Private Information Retrieval (PIR)

Privacy: Does not learn index $i$

Efficiency: communication is sublinear in database size (ideally: polylog($N$))

record $i$
Private Information Retrieval (PIR)

record $i$

Basic building block in many privacy-preserving protocols

- Metadata-private messaging
- Certificate transparency auditing
- Private content delivery
- Contact discovery
- Private web search
- Private navigation
- Private contact tracing
- Private DNS
- Password breach checking
Certificate Transparency

Goal: monitor issuance of certificates and detect rogue certificates
A valid SCT means that the certificate was deposited into a log server. But is the log server honest? Clients will periodically audit the log server to check that SCT is actually present.

**Privacy concern:** clients reveal browsing habits to the log server.

**Goal:** monitor issuance of certificates and detect rogue certificates.
Google Chrome’s approach (*opt-out SCT auditing*): reveal a $\approx 20$-bit hash of the SCT to the log server

- Log server replies with all websites with the particular hash ($\approx 1000$ websites)
- Scheme provides $k$-anonymity notion of privacy (client visited one of 1000 possible websites)

*Can we do better?*

View this problem as a private information retrieval (PIR) problem

**Option 1:** Hash SCTs into buckets; client uses PIR to privately retrieve all SCTs in the target bucket

**Option 2:** Use a Bloom filter to represent the set of SCTs and use PIR to retrieve relevant bit(s) of the Bloom filter

**Advantage:** Provides cryptographic privacy: server learns nothing about client’s browsing habits

*But is PIR actually practical?*
Efficiency of PIR

On the Computational Practicality of Private Information Retrieval

Radu Sion ¹
Network Security and Applied Cryptography Lab
Computer Sciences, Stony Brook University
sion@cs.stonybrook.edu

Bogdan Carbunar
Pervasive Platforms and Architectures
Motorola Labs
carbunar@motorola.com

Abstract

We explore the limits of single-server computational private information retrieval (PIR) for the purpose of preserving client access patterns leakage. We show that deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. We stress that these results are beyond existing knowledge of mere “impracticality” under unfavorable assumptions. They rather reflect an inherent limitation with respect to modern hardware, likely the result of a communication-cost centric protocol design. We argue that this is likely to hold on non-specialized traditional hardware in the foreseeable future. We validate our reasoning in an experimental setup on modern off-the-shelf hardware. Ultimately, we hope our results will stimulate practical designs.

Here we discuss single-server computational PIR for the purpose of preserving client access patterns leakage. We show that deployment of non-trivial single server private information retrieval protocols on real hardware of the recent past would have been orders of magnitude more time-consuming than trivially transferring the entire database. The deployment of computational PIR would in fact increase overall execution time, as well as the probability of forward leakage, when the deployed present trapdoors become eventually vulnerable – e.g., today’s queries will be revealed once factoring of today’s values will become possible in the future.

We stress that this is beyond existing knowledge of mere “impracticality” under unfavorable assumptions. On real hardware, no existing non-trivial single server PIR protocol could have possibly had outperformed the trivial client-to-server transfer of records in the past, and is likely not to do so in the future either. This is due to the fact that on any

Take-away (2007): PIR schemes are too expensive and better to just have client download the database; need new constructions

Recurring theme in cryptography: powerful tools, but often (concretely) expensive

NDSS 2007
throughput = \frac{\text{database size}}{\text{server computation time}}

how fast the server can process a query as a function of database size
throughput = \frac{\text{database size}}{\text{server computation time}}

how fast the server can process a query as a function of database size

Without server preprocessing, server must perform a linear scan over the full database.

Possible to have PIR with sublinear server computation, but with a (concretely) expensive precomputation [LMW23] or require the client to stream the database in an offline phase [ZPSZ24, MSR23, GZS24]
25 Years of PIR Research

throughput = \frac{\text{database size}}{\text{server computation time}}

how fast the server can process a query as a function of database size

Early number-theoretic constructions: throughput on the order of KB/s (often slower than network bandwidth!)

[KO97, Pai99]
throughput = \frac{\text{database size}}{\text{server computation time}}

First lattice-based PIR instantiation: significantly faster than number-theoretic constructions (first scheme better than trivial PIR in many settings!)
25 Years of PIR Research

throughput = \frac{\text{database size}}{\text{server computation time}}

Introduced “client-side” hints to reduce communication overhead (and modestly improve throughput) in PIR
Leveraging techniques from fully homomorphic encryption to achieve higher throughput

On databases with large records, achieves throughputs of 300 MB/s to 2 GB/s

Demo: PIR for private Wikipedia: https://spiralwiki.com/
throughput = \frac{\text{database size}}{\text{server computation time}}

how fast the server can process a query as a function of database size

Considers a model where clients download a large database-dependent hint, but achieves extremely fast throughput (86% of memory bandwidth of the system)

Hint can be large: for an 8 GB database, SimplePIR hint is 362 MB and DoublePIR hint is 16 MB
25 Years of PIR Research

Server throughput (GB/s/core)

throughput = \frac{\text{database size}}{\text{server computation time}}

how fast the server can process a query as a function of database size

SimplePIR/DoublePIR without hints (using techniques from fully homomorphic encryption); YPIR’s throughput is 83% of the memory bandwidth of the system (12.1 GB/s)


Slide adapted from similar one by Henry Corrigan-Gibbs
Assuming a client makes $10^4$ TLS connections each week and performs 20 audits each week (same assumptions described in Chrome’s approach)

Assume certificate transparency log server contains 5 billion SCTs

**Using YPIR:** 29 MB of communication per client, 13.7 core-seconds of computation

(Estimated AWS costs: $228/million clients/week)

**Chrome’s $k$-anonymity approach:** 2.3 MB of communication per client

**Bottom line:** $12.6 \times$ **communication overhead** to achieve **cryptographic privacy**
We explore the limits of single-server computational private information retrieval (PIR) for the purpose of preserving client access patterns leakage. We show that deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. We stress that these results are beyond existing knowledge of mere “impracticality” under unfavorable assumptions. They rather reflect an inherent limitation with respect to modern hardware, likely the result of a communication-cost centric protocol design. We argue that this is likely to hold on non-specialized traditional hardware in the foreseeable future. We validate our reasoning in an experimental setup on modern off-the-shelf hardware. Ultimately, we hope our results will stimulate practical designs.

Lattice-based cryptography has provided the tool to realize new practical designs!

Algebraic techniques from fancy cryptography (fully homomorphic encryption) played an important role to better concrete efficiency.
The Next 5 Years of PIR Research

Two classes of constructions:

- **High throughput schemes:** \(\approx\) memory bandwidth throughput, need to communicate a few MB to retrieve a bit/byte of payload
- **High rate schemes:** communication overhead is small (<2× over direct retrieval), but throughput is limited (300-400 MB/s)

Can we combine ideas to get the best of both worlds?

Can we build concretely-efficient PIR with sublinear server computation (without having the client first stream the database)?

Can we leverage techniques from efficient PIR schemes to other domains (e.g., private set intersection, privacy-preserving machine learning)?

What will it take for companies to use PIR to better safeguard user privacy?