Building Privacy-Aware Database Systems

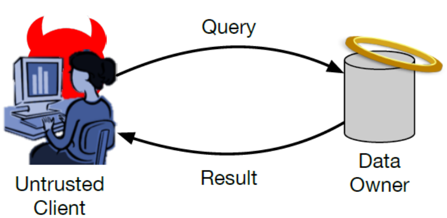
CS848 Winter 2021

Module 3

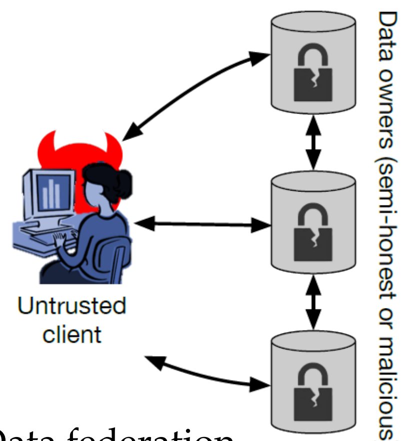
This course will explore …

- How to define a good privacy promise?
- How to design a privacy-preserving algorithms?
- How to build a privacy-aware database systems?

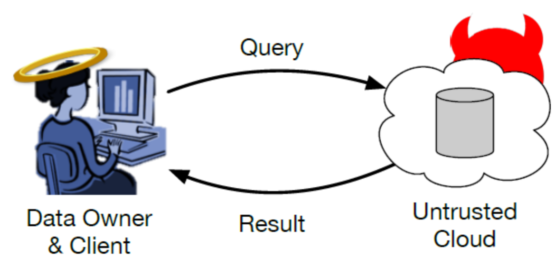
Greatly depend on
the architecture setup and trust assumptions



Client-server with trusted data curator



Data federation

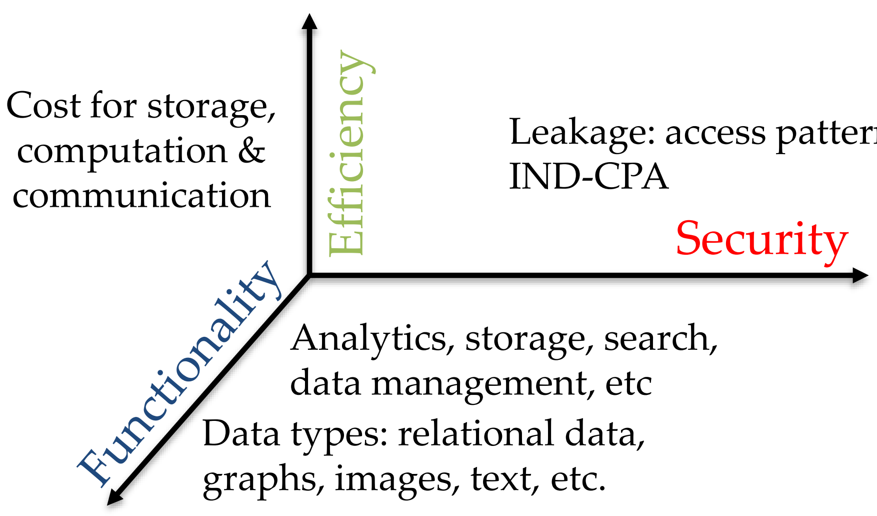


Cloud service provider

Cloud Setting

- But data leaks can happen
  - Hackers with illegal access
  - Snooping administrators

- Encrypted databases and evaluation metrics



Lecture Focus

- Functionality: SQL-aware
- Passive adversary
  - Honest but curious (HBC)
  - Does not alter database or results
- Desired functionality under encryption
  - Encrypted data stored, encrypted queries, and encrypted results
  - But that does not guarantee security

Lecture slides are mainly adapted from: "Querying Encrypted Data", Arasu et al. ICDE 2016

Outline

- Part I: Design Choices
- Part II: Security Guarantees
- Upcoming Papers and Announcements

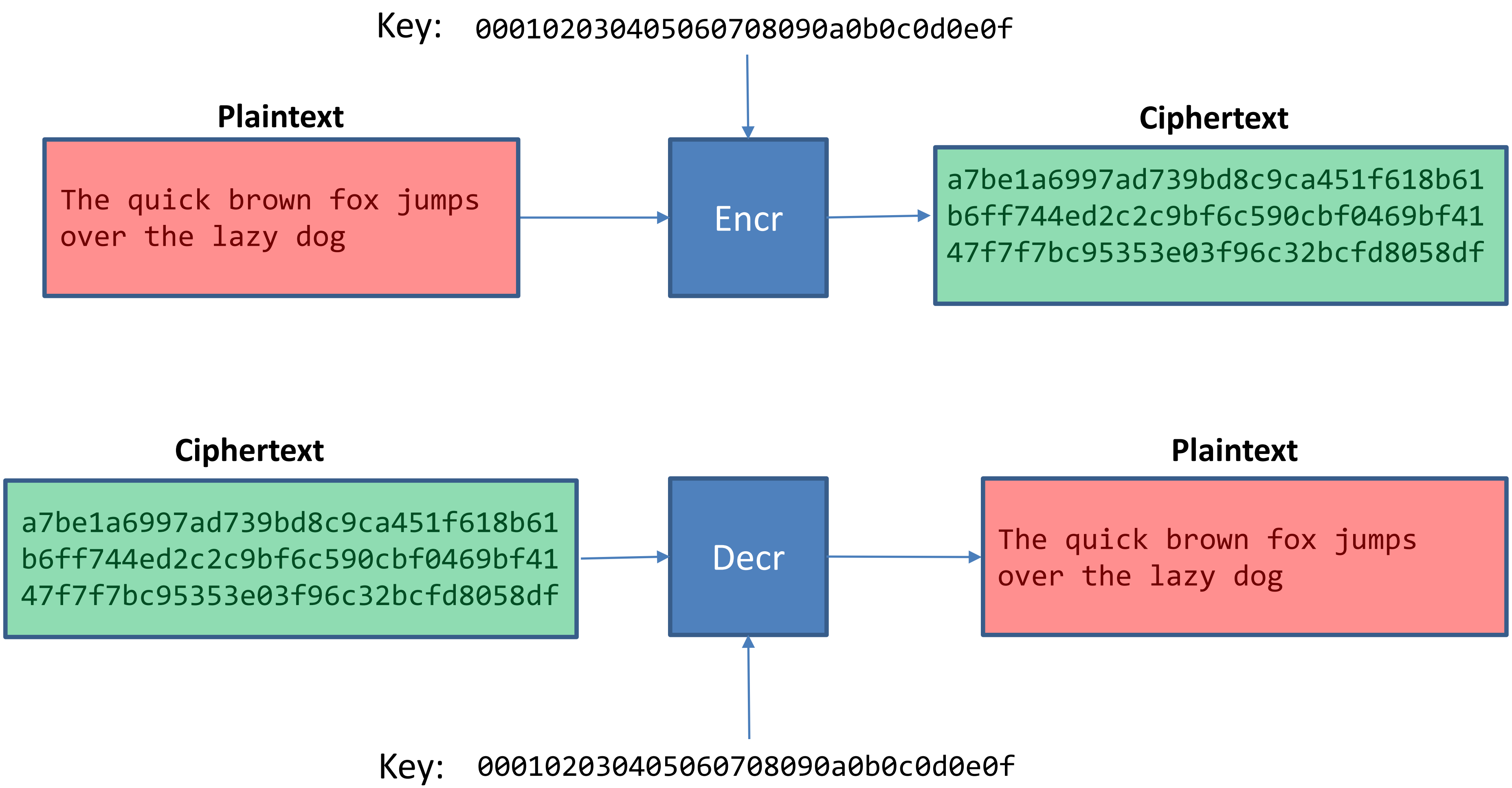
Outline

- Part I: Design Choices
  - Encryption basics
  - Trusted client-based systems
  - Secure in-cloud computing

- Part II: Security Guarantees

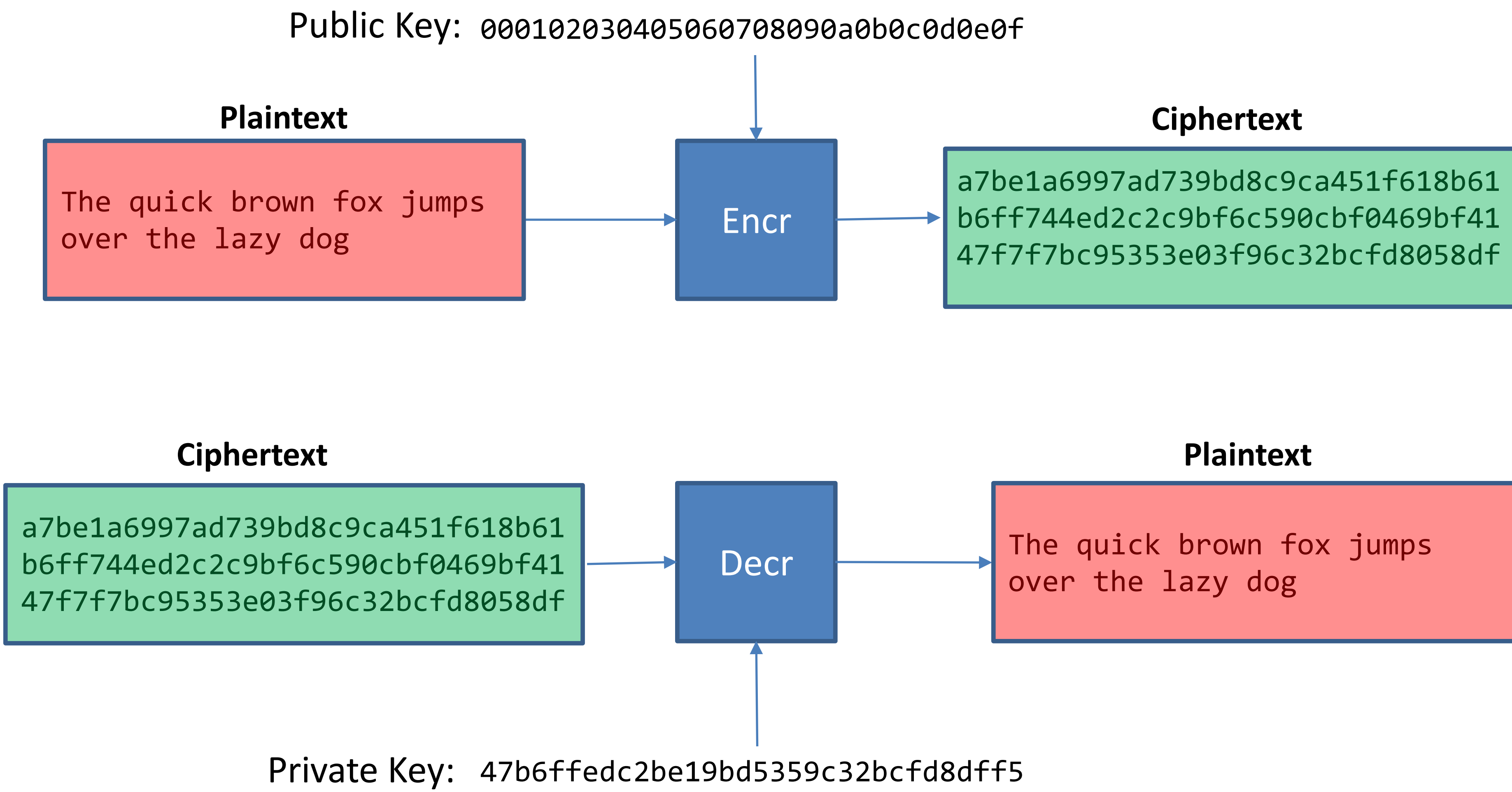
- Upcoming Papers and Announcements

Encryption Scheme



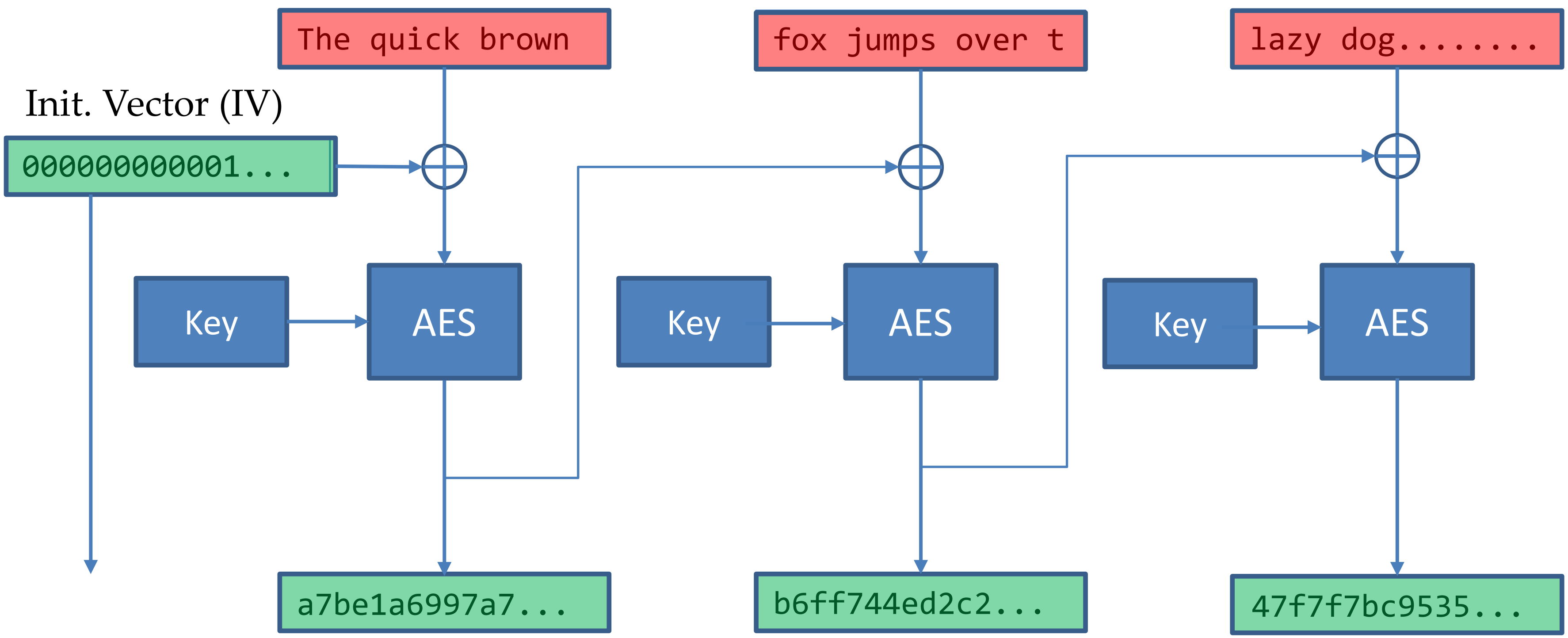
Crypto Textbook: [KL 07]

Encryption Scheme



Crypto Textbook: [KL 07]

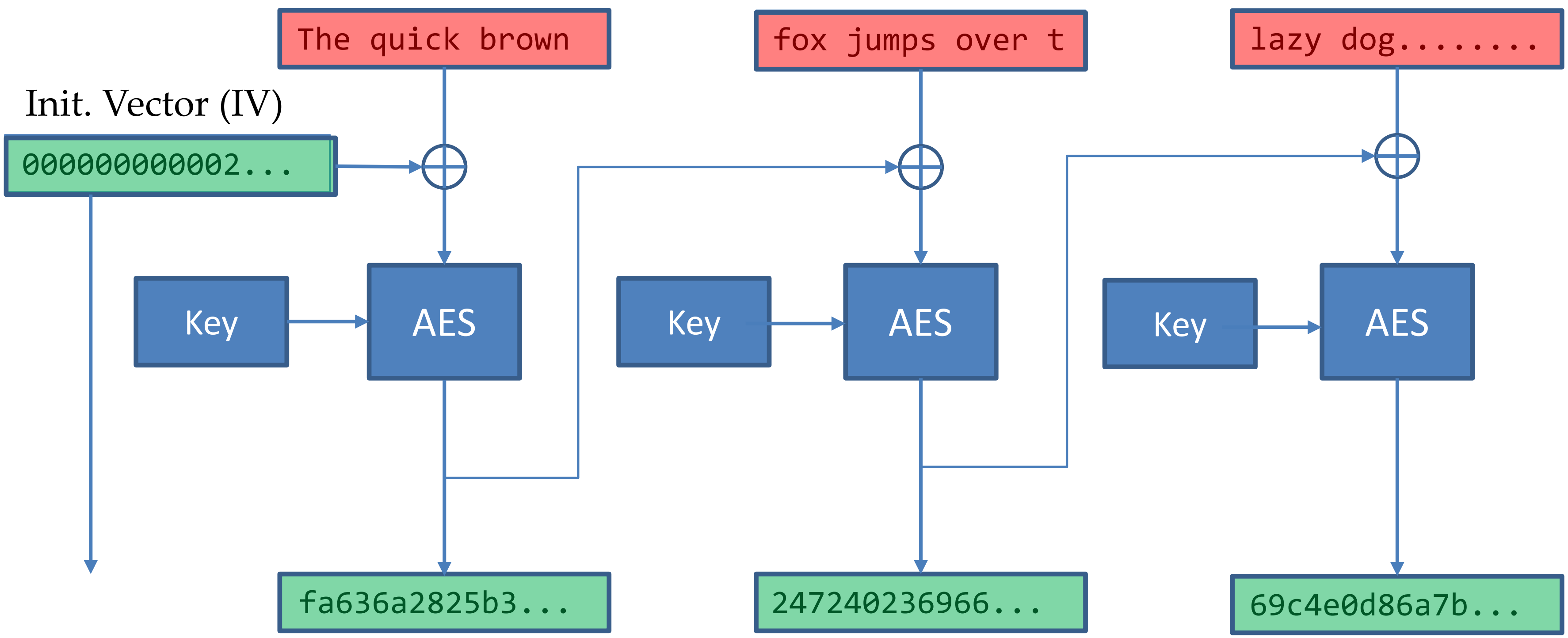
AES + CBC Mode



Variable IV => Non-deterministic

[AES, KL 07]

AES + CBC Mode



Variable IV => Non-deterministic

[AES, KL 07]

Nondeterministic Encryption Scheme

Key: `000102030405060708090a0b0c0d0e0f`

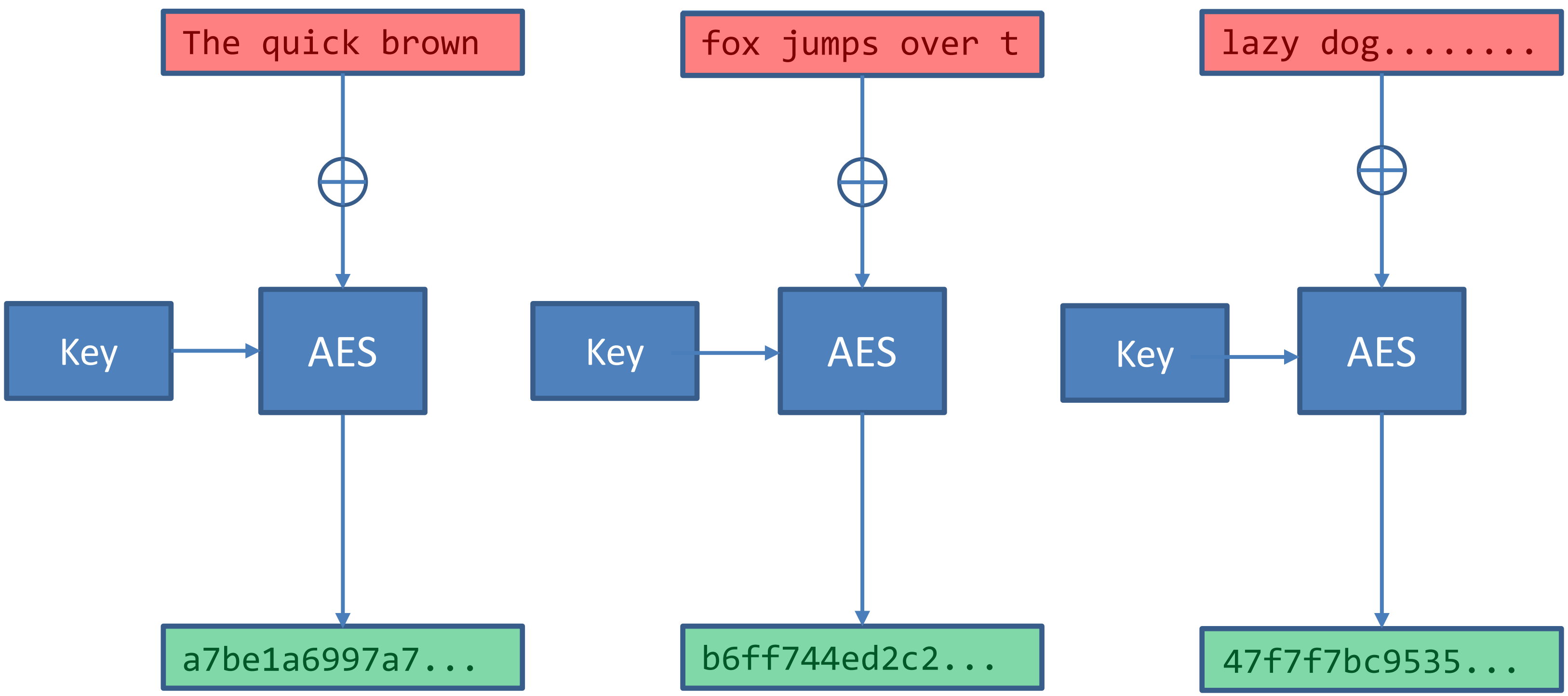
`The quick brown fox jumps over the lazy dog` → **Encr** → `a7be1a6997ad739bd8c9ca451f618b61b6ff744ed2c2c9bf6c590cbf0469bf4147f7f7bc95353e03f96c32bcfd8058df`

`000102030405060708090a0b0c0d0e0f`

`The quick brown fox jumps over the lazy dog` → **Encr** → `fa636a2825b339c940668a3157244d1724724023696 6b3fa6ed2753288425b6c69c4e0d86a7b0430d8cdb78070b4c55a`

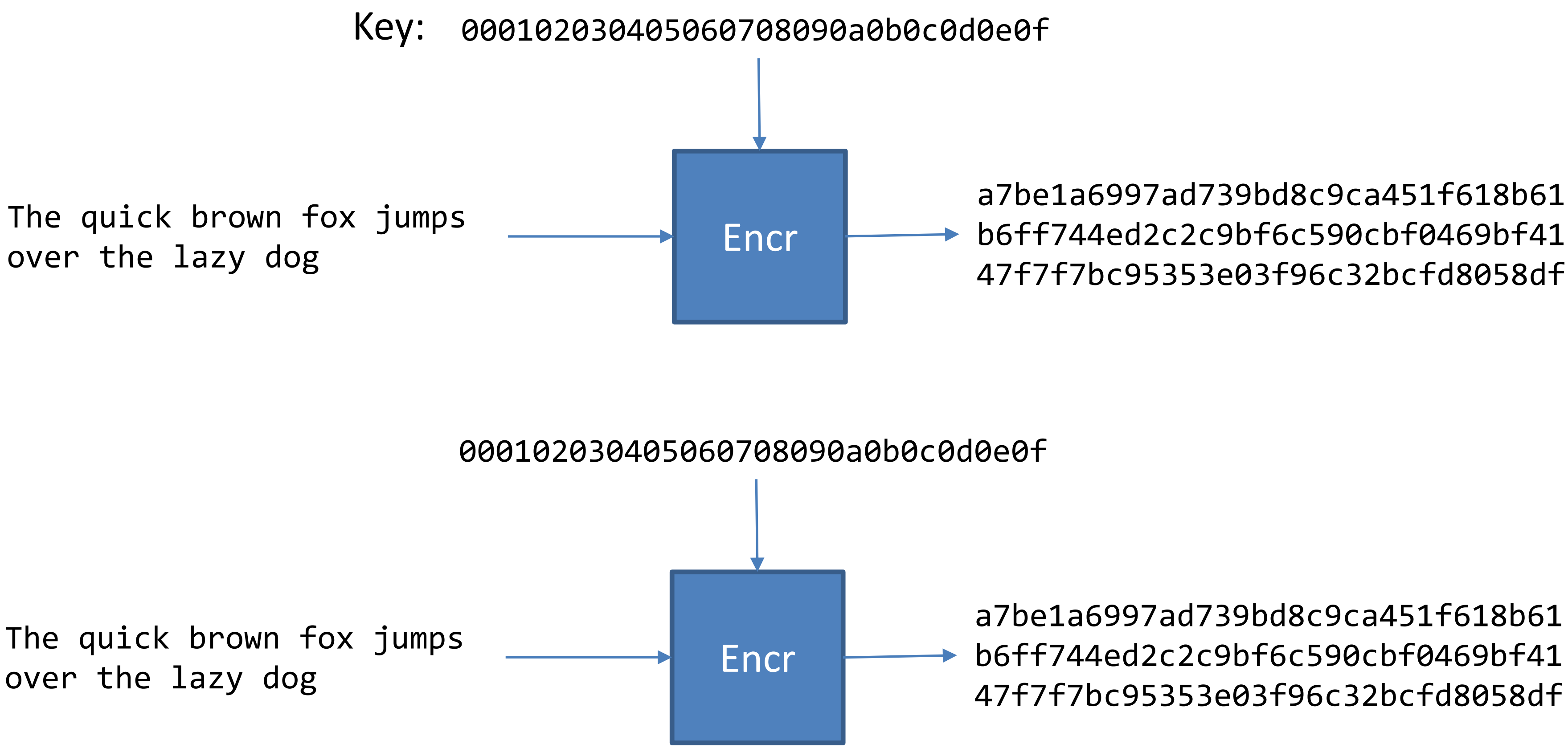
Example: AES + CBC + variable IV

AES + ECB Mode



[AES, KL 07]

Deterministic Encryption Scheme

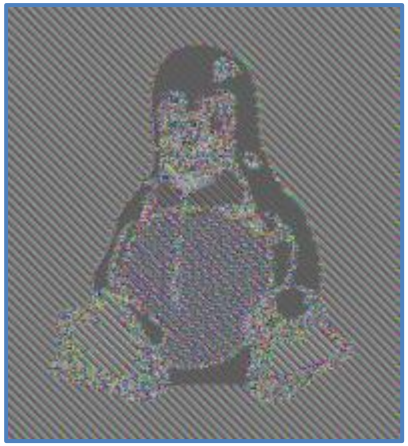


Example: AES + ECB

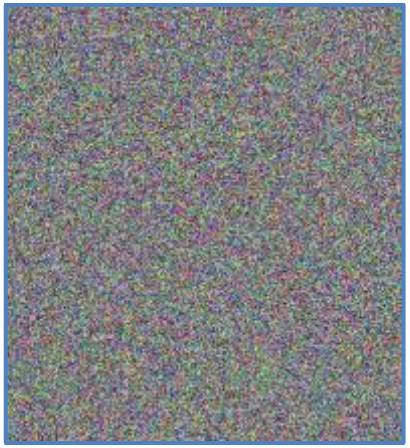
More secure deterministic encryption: [PRZ+11]

Strong Security => Non-Deterministic

Original



Deterministic



Non-Deterministic

Source: http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Deterministic Encryption

```
select *
from assignment
where studentid = 1
```

$$\sigma_{StudentId=1}$$

StudentId	AssignId	Score
1	1	68
1	2	71
3	4	99
…	…	…

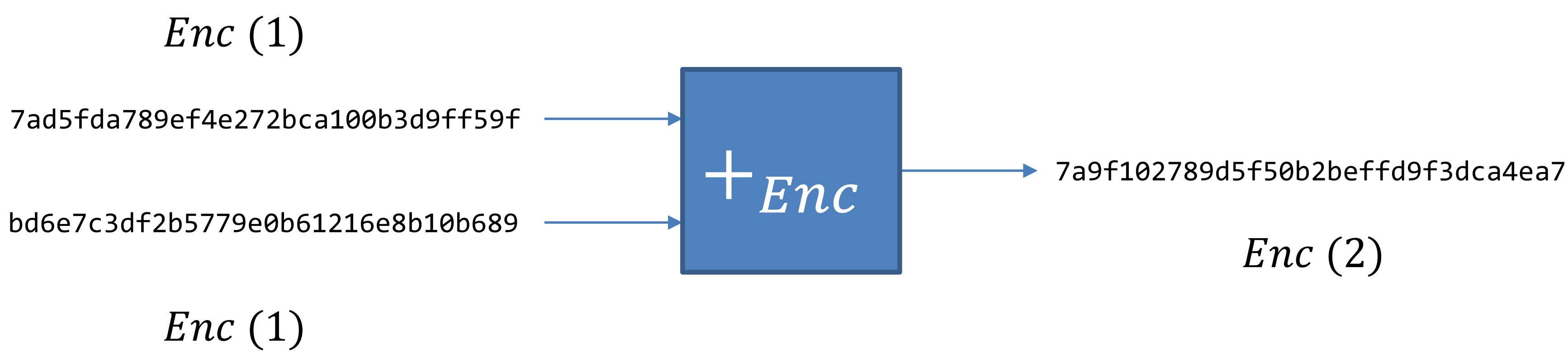
Deterministic Encryption

```sql
select *
from assignment
where studentid_det = bd6e7c3df2b5779e0b61216e8b10b689
```

$\sigma_{StudentId\_det=bd6\ldots}$

StudentId_DET	AssignId	Score
bd6e7c3df2b5779e0b61216e8b10b689	1	68
bd6e7c3df2b5779e0b61216e8b10b689	2	71
7ad5fda789ef4e272bca100b3d9ff59f	4	99
…	…	…

Homomorphic Encryption



Encryption key is not an input

Order Preserving Encryption

Value	Enc (Value)
1	0x0001102789d5f50b2beffd9f3dca4ea7
2	0x0065fda789ef4e272bcf102787a93903
3	0x009b5708e13665a7de14d3d824ca9f15
4	0x04e062ff507458f9be50497656ed654c
5	0x08db34fb1f807678d3f833c2194a759e

$$x < y \rightarrow Enc\,(x) < Enc\,(y)$$

[BCN11, PLZ13]

Order-Preserving Encryption

```sql
select *
from assignment
where score >= 90
```

$\sigma_{Score \geq 90}$

StudentId	AssignId	Score
1	1	68
1	2	71
3	4	99
…	…	…

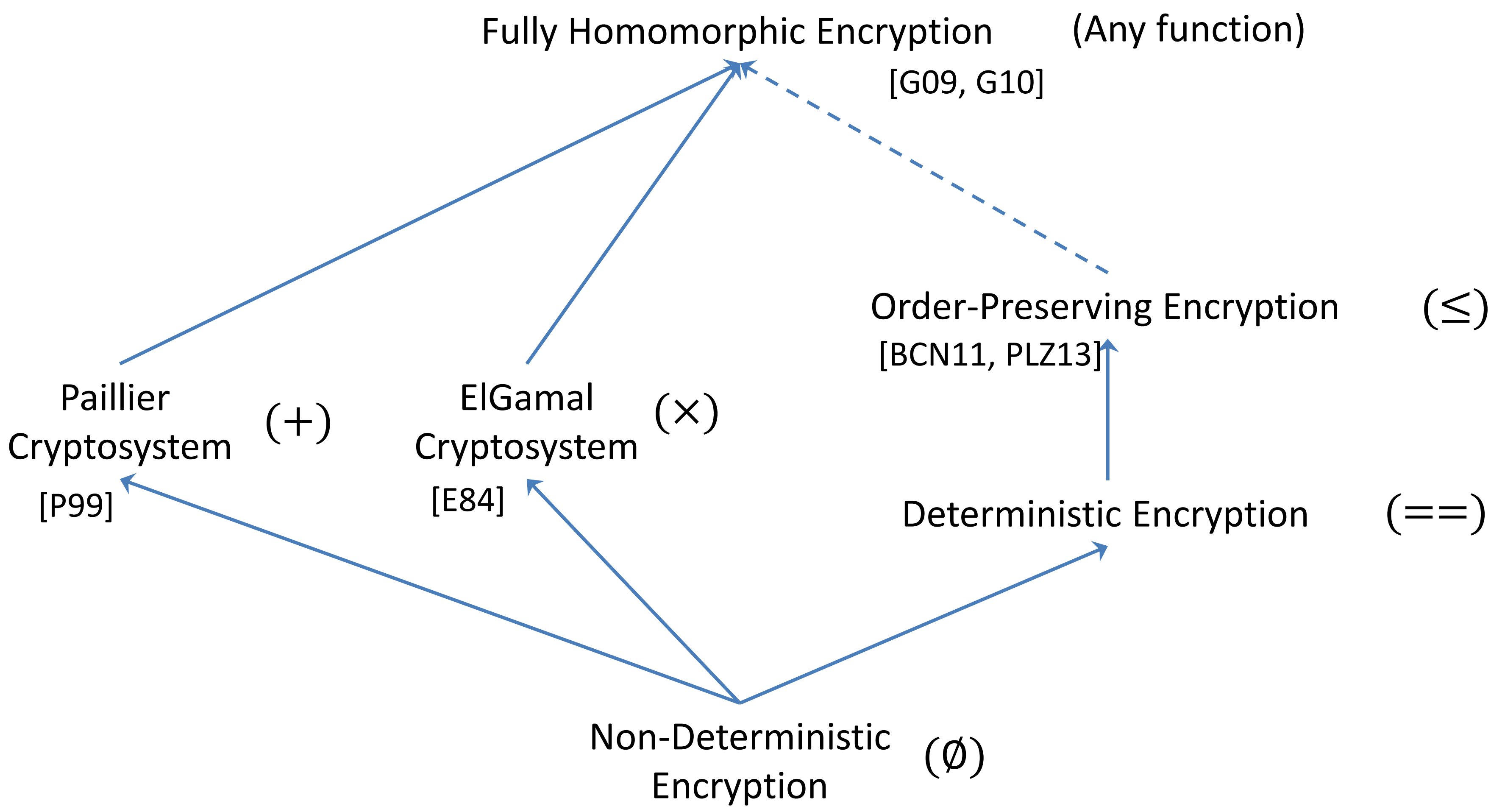
Order-Preserving Encryption

```sql
select *
from assignment
where score_OPE >= 0x04e062ff507458f9be50497656ed654c
```

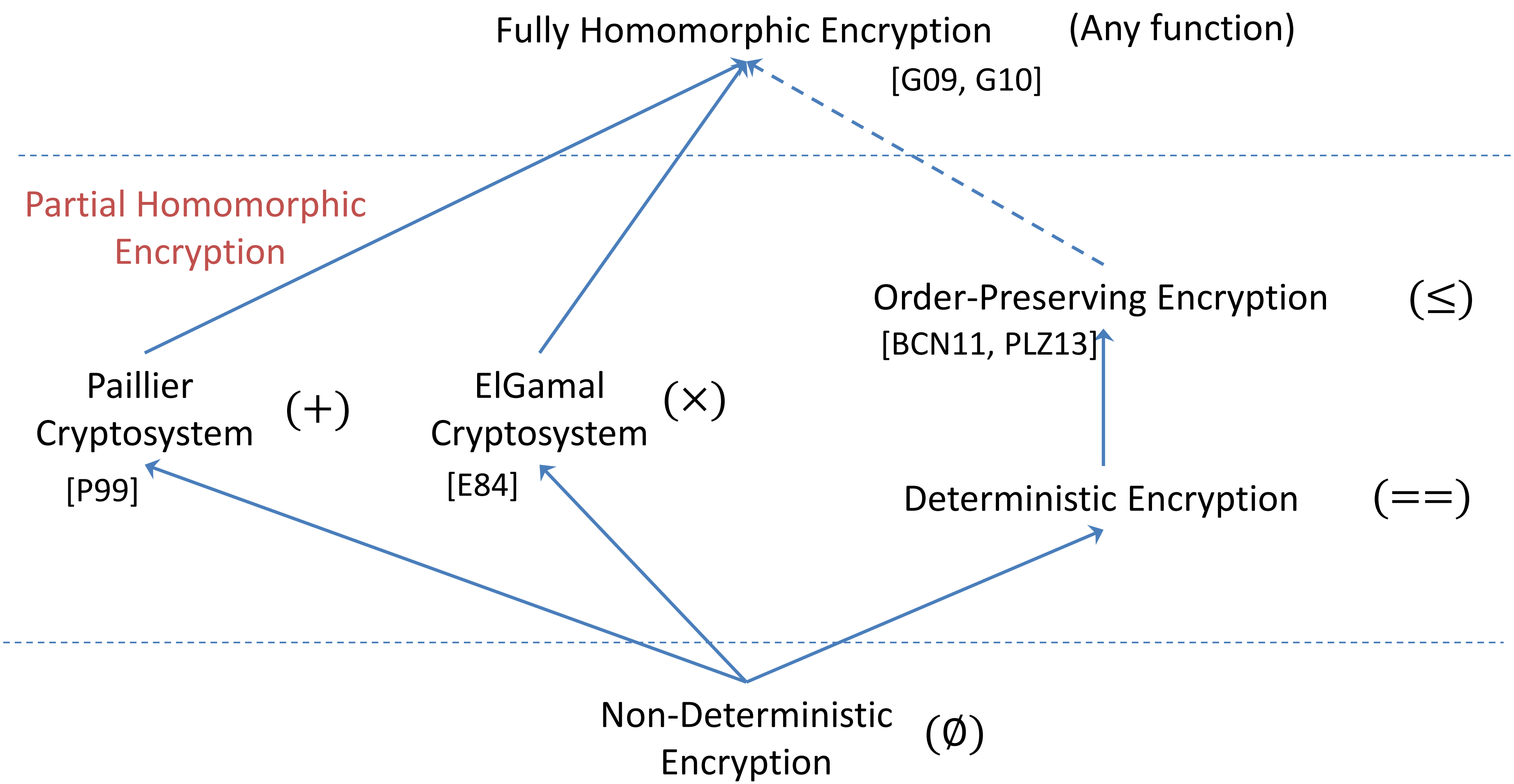
$$\sigma_{Score\_ope\ \geq 04e0\ldots}$$

StudentId	AssignId	Score_OPE
1	1	0x0065fda789ef4e272bcf102787a93903
1	2	0x009b5708e13665a7de14d3d824ca9f15
3	4	0x08db34fb1f807678d3f833c2194a759e
...	...	...

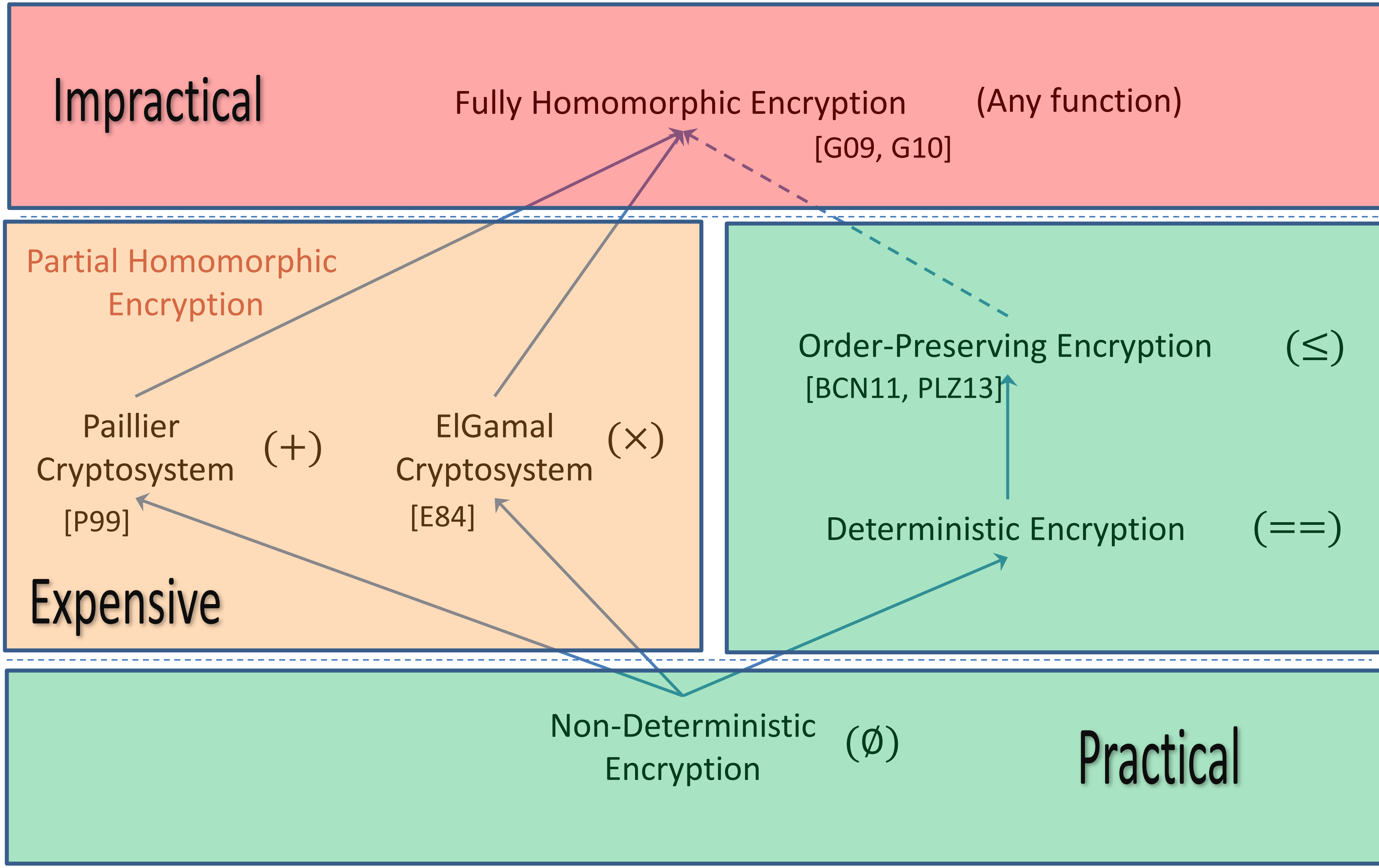
Homomorphic Encryption Schemes



Homomorphic Encryption Schemes



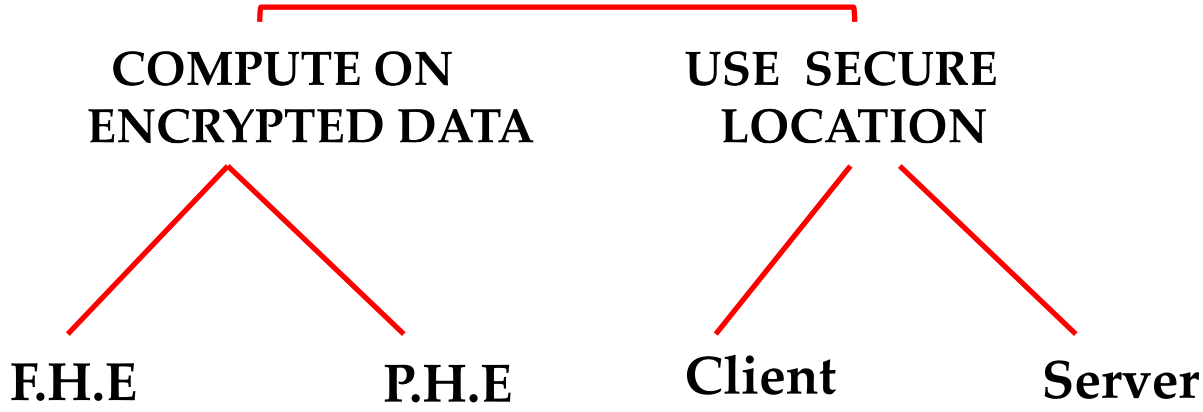
Homomorphic Encryption Schemes



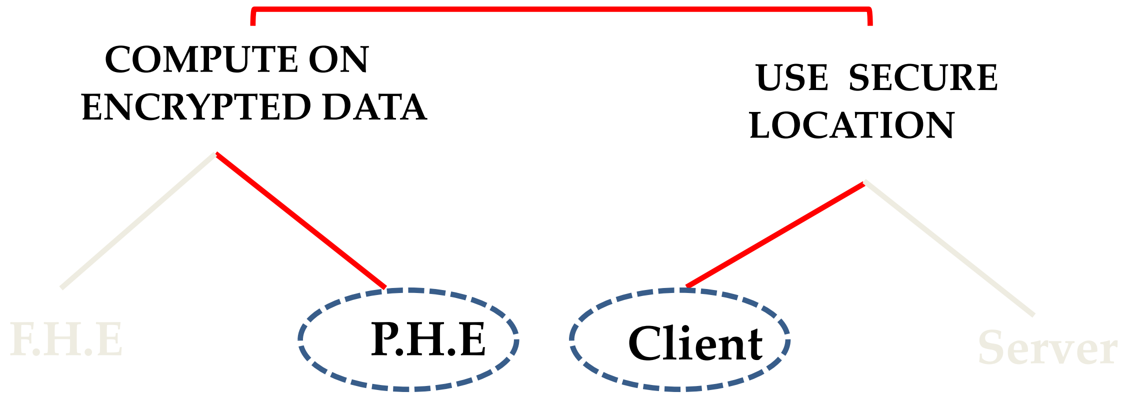
Homomorphic Encryption Schemes: Performance

Scheme	Space for 1 integer (bits)	Time for 1 operation
Fully Homomorphic Encryption	2^{14}	Cosmic time scales
Paillier ElGamal	2048	$\approx$ ms
Deterministic Order-preserving	128	$\approx$ µs

Design Choices



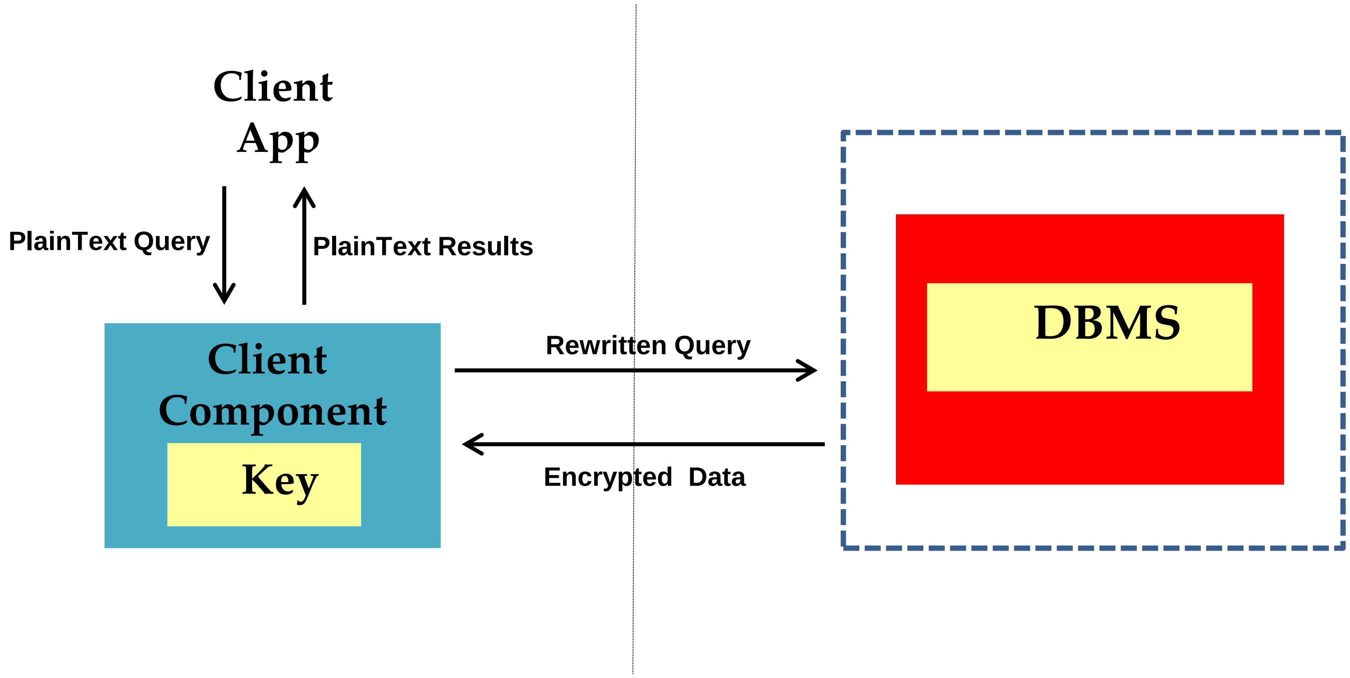
Trusted Client based Systems



Outline

- Part I: Design Choices
  - Encryption basics
  - Trusted client-based systems
  - Secure in-cloud computing
- Part II: Security Guarantees
  - Security for data at rest
  - Dynamic security
- Upcoming Papers and Announcements

Trusted Client Architecture

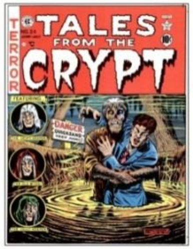


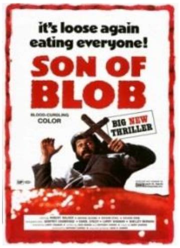
- Data not decrypted in DBMS
  - Only ciphertext seen in the DBMS
- No changes to DBMS/Client App

Trusted Client-based Systems

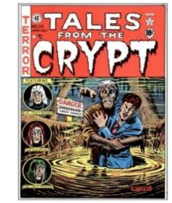
- Minimal Client Computation
  - Use P.H.E (CryptDB)

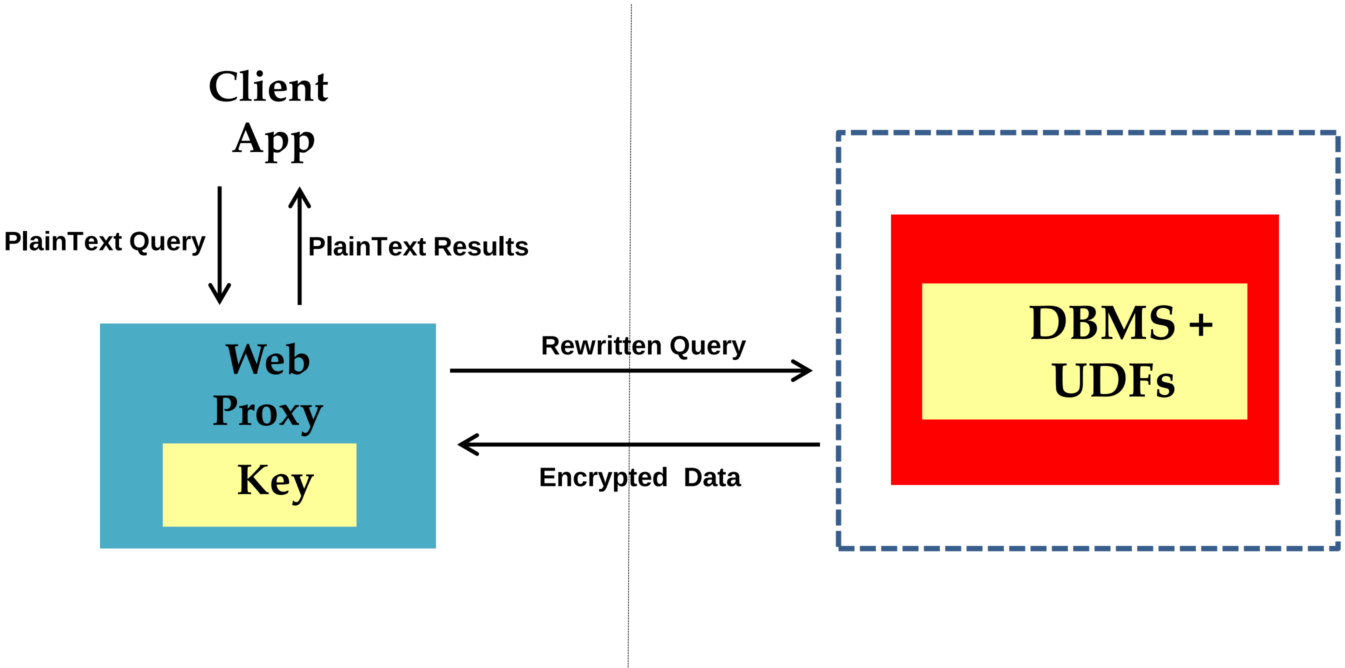
- Residual Query Processing in Client
  - Blob Store
  - Use in conjunction with P.H.E (Monomi)





CryptDB Architecture





- Web proxy rewrites queries, decrypts result
- Leverage P.H.E techniques

[PRZ+11]

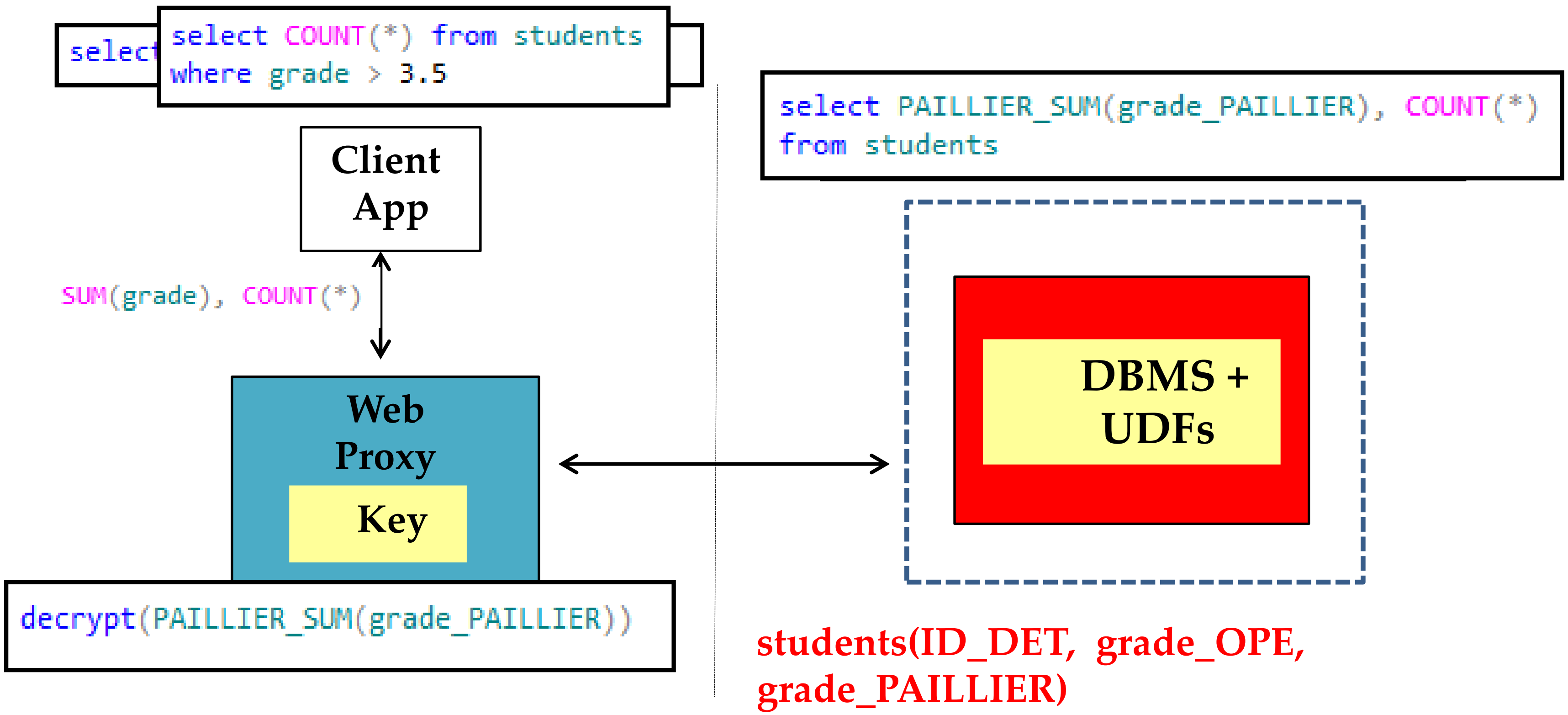
Database Design

- Plaintext Schema
  - students(ID, grade)
    - Point Lookups on ID column
    - SELECT and AGGREGATION queries on grade

- Ciphertext Schema
  - students(ID_DET, grade_OPE, grade_PAILLIER)
    - Need to store columns encrypted in multiple ways
    - Static/Dynamic design based on workload

[PRZ+11]

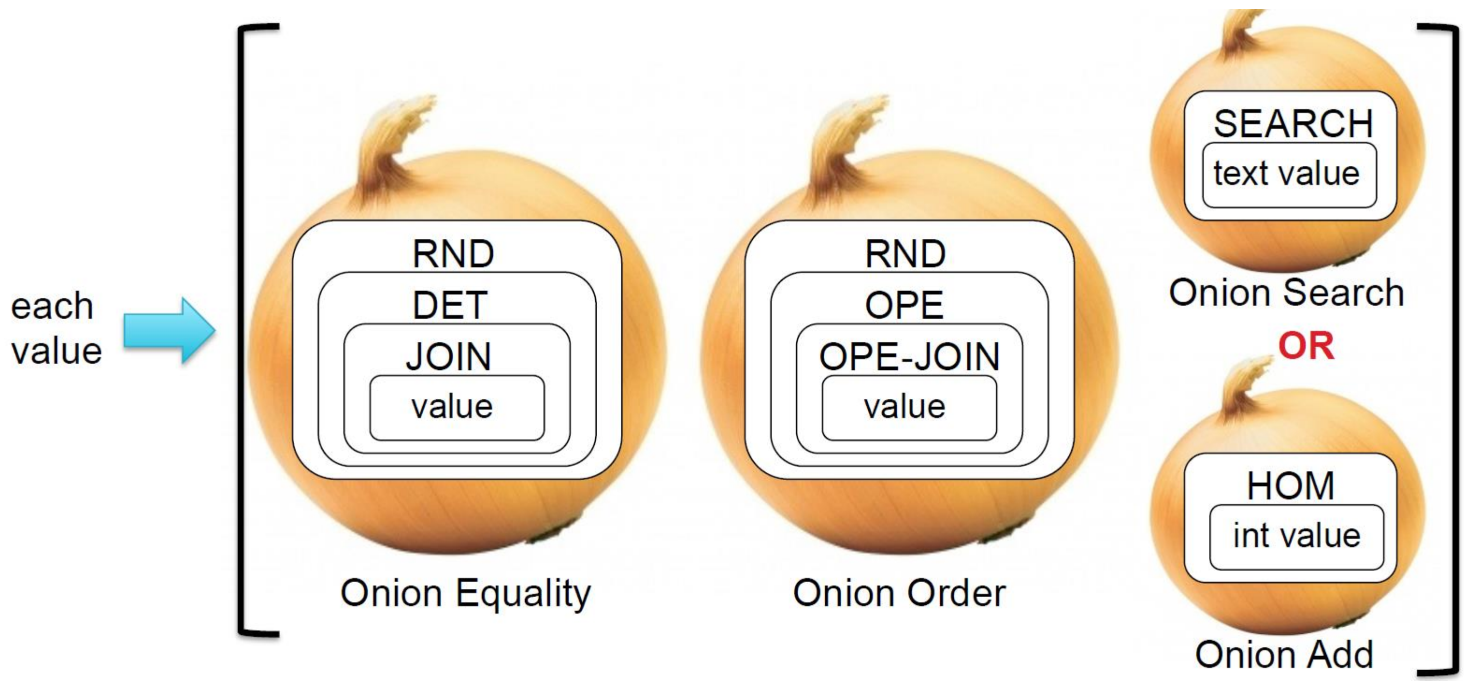
Query Processing



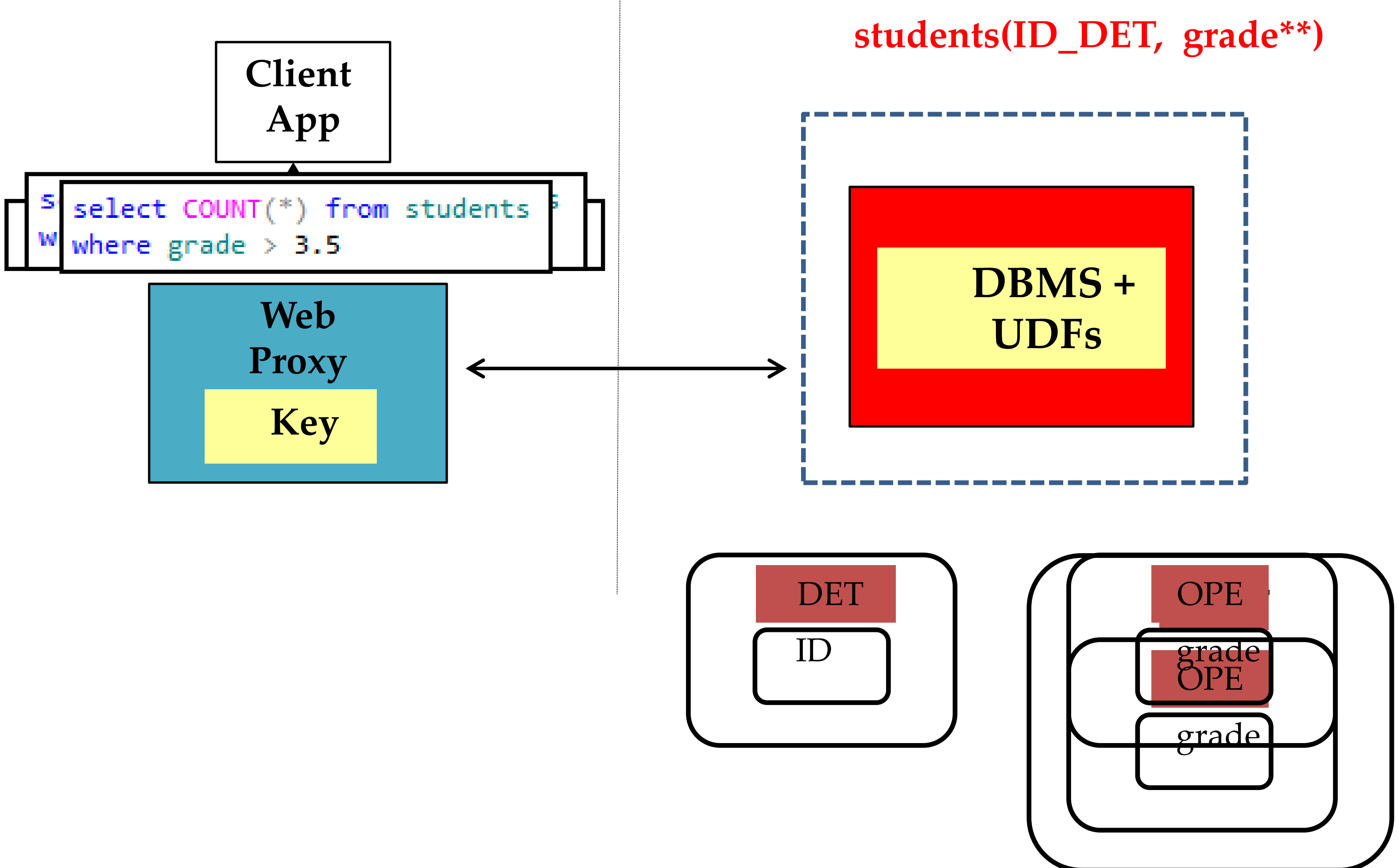
[PRZ+11]

Onions of Encryptions

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



Dynamic Database Design



[PRZ+11]

Limitations

- P.H.E is not "free" – space overheads
  - For Paillier, to store one integer (32 bits), the ciphertext need to use 2048 bits!
  - Compact representation for paillier that is updatable is an open problem.

- P.H.E is inherently limited – cannot address all of SQL

```
select STDEV(grade) from students
```

[GZ07]

Pre-computation for complex queries

- Find student submissions that have been handled in a day late

```
select id, count(*) from students
where submissiondate = deadline + 1
group by id
```

- Students(ID_DET, submissiondate_DET, deadline_DET, deadline_PAILLIER)
  deadline_DET, deadlineplusone_DET)
  - Cannot "Mix and Match" different encryptions

```
select id, count(*) from students
where submissiondate = deadline_plusone
group by id
```

[TFM13]

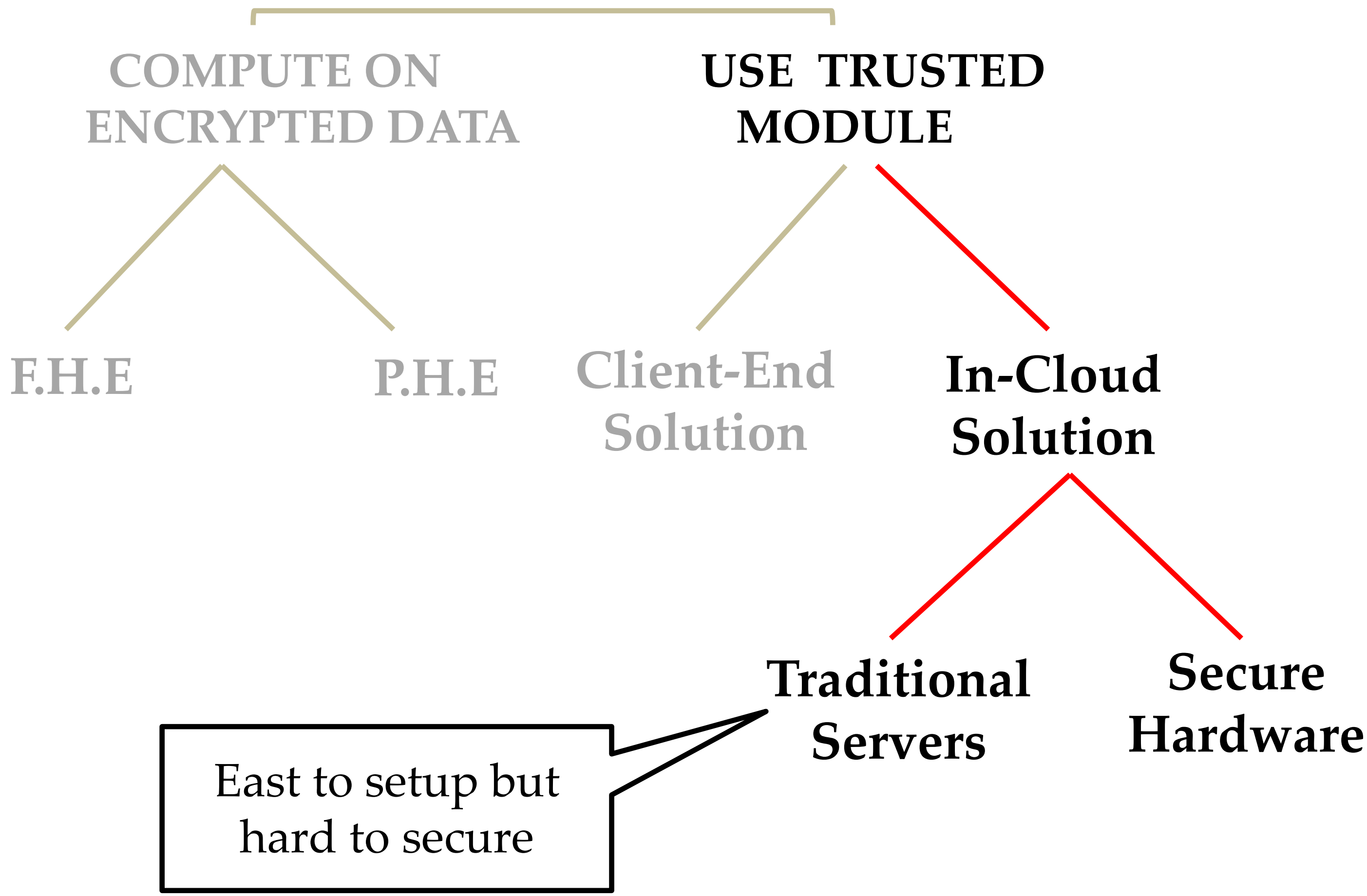
Trusted Client: Summary

- No server changes required to DBMS
- Works well for workloads where amount of data shipped is small
  - Physical design is important for distributed queries
  - Pre-computation is not free
- Generality of approach is unproven
  - Integrity constraints, Triggers etc.
  - Automated tools to migrate database applications

Outline

- Part I: Design Choices
  - Encryption basics
  - Trusted client-based systems
  - Secure in-cloud computing
- Part II: Security Guarantees
  - Security for data at rest
  - Dynamic security
- Upcoming Papers and Announcements

Secure In-Cloud Processing



Secure In-Cloud Processing

COMPUTE ON ENCRYPTED DATA

USE TRUSTED MODULE

F.H.E

P.H.E

Client-End Solution

In-Cloud Solution

Traditional Servers

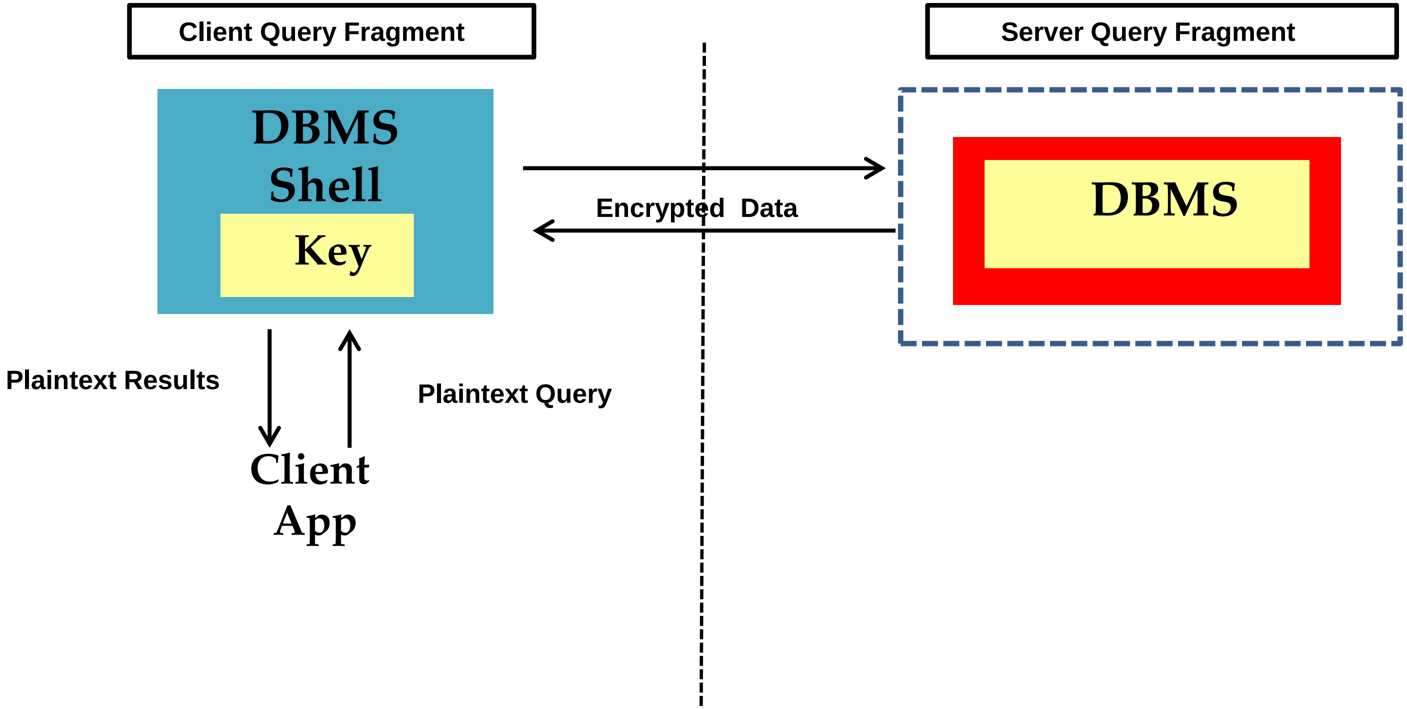
Secure Hardware

Limited Recourses!

IBM 4764 PCI-X Cryptographic Coprocessor

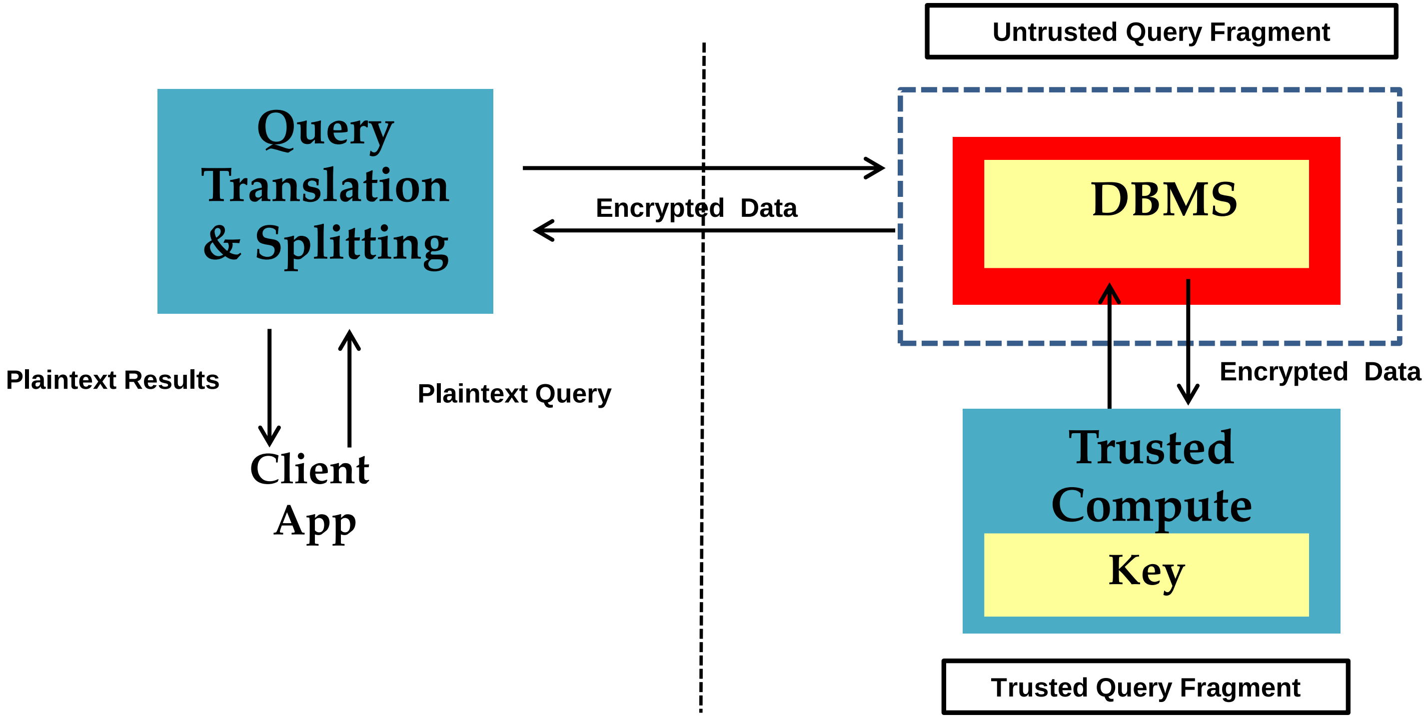
Secure FPGA

Trusted Client Architecture



- Distributed query processing between untrusted DBMS and client-end DBMS shell

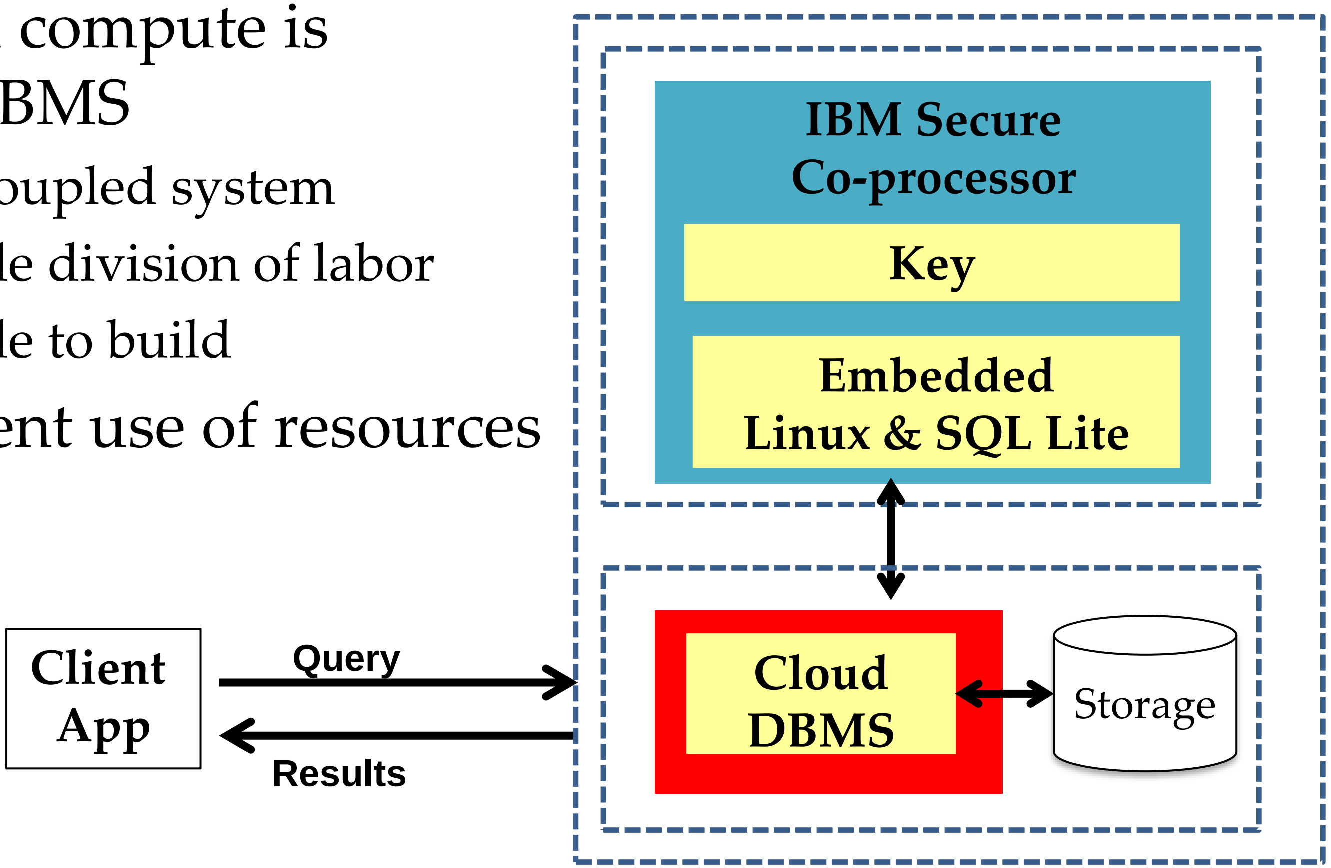
Secure In-Cloud Compute Architecture



- Distributed query processing between untrusted DBMS and trusted cloud compute
- Solutions differ in <u>granularity of integration</u>

Secure Processors

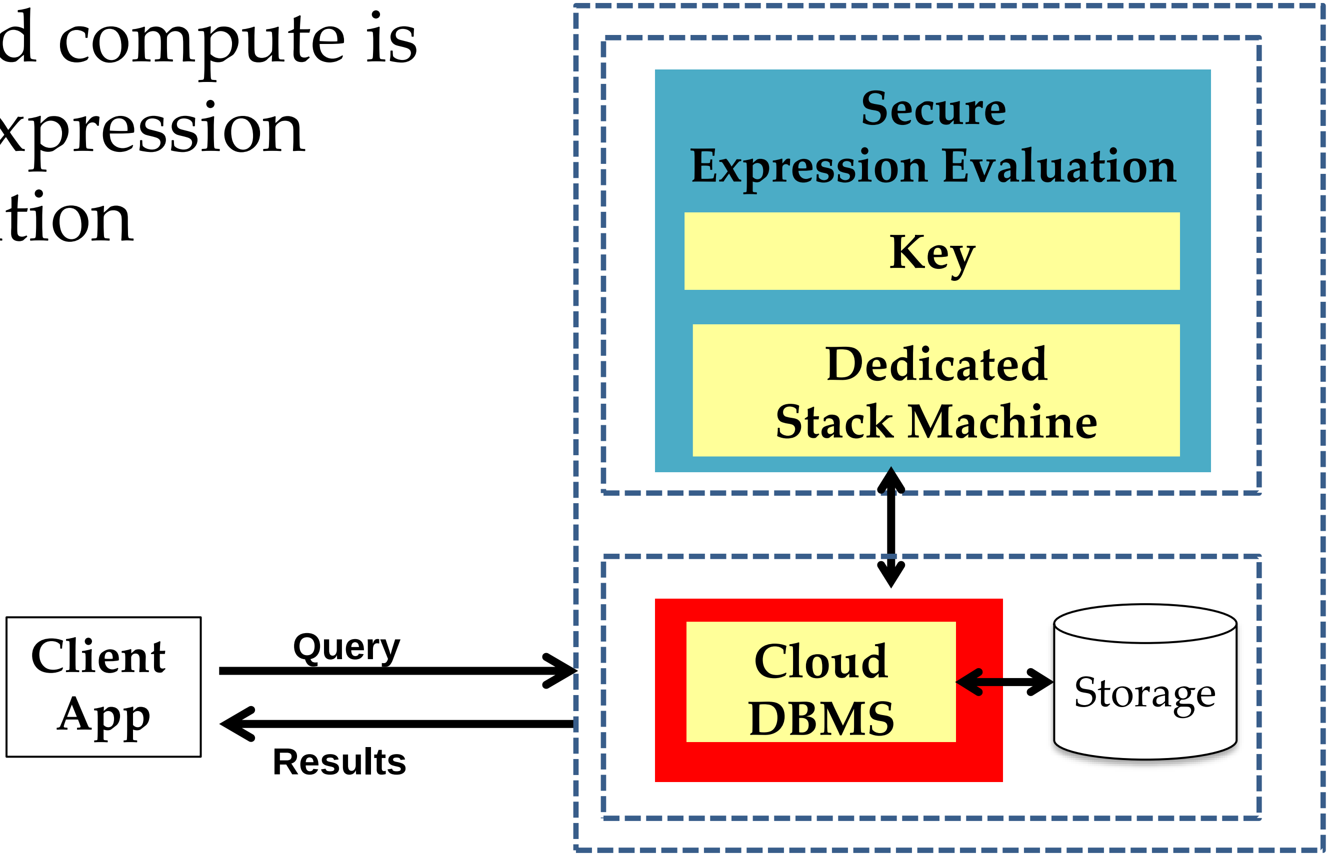
- TrustedDB
  - Trusted compute is a full DBMS
  - loosely coupled system
    - Simple division of labor
    - Simple to build
  - Inefficient use of resources



[BS11]

Dedicated Expression Evaluation

- Cipherbase
  - Trusted compute is only expression evaluation



[ABE+12, ABE+13]

Dedicated Expression Evaluation

- Advantages
  - Efficiency of trusted compute resources
  - Dedicated circuits ➔ virus-proof
  - Small footprint ➔ formal verification

- Drawbacks
  - Fundamentally changes expression evaluation ➔ non-trivial changes to host DBMS

[ABE+12, ABE+13]

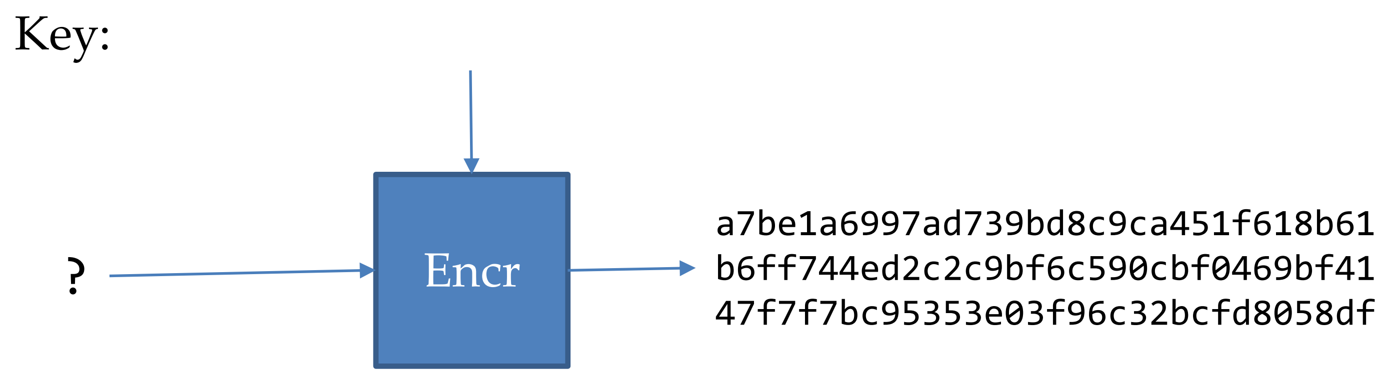
Secure In-cloud: Summary

- Secure in-cloud trusted compute resources
- Open issues
  - Query optimization
    - e.g. Statistics on encrypted data, security-aware type matching
  - Execution engine
    - e.g. Data/computation reuse, masking latency to trusted computation
  - Physical Design
    - e.g. Leveraging stronger encryption

Outline

- Part I: Design Choices
  - Encryption Basics
  - Trusted Client based Systems
  - Secure In-Cloud Computing
- Part II: Encryption & Security
  - Security for data at rest
  - Dynamic security
- Upcoming Papers and Announcements

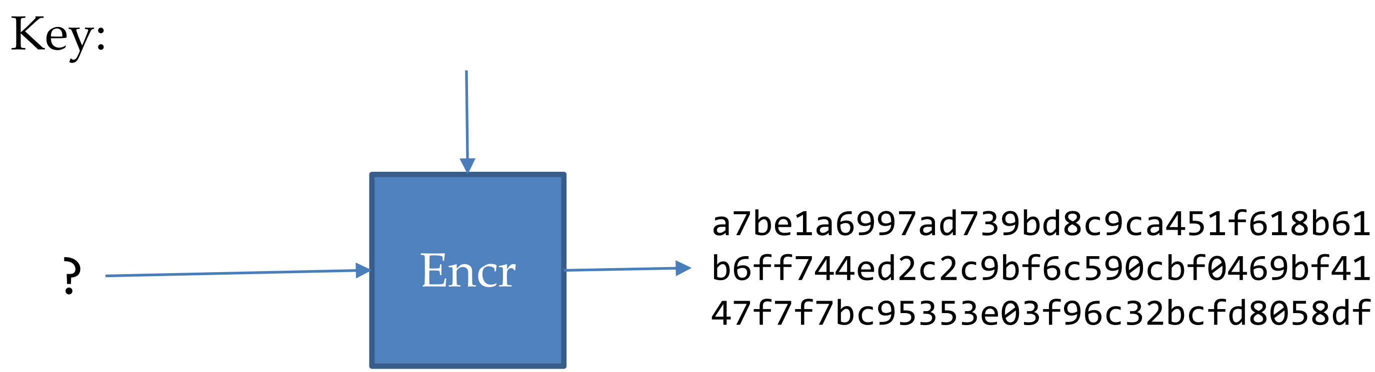
Encryption and Security



- **Semantic security**:
  - No information leakage except input length
- Winner Turing Award

[KL07]

Encryption and Security



- Encryption schemes such as AES in CBC mode (non-deterministic) are believed to be semantically secure

Security of Database Encryption

Disease
Flu
Diabetes
Flu
Cold

Disease_NDET
!x8J
)zFr#x
^@tG
BxU3

- Apply AES-CBC to every cell
- Leaks cell lengths

Deterministic Encryption

Disease
Flu
Diabetes
Flu
Cold

Disease_DET
!x8J
)zFr#x
!x8J
BxU3

- Leaks cell lengths
- Also, no. of distinct values + frequency distribution [BFO+08]

Order-Preserving Encryption

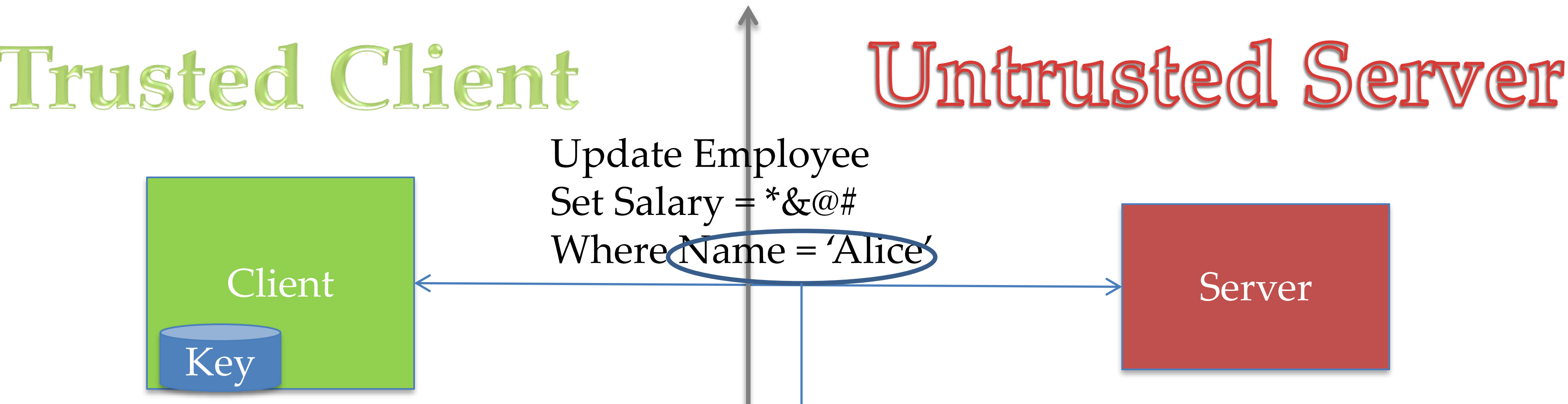
Age
12
51
24
36

Age_OPE
0x000a
0x0f12
0x00a1
0x00b2

- Leaks cell lengths
- Also, order of cell values and more [AKS+04, BCN11, NKW15, Grubbs et al. S&P17]

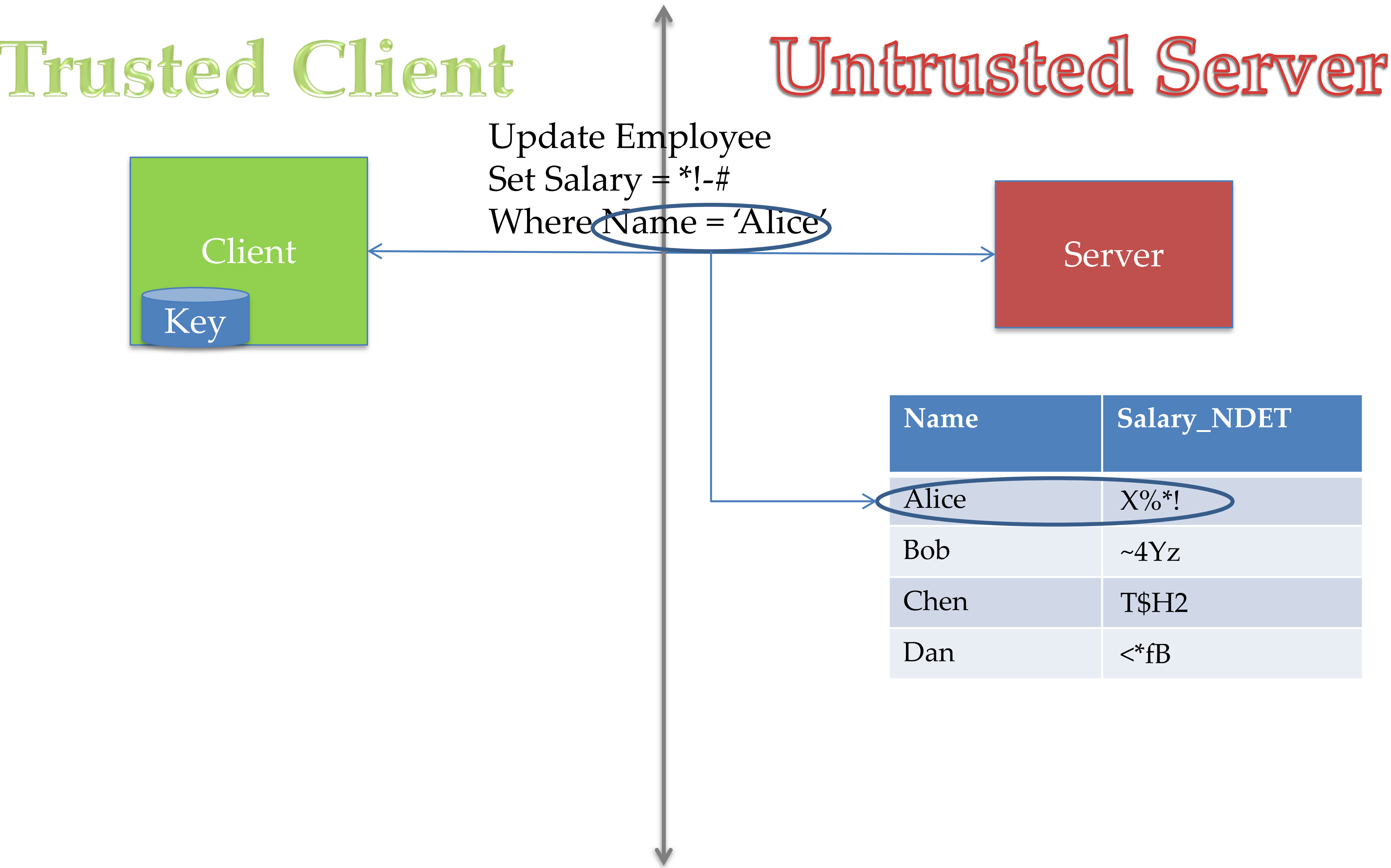
Design of order-preserving encryption

Impact of Querying & Updating



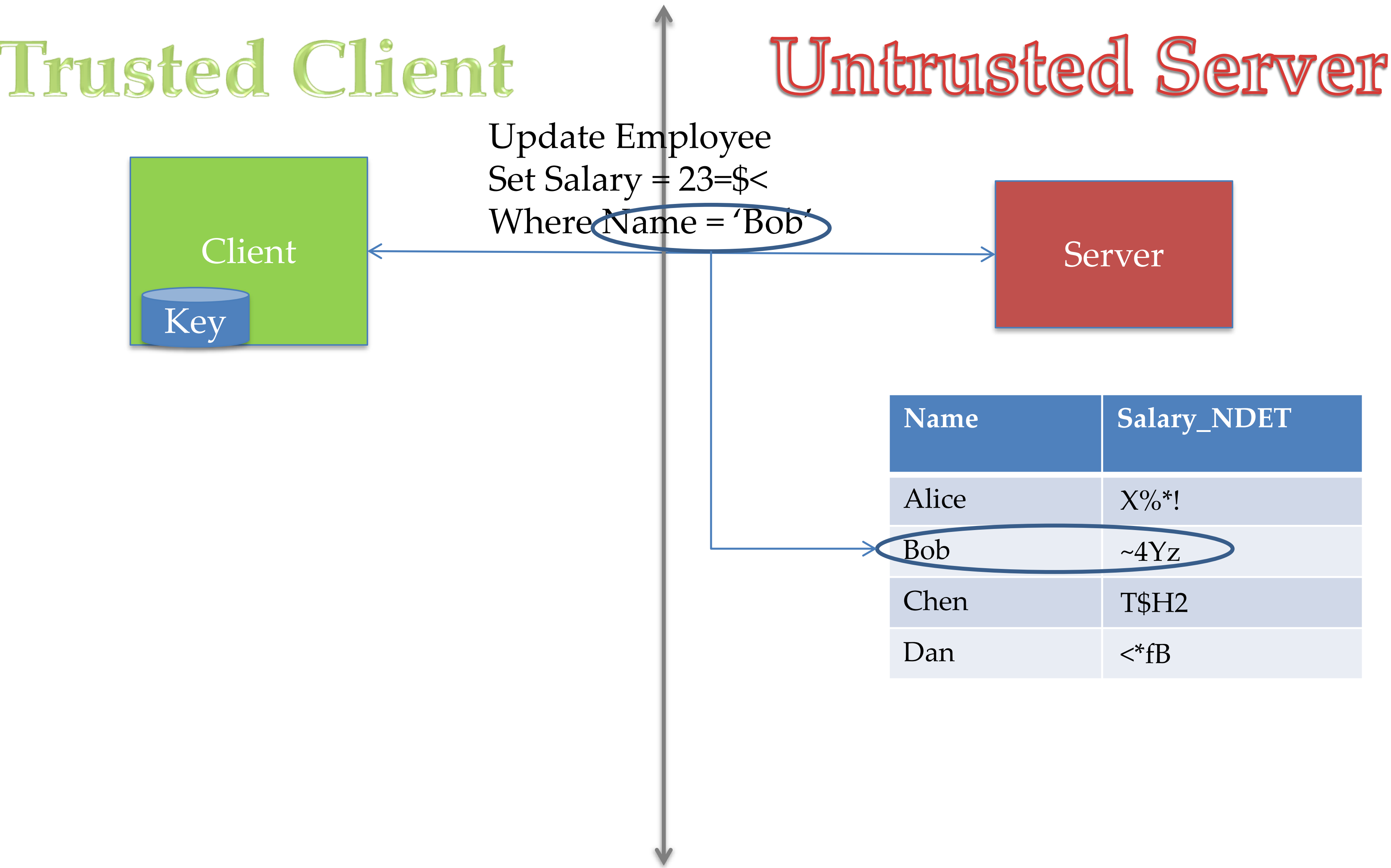
Name	Salary_NDET
Alice	X%*!
Bob	~4Yz
Chen	T$H2
Dan	<*fB

Impact of Querying & Updating



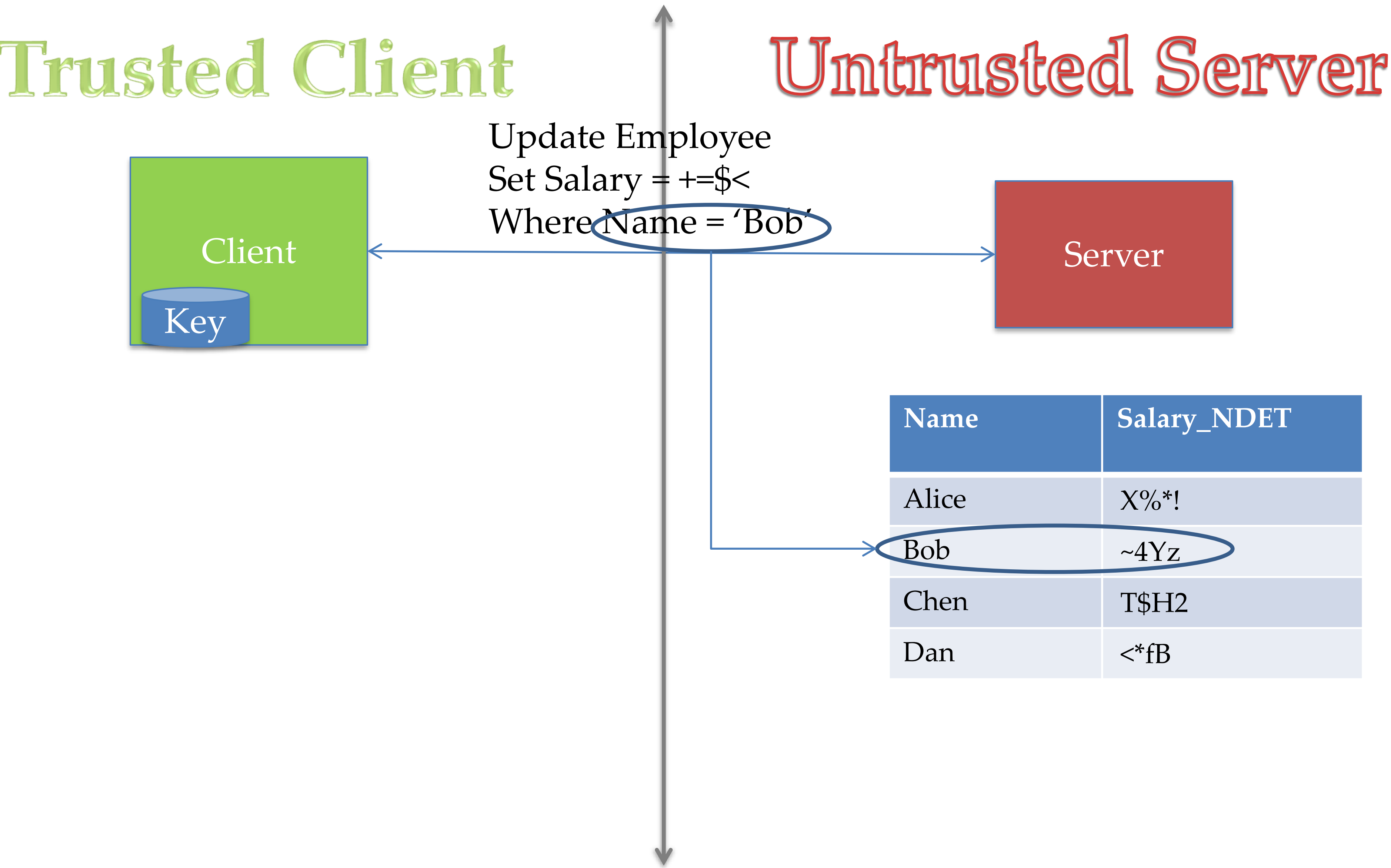
Name	Salary_NDET
Alice	X%*!
Bob	~4Yz
Chen	T$H2
Dan	<*fB

Impact of Querying & Updating



Name	Salary_NDET
Alice	X%*!
Bob	~4Yz
Chen	T$H2
Dan	<*fB

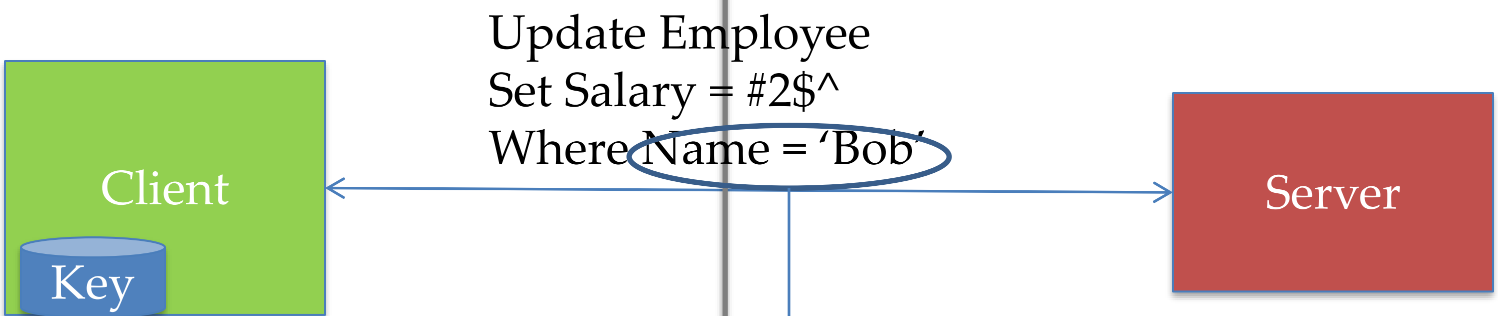
Impact of Querying & Updating



Name	Salary_NDET
Alice	X%*!
Bob	~4Yz
Chen	T$H2
Dan	<*fB

Impact of Querying & Updating

Trusted Client | **Untrusted Server**

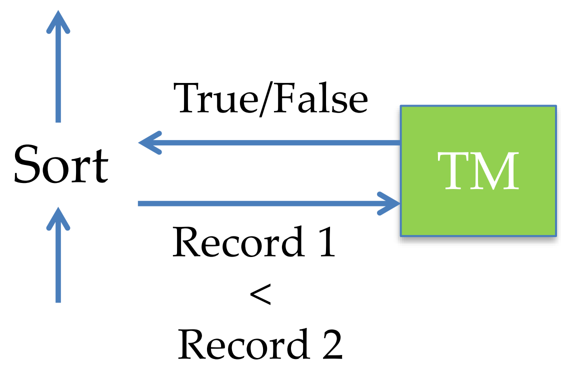


Name	Salary_NDET
Alice	X%*!
Bob	~4Yz
Chen	T$H2
Dan	<*fB

- Background knowledge
  - Full-time employees earn more
  - Salaries of hourly-wage employees updated more
- Learn partial ordering of employee salary
  - Alice's salary > Bob's

Query access patterns reveal information! [OS07]

Impact of Querying & Updating



- Sort leaks ordering
- Encryption across the stack (disk + in-memory) does NOT imply no information leakage

The overall query workflow reveals information

***Dynamic* security** (different from security of data at rest)

Design Space

Cipherbase, TrustedDB, CryptDB, Monomi, BlobStore

Stop with encryption

Can we bridge this gap?

Full Leakage

Operations on column	Leakage
Equality (including joins)	Frequency distribution
Indexing/Sorting/range predicates	Order

No Leakage
Impractical

Design Space

Cipherbase,
TrustedDB,
CryptDB,
Monomi,
BlobStore

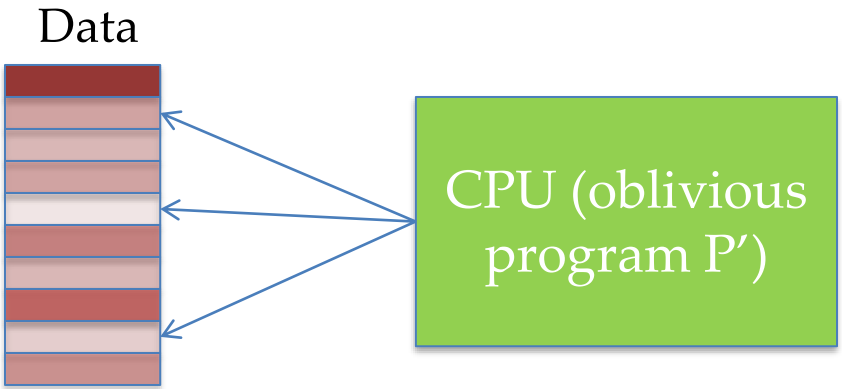
Stop with encryption

Full Leakage

Output Size,
Running Time

No Leakag
Impractical

Oblivious Simulation

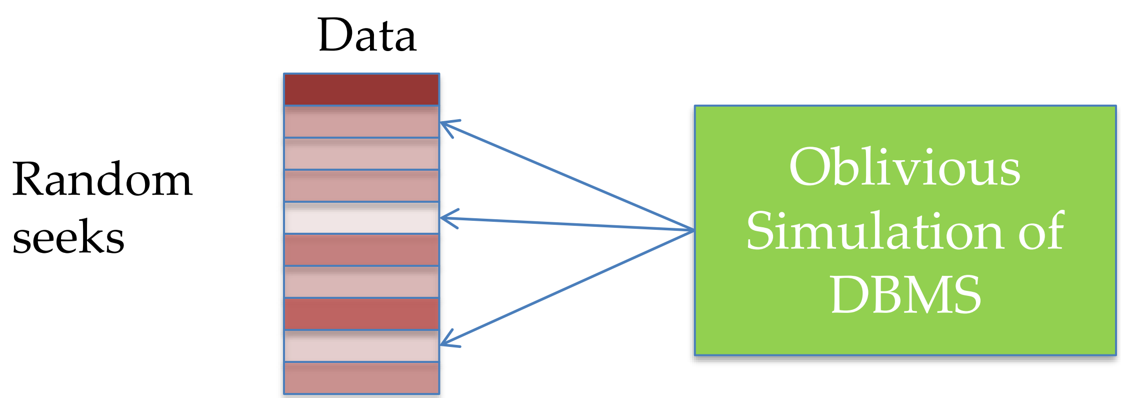


- Simulation: P' equivalent to P
- Theoretically Efficient: Running time of P' within polylog factor of running time of P
- Oblivious: Access patterns of P' look random
- Information leakage: input size, output size, running time

[GO96, W12, SS13]

Application to DBMS

- Destroys spatial and temporal locality of reference



- Range scan of 100M records on hard disk → 100M seeks
  - 10^5 seconds (~1 day)

Design Space

Cipherbase,
TrustedDB,
CryptDB,
Monomi,
BlobStore

Stop with encryption

Full Leakage

Output Size,
Running Time
Impractical

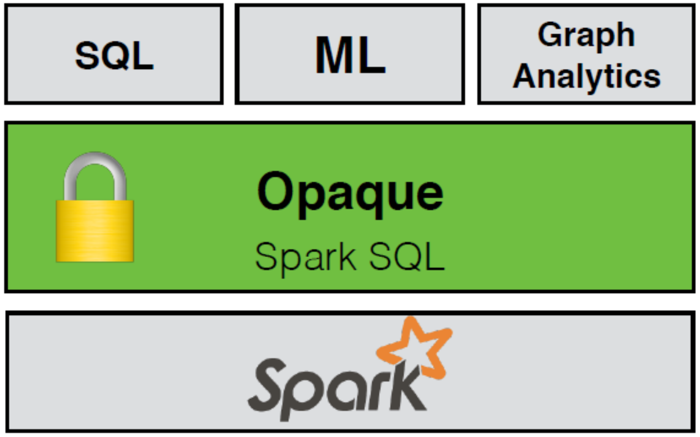
No Leakag
Impractica

Is there a stronger and practically achievable security model?

e.g. Opaque Zheng et al. NSDI 2017

Opaque: Secure Distributed Analytics

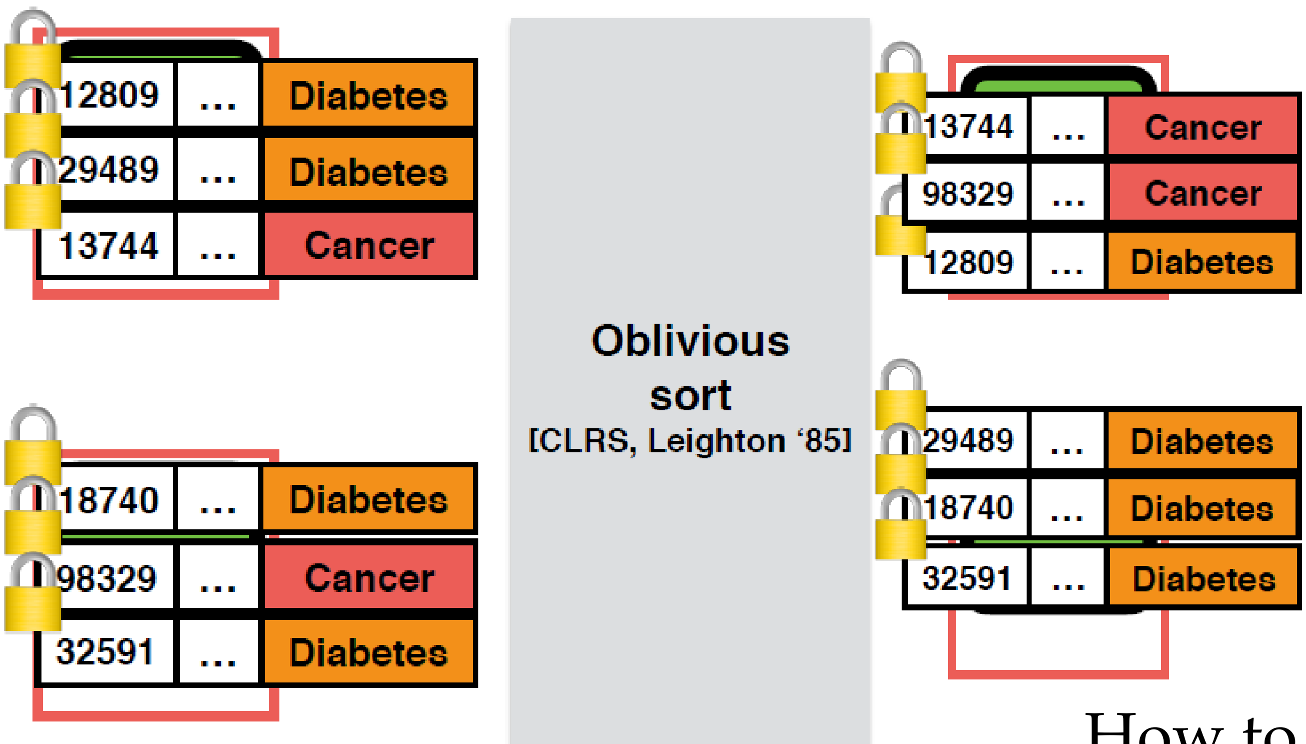
- Obliviousness: hiding access control
- Two-part solution:
  - Distributed oblivious SQL operators:
    - Oblivious filter
    - Oblivious sort
    - **Oblivious aggregation**
    - Oblivious join
  - Query optimization
    - Rule-based optimization
    - Cost model
    - Cost-based optimization



[Zheng et al. NSDI 2017]

Oblivious Aggregation Example

- SELECT count(*) FROM medical GROUP BY disease

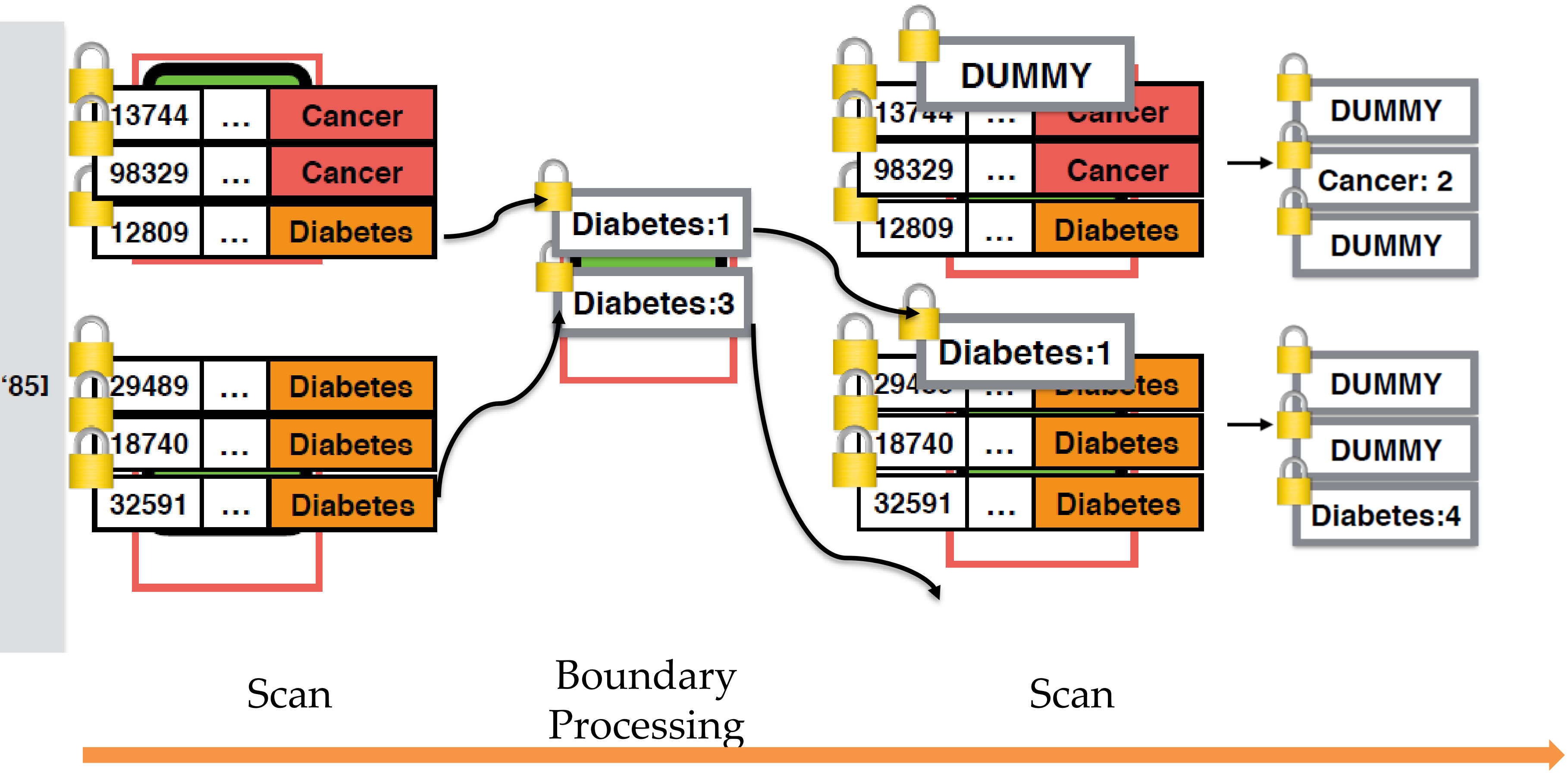


The "diabetes" group is split

How to aggregate obliviously and in parallel?

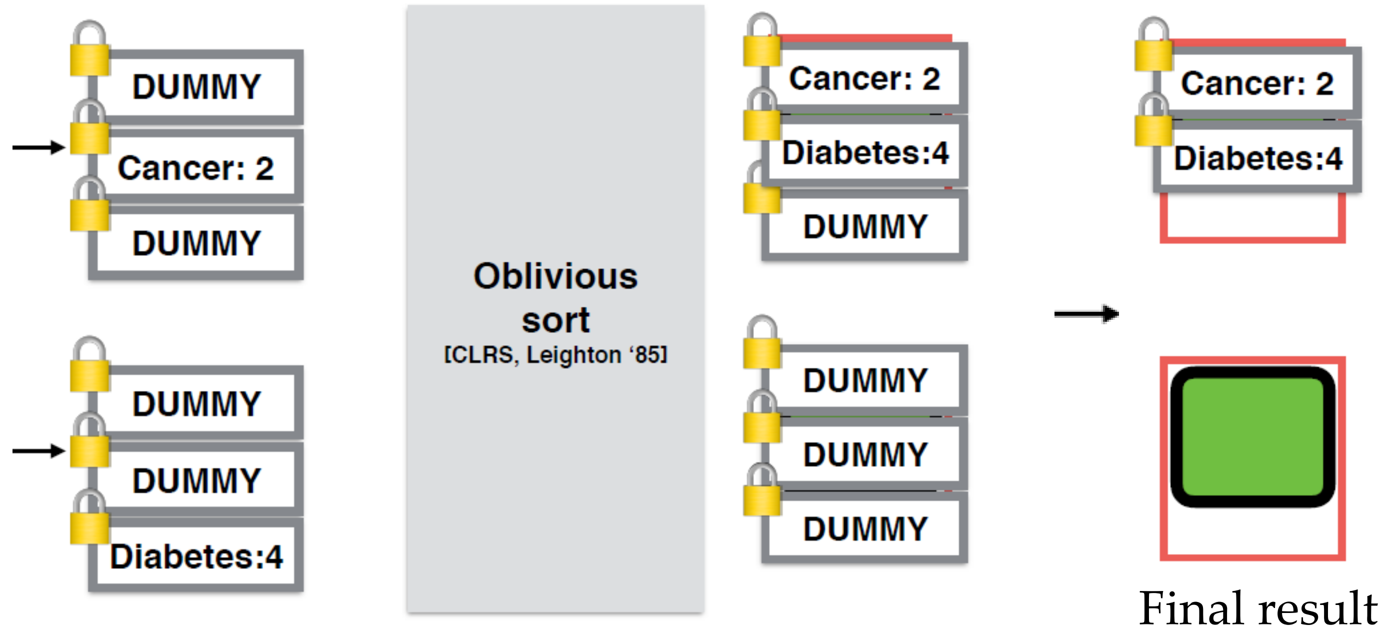
Oblivious Aggregation Example

- SELECT count(*) FROM medical GROUP BY disease



Oblivious Aggregation Example

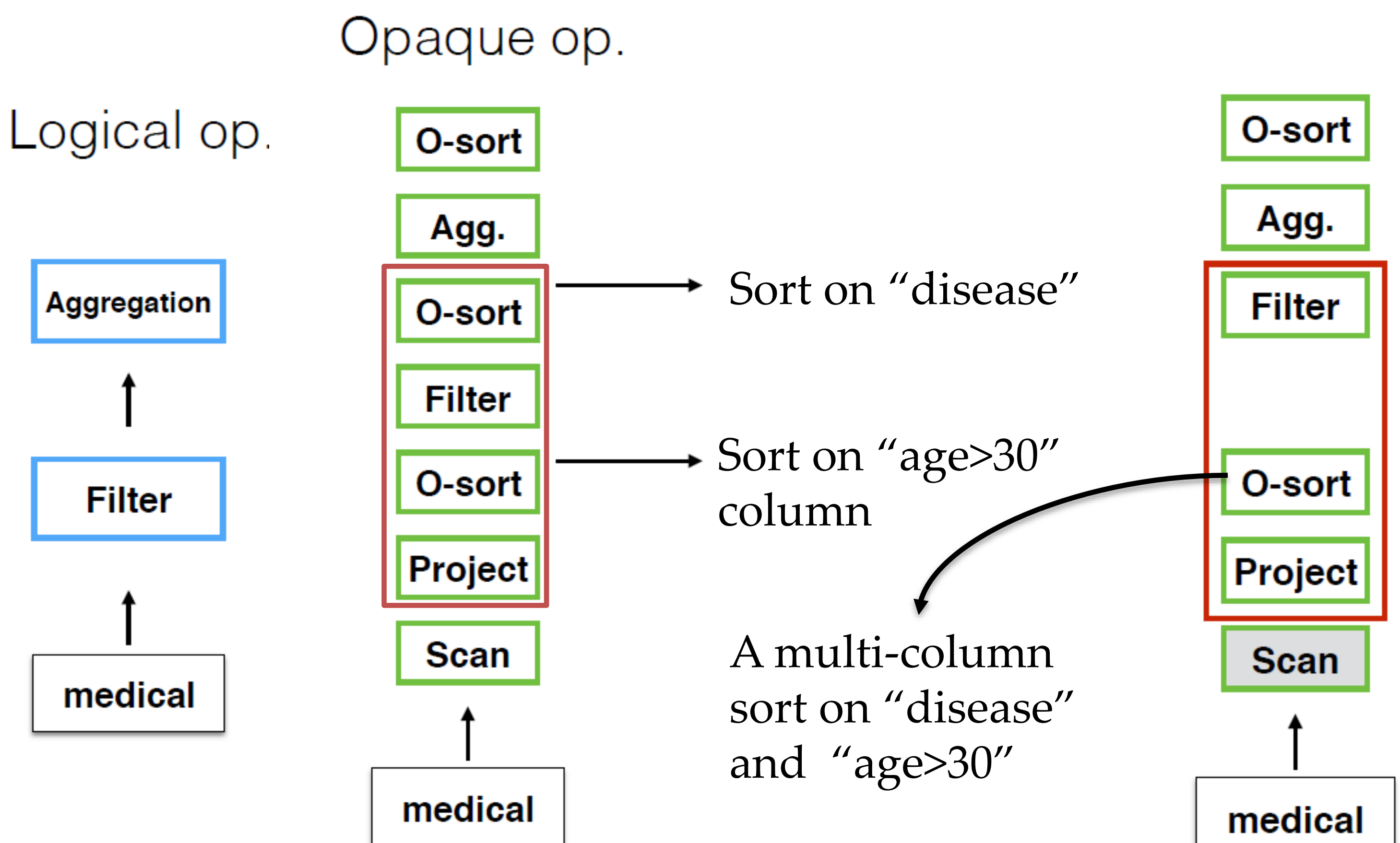
- SELECT count(*) FROM medical GROUP BY disease



Aggregation has at least 2 oblivious sorts,
Can Opaque do better?

Rule-based Optimization Example

- SELECT count(*) FROM medical WHERE age> 30 GROUP BY disease;



Summary

- Part I: Design Choices
  - Encryption basics
  - Trusted client-based systems
  - Secure in-cloud computing
- Part II: Security Guarantees
  - Security for data at rest
  - Dynamic security

Other Challenges

- Application Security
  - DBMS is only a part of the overall system stack
- Usability
  - Clients need tools and interpretable security models to navigate security-performance tradeoff
- Connections to other areas of security
  - Data privacy, access control, auditing

Discussion Time

Paper Readings

- Week 11
  - 3a. Seabed
  - 3b. OPE Leakage
  - 3c. Partitioned Data Security
- Week 12
  - 3d. StealthDB
  - 3e. EncDBDB

Announcement

- Assignment 3
  – Release after Wed session
  – Latex file will be available
  – Submit pdf on Learn, by Mar 29, 11pm

- Project midterm report
  – Submit pdf on Learn, by March 22, 11pm

Slides Credits

- Slides for “Querying Encrypted Data”, Arasu et al. ICDE 2016
- Slides for “Opaque”, Zheng et al. NSDI 2017