A Framework for "Efficient" Storage Security in RDBMS

B.Iyer*, S. Mehrotra^, E. Mykletun^, G. Tsudik^, Y. Wu^

* IBM Silicon Valley Lab

^ CS Dept, University of California, Irvine

Research supported by NSF ITR Grant CCR-0220069
"Privacy in the Database-as-a-Service (DAS) Model"

Outline of Talk

- The need for database security and privacy
- Problems with current solutions
- Outsourced Database Model
- Encryption cost and level of granularity
- Our storage model - PPC
- Experiments / Performance (TPC-H)
- Conclusion

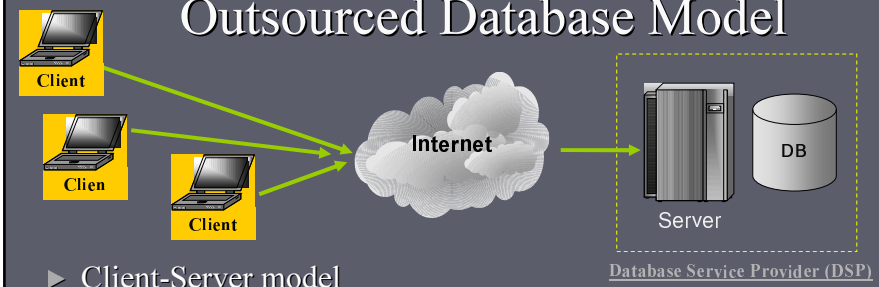
Motivation

- Concerns about security & privacy prompted new legislations
  - Anyone storing sensitive data must do so in encrypted fashion
- Trend: vendors adding security/privacy features to existing products
  - E.g., Oracle, IBM have DBMS-s supporting encryption
- Coming up with efficient security solutions requires understanding of:
  - Points of vulnerability
  - Attack model
  - Encryption and integrity granularity
  - Choice of encryption function(s) to use
  - Key management issues
- Challenge
  - Introduce security functionality without too much overhead (performance and storage)

Purpose

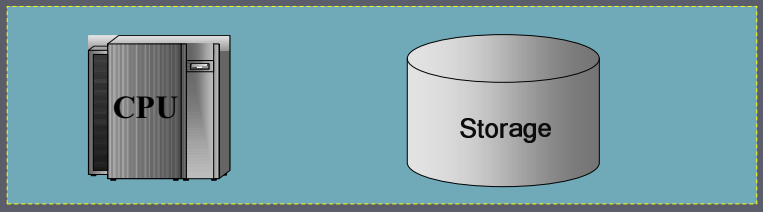
- Adding privacy/security features **as an afterthought** results in poor performance
- Efficient solutions require fundamental changes to underlying storage implementation
  - Minimize number of encryption operations
- We suggest a new DMBS storage model
  - Group sensitive data to minimize # of encryption operations → minimize crypto overhead
  - Take advantage of bulk encryption performance
  - Minimize risks of key/data compromise
- Motivated by the Outsourced Database paradigm

Outsourced Database Model



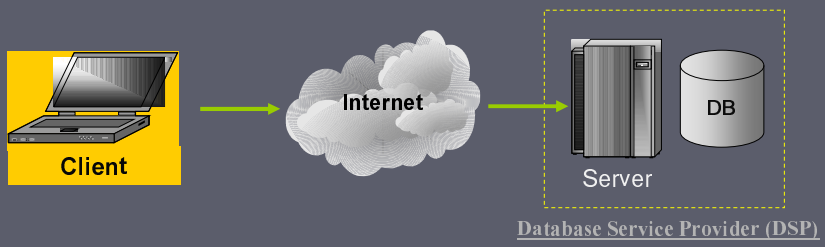
- Client-Server model
  - Client stores both sensitive and non-sensitive data at the server
  - Server acts as an outsourced database application
    - Protects client's sensitive data through encryption and access control
    - Ensures data's confidentiality and prevent unauthorized access
- Trust in server
  - Ranges from fully trusted to fully un-trusted
  - We focus on partially trusted
  - Fully un-trusted = Database-as-a-Service (DAS)
    - Server runs queries over encrypted data
    - Ongoing work at UCI

Partially Trusted Server



- Untrusted Storage – subject to takeover
- Trusted CPU
  - Viable if:
    - No possibility of compromise
    - Temporary compromise while CPU has no access to keys

Points of Vulnerability



- Client-server communication
  - Client queries can involve sensitive inputs
  - Query reply can contain sensitive data
  - SSL (and/or IPSec) is a *de facto* standard/solution
    - Not discussed any further
- Data stored at the server
  - Storage not protected → can be compromised
  - Stored data needs to be encrypted and integrity-protected
  - Protect against outsiders and malicious insiders

Problems with today's DB security solutions

- Poor efficiency
  - Added security introduces significant computational overhead
  - Mainly due to underlying model used by DBMS
  - Difficult to find efficient solutions without modifying the way in which records are stored in blocks on disk
- Inflexibility – wrong encryption granularity
  - Hard to differentiate between sensitive and non-sensitive data
  - Over-encryption: non-sensitive data is encrypted
- Indexes are often not encrypted
  - Some database vendors do not support encrypted indexes
  - Some build indexes based on encrypted data
    - Most encryption is not order-preserving → lose range query functionality
    - Assume selection – requires deterministic encryption, susceptible to statistical attacks

Encryption Costs Example – 10 MBytes

- Start-up Cost
  - Includes creating key schedule
  - Start-up cost incurred for each encryption operation
  - Blowfish has the highest start-up cost (key expansion)
- Encryption Speed
  - Blowfish > AES > DES
  - Bulk encryption is fastest: fewer encryption "invocations"
- Fewer "large" encryptions better than many "small"

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

All times in msecs

Encryption Granularity

- Granularity affects performance
  - Too Fine
    - Too many encryption calls
    - Can cause prohibitively high CPU (and sometimes storage) overhead
  - Too Coarse
    - Inflexibility when dealing with mixed data
    - Unnecessary encryption operations, i.e., non-sensitive data is encrypted
- Choices?
  - Attribute value
  - Record
  - Attribute / Column – only sensitive attributes are encrypted
  - Page / Block

Encryption Granularity

Record ID	Age	Salary
1	27	40K
2	38	50K
3	31	44K
4	60	50K

Attribute

- **Attribute-Level Integrity**
  - Can't use deterministic encryption (leaks information)
  - Need to "factor in" a unique random value
  - Padding can significantly increase storage overhead
    - AES uses 16 byte blocks: 2-byte attribute → 14 bytes of padding
  - Advantage: can implement SELECTIVE per-attribute policies

Encryption Granularity

Record ID	Age	Salary
1	27	40K
2	38	50K
3	31	44K
4	60	50K

Record

- **Record Level Integrity**
  - Reduce # of encryption ops and padding required
  - Does not differentiate between sensitive and non-sensitive data

Encryption Granularity

Record ID	Age	Salary
1	27	40K
2	38	50K
3	31	44K
4	60	50K

Attribute / Column

- **Attribute / Column Level Integrity**
  - Only sensitive attributes are encrypted
  - No encryption overhead when querying over non-sensitive data
  - But, often need to "jump" into the middle of column to decrypt

Encryption Granularity

Record ID	Age	Salary
1	27	40K
2	38	50K
3	31	44K
4	60	50K

Page / Block

- **Page / Block Level Integrity**
  - Entire page is encrypted as a unit
  - Does not differentiate between sensitive and non-sensitive data

Encryption Modes and Key Management

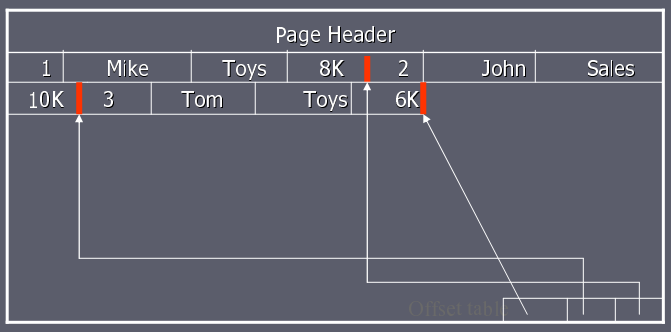
- Stream or Block?
  - Stream is cheap but causes problems with updates…
- Pure Block or Chained?
  - Pure block – overhead (need block #)
- Chained-Block-Cipher (CBC)

- Key Granularity
  - Per encrypted unit?
  - Multiple units?
- "Gospel": minimize amount of ciphertext under same key

Outline of Talk

- The need for database security and privacy
- Problems with current solutions
- Outsourced Database Model
- Encryption cost and level of granularity
- Storage model - PPC
- Experiments / Performance (TPC-H)
- Conclusion

N-ary Storage Model

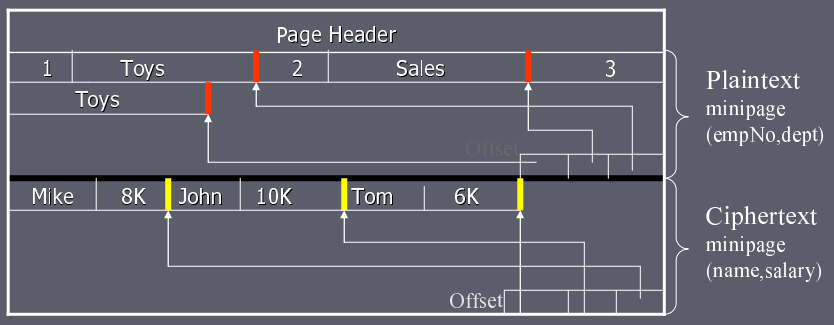


- Records stored continuously within page
- Ill-suited for encryption
  - Doesn't differentiate between mixed attributes
  - Minimum one encryption and padding per record

PPC: Partitioned Plaintext-Ciphertext Model

- Our goals
  - Minimize # of encryption operations
  - Separate sensitive and non-sensitive data
    - Create homogeneous mini-pages
  - No overhead when querying non-sensitive data
  - Reduce amount of padding overhead
    - Each mini-page needs padding (versus each attribute / record)
- Based upon "Weaving Relations for Cache Performance", by Ailamaki, et al.
  - Divide each page into many minipages – one attribute per minipage
  - In-page data placement is key to performance
  - Improves cache performance by loading relevant part of record into cache

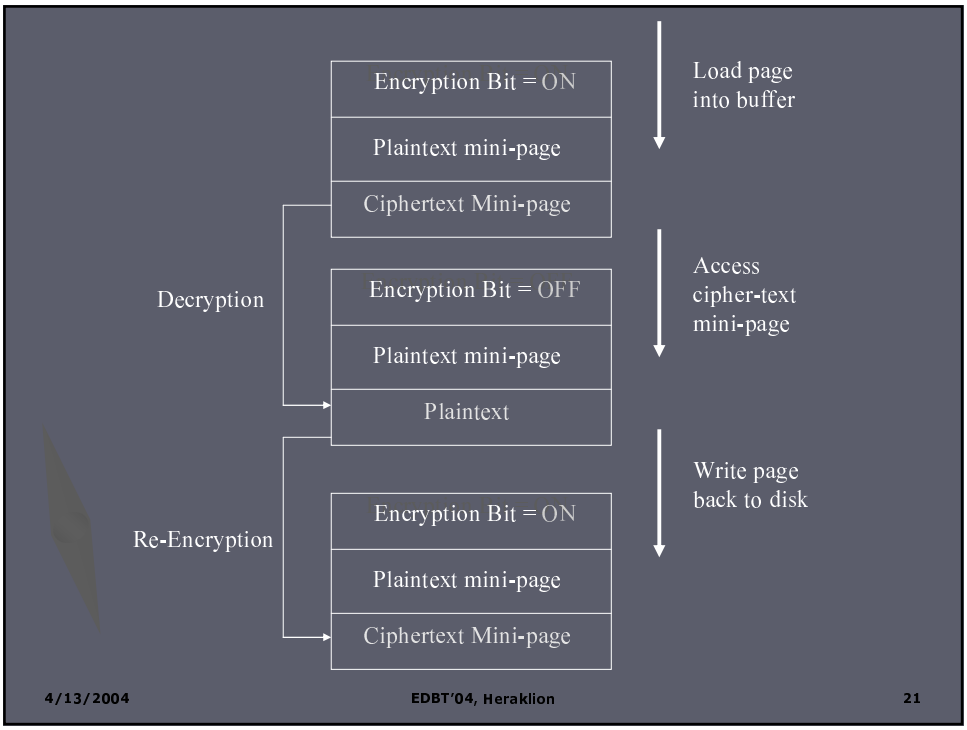
PPC Page Layout



- Differentiate between non-sensitive and sensitive data
  - Plaintext and Ciphertext Mini-pages
- Layout amenable to efficient use of encryption
  - Reduces # of encryption operations
  - Only one padding required
  - Can take benefit from bulk encryption speeds

Buffer Manager Support

- Introduce "encryption bit" in page header
  - ON indicates that the ciphertext mini-page is encrypted.
- Load page into buffer
  - Same as loading a regular page
- Access page in the buffer
  - If ciphertext mini-page is accessed and encryption bit ON
    - Decrypt and set encryption bit to OFF
- Write page back to the disk
  - If encryption bit is OFF, re-encrypt ciphertext mini-page



Schema Changes

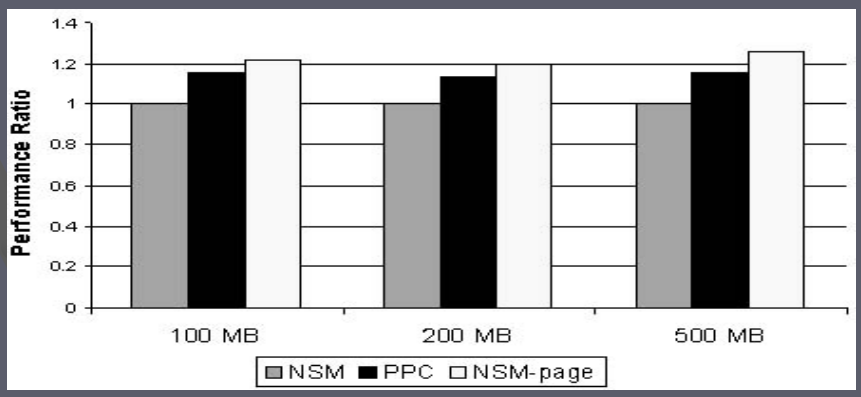
- PPC stores schema versions of each relation in catalog file
  - Header of each record contains schema version
  - Can get the schema by only looking up in the sub-record
- Modifying attribute classification
  - Previously non-sensitive to sensitive
    - Lazy encryption
  - Previously sensitive to non-sensitive
    - When ciphertext mini-pages are accessed

Experiments

- Implemented PPC
  - MySQL, Version 4.1.0-alpha
  - Modified InnoDB storage model
  - Altered page/record structure to create plaintext and ciphertext mini-pages
- Comparison between
  - NSM no encryption
  - NSM page level encryption (NSM-page)
  - PPC
- Experiments – TPC-H data set
  - Bulk Data Insertion
  - Varying # of encrypted attributes
  - Select TPC-H queries

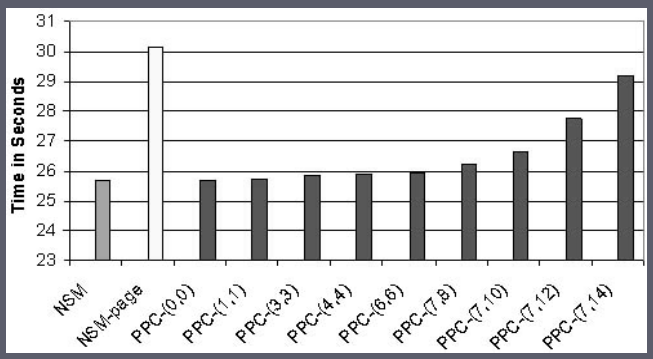
Bulk Data Insertion

- At least one attribute from each table is marked sensitive
- Loading time for bulk insertion of: 100, 200, and 500 MB
- Overhead incurred by encryption solutions (relative to NSM without encryption)
  - NSM-page = 24%
  - PPC = 15%
- PPC outperforms NSM-page since less data gets encrypted
  - Non-sensitive data is not encrypted



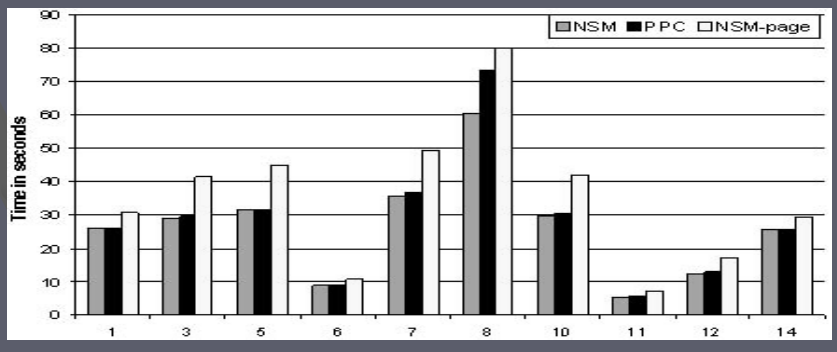
Varying # of encrypted attributes

- Idea
  - Isolate single table *lineitem*. TPC-H Query 1.
  - Analyze performance of PPC as we increase number of sensitive attributes
  - PPC-(x,y) means that y attributes in *lineitem* were marked sensitive, x were included in query
- Performance
  - Overhead is minimal when adding individual sensitive attributes
  - PPC with few sensitive attributes performs almost identical to NSM
  - Last two columns have spikes because the added sensitive attributes are *l_shipinstruct* and *l_comment*, the largest attributes in lineitem



Selected TPC-H Queries

- TPC-H Queries over a 200 MB database: highlight advantages of PPC
  - No overhead when querying non-sensitive attributes
- Queries 1 and 6: range queries, no sensitive attributes
  - NSM and PPC identical, NSM-page suffers from over-encryption
- Query 8: encrypted attributes from 3 largest tables (*lineitem*, *partsupp*, *orders*)
  - PPC clearly outperforms NSM-page, less data involved in encryptions
- Overall overhead incurred
  - NSM-page = 33%
  - PPC = 6%



Conclusions & Future Work

- Proposed a new DBMS storage model
  - Facilitates efficient implementation of encryption techniques
  - Groups sensitive data, reduces # of encryption operations
  - In turn, reduces encryption overhead
  - No encryption of non-sensitive attributes
- Future work
  - Extensions/variations of the PPC model
  - Less trusted server model
  - Secure hardware to aid with queries
  - Integrity/Authenticity issues

For More Information:

- H. Hacigumus, B. Iyer, C. Li and S. Mehrotra,
  Executing SQL over Encrypted Data in the Database Service Provider Model,
  SIGMOD 2002.
- H. Hacigumus, B. Iyer and S. Mehrotra,
  Providing Database as a Service
  ICDE 2002.
- H. Hacigumus, B. Iyer and S. Mehrotra,
  Efficient Execution of Aggregation Queries over Encrypted Relational Databases,
  DASFAA 2004
- B. Hore, S. Mehrotra and G. Tsudik,
  A Privacy-Preserving Index for Range Queries,
  UCI-ICS RESCUE-ITR Technical Report (in submission)
- E. Mykletun, M. Narasimha and G. Tsudik,
  Authentication and Integrity in Outsourced Databases,
  NDSS 2004