An Efficient PIR Construction Using Trusted Hardware

Yanjiang Yang[1], *Xuhua Ding[2]*, Robert Deng[2], Feng Bao[1]

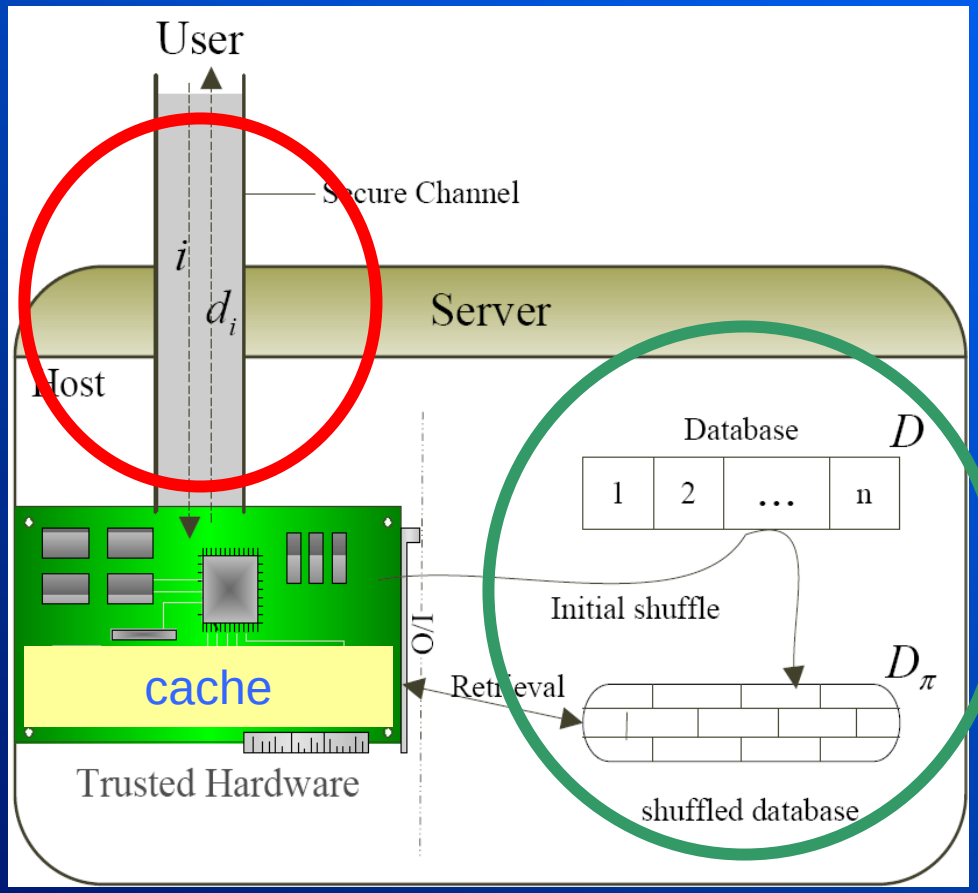
[1] Institute for Infocomm Research, Singapore

[2] Singapore Management University

Private Information Retrieval

- Private Information Retrieval (PIR)
  - A database is modeled as a n-bit string $x=x_1x_2\ldots x_n$, and a user is interested in retrieving one bit from x.
  - No information about the user's retrieval is exposed to the database server.
- Two types of PIR:
  - Traditional PIR without trusted hardware
    - Single/multiple server(s), information theoretic/computational privacy: **O(n)** computation cost
  - Trusted-hardware based PIR
    - Reduce the communication cost to **O(log n)**
    - **O(n)** computation cost for offline re-shuffle;

System Model



Server: a host of a database service
User: retrieves data items from the database by interacting with a trusted hardware (**TH**) embedded in Server
TH: has a cache for k items

Database **D** is encrypted and shuffled into **D**π by **TH**

User sends query q for the i-th item in D.
TH "returns" d_i through a secure channel.

How does TH retrieve the right item securely and efficiently?

Previous Approach [WDDB06]

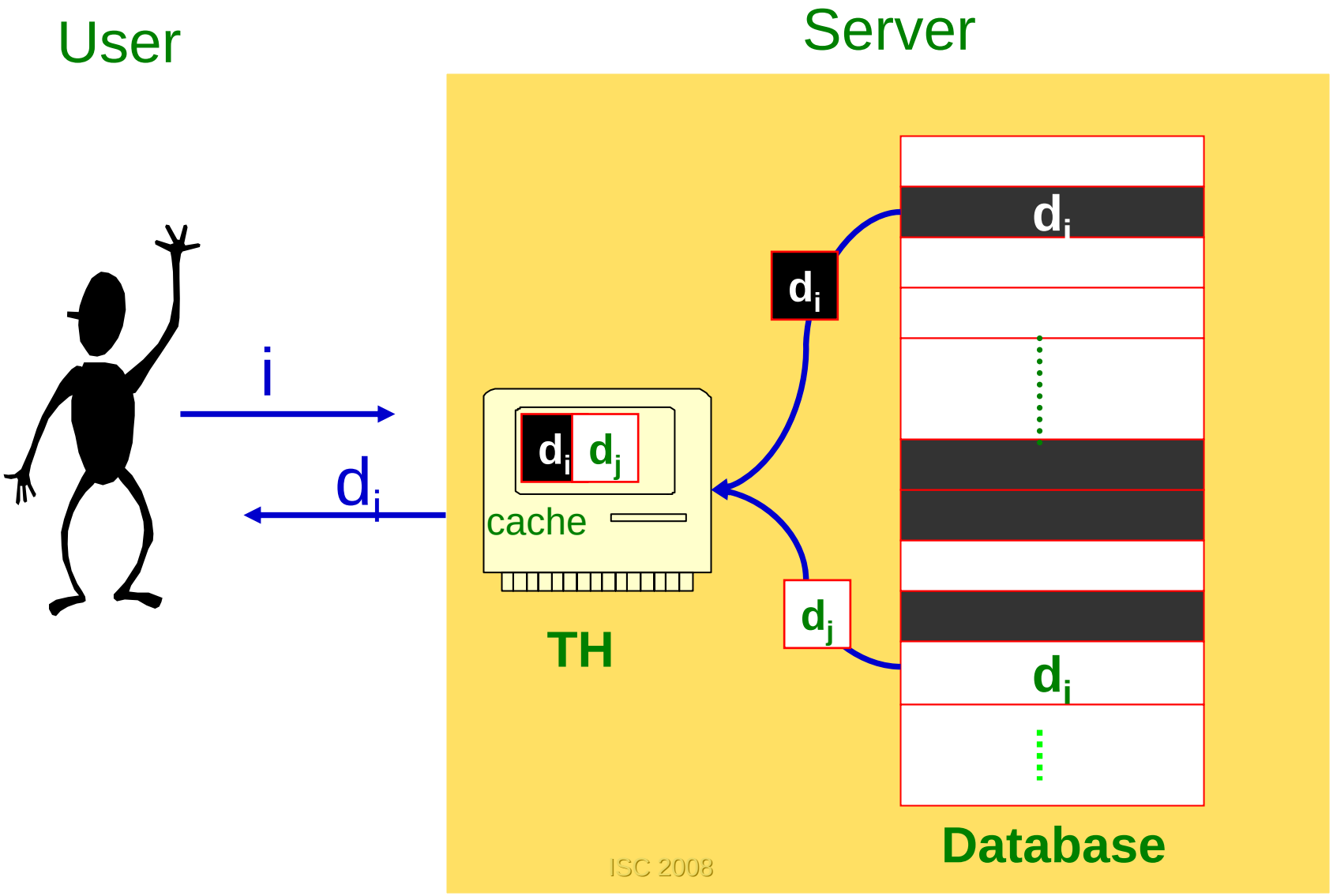
- Retrieval (on *i*):
  - if *i* is in TH's cache, randomly fetch one from the database;
  - otherwise, fetch *i*
- Session-based Shuffle:
  - whenever TH's cache is full (end of a session)
  - re-permute and re-encrypt the entire database
  - Costly! O(n) operations

Our contribution: O(n) → O(\sqrt(n))

Our New Approach

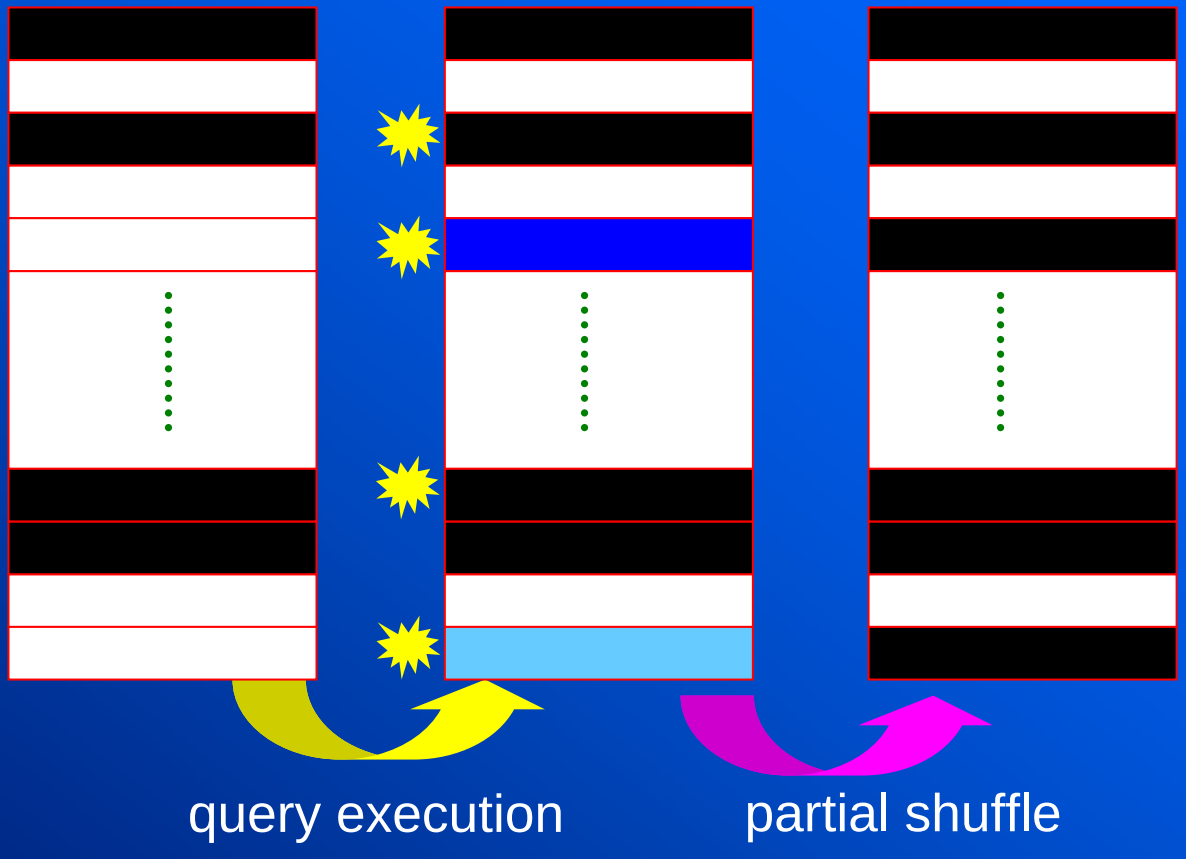
- Black & White Records
  - Initially, all records are White.
  - A record becomes Black after it has been touched by the trusted hardware.
- Twin Retrieval (on input *i,* executed by the hardware)
  - If *i* in cache, randomly read a pair of records: black and white.
  - If *i* is not in cache, read *i* and another random record in a different color.

An Example of Twin Retrieval



Offline Partial Shuffle

- Only those black records are shuffled.
- White records remain untouched.
- The shuffle algorithm is similar to the one used in [WDDB06].
- Gradually growing cost. Linear with the number of queries.



Implementation Challenges

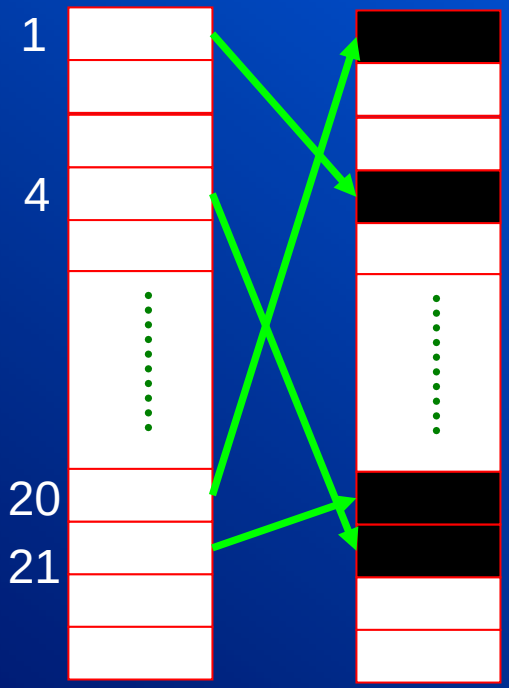
- The colors of records are randomly distributed.
- Given an index *i,* How does the trusted hardware know the color of the corresponding record?
- How to implement “random fetching” a white/black record?

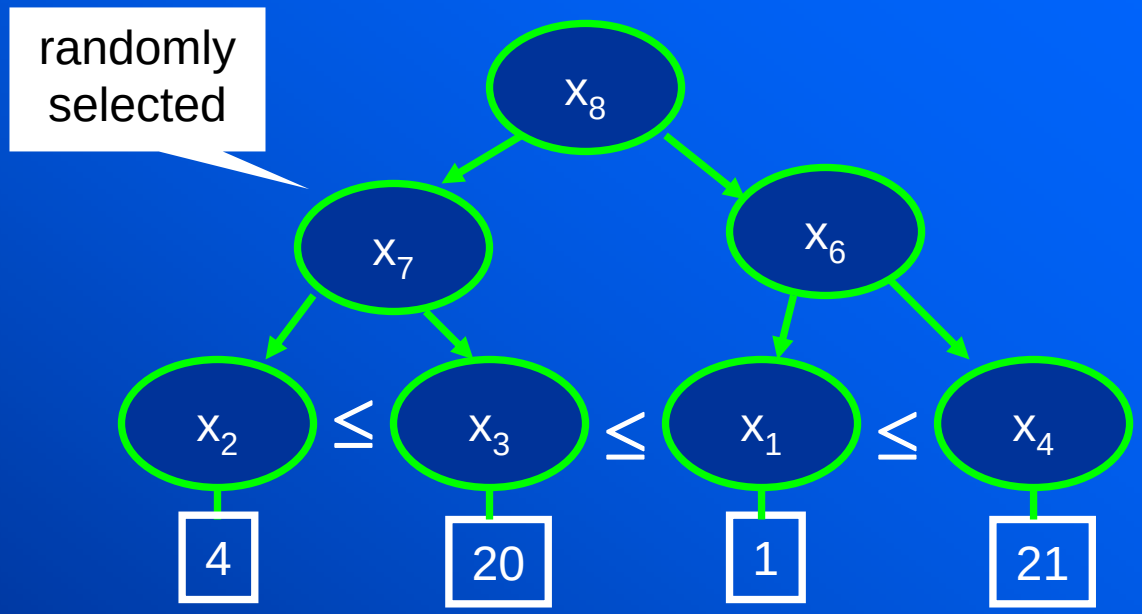
Any access to the server’s space may leak information!

Solution

- We use two auxiliary data structures, which are stored in the server's space.
  - one for managing black records and the other for white records
- Accesses to these structures are oblivious.
- The idea is to combine color-detection and record-locating with one process.

Management of Black Records





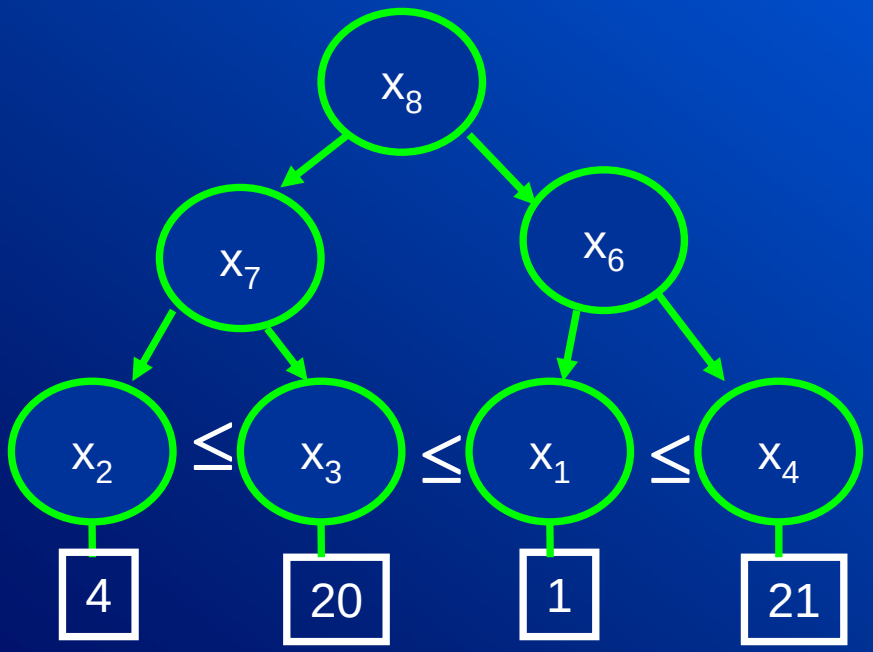
(20,1) (1,4) (21,20) (4,21)

⬇ F( ), encryption function

(x_1,1) (x_2,4) (x_3,20) (x_4,21)

One-stone-two-birds: retrieval of a black record

Query: i-th record



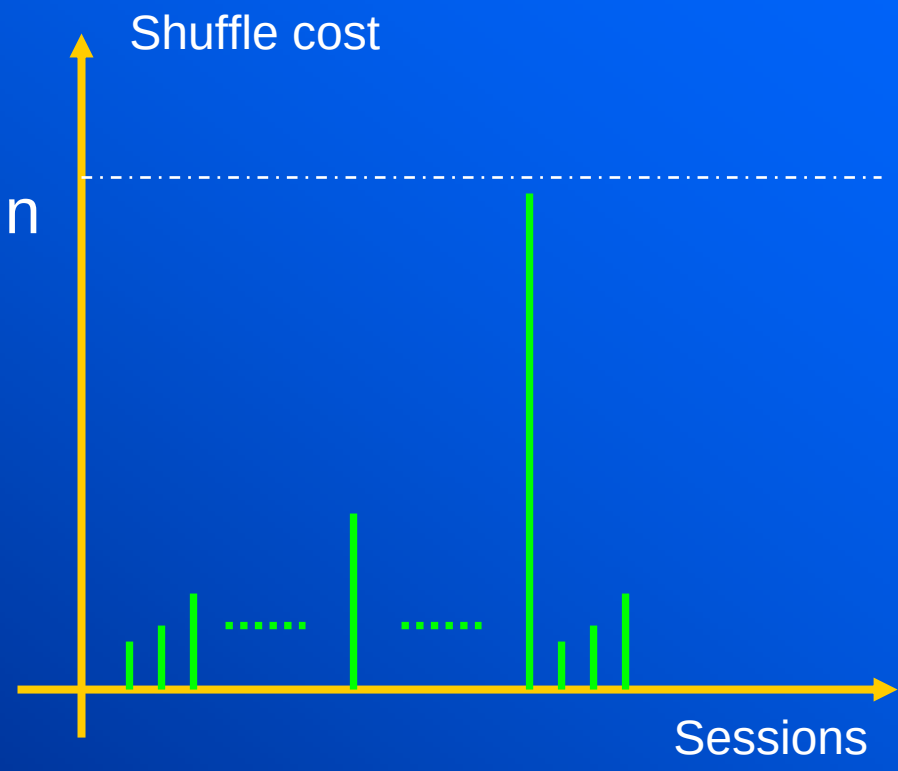
- The algorithm
  - x= F(i)
  - $x < x_2$ or $x > x_4$, it is White. do a random search
  - Otherwise, search x. Return the leaf-node
  - Two possibilities.

Remarks

- A similar method is used for management of white records.
- Initialization is trivial since the trusted hardware knows all necessary information after the first session.
- The auxiliary data structures are updated as part of the shuffle algorithm executed by the trusted hardware after each session.
- The initialization and update process are oblivious.

Performance

- The shuffle cost grows linearly with the number of queries executed.
- Our scheme does not require a large cache in the hardware.
- Perform a FULL shuffle to reset the system when necessary.
- The amortized cost per query is **O(**\sqrt(n)**)**



Thank You

Q/A