Privacy and Integrity in Outsourced Databases

Gene Tsudik
Computer Science Department
School of Information & Computer Science
University of California, Irvine
gts@ics.uci.edu

1

Software as a Service

- Get
  - what you need
  - when you need it
- Pay for
  - what you use
- Don't worry about:
  - Deployment, installation, maintenance, upgrades
  - Hire/train/retain people

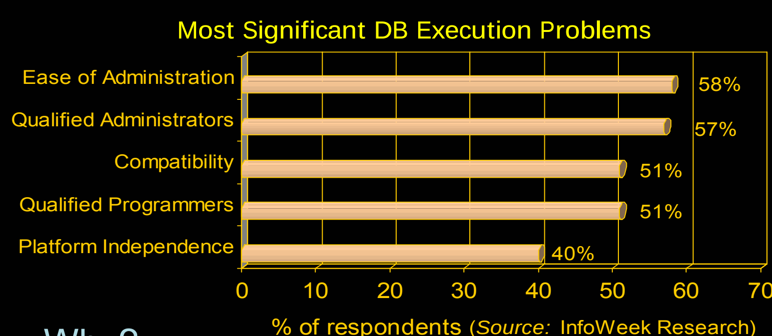
2

Software As a Service: Why?

- Advantages
  - reduced cost to client
    - pay for what you use and not for: hardware, software infrastructure or personnel to deploy, maintain, upgrade…
  - reduced overall cost
    - cost amortization across users
  - better service
    - leveraging experts across organizations
- **Driving Forces**
  - **Faster, cheaper, more accessible networks**
  - **Virtualization in server and storage technologies**
  - **Established e-business infrastructures**
- **Already in Market**
  - **Horizontal storage services, disaster recovery services, e-mail services, rent-a-spreadsheet services etc.**
  - **Sun ONE, Oracle Online Services, Microsoft .NET My Services, etc**

Better Service → Cheaper

Emerging Trend: Database As a Service



- Why?
  - Most organizations need DBMSs
  - DBMSs extremely complex to deploy, setup, maintain
  - require skilled DBAs (at very high cost!)

The DAS Project**

Goal: Security for the Database-as-a-Service

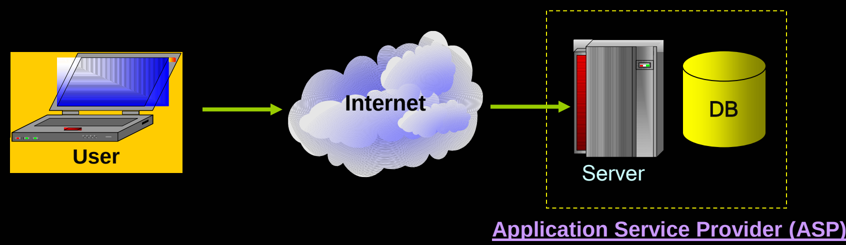
People: Sharad Mehrotra, Gene Tsudik
Ravi Jammala, Maithili Narasimha,
Bijit Hore, Einar Mykletun, Yonghua Wu

** Supported in part by NSF ITR grant "Security & Privacy in Database as a Service"

Rough Outline

- What we want to do
- Design space
- Challenges
- Architecture
- Bucketization
- Integrity & Authenticity
- Aggregated signatures
- Hash trees
- Related work

What do we want to do?



- *Database as a Service* (DAS) Model
  - DB management transferred to service provider for
    - backup, administration, restoration, space management, upgrades etc.
  - use the database "as a service" provided by an ASP
    - use SW, HW, human resources of ASP, instead of your own

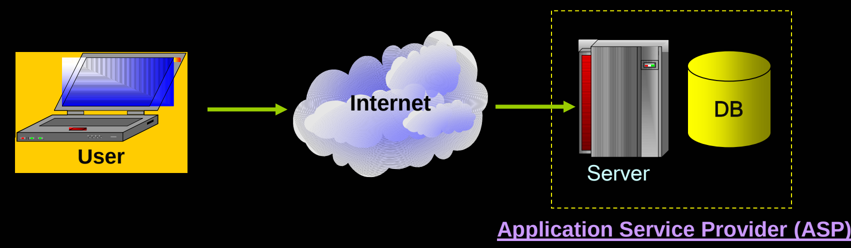
7

DAS variables

- Database types
- Interaction dynamics
- Trust in Server

8

What do we want to do?

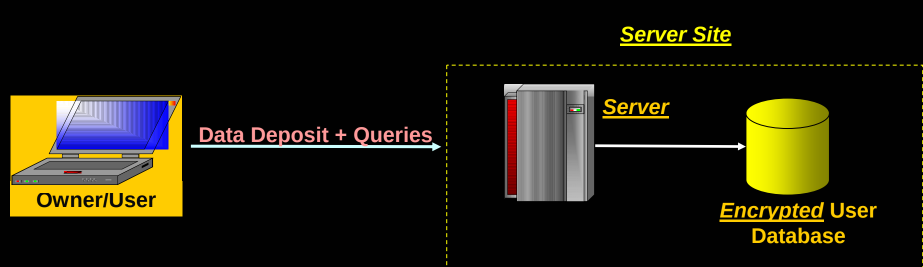


Database Types in the DAS Model:

– Warehousing (write once, read many)
– Archival (append only)
– Dynamic

9

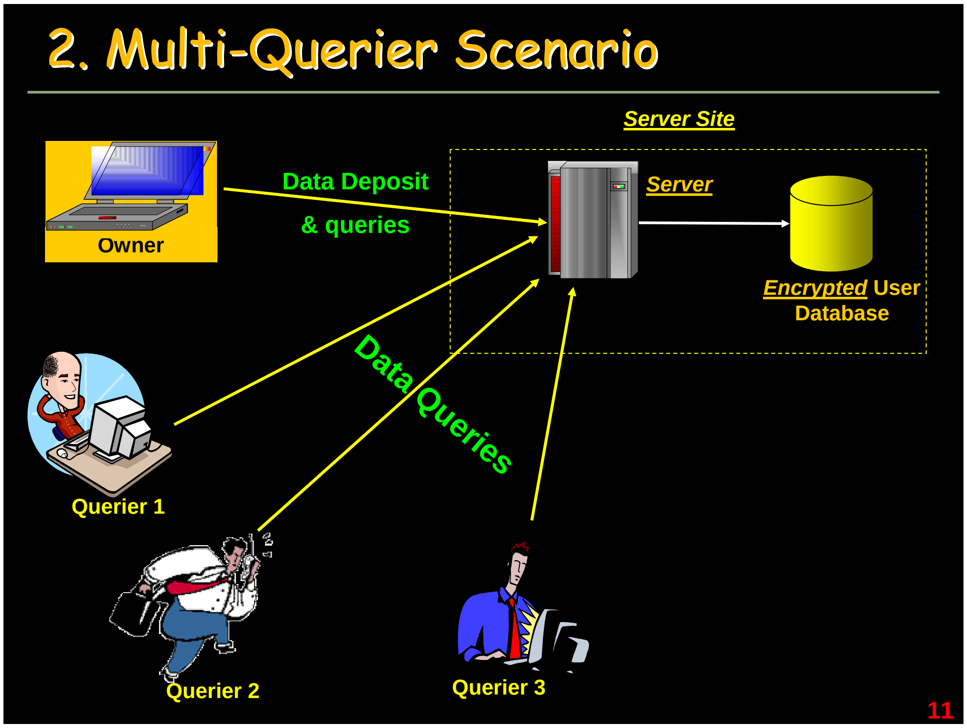
1. Unified Owner Scenario

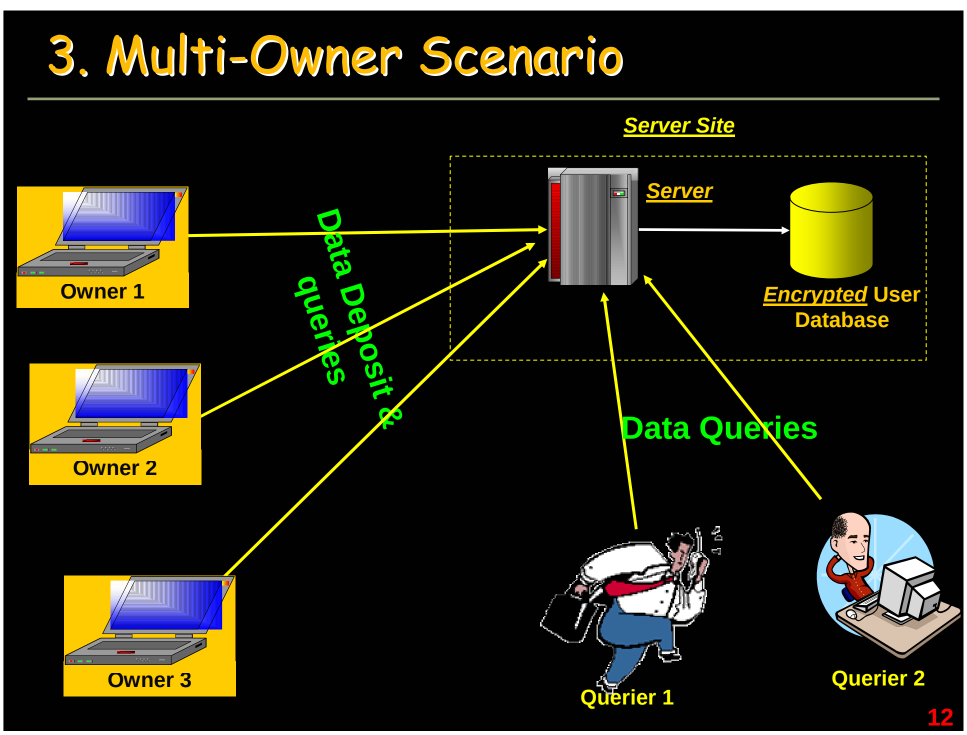


BTW:

• Owner may be anemic (battery, CPU, storage)
• Owner may have a slow/unreliable link
• Data “deposit” is << frequent than querying

10


2. Multi-Querier Scenario
Server Site
Data Deposit
& queries
Owner
Server
Encrypted User
Database
Data Queries
Querier 1
Querier 2
Querier 3
11


3. Multi-Owner Scenario
Server Site
Server
Encrypted User
Database
Owner 1
Data Deposit &
queries
Owner 2
Owner 3
Data Queries
Querier 1
Querier 2
12

Challenges

- Economic/business model?
  - How to charge for service, what kind of service guarantees can be offered, costing of guarantees, liability of service provider.
- Powerful interfaces to support complete application development environment
  - User Interface for SQL, support for embedded SQL programming, support for user defined interfaces, etc.
- Scalability in the web environment
  - Overhead costs due to network latency (data proxies?)
- **Privacy/Security**
  - Protection of outsourced data from intruders and attacks
  - Protecting clients from misuse of data by service providers
  - Ensuring result integrity+authenticity
  - Protecting service providers from "litigious" clients

13

Core Problem

We do not fully trust the service provider with sensitive information!

14

What do we mean by: "do not fully trust"?

Degrees of mistrust in Server:

1. **Trusted: outsider attacks only (e.g., on communication)**
   - **Encrypt data in transit, apply usual security measures**

2. **Partially trusted: break-ins, attacks on storage only**

3. **Untrusted: server can be subverted or become malicious**

15

Partially trusted server

Break-ins, attacks on storage

- Storage may be de-coupled from CPU
- Encrypt data "in situ", keep keys elsewhere
- Where: in CPU, in secure HW (tamper-resistant, or token-style), at user side, etc.

16

Secure and Efficient RDBMS Storage Model

- Need to reduce overhead associated with encryption
  - Today's storage models don't lend themselves to efficient encryption solutions
- Server is partially trusted
  - Data encrypted on disk, unencrypted in memory
- We developed RDBMS storage model to:
  - Reduce number of encryption calls (start-up cost dominates)
  - Reduce padding overhead: database attributes can be especially sensitive
    - 16 byte blocks: 2 byte attribute requires 14 bytes padding (w/AES)
  - Avoid over-encrypting: queries on non-sensitive data should run with minimal overhead

17

Secure and Efficient RDBMS Storage Model

- Start-up Cost
  - Includes creating key schedule
  - Start-up cost incurred for each encryption operation
  - Fine encryption granularity results in many encryption operations

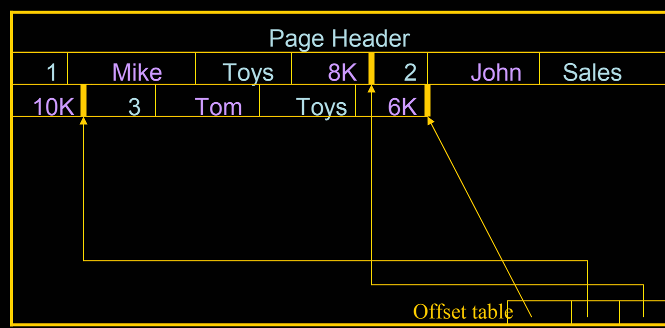
Encryption Algorithm	100 Byte * 100,000	120 Byte * 83,333	16 Kbytes * 625
AES	**365**	**334**	194
DES	372	354	229
Blowfish	5280	4409	**170**

Encryption of 10 Mbytes - all times in Msec

Fewer "large" encryptions better than many "small"

18

N-ary Storage Model (used today)

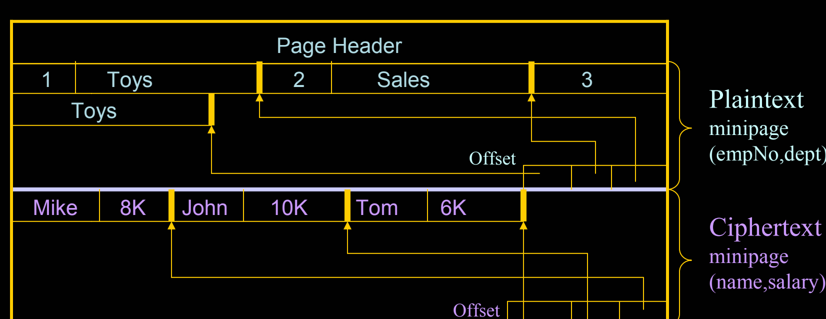


- Records stored sequentially
  - How do distinguish sensitive from non-sensitive?
  - Attribute level encryption (padding, cost)

19

PPC – Partition Plaintext Ciphertext Model (EDBT'04)

- Fewer "large" encryptions better than many "small"
- Create homogeneous mini-pages
- Distinguish sensitive from non-sensitive data
  - Maximum one encryption operation per page
  - Padding per mini-page (versus attribute / record)
  - No overhead when querying non-sensitive data



20

Untrusted server

Cannot trust server with database contents

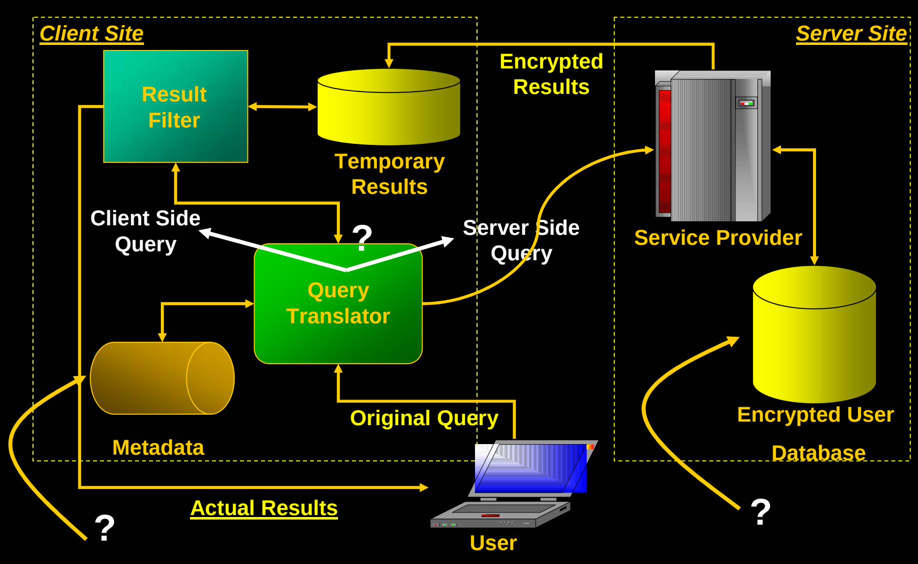
21

Rough Goals

- Encrypt client's data and store at server
- Client:
  runs queries over encrypted remote data
  and
  verifies integrity/authenticity of results
- Most of the work to be done by the server

22

System Architecture (ICDE'02)



23

Query Processing 101…

- At its core, query processing consists of:
  - Logical comparisons (> , <, = , <=, >=)
  - Pattern based queries (e.g., *Arnold*egger*)
  - Simple arithmetic (+, *, /, ^, log)
- Higher level operators implemented using the above
  - Joins
  - Selections
  - Unions
  - Set difference
  - …
- To support any of the above over encrypted data, need to have mechanisms to support **basic** operations over encrypted data

24

Fundamental Observation…

- Basic operations do not need to be fully implemented over encrypted data
- To test (AGE > 40), it might suffice to devise a strategy that allows the test to succeed in most cases (might not work in all cases)
- If test does not result in a clear positive or negative over encrypted representation, resolve later at client-side, after decryption.

25

Relational Encryption

NAME	SALARY	PIN
John	50000	2
Mary	110000	2
James	95000	3
Lisa	105000	4

etuple	N_ID	S_ID	P_ID
fErf!$Q!!vddf>></|	50	1	10
F%%3w&%gfErf!$	65	2	10
&%gfsdf$%343v<l	50	2	20
%%33w&%gfs##!	65	2	20

Server Site

- Store an encrypted string – *etuple* – for each tuple in the original table
  - This is called “row level encryption”
  - Any kind of encryption technique can be used
- Create an index for each (or selected) attribute(s) in the original table

26

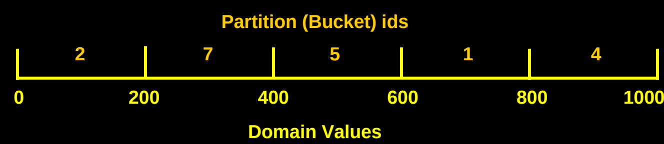
Building the Index:

- Partition function divides domain values into partitions (buckets)

  Partition (*R.A*) = { [0,200], (200,400], (400,600], (600,800], (800,1000] }

  - partition function has impact on performance as well as privacy
  - very much domain/attribute dependent
  - equi-width vs. equi-depth partitioning?

- Identification function assigns a partition id to each partition of attribute *A*



  - e.g. $ident_{R.A}$((200,400]) = 7
  - Any function can be use as identification function, e.g., hash functions
  - Client keeps partition and identification functions secret (as metadata)

27

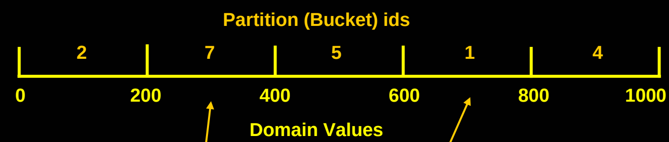
Bucketization / Partitioning / Indexing

- Primitive form of encryption, sort of a "substitution/permutation cipher"
- Can be viewed as partial encryption
- Works fine with warehoused data but needs to be periodically re-done with highly dynamic data
- Attacks (assume domain known)
  - Ciphertext only
  - "Existential" plaintext
  - Known plaintext
  - Chosen plaintext
  - Adaptive chosen plaintext

28

Mapping Functions (SIGMOD'02)

- Mapping function maps a value v in the domain of attribute A to partition id



– e.g., $Map_{R.A}(250) = 7$ $Map_{R.A}(620) = 1$

Storing Encrypted Data

$R = < A, B, C > \Rightarrow R^S = < etuple, A\_id, B\_id, C\_id >$

etuple = *encrypt* (A | B | C)

A_id = $Map_{R.A}$(A), B_id = $Map_{R.B}$(B), C_id = $Map_{R.C}$(C)

Table: EMPLOYEE

NAME	SALARY	PIN
John	50000	2
Mary	110000	2
James	95000	3
Lisa	105000	4

Table: $EMPLOYEE^S$

Etuple	N_ID	S_ID	P_ID
fErf!$Q!!vddf>></|	50	1	10
F%%3w&%gfErf!$	65	2	10
&%gfsdf$%343v<l	50	2	20
%%33w&%gfs##!	65	2	20

Mapping Conditions

Q: SELECT name, pname FROM employee, project
WHERE employee.pin=project.pin AND salary>100k

- Server stores attribute indices determined by mapping functions
- Client stores metadata and uses it to translate the query

Conditions:

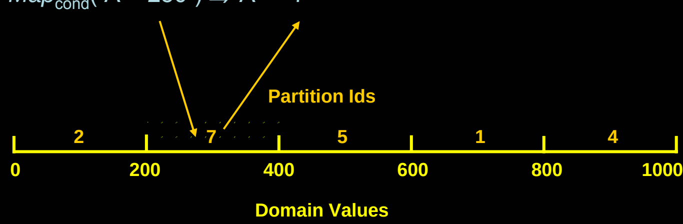
- Condition $\leftarrow$ Attribute *op* Value
- Condition $\leftarrow$ Attribute *op* Attribute
- Condition $\leftarrow$ (Condition $\vee$ Condition) | (Condition $\wedge$ Condition) | (not Condition)

31

Mapping Conditions (2)

Example: Equality

- Attribute = Value
  - $Map_{cond}(A = v) \Rightarrow A^S = Map_A(v)$
  - $Map_{cond}(A = 250) \Rightarrow A^S = 7$

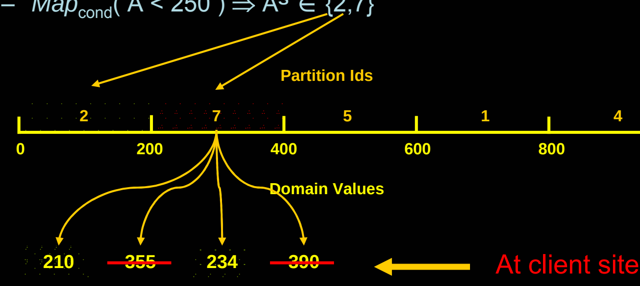


32

Mapping Conditions (3)

Example: Inequality (<, >, etc.)

- Attribute < Value
  - *Map*$_{cond}$(A < v) $\Rightarrow$ $A^S \in$ { ident$_A$(p_j) | p_j.low $\le v$) }
  - *Map*$_{cond}$(A < 250) $\Rightarrow$ $A^S \in$ {2,7}



Mapping Conditions (4)

- Attribute1 = Attribute2
  - *Map*$_{cond}$(A = B) $\Rightarrow \bigvee_N$ (A^S = *ident*$_A$(p_k) $\wedge$ B^S = *ident*$_B$(p_l))
    where N is $p_k \in$ *partition* (A), $p_l \in$ *partition* (B), $p_k \cap p_l \neq \varnothing$

Partitions	A_id
[0,100]	2
(100,200]	4
(200,300]	3

Partitions	B_id
[0,200]	9
(200,400]	8

C : A = B $\Rightarrow$ C' : (A_id = 2 $\wedge$ B_id = 9)
$\vee$ (A_id = 4 $\wedge$ B_id = 9)
$\vee$ (A_id = 3 $\wedge$ B_id = 8)

Relational Operators over Encrypted Relations

- Partition the computation of the operators across client and server
- Compute (possibly) superset of answers at the server
- Filter the answers at the client
- *Objective* : *minimize the work at the client* and process the answers as soon as they arrive *requiring minimal storage* at the client

Operators studied:

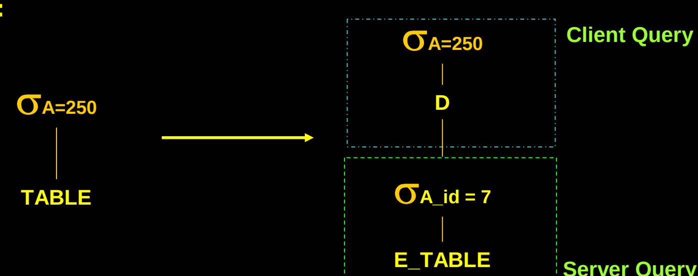
- **Selection**
- **Join**
- Grouping and Aggregation (in progress)
- Sorting
- Duplicate Elimination
- Set Difference
- Union
- Projection

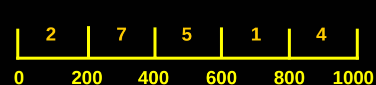
35

Selection Operator

$$\sigma_c(R) = \sigma_c(\mathbf{D} (\sigma^S_{Mapcond(c)}(R^S))$$

Example:



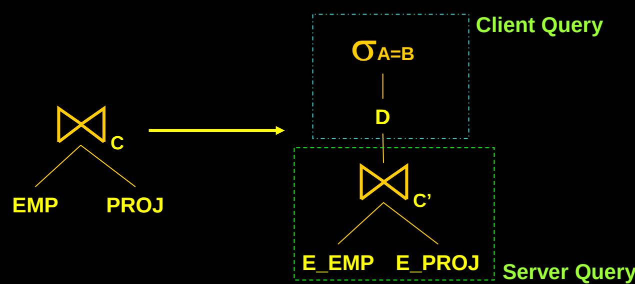


36

Join Operator

$$R \bowtie_c T = \sigma_c(\mathbf{D} (R^S \bowtie^S_{Mapcond(c)} T^S)$$

Example:



Partitions	A_id
[0,100]	2
(100,200]	4
(200,300]	3

Partitions	B_id
[0,200]	9
(200,400]	8

$C : A = B \Rightarrow C' : (A\_id = 2 \wedge B\_id = 9)$
$\vee (A\_id = 4 \wedge B\_id = 9)$
$\vee (A\_id = 3 \wedge B\_id = 8)$

37

Research Challenges..

- Aggregation queries, e.g., how to do: $\Sigma(a*b+c)$
  - RSA can do *
  - Pailler can do +
  - How to do both?
- Complex queries
  - Nested
  - Embedded
  - Stored procedures
  - Updates
- Query optimization
- Privacy guarantees
  - Against different types of attacks -- ciphertext only attack, known plaintext attack, chosen plaintext attack (work-in-progress)
- Generalized DAS models
  - What if there are more than a single owner and server?
  - Can the model work for storage grid environments
- Key management policies

38

Integrity and Authenticity in DAS

- Not all outsourced data needs to be encrypted
- Some data might be only partially encrypted
- At times, authenticity is more important, especially, in multi-querier and multi-owner scenarios
- This is different from query completeness, i.e., making sure that server returned all records matching the query

- Need to minimize overhead:
  1. Bandwidth, storage, computation overhead at querier
  2. Bandwidth, storage, computation overhead at owner?
  3. Bandwidth, storage, computation overhead at server?

39

Integrity and Authenticity in DAS

Challenge: **how to provide efficient authentication + integrity for a potentially large and unpredictable set of records returned?**

40

Integrity and Authenticity in DAS

- What granularity of integrity: page, relation, attribute, record?
- What mechanism: MACs, signatures?
- Not a problem in unified owner scenario (use MACs)
- For others: record-level signatures but what kind?
  - Boneh, et al. → aggregated multi-signer signatures
  - Batch RSA
  - Batch DSA or other DL-based signature schemes
  - Hash Trees and other data structures

41

Batch Verification of RSA Signatures

- Batching: useful when many signature verifications need to be performed simultaneously
- Reduces computational overhead
  - By reducing the total number of modular exponentiations
- Fast screening of RSA signatures (Bellare et al.):
  - Given a batch instance of signatures $\{\sigma_1, \sigma_2 \ldots \sigma_t\}$ on distinct messages $\{m_1, m_2 \ldots m_t\}$

$$\left(\prod_{i=1}^{t} \sigma_i\right)^e \equiv \prod_{i=1}^{t} h(m_i) \pmod{n}$$

where h() is a full domain hash function

42

Fast Screening

- Reduces querier computation but not bandwidth overhead
  - Individual signatures are sent to the querier for verification
- Bandwidth overhead can be overwhelming
  - Consider weak (anemic) queriers
  - Query reply can have thousands of records
  - Each RSA signature is at least 1024 bits!

Can we do better?

Condensed RSA (NDSS'04)

- Server:
  - Selects records matching posed query
  - Multiplies corresponding RSA signatures
  - Returns single signature to querier

Server		Querier
Given t record signatures: $\{\sigma_1, \sigma_2 \dots \sigma_t\}$, compute combined signature $\sigma_{1,t} = \prod \sigma_i \bmod n$. Send $\sigma_{1,t}$ to the querier	$\xrightarrow{\sigma_{1,t}}$	Given t messages: $\{m_1, m_2 \dots m_t\}$ and $\sigma_{1,t}$ verify combined signature: $(\sigma_{1,t})^e = ? = \prod h(m_i) \pmod n$

Condensed RSA

- Reduced querier computation costs
  - Querier performs *(t-1)* mult-s and a **one** exponentiation
- Constant bandwidth overhead
  - Querier receives a single RSA signature
- As secure as batch RSA (with FDH)

However, still can't aggregate signatures by different signers!

(an RSA modulus cannot be shared)

Condensed RSA → efficient for Unified-owner and Multi-querier but **NOT** great for Multi-owner

45

Batching DL-based signatures

- DL-based signatures (e.g., DSA) are efficient to generate
- Batch verification possible
- Unlike RSA, different signers can share the system parameters
  - → useful in the Multi-Owner Model?

Unfortunately, no secure way to aggregate DL-based signatures !

46

DL-based signatures…(cont'd)

- All current methods for batch verification of DL-based signatures require "small-exponent test"
- Involves verifier performing a mod exp (with a small exponent) on each signature before batching the verification.
  - Without this, adversary can create a batch instance which satisfies verification test without possessing valid individual signatures
- Thus, individual signatures are needed for verification
  ➔ aggregation seems impossible.

So far…

1. Condensed RSA
   - Cannot combine signatures by multiple signers
   - Querier computation, bandwidth overhead linear in # of signers
2. Batch DSA (and variants)
   - Can batch-verify signatures by distinct users and but cannot aggregate or condense
   - Querier computation as well as bandwidth overhead linear in # of signatures (records)!

Aggregated signature scheme by Boneh, et al.

- Signatures on different messages by multiple signers can be combined into one small signature.
- Scheme requires bilinear map (in Gap DH groups)
- BGLS Details:

Key Generation:

pick a random x Є Z_p and compute $v=g^x$

v - public key, x - secret key.

Signing:

let h = h(m) -- hash of message

$\sigma = h^x$

Aggregation:

To aggregate *t* signatures, compute their product $\sigma_{1,t} = \prod_{i=1}^{t} \sigma_i$

Verification:

Compute the product of the hashes and verify where e() is a computable bilinear mapping $e(\sigma_{1,t}, g) = \prod_{i=1}^{t} e(h_i, v_i)$

$$e(\sigma_{1,t}, g) = e(\prod_{i=1}^{t} (h_i^{x_i}, g) = \prod_{i=1}^{t} e(h_i, g)^{x_i} = \prod_{i=1}^{t} e(h_i, g^{x_i}) = \prod_{i=1}^{t} e(h_i, v_i)$$

49

Aggregated signature scheme by Boneh, et al.

- Applicable to all DAS flavors
- Constant bandwidth overhead
- For Unified-owner and Multi-querier, querier verification costs *(t-1)* EC mults (where t is # returned records) and two bilinear mappings
- For Multi-owner, verification of aggregated signature costs *(k+1)* bilinear mappings (where k is # signers) and (*t-k*) multiplications
  - Bilinear mappings are expensive
  - Computing a single mapping in F_p (for |p|=512) on a 1GHz PIII takes 31 msecs!

50

Cost Comparisons

1. Querier computation:

(P3-977MHz, Time in mSec)

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

Parameters:

For RSA: |n| = 1024
For DSA: |p| = 1024 and |q| = 160
For BGLS: Field Fp with |p| = 512

51

Cost Comparisons

2. Bandwidth overhead:

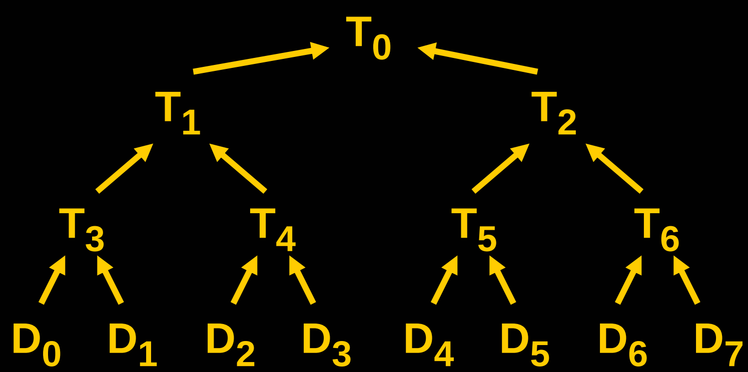
(unit: bits)

	Condensed RSA	Batch DSA	BGLS
1 signature	1024	1184	**512**
t =1000 sigs, k=1 signer	1024	1184000	**512**
t =100 sigs, k=10 signers	10240	1184000	**512**
t =1000 sigs, k = 10 signers	10240	11840000	**512**

52

Merkle Hash Tree (MHT)

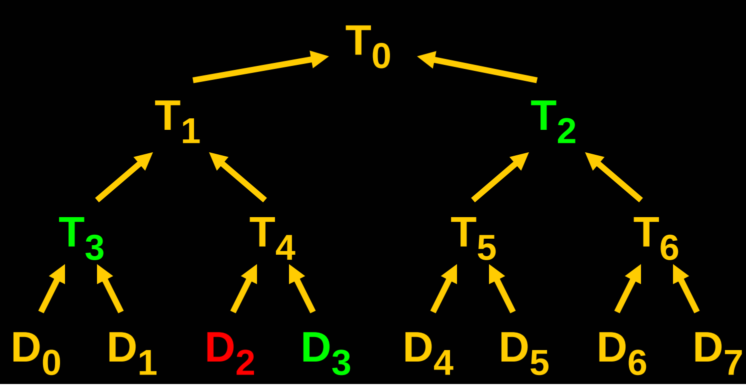
- Authenticate a sequence of data values D_0 , D_1 , …, D_N
- Construct binary tree over data values



53

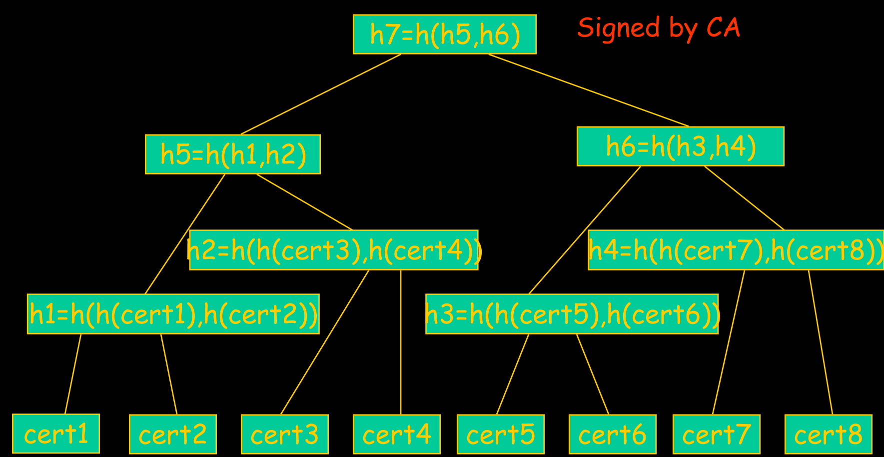
MHT contd.

- Verifier knows T_0
- How can verifier authenticate leaf D_i ?
- Solution: re-compute T_0 using D_i
- Example authenticate D_2 , send: D_3 , T_3 , T_2
- Verify T_0 = H(H(T_3 || H(D_2 || D_3)) || T_2)



54

MHT Example -- Certificate Revocation Tree



55

- Can use MHTs with leaves representing records and the root signed by the owner
  - Authentic 3rd party publishing
  - Prior work by Martel, Stubblebine, Devanbu, et al.
- For Multi-owner scenario:
  - Individual trees for each owner OR
  - A single tree with a shared signing key among all owners
  - Mixed tree

56

MHT contd.

As a response to a posed query, server

1. Selects records that match query predicate
2. Sends records along with hashes on co-paths for each record.
3. Attach a single signature corresponding to root of hash tree

Upon receiving query reply, querier

1. Computes hashes of all records returned
2. Using hashes of nodes on co-paths, computes hashes for each path to the root
3. Verifies signature of root node

57

MHT Overhead

- For n leaf nodes and t records in the query reply
  - Lower server-storage overhead compared to per-record signatures
    - At most: (2n-1)*|hash| + |sig| as opposed to n*|sig|
  - Record insertion (owner computation overhead) requires 2 extra rounds of communication
    - to make structural changes to the tree
  - Querier computation cost lower since verification involves computing hashes
    - Compared with Combined RSA which involves mod mults…
  - However, bandwidth overhead increases!
    - Hashes for all nodes on co-paths must be supplied

58

Bandwidth overhead

- Expected overhead
  - For n leaf nodes and t records in query reply
  - Let $n=2^h$ and wlog, let P(a leaf node is selected) = t/n
  - Expected # of additional hashes (non-leaf nodes) returned is given by:

$$\sum_{k=0}^{h-1} 2^{h-k}\left(1-\left(1-\frac{t}{n}\right)^{2^k}\right)\left(1-\frac{t}{n}\right)^{2^k}$$

e.g., if h=30, t=1024, and |hash| = 160 then,
Bandwidth overhead = 3,132,000 bits
(for combined RSA, 1024 bits)

59

In conclusion…

- No clear winners!
- MHTs: good for computation, bad for bw and dynamic databases
  - Can be used to guarantee query completeness (for range queries)
  - Needs a sorted MHT for each attribute
- Currently investigating hybrid model
- Is it possible to aggregate/condense DSA-like signatures?
- Is it possible to aggregate multi-signer RSA?
- Any new efficient and practical signature scheme that allows multi-signer aggregation?
- How to prevent mutability in aggregated/condensed signatures?

60

Related Work

- Authentic 3rd party publishing
- Private information retrieval (PIR)
- Searching encrypted data for keywords
  - Boneh, et al.
  - Song, et al.
- Encrypted aggregation
  - Privacy Homomorphisms (Rivest, et al.)
- Watermarking databases
  - Attallah, et al.
- Privacy-preserving data mining
  - Agrawal, et al.
- Batch signature verification (RSA, DSA, etc.)

61

Some references

1. Hakan Hacigumus, Bala Iyer, Chen Li and Sharad Mehrotra
   Executing SQL over Encrypted Data in the Database-Service-Provider Model
   SIGMOD 2002

2. Hakan Hacigumus, Bala Iyer and Sharad Mehrotra
   Providing Database as a Service
   ICDE-2002

3. Maithili Narasimha, Einar Mykletun and Gene Tsudik
   Efficient Data Integrity in Outsourced Databases
   NDSS 2004

4. Bala Iyer, Sharad Mehrotra, Einar Mykletun, Gene Tsudik and Yonghua Wu
   A Framework for Efficient Storage Security in RDBMS
   EDBT 2004

5. Bijit Hore, Sharad Mehrotra and Gene Tsudik
   A Privacy-Preserving Index for Range Queries
   in submission

6. Maithili Narasimha, Einar Mykletun and Gene Tsudik
   Signature Bouquets: Immutability for Aggregated Signatures
   in submission

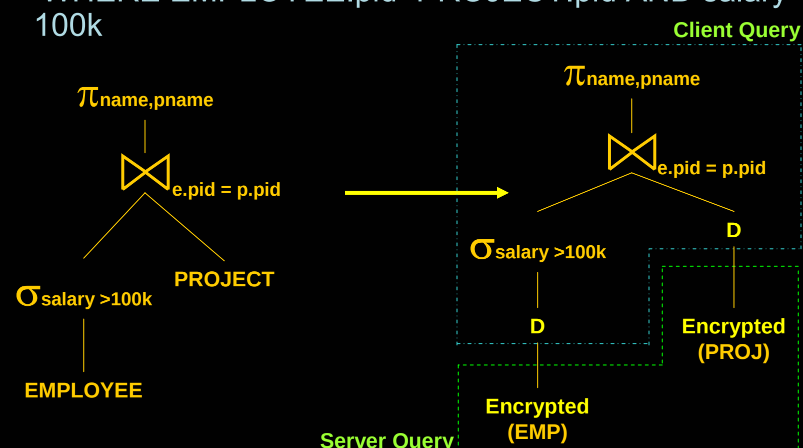
62

Thank you!

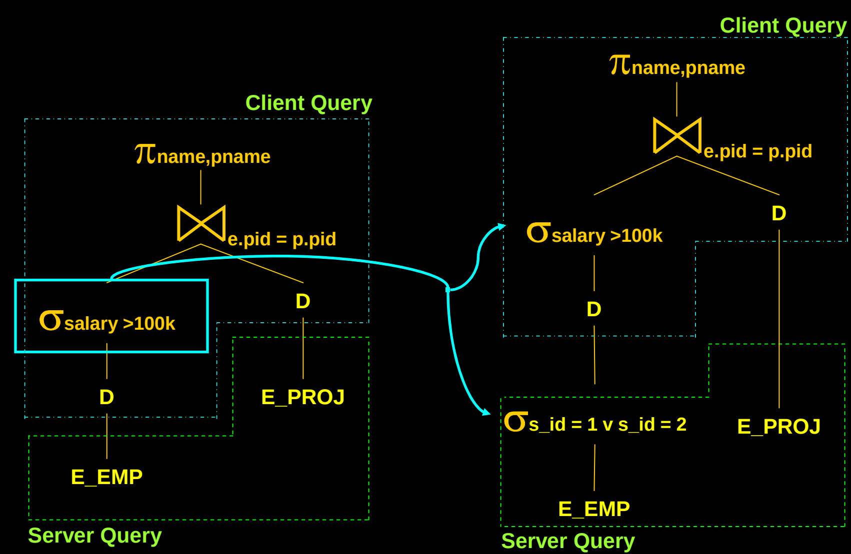
Questions?

Query Decomposition

Q: SELECT name, pname FROM EMPLOYEE, PROJECT WHERE EMPLOYEE.pid=PROJECT.pid AND salary > 100k

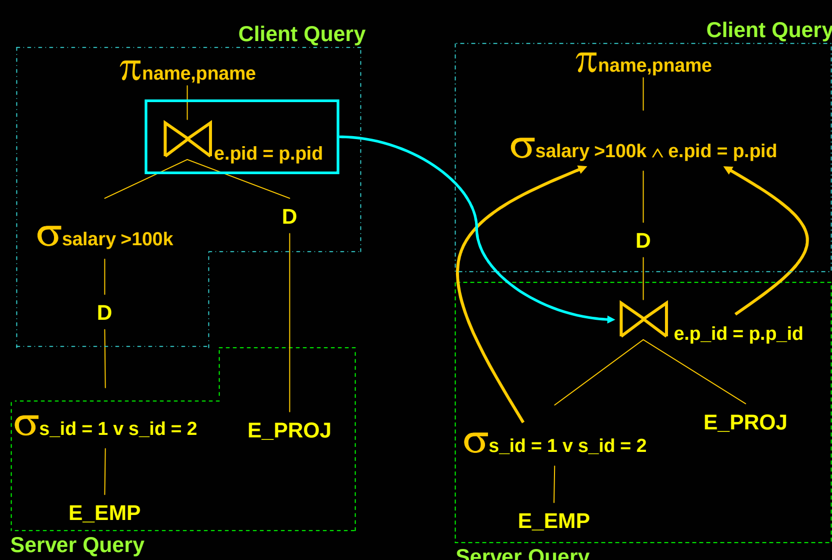


Query Decomposition (2)



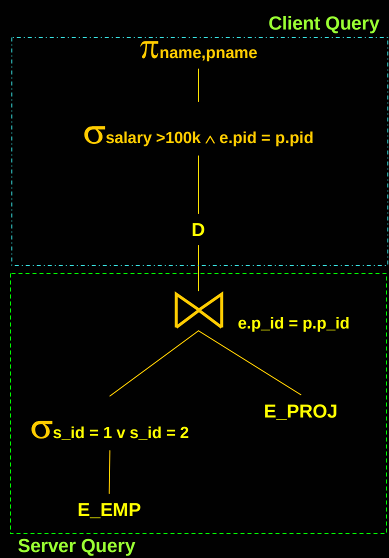
65

Query Decomposition (3)



66

Query Decomposition (4)




```
Q:  SELECT   name, pname
      FROM   EMPLOYEE, PROJECT
    WHERE
             EMPLOYEE.pid=PROJECT.pid
             AND salary > 100k
```

```
QS: SELECT e_emp.etuple, e_proj.etuple
      FROM   e_emp, e_proj
    WHERE
             e.p_id=p.p_id AND
             s_id = 1 OR s_id = 2
```

```
QC: SELECT   name, pname
      FROM   temp
    WHERE
             emp.pid=proj.pid AND
             salary > 100k
```

67