Secure Data Management at UW

Gerome Miklau and Dan Suciu

Trustworthy computing seminar
Autumn 2004

Today's talk

1 Controlled data publishing
secrecy: crypto + XML [VLDB 2003]

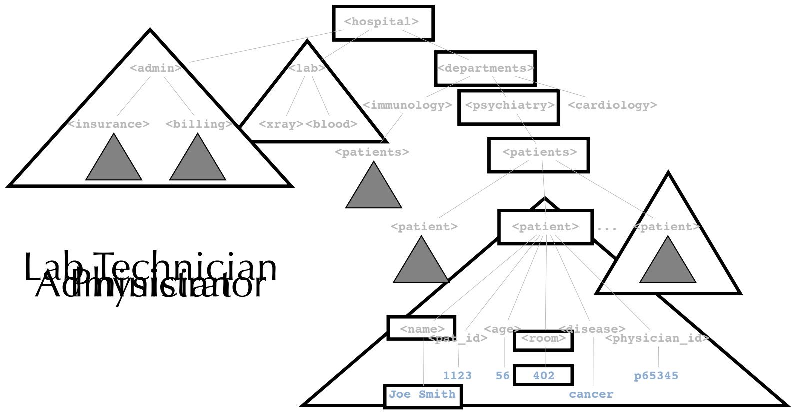
2 Analyzing information disclosure
secrecy: theory + relations [SIGMOD 2004] [ICDT 2005]

3 Tamper-resistant databases
integrity: crypto + relations [Current work]

Ingredients for data sharing

- File formats and tools
  - XML, XML query languages
- Distributed processing
  - Networked data sources
  - Mediator systems, distributed systems
- Access control
  - controlled distribution of data

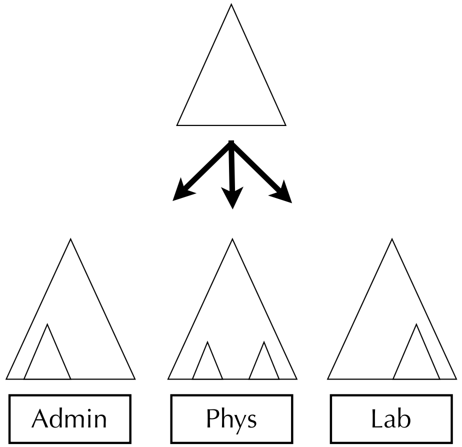
XML data and access rights



Publishing Many Views

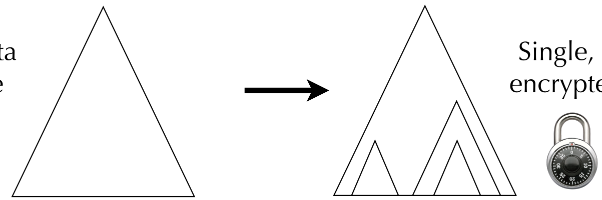
XML data source

3 versions published



Publishing one view + keys

XML data source



Single, partially-encrypted version

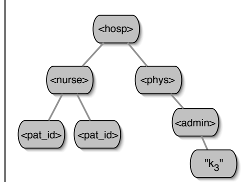
3 sets of keys transmitted

{k1,k2}	{k2,k3}	{k1}
Admin	Phys	Lab

Publishing protected data

① Policy Query	② Tree Protection	③ XML Output

```
FOR     $x in /doc/subj/subj
        $y in /doc/psy/psych
WHERE   $x/exam-psych/id = $y/id
KEY     getKey( $y )
TARGET  $x
```



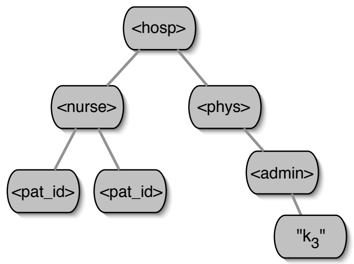

```
<EncryptedData>
  <EncryptionMethod Alg="AES"
KeySize="128"/>
  <KeyInfo>
    <name>k1</name>
  </KeyInfo>
  <CipherData>
    <CipherValue>
    qZk+NkcGgWq6PiVxeFDCbJz
    DCbJzQ2JkcGgWq6PiVxeFFD
    </CipherValue>
  </CipherData>
</EncryptedData>
```

Evaluate policy on data

Encrypt, write as XML

Tree Protection

- XML data tree guarded with formulas over keys.
- Node is accessible if "satisfying" keys are supplied.



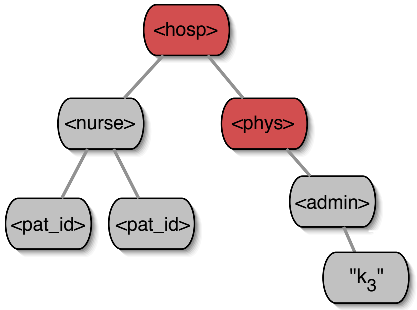
Grammar for key formulas

$\sigma := T \mid F \mid k \mid \sigma \wedge \sigma' \mid \sigma \vee \sigma'$

Access Function

Given a set of keys K,
access(K) computes the accessible nodes in a tree protection.

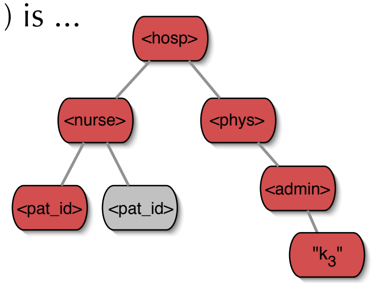
access({ k_1 }) is:



Access Function

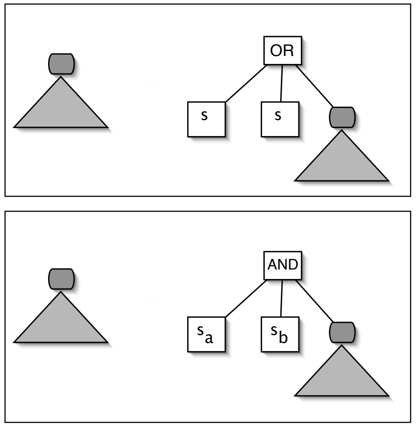
Access function can discover data values and use them as keys.

Example: access({ k_1, k_2 }) is ...



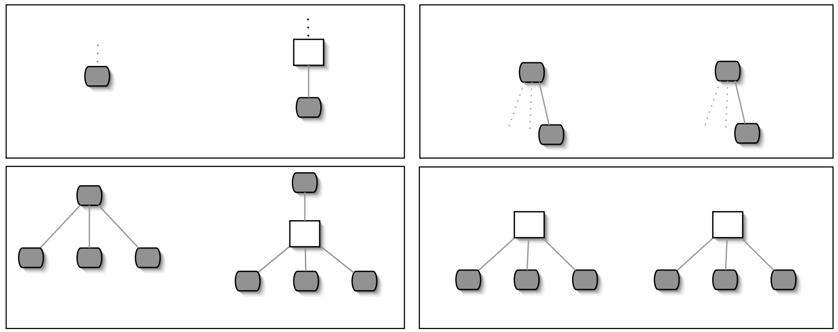
Normalization Rules

- a tree protection is normalized if every formula is atomic
- soundness: access function invariant under rewritings



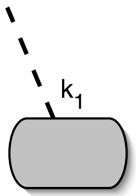
Benaloh and Leichter. CRYPTO 1988

Rewrite rules



Rewrite rules can be used to optimize formula expressions.

Implementing a Protection




```
...
<EncryptedData>
  <EncryptionMethod Alg="AES" KeySize="128"/>
  <KeyInfo>
    <name>k1</name>
  </KeyInfo>
  <CipherData>
    <CipherValue>
    qZk+NkcGgWq6PiVxeFDCbJz
    DCbJzQ2JkcGgWq6PiVxeFFD
    </CipherValue>
  </CipherData>
</EncryptedData>
...
```

Leaf node

Encrypted leaf node, following XML Encryption Recommendation

Security

Security Property

given document D, set of keys K:

- if node x ∈ access(K) then x is efficiently computable from D.
- if node x ∉ access(K) then the best algorithm for finding x requires guessing keys.

Additional disclosure:

number of children of a node

size of ciphertext, duplicate subtrees,

policy information.

Today’s talk

1 Controlled data publishing

secrecy: crypto + XML

2 Analyzing information disclosure

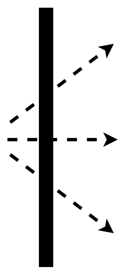
secrecy: theory + relations

3 Tamper-resistant databases

integrity: crypto + relations

Protecting data using views

confidential database D



View V1(D) Alice

View V2(D) Bob

View V3(D) Carol

Problem statement

Query - View security

- a published view V
- a sensitive query S
- <u>Does V reveal anything about S</u> ?

Spectrum of disclosure

<u>Employee</u> (name, dept, phone)

Published View	Sensitive Query	Disclosure?
V_1(name, phone)	S_1(name)	**total**

Insecure
Secure
Insecure
Secure

Existing techniques

- Logical inference
- Answering queries using views
- Statistical databases

exact disclosure

aggregation of numerical attributes

Our basic strategy

Compare Mallory's knowledge
about sensitive query S:

Knowledge about S, *a priori* $\overset{?}{=}$ Knowledge about S, given V

When these are equal,
V provides ***no information*** about S

The adversary's knowledge

Knowledge about S, *a priori*	$\stackrel{?}{=}$	Knowledge about S, given V
• Schema • Domain • Probability of each database		+ V + V(I) = ans
Probability of each answer to S	$\stackrel{?}{=}$	Probability of each answer to S

Probabilities of databases

- Fix schema R, and domain.
- For each tuple t: $0 \leq \Pr[t] \leq 1$
- The probability of a database instance I is:

$$\Pr[I] = \prod_{t \in I} \Pr[t] \times \prod_{t \notin I} (1 - \Pr[t])$$

assumption: tuple-independence

Definition: query-view security

S and V are secure (denoted S|V) if:

Pr[S(I)=x] = Pr[S(I)=x | V(I)=y] ∀x∀y

a priori knowledge — **knowledge given V** — **must hold for all answers**

- independence of probabilistic events
- inspired by Shannon's perfect secrecy [1949]

Concrete example

- relation Edge(X,Y)
- nodes={a,b}
- tuple probability = 1/2
- possible graphs:

1	
2	(a,a)
3	(a,b)
4	(b,a)
5	(a,a) (a,b)
6	(a,a) (b,a)
⋮	
15	(a,a) (a,b) (b,a)
16	(a,a) (a,b) (b,a) (b,b)

Sensitive query:
S(x) :- Edge(x,y)

for S(I)={ (a) },
Pr[S(I)={ (a) }] = **3/16**

Published view:
V(y) :- Edge(x,y)

for V(D)={ (a) },

Pr[S(I)={ (a) } | V(I)={ (a) }]
= **1/3**

Goal: logical criterion

<u>Brute force</u> ⟶ <u>Logical criterion</u>

For each
answer to S
answer to V
Compare probabilities

depends on domain &
probability distribution

Analyze the <u>view expressions</u> S and V

Deciding query-view security

Theorem

Given query S and view V, deciding whether S | V is Π_2^P-complete

when S and V secure, then they are secure:

- for any (sufficiently large) domain
- for any probability distribution

Other consequences

Supplemental knowledge

- security standard easily extended
- compare

Encrypted view

- no sensitive query is secure

Supplemental info: cardinality

- no sensitive query is secure

Measuring disclosure

When query-view security fails:

for some x,y:
Pr[S(I)=x] $\neq$ Pr[S(I)=x | V(I)=y]

how do we evaluate the difference?

- magnitude, or closeness to 1?
- aggregate over many answers
- see paper for limited case.

Partial disclosure

- Intuition
  - domains are large
  - databases are small (and of known size)
- New probabilistic model
  - each tuple t equally-likely...
  - prob[t] s.t. database size constant
- Practical security
  - lim Pr(S | V) = 0 as domain $\rightarrow \infty$

[Dalvi, Miklau, Suciu. ICDT 2005 (to appear)]

Today's talk

1 Controlled data publishing
secrecy: crypto + XML

2 Analyzing information disclosure
secrecy: theory + relations

3 Tamper-resistant databases
integrity: crypto + relations

Integrity

Data integrity

an assurance that data has not been modified by an unauthorized party.

Consistency

an assurance that the items in a collection are "fresh".

Query integrity

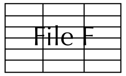
an assurance that a query answer is correct and complete.

Tampering threats to DB systems

- DB access control vulnerabilities
  - specification failure
  - enforcement failure
  - subversion (e.g. sql injection, weak authentication)
- DB extensions - user defined functions
- general OS and network threats
- privileged parties: OS admin, DB admin
- service provider model

Data integrity with hashing

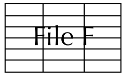
Client



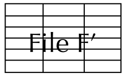
compute y=hash(F)

send F to server →

Server
(untrusted)



← retrieve F′ from server

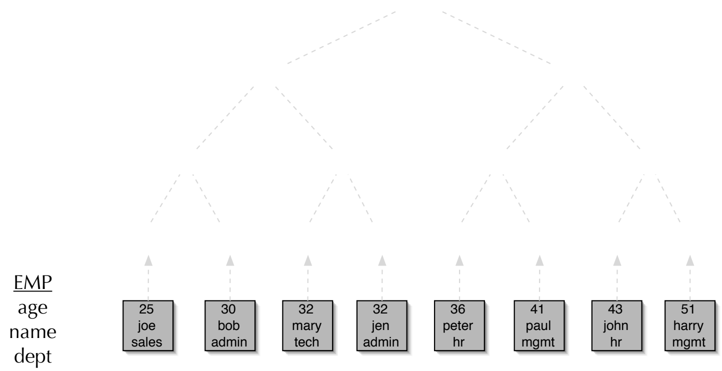


verify:
compute y′=hash(F′)
check y = y′

Problems:
block retrieval
updates

Merkle hash tree on relation

Client stores root hash locally



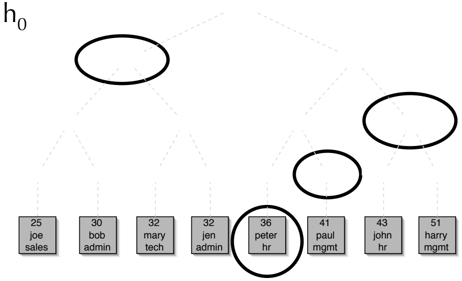
[Merkle 1989] [Devanbu, et al. 2001]

Verifiable query (1)

CLIENT: Give me the record for 'peter'

SERVER: answer is t_5
hash path is h_{101}, h_{11}, h_0

CLIENT: verify by computing
new root hash h'_ϵ
check $h'_\epsilon = h_\epsilon$



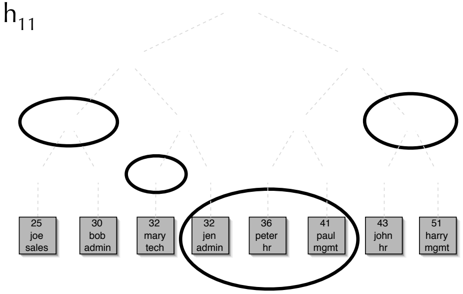
This proves correctness and consistency

Verifiable query (2)

CLIENT: Return all employees with age between 35 and 40

SERVER: answer is t_5. Also return t_4, t_6.
hash path is h_{010}, h_{00}, h_{11}

CLIENT: verify by computing
new root hash h'_ϵ
check $h'_\epsilon = h_\epsilon$



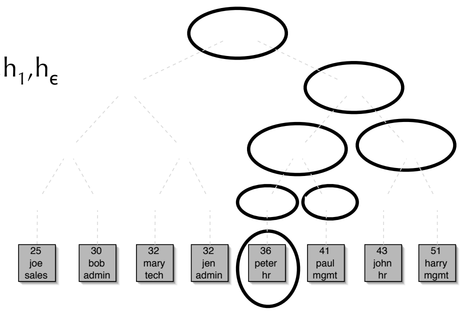
This proves correctness, completeness, and consistency

Updating the database

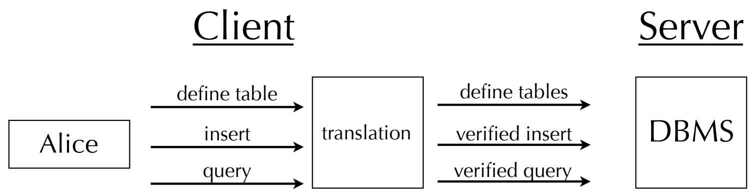
CLIENT: Move Peter from department HR to MGMT

-- verify peter's record --

CLIENT: compute new $h_{100}, h_{10}, h_1, h_\epsilon$
send to server



Implementing a tamper-resistant database



- Smart client, oblivious server
- Relational representation of hash tree
- Query definition
- Index selection

Cost of integrity

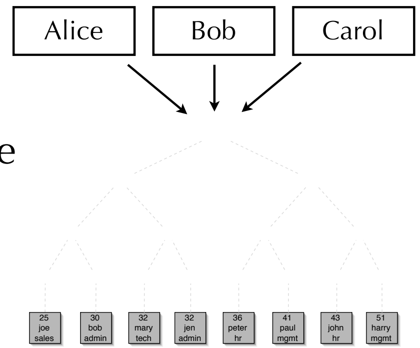
- Reasonable communication overhead
- Reasonable client computation
- Modest storage overhead
- Good scalability
- Throughput:

preliminary results

	integrity	no integrity	multiple
Query	2.0 ms	.4 ms	5.0
Range query	6.1 ms	1.3 ms	4.7
Insert	8.3 ms	.8 ms	10

Multiple clients

How do we manage integrity with multiple users?



Questions ?