Challenges in Low-communication PIR for Ephemeral Clients (and Potential Solutions…)

Rasoul Akhavan Mahdavi
University of Waterloo

WIP 2024

Acknowledgement/Outline

- Peer2PIR: Private Queries for IPFS
  - Miti Mazmudar, Shannon Veitch, Rasoul Akhavan Mahdavi

- ZipPIR: Low-communication PIR by compressing LWE ciphertexts
  - Rasoul Akhavan Mahdavi, Abdulrahman Diaa, Florian Kerschbaum

The Problem

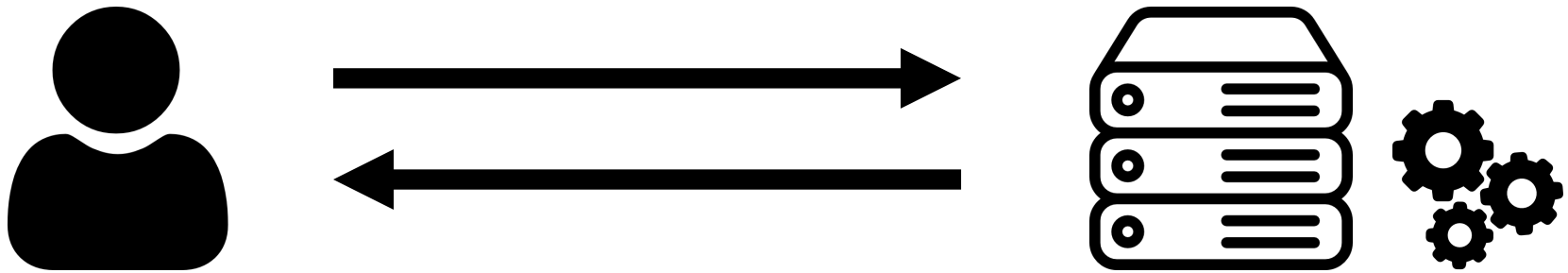
The Problem

PIR in papers

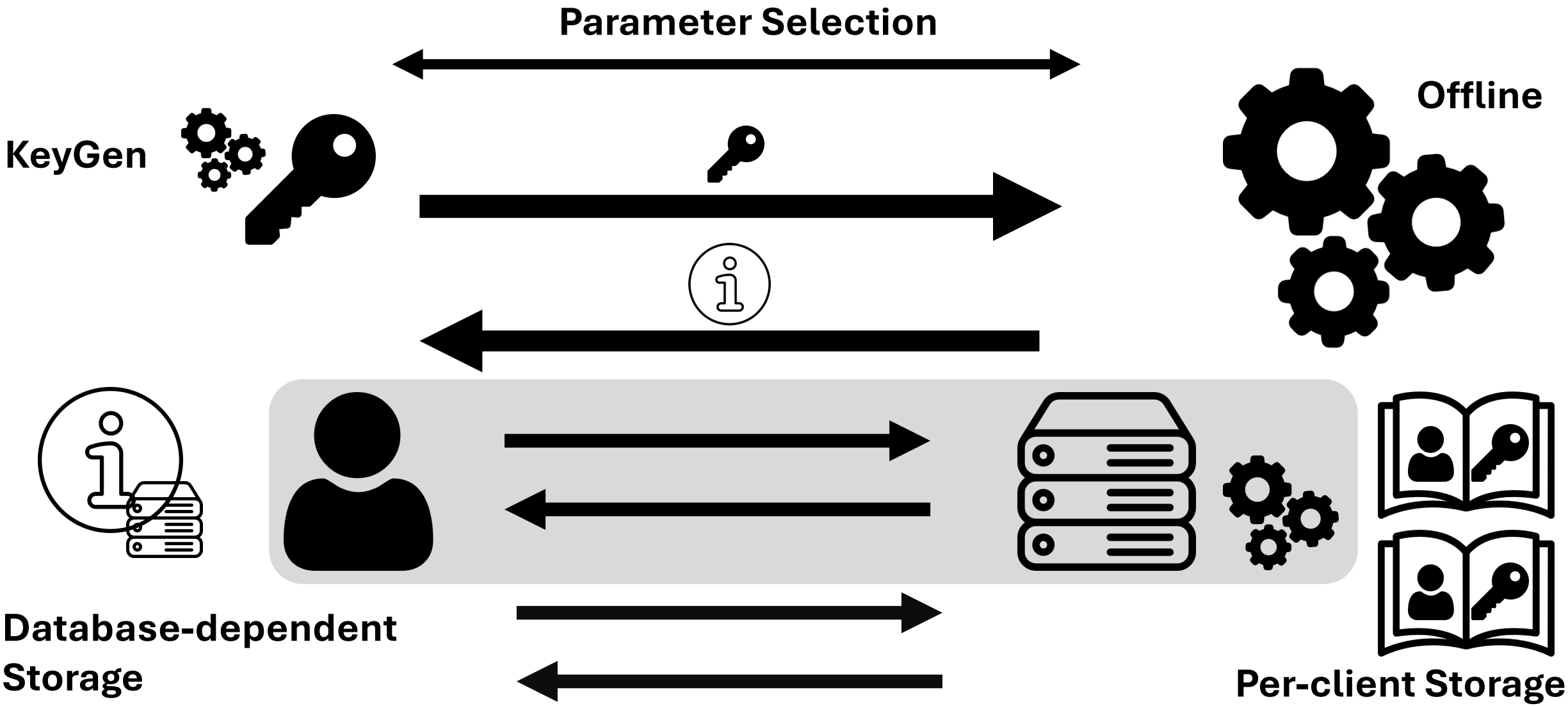
PIR in practice

- Unmentioned overhead
- Hidden assumptions
- Unrealistic security

What we want



What it actually is...

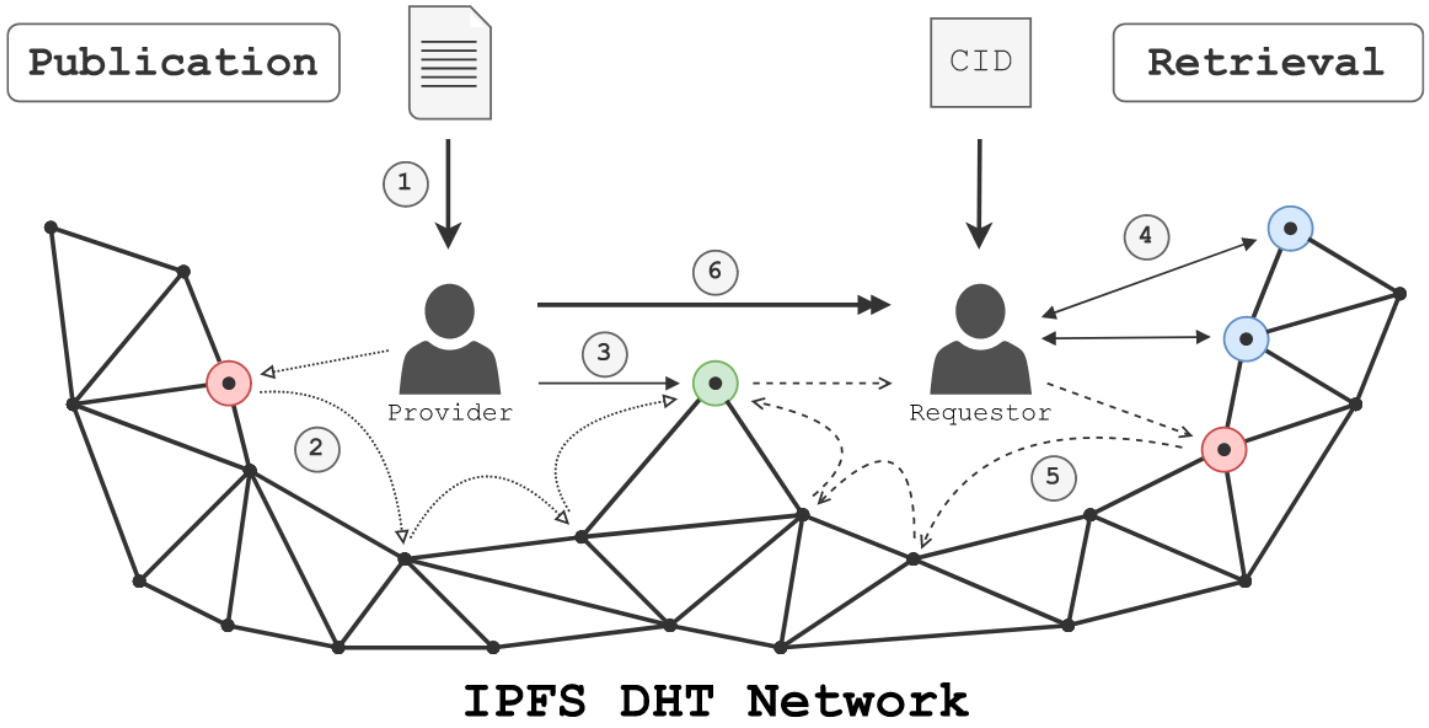


Metrics are also vague

- Only online runtime
- Throughput
  - Does not capture communication costs
- Communication costs
  - Assumed to be amortized
- Storage costs are not reasonable
  - Particularly in deployment
  - Per-client storage
  - Resource constrained clients
- Response size not considered

The PIR setup of IPFS

- IPFS is a distributed file system
- Query different nodes to access file



The PIR setup of IPFS

- IPFS is a distributed file system
- Query different nodes to access file
- Queries performed for routing to file, then fetching file

The PIR setup of IPFS

- IPFS is a distributed file system
- Query different nodes to access file
- Queries performed for routing to file, then fetching file
- Client queries a server that
  - it has never seen → no keys
  - will never see again → no future queries/amortization

The PIR setup of IPFS

- IPFS is a distributed file system
- Query different nodes to access file
- Queries performed for routing to file, then fetching file
- Client queries a server that
  - it has never seen → no keys
  - will never see again → no future queries/amortization
- Nodes have different databases

The PIR setup of IPFS

- IPFS is a distributed file system
- Query different nodes to access file
- Queries performed for routing to file, then fetching file
- Client queries a server that
  - it has never seen → no keys
  - will never see again → no future queries/amortization
- Nodes have different databases
- Single-round, parallelizable, changing database, query by keywords, …

Approaches to PIR

- Sublinear-time approaches

- Linear-time approaches

Approaches to PIR

- Sublinear-time approaches
  - Global preprocessing → ideal solution but impractical
    - Unkeyed Doubly efficient PIR
  - Client-specific preprocessing → requires “subscription”
    - Keyed Doubly efficient PIR, Piano
- Linear-time approaches

Approaches to PIR

- Sublinear-time approaches
  - Global preprocessing → ideal solution but impractical
    - Unkeyed Doubly efficient PIR
  - Client-specific preprocessing → requires “subscription”
    - Keyed Doubly efficient PIR, Piano
- Linear-time approaches
  - Client uploads keys, makes many queries
    - Large keys: SealPIR, MulPIR, OnionPIR, Spiral, FastPIR
    - Small keys: HintlessPIR, WhisPIR, YPIR
  - Client download database-dependent hint → no amortization
    - Global hint: SimplePIR, DoublePIR, FrodoPIR
    - Client-specific hint: …

Approaches to PIR

- Sublinear-time approaches
  - ~~Global preprocessing → ideal solution but impractical~~
    - ~~Unkeyed Doubly efficient PIR~~
  - ~~Client-specific preprocessing → requires “subscription”~~
    - ~~Keyed Doubly efficient PIR, Piano~~
- Linear-time approaches
  - Client uploads keys, makes many queries
    - Large keys: SealPIR, MulPIR, OnionPIR, Spiral, FastPIR
    - Small keys: HintlessPIR, WhisPIR, YPIR
  - ~~Client download database-dependent hint → no amortization~~
    - ~~Global hint: SimplePIR, DoublePIR, FrodoPIR~~
    - ~~Client-specific hint: …~~

The Setup in IPFS

- Small database (< 256 rows, < 2KB per row)
- Medium-sized database (~ 4k rows, ~ 1MB per row)
- Large database (1k – 1M rows, 256 KB payloads)

The Setup in IPFS

- Small database (< 256 rows, < 2KB per row)
  - Compete with trivial download
  - Linear PIR is enough, low rate
  - Additive schemes with **reduced upload**
- Medium-sized database (~ 4k rows, ~ 1MB per row)

- Large database (1k – 1M rows, 256 KB payloads)

The Setup in IPFS

- Additive schemes with **reduced upload**
  - Martin Beck, Randomized Decryption (RD) Mode of Operation for Homomorphic Cryptography

$$Enc(x_i) = Enc(r_i) \; + \; (x_i - r_i)$$
$$\lambda \rightarrow R_i = Enc(r_i)$$

The Setup in IPFS

- Additive schemes with **reduced upload**
  - Martin Beck, Randomized Decryption (RD) Mode of Operation for Homomorphic Cryptography

$$Enc(x_i) = Enc(r_i) \; + \; (x_i - r_i)$$
$$\lambda \rightarrow R_i = Enc(r_i)$$

$(\lambda, x_i - r_i)$ instead of $Enc(x_i)$

The Setup in IPFS

- Small database (< 256 rows, < 2KB per row)
  - Compete with trivial download
  - Linear PIR is enough, low rate
  - Additive schemes with reduced upload
- Medium-sized database (~ 4k rows, ~ 1MB per row)
  - PIR-with-keys is possible
  - Adaption/simplification to existing protocols
- Large database (1k – 1M rows, 256 KB payloads)

The Setup in IPFS

- Small database (< 256 rows, < 2KB per row)
  - Compete with trivial download
  - Linear PIR is enough, low rate
  - Additive schemes with reduced upload
- Medium-sized database (~ 4k rows, ~ 1MB per row)
  - PIR-with-keys is possible
  - Adaption/simplification to existing protocols
- Large database (1k – 1M rows, 256 KB payloads)
  - PIR-with-keys is a good solution
  - Keys are as big as response

Summary of evaluation

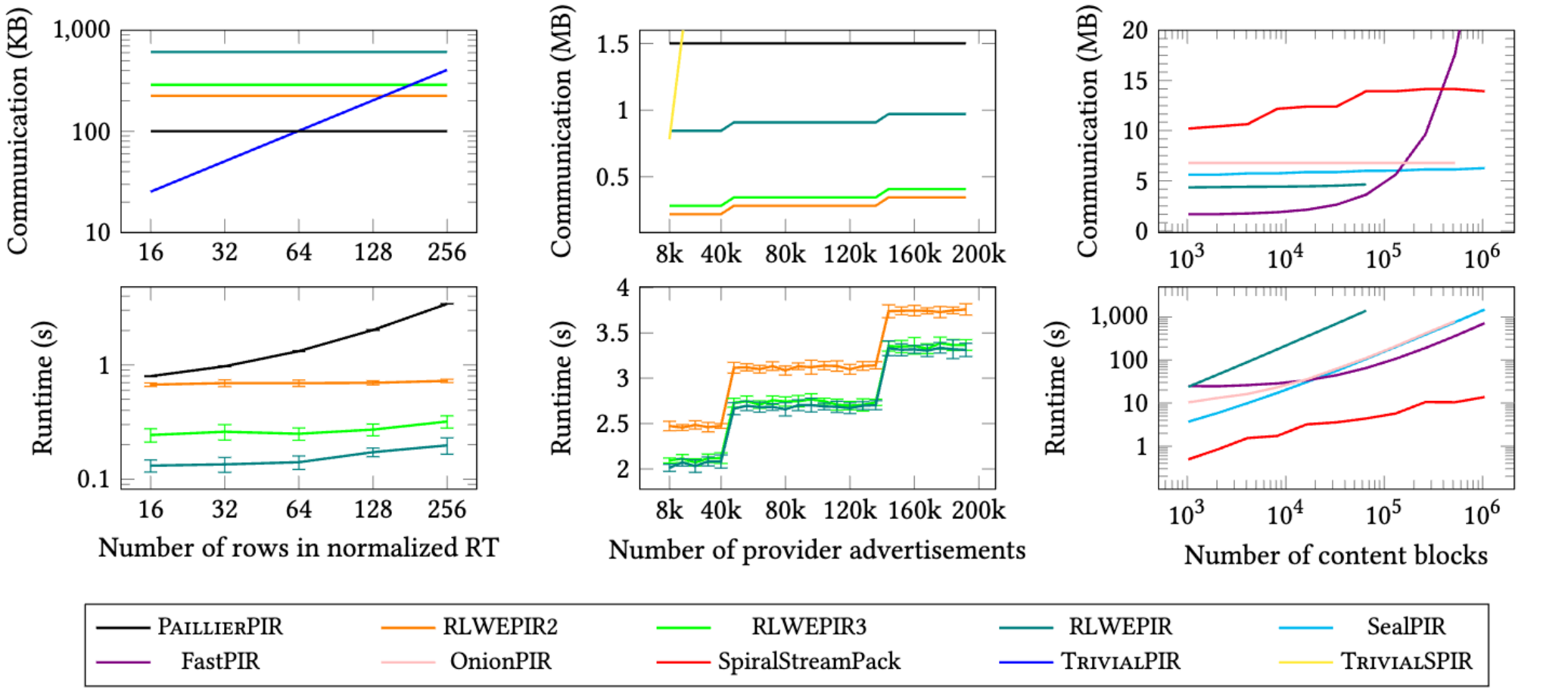


Figure 1: Communication costs (top graphs) and computation costs (bottom graphs) for private peer routing (left graphs), private provider advertisements (middle graphs), and private content retrieval (right graphs).

Conclusion

- Linear-time PIR is (for now) better in practice
- PIR-with-keys are (currently) best for ephemeral clients
  - Better for large responses/large databases
  - Possible for small responses/medium size databases
- Gap in literature for small databases
- Require low-communication solutions for few queries

Can we do better? (WIP)

Low communication for ephemeral clients

ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts

procedure LWEENCRYPT($\mathtt{sk}, \mu$)

Sample $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n$ and $e \leftarrow \chi$

$b = \sum_{i \in [n]} \mathbf{a}[i] \cdot \mathtt{sk}[i] + \Delta \cdot \mu + e \mod q$

return $\mathrm{ct} = (\mathbf{a}, b)$

procedure LWEDECRYPT($\mathtt{sk}, \mathrm{ct} = (\mathbf{a}, b)$)

$\mu^* = \left(b - \sum_{i \in [n]} \mathbf{a}[i] \cdot \mathtt{sk}[i]\right) \mod q$

$\mu' = \lfloor \mu^* / \Delta \rceil$

return μ'

- Partially decrypt LWE ciphertexts using **Enc(sk[i])**
- Use additive scheme, e.g., **Paillier, ECC**
- Removes dependence on n

ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts

Parameters	LWE $(n, \log_2 q)$ (630, 64)	(742, 64)	(870, 64)	(1305, 11)
Compression Time	9.7 ms	11.0 ms	12.9 ms	16.6 ms
Compressed Ciphertext	768 B	768 B	768 B	768 B
Uncompressed Ciphertext	5.05 KB	5.94 KB	6.97 KB	1.80 KB
Size Reduction	**84.78 %**	**87.08 %**	**88.98%**	**57.23%**

$(n, \log_2 q)$	(630,64)	(742,64)	(870,64)	(1305,11)
Unpacked Key	240 KB	284 KB	334 KB	501 KB
Packed Non-binary Key	22 KB	26 KB	30 KB	11 KB
Unpacking Time	14 ms	25 ms	74 ms	15 ms
Packed Binary Key	12 KB	14 KB	16 KB	7 KB
Unpacking Time	13 ms	12 ms	13 ms	15 ms

ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts
  - Optimized for batches → $1000\times$ smaller
  - Smaller compression key

ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts
  - Optimized for batches → $1000\times$ smaller
  - Smaller compression key
- Apply to SimplePIR hint

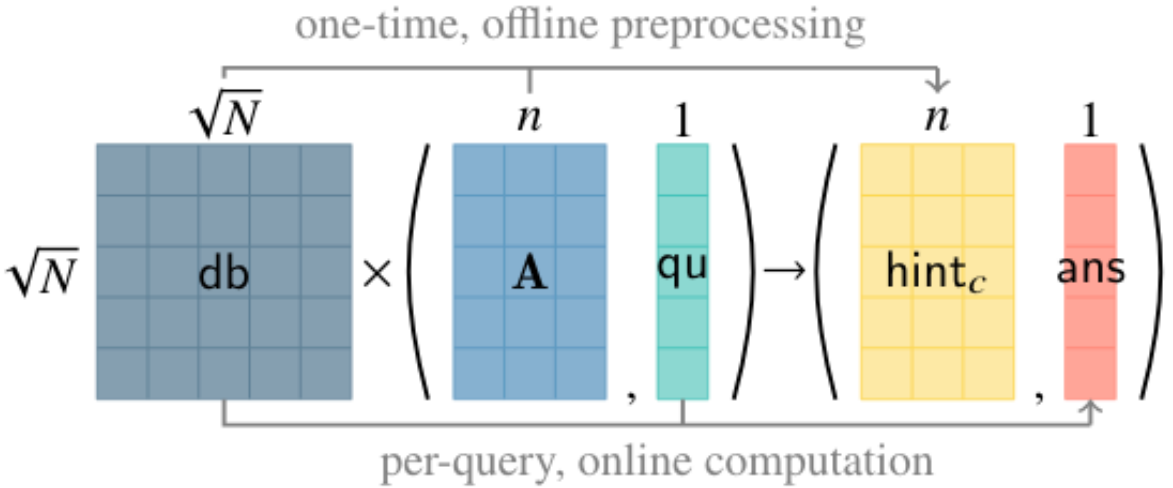


Figure 3: The server computation in SimplePIR. Each cell represents a $\mathbb{Z}_q$ element, and × denotes matrix multiplication. The server performs the bulk of its work in a one-time preprocessing step. Thereafter, the server can answer each client's query with a lightweight online phase.

ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts
  - Optimized for batches → $1000\times$ smaller
  - Smaller compression key
- Apply to SimplePIR hint
  - Hint size reduced 100-1000x
  - Client-specific
  - Compressed hint in response

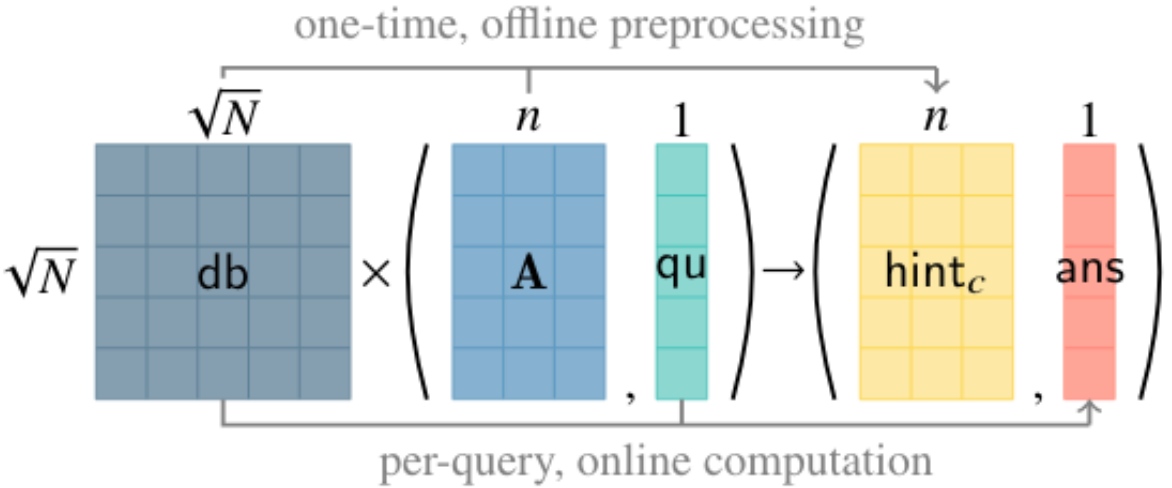
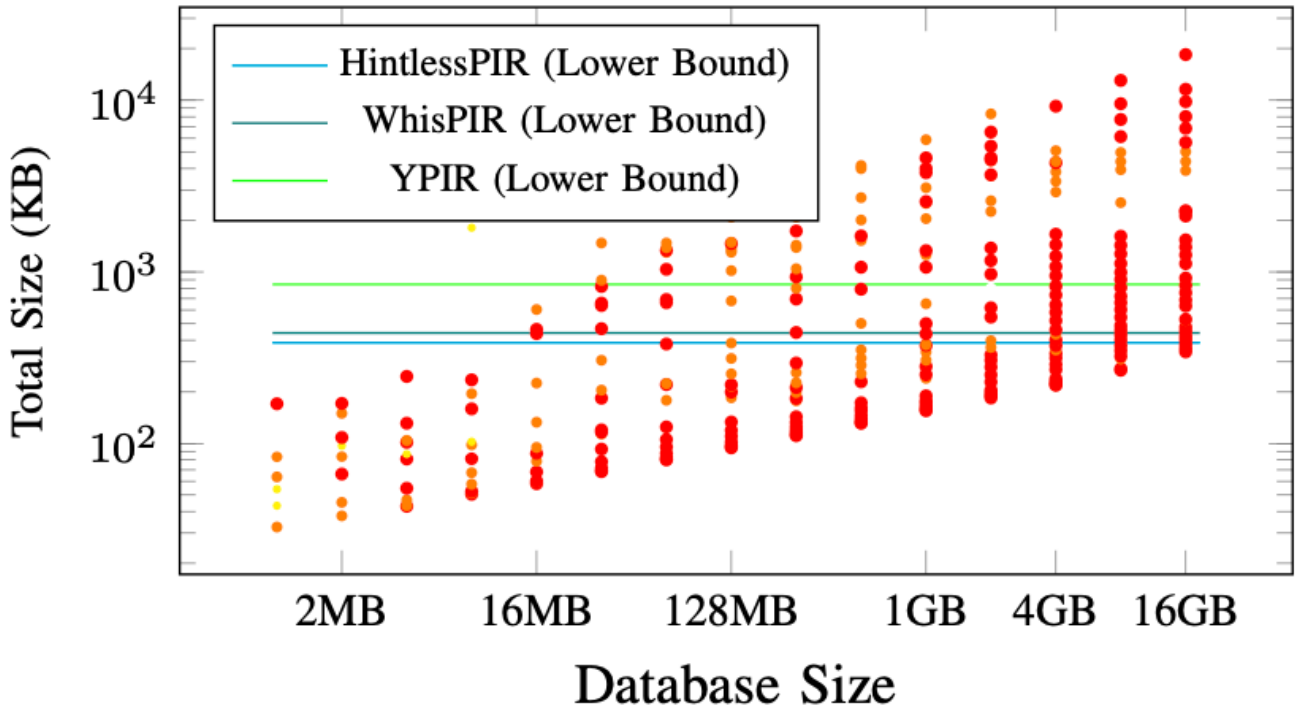


Figure 3: The server computation in SimplePIR. Each cell represents a $\mathbb{Z}_q$ element, and $\times$ denotes matrix multiplication. The server performs the bulk of its work in a one-time preprocessing step. Thereafter, the server can answer each client's query with a lightweight online phase.

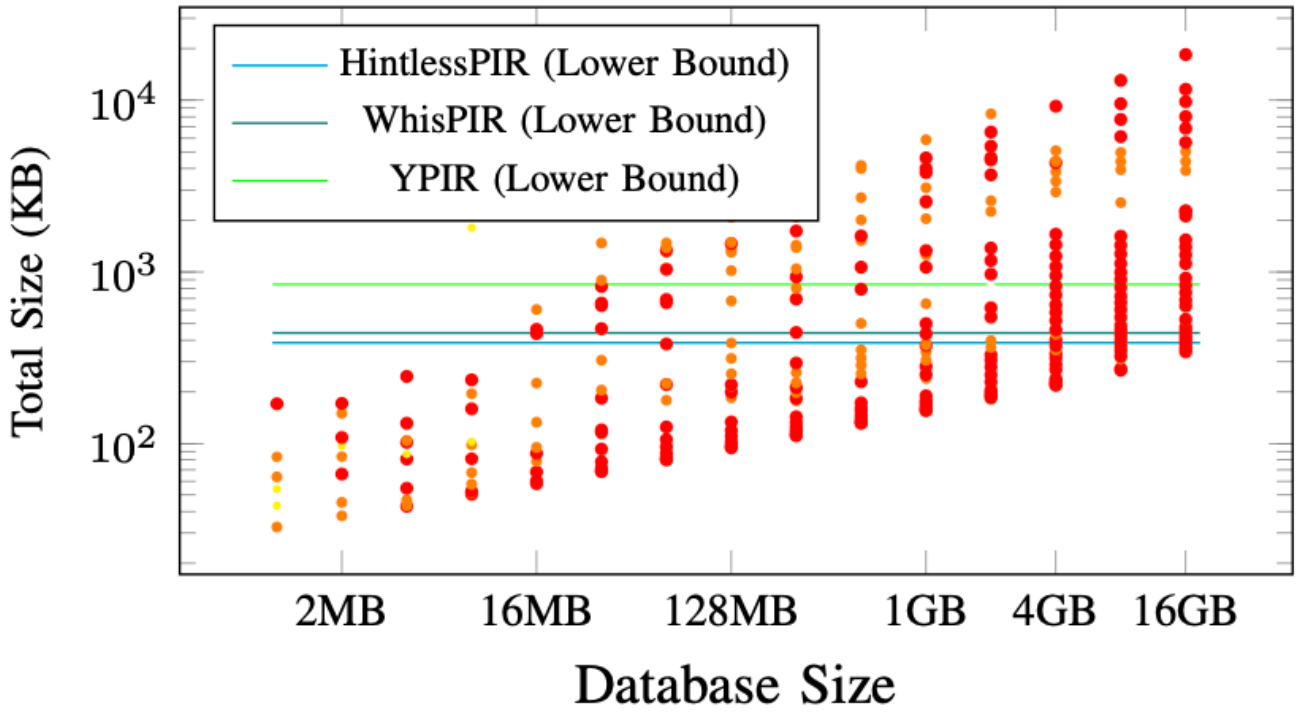
ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts
  - Optimized for batches → $1000\times$ smaller
  - Smaller compression key
- Apply to SimplePIR hint
  - Hint size reduced 100-1000x
  - Client-specific
  - Compressed hint in response
- Results?



ZipPIR: A low-communication PIR protocol

- Compressing LWE ciphertexts
  - Optimized for batches → $1000\times$ smaller
  - Smaller compression key
- Apply to SimplePIR hint
  - Hint size reduced 100-1000x
  - Client-specific
  - Compressed hint in response
- Results?
  - Slow…



Offline ZipPIR

- Reminder: Beck
  - $Enc(x_i) = Enc(r_i) + (x_i - r_i)$
  - $\lambda \rightarrow R_i = Enc(r_i)$
- Idea: Perform (linear) operations with $Enc(r_i)$

$$a1 * Enc(x_1) + a2 * Enc(x_2)$$
$$= a_1 * Enc(r_1) + a_2 * Enc(r_2) \quad \textbf{Offline}$$
$$+ a_1 * (x_1 - r_1) + a_2 * (x_2 - r_2) \quad \textbf{Online}$$

Offline ZipPIR

- Reminder: Beck
  - $Enc(x_i) = Enc(r_i) + (x_i - r_i)$
  - $\lambda \rightarrow R_i = Enc(r_i)$
- Idea: Perform (linear) operations with $Enc(r_i)$
- Problem:
  - Paillier → Modulus is client-specific
  - Exponential ElGamal, ECC → Can't decrypt random ct
  - Client-specific offline preprocessing
  - "Subscription model"

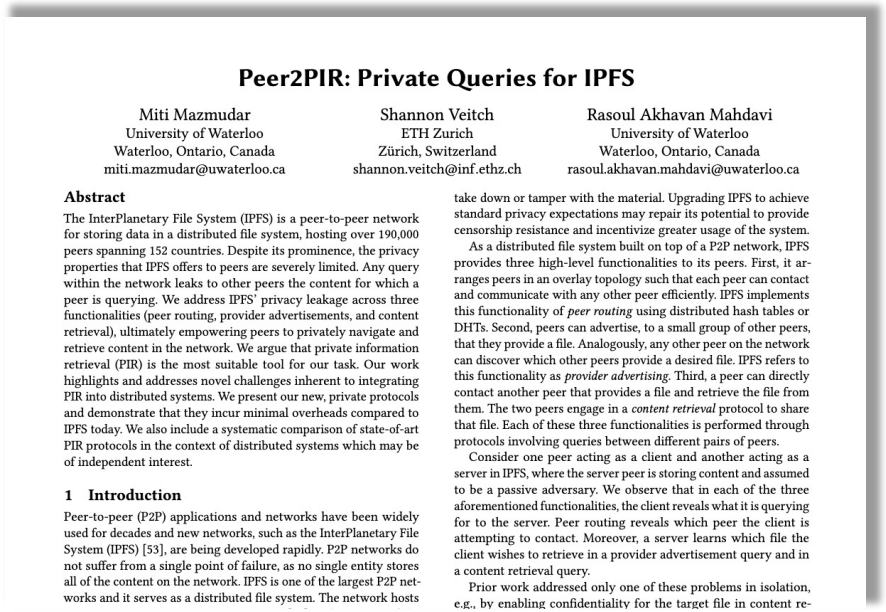
Approaches to PIR

- Sublinear-time approaches
  - Global preprocessing -> ideal solution but impractical
    - Unkeyed Doubly efficient PIR
  - Client-specific preprocessing
    - Keyed Doubly efficient PIR, Piano
- Linear-time approaches
  - Client uploads keys, makes many queries
    - Large keys: SealPIR, MulPIR, OnionPIR, Spiral, FastPIR
    - Small keys: HintlessPIR, WhisPIR, YPIR
  - Client download database-dependent hint
    - Global hint: SimplePIR, DoublePIR, FrodoPIR
    - Client-specific hint: **Offline ZipPIR**

Evaluation

- Advantages
  - Client-specific hint much smaller than global hint
  - Hint much smaller than keys
  - No client overhead
- Disadvantages
  - Client-specific

Protocol	SimplePIR	Piano	(Offline) ZipPIR
Preprocessing Communication	123 MB	1GB	1 KB
Hint size	123 MB	66 MB	~ 400KB
Online time	~ 130 ms	~ 10 ms	~ 260 ms



Peer2PIR: Private Queries for IPFS

Miti Mazmudar
University of Waterloo
Waterloo, Ontario, Canada
miti.mazmudar@uwaterloo.ca

Shannon Veitch
ETH Zurich
Zürich, Switzerland
shannon.veitch@inf.ethz.ch

Rasoul Akhavan Mahdavi
University of Waterloo
Waterloo, Ontario, Canada
rasoul.akhavan.mahdavi@uwaterloo.ca

Abstract

The InterPlanetary File System (IPFS) is a peer-to-peer network for storing data in a distributed file system, hosting over 190,000 peers spanning 152 countries. Despite its prominence, the privacy properties that IPFS offers to peers are severely limited. Any query within the network leaks to other peers the content for which a peer is querying. We address IPFS' privacy leakage across three functionalities (peer routing, provider advertisements, and content retrieval), ultimately empowering peers to privately navigate and retrieve content in the network. We argue that private information retrieval (PIR) is the most suitable tool for our task. Our work highlights and addresses novel challenges inherent to integrating PIR into distributed systems. We present our new, private protocols and demonstrate that they incur minimal overheads compared to IPFS today. We also include a systematic comparison of state-of-art PIR protocols in the context of distributed systems which may be of independent interest.

1 Introduction

Peer-to-peer (P2P) applications and networks have been widely used for decades and new networks, such as the InterPlanetary File System (IPFS) [53], are being developed rapidly. P2P networks do not suffer from a single point of failure, as no single entity stores all of the content on the network. IPFS is one of the largest P2P networks and it serves as a distributed file system. The network hosts

take down or tamper with the material. Upgrading IPFS to achieve standard privacy expectations may repair its potential to provide censorship resistance and incentivize greater usage of the system.

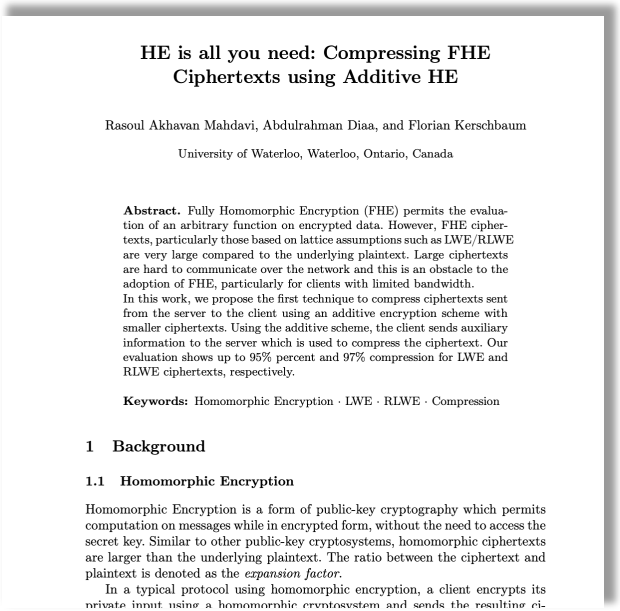
As a distributed file system built on top of a P2P network, IPFS provides three high-level functionalities to its peers. First, it arranges peers in an overlay topology such that each peer can contact and communicate with any other peer efficiently. IPFS implements this functionality of *peer routing* using distributed hash tables or DHTs. Second, peers can advertise, to a small group of other peers, that they provide a file. Analogously, any other peer on the network can discover which other peers provide a desired file. IPFS refers to this functionality as *provider advertising*. Third, a peer can directly contact another peer that provides a file and retrieve the file from them. The two peers engage in a *content retrieval* protocol to share that file. Each of these three functionalities is performed through protocols involving queries between different pairs of peers.

Consider one peer acting as a client and another acting as a server in IPFS, where the server peer is storing content and assumed to be a passive adversary. We observe that in each of the three aforementioned functionalities, the client reveals what it is querying for to the server. Peer routing reveals which peer the client is attempting to contact. Moreover, a server learns which file the client wishes to retrieve in a provider advertisement query and in a content retrieval query.

Prior work addressed only one of these problems in isolation, e.g., by enabling confidentiality for the target file in content re-

Thank you!

Questions?



HE is all you need: Compressing FHE Ciphertexts using Additive HE

Rasoul Akhavan Mahdavi, Abdulrahman Diaa, and Florian Kerschbaum

University of Waterloo, Waterloo, Ontario, Canada

Abstract. Fully Homomorphic Encryption (FHE) permits the evaluation of an arbitrary function on encrypted data. However, FHE ciphertexts, particularly those based on lattice assumptions such as LWE/RLWE are very large compared to the underlying plaintext. Large ciphertexts are hard to communicate over the network and this is an obstacle to the adoption of FHE, particularly for clients with limited bandwidth.
In this work, we propose the first technique to compress ciphertexts sent from the server to the client using an additive encryption scheme with smaller ciphertexts. Using the additive scheme, the client sends auxiliary information to the server which is used to compress the ciphertext. Our evaluation shows up to 95% percent and 97% compression for LWE and RLWE ciphertexts, respectively.

Keywords: Homomorphic Encryption · LWE · RLWE · Compression

1 Background

1.1 Homomorphic Encryption

Homomorphic Encryption is a form of public-key cryptography which permits computation on messages while in encrypted form, without the need to access the secret key. Similar to other public-key cryptosystems, homomorphic ciphertexts are larger than the underlying plaintext. The ratio between the ciphertext and plaintext is denoted as the *expansion factor*.

In a typical protocol using homomorphic encryption, a client encrypts its private input using a homomorphic cryptosystem and sends the resulting ci-