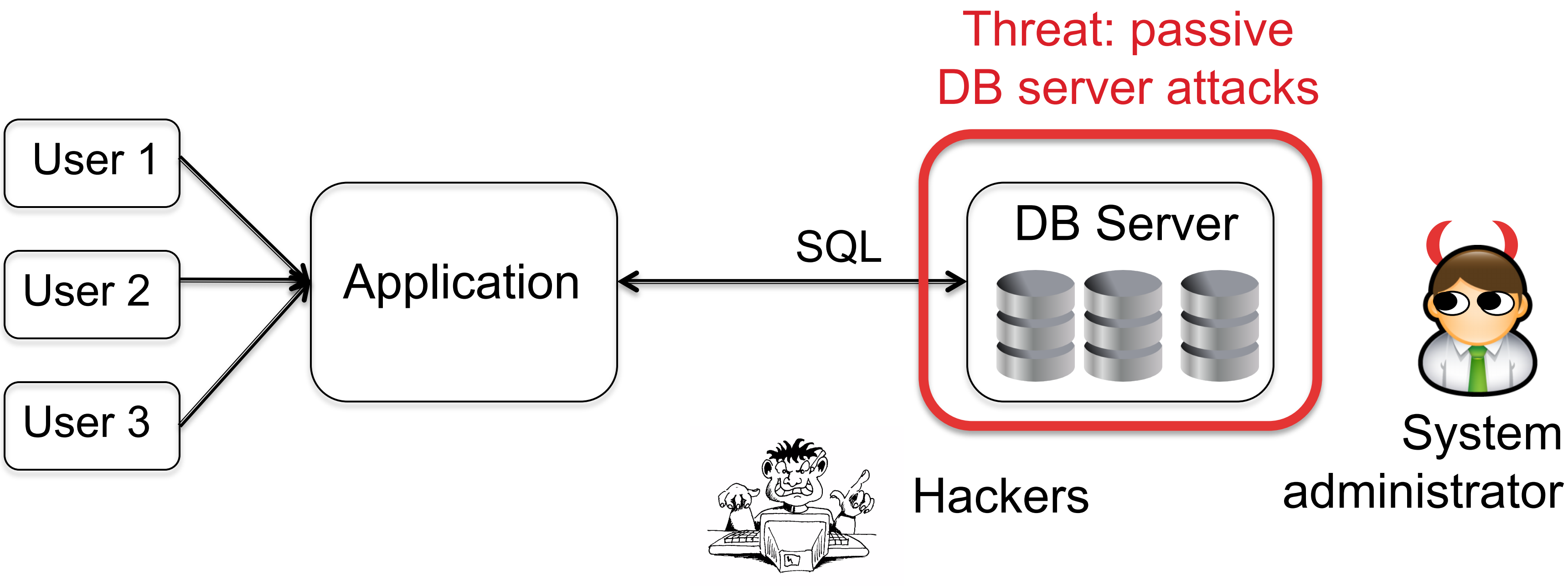
CryptDB: Processing Queries on an Encrypted Database

Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan
MIT CSAIL

Problem

- Confidential data leaks from databases (DB)
  - 2012: hackers extracted 6.5 million hashed passwords from the DB of LinkedIn



Threat: passive DB server attacks

User 1

User 2

User 3

Application

SQL

DB Server

Hackers

System administrator

- Process SQL queries on encrypted data

Contributions

1. First *practical* DBMS to process most SQL queries on encrypted data

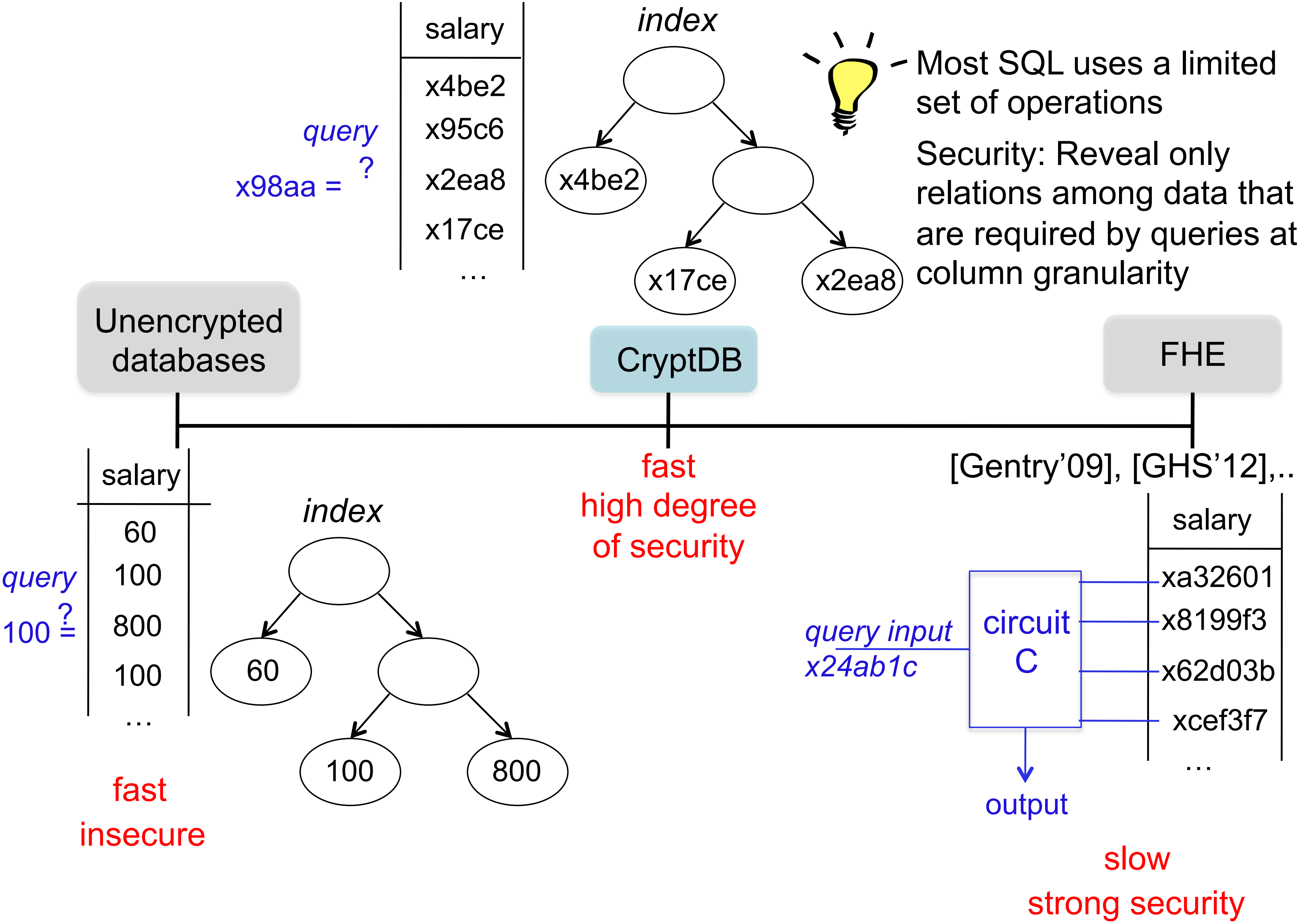
   ⇨ Hide DB from sys. admins., outsource DB to the cloud

2. Modest overhead: 26% throughput loss for TPC-C

3. No changes to DBMS (e.g., Postgres, MySQL) and no changes to applications

Most SQL uses a limited set of operations

Security: Reveal only relations among data that are required by queries at column granularity



Unencrypted databases

CryptDB

FHE

[Gentry'09], [GHS'12],..

query

x98aa = ?

salary
x4be2
x95c6
x2ea8
x17ce
…

index

query

100 ≟

salary
60
100
800
100
…

index

fast

insecure

fast

high degree of security

query input

x24ab1c

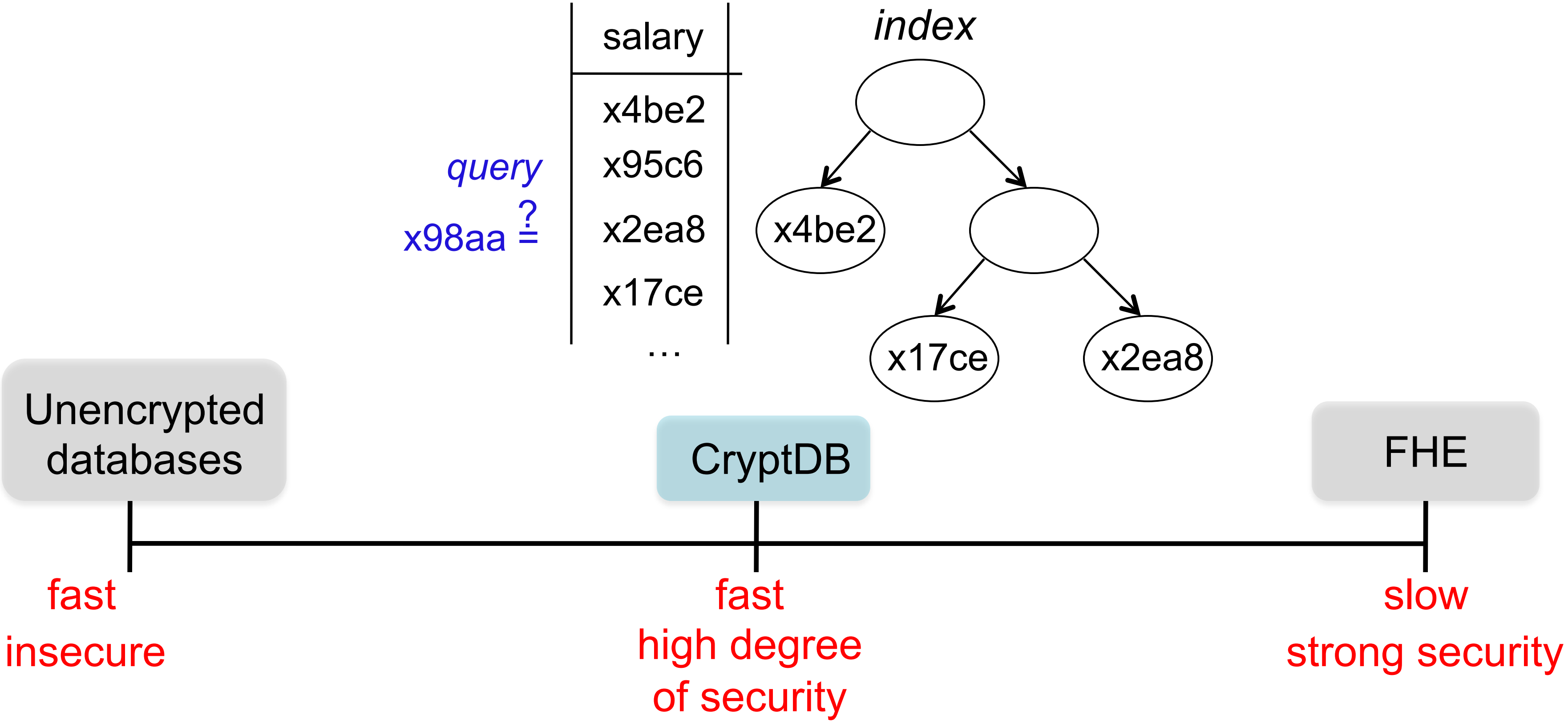
circuit C

output

salary
xa32601
x8199f3
x62d03b
xcef3f7
…

slow

strong security



salary

x4be2

x95c6

x2ea8

x17ce

…

query

x98aa $\overset{?}{=}$

index

x4be2

x17ce

x2ea8

Unencrypted databases

CryptDB

FHE

fast

insecure

fast

high degree of security

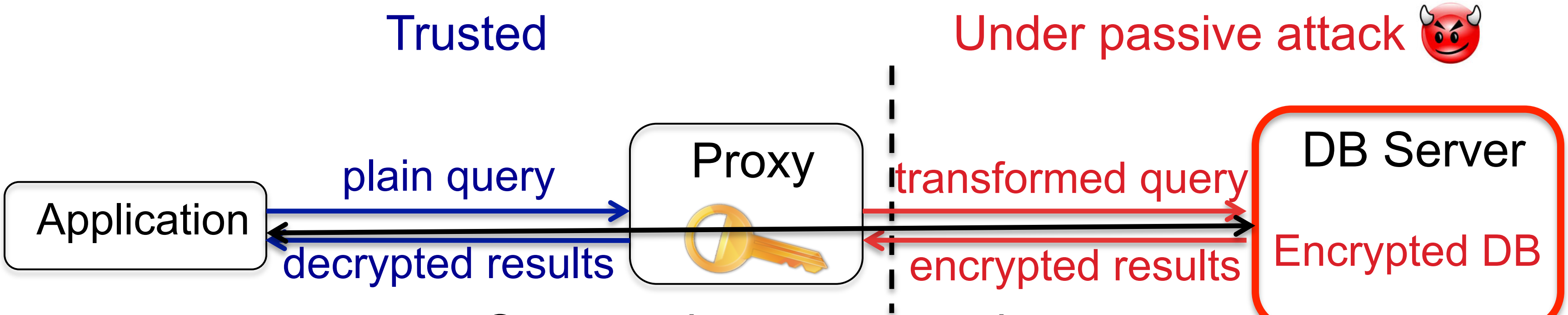
slow

strong security

Other work: weaker security, functionality, and/or efficiency:

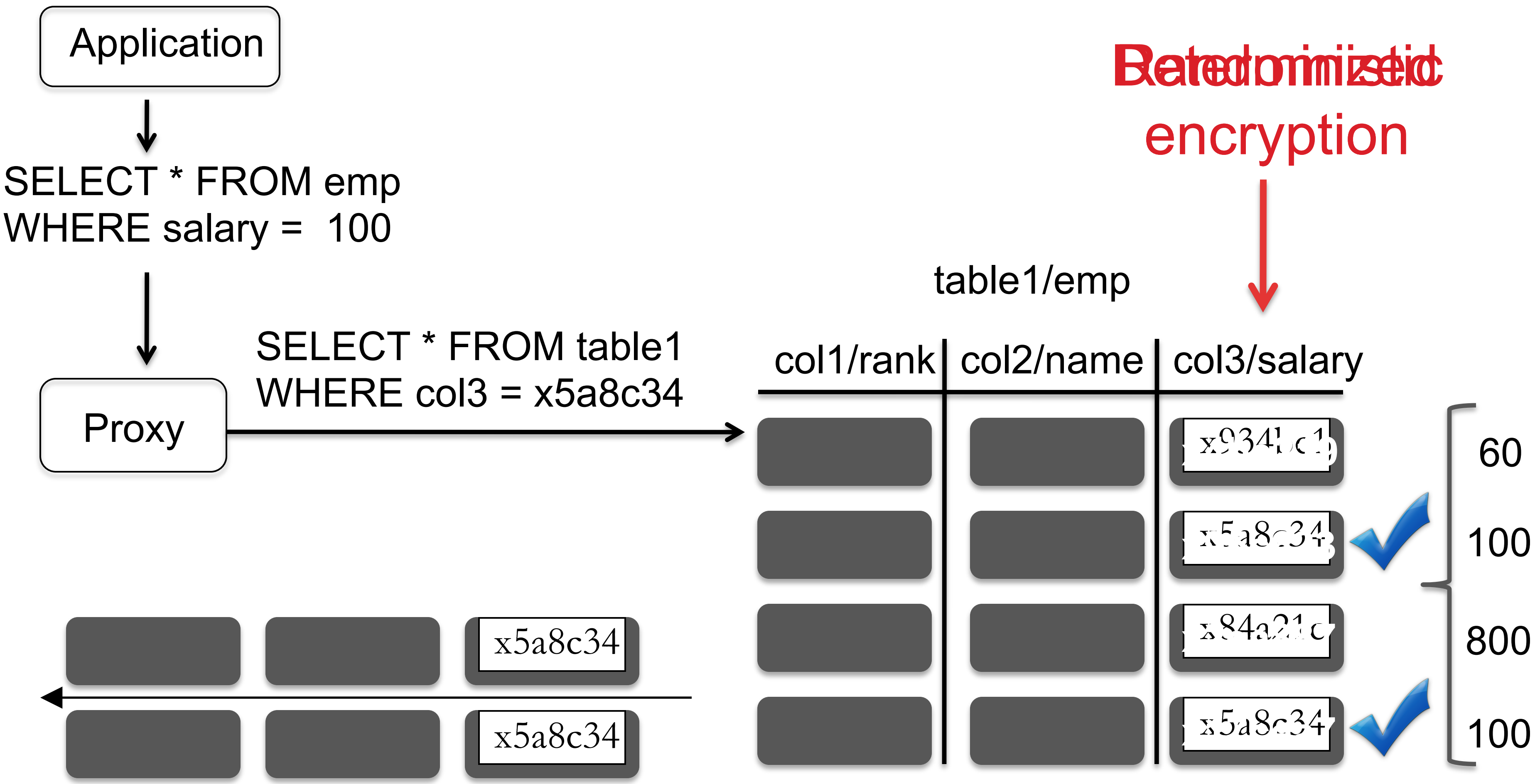
- Search on encrypted data (e.g., [Song et al.,'00])
- Systems proposals (e.g., [Hacigumus et al.,'02])
  - Require significant client-side processing

System Setup



- Stores schema, master key
- No data storage
- No query execution

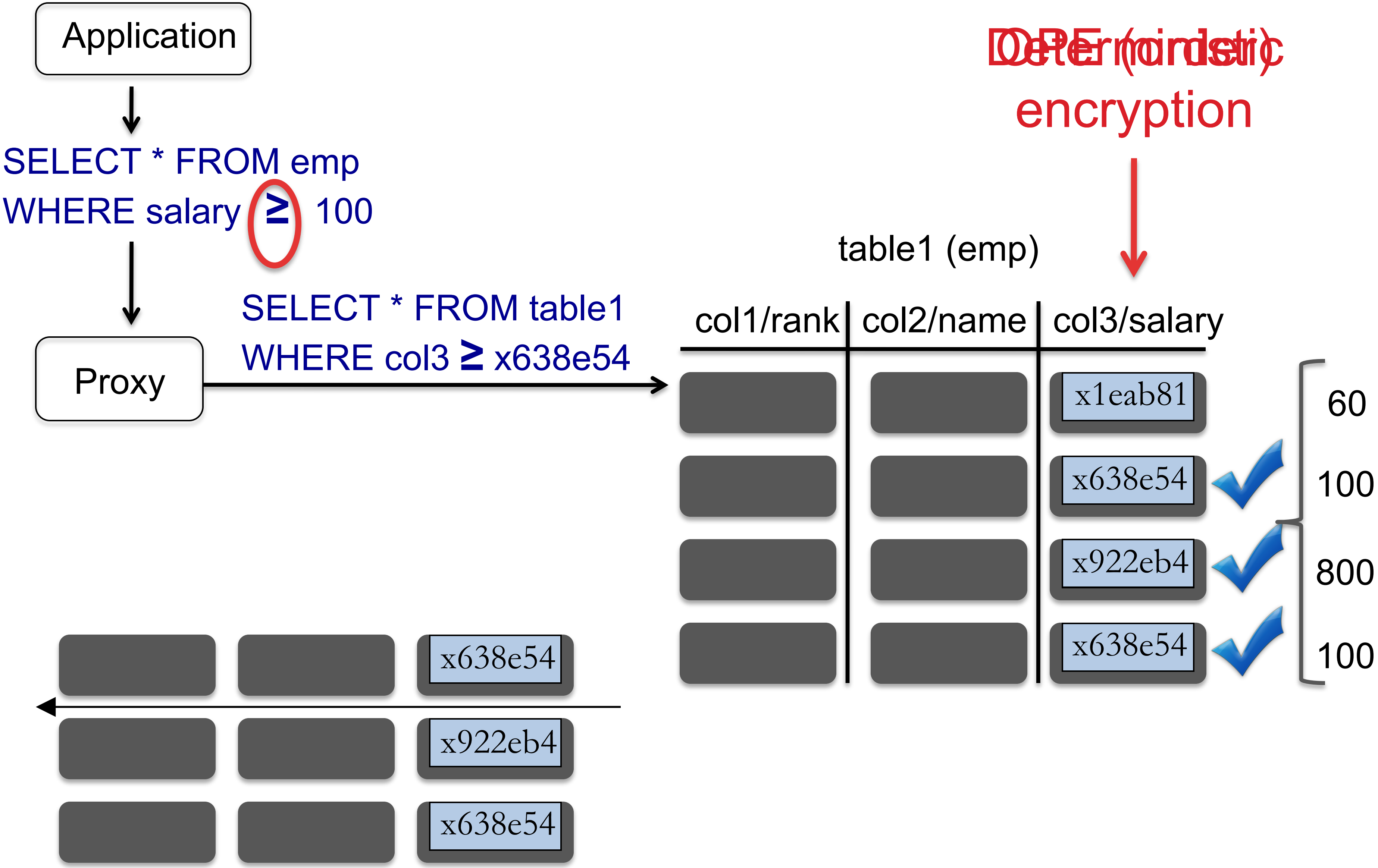
- Process queries completely at the DBMS, on encrypted database


Application
SELECT * FROM emp
WHERE salary = 100
Proxy
SELECT * FROM table1
WHERE col3 = x5a8c34
Randomized
Deterministic
encryption
table1/emp
col1/rank
col2/name
col3/salary
x934bc1
x5a8c34
x84a21c
x5a8c34
60
100
800
100
✓
✓
x5a8c34
x5a8c34

Deterministic

OPE (order)

encryption



Application

SELECT * FROM emp
WHERE salary ≥ 100

Proxy

SELECT * FROM table1
WHERE col3 ≥ x638e54

table1 (emp)

col1/rank	col2/name	col3/salary	
		x1eab81	60
		x638e54 ✓	100
		x922eb4 ✓	800
		x638e54 ✓	100

		x638e54
		x922eb4
		x638e54

Two techniques

1. Use SQL-aware set of encryption schemes

2. Adjust encryption of database based on queries

Encryption schemes

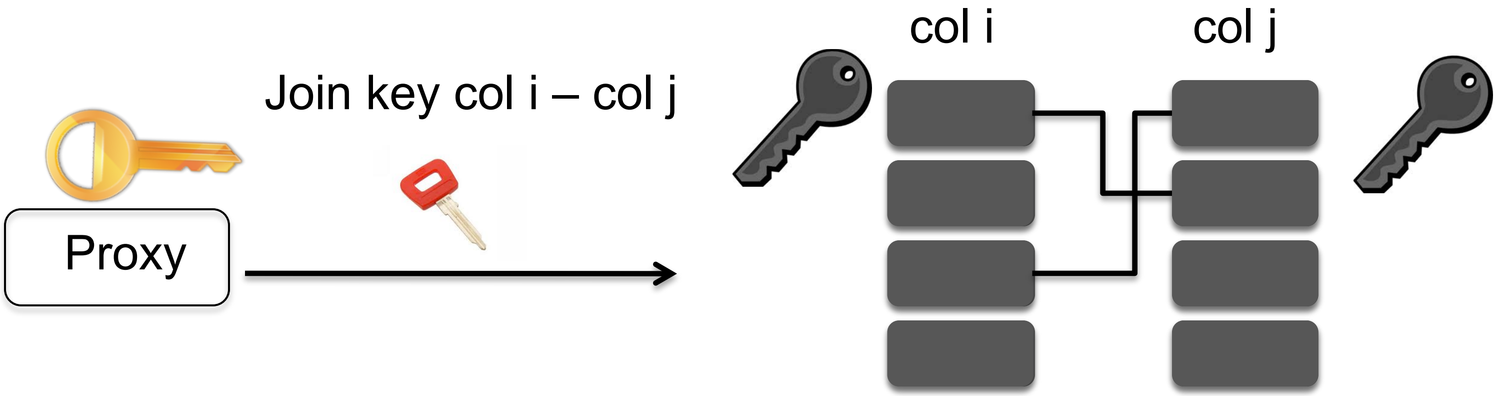
Highest

Security

Scheme	Construction	Function	
RND	AES in CBC	none	
HOM	Paillier	+	e.g., sum
SEARCH	Song et al.,‘00	word search	restricted ILIKE
DET	AES in CMC	equality	e.g., =, !=, IN, COUNT, GROUP BY, DISTINCT
JOIN	our new scheme	join	
OPE	BCLO’09 + our new scheme	order	e.g., >, <, ORDER BY, SORT, MAX, MIN, GREATEST

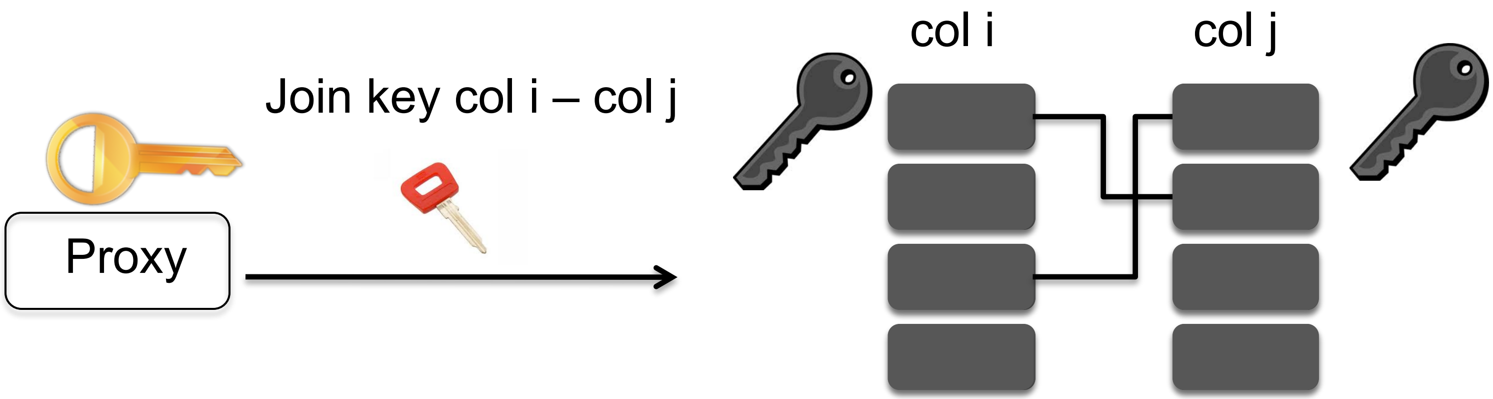
JOIN

- Do not know columns to be joined a priori!



- *KeyGen* (sec. param): SK 🔑
- *Encrypt* (SK, m, col i): C_m^i (with 🔑) - deterministic
- *Token* (SK, col i, col j): (t_i, t_j) 🔑
- *Adjust* (t_i, C_m^i): C_m (with 🔑)

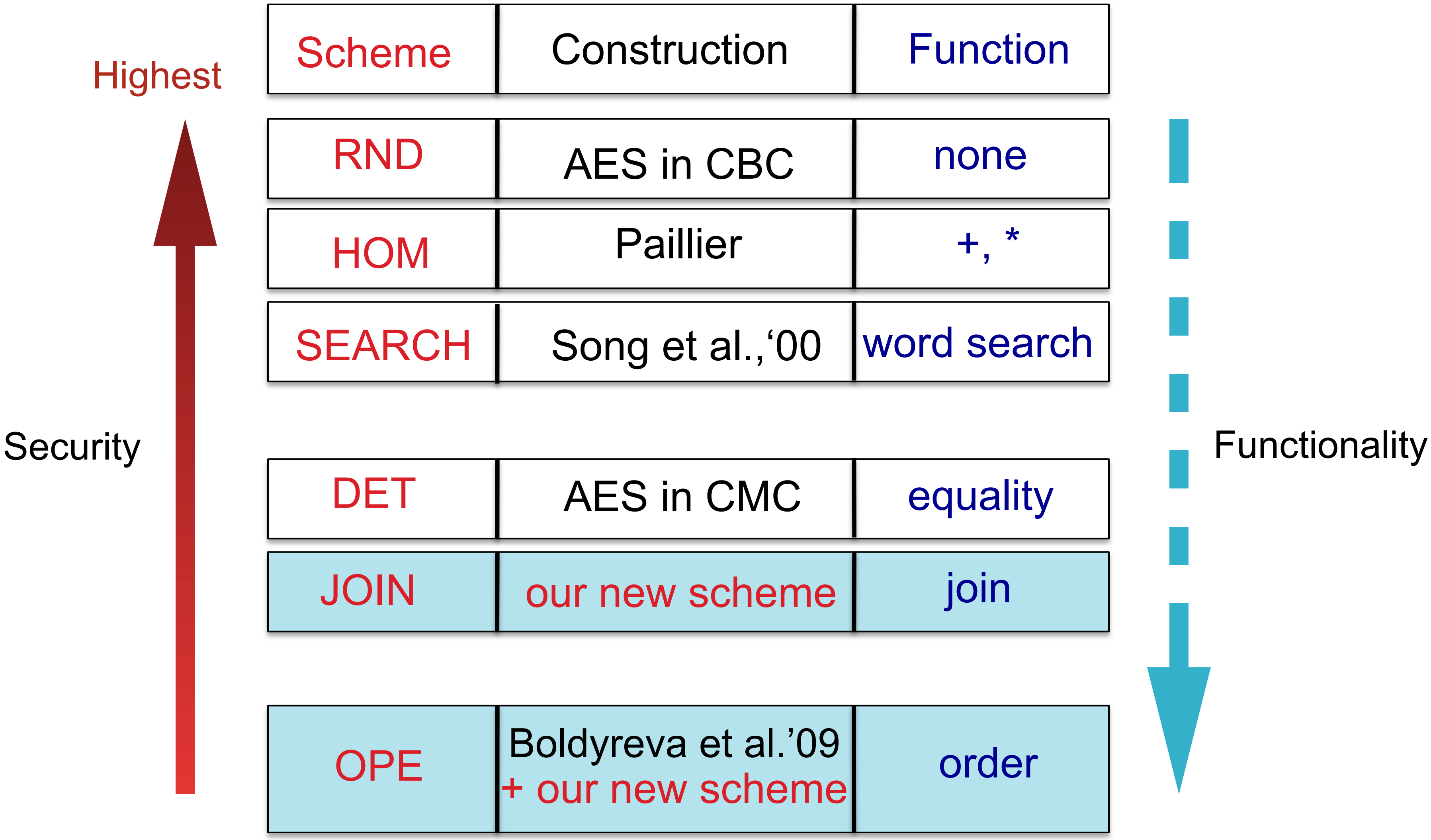
JOIN (cont'd)



- Security: do not learn join relations without token

- Implementation:
  - 192 bits long, 0.52 ms encrypt, 0.56 ms adjust

Encryption schemes



Highest

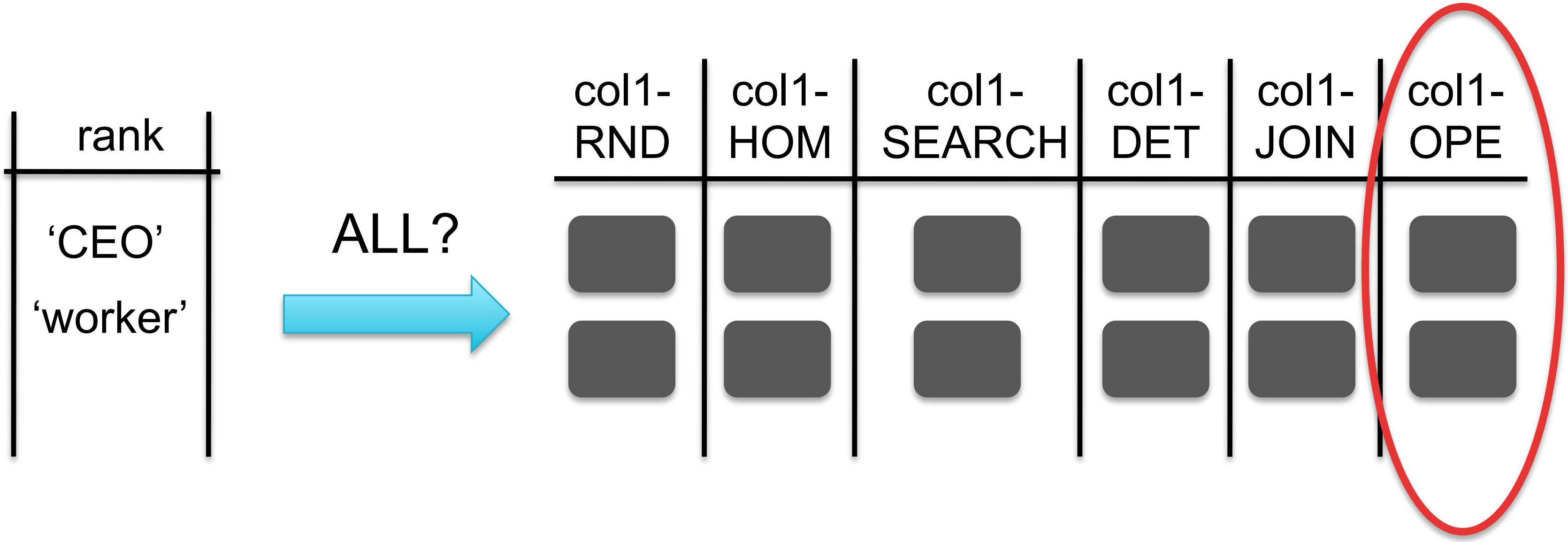
Security

Scheme	Construction	Function
RND	AES in CBC	none
HOM	Paillier	+, *
SEARCH	Song et al.,‘00	word search
DET	AES in CMC	equality
JOIN	our new scheme	join
OPE	Boldyreva et al.’09 + our new scheme	order

Functionality

How to encrypt each data item?

- Encryption schemes needed depend on queries
- May not know queries ahead of time



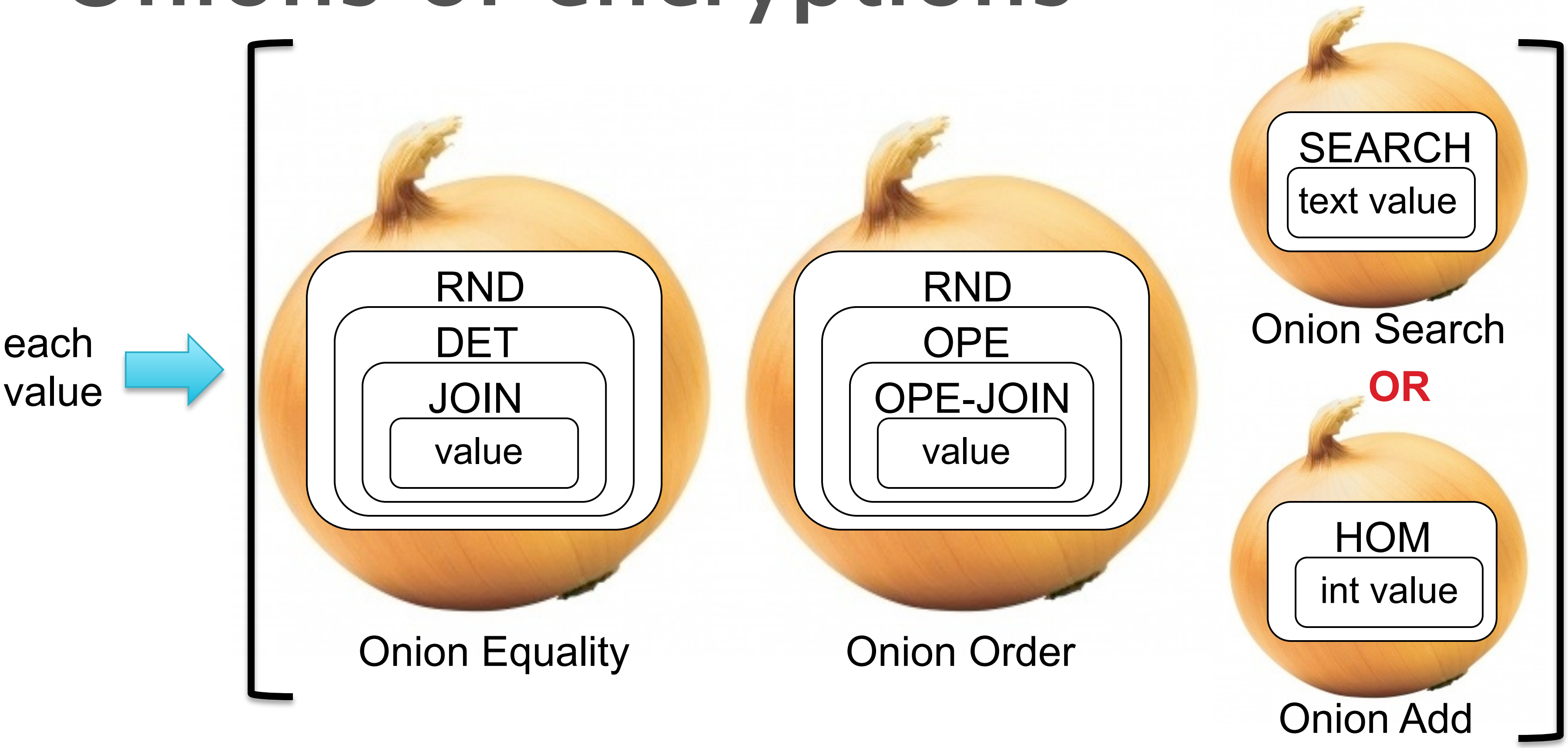
rank
‘CEO’
‘worker’

ALL?

col1-RND | col1-HOM | col1-SEARCH | col1-DET | col1-JOIN | col1-OPE

Leaks order!

Onions of encryptions

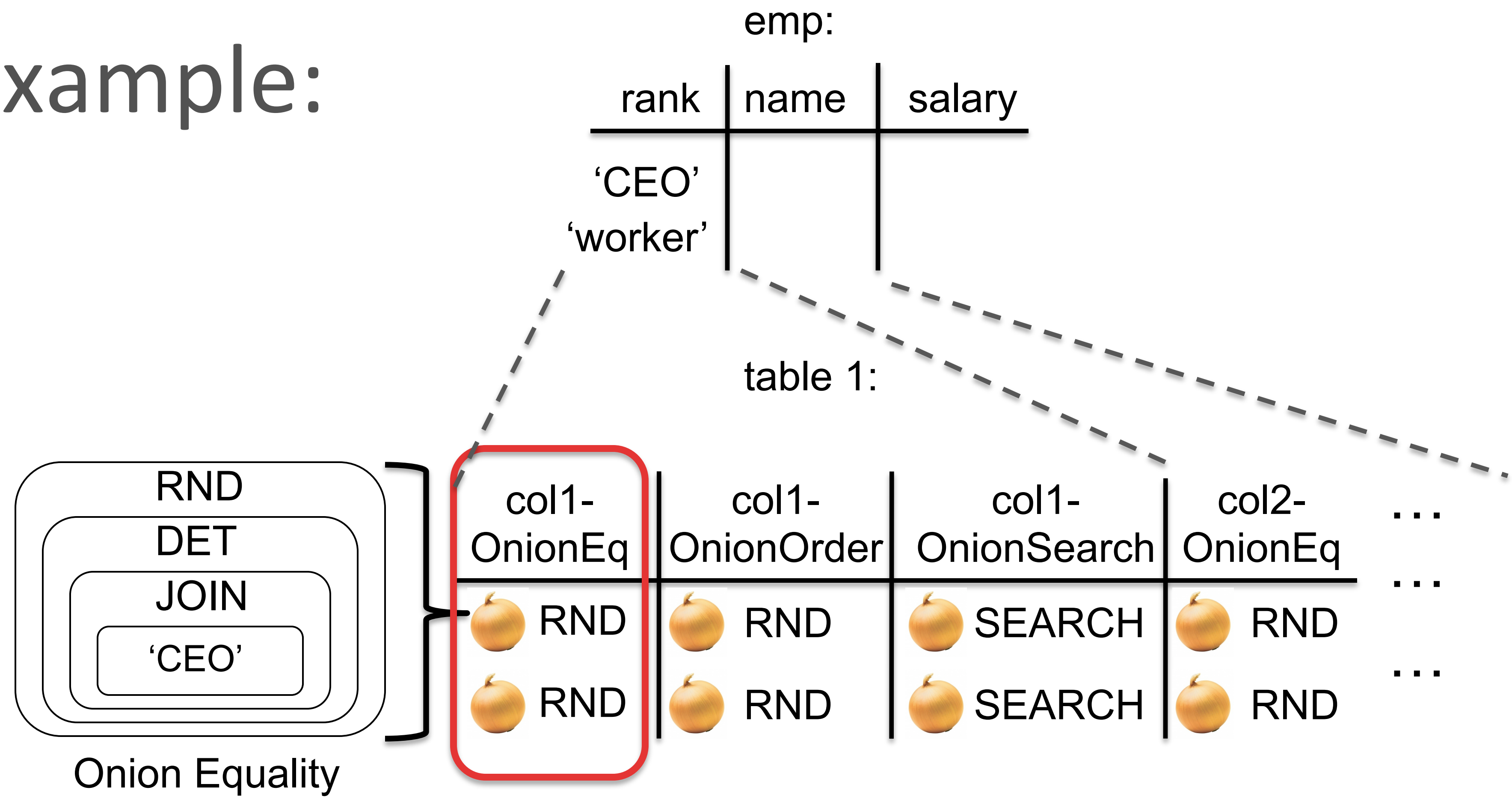


- Same key for all items in a column for same onion layer
- Start out the database with the most secure encryption scheme

Adjust encryption

- Strip off layers of the onions
  - Proxy gives keys to server using a SQL UDF (“user-defined function”)
  - Proxy remembers onion layer for columns
- Do not put back onion layer

Example:



emp:

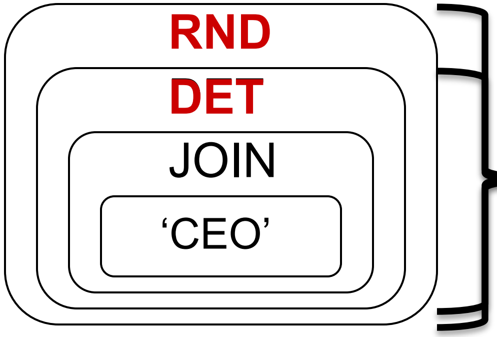
rank	name	salary
'CEO'		
'worker'		

table 1:

col1-OnionEq	col1-OnionOrder	col1-OnionSearch	col2-OnionEq	…
RND	RND	SEARCH	RND	…
RND	RND	SEARCH	RND	…

SELECT * FROM emp WHERE rank = 'CEO';

Example (cont’d)



Onion Equality

table 1

col1-OnionEq	col1-OnionOrder	col1-OnionSearch	col2-OnionEq	…
RND DET	RND	SEARCH	RND	…
RND DET	RND	SEARCH	RND	…

SELECT * FROM emp WHERE rank = ‘CEO’;

UPDATE table1 SET col1-OnionEq =
Decrypt_RND(key, col1-OnionEq);

SELECT * FROM table1 WHERE col1-OnionEq = xda5c0407;

Security guarantees

Queries → encryption schemes → leakage

- Encryption schemes exposed for each column are the most secure enabling queries

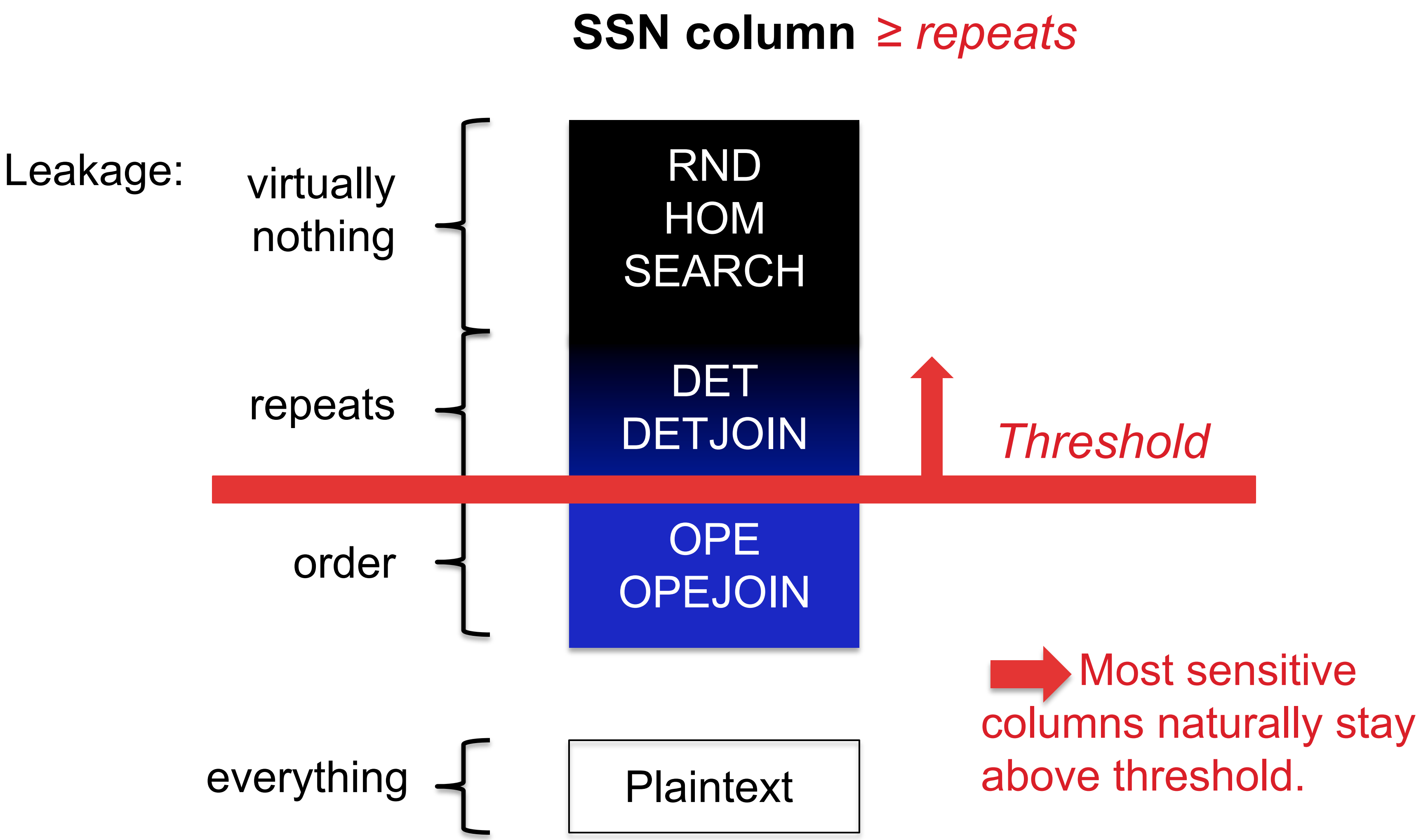
- Overall: Reveal only data relations needed for query type, at column granularity

  - equality predicate on a column → DET → repeats
  - aggregation on a column → HOM → nothing
  - no filter on a column → RND → nothing

common in practice

→ Never reveals plaintext

Security threshold



SSN column ≥ *repeats*

Leakage:

virtually nothing

RND
HOM
SEARCH

repeats

DET
DETJOIN

Threshold

order

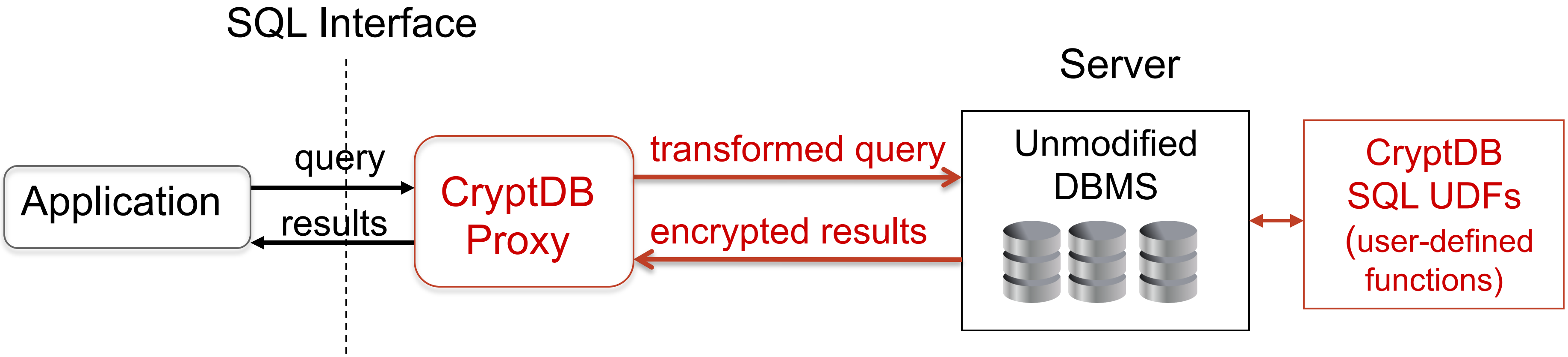
OPE
OPEJOIN

everything

Plaintext

Most sensitive columns naturally stay above threshold.

Implementation



- No change to the DBMS
- Portable: from Postgres to MySQL with 86 lines
- No change to applications

Evaluation

1. Does it support real queries/applications?
2. What is the resulting confidentiality?
3. What is the performance overhead?

Queries not supported

- More complex operators, e.g., trigonometry
- Operations that require combining encryption schemes
  - e.g., T1.a + T1.b > T2.c

⇨ Extensions: split queries, precompute columns, use FHE or other encryption schemes

Real queries/applications

Application	Total columns	Encrypted columns	# cols not supported
phpBB	563	23	0
HotCRP	204	22	0
grad-apply	706	103	0
TPC-C	92	92	0
sql.mit.edu	128,840	128,840	1,094

SELECT 1/log(series_no+1.2) …

… WHERE sin(latitude + PI()) …

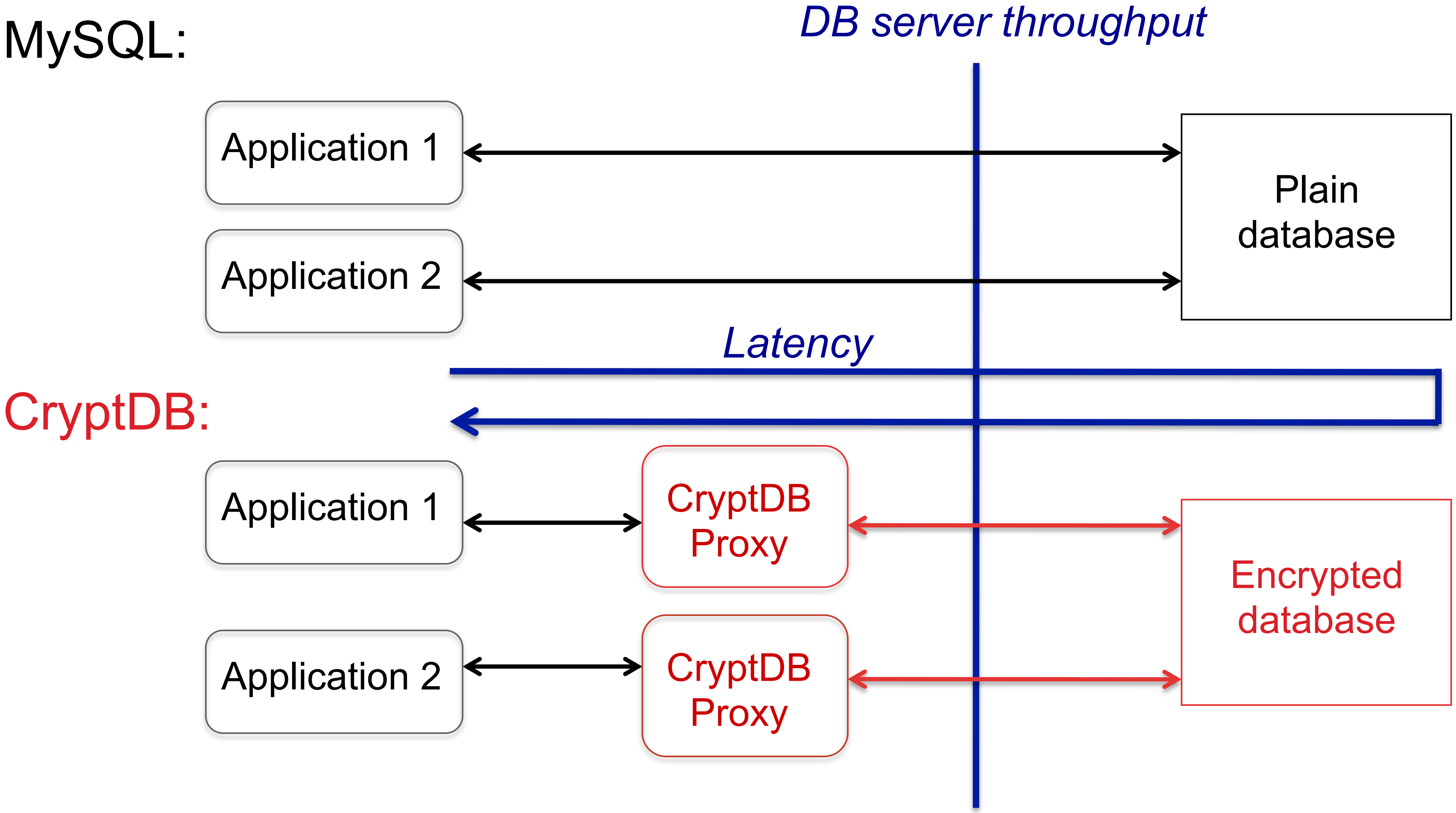
Resulting confidentiality

Application	Total columns	Encrypted columns	Min level is RND	Min level is DET	Min level is OPE
phpBB	563	23	21	1	1
HotCRP	204	22	18	1	2
grad-apply	706	103	95	6	2
TPC-C	92	92	65	19	8
sql.mit.edu	128,840	128,840	80,053	34,212	13,131

Most columns at RND

Most columns at OPE analyzed were less sensitive

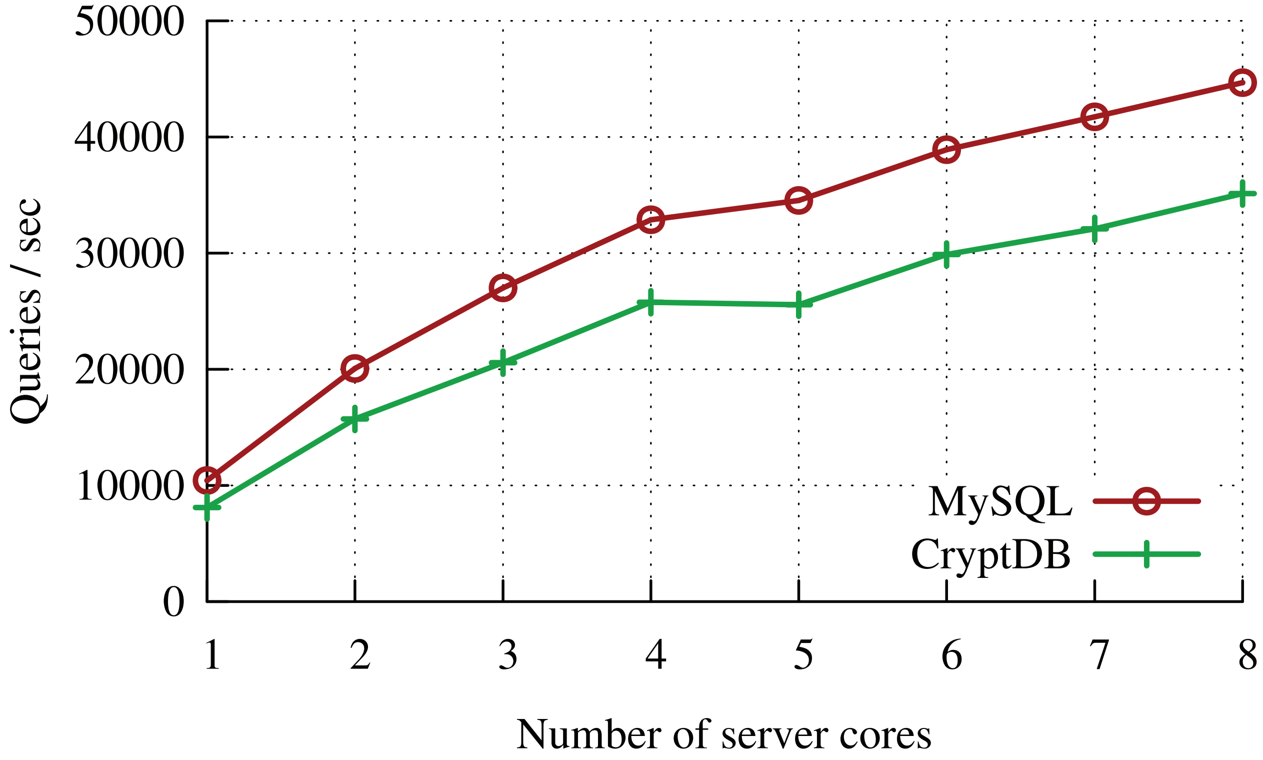
Performance



- Hardware: 2.4 GHz Intel Xeon E5620 – 8 cores, 12 GB RAM

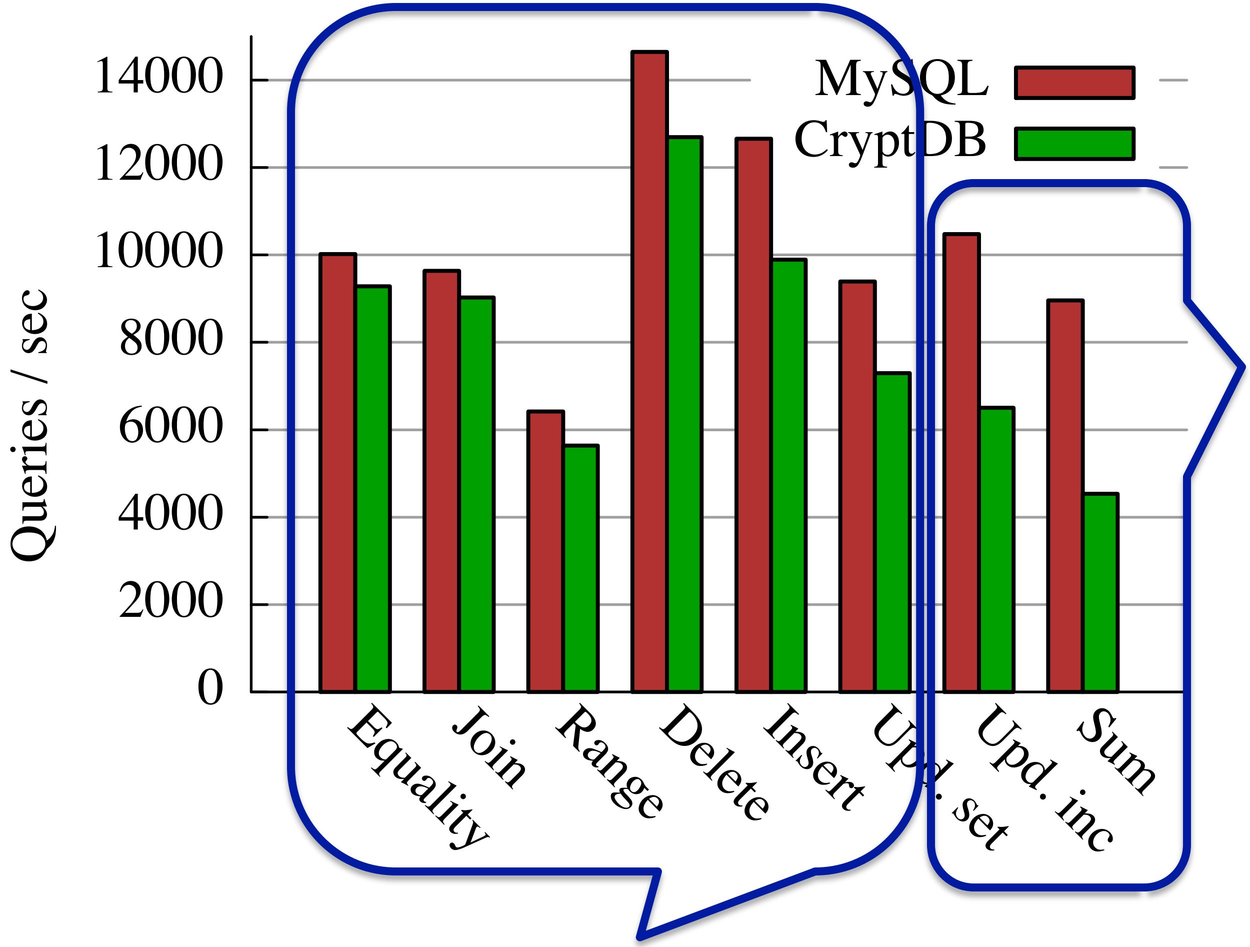
TPC-C performance

- Latency (ms/query): 0.10 MySQL vs. 0.72 CryptDB



Throughput loss 26%

TPC-C microbenchmarks



Homomorphic addition

No cryptography at the DB server in the steady state!

➡ CryptDB is practical

Demo

Conclusions

CryptDB:

1. The first practical DBMS for running most standard queries on encrypted data
2. Modest overhead and no changes to DBMS

Website: http://css.csail.mit.edu/cryptdb/

Thanks!