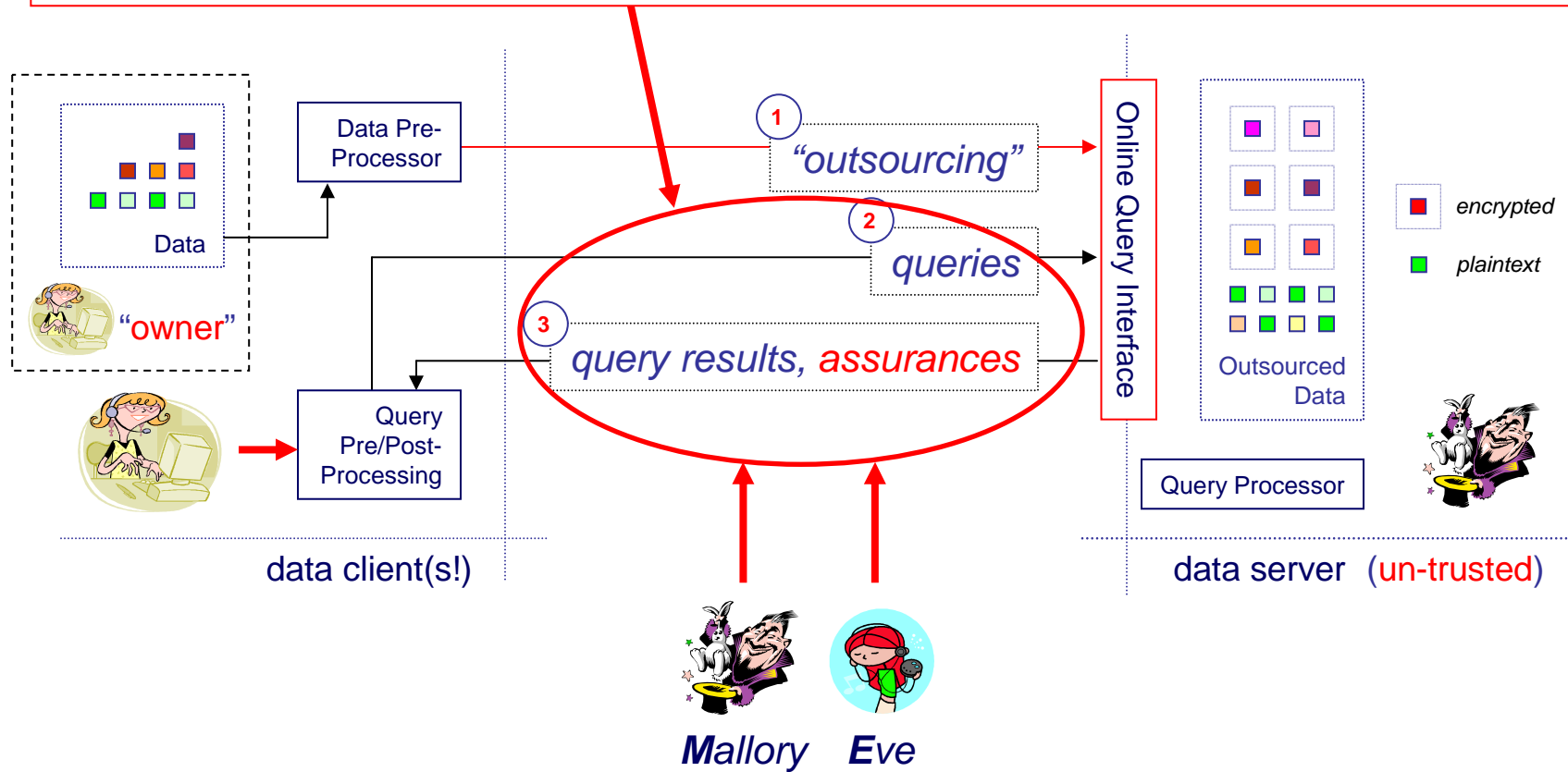


- ➡ *Crypto Crash Course*
- ➡ *Data Outsourcing*
- ➡ *Query Correctness*
- ➡ *Data Confidentiality*
- ➡ *Access Privacy*
- ➡ *Searching on Encrypted Data*
- ➡ *Trusted Hardware*

Data Outsourcing

$assurances \subseteq \{query\ correctnes, data\ confidentiality, access\ privacy\}$



Outsourcing Challenges

Un-trusted server:

- lazy: incentives to perform less
- curious: incentives to acquire information
- malicious:
 - denial of service
 - incorrect results
 - possibly compromised

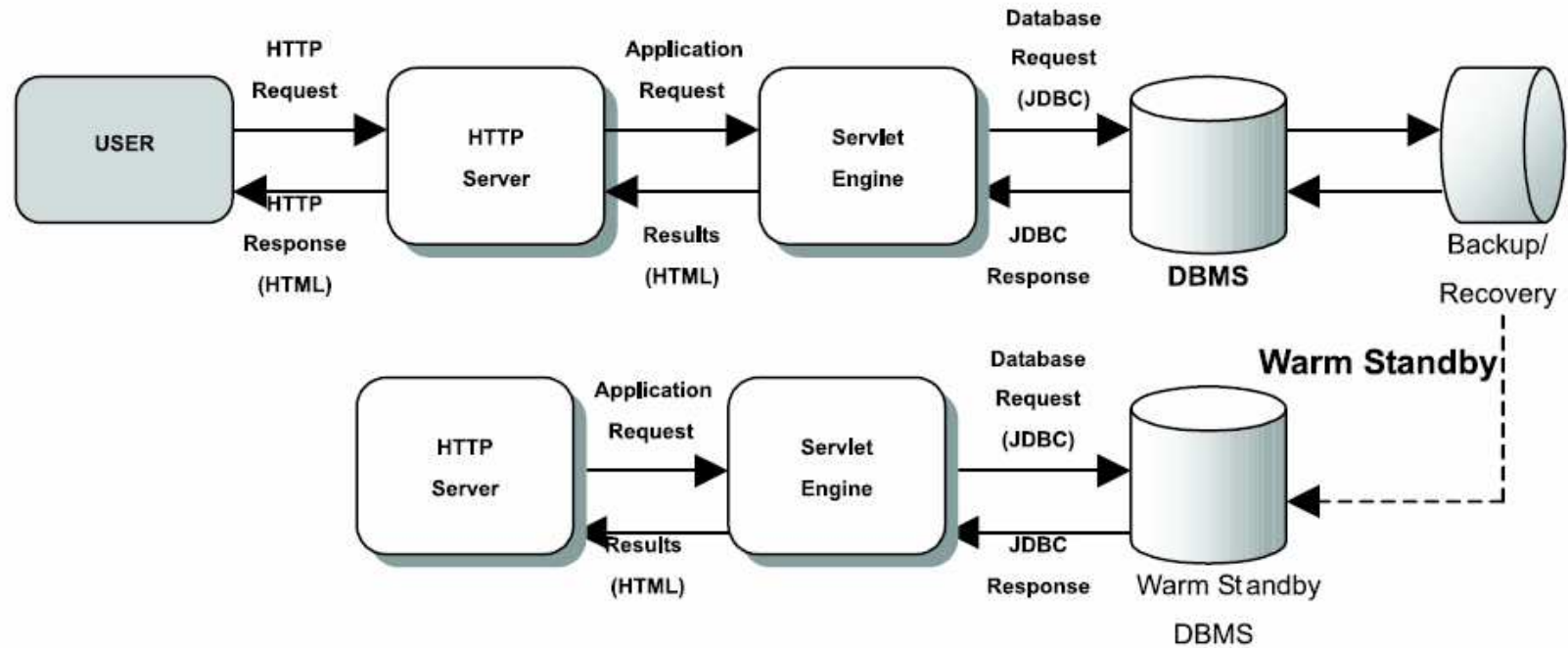
Why is this hard ?

- how ?
- arbitrary expressivity
- overheads
 - network
 - computational costs

What do we do ?

- query assurances
- full privacy
 - of queries (even encrypted)
 - of access patterns
- data confidentiality

Hacigumus (2002)



System architecture of NetDB2

Stored Data Confidentiality

```
SELECT decrypt(discount, key)
FROM lineitem
WHERE custid = 300
```

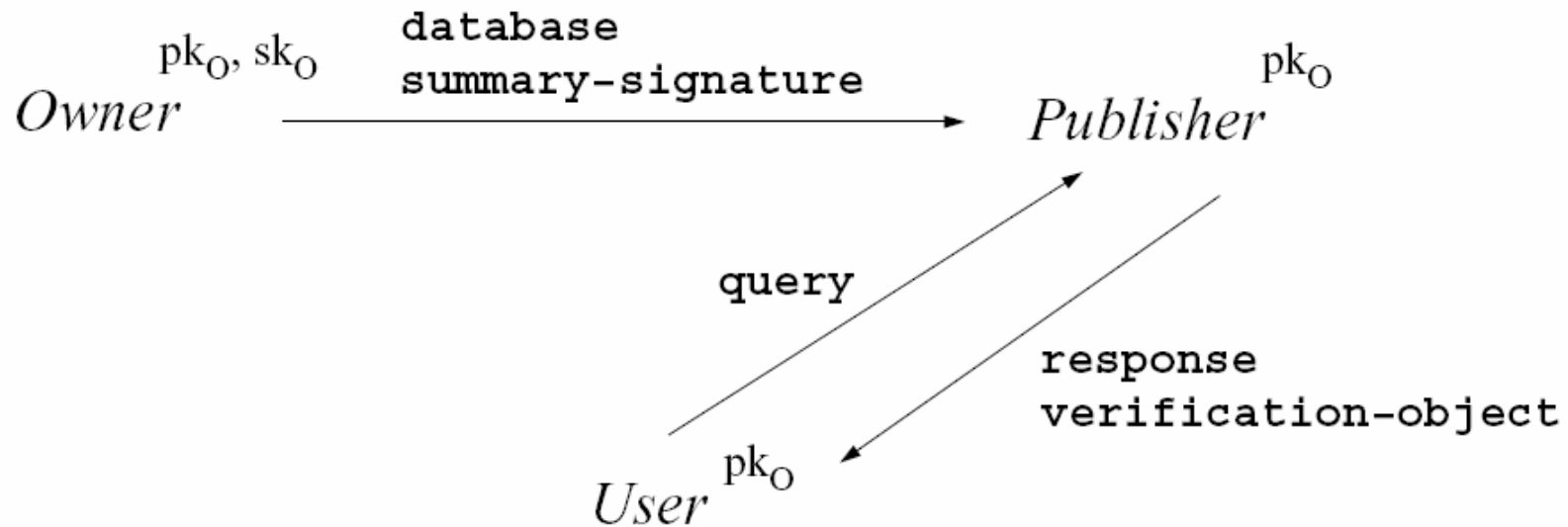
H. Hacigumus, B. R. Iyer, and S. Mehrotra.
Providing database as a service, ICDE 2002.

- ➔ *Crypto Crash Course*
- ➔ *Data Outsourcing*
- ➔ *Query Correctness*
- ➔ *Data Confidentiality*
- ➔ *Access Privacy*
- ➔ *Searching on Encrypted Data*
- ➔ *Trusted Hardware*

Client requires quantifiable assurances that query results are correct, for arbitrary query types in the presence of a server that could be ...

... lazy

... and/or fully malicious (!)

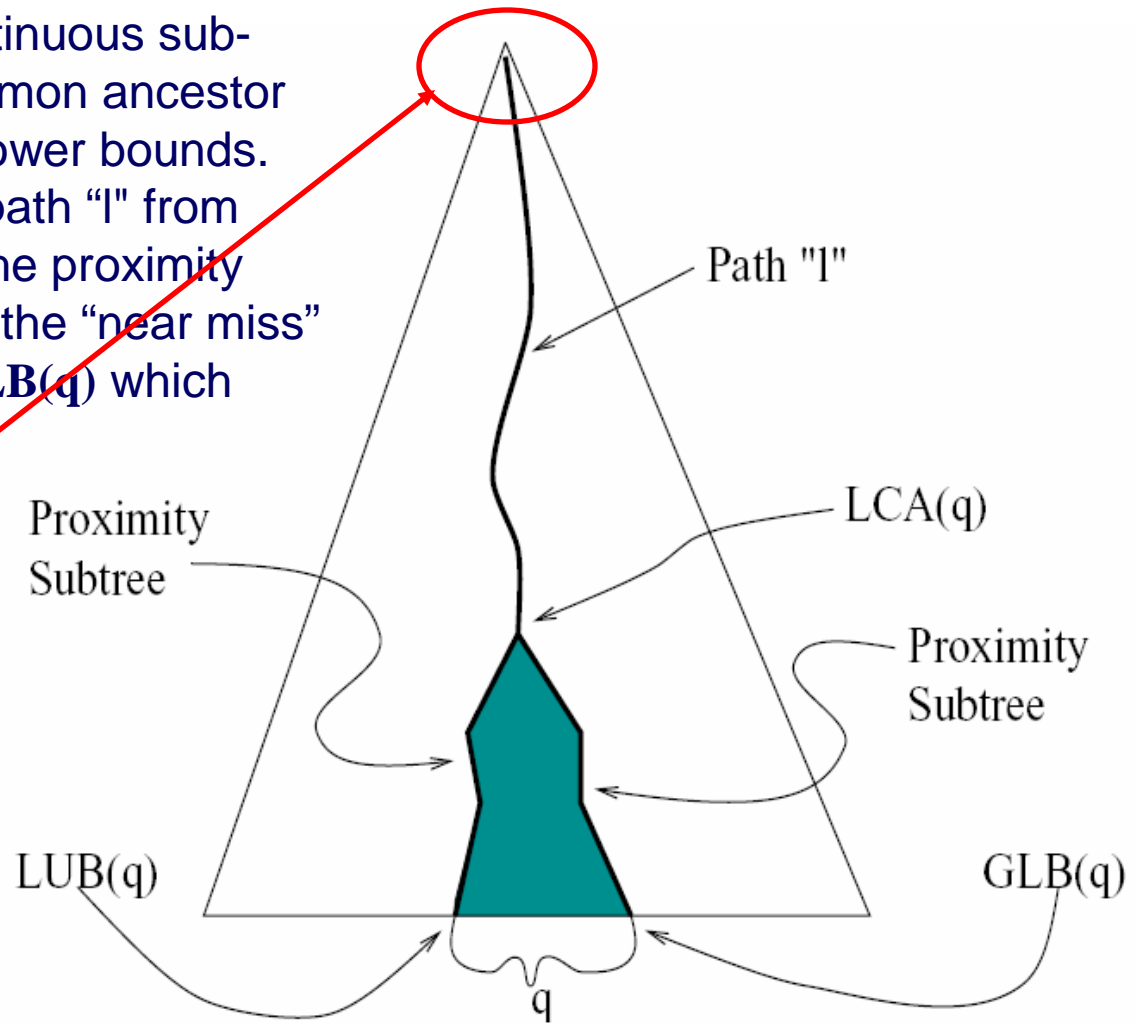


The owner provides database updates and summary signatures to the un-trusted publisher. When users make inquiries with the publisher, they get responses which can be verified using a returned verification-object. Only sk_O is secret, pk_O is authenticated.

Devanbu et. al. (2000)

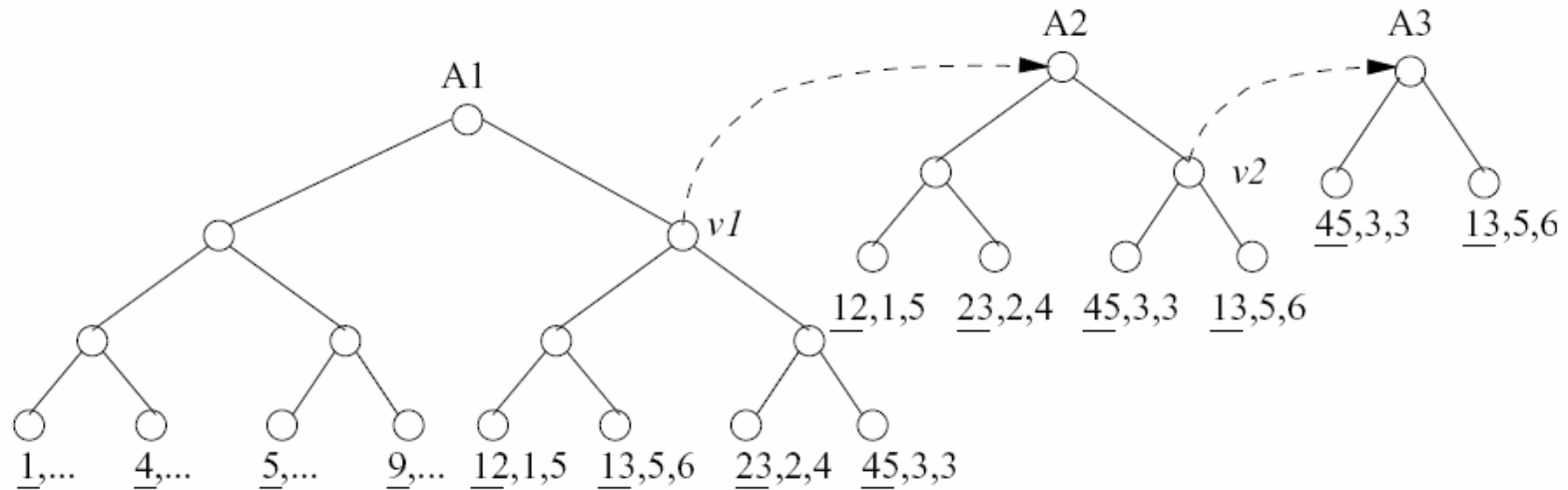
A Merkle tree, with a continuous sub-range q , with a least common ancestor $LCA(q)$, and upper and lower bounds. Note the verifiable hash path "I" from $LCA(q)$ to the root, and the proximity sub-trees (thick lines) for the "near miss" tuples for $LUB(q)$ and $GLB(q)$ which show that q is complete.

authenticated via signature

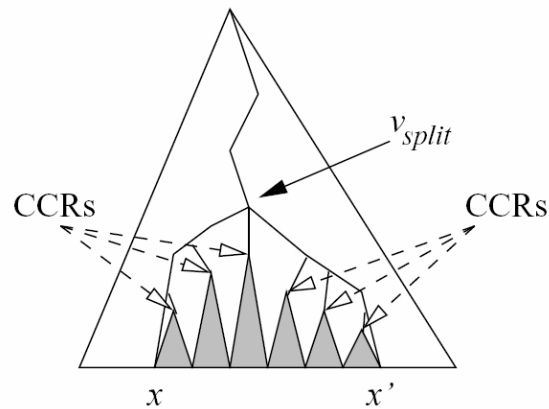


Supported claimed operations:

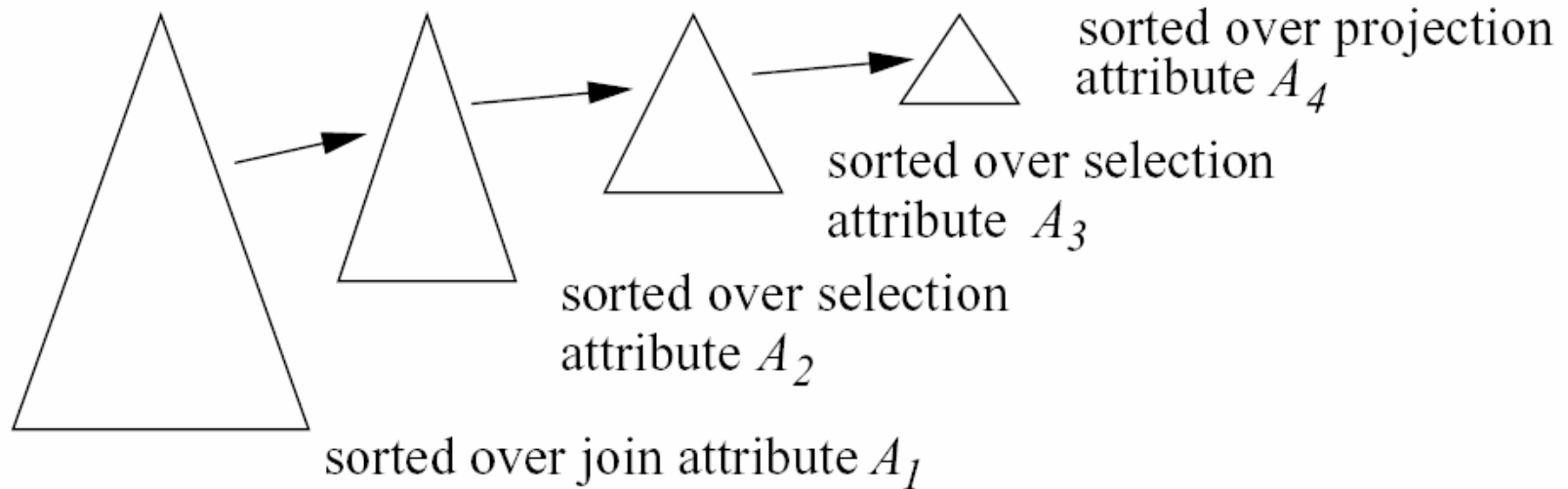
- selections
- projections
 - (1) maintaining VOs before duplicate elimination
 - (2) pre-computing VOs for common projections
- equiJOIN
 - (1) keep materialized cartesian product $S \times R$
 - construct VO on sorted version of product (according to difference (S.A-R.A)) – this yields 3 types of leaf nodes (“0”, “<”, “>”) in Merkle tree
 - (2) all kinds of other tricks
- set operations
 - union (client does it and verifies VOs for input sets)
 - intersection (?)
 - multi-dimensional range queries (generalizing hash tree to “multi-dimensional range tree”)



Excerpt of a 3-dimensional range tree, sorted by attributes A_1, A_2 and A_3



Covering canonical roots (CCR):
 roots of the canonical sub-trees
 precisely covering the leaves
 with values in the interval.



```
SELECT S.A4 FROM S,R  
WHERE S.A1=R.A1 AND A2<10 AND A3>17
```

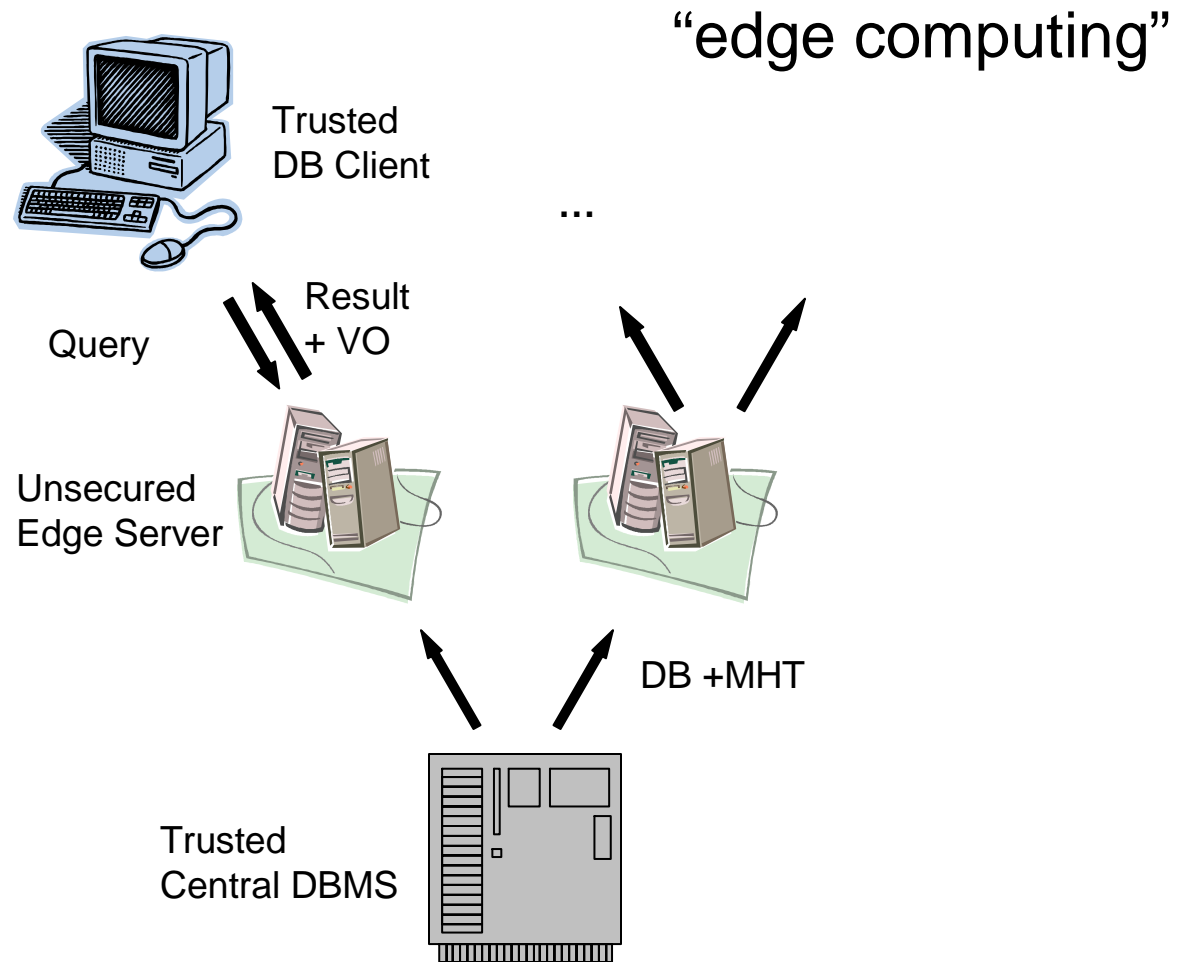
Issues:

- query expressiveness
- query flexibility
 - works only on data with VOs
- “universe split” phenomenon
 - use timestamps, expiration times
- expensive operations (!)

Discusses the use of batch verification of signatures and similar techniques (condensed RSA) to authenticate results.

		Condensed-RSA	Batch-DSA	BGLS
Sign	1 signature	6.82	3.82	3.54
Verify	1 signature	0.16	8.52	62
	t = 1000, k = 1	44.12	1623.59	184.88
	t = 100, k = 10	45.16	1655.86	463.88
	t = 1000, k = 10	441.1	16203.5	1570.8

Cost comparison (in msec): verification and signing. Notation: t – # signatures, k – # signers



Claimed problems with [Devanbu 2000]

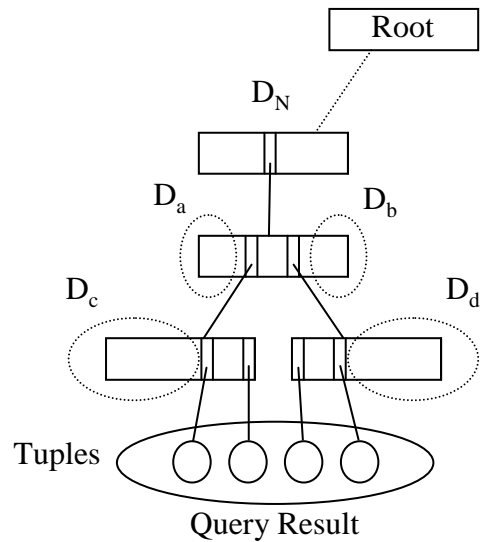
- A hash tree is needed for every sort-order
- VOs need to contain links all the way to the root,
 - VOs grow linearly to query result and logarithmic to base table size
- Projections may have to be performed by clients
- No provision for dynamic updates on the database

Aim 1: VO size just linear in query result

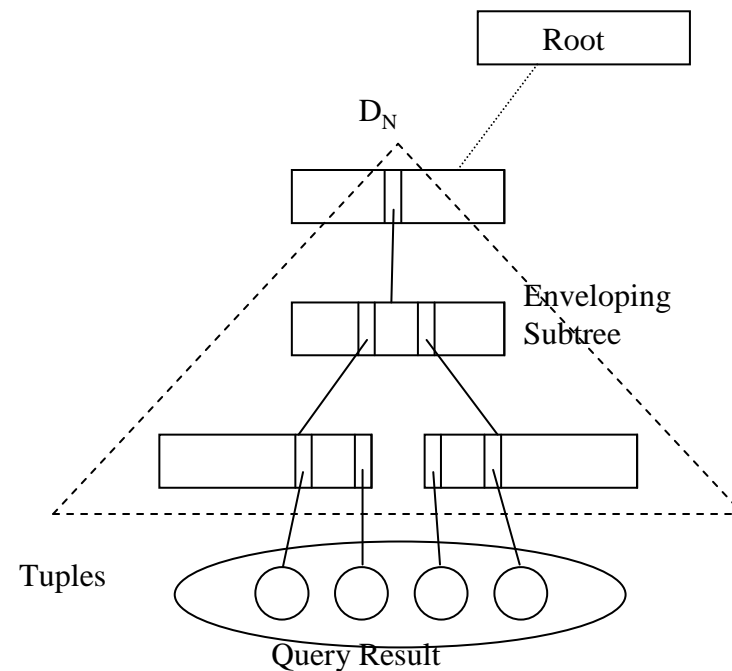
Aim 2: do not push projections to client

Idea: use different hash function

- $h(x) = g^x \text{ mod } q$
- h is commutative, $h(x+y) = h(y+x)$
 - Digests can be combined arbitrarily
 - Projection can be performed at the edge servers
 - Facilitates insertion of new tuples with minimal effect on other digests
- but: significantly (1000-10000 times) slower
- trade-off: computation vs. communication



Verification object = $D_N + D_S$
 where $D_S = \{D_a, D_b, D_c, D_d\}$



Verifying Selection
 (no need to go up to the root
 as everything is also signed)

Similar expressiveness. But ...

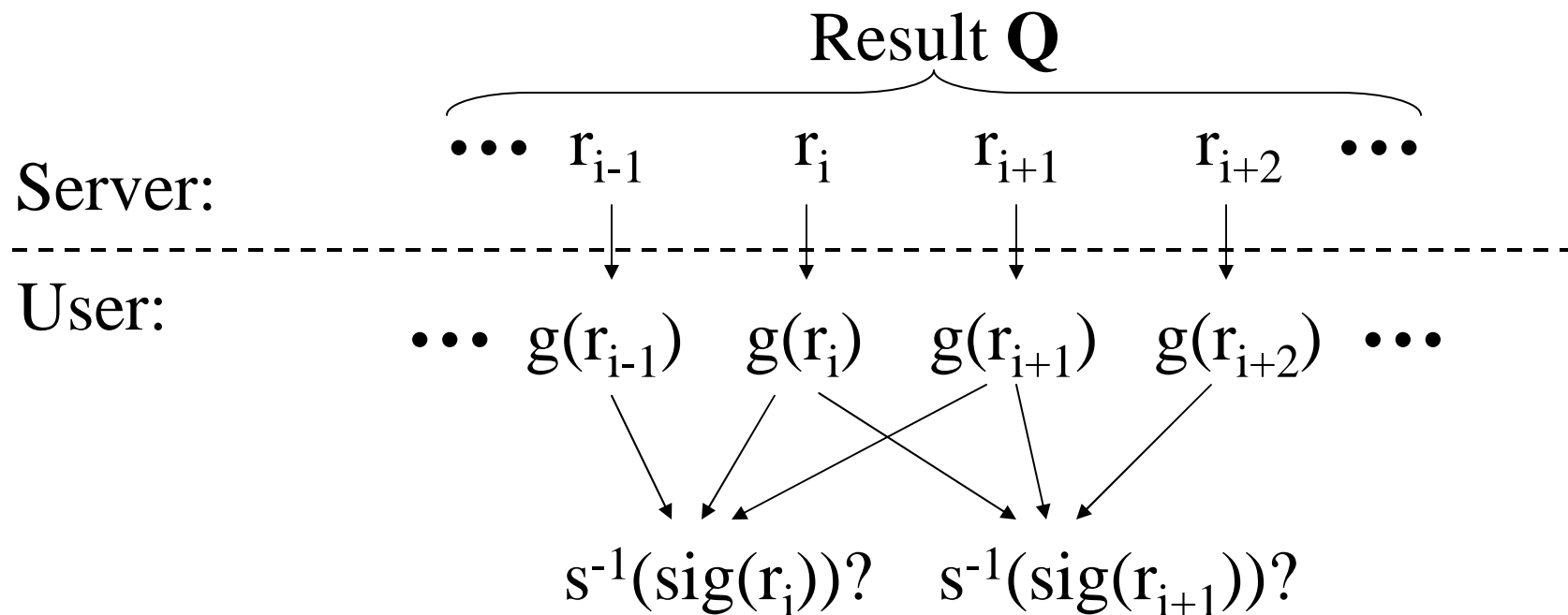
Asks: what about access control rules ?
(Devanbu seems to reveal too much: boundary tuples)

Also claims: lower overheads for queries
and updates.

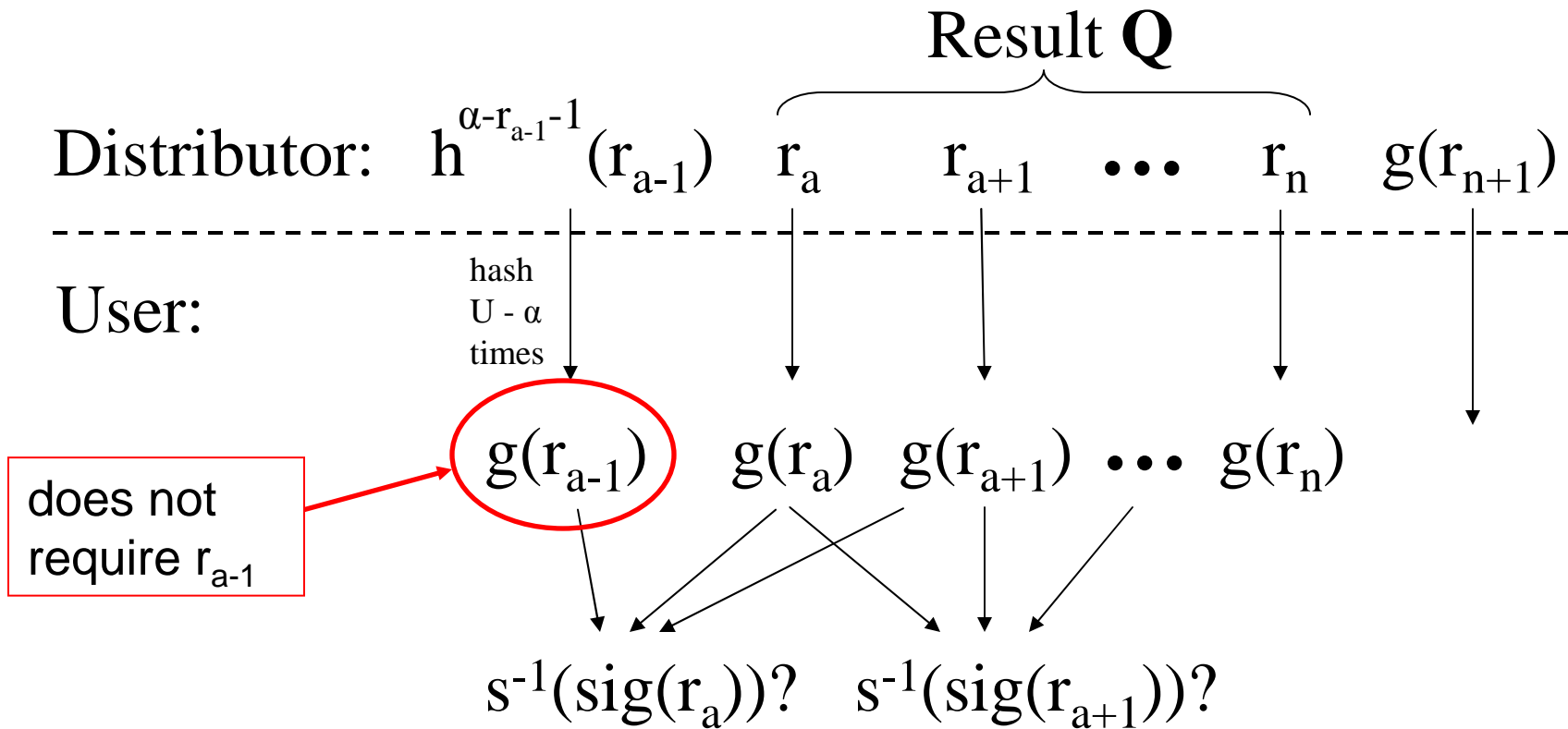
Introduces “precision” (only data
matching the query should be returned)

Idea: use signature chains – thus no need to reveal boundary elements.

$$\text{sig}(r_i) = s(\text{h}(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1})))$$



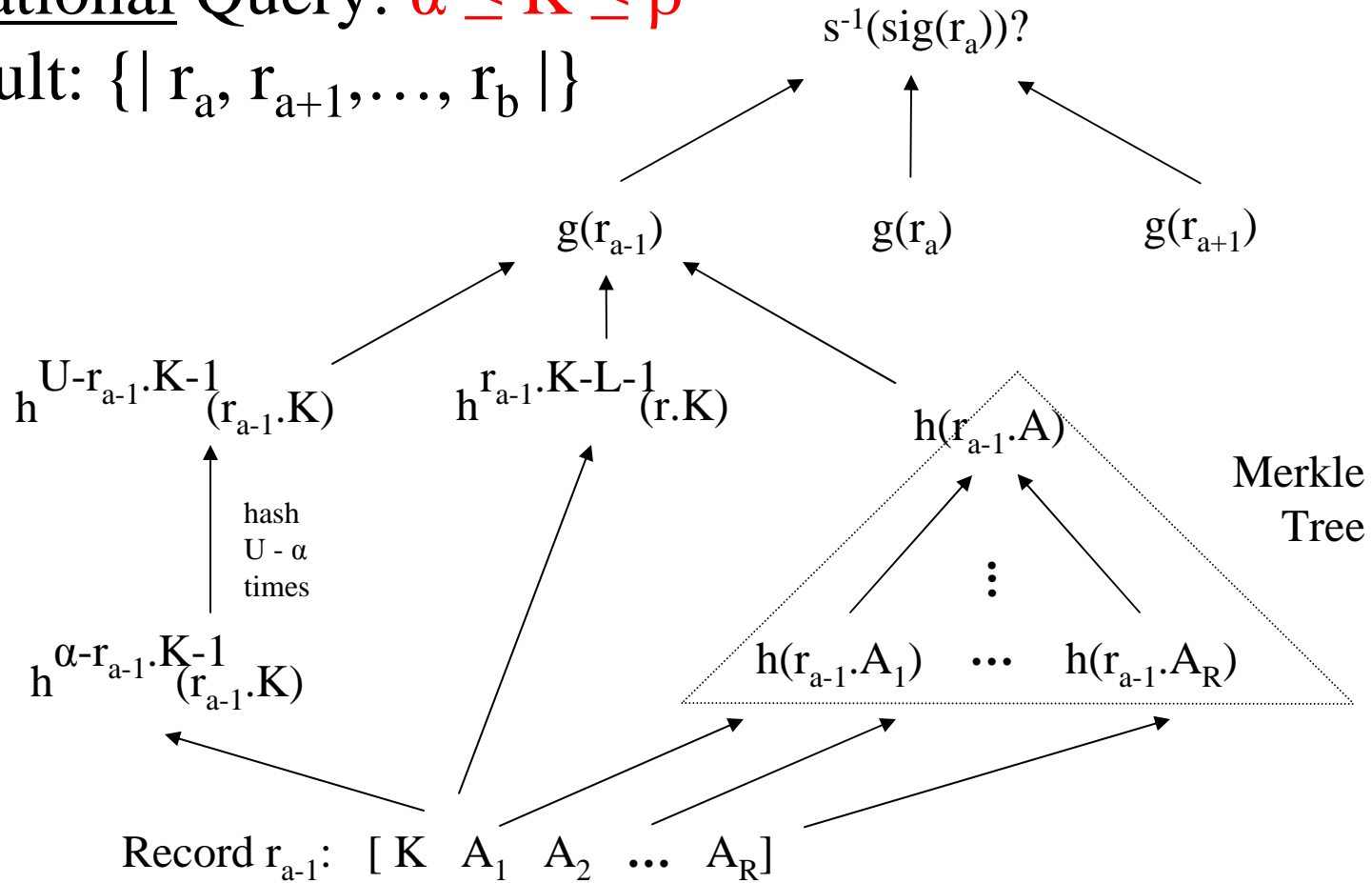
But what is g : $g(r) = h^{U-r-1}(r)$



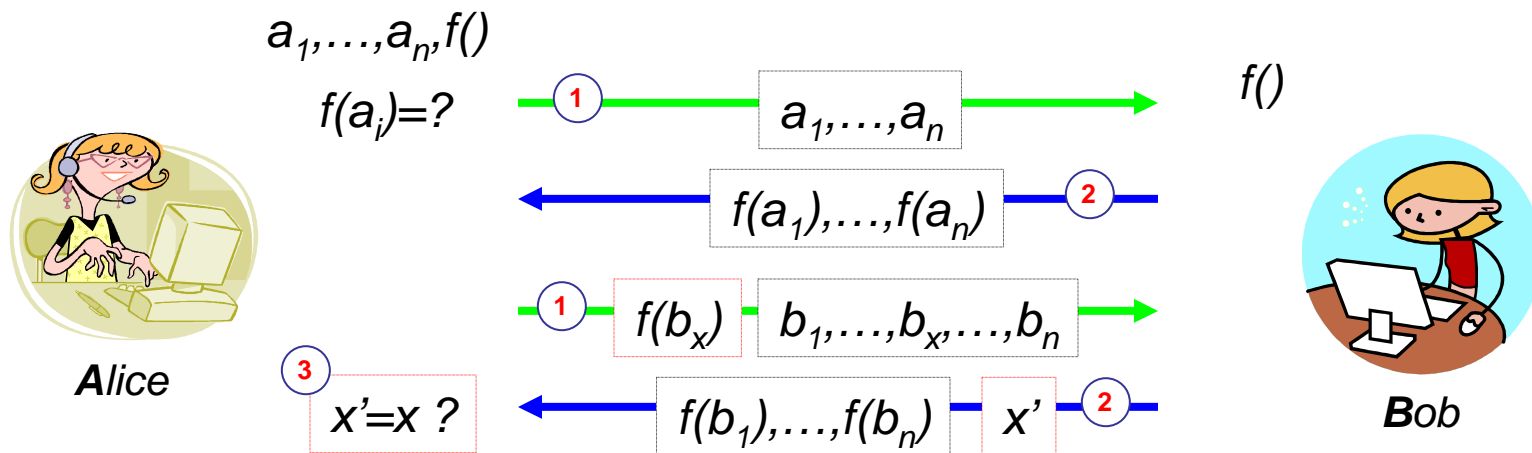
Query: $\alpha \leq r$

Relational Query: $\alpha \leq K \leq \beta$

Result: $\{ | r_a, r_{a+1}, \dots, r_b | \}$

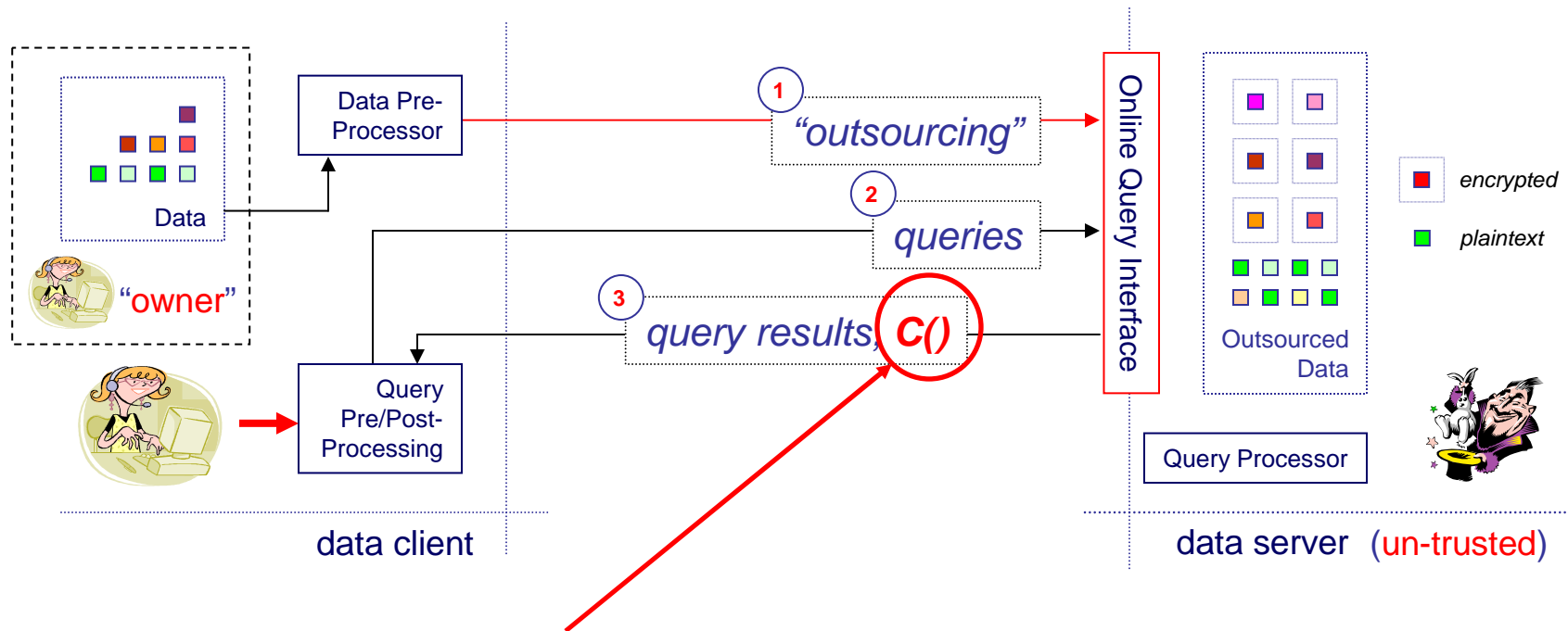


Asks: What about arbitrary queries ?



P. Golle and I. Mironov, "Uncheatable Distributed Computations", RSA 2001 (Cryptographer's track)

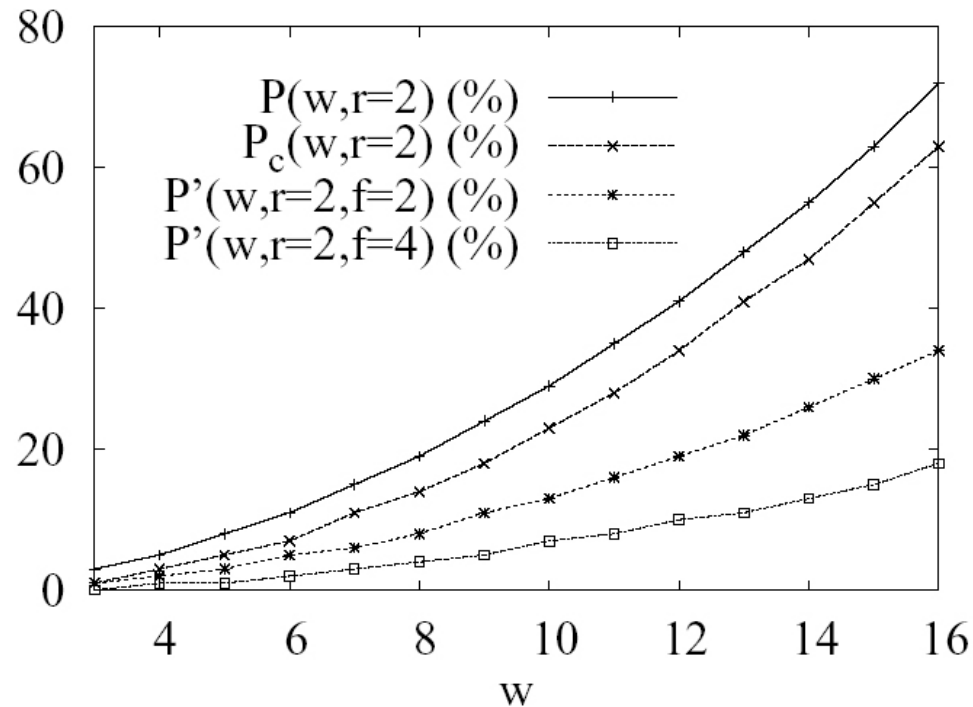
Sion: Execution Proofs



$$C(Q, x, \epsilon) = \{H(\epsilon || \rho(Q_x)), \epsilon\}$$

A challenge token (computed by client) is sent together with the batch of queries. Upon return, batch execution is proved if $x=x'$.

Sion: Cheating Probability

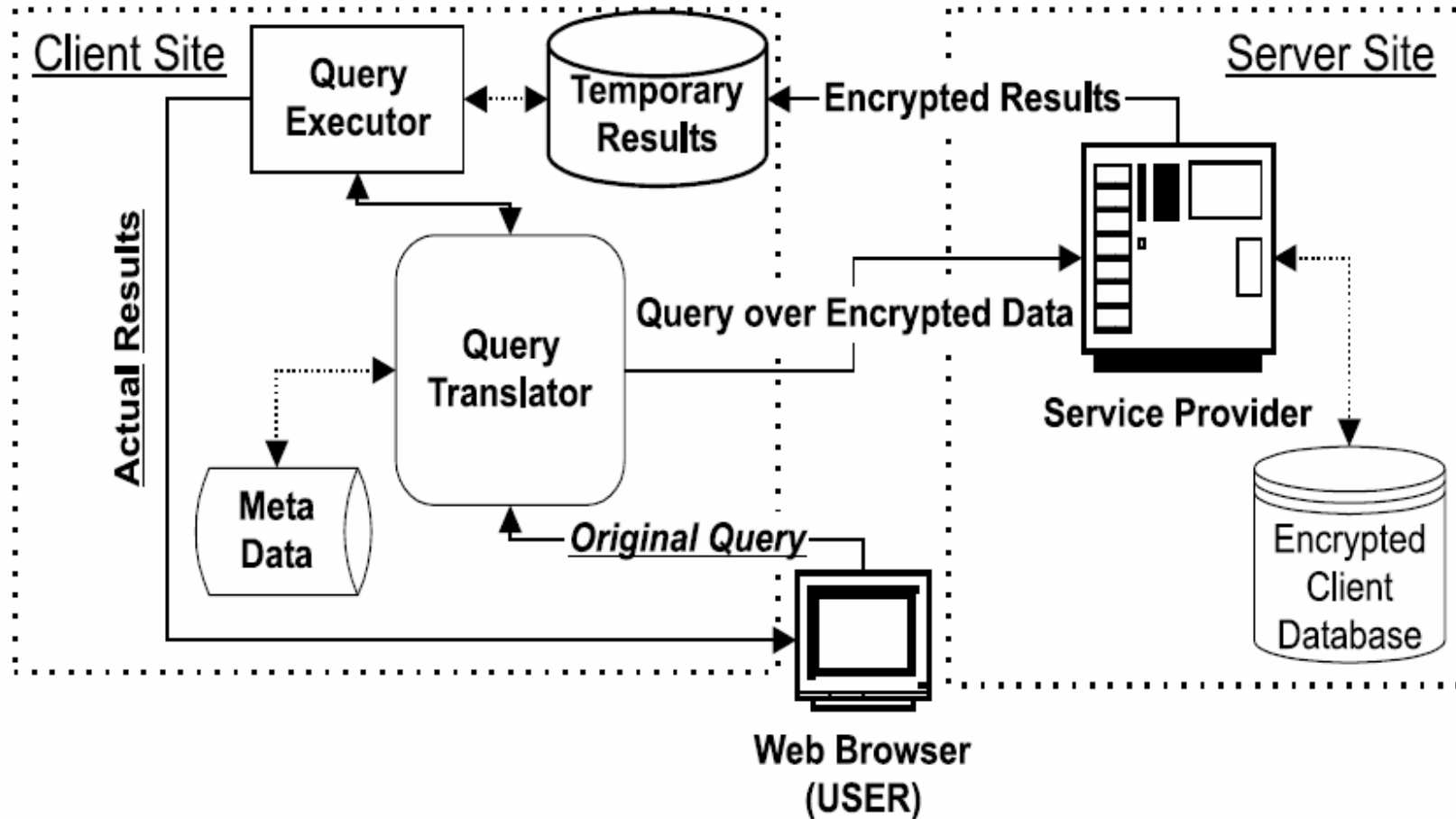


Only handles lazy server !

The behavior of $P'(w, r, f)$ (fake tokens) plotted against $P_c(w, r)$ (client-side result checking mechanism) showing that the query execution proof mechanism (with fake tokens) significantly decreases the ability to “get away” with less work.

- ➔ *Crypto Crash Course*
- ➔ *Data Outsourcing*
- ➔ *Query Correctness*
- ➔ *Data Confidentiality*
- ➔ *Access Privacy*
- ➔ *Searching on Encrypted Data*
- ➔ *Trusted Hardware*

Hacigumus (SIGMOD 2002)

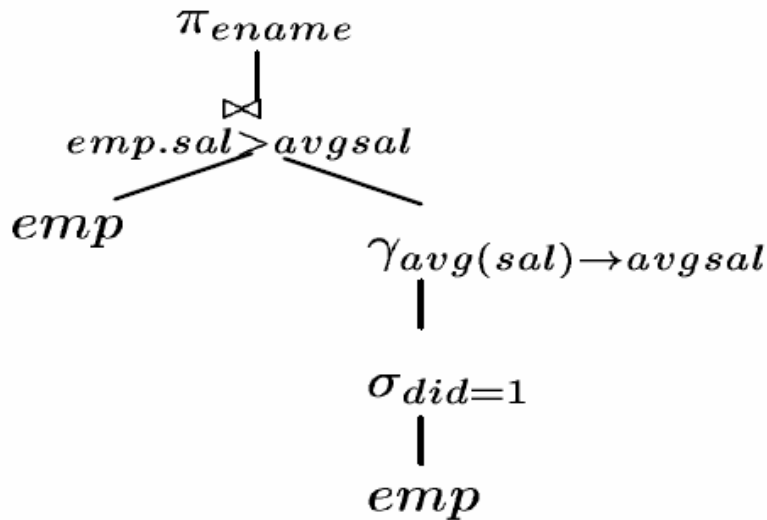


Main Steps:

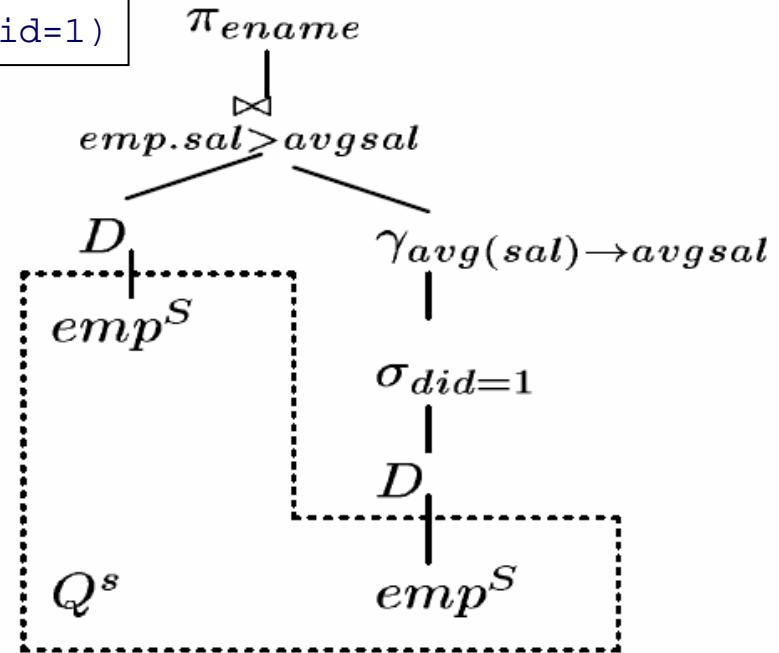
1. Partition sensitive domains
 - Order preserving: supports comparison
 - Random: query rewriting becomes hard
2. Rewrite queries to target partitions
3. Execute queries and return results
4. Prune/post-process results on client

Hacigumus (SIGMOD 2002)

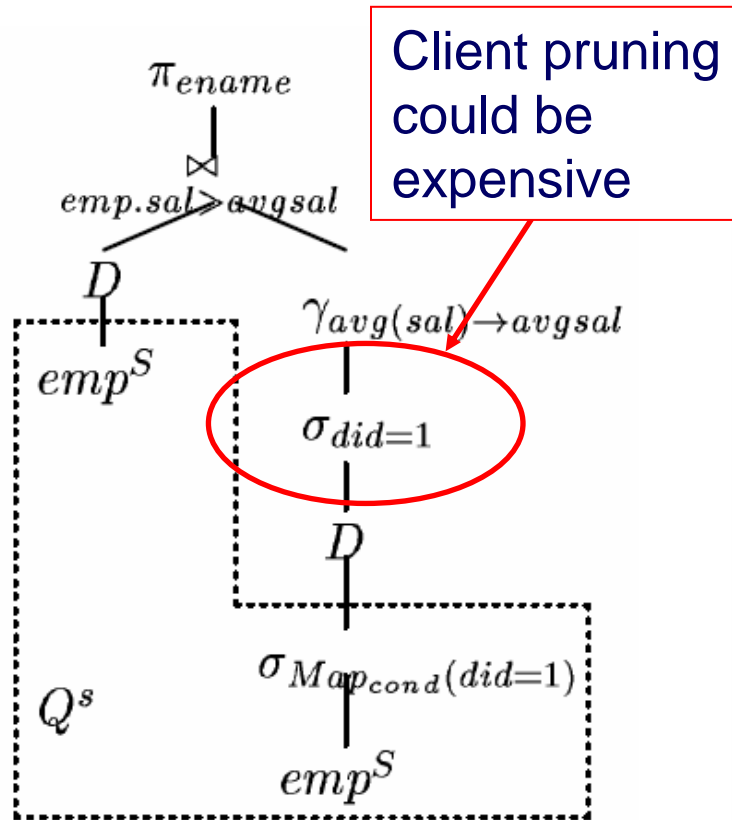
```
SELECT emp.name FROM emp
WHERE emp.salary >
  (SELECT AVG(salary) FROM emp WHERE did=1)
```



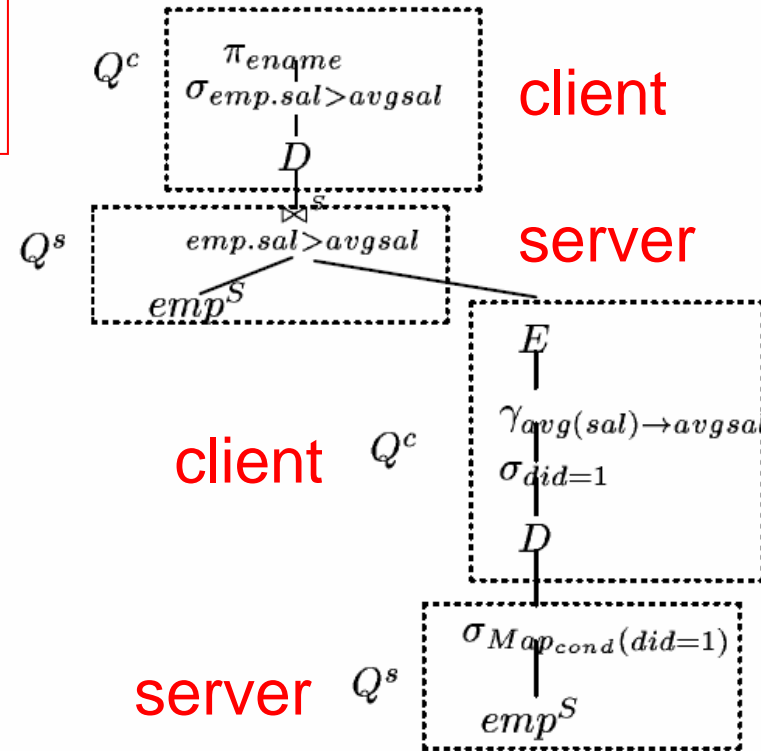
(a) Original query tree.



(b) Replacing encrypted relations.



(c) Doing selection at server.



(d) Multiple interactions between Client and Server.

Confidentiality-Overhead Trade-off

Larger segments ==
increased privacy ==
increased overheads

Goal: For a uniform distribution of queries
- minimize any leaks to any adversaries
(even) knowing segmentation parameters.

Idea 1: Maximize variance of distribution
of values in segment

Idea 2: Increase segment entropy

Issue: What about performance ?

Solution: “Controlled Diffusion”

Idea:

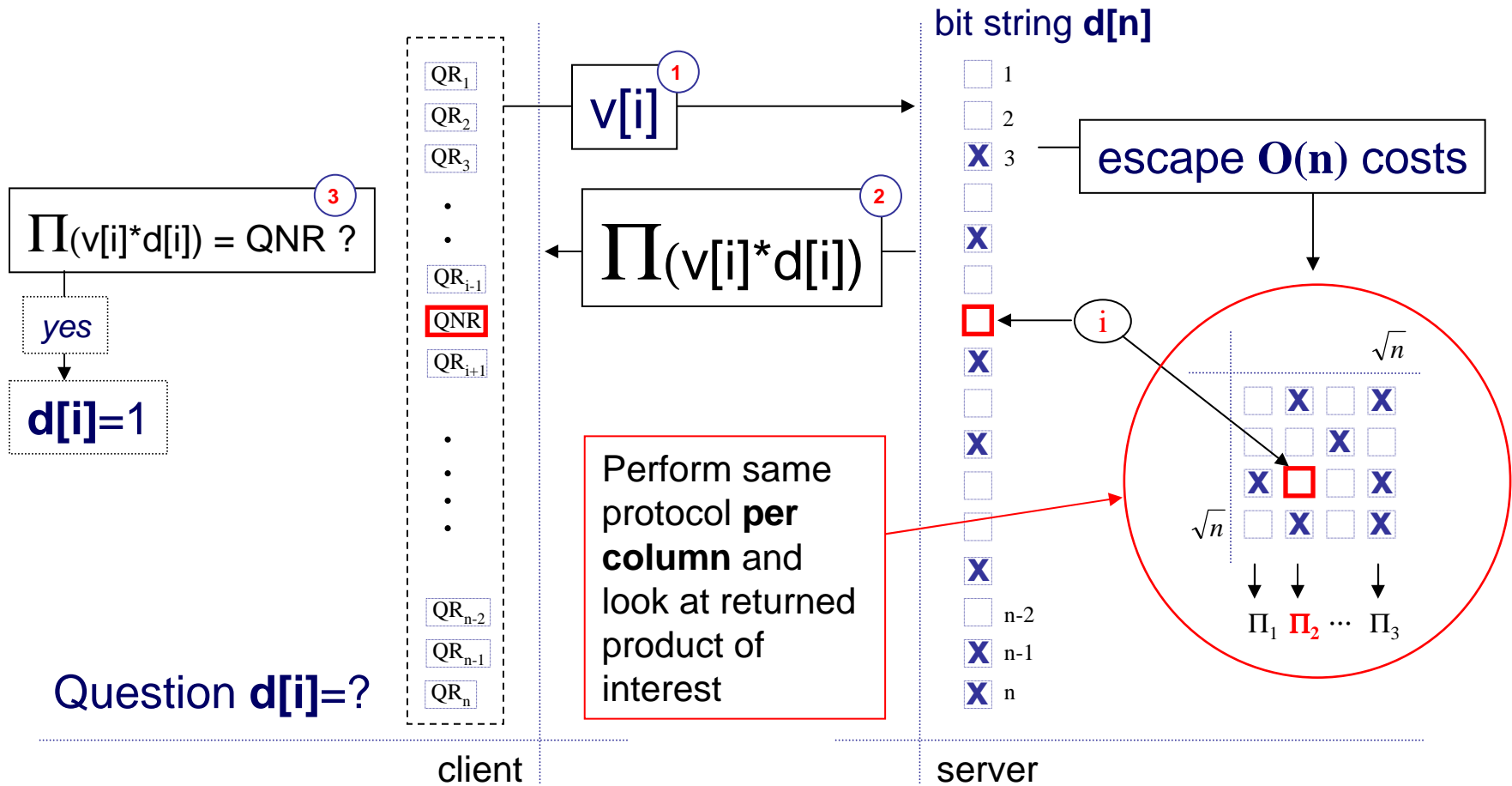
1. design for efficiency, then ...
2. ... diffuse (re-distribute) elements inside the segments to increase per-segment entropy and variance

Asks: Similarly, how to structure query trees to optimally balance the security-efficiency trade-off in [Hacigumus 2002].

Idea: client generates optimal partitioned query execution plans given statistics and metadata input from the server.

- ➔ *Crypto Crash Course*
- ➔ *Data Outsourcing*
- ➔ *Query Correctness*
- ➔ *Data Confidentiality*
- ➔ *Access Privacy*
- ➔ *Searching on Encrypted Data*
- ➔ *Trusted Hardware*

QR PIR



The n bits of the database are organized logically at the server as a bi-dimensional matrix M of size $\sqrt{n} \times \sqrt{n}$. To retrieve bit $M(x, y)$ with computational privacy, the client:

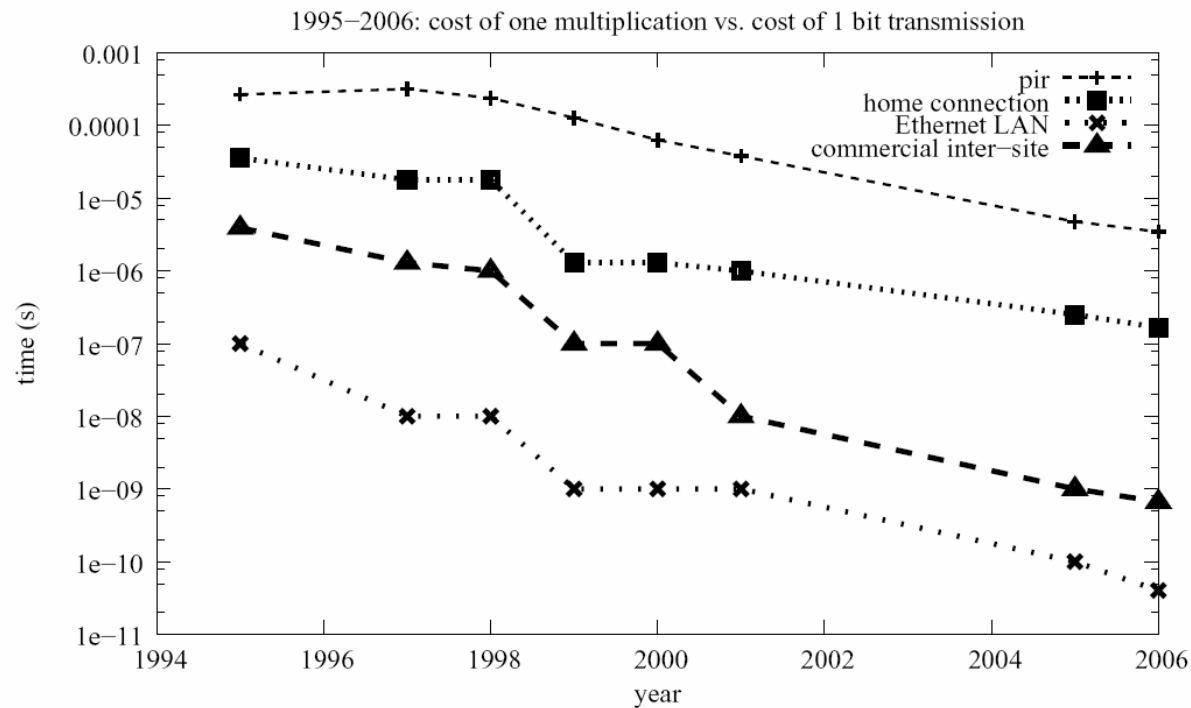
- randomly chooses two prime numbers p and q of similar bit length, computes their product, $N = pq$ and sends it to the server.
- generates \sqrt{n} numbers $s_1, s_2, \dots, s_{\sqrt{n}}$, such that s_x is a quadratic non-residue (QNR) and the rest are quadratic residues (QR) in \mathbb{Z}_N^* .
- sends $s_1, s_2, \dots, s_{\sqrt{n}}$ to the server.

For each “column” $j \in (1, \sqrt{n})$ in the $\sqrt{n} \times \sqrt{n}$ matrix, the server:

- computes the product $r_j = \prod_{0 < i < \sqrt{n}} q_{ij}$ where $q_{ij} = s_i^2$ if $M(i, j) = 1$ and $q_{ij} = s_i$ otherwise ².
- sends $r_1, \dots, r_{\sqrt{n}}$ to the client

The client then simply checks if r_y is a QR in \mathbb{Z}_N^* which implies $M(x, y) = 1$, else $M(x, y) = 0$.

PIR is (still) impractical

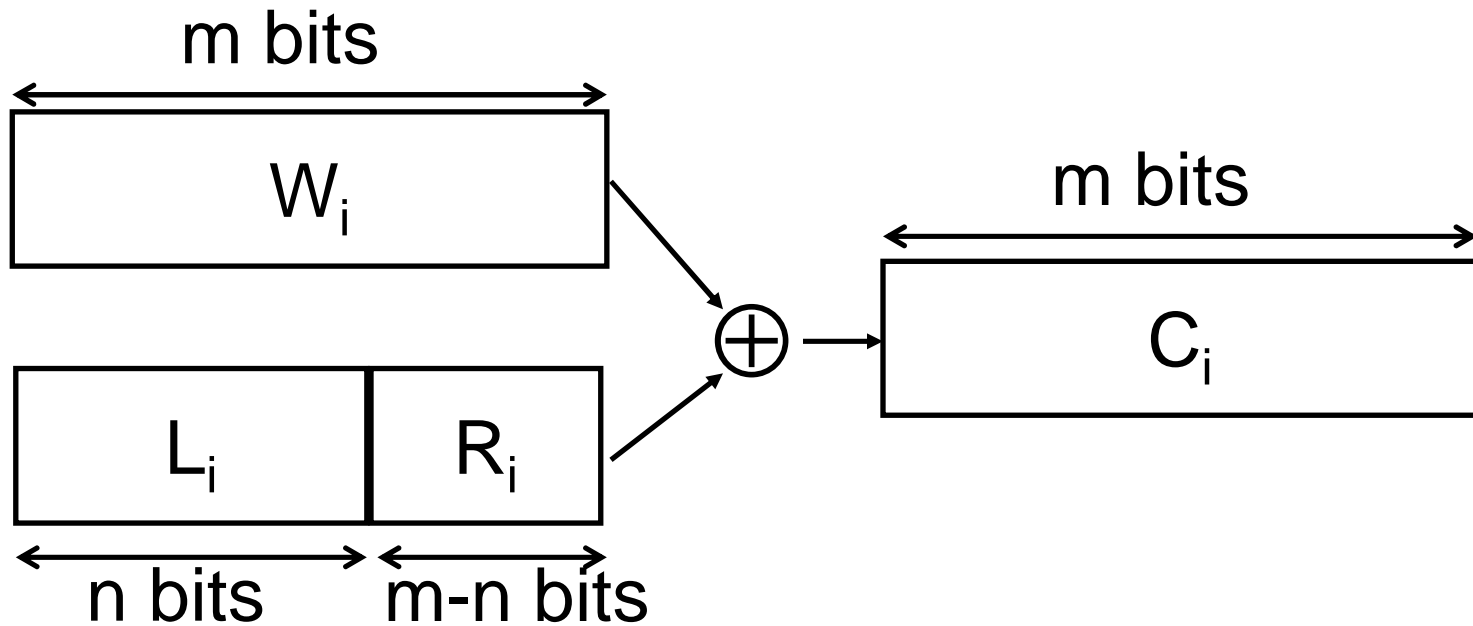


Comparison between the time required to perform PIR and the time taken to transfer the database, between 1995 and 2005. (logarithmic)

- ➔ *Crypto Crash Course*
- ➔ *Data Outsourcing*
- ➔ *Query Correctness*
- ➔ *Data Confidentiality*
- ➔ *Access Privacy*
- ➔ *Searching on Encrypted Data*
- ➔ *Trusted Hardware*

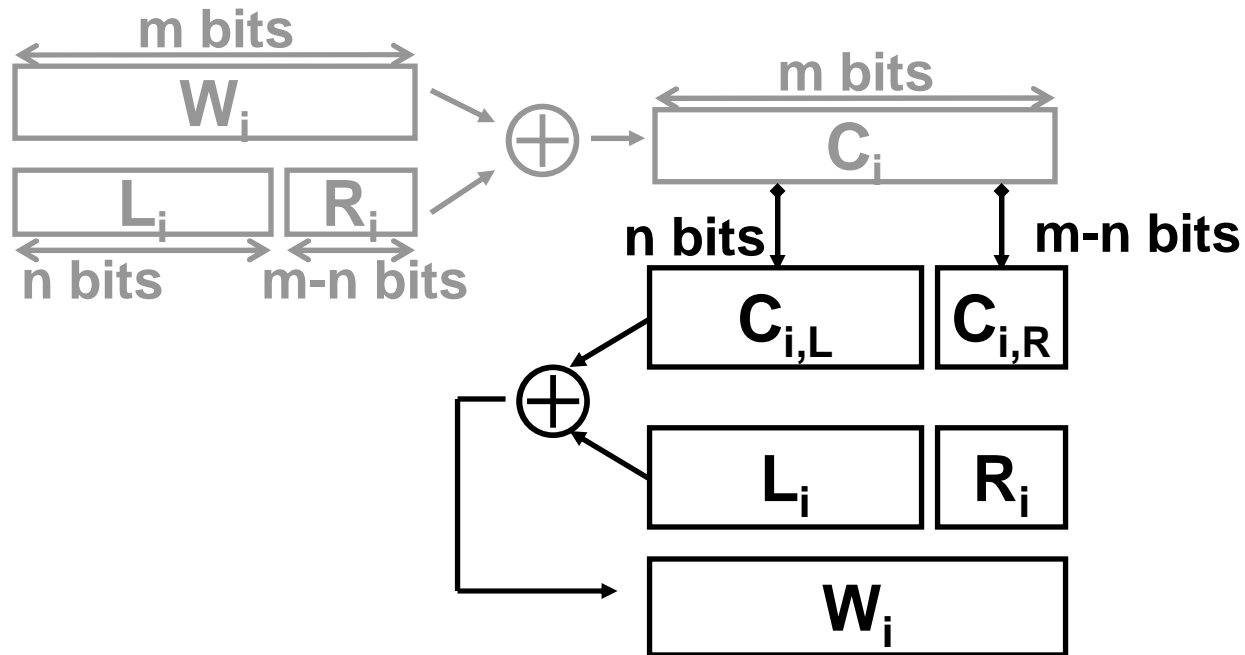
- Sequential Scan
- Index-based

Encryption:



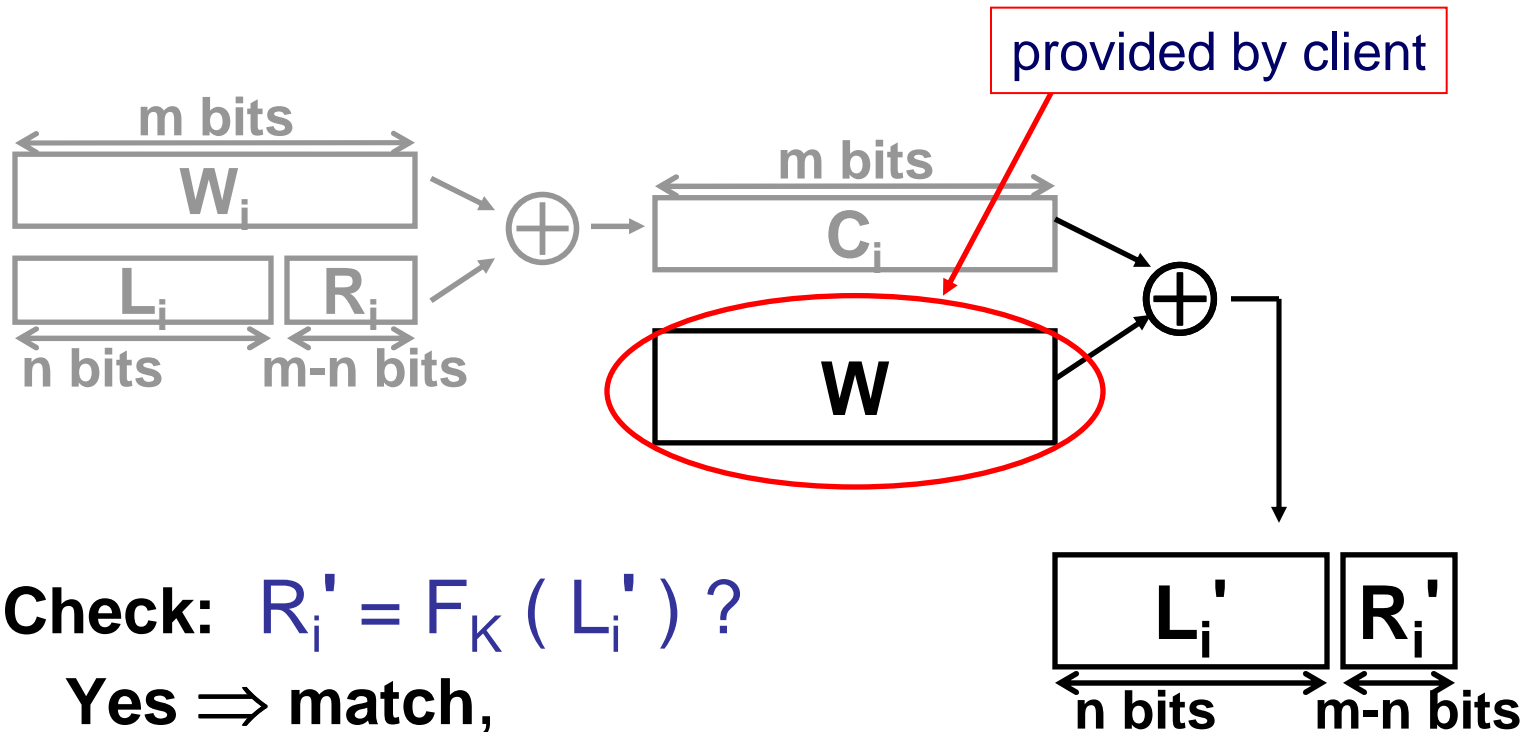
$$L_i \leftarrow G_i(\text{seed}), \quad R_i \leftarrow F_K(L_i)$$

Decryption:



$$L_i \leftarrow G_i(\text{seed}), \quad R_i \leftarrow F_K(L_i)$$

Search:

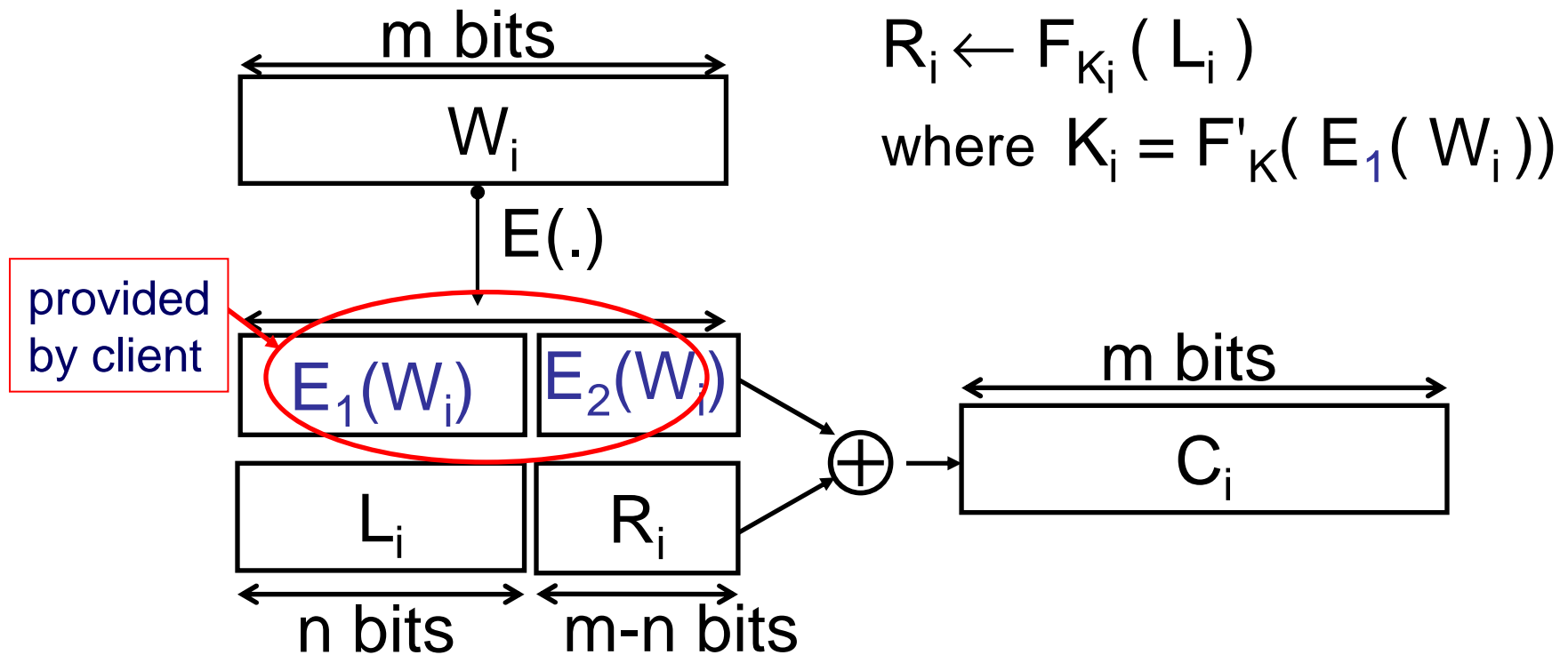


Check: $R'_i = F_K(L'_i)$?

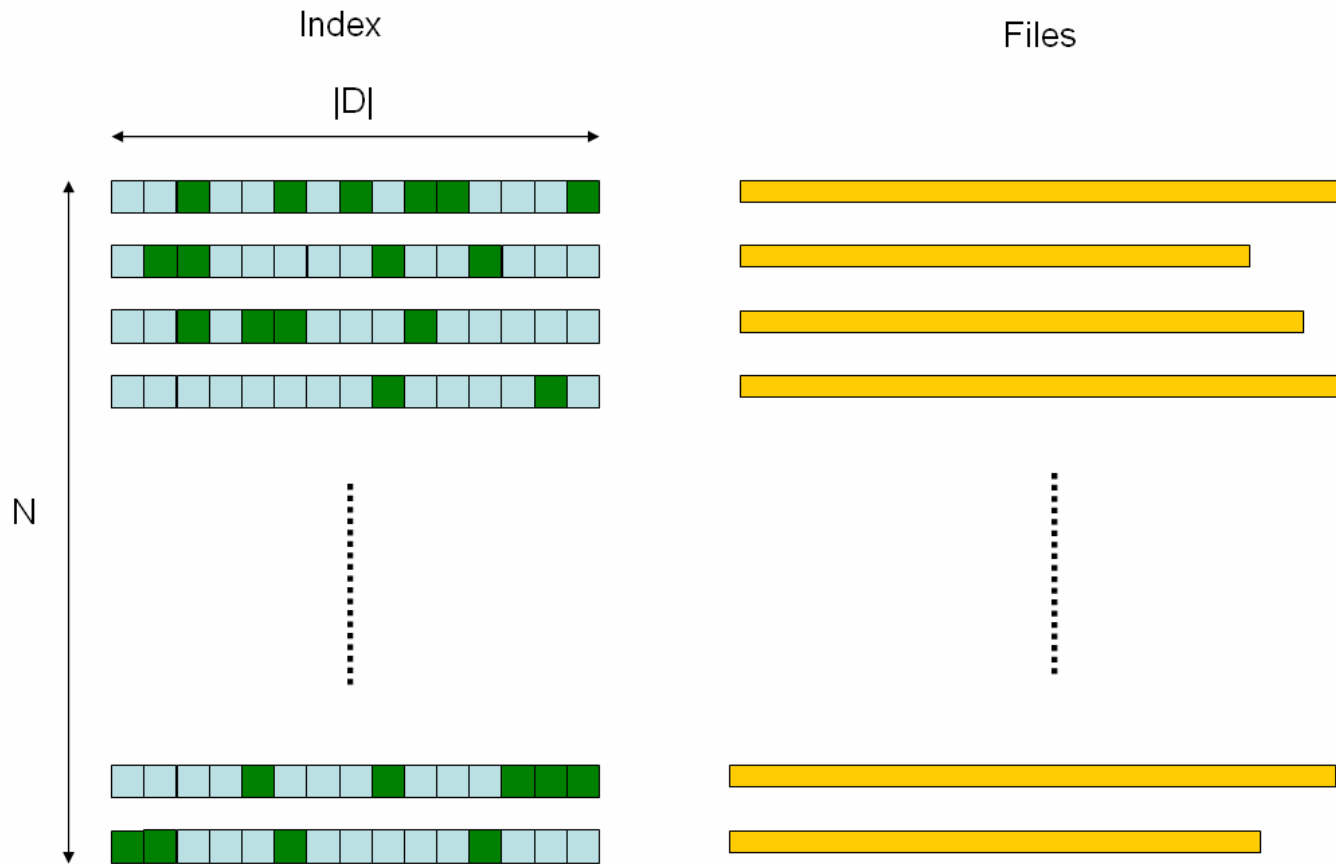
Yes \Rightarrow match,

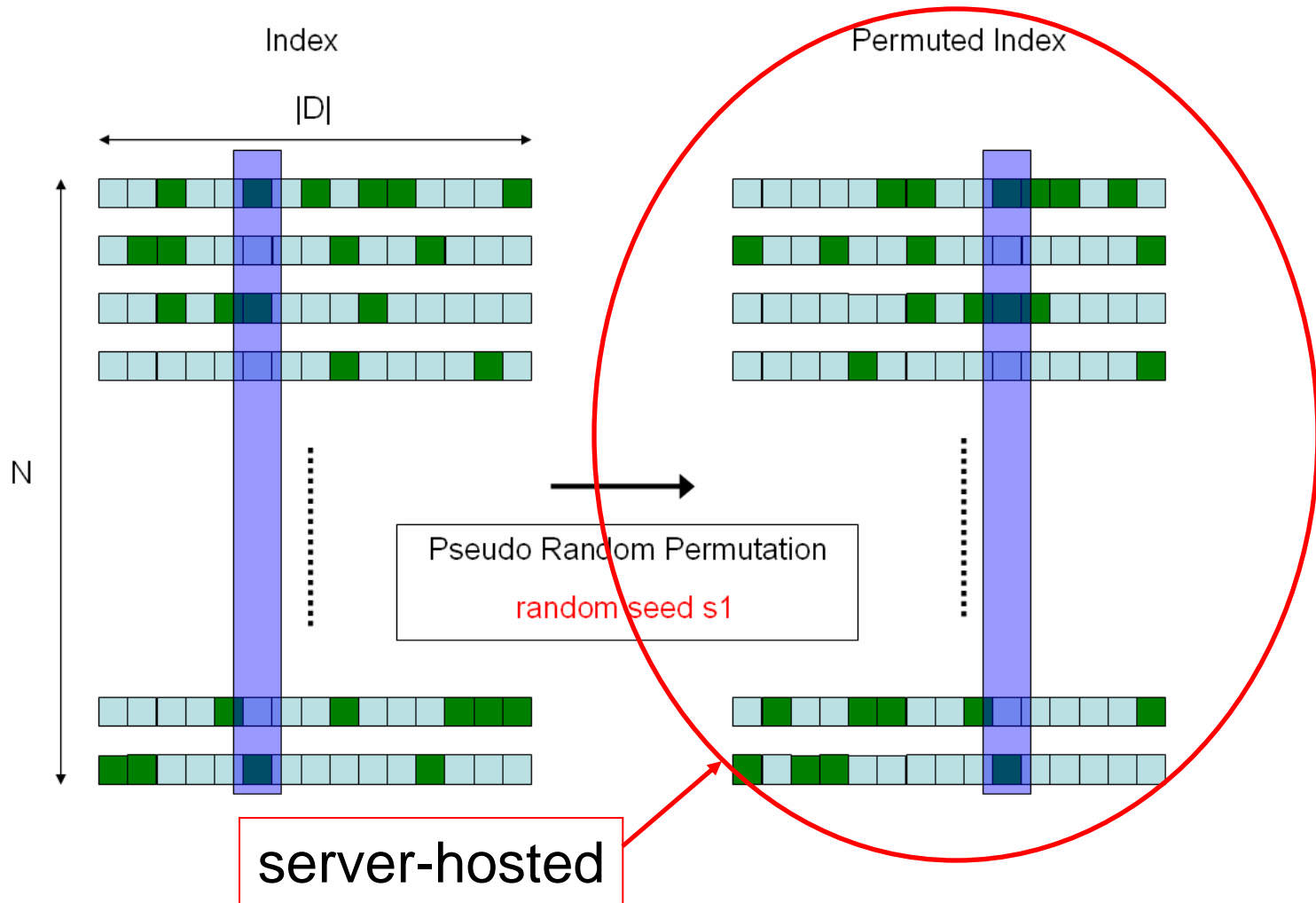
(false positive rate = $1 / 2^{m-n}$)

“Hidden” Search:

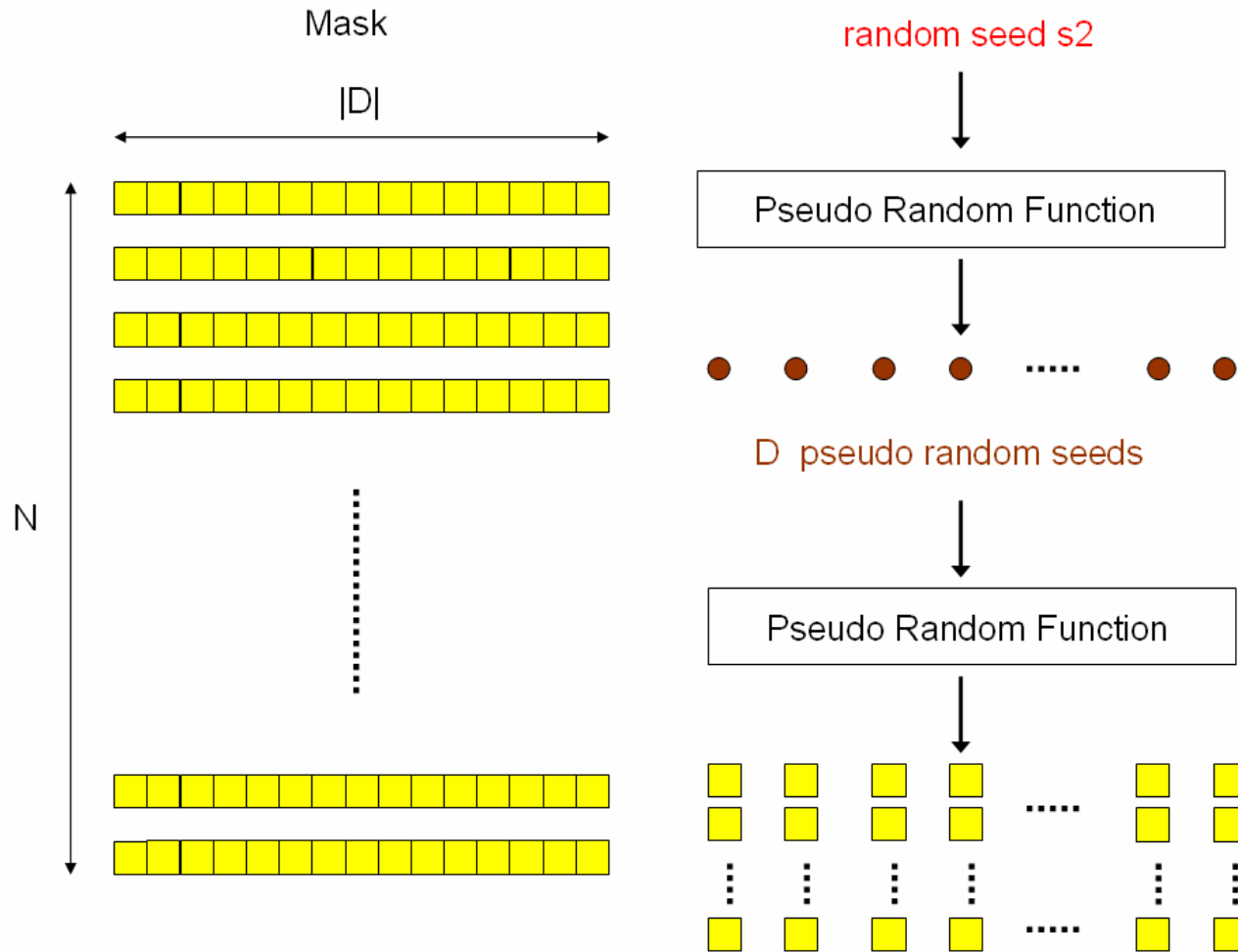


Chang (2004)

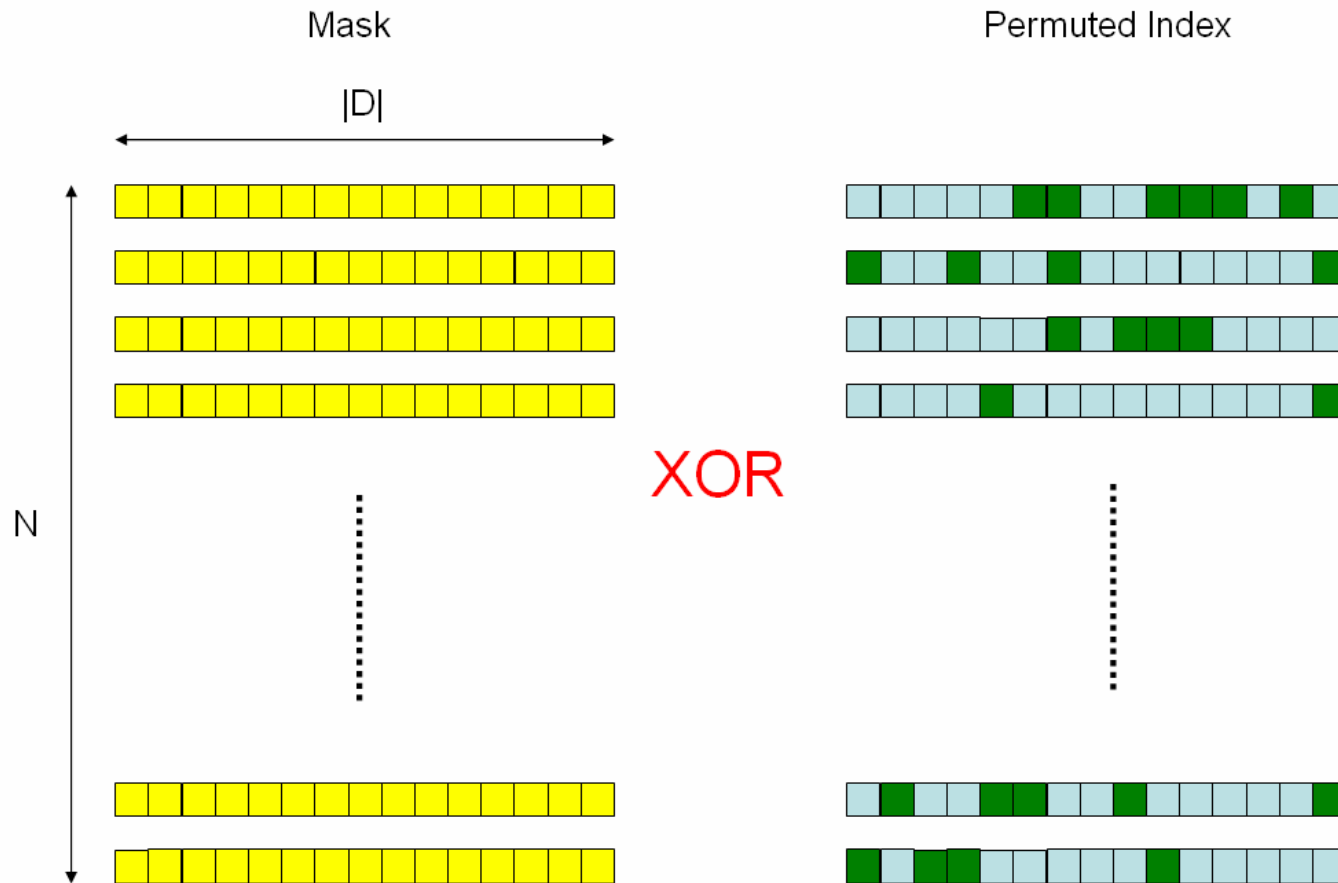


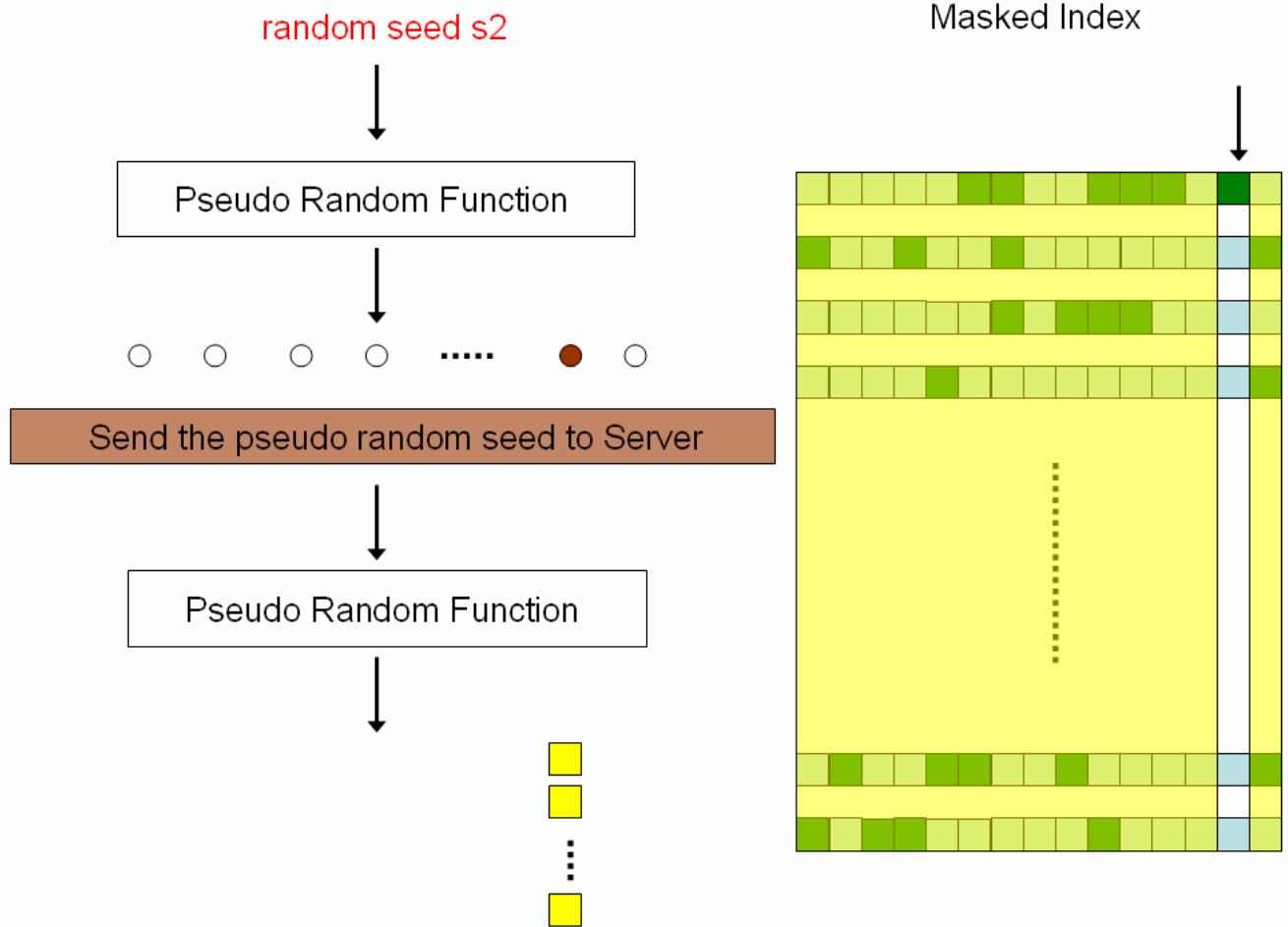


Chang (2004)



Chang (2004)





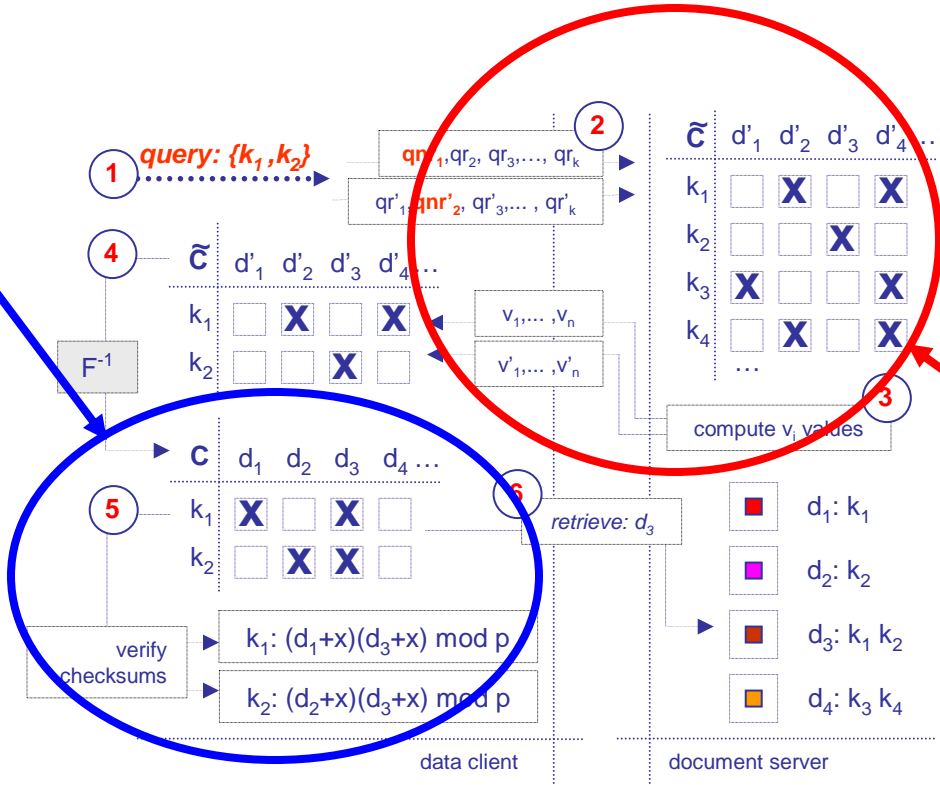
Server stores **capabilities** for conjunctive queries (linear in the total number of documents). These can be transferred offline.

The client is required to know before-hand future conjunctive queries.

Query part is sent online at the time of search. It is of constant size (number of keyword fields per documents).

Asks: What about correctness + privacy ?

Query Correctness



Computational Privacy

Idea: Deploy modified version of computational PIR targeted at a server-side index. Augment with “multiplicative checksums”.