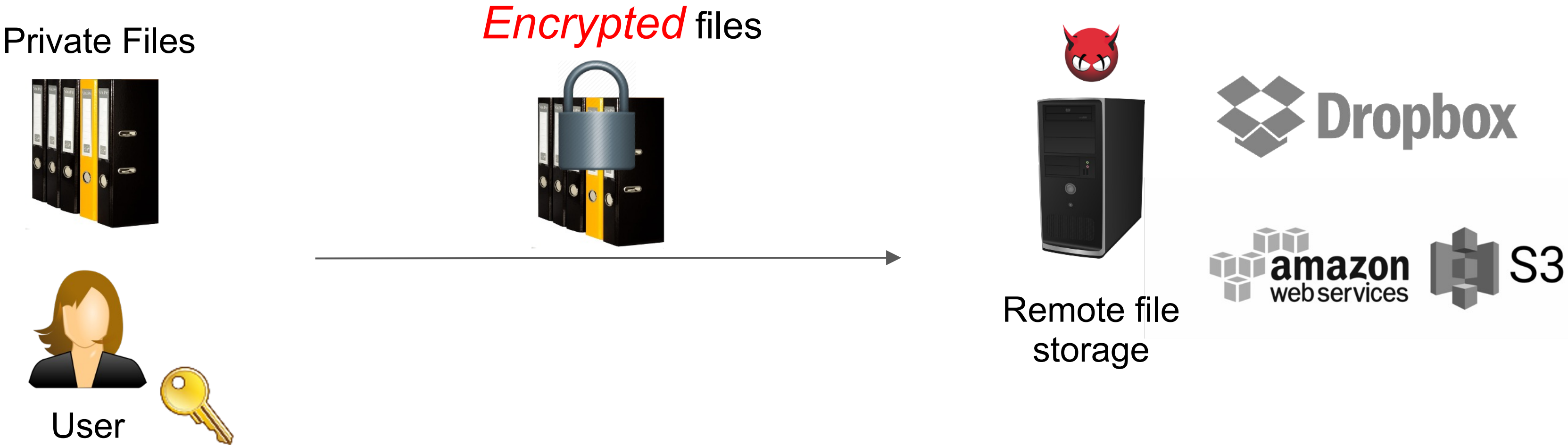
Private Information Retrieval in Large Scale Public Data Repositories

Ishtiyaque Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta

University of California, Santa Barbara

The problem of protecting *private data repositories* stored remotely is well-studied



Encryption hides file contents from an attacker.

Encryption does not hide data access patterns

The access patterns leaks:

- Which file is being accessed?
- When was it last accessed?
- Is it being accessed for a read or a write?
- Is it being accessed sequentially or randomly?
- …

ORAM (STOC ‘87) hides data access patterns for private files

Private Files

User

Oblivious RAM

(Goldreich STOC ‘87, Path ORAM JACM ‘18, SCORAM CCS ‘14, …)

Encryption +
randomized data accesses

Remote file storage

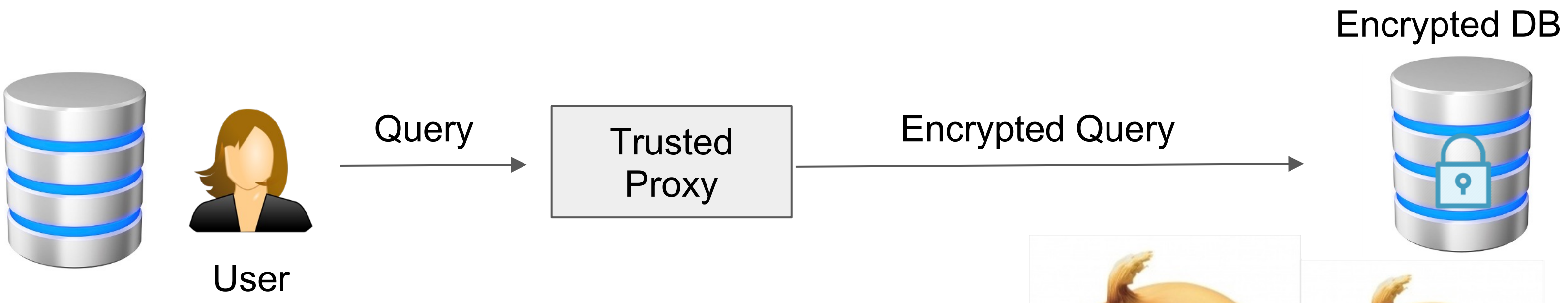
Hidden:

- ➔ Which file is being accessed?
- ➔ Whether the access is a read or write
- ➔ When was the file accessed last

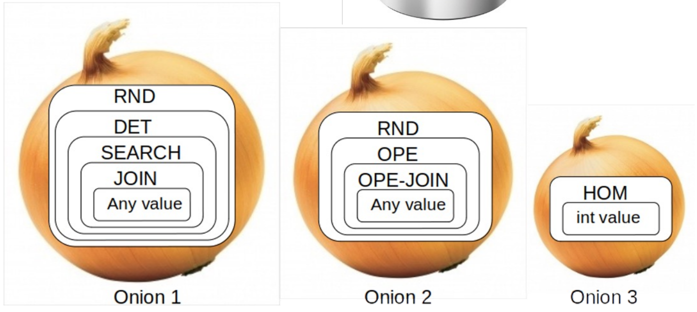
…

We can extend protection to *private relational databases* stored remotely

CryptDB SOSP ‘11, MONOMI VLDB ‘13, …



Hidden:
- ➔ Database content
- ➔ Query parameters



Adjustable query-based encryption (onion)

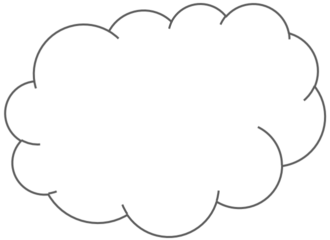
What is common to all of these cases?

Private Files

Private database

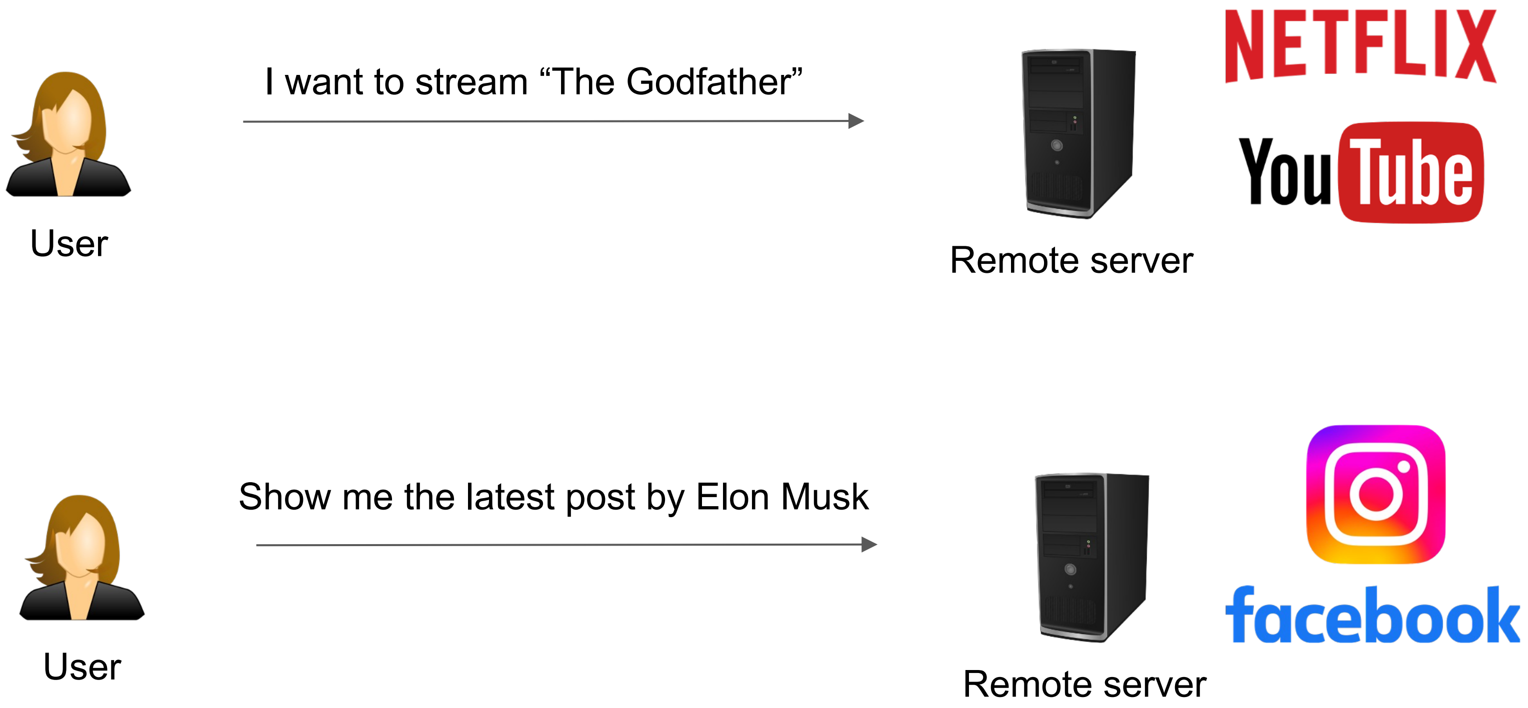
User

Securely outsource storage

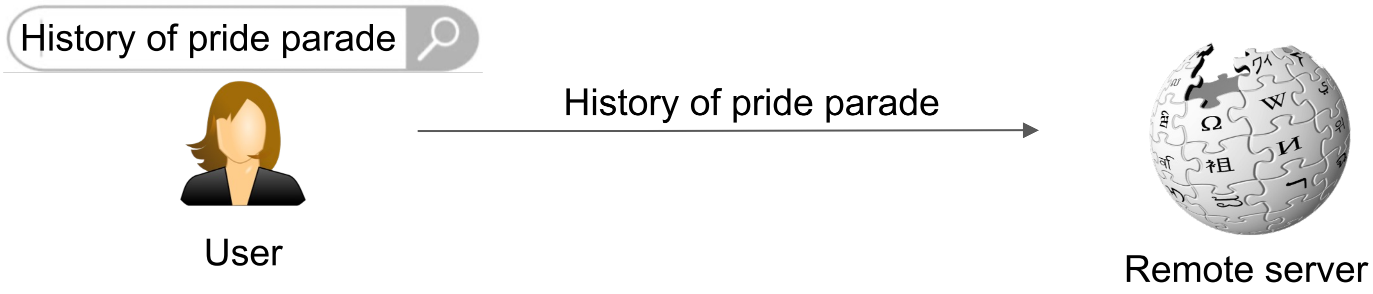


The user owns the data!

But, much of the content on the Internet is in ***public*** *data repositories*



But, much of the content on the Internet is in ***public*** *data repositories*



Cannot use:

- Encryption
- ORAM
- CryptDB-like solution

How can we hide access patterns (queries) over public data repositories?

Both users and service providers want to hide access patterns over public repositories

User may:
- Consider queries private
- Belong to a vulnerable population or a minority group

Server can be:
- Hacked by an outsider
- Compromised by an insider
- Coerced by a nation state [1, 2]

1. Brian Fung. *Analysis: There is now some public evidence that China viewed TikTok data.* CNN, 2023.
2. Sapna Maheshwari and Ryan Mac. *Driver's Licenses, Addresses, Photos: Inside How TikTok Shares User Data.* New York Times, 2023

This tutorial:

Discuss a cryptographic method to privately retrieve data from public data repositories, thus making server *opaque* to data access patterns

Private retrieval from public databases can be abstracted into the key-value store model

k

Client retrieves:

- v, if (k,v) at Server
- ∅, otherwise

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

Focus on **performance**, **scalability**, and **practicality**

This tutorial is in two parts

Part 1: Retrieval by location

key	location
k_0	0
k_1	1
…	…
k_{n-1}	n-1

Has (key, location) mapping

Give me the *i*-th value

0	v_0
1	v_1
2	v_2
…	…
n-1	v_{n-1}

Untrusted Server

Part 1: How can the client privately retrieve the value corresponding to a given location?

This tutorial is in two parts

Part 2: Retrieval by key

key	location
k_0	0
k_1	1
…	…
k_{n-1}	n-1

k

Client retrieves:

- v, if (k,v) at Server
- ∅, otherwise

Give me value for key k

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

Part 2: How can the client privately retrieve the value corresponding to a given key?

This tutorial is in two parts

Part 1: Retrieval by location

key	**location**
k_0	0
k_1	1
…	…
k_{n-1}	n-1

Has (key, location) mapping

Give me the *i*-th value

0	v_0
1	v_1
2	v_2
…	…
n-1	v_{n-1}

Untrusted Server

Part 1: How can the client privately retrieve the value corresponding to a given location?

This problem can be solved using **Private Information Retrieval** (PIR) (Chor et al. FOCS ‘95)

PIR: `Query, Answer, Decode`

I want db[i]

User

Step 1:
q = `Query`(*i*)

q

Step 3:
db[i] = `Decode`(ans)

ans

db

0	01011001010101001010
1	01110001010101001010
2	10011001010101001010
.	11011001010101001010
.	01011011010101001110
n - 1	01101001110101101010

Untrusted Server

Step 2:
ans = `Answer`(db, q)

PIR has two key requirements

Correctness

Query for db[i] returns db[i] to the user

`Decode`(`Answer`(db, `Query`(*i*))) = db[i]

Privacy

Server learns “nothing” about the location i

For all locations i, j,

{View of the server in answering `Query`(i)} ≈

{View of the server in answering `Query`(j)}

We are also interested in performance considerations

Network cost

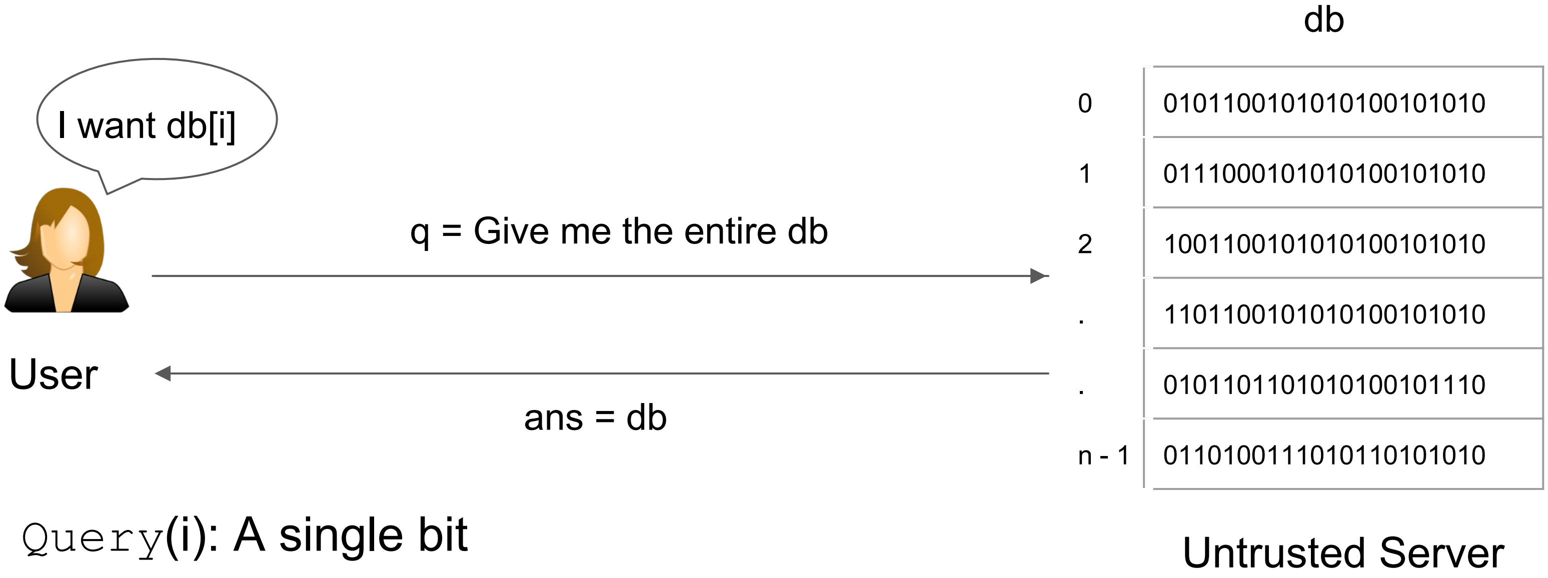
Request size: |`Query`(i)|

Response size: |`Answer`(db, `Query`(i))|

Compute cost

Time to compute `Answer`(db, `Query`(i))

One solution to private information retrieval in *Trivial PIR*



`Query`(i): A single bit

`Answer`(db, q): db

`Decode`(i, ans): select the i-th item from ans

Performance characteristics of trivial PIR

Network cost

Request size: **1 bit**

Response size: **n x |db[i]|**

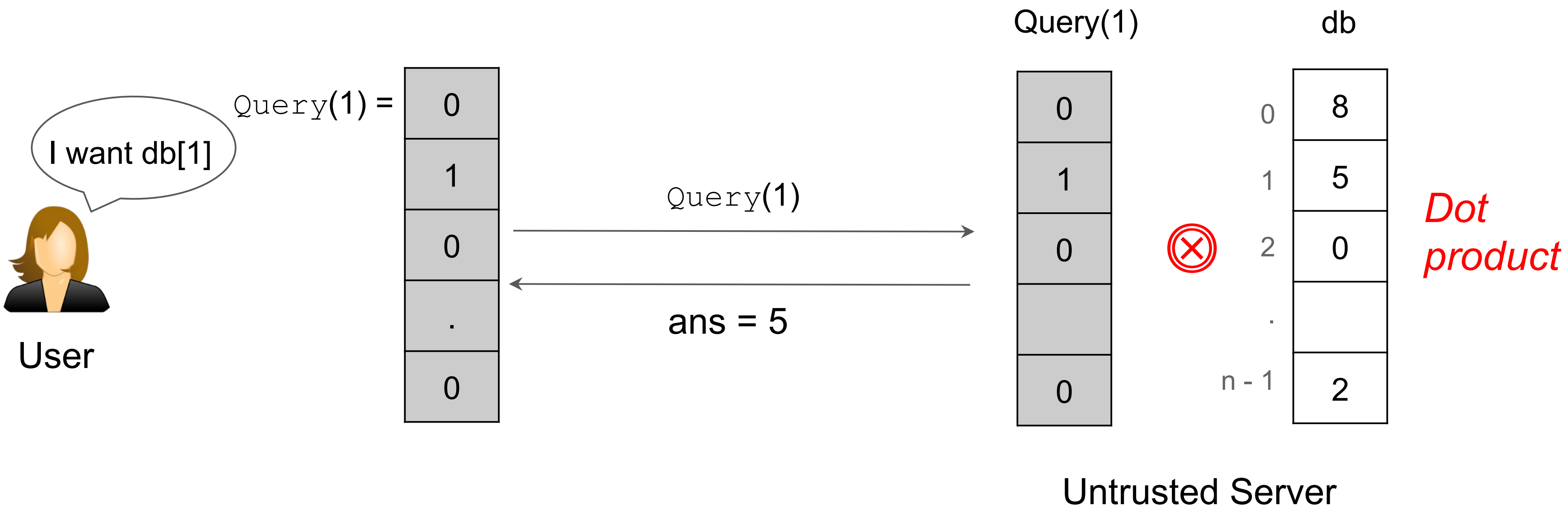
Compute cost

Time to compute `Answer`(db, `Query`(i))

Can we do better than sending the entire database? If so, how?

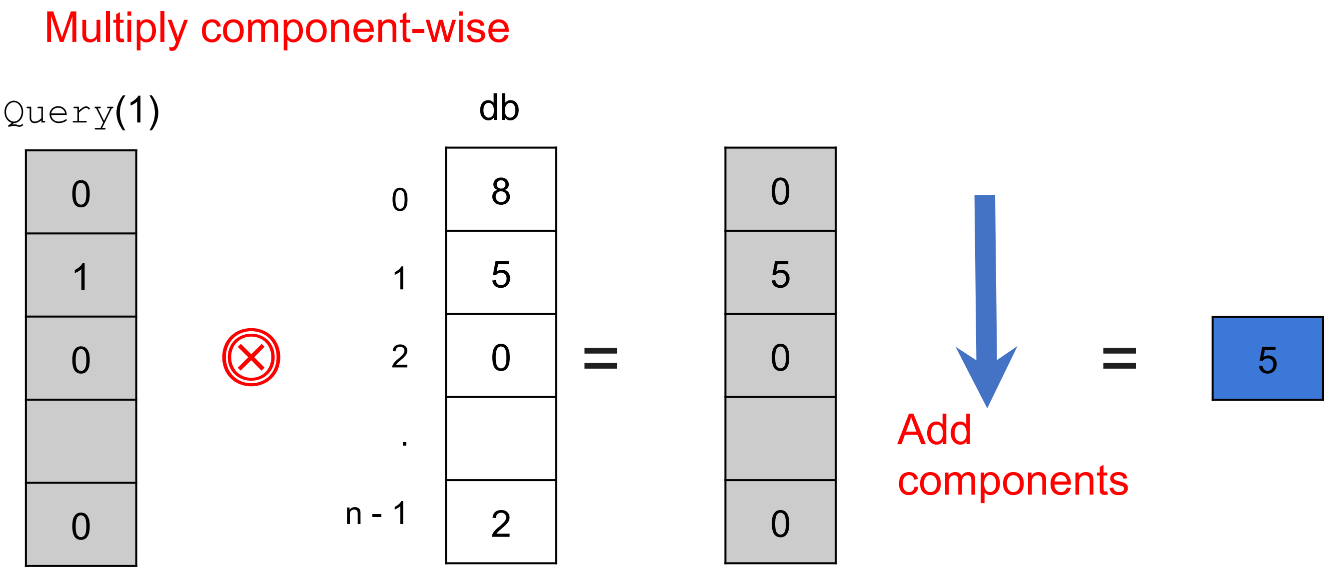
Warmup for (non-trivial) PIR

Assume that we do not care about privacy yet; only correctness



Retrieval is equivalent to computing a dot product

Warmup for (non-trivial) PIR in more detail



Dot product requires two types of operations:

- ➔ *Multiplications (8 x 0, 5 x 1, etc.)*
- ➔ *Additions (e.g., 0 + 5 + …)*

Detour: Introduction to Homomorphic Encryption

A form of encryption which allows computations over encrypted data

Two classes of homomorphic encryption

Fully Homomorphic Encryption [Gentry'09]

- Supports computations for any arbitrary function
- Challenge: Can be Quite inefficient

Partially Homomorphic Encryption

- Supports a particular type of operation

Additive Homomorphic encryption

Enc(4) $\oplus$ Enc(8) = Enc(4 + 8) = Enc(12)

Multiplicative Homomorphic encryption

Enc(4) $\otimes$ Enc(8) = Enc(4 x 8) = Enc(32)

Detour: Introduction to Homomorphic Encryption

A form of encryption which allows computations over encrypted data

Two classes of homomorphic encryption

Fully Homomorphic Encryption [Gentry'09]

- Supports computations for any arbitrary function
- Challenge: Can be Quite inefficient

Partially Homomorphic Encryption

- Supports a particular type of operation

Additive Homomorphic encryption

$\text{Enc}(4) \oplus \text{Enc}(8) = \text{Enc}(4 + 8) = \text{Enc}(12)$

Multiplicative Homomorphic encryption

$\text{Enc}(4) \otimes \text{Enc}(8) = \text{Enc}(4 \times 8) = \text{Enc}(32)$

Example: El Gamal additive homomorphic encryption

We have a message m which we want to encrypt

Encryption key: (g, h)

Encryption procedure:

Pick a random number r

$\mathrm{Enc}(m, r) = (g^r, g^m h^r)$

Example: El Gamal additive homomorphic encryption

$Enc(m1) \times Enc(m2) = Enc(m1+ m2)$

$Enc(m, r) = (g^r, g^m h^r)$

The product of the encryptions of two messages is *an* encryption of the sum of the two messages.

Given two messages m1 and m2

$Enc(m1, r1) = (g^{r1}, g^{m1}h^{r1})$

$Enc(m2, r2) = (g^{r2}, g^{m2}h^{r2})$

$$Enc(m1, r1) \times Enc(m2, r2) = (g^{r1}, g^{m1}h^{r1}) \times (g^{r2}, g^{m2}h^{r2})$$

$$= (g^{r1+r2}, g^{m1+m2}h^{r1+r2})$$

$$= Enc(m_1 + m_2 , r_1 + r_2)$$

Example: El Gamal additive homomorphic encryption

$\boxed{\mathrm{Enc}(m1) \times \mathrm{Enc}(m2) = \mathrm{Enc}(m1+m2)}$

Additive Homomorphic Encryption supports multiplying an encrypted value with a plaintext value

We have a message m, encrypted as Enc(m)

We have another message k (not encrypted)

$$
\begin{aligned}
[\mathrm{Enc}(m)]^k &= \mathrm{Enc}(m) \times \mathrm{Enc}(m) \times \ldots. \times \mathrm{Enc}(m) \\
&= \mathrm{Enc}(m + m + \ldots + m) \\
&= \mathrm{Enc}(m * k)
\end{aligned}
$$

$\boxed{\mathrm{Enc}(m)^k = \mathrm{Enc}(m * k)}$

We only need additive homomorphic encryption for PIR

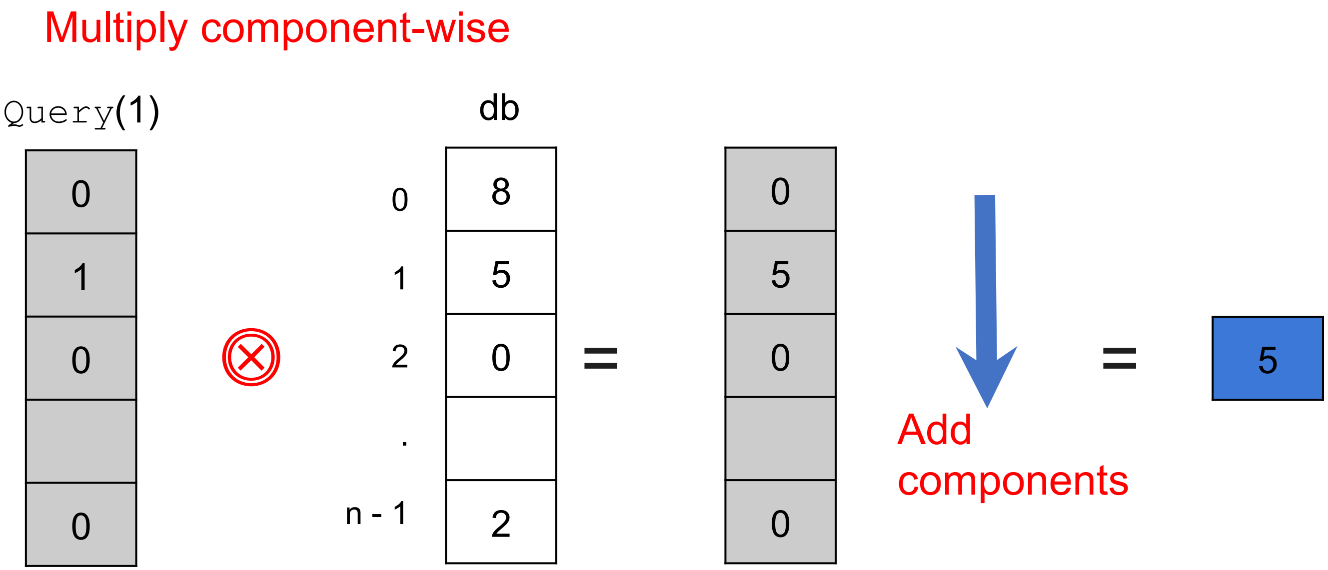
Homomorphic addition

$\text{Enc}(m_1) \times \text{Enc}(m_2) = \text{Enc}(m_1 + m_2)$

Homomorphic plaintext multiplication

$\text{Enc}(m)^k = \text{Enc}(m * k)$

Recall the warmup for (non-trivial) PIR

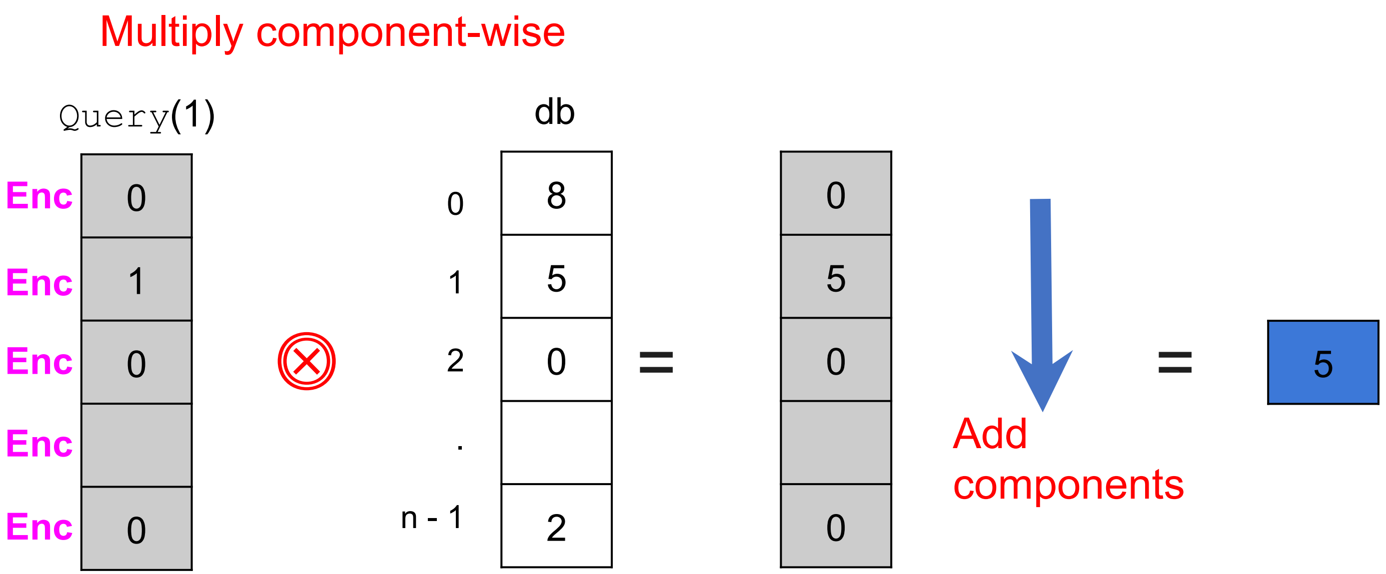


Dot product requires two types of operations:

➔ *Multiplications (8 x 0, 5 x 1, etc.)*

➔ *Additions (e.g., 0 + 5 + …)*

Recall the warmup for (non-trivial) PIR



Dot product requires two types of operations:

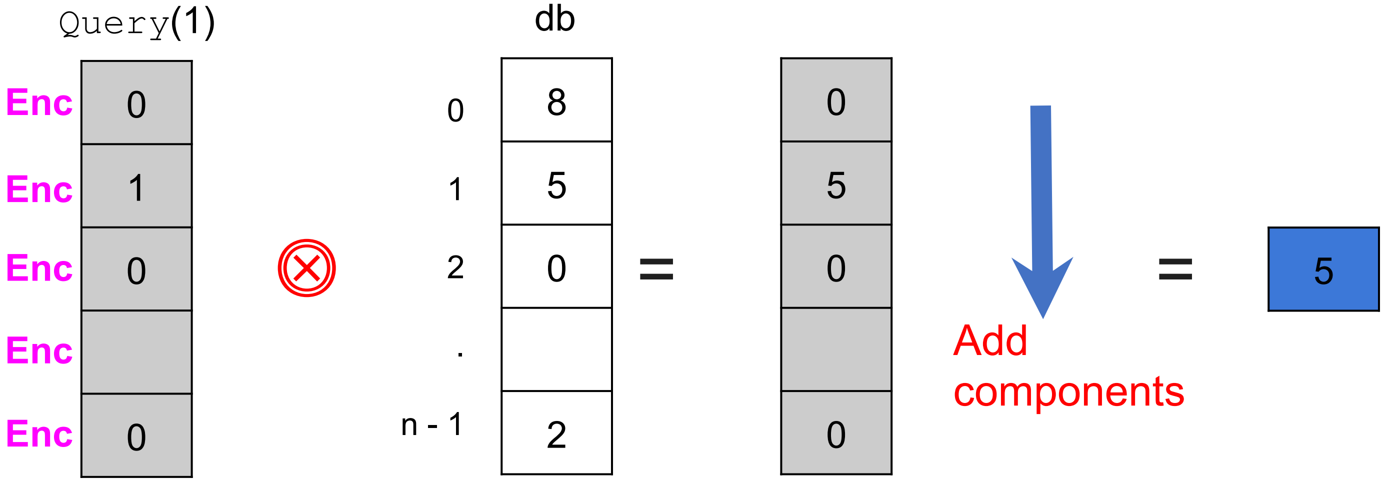
➔ *Multiplications (8 x 0, 5 x 1, etc.)*

➔ *Additions (e.g., 0 + 5 + …)*

Recall the warmup for (non-trivial) PIR

Homomorphically multiply component-wise

$\mathrm{Enc}(m)^k = \mathrm{Enc}(m * k)$



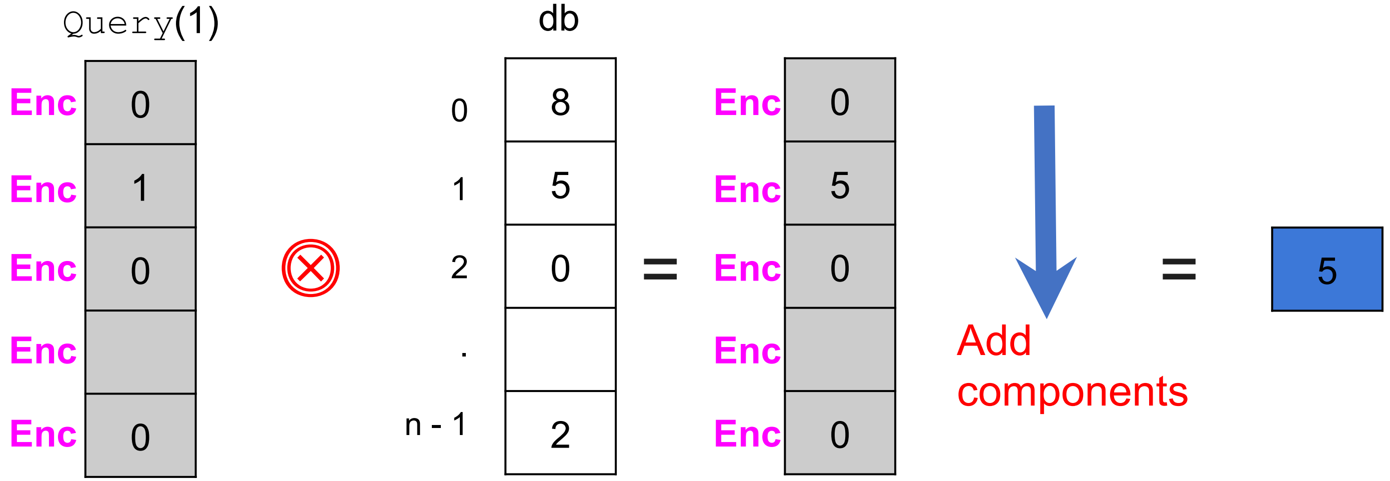
Dot product requires two types of operations:

➔ *Multiplications (8 x 0, 5 x 1, etc.)*

➔ *Additions (e.g., 0 + 5 + …)*

Recall the warmup for (non-trivial) PIR

Homomorphically multiply component-wise $\boxed{\mathrm{Enc}(m)^k = \mathrm{Enc}(m * k)}$



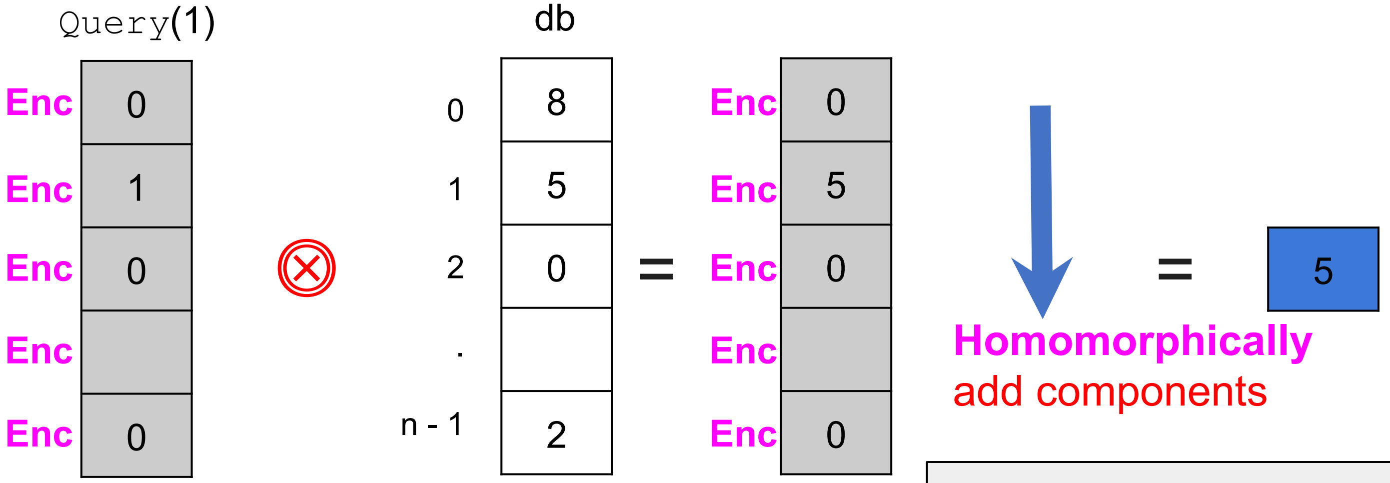
Dot product requires two types of operations:

- ➔ *Multiplications (8 x 0, 5 x 1, etc.)*
- ➔ *Additions (e.g., 0 + 5 + …)*

Recall the warmup for (non-trivial) PIR

Homomorphically multiply component-wise

$\mathrm{Enc}(m)^k = \mathrm{Enc}(m * k)$



$\mathrm{Enc}(m_1) \times \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 + m_2)$

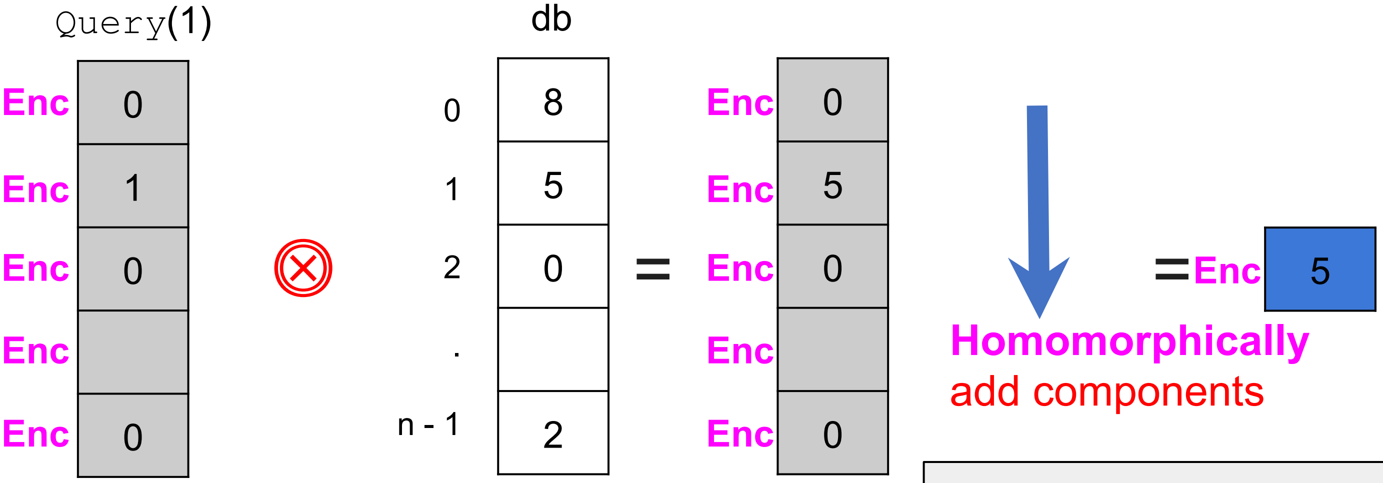
Dot product requires two types of operations:

- ➔ *Multiplications (8 x 0, 5 x 1, etc.)*
- ➔ *Additions (e.g., 0 + 5 + …)*

Recall the warmup for (non-trivial) PIR

Homomorphically multiply component-wise

$\mathrm{Enc}(m)^k = \mathrm{Enc}(m * k)$



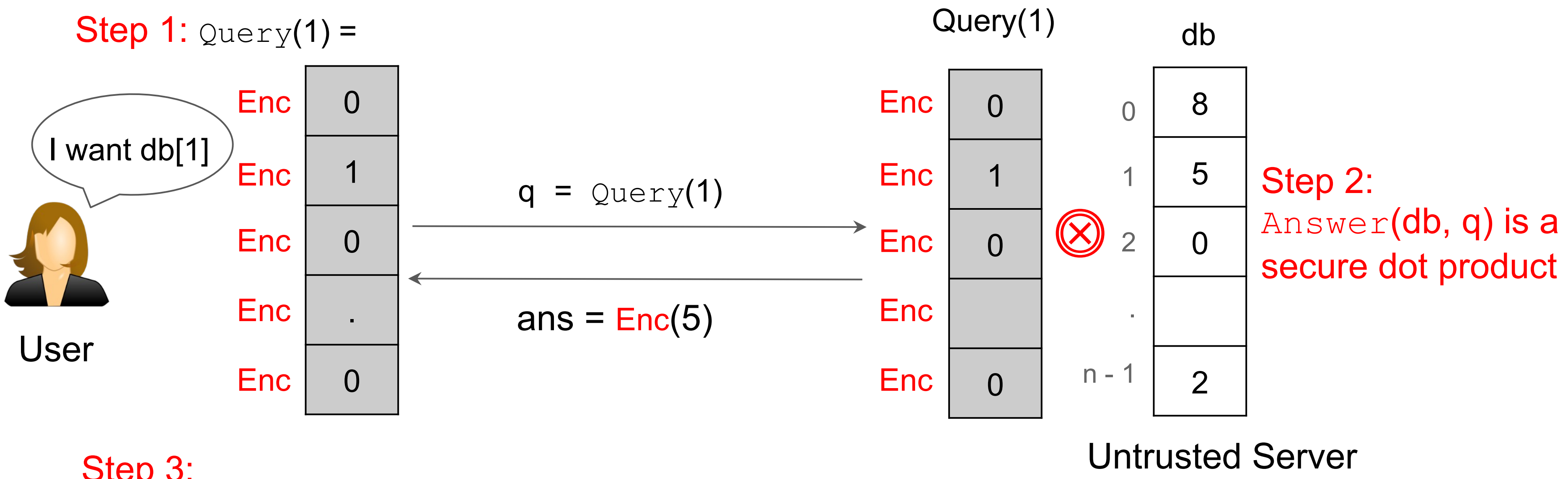
Homomorphically add components

$\mathrm{Enc}(m_1) \times \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 + m_2)$

Dot product requires two types of operations:

- ➔ *Multiplications (8 x 0, 5 x 1, etc.)*
- ➔ *Additions (e.g., 0 + 5 + …)*

Putting it all together: A PIR protocol



Step 3:
db[1] = Decode(ans) = Decrypt(ans)

Retrieval is equivalent to computing a secure dot product

What is the size of the PIR response?

Response is a ciphertext: *Enc(db[i])*

Recall:

$$\mathrm{Enc}(m, r) = (g^r, g^m h^r)$$

Encrypting 1 message yields 2 components

Expansion factor, *f = size of ciphertext / size of plaintext*

Expansion factor for El Gammal = 2

Performance characteristics of additively HE-based PIR

Network cost

Request size: **n x |ciphertext|**

Response size: **|ciphertext|**

Expansion factor: f = **|ciphertext| / |db[i]|**

Compute cost

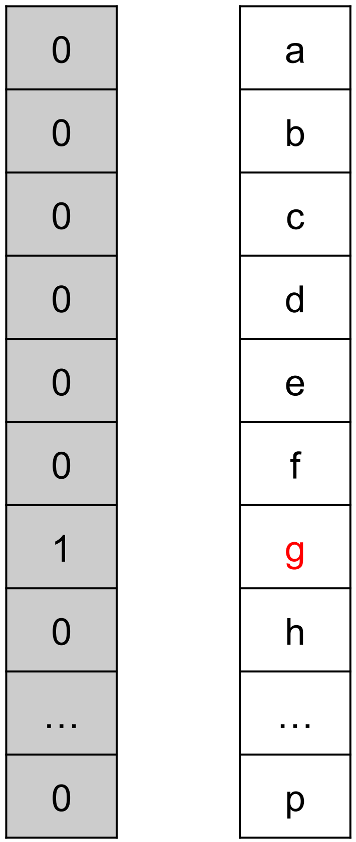
Time to compute `Answer`(db, `Query`(i)) is **O(n) homomorphic ops**

This linear compute overhead is a fundamental lower bound (Beimel et al. CRYPTO '00)

Much of the research on PIR is on reducing request size and server-side compute overhead

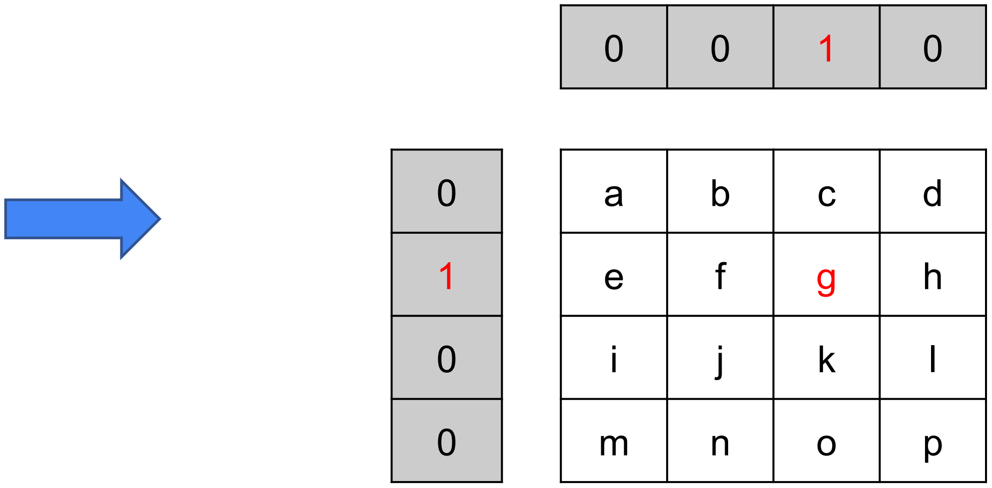
Overhead	High-level technique
Request size	• Recursion (Stern 1998) • Cryptographic query compression (SealPIR ‘18)
Server-side compute	• PIR with preprocessing (Beimel et al. ‘00, SimplePIR ‘23) • Lattice-based cryptography (FastPIR ‘21)

How to reduce query size?



Instead of 1 dim database, view it in 2 dims.

Instead of 1 query, use 2 queries.



Two-stage query execution

$$\begin{bmatrix} 0 \\ {\color{red}1} \\ 0 \\ 0 \end{bmatrix} * \begin{bmatrix} a & b & c & d \\ e & f & {\color{red}g} & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ e & f & {\color{red}g} & h \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Add rows

$$\begin{bmatrix} e & f & {\color{red}g} & h \end{bmatrix}$$

In first pass, extract the row of interest

Two-stage query execution

e	f	g	h

$*$

0	0	1	0

$=$

0	0	g	0

Add columns

g

So, query size is down from n to $2\sqrt{n}$.

But result is double encrypted

- After first stage, each element is a ciphertext, size is *f * plaintext size*
- After second stage, result size is f^2 * plaintext size
- The efficient homomorphic encryption schemes can have $f \geq 8$

Trade-off between query and response size

Stern (1998) recursion scheme

- Reduce query size to $d*\sqrt[d]{n}$
- Expand result size by f^d
- Used in XPIR (2016)

Much of the research on PIR is on reducing request size and server-side compute overhead

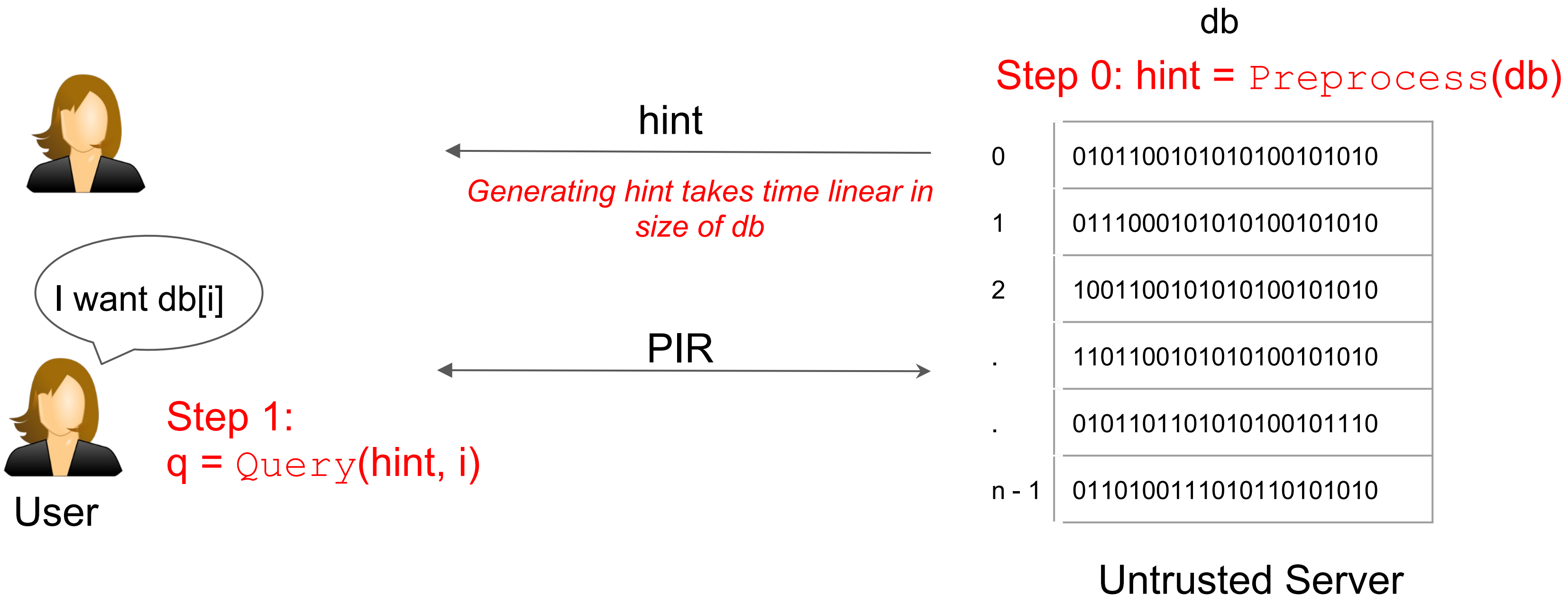
Overhead	High-level technique
Request size	• Recursion (Stern 1998) • Cryptographic query compression (SealPIR '18)
Server-side compute	• PIR with preprocessing (Beimel et al. '00, SimplePIR '23) • Lattice-based cryptography (FastPIR '21)

SealPIR (Microsoft Research - 2018)

- Compress query by a large factor (2^{11})
- Trade-off: query expansion at the server requires high compute cost

How to reduce server-side compute overhead?

PIR with preprocessing (Beimel et al CRYPTO ‘00, SimplePIR ‘23)



Does not violate the linear compute lower bound (Beimel et al. CRYPTO ‘00)

How to reduce server-side overhead?

Another option is to pay linear overhead but improve the constant

Key techniques in FastPIR (OSDI ‘21)

- Use lattice-based additively homomorphic encryption scheme
- Single-input multiple data (SIMD) capabilities
- Query and response compression using homomorphic rotation operations

FastPIR has lower processing time than all other variants (that do not use preprocessing)

Experiment results (c5.12x large in AWS; 1M values, 256 bytes each)

PIR Scheme	Processing time (ms)	Response size (KB)
FastPIR	**947**	64
XPIR-1	3,389	32
XPIR-2	1,894	288
SealPIR-1	76,216	32
SealPIR-2	2,556	320

This tutorial is in two parts

Part 1: Retrieval by location

k_0	0
k_1	1
k_2	2
…	…
k_{n-1}	n-1

Has (key, location) mapping

Give me the *i*-th value

0	v_0
1	v_1
2	v_2
…	…
n-1	v_{n-1}

Untrusted Server

Part 1: How can the client privately retrieve the value corresponding to a given location?

This tutorial is in two parts

Part 2: Retrieval by key ?

k_1	1
k_2	2
k_3	3
…	…
k_n	n

k

Client retrieves:

- v, if (k,v) at Server
- ∅, otherwise

Give me value for key k

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

Part 2: How can the client privately retrieve the value corresponding to a given key?

This area originated as Private retrieval by keywords in 1998
(Chor et al. TOC '98)

Private Keyword retrieval can be performed by two stages:

Stage 1: Retrieve the key location

k

Give me the location for key k

i

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Stage 2: Perform PIR with location

Has (key, location) mapping

Give me the *i*-th value

0	v_0
1	v_1
2	v_2
…	…
n-1	v_{n-1}

PIR-by-keywords has two requirements

Correctness

Query for k returns v iff (k, v) is in db

Privacy

Server learns "nothing" about the key k

For any two possible keys k_i, k_j

{View of the server in answering `Query`(k_i)} ≈

{View of the server in answering `Query`(k_j)}

We are also interested in performance considerations

Network cost

Request size, Response size

Number of round trips between user and server

Compute cost

Time to compute the response

This area originated as Private retrieval by keywords in 1998

(Chor et al. TOC '98)

Private Keyword retrieval can be performed by two stages:

Stage 1: Retrieve the key location

k

Give me the location for key k

i

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Stage 2: Perform PIR by index

Has (key, location) mapping

Give me the *i*-th value

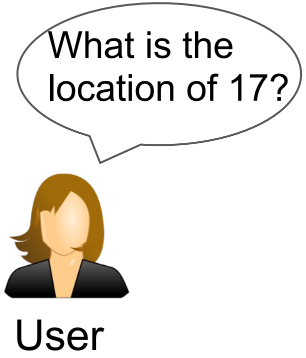
0	v_0
1	v_1
2	v_2
…	…
n-1	v_{n-1}

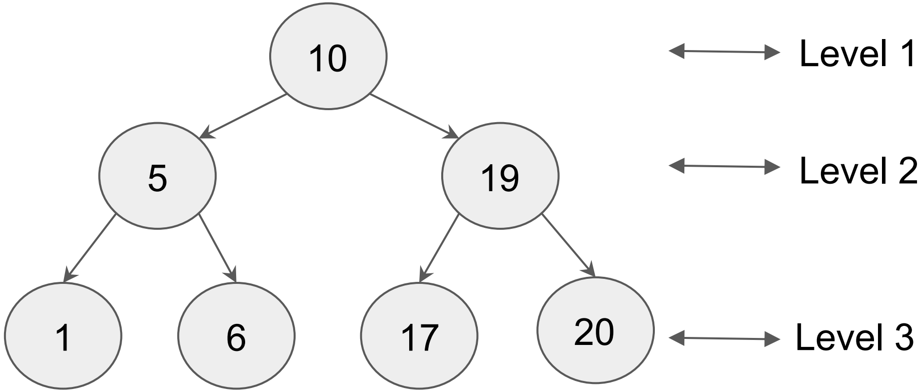
Key location can be retrieved using PIR-by-index

(Chor et al. TOC '98)

Assume keys are integers and arranged in a BST

K = {1, 5, 6, 10, 17, 19, 20}





Untrusted Server

Key location can be retrieved using PIR-by-index

(Chor et al. TOC ‘98)

Assume keys are integers and arranged in a BST

K = {1, 5, 6, 10, 17, 19, 20}

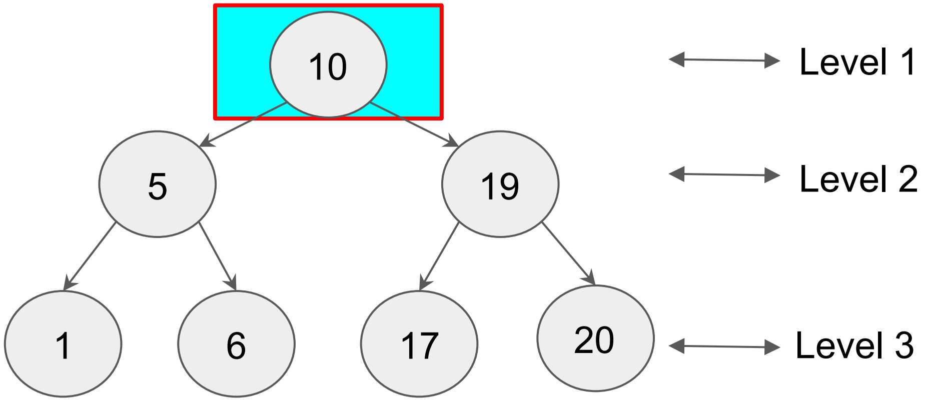
What is the location of 17?

User

Level 1: Retrieve element at index 0 (trivial)

10 < 17

Go right



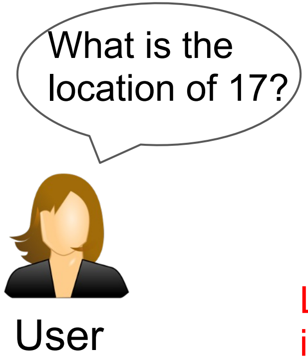
Untrusted Server

Key location can be retrieved using PIR-by-index

(Chor et al. TOC ‘98)

Assume keys are integers and arranged in a BST

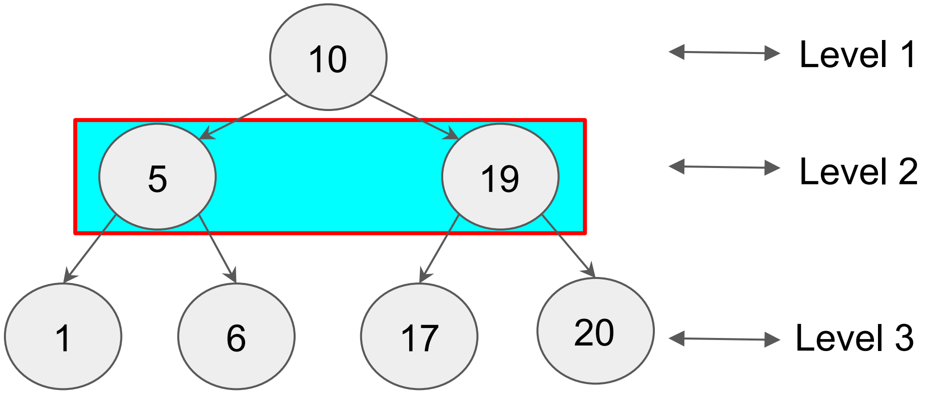
K = {1, 5, 6, 10, 17, 19, 20}



Level 2: Retrieve element at index 1 using PIR-by-index

17 < 19

Go left



Untrusted Server

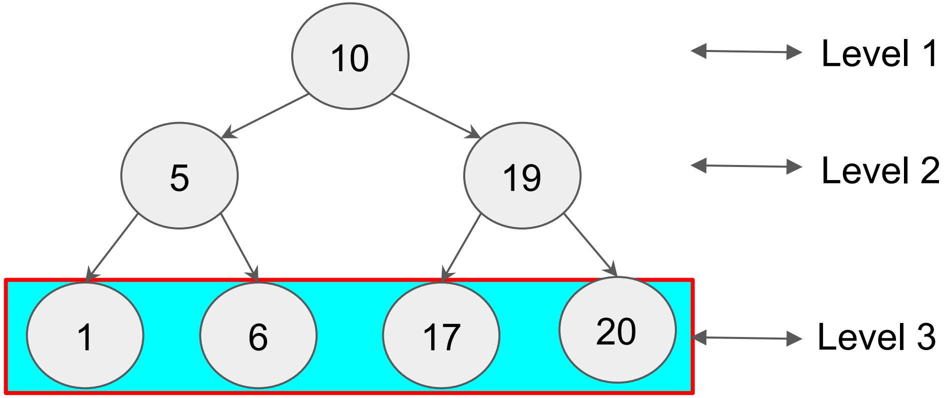
Key location can be retrieved using PIR-by-index

(Chor et al. TOC ‘98)

Assume keys are integers and arranged in a BST

$K = \{1, 5, 6, 10, 17, 19, 20\}$

User



Level 3: Retrieve element at index 2 using PIR-by-index

17 = 17 (found it!)

Path from root to leaf is index of k in keyset K

Untrusted Server

This area originated as Private retrieval by keywords in 1998
(Chor et al. TOC '98)

Private Keyword retrieval can be performed by two stages:

Stage 1: Retrieve the key location

k

Give me the location for key k

i

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Stage 2: Perform PIR by index

Has (key, location) mapping

Give me the *i*-th value

0	v_0
1	v_1
2	v_2
…	…
n-1	v_{n-1}

Performance of BST-based PIR-by-keywords

Stage 1 + Stage 2

Network cost: $0 < \text{level} < \log(n)$

Request size: $\sum$ PIR-request-size(2^{level}) + PIR-request-size(n)

Response size: $\sum$ PIR-response-size(2^{level}) + PIR-response-size(n)

Number of round trips between user and server: **log(n) + 1**

Compute cost: $0 < \text{level} < \log(n)$

Time to compute response: $\sum$ PIR-compute-time(2^{level}) + PIR-compute-time(n)

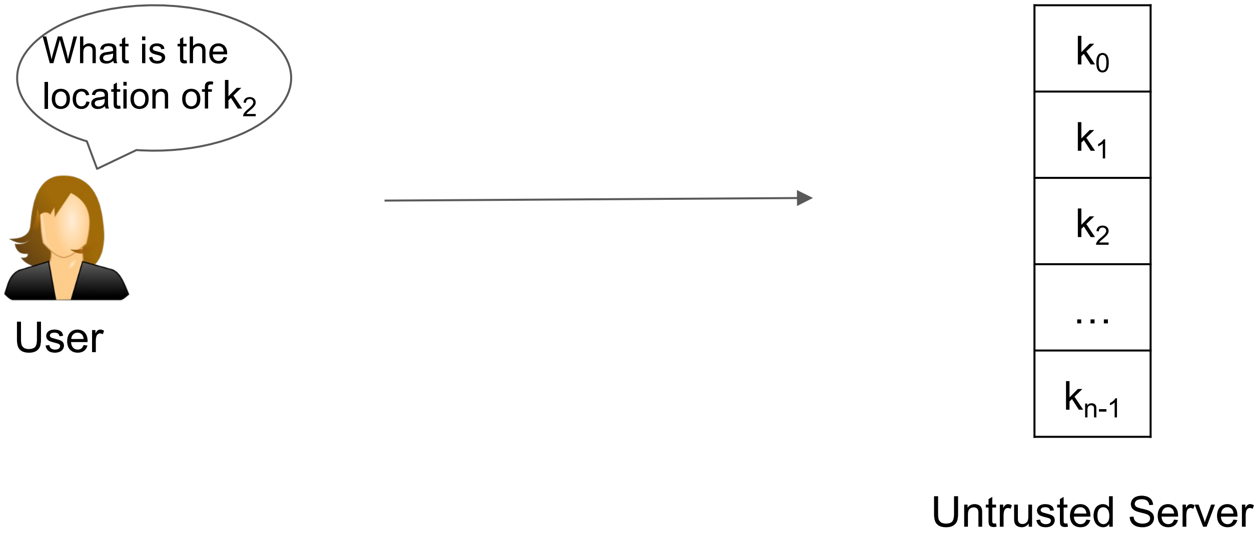
BST-based solution is also not database-updates friendly

- Client must know *n*, the total number of keys
- Server cannot insert / delete keys while a client is executing the $\log(n) + 1$ rounds

Current research on PIR-by-keywords is on reducing the number of round trips and dynamic keyset issues

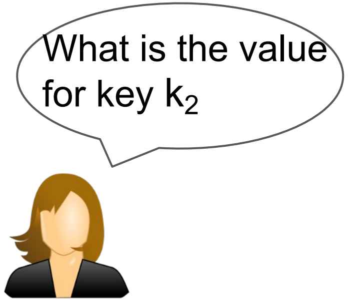
Overhead	High-level technique
Round trips	• Constant-weight equality operator (SEC '22) • Pantheon (tomorrow at H3 — 10:30 AM session)
Dynamic keyset	• Pantheon (tomorrow at H3 — 10:30 AM session)

Pantheon: A single round approach for PIR-by-keywords



- *Can we retrieve the location in single-round?*
- *Can we make the query independent of the number of keys (n)?*

Pantheon: A single round approach for PIR-by-keywords



User

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

- *Can we compose the two stages without involving the client in between?*

Pantheon: A single round approach for PIR-by-keywords

I want value for key $\mathbf{k_2}$

Enc($\mathbf{k_2}$)

Key	Value
k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

Pantheon: A single round approach for PIR-by-keywords

I want value for key $\mathbf{k_2}$

Enc($\mathbf{k_2}$)

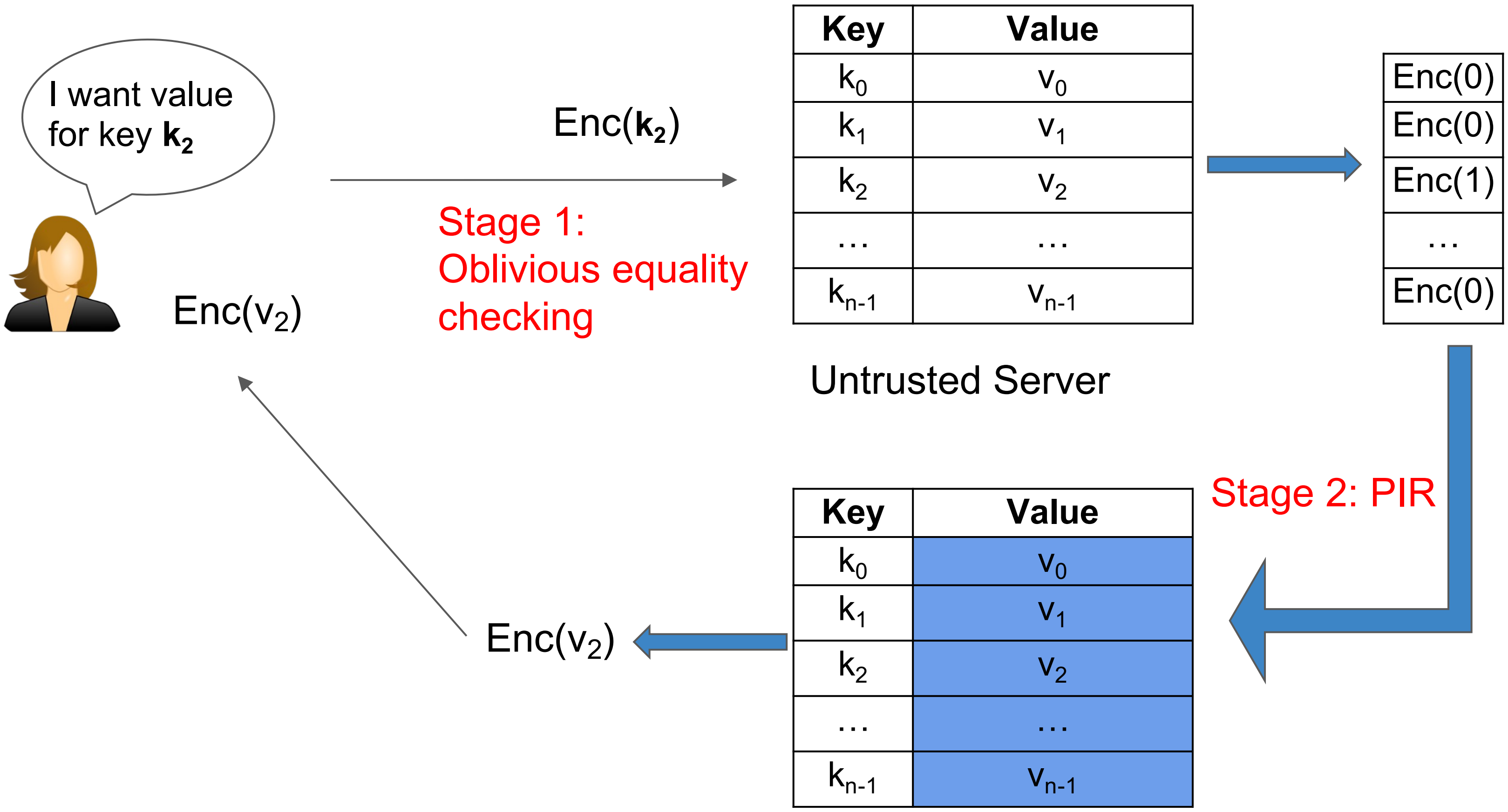
Stage 1: Oblivious equality checking

Key	Value
k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

Enc(0)
Enc(0)
Enc(1)
…
Enc(0)

Pantheon: A single round approach for PIR-by-keywords



Pantheon: A single round approach for PIR-by-keywords

I want value for key $\mathbf{k_2}$

$\mathbf{v_2}$

Enc($\mathbf{k_2}$)

Stage 1:
Oblivious equality checking

Key	Value
k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

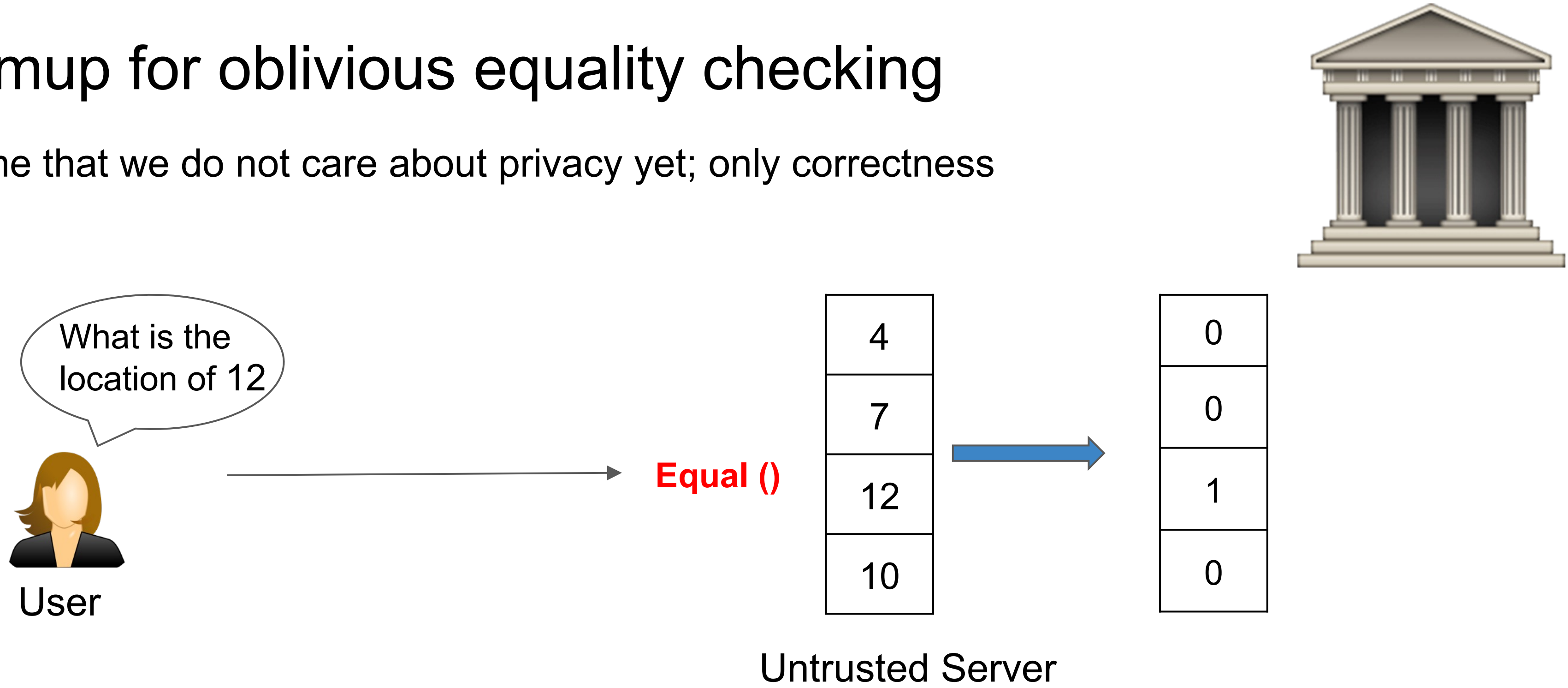
Enc(0)
Enc(0)
Enc(1)
…
Enc(0)

Stage 2: PIR

Key	Value
k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Warmup for oblivious equality checking

Assume that we do not care about privacy yet; only correctness



Warmup for oblivious equality checking

Assume that we do not care about privacy yet; only correctness

Step 1: Subtraction *Step 2: Binarization* *Step 3: Complement*

query		K			binarize ?		1's complement	
12		4		8		1		0
12	−	7	=	5	→	1	→	0
12		12		0		0		1
12		10		2		1		0

Homomorphic addition

$$\mathrm{Enc}(m_1) \times \mathrm{Enc}(m_2) = \mathrm{Enc}(m_1 + m_2)$$

Fermat’s little theorem

if p is a prime number and a is a number not divisible by p, then,

$$a^{(p-1)} \equiv 1 \pmod{p}$$

Example:

Let, p =17. Then for any $0 < a < 17$,

$a^{16} \% p = 1$

$2^{16} \% 17 = 65536 \% 17 = 1$

$3^{16} \% 17 = 43046721 \% 17 = 1$

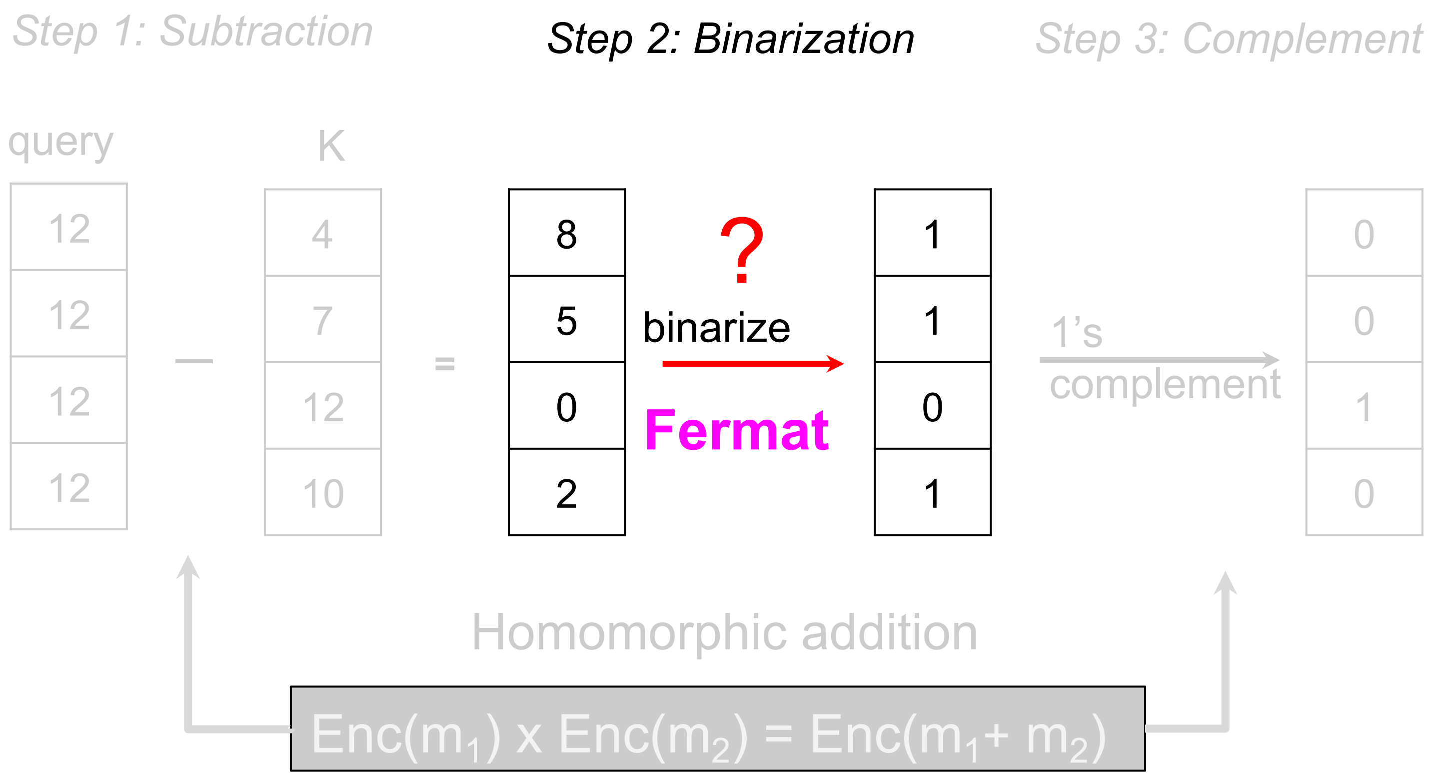
…………………………..

However, if a = 0, then $0^{16} \% p = 0$

Fermat’s little theorem enables distinction between zero and non-zero value!

Recall the warmup for oblivious equality checking

Assume that we do not care about privacy yet; only correctness



Pantheon: An efficient and scalable solution

For more details, please attend the paper presentation:

Wednesday 10:30—noon session (H3)

Current research on PIR-by-keywords is on reducing the number of round trips and dynamic keyset issues

Overhead	High-level technique
Round trips	• Constant-weight equality operator (SEC '22) • Pantheon (tomorrow at H3 — 10:30 AM session)
Dynamic keyset	• Pantheon (tomorrow at H3 — 10:30 AM session)

This tutorial is in two parts

Part 2: Retrieval by key

k_1	1
k_2	2
k_3	3
…	…
k_n	n

k

Client retrieves:

- v, if (k,v) at Server
- ∅, otherwise

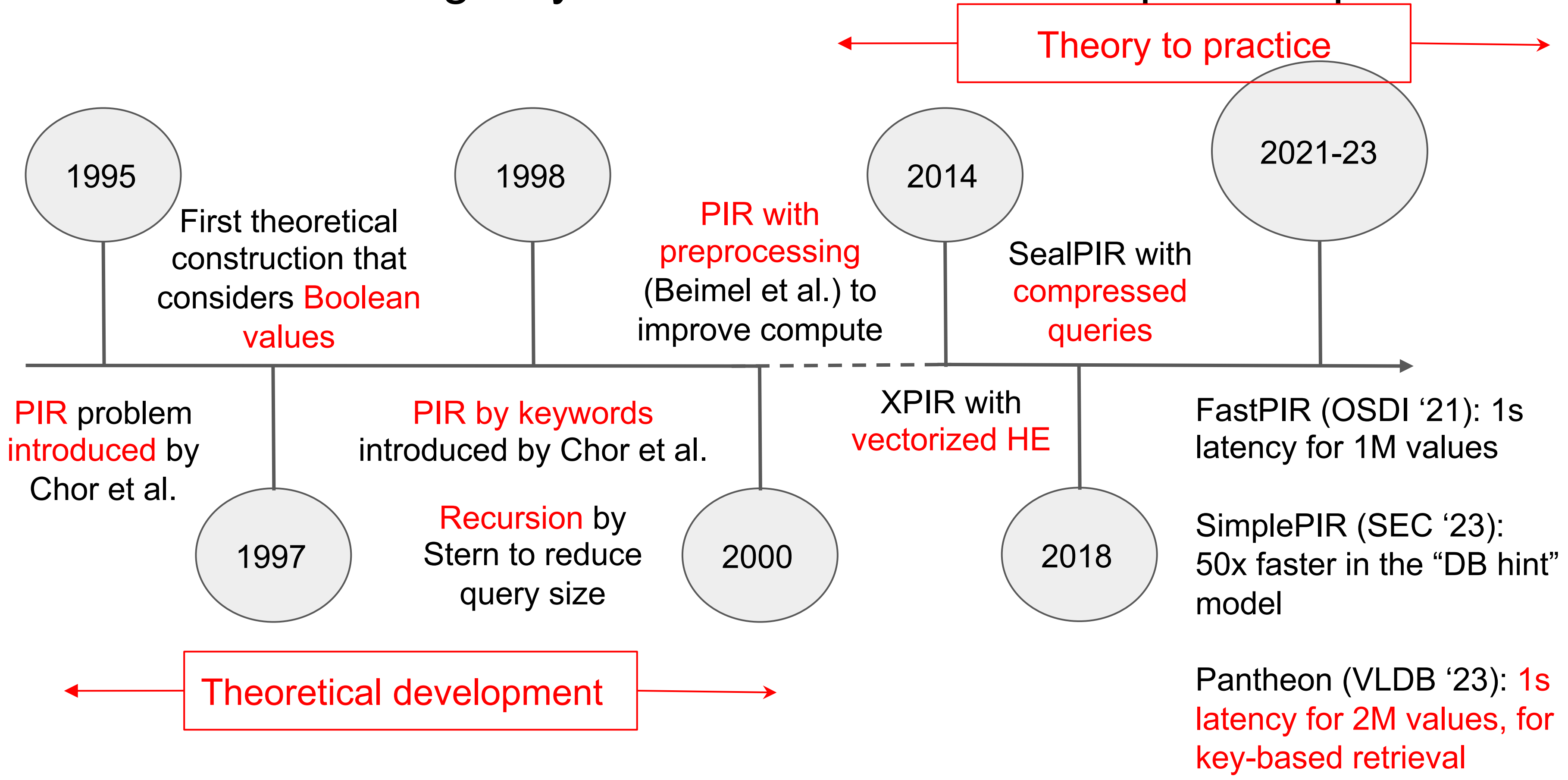
Give me value for key k

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Untrusted Server

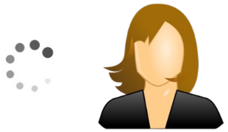
Part 2: How can the client privately retrieve the value corresponding to a given key?

We have come a long way — Private retrieval from public repositories



Looking ahead — Private retrieval over public repositories

But overheads still high



k

Latency ~ 1 second

Private GET for key k

Untrusted Server

k_0	v_0
k_1	v_1
k_2	v_2
…	…
k_{n-1}	v_{n-1}

Needs high compute resources

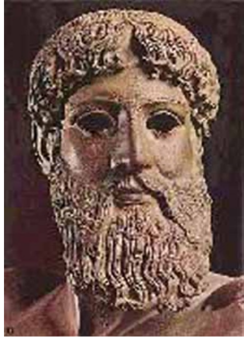
Looking ahead — Private retrieval over public repositories

Query interface is narrow

- PIR-by-location (Chor et al. FOCS ‘95)
- PIR-by-keywords (Chor et al. TOC ‘98)

- Private top-K queries?
  - *Retrieve price for 5 stocks similar to AAPL*
- Private range queries?
  - *Retrieve daily price of AAPL between a start and end date*
- Private aggregation queries?
  - *Calculate the average price of AAPL within a date range*

Coeus: Oblivious top-K ranking & retrieval (SOSP ‘21)



Search keyword:
“red apple”

Give me top-K matching documents

Document Provider (D)

doc1
doc2
doc3
doc4
...
...
...

Simple IR with ranking in one round of communication

"red apple"

tf-idf matrix

	apple	bat	red	....
Doc1	0.5	0.2	0	...
Doc2	0.8	0.1	0.1	...
Doc3	0	0	0.6	...
...	...	...	...	...
...	...	...	...	...

Simple IR with ranking in one round of communication

"red apple"

1
0
1
...

tf-idf matrix

	apple	bat	red	....
Doc1	0.5	0.2	0	...
Doc2	0.8	0.1	0.1	...
Doc3	0	0	0.6	...
...	...	...	...	...
...	...	...	...	...

Simple IR with ranking in one round of communication

matrix-vector multiplication

1
0
1
...

tf-idf matrix

	apple	bat	red	....
Doc1	0.5	0.2	0	...
Doc2	0.8	0.1	0.1	...
Doc3	0	0	0.6	...
...	...	...	...	...
...	...	...	...	...

Simple IR with ranking in one round of communication

matrix-vector multiplication

1
0
1
...

Score 1
Score 2
Score 3
...

tf-idf matrix

	apple	bat	red	....
Doc1	0.5	0.2	0	...
Doc2	0.8	0.1	0.1	...
Doc3	0	0	0.6	...
...	...	...	...	...
...	...	...	...	...

Simple IR with ranking in one round of communication

Server picks top-k scores

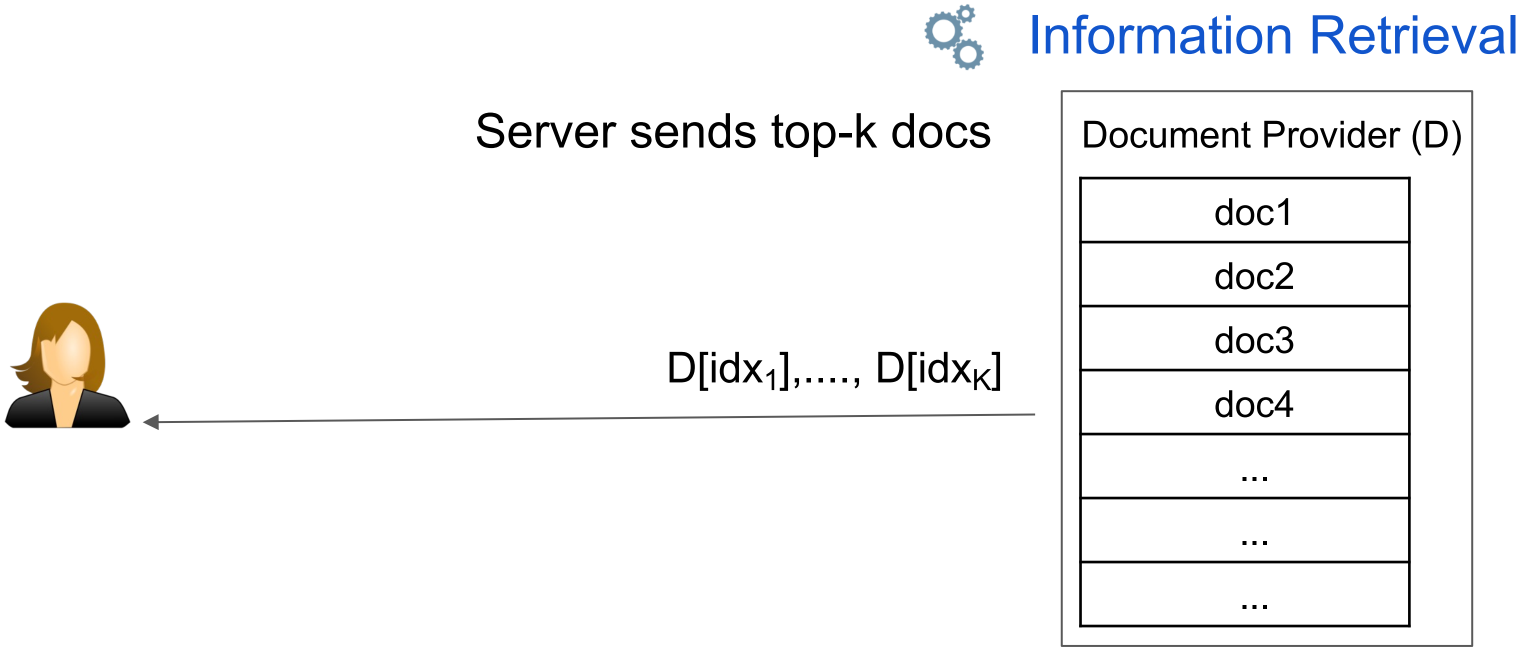
$idx_1, \ldots, idx_K$

Score 1
Score 2
Score 3
...

Document Provider (D)

doc1
doc2
doc3
doc4
...
...
...

Simple IR with ranking in one round of communication



Simple IR with ranking in one round of communication

Client reads relevant document

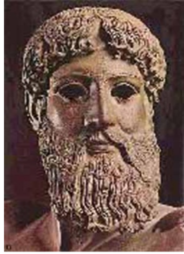
$D[idx^*]$

$D[idx_1], \ldots, D[idx_K]$

Document Provider (D)

doc1
doc2
doc3
doc4
...
...
...

Coeus: A novel 3 round protocol for *oblivious* top-K



- Ranks documents using scores computed against tf-idf matrix
- A new large-scale secure matrix-vector multiplication protocol
- Composes secure multiplication with PIR to retrieve documents
- End-to-end latency of 3.9 seconds for 5M documents in English Wikipedia

How can we expand the query interface beyond point queries?

- Private top-K queries?
  - *Retrieve price for 5 stocks similar to AAPL*
- Private range queries?
  - *Retrieve daily price of AAPL between a start and end date*
- Private aggregation queries?
  - *Calculate the average price of AAPL within a date range*

…

Coeus SOSP ‘21

Summary and takeaway points

- Private access over public data repositories is underserved
- This area derives from private information retrieval (PIR)
  - PIR-by-location, PIR-by-keywords
  - Applications of homomorphic encryption, secure dot-product
- Much research focuses on reducing overhead (compute, network) or improving suitability for dynamic databases
- An exciting area for future research
  - How can we further improve performance?
  - How can we expand to a full-fledged key-value database?

Thank You!

https://github.com/ishtiyaque/