PRIVATE INFORMATION RETRIEVAL:

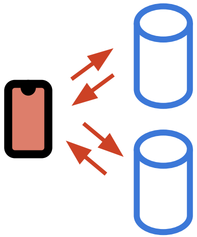
ARE WE CLOSE TO MAKE IT PRACTICAL?

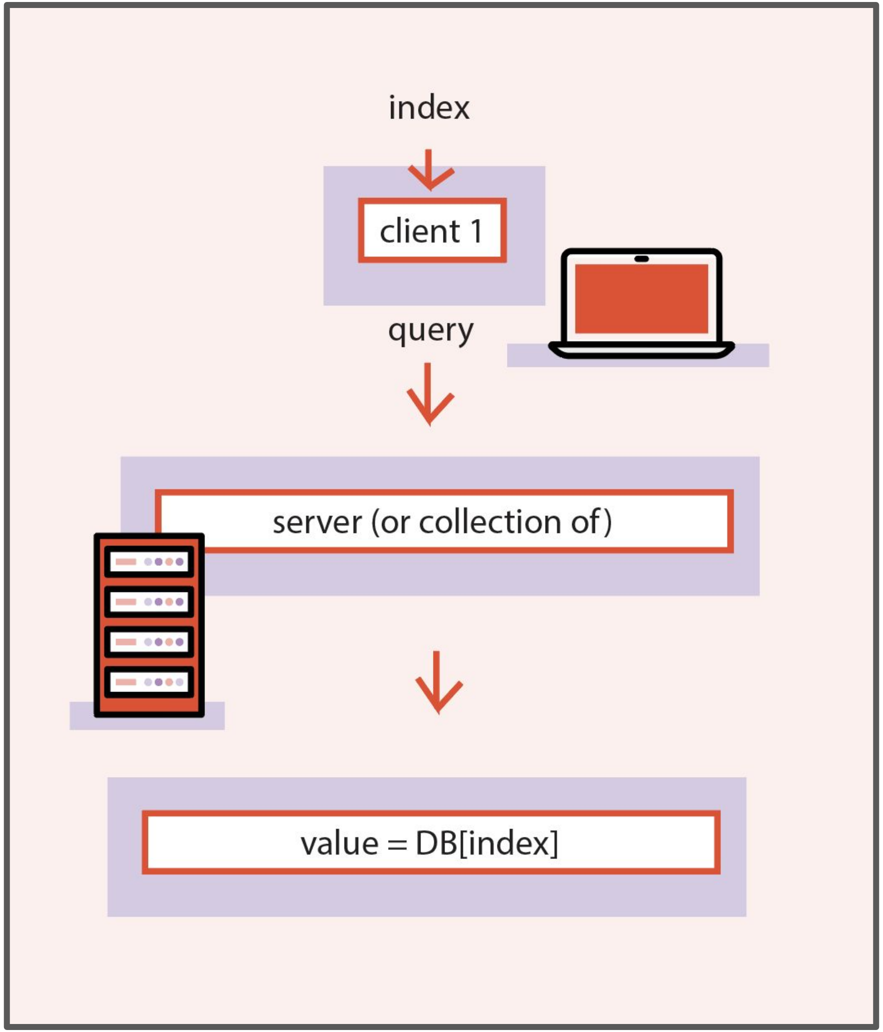
Summer School in Cryptography
Sofía Celi
cherenkov@riseup.net

Private Information Retrieval (PIR)

A Private Information Retrieval (PIR) scheme provides the ability for clients to **retrieve items from an online public (*) database of *m elements***, without **revealing anything about their queries** to the untrusted host server(s)

- Parties:
  a. Client(s)
  b. Server (one or multiple)
- Steps:
  - Query
  - Response
  - Parse





assuming the DB is public and it is index by digits

Private Information Retrieval (PIR)

Two types (sort of):

1. **Information-theoretic PIR:** client interacting with multiple non-colluding servers
2. **Computational-theoretic PIR**: client interacting with a single server, provides computational security based on cryptographic assumptions:
    a. Stateless PIR:
        - The client does not store any (pre)information in order to launch queries
        - The schemes (a bunch!) perform **worse than downloading the whole DB or they require computational overheads**

Private Information Retrieval (PIR)

Two types (sort of):

1. Information-theoretic PIR: client interacting with multiple non-colluding servers
2. **Computational-theoretic PIR**: client interacting with a single server, provides computational security based on cryptographic assumptions:
   a. Stateless PIR
   b. **Stateful PIR:** provides a “state” (or hint/digest) used as a “preprocessing” step amortised over *n* client queries

Private Information Retrieval (PIR)

Two types (sort of):

1. Information-theoretic PIR: client interacting with multiple non-colluding servers
2. **Computational-theoretic PIR**: client interacting with a single server, provides computational security based on cryptographic assumptions:
    a. Stateless PIR
    b. **Stateful PIR**

Idea: *encrypt the query* instead of *secret-sharing* it

Private Information Retrieval (PIR)

Limitations in **Computational-theoretic PIR**:

- Expensive pre-processing in terms of computation or communication
- High online-phase bandwidth consumption
- Lack of practical security parameters
- Lack of simple, open-source, available, verified implementations

Current look

- Very active research area
- Promising efficiency (computational/communicational/financial)
- Variety of applications

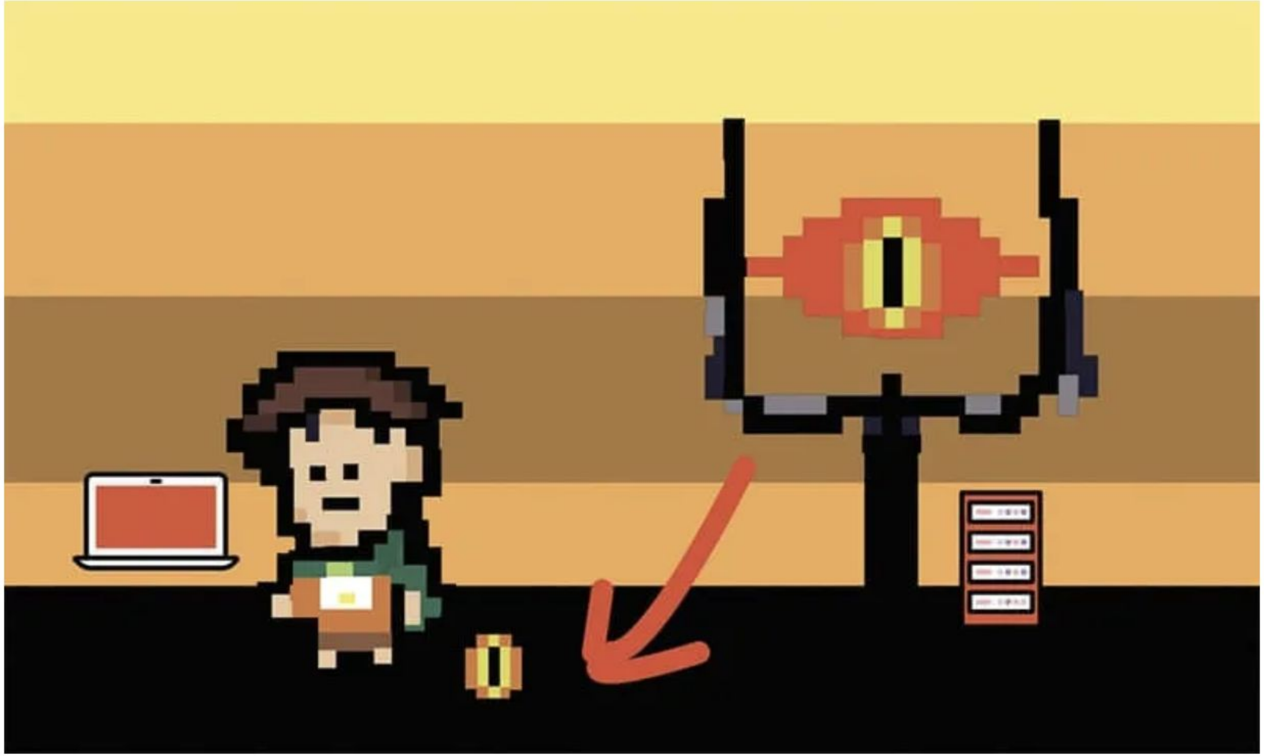
FrodoPIR

(but also *SimplePIR*)

https://eprint.iacr.org/2022/981

https://eprint.iacr.org/2022/949

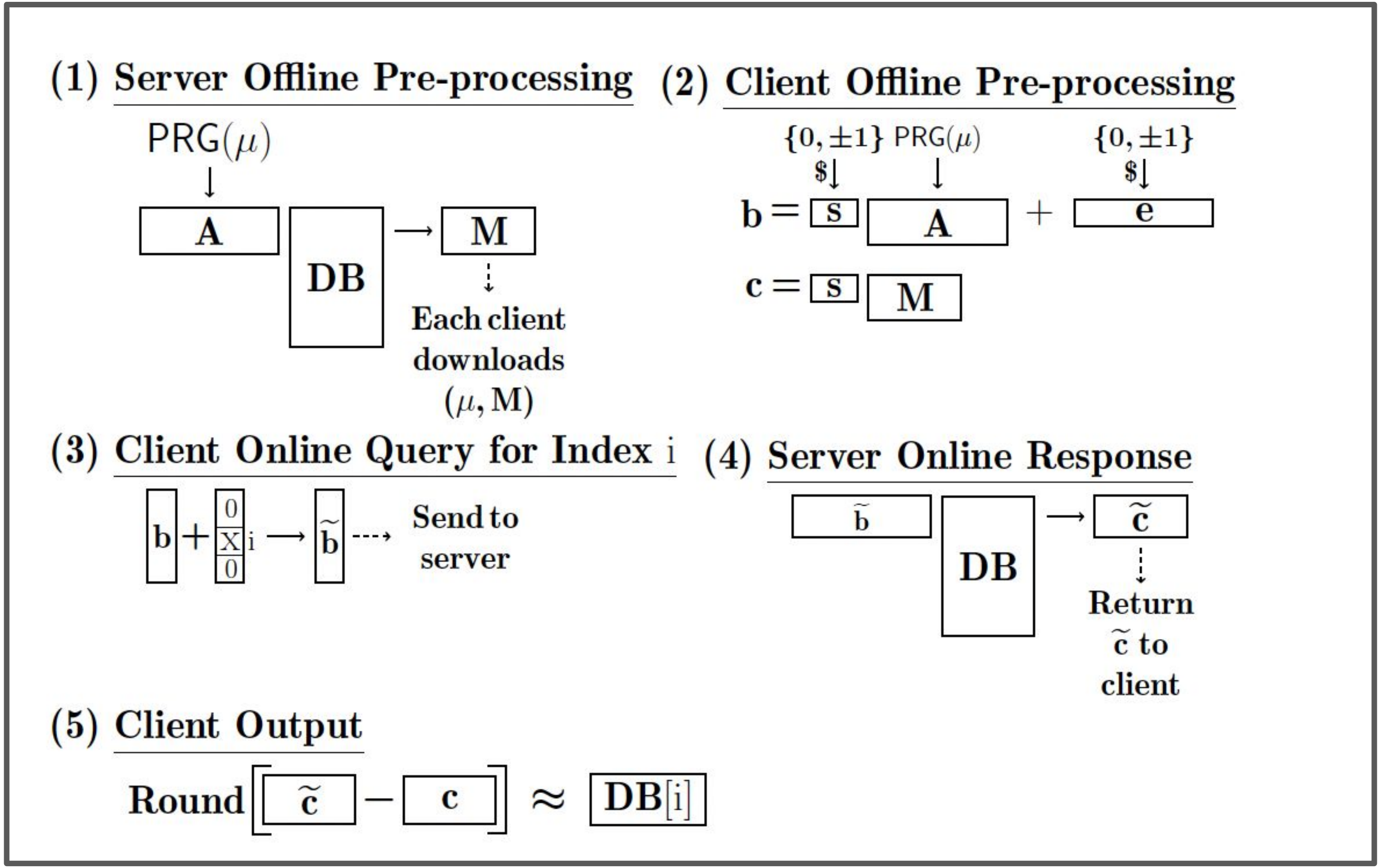
Announcing FrodoPIR!

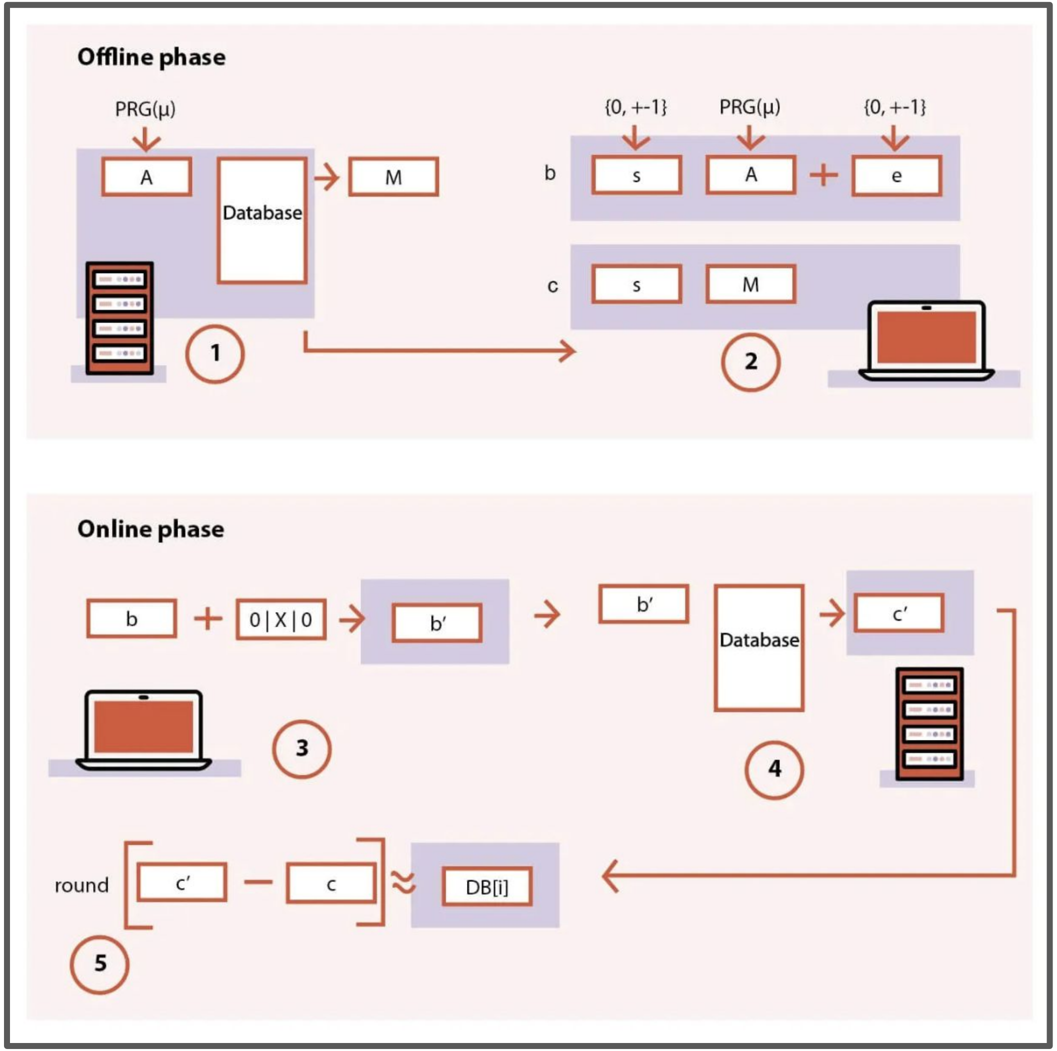


Just as the state of Sauron (its ring) moved to Frodo, we can move the mu and A to the client. The client then can then perform hidden queries to the server, just as Frodo remains hidden from Sauron.

Core ideas

- Built directly upon the learning with errors (LWE) problem *only* (similar to FrodoKEM)
  - Security relies on decisional LWE
  - Security is conservative (128 bits for 2^52 client queries): some parameters can be modified in order to make the scheme more efficient
- Highly configurable
  - Differences with *SimplePIR*: different pre-processing encoding, and the addition of a query pre-processing stage
- Tailored for efficiency and real-world applications


(1) Server Offline Pre-processing
PRG(μ)
A
DB
M
Each client downloads (μ, M)
(2) Client Offline Pre-processing
{0, ±1}
PRG(μ)
{0, ±1}
$
b = s A + e
c = s M
(3) Client Online Query for Index i
b + 0 X 0 i → b̃
Send to server
(4) Server Online Response
b̃
DB
c̃
Return c̃ to client
(5) Client Output
Round [c̃ − c] ≈ DB[i]



Offline phase
PRG(μ)
A
Database
M
b
{0, +-1}
PRG(μ)
{0, +-1}
s
A
e
c
s
M
1
2
Online phase
b
0 | X | 0
b′
b′
Database
c′
3
4
round
c′
c
DB[i]
5

Notation

- DB is an array of *m* elements, each made up of *w* bits.
- Each entry is associated with the index *i* that corresponds to its position in the array.
- There are *C* clients that will each launch a maximum of *c* queries against DB.
- LWE:
    a. *n* and *q* are the LWE dimension and modulus, respectively
    b. ρ is the number of bits packed into each entry of the DB matrix ($0 < \rho < q$)
    c. χ is the uniform ternary distribution over {-1, 0, 1}
    d. λ is the concrete security parameter.
- PRG(μ, n, m, q) denotes a pseudorandom generator that expands a seed

$$\mu \in \{0,1\}^{\lambda} \in \mathbb{Z}_q^{x \times y}$$

FrodoPIR (offline: server)

- **Server setup**: The server constructs their database containing *m* elements, and samples a seed $\mu \in \{0,1\}^\lambda$
- **Server pre-processing:** The server:

- Derives a matrix $\mathbf{A} \leftarrow \mathbf{PRG}(\mu, n, m, q)$

- Runs $\mathbf{D} \leftarrow \mathbf{parse}(DB, \rho)$

- Runs $\mathbf{M} \leftarrow \mathbf{A} \cdot \mathbf{D}$

- Publishes the pair $(\mu, \mathbf{M}) \in \{0,1\}^\lambda \times \mathbb{Z}_q^{n \times \omega}$

$\mathbf{A} \in \mathbb{Z}_q^{n \times m}$

$\mathbf{D} \in \mathbb{Z}_q^{n \times \omega}$

$\omega = w / \log(\rho)$

The "hint" is $\mathbf{M} \leftarrow \mathbf{A} \cdot \mathbf{D}$

FrodoPIR (offline: server)

- **Setup**: The server constructs their database containing *m* elements, and samples a seed $\mu \in \{0,1\}^\lambda$
- **Pre-processing:** The server:

- Derives a matrix $\mathbf{A} \leftarrow \mathbf{PRG}(\mu, n, m, q)$

- Runs $\mathbf{D} \leftarrow \mathbf{parse}(DB, \rho)$

- Runs $\mathbf{M} \leftarrow \mathbf{A} \cdot \mathbf{D}$

- Publishes the pair $(\mu, \mathbf{M}) \in \{0,1\}^\lambda \times \mathbb{Z}_q^{n \times \omega}$

$\mathbf{A} \in \mathbb{Z}_q^{n \times m}$

$\mathbf{D} \in \mathbb{Z}_q^{n \times \omega}$

$\omega = w / \log(\rho)$

$\mathbf{A}$ *remains secure even with multiple queries -2^52-.*

FrodoPIR (offline: client)

Pre-processing. Each client:

- Downloads $(\mu, \mathbf{M})$
- Derives $\mathbf{A} \leftarrow \mathbf{PRG}(\mu, n, m, q)$
- Samples *c* vectors: $j \in [c]$
    - $s_j \leftarrow \chi^n \quad e_j \leftarrow \chi^m$
- Computes:

$$b_j \leftarrow s_j^T \cdot \mathbf{A} + e_j^T \qquad \in \mathbb{Z}_q^m$$

$$c_j \leftarrow s_j^T \cdot \mathbf{M} \qquad \in \mathbb{Z}_q^\omega$$

- Stores the pairs as the set $X = (b_j, c_j)_{j \in [c]}$

Essentially, computes c sets of preprocessed query parameters (optional step).

FrodoPIR (online: client)

Query generation. For the index *i* that the client wishes to query, the client generates a vector (the all-zero vector except where $f_i[i] = q/\rho$):

$$f_i = (0, \cdots, 0, q/\rho, 0, \cdots, 0)$$

It then pops a pair (b, c) from internal state and computes:

$$b' = b + f_i$$

The client uses a single set of preprocessed query parameters to produce an "encrypted" query vector, which is sent to the server

FrodoPIR (online: server)

Response. The server receives *b'* from the client, and responds with:

$$c' \leftarrow b' \cdot \mathbf{D} \qquad \in \mathbb{Z}_q^{\omega}$$

Essentially, the server responds by multiplying the vector with their DB matrix

Post-processing. The client receives *c'*, and calculates:

$$v \leftarrow \lfloor (c' - c)_\rho \rceil$$

Essentially, the client get the value by "decrypting" using their pre-processed query parameters)

FrodoPIR Properties

Security: Indistinguishability of client queries. It assumes a semi-honest server that follows the protocol correctly and attempts to learn more based on the client queries they receive:

Server view: b' is distributed uniformly in $\mathbb{Z}_q^m$

under the assumption that decional-LWE is difficult to solve

- Regev encryption remains secure even when the same matrix *A* is used to encrypt many messages, provided that each ciphertext uses an independent secret vector *s* and error vector *e*

[83] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer.

FrodoPIR Properties

Efficiency. PIR schemes require a communication overhead smaller than the solution of having clients download *the entire server database*. In the stateful PIR case, it should hold when amortizing costs over the number of client queries.

Definition 5. (Efficiency) *For a single client launching c queries, a PIR scheme is* efficient *if the total client communication overhead is smaller than* $|\mathsf{DB}|$.

Therefore, for stateful schemes, the total client communication cost is calculated using the equation: $comms(\texttt{offline}) + c \cdot comms(\texttt{online})$.

FrodoPIR Properties

Efficiency.

Definition 5. (Efficiency) *For a single client launching c queries, a PIR scheme is* efficient *if the total client communication overhead is smaller than* $|\mathsf{DB}|$.

Therefore, for stateful schemes, the total client communication cost is calculated using the equation: $comms(\texttt{offline}) + c \cdot comms(\texttt{online})$.

	Offline	Online
Client upload	—	$m \log(q)$
Client download	$128 + n\omega \log(q)$	$\omega \log(q)$

FrodoPIR Properties

Efficiency.

	Offline	Online
Client upload	—	$m\log(q)$
Client download	$128 + n\omega\log(q)$	$\omega\log(q)$

$$128 + n\omega\log(q) + c\omega\log(q) < |DB|.$$

	Number of DB items ($\log(m)$)	16	17	18	19	20
Offline	Client download (KB)	5682.47	5682.47	5682.47	6313.07	6313.07
	Database preprocessing (s)	92.409	185.30	374.56	825.50	1679.8
	Client derive params (s)	0.5208	1.042	2.1	4.29	8.39
	Client query preprocessing (s)	0.134	0.265	0.532	1.058	2.111
Online	Client query (KB)	256	512	1024	2048	4096
	Server response (KB)	3.203	3.203	3.203	3.556	3.556
	Client query (ms)	0.0177	0.0454	0.0813	0.1565	0.3328
	Server response (ms)	45.74	89.57	179.3	397.06	779.75
	Client output (ms)	0.418	0.4182	0.416	0.4559	0.4627

https://github.com/brave-experiments/frodo-pir

Online Performance (ms) of Index-based FrodoPIR [29]				
	DB ($m \times w$)	Query	Response	Parsing
Macbook M1 Max	$2^{16} \times 1024$ B	0.0076956	5.2735	0.18083
	$2^{17} \times 1024$ B	0.017356	10.545	0.18544
	$2^{18} \times 1024$ B	0.055522	21.101	0.18061
	$2^{19} \times 1024$ B	0.1023	47.675	0.20108
	$2^{20} \times 1024$ B	0.21222	100.63	0.20483
EC2 "t2.t2xlarge"	$2^{16} \times 1024$ B	0.11887	29.482	0.34437
	$2^{17} \times 1024$ B	0.080101	50.585	0.34515
	$2^{18} \times 1024$ B	0.20374	118.54	0.3466
	$2^{19} \times 1024$ B	0.48432	263.83	0.3768
	$2^{20} \times 1024$ B	0.85748	537.28	0.37458
EC2 "c5.9xlarge"	$2^{20} \times 256$ B	1.2324	118.46	0.065281
	$2^{17} \times 30$ kB	0.036396	36.396	8.1519
	$2^{14} \times 100$ kB	0.0033412	637.81	26.599

Online Performance (ms) of Index-based FrodoPIR [29]				
	DB ($m \times w$)	**Query**	**Response**	**Parsing**
Macbook M1 Max	$2^{16} \times 1024$ B	0.0076956	5.2735	0.18083
	$2^{17} \times 1024$ B	0.017356	10.545	0.18544
	$2^{18} \times 1024$ B	0.055522	21.101	0.18061
	$2^{19} \times 1024$ B	0.1023	47.675	0.20108
	$2^{20} \times 1024$ B	0.21222	100.63	0.20483
EC2 "t2.t2xlarge"	$2^{16} \times 1024$ B	0.11887	29.482	0.34437
	$2^{17} \times 1024$ B	0.080101	50.585	0.34515
	$2^{18} \times 1024$ B	0.20374	118.54	0.3466
	$2^{19} \times 1024$ B	0.48432	263.83	0.3768
	$2^{20} \times 1024$ B	0.85748	537.28	0.37458
EC2 "c5.9xlarge"	$2^{20} \times 256$ B	1.2324	118.46	0.065281
	$2^{17} \times 30$ kB	0.036396	36.396	8.1519
	$2^{14} \times 100$ kB	0.0033412	637.81	26.599

What are the advantages?

1. It is simple: easy to explain, easy to push to production

2. LWE-based PIR schemes are simple to implement: they require no polynomial arithmetic or fast Fourier transforms

3. LWE-based PIR schemes do not require the server to store any extra per-client state. In contrast, many schemes based on Ring LWE rely on optimizations that require the server to store one “key-switching hint” for each client

4. LWE-based PIR schemes are faster and cheaper: the encryption scheme needs to be linearly (not fully) homomorphic, so we can use smaller and more efficient lattice parameters

But, is this enough?

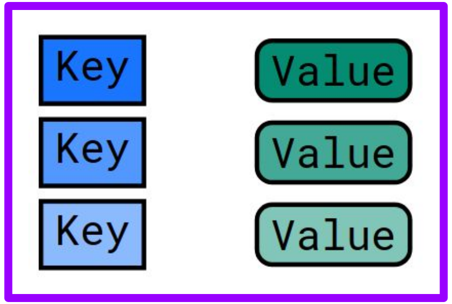
- Databases are not structured in this simple way
  - They are indexed by keywords
  - They are structured as JSON, Graphs, Excel spreadsheets
- The queries we are interested in are not simple:
  - Complex queries with AND/OR statements
  - Combination of database systems
  - Approximate nearest neighbor (ANN) elements
- Databases are constantly updated
- Is the security we assume enough?
  - What about malicious security?
  - What about private databases?

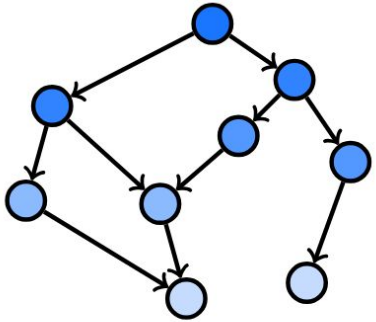
But, is this enough?

- Databases are not structured in this simple way
  - They are indexed by keywords
  - They are structured as JSON, Graphs, Excel spreadsheets
- The queries we are interested in are not simple:
  - Complex queries with AND/OR statements
  - Combination of database systems
  - Approximate nearest neighbor (ANN) elements
- Databases are constantly updated
- Is the security we assume enough?
  - What about malicious security?
  - What about private databases?

Not all systems are created equally

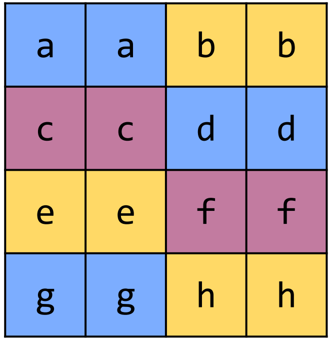
Real databases





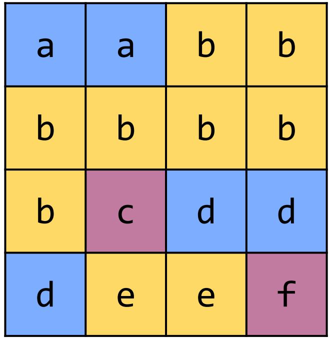
col_1	col_2	col_3
x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_3	z_3

Non-uniform data




```
{
    "firstName": "Joe",
    "lastName": "Jackson",
    "gender": "male",
    "age": 28,
    "address": {
        "streetAddress": "101",
        "city": "San Diego",
        "state": "CA"
    },
    "phoneNumbers": [
        { "type": "home", "number": "7349282382" }
    ]
}
```

- ❏ Data-specific privacy?
- ❏ Efficiency for multi-layer keys?
- ❏ Client storage?



Goals:

- Design PIR with real databases in mind.

- Security and performance modelling should take **database format** into account.

Which applications?

Some deployments / related technologies exist:

- Brave (compromised credential-checking, TBD)
- Blyss (https://github.com/blyssprivacy/sdk)
- Google (Device Enrollment)
- Microsoft (Password Monitor)

More complex use-cases (not deployed):

- Approximate nearest-neighbor: Brave News
- Private search: TipToe
- Oblivious document ranking: Coeus

Open questions:
- Build complex functions embedded directly into queries
- Basic PIR used as part of higher-level application

Updatable databases

Differing update-cycles depending on application

- ❏ Slower cadence: contact discovery, compromised credentials
- ❏ Faster cadence: safe browsing, recommendation systems (*)

Stateful PIR: require state regeneration with every update

Goals:

- More benchmarking of stateful PIR with incremental updates
- More efficient (and simpler®) stateless PIR

Configurability

Different performance metrics matter to different systems

- ❑ **Financial costs** may be more important than bandwidth for those without hardware
- ❑ **Server load** may be more important for CDNs, Google, etc.
- ❑ **Client load / bandwidth** for mobile devices

Question: Separate approaches for each criteria? Or support for simple re-parametrisation?

Important security properties

- Does a semi-honest, public DB satisfy all applications?
  - **Probably not**: compromised credentials, contact-checking...

- Private DB + semi-honest seems important
  - Privacy measures are *ad-hoc* (OPRF, masking)
  - Implications: sub-optimal rounds, not post-quantum...

- Authenticated/verifiable/malicious PIR exists, is this what we should be using everywhere?

Simplicity®

- FHE-based PIR is very complex
  - Libraries are hard to audit/verify
  - Non-standard security parameters
  - Low-level optimisations required for PIR

- AHE-based is simpler and configurable

Question: Do we want **widespread** or **centralised** deployments?

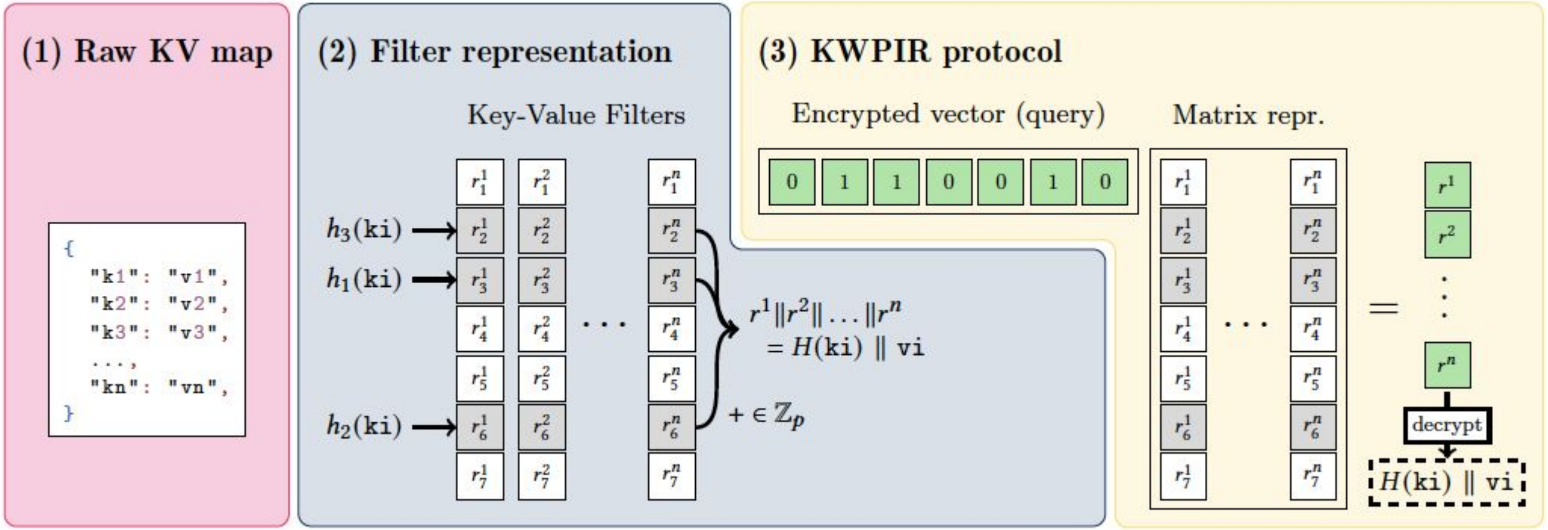
CHALAMET-PIR

(one solution)

https://eprint.iacr.org/2024/092

Core ideas

- Very simple (®) idea



Core ideas

1. Have a DB structured as a Key-Value (KV) map (size *m*, where each element *v* is indexed by a key *k*)
2. Convert this map into a filter (*F*) structure (think on a Bloom Filter) with a set of *k* hash functions and some false positive probability
   a. The filter has a function that allows to recover *v*: $fpt_\epsilon(v) \leftarrow F.check(k)$
   b. The filter is broken into *d* columns: interpret it as a matrix with *çm* (*) rows
3. Query for an element with a long vector where there are 1s on $h_i(k)$

(*) 1.08 ≤ ç ≤ 1.13, depending on the choice of *k* = *{3, 4}*

Basic construction

- Same ideas as previous in literature, but:
  - We leverage the usage of *Binary Fuse Filters*
    - Minimise the space and query overheads of key-value filters, while maintaining quick access times
    - Reconstruct using XOR
    - Divide the filter into many more segments
  - We can use *any* LWE-based PIR scheme

https://lib.rs/crates/haveibeenpwned

https://sts10.github.io/2023/01/11/playing-with-binary-fuse-filters.html

	DB ($m \times w$)	Query	Response	Parsing
Macbook M1 Max	$2^{16} \times 1024\,B$	0.010597	6.5508	0.22001
	$2^{17} \times 1024\,B$	0.038866	12.473	0.21894
	$2^{18} \times 1024\,B$	0.051996	24.452	0.21658
	$2^{19} \times 1024\,B$	0.14442	54.053	0.24204
	$2^{20} \times 1024\,B$	0.24049	116.89	0.24384
EC2 "t2.t2xlarge"	$2^{16} \times 1024\,B$	0.050048	37.830	0.47251
	$2^{17} \times 1024\,B$	0.1787	74.733	0.47046
	$2^{18} \times 1024\,B$	0.19739	143.82	0.46782
	$2^{19} \times 1024\,B$	0.4219	319.82	0.50735
	$2^{20} \times 1024\,B$	0.8471	634.21	0.56381
EC2 "c5.9xlarge"	$2^{20} \times 256\,B$	1.3699	133.58	0.090116
	$2^{17} \times 30\,kB$	0.055415	1846.6	10.663
	$2^{14} \times 100\,kB$	0.0040465	760.64	35.485

Table 2: Online performance (milliseconds) of ChalametPIR (LWEPIR = FrodoPIR, $k = 3$). Response is a server operation, while Query and Parsing are run by the client.

	DB ($m \times w$)	Query	Response	Parsing
Macbook M1 Max	$2^{16} \times 1024$ B	0.010597	6.5508	0.22001
	$2^{17} \times 1024$ B	0.038866	12.473	0.21894
	$2^{18} \times 1024$ B	0.051996	24.452	0.21658
	$2^{19} \times 1024$ B	0.14442	54.053	0.24204
	$2^{20} \times 1024$ B	0.24049	116.89	0.24384
EC2 "t2.t2xlarge"	$2^{16} \times 1024$ B	0.050048	37.830	0.47251
	$2^{17} \times 1024$ B	0.1787	74.733	0.47046
	$2^{18} \times 1024$ B	0.19739	143.82	0.46782
	$2^{19} \times 1024$ B	0.4219	319.82	0.50735
	$2^{20} \times 1024$ B	0.8471	634.21	0.56381
EC2 "c5.9xlarge"	$2^{20} \times 256$ B	1.3699	133.58	0.090116
	$2^{17} \times 30$ kB	0.055415	1846.6	10.663
	$2^{14} \times 100$ kB	0.0040465	760.64	35.485

Table 2: Online performance (milliseconds) of ChalametPIR (LWEPIR = FrodoPIR, $k = 3$). Response is a server operation, while Query and Parsing are run by the client.

Online Performance (ms) of Index-based FrodoPIR [29]				
	DB ($m \times w$)	Query	Response	Parsing
Macbook M1 Max	$2^{16} \times 1024$ B	0.0076956	5.2735	0.18083
	$2^{17} \times 1024$ B	0.017356	10.545	0.18544
	$2^{18} \times 1024$ B	0.055522	21.101	0.18061
	$2^{19} \times 1024$ B	0.1023	47.675	0.20108
	$2^{20} \times 1024$ B	0.21222	100.63	0.20483
EC2 "t2.t2xlarge"	$2^{16} \times 1024$ B	0.11887	29.482	0.34437
	$2^{17} \times 1024$ B	0.080101	50.585	0.34515
	$2^{18} \times 1024$ B	0.20374	118.54	0.3466
	$2^{19} \times 1024$ B	0.48432	263.83	0.3768
	$2^{20} \times 1024$ B	0.85748	537.28	0.37458
EC2 "c5.9xlarge"	$2^{20} \times 256$ B	1.2324	118.46	0.065281
	$2^{17} \times 30$ kB	0.036396	36.396	8.1519
	$2^{14} \times 100$ kB	0.0033412	637.81	26.599

Properties

- **Security:** Same as FrodoPIR (LWE-based), but:
  - We allow for false-positives, as we assume a public database. What impact does this have?
  - We provide a random value in case of non-inclusion -> leakage impact
- **Efficiency:** Same as FrodoPIR (LWE-based), but:
  - Blow-up due to filter: ç
- Is it sufficient?
  - Assumes the same length of elements

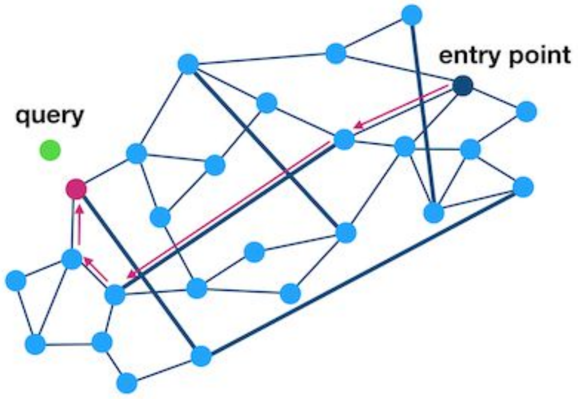
WHAT ELSE?

Upcoming solutions

- PIR for k-ANN
  - Real applications to search engines and recommendation systems
  - State-of-the-art k-ANN algorithms
- Integrate new security properties:
  - Expand to symmetric model
  - Expand to malicious security
- Deal with complex queries and complex databases
  - Not all databases are created equally:
    - Brave News
    - Brave Search
    - Brave CT

Upcoming solutions

- PIR for k-ANN
  - Real applications to search engines and recommendation systems
  - Graph-like structure that can be represented as a matrix
    - Dig into graph/matrix techniques to spectrally reason about them
  - State-of-the-art k-ANN algorithms



Future paper soon!, but:

- How to deal with updates in a graph/matrix structure?

Upcoming solutions

- PIR for k-ANN
- Integrate new security properties:
  - Expand to symmetric model
  - Expand to malicious security
- Deal with complex queries and complex databases
  - Not all databases are created equally:
    - Brave News
    - Brave Search
    - Brave CT
- A simple but needed SoK

My (sad) take

- We are very behind real databases-systems
- We are very behind state-of-the-art data structure/graph’s research
    - Why haven’t we look beyond Cuckoo filters and Merkle Trees?
- We are very behind actual deployment

But we are making progress!

Building steps

- Keyword-based PIR:
  - "Call Me By My Name: Simple, Practical Private Information Retrieval for Keyword Queries": https://eprint.iacr.org/2024/092
  - "Binary Fuse Filters: Fast and Smaller Than Xor Filters": https://arxiv.org/abs/2201.01174
- Security:
  - "Fully Malicious Authenticated PIR": https://eprint.iacr.org/2023/1804
  - "VeriSimplePIR: Verifiability in SimplePIR at No Online Cost for Honest Servers": https://eprint.iacr.org/2024/341
- Complex queries:
  - "Private Web Search with Tiptoe": https://eprint.iacr.org/2023/1438
  - "Coeus: A System for Oblivious Document Ranking and Retrieval": https://eprint.iacr.org/2022/154
- Updatability:
  - "Incremental Offline/Online PIR" https://www.cis.upenn.edu/~sga001/papers/incpir-sec22.pdf

So, you want to research on this?

- Expand the security model:
  - How does leakage impact it?
  - Is it attackable?
- Introduce 'updatable' techniques
- Look at other applications of DB:
  - Do we fulfil them?
- How do we deal with variable-length elements?
  - Is padding enough?
- Can we make it simple with the ring?
- Can we look at state-of-the-art data structures/graphs/matrix theory?

Thank you Henry Corrigan-Gibbs, Alex Davidson, Alexandra Henzinger, Stefano Tessaro, Eli Richarson for input and discussing all of this!

An announcement

PIR workshop at PETS: https://github.com/private-retrieval/wip

https://www.womenincryptography.com/

https://criptolatino.org/

THANK YOU!

@claucece

www.sofiaceli.com