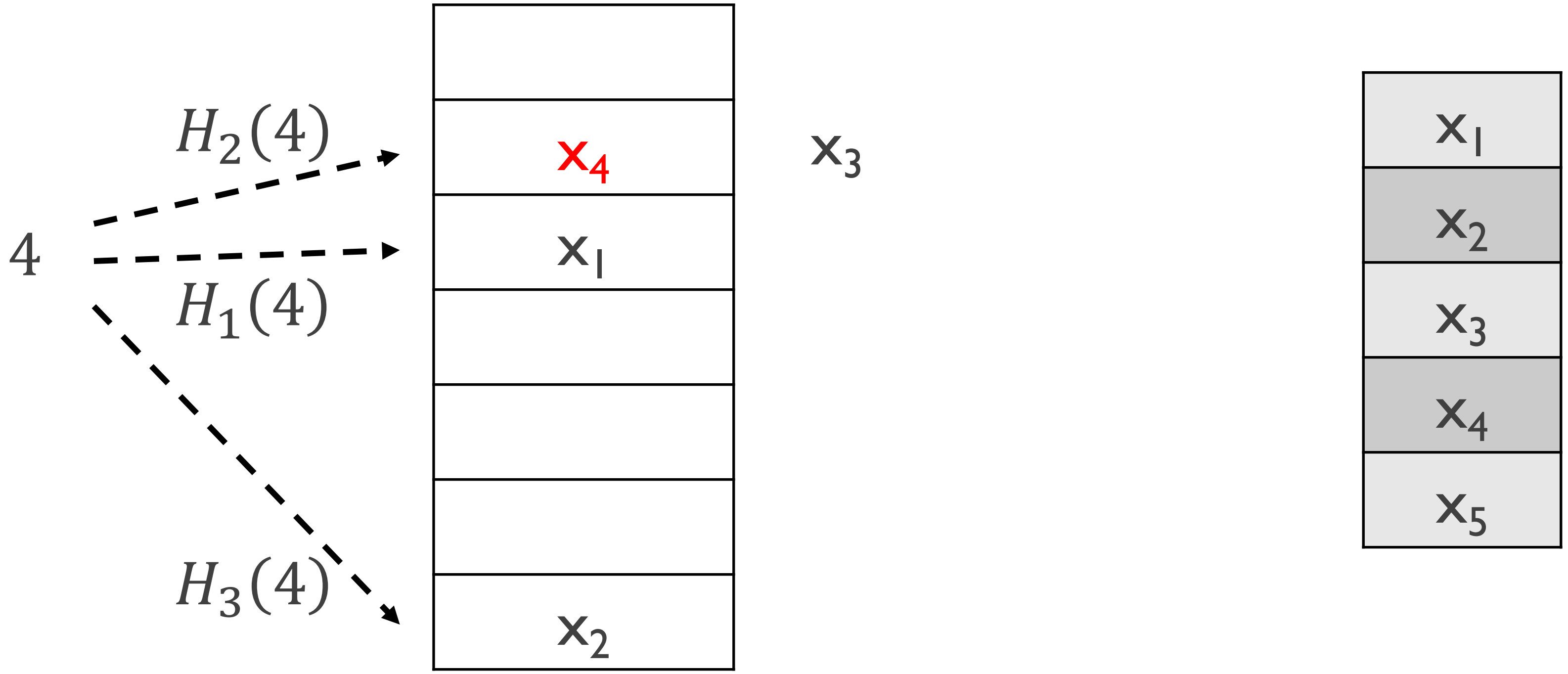
- What if none of the candidate locations is vacant?





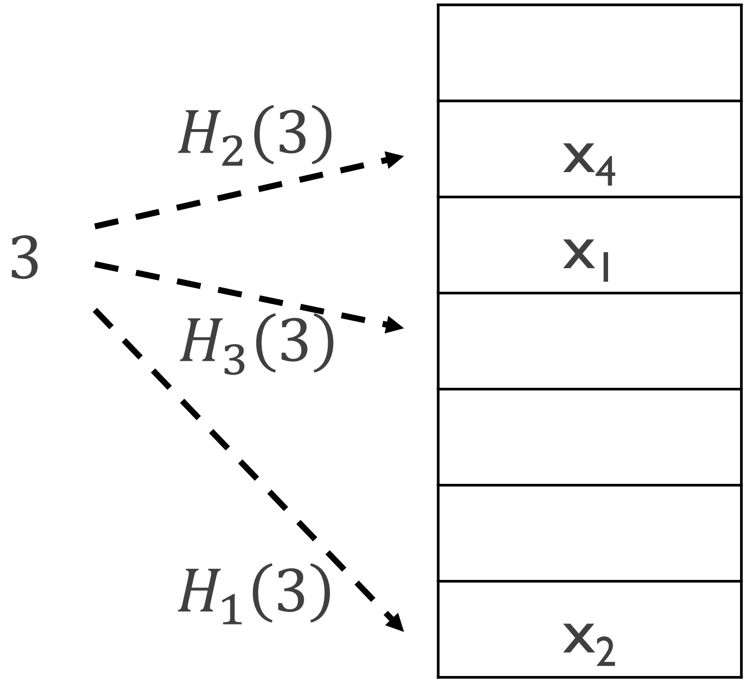
Background: Cuckoo Hashing

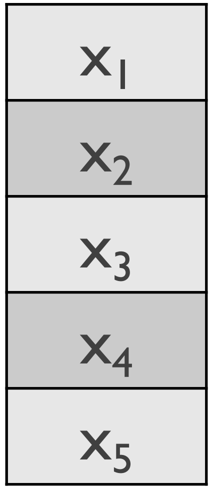
- What if none of the candidate locations is vacant?
  - Insert at a random candidate location and evict the entry already there



Background: Cuckoo Hashing

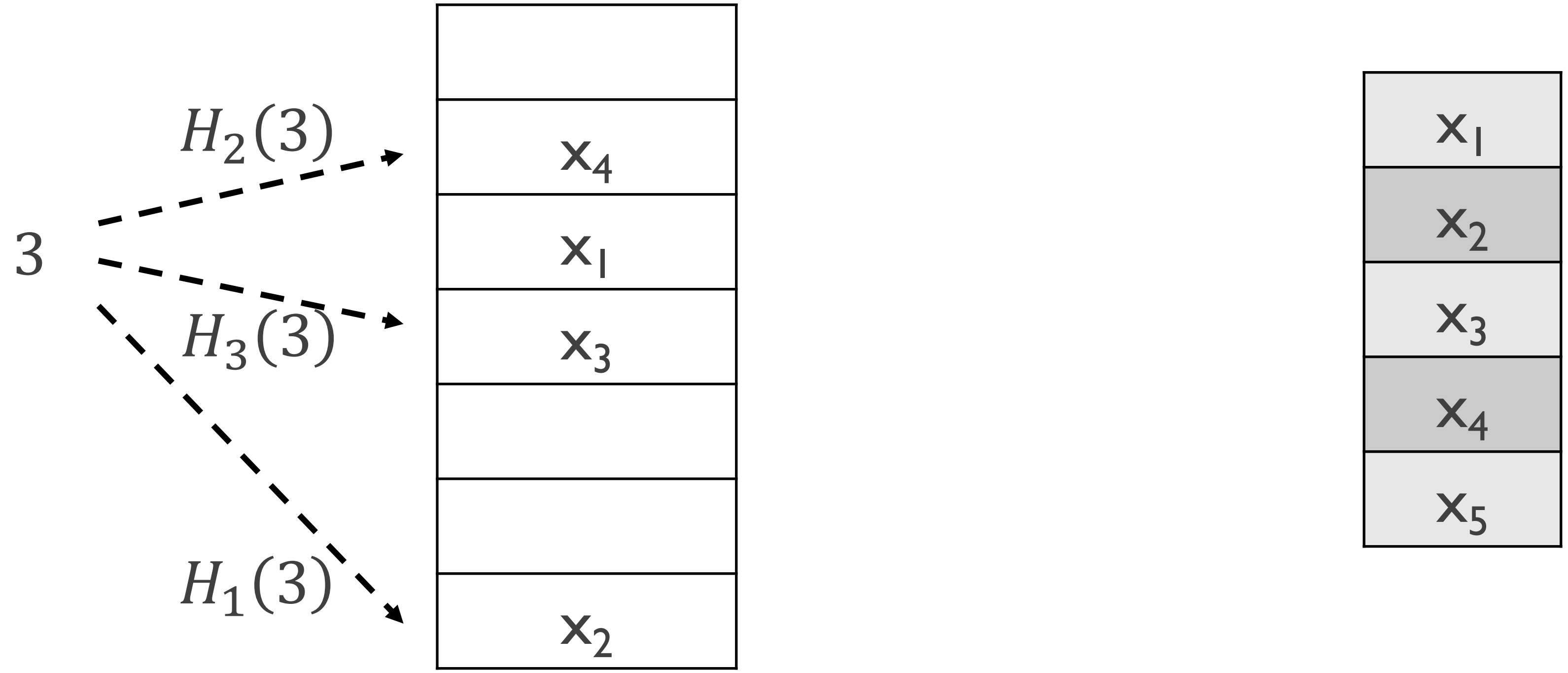
- What if none of the candidate locations is vacant?
  - Insert at a random candidate location and evict the entry already there
  - Re-insert the evicted entry



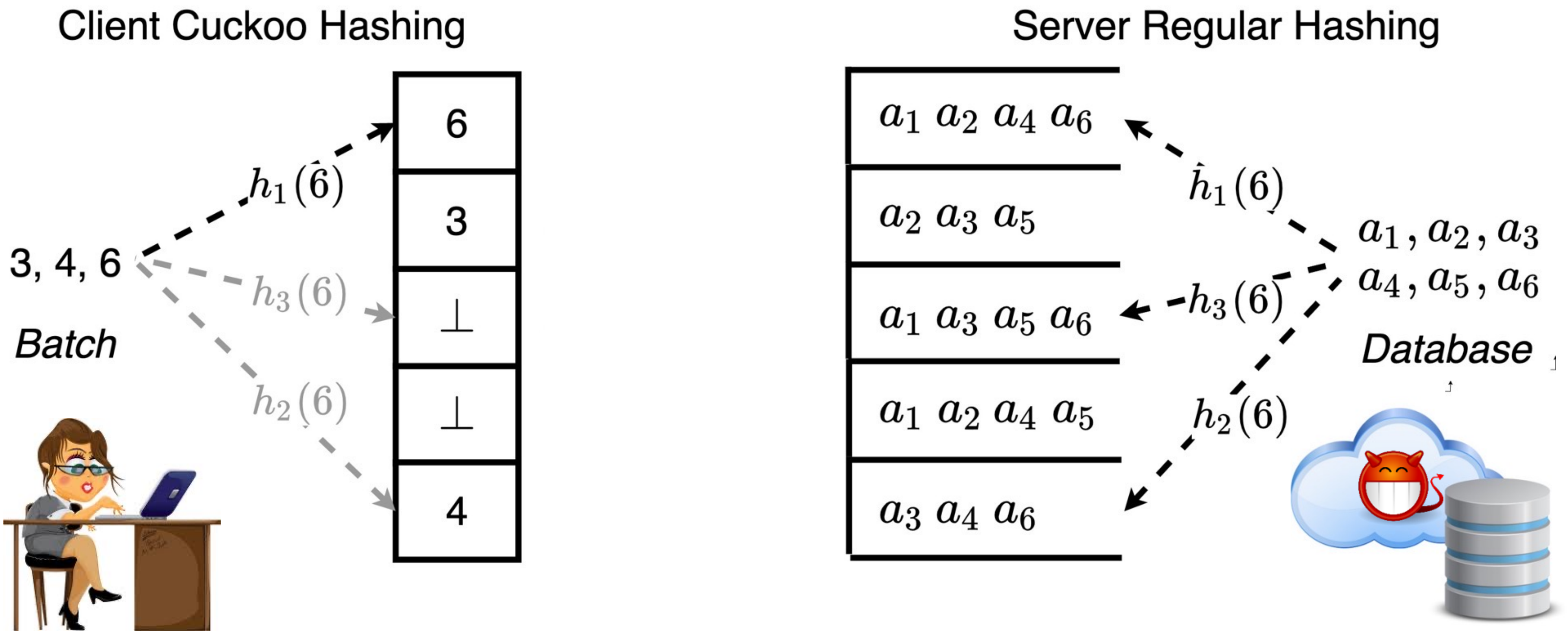


Background: Cuckoo Hashing

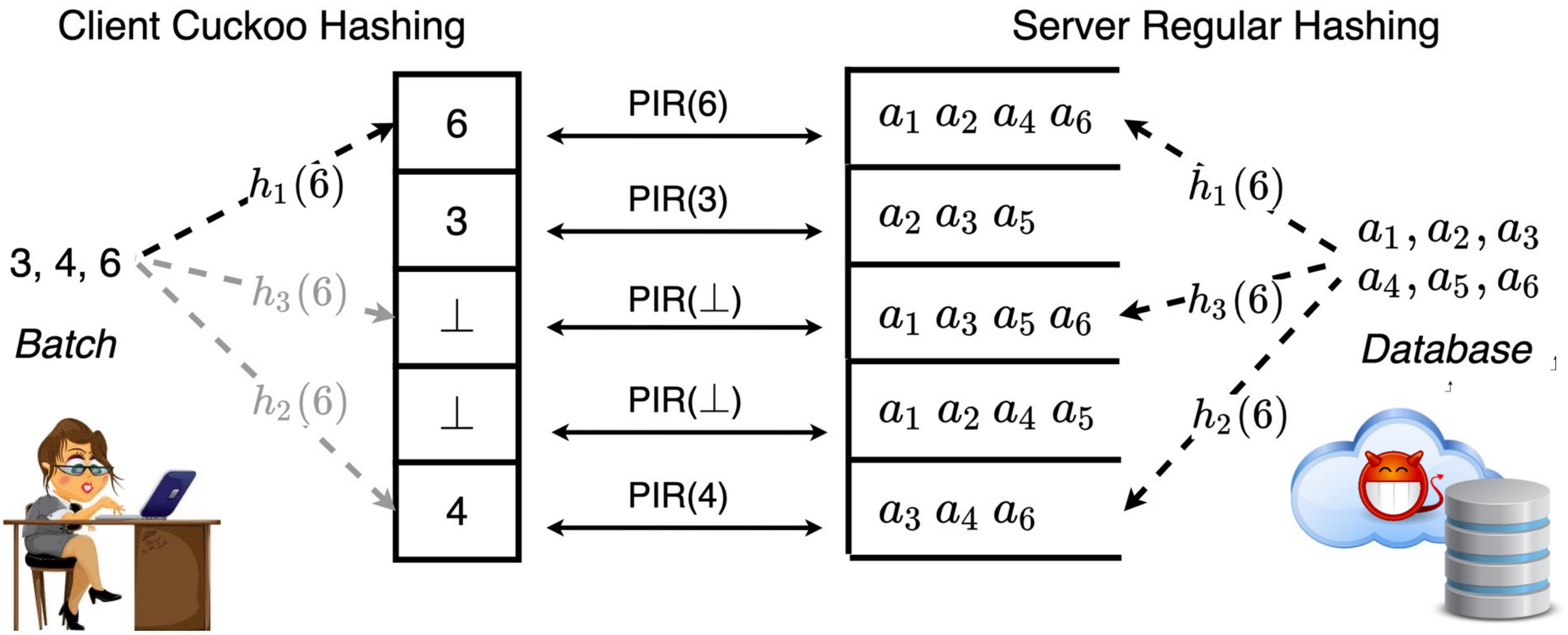
- What if none of the candidate locations is vacant?
  - Insert at a random candidate location and evict the entry already there
  - Re-insert the evicted entry, possibly evicting another entry
  - With proper table size, re-insertion won't continue forever



Batch PIR of [ACLS'18]



Batch PIR of [ACLS'18]



Batch PIR of [ACLS'18]

- Client cuckoo hashing, server regular hashing, per-bucket PIR
- ~3N computation (independent of batch size b)
- Response size: b ciphertexts, still inefficient for small entries

- Resolved recently using vectorized SHE in [MR'23], response can be a single ciphertext

Outline

- Single-server PIR using homomorphic encryption
- Limits of single-server PIR in the standard model
- Batch PIR
- **Amortized sublinear stateful PIR**

Stateful PIR [PPY'18, CK'20]

- Client wants to fetch multiple entries, but one at a time

Offline Phase

PIR(i_1)

PIR(i_2)

⋮

PIR(i_b)

Client stores hints

x_1
x_2
x_3
x_4
x_5

Amortized Sublinear PIR [CK'20]

- Client retrieves *hints* privately offline
- Each hint is the parity of a random subset (of size $\sqrt{n}$)
  - Need $\lambda\sqrt{n}$ hints to guarantee one such hint exists except exp(-λ) prob
- Online query for i: find a hint that contains x_i

$$H_1 = x_{68} \oplus x_{33} \oplus x_{19} \oplus x_{43}$$
$$H_2 = x_{31} \oplus x_{52} \oplus x_{14} \oplus x_{29}$$
$$\vdots$$
$$H_{23} = x_{25} \oplus x_{41} \oplus x_{29} \oplus x_{57}$$

Amortized Sublinear PIR [CK'20]

- Online query for i: find a hint that contains x_i
- Ideally, request = S \ {i}
- Server computes parity as response
- Answer = response $\oplus$ hint

$x_{25} \oplus x_{29} \oplus x_{57}$

$Q = \{25, 29, 57\}$

$S = \{25, 41, 29, 57\}$

$H_{23} = x_{25} \oplus x_{41} \oplus x_{29} \oplus x_{57}$

Amortized Sublinear PIR [CK'20]

- Client retrieves *hints* privately offline
- Each hint is the parity of a random subset (of size $\sqrt{n}$)
- Online query for i: find S ∋ i, (ideally) send S \ {i}, rest is easy
- Insecure: i won't appear in the request!
- Current solution: occasionally, keep i → correctness failure

  → λ parallel repetition → λ blowup to all metrics

S = {25, ~~41~~, 29, 57}

$H_{23} = x_{25} \oplus x_{41} \oplus x_{29} \oplus x_{57}$

Our New Protocol [MIR'23]

- Amortized sublinear stateful PIR without need for repetition
- Key idea: dummy subset of random indices
  - Make i appear with the "right" probability
  - Permute real and dummy subsets

41

Q' = {43, 16, 35}

Q = {25, 29, 57}

S = {25, 41, 29, 57}

$H_{23} = x_{25} \oplus x_{41} \oplus x_{29} \oplus x_{57}$

Our New Protocol [MIR'23]

- Amortized sublinear stateful PIR without need for repetition
- Key idea: dummy subset of random indices
  - Make i appear with the "right" probability and permute the two subsets
- Security: server cannot tell real vs dummy, i shows nothing special

Q' = {43, 16, 35}

Q = {25, 29, 57}

S = {25, 41, 29, 57}

$H_{23} = x_{25} \oplus x_{41} \oplus x_{29} \oplus x_{57}$

Our New Results [MIR'23]

	Communication	Computation	Client storage
Standard	28 KB	767 ms	
Stateful	3 KB	0.25 ms	6.25 MB

For a database of 2^{20} entries, each of 32 byte (32 MB in total)

	Communication	Computation	Client storage
Standard	35 KB	30 s	
Stateful	47 KB	4.5 ms	100 MB

For a database of 2^{28} entries, each of 32 byte (8 GB in total)

Summary

- State-of-art PIR in standard model: hierarchical PIR with SHE
  - 36 KB request and 3x response
  - Expensive computation, large response for small entries, per-client storage
- Batch PIR with vectorized SHE
  - $O(n)$ computation per batch, single ciphertext response
  - Must query in batch, request size grows with *n*
- Amortized sublinear stateful PIR
  - $O(\sqrt{n})$ request, millisecond computation, 2x online response
  - $O(\lambda\sqrt{n})$ client storage, update is a challenge