Secure Data Management

Search on Encrypted Data

Outline

- Motivation
- Some Solutions
  - SQL based Query
  - Preparation for Encrypted XML Metadata Query
  - Keyword based Search
- Challenges Ahead

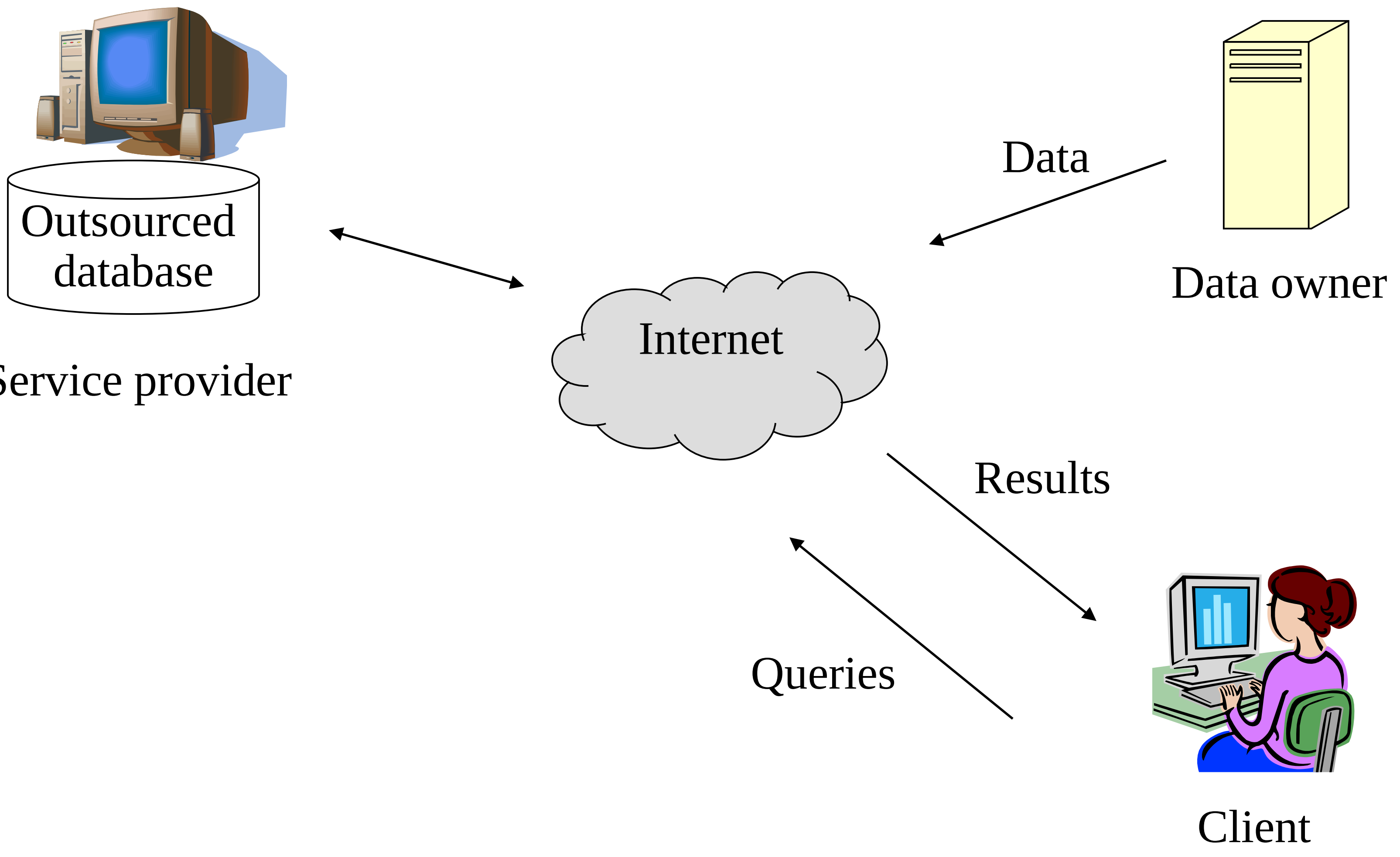
Outline

- Motivation
- Some Solutions
  - SQL based Query
  - Preparation for Encrypted XML Metadata Query
  - Keyword based Search
- Challenges Ahead

Database as a Service

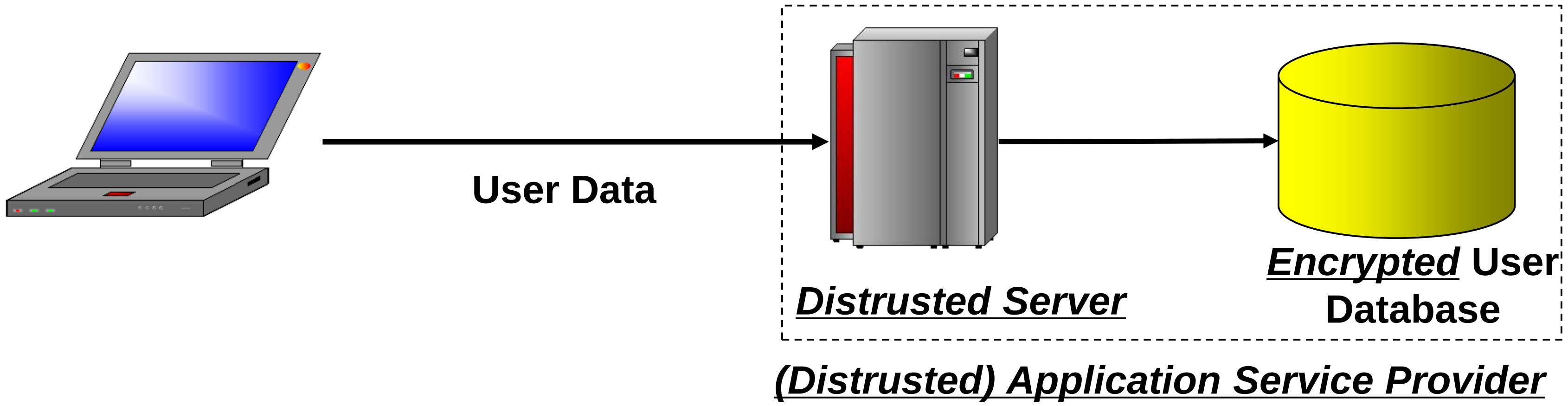
- Reduced cost to clients
  - Most organizations need efficient data management.
  - DBMSs are extremely complex to deploy, setup, and maintain.
  - Skilled DBAs are needed at very high costs.
  - Paying for what they use and not for hardware, software infrastructure or personnel to deploy, maintain, upgrade …
- Better service
- Driven by faster, cheaper, and more accessible networks

Database Service Diagram (Data Outsource)



Security Concern I

- Outsource is a trend – but cannot be trusted.
  - Remote storage and backup
  - External system administrators have root access
  - Hackers break in
- Clients require confidentiality (privacy), integrity, authenticity, ownership protection, etc..



Security Concern II

- Ambient Intelligence requires personal data to be available ‘everywhere’
- Consumers get more and more concerned about their privacy

Why Search on Encrypted Data?

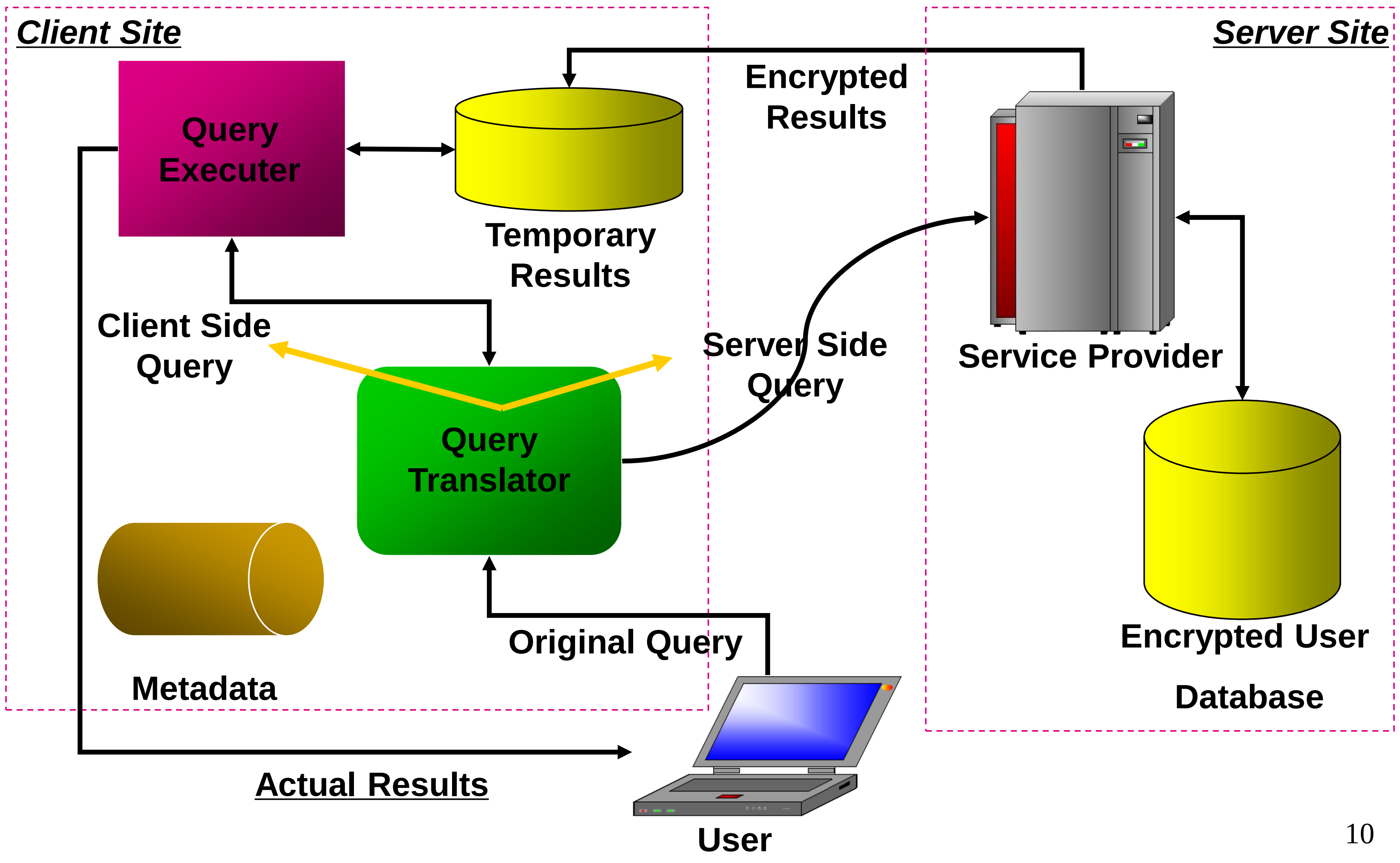
- Philosophy: ***Encryption Always-On***
- Data management and manipulation move to the encrypted domain.
- Search on encrypted databases, text files, and emails, etc.

Solution 1: SQL-Based Query

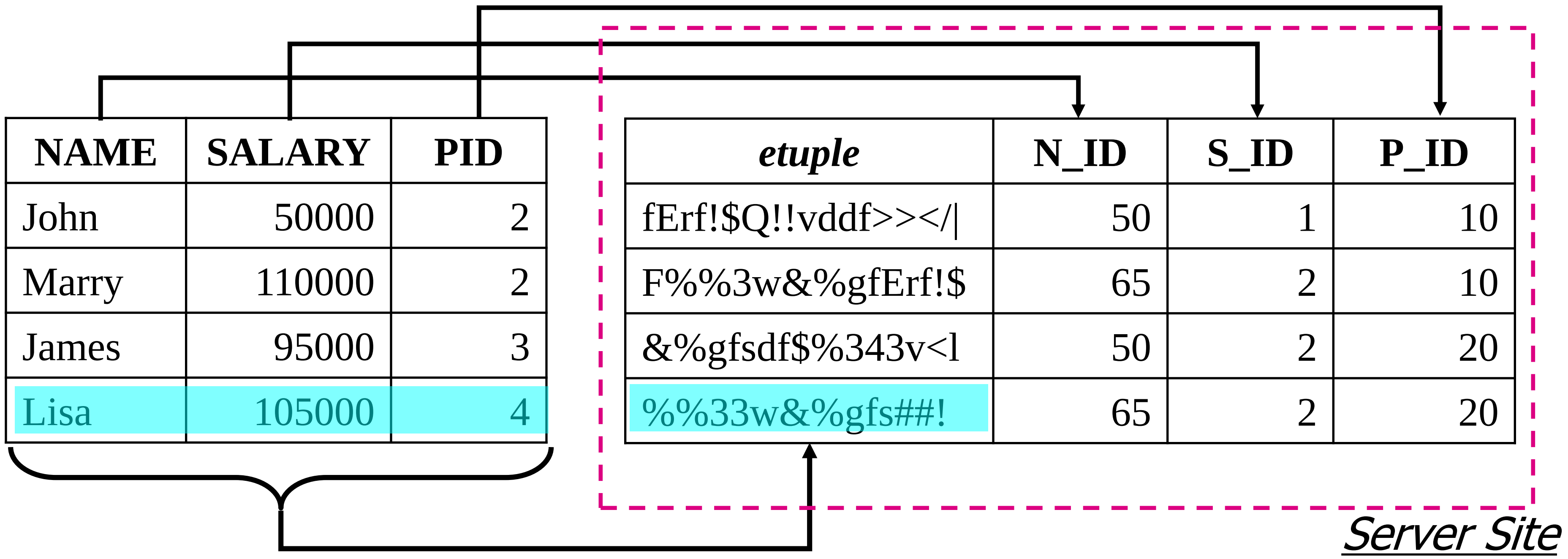
"Executing SQL over Encrypted Data in the Database Service - Provider Model"

-- *H. Hacigumus*, *B. Iyer*, *C. Li*, and *S. Mehrotra*

Query Execution Framework



Relational Encryption



NAME	SALARY	PID
John	50000	2
Marry	110000	2
James	95000	3
Lisa	105000	4

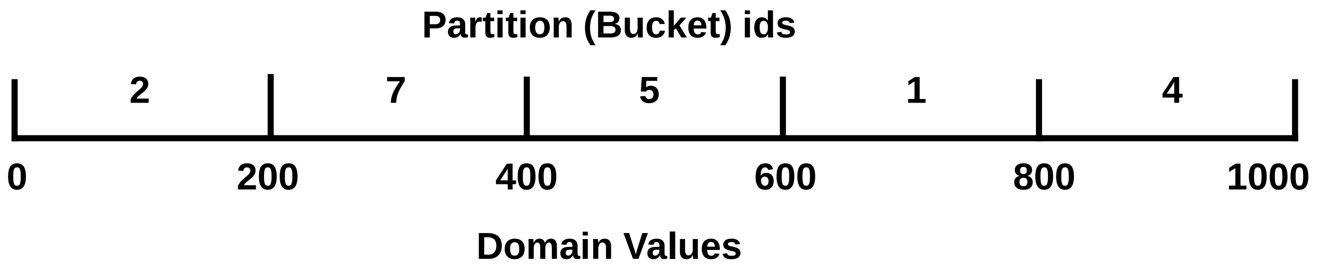
etuple	N_ID	S_ID	P_ID
fErf!$Q!!vddf>></|	50	1	10
F%%3w&%gfErf!$	65	2	10
&%gfsdf$%343v<l	50	2	20
%%33w&%gfs##!	65	2	20

- 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

Building the Index: Partition and Identification Functions

- Partition function divides domain values into partitions (buckets)

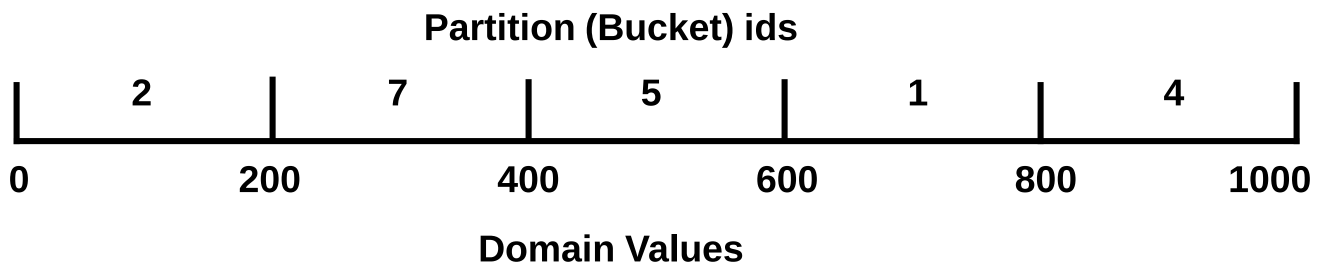
  Partition (*R.A*) = { [0,200], (200,400], (400,600], (600,800], (800,1000] }



- Partitioning function has an impact on performance as well as privacy

Building the Index: Partition and Identification Functions (*cont.*)

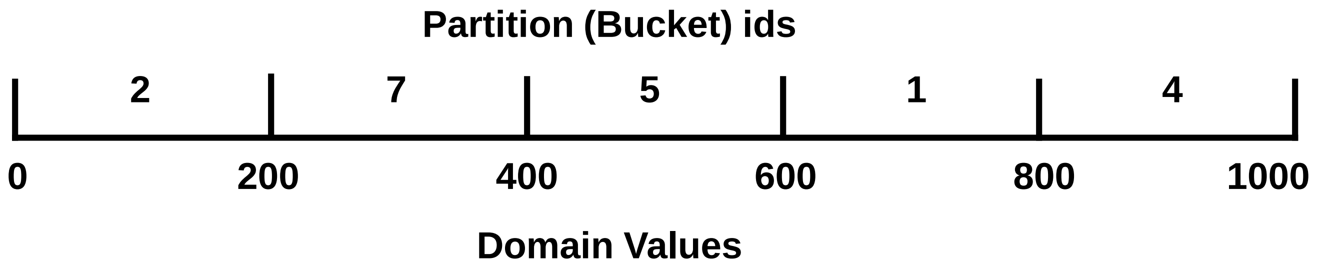
❖ Identification function assigns a partition id to each partition of attribute *A*



- e.g. $ident_{R.A}(\ (200,400]\ ) = 7$

Mapping Functions

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



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

Storing Encrypted Data

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

$etuple = \mathit{encrypt}\ (A \mid B \mid C)$

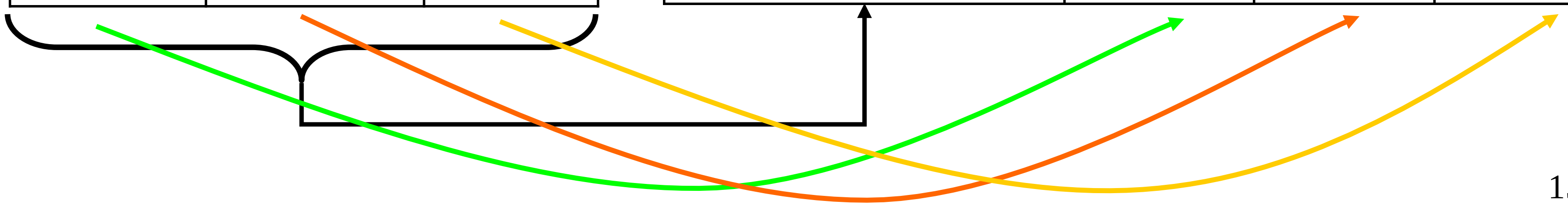
$A\_id = \mathit{Map}_{R.A}(A),\ B\_id = \mathit{Map}_{R.B}(B),\ C\_id = \mathit{Map}_{R.C}(C)$

Table: EMPLOYEE

NAME	SALARY	PID
John	50000	2
Marry	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 emp, proj
 WHERE emp.pid=proj.pid AND salary > 100k

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

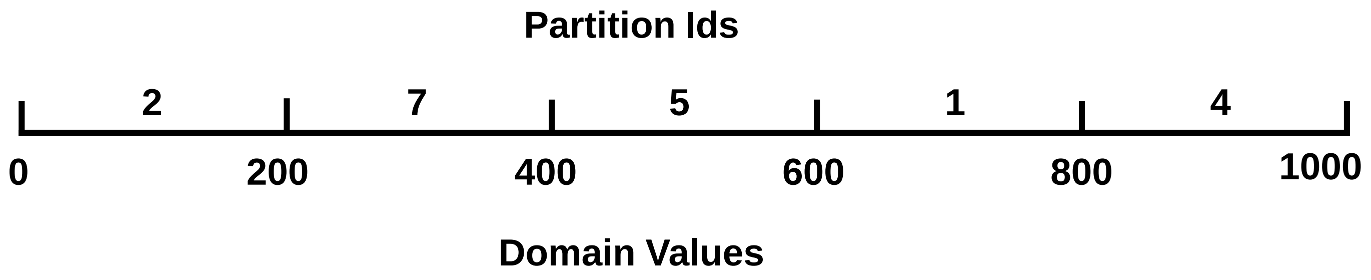
Conditions:

- Condition $\leftarrow$ Attribute *op* Value
- Condition $\leftarrow$ Attribute *op* Attribute
- Condition $\leftarrow$ (Condition $\wedge$ Condition) | (Condition $\vee$ Condition)
  | ($\neg$ Condition)

Mapping Conditions (*cont.*)

Example:

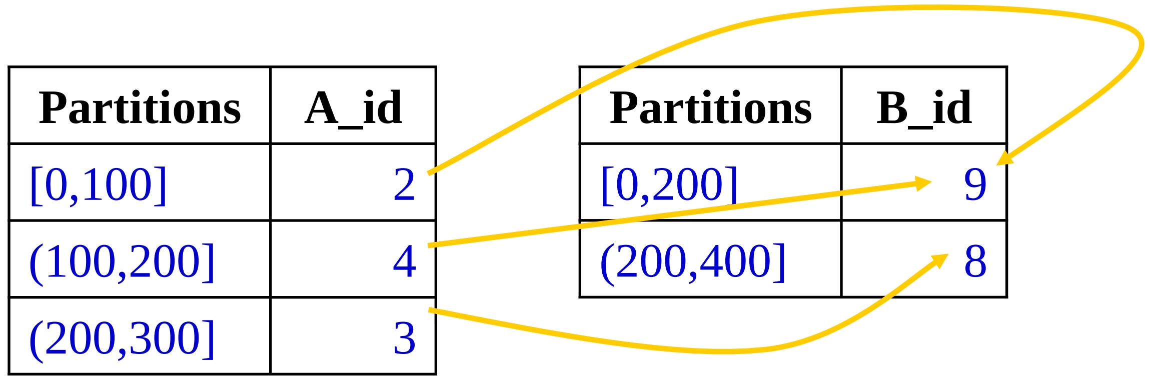
- Attribute = Value
  - $Map_{\text{cond}}(\ A = \boldsymbol{v}\ ) \Rightarrow A^{S} = Map_{A}(\ \boldsymbol{v}\ )$
  - $Map_{\text{cond}}(\ A = 250\ ) \Rightarrow A^{S} = 7$



Mapping Conditions (*cont.*)

- Attribute1 = Attribute2
  - $Map_{cond}(\ A = B\ ) \Rightarrow \vee_N\ (A^S = ident_A(\ \boldsymbol{p}_k\ ) \wedge\ B^S = ident_B(\ \boldsymbol{p}_l\ ))$

where N is $\boldsymbol{p}_k \in partition\ (A), \boldsymbol{p}_l \in partition\ (B), \boldsymbol{p}_k \cap\ \boldsymbol{p}_l \neq \varnothing$



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

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

$C : A = B \quad \Rightarrow \quad C' : \quad (A^S = 2 \wedge B^S = 9)$

$\vee\ (A^S = 4 \wedge B^S = 9)$

$\vee\ (A^S = 3 \wedge B^S = 8)$

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 without requiring storage at the client

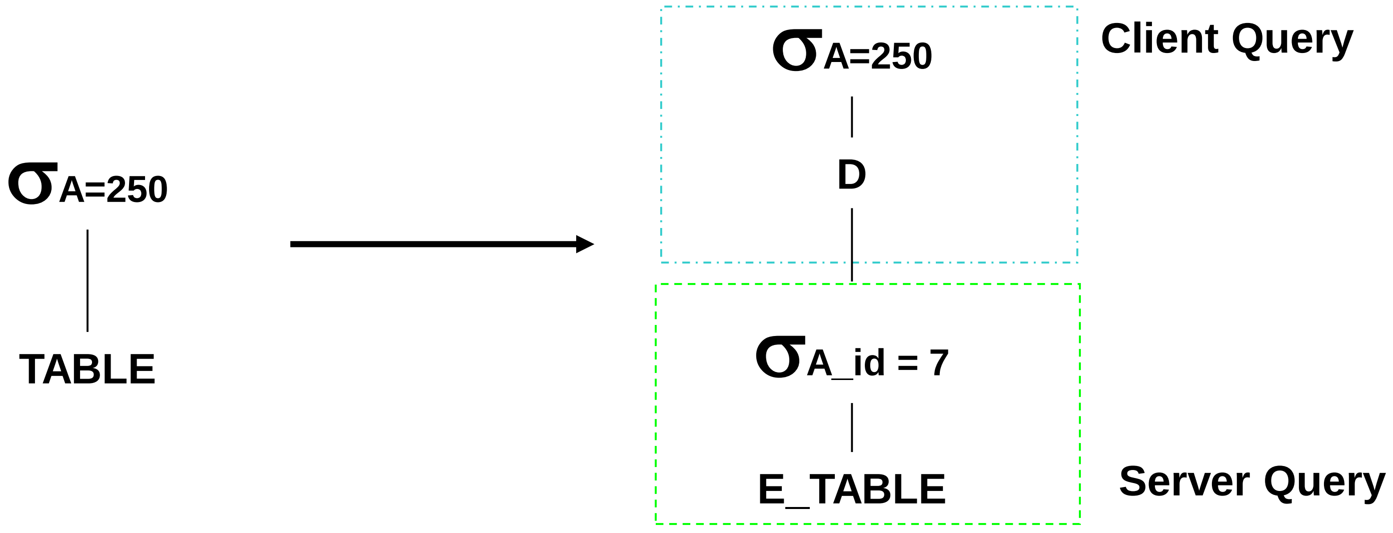
Operators studied:

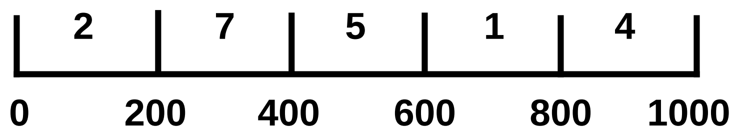
- Selection
- Join
- Others: Grouping and Aggregation, Sorting, Duplicate Elimination, Set Difference, Union, Projection

Selection Operator

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

Example:

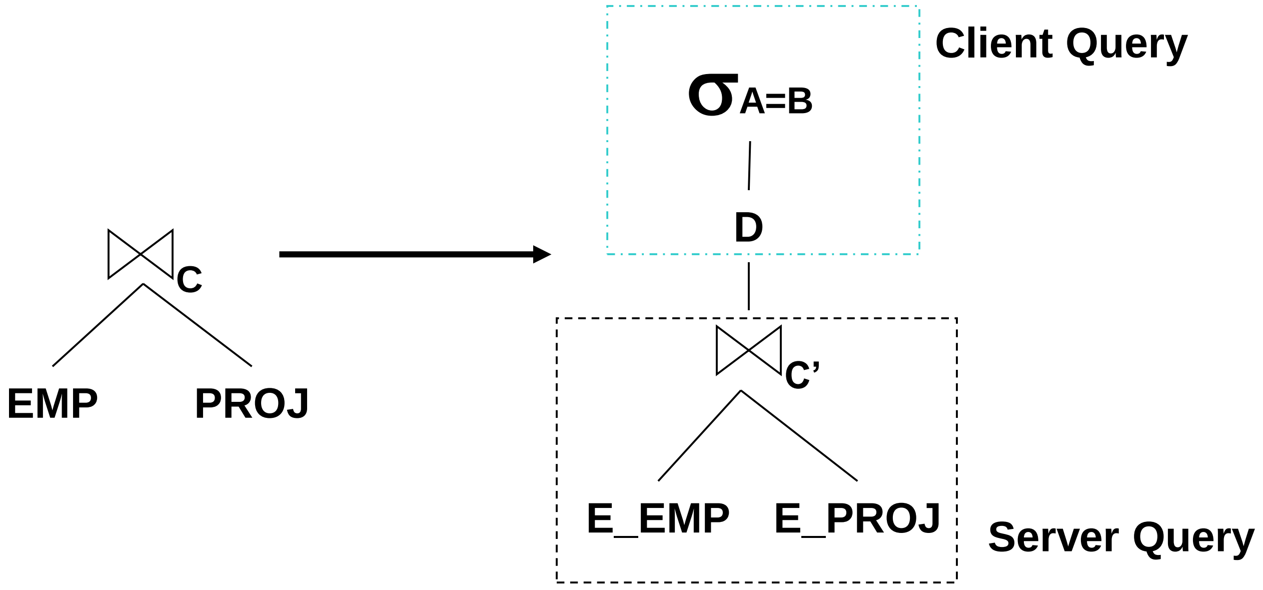




Join Operator

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

Example:



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

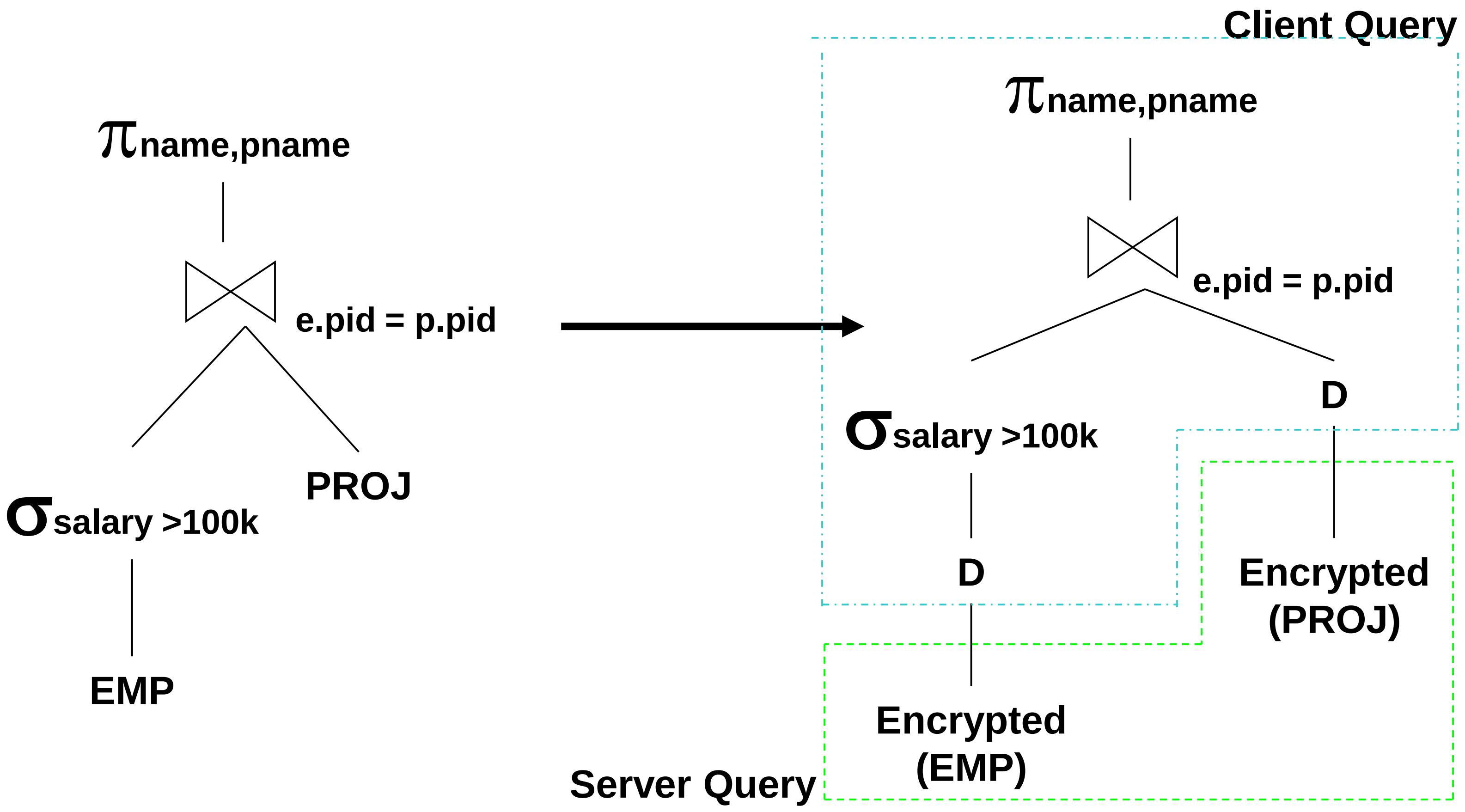
Partitions	B_id
[0,200]	9
(200,400]	8

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

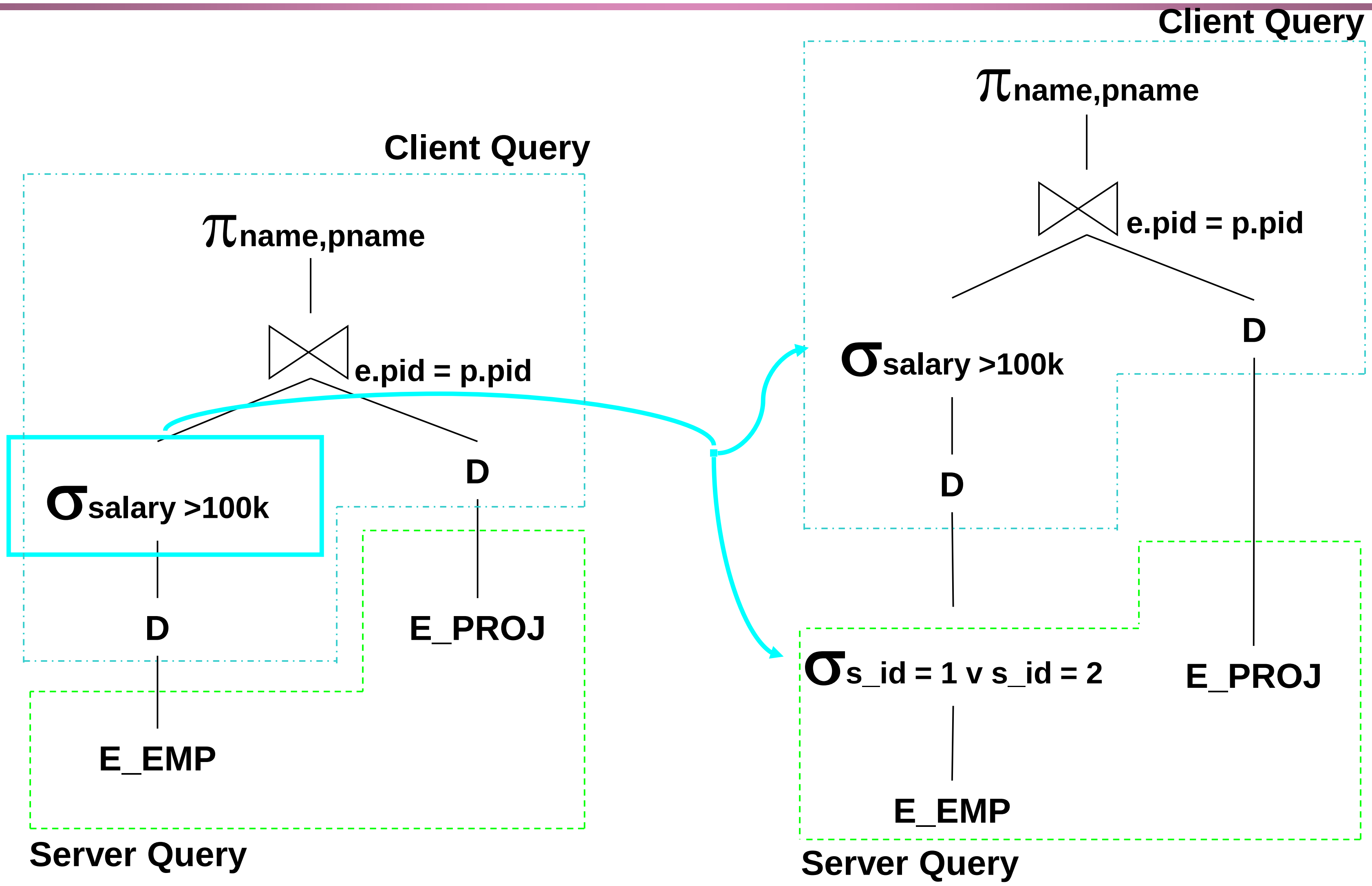
Query Decomposition

Q: SELECT name, pname FROM emp, proj

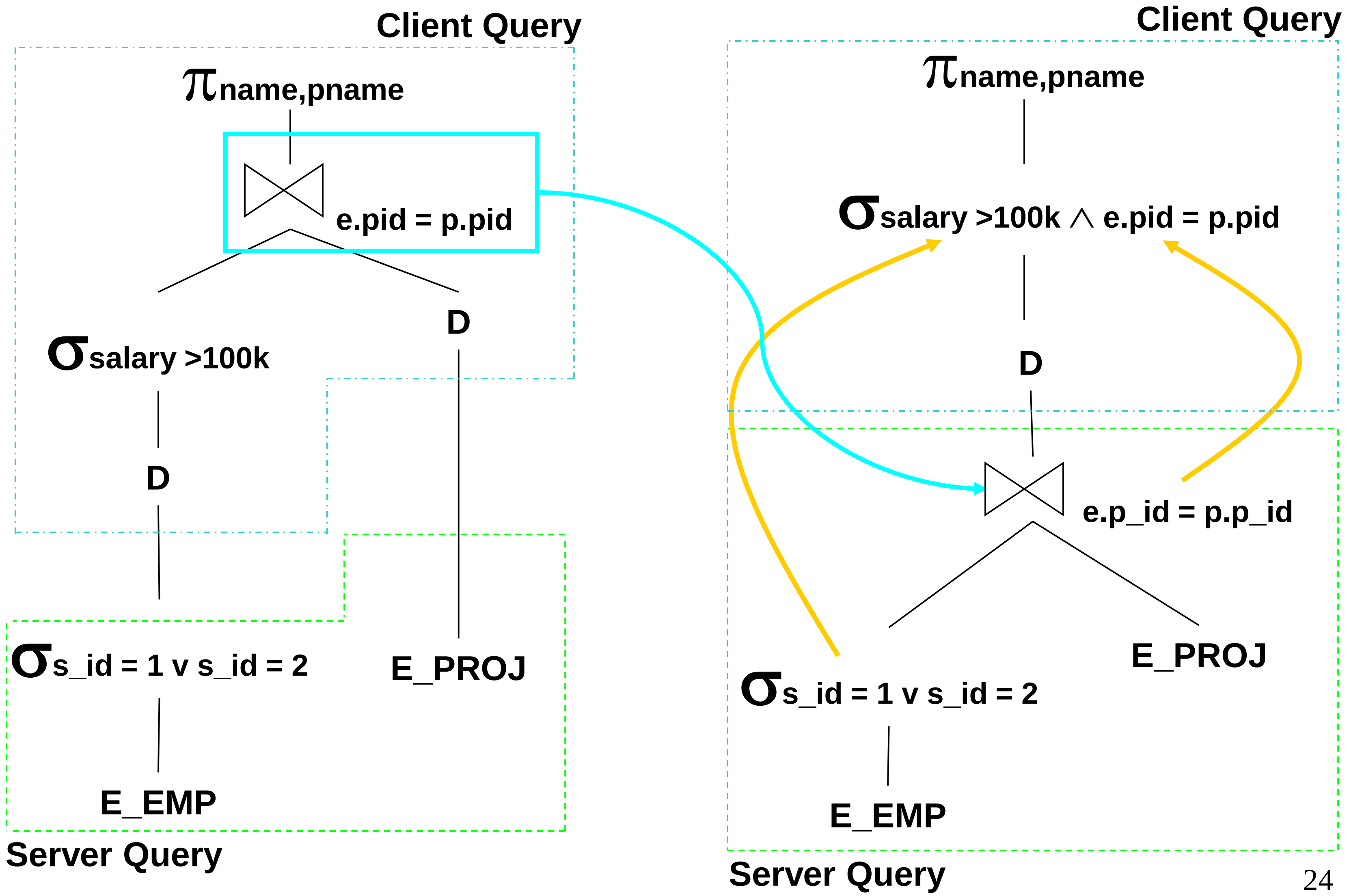
WHERE emp.pid=proj.pid AND salary > 100k



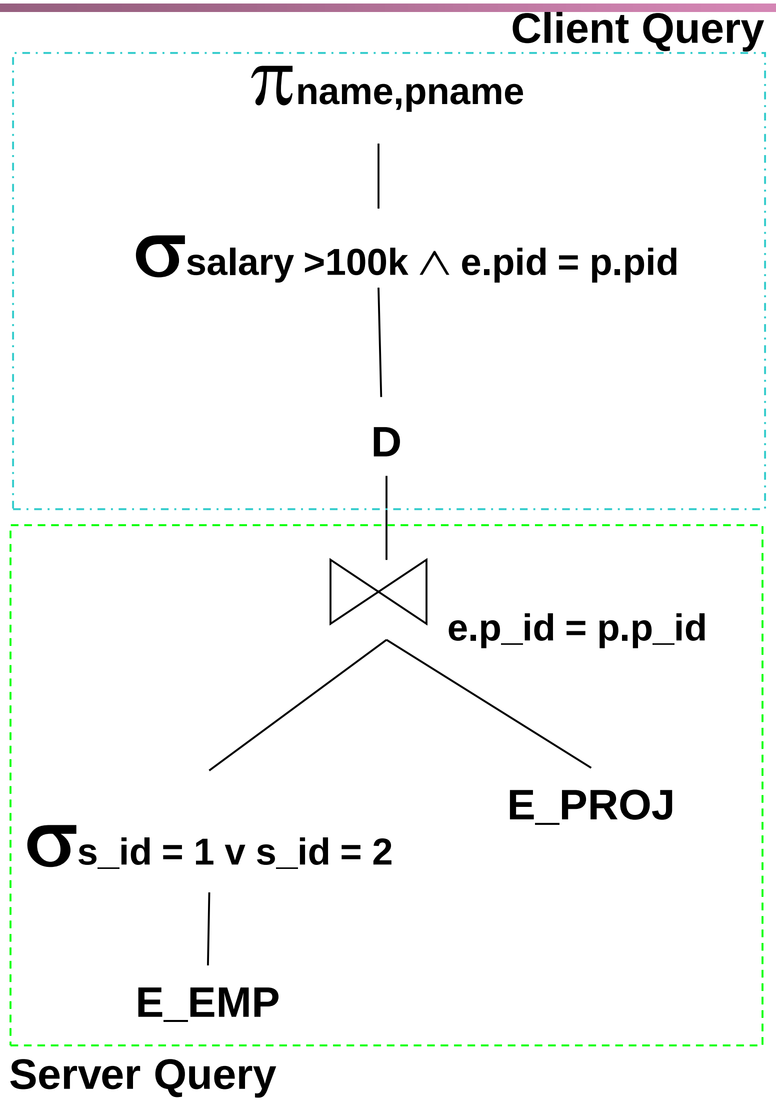
Query Decomposition (*cont.*)



Query Decomposition (*cont.*)



Query Decomposition (*cont.*)



Client Query
$\pi_{name,pname}$
$\sigma_{salary > 100k \wedge e.pid = p.pid}$
D
e.p_id = p.p_id
E_PROJ
$\sigma_{s\_id = 1 \vee s\_id = 2}$
E_EMP
Server Query

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

Q^S: 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

Q^C: SELECT name, pname
FROM temp
WHERE emp.pid=proj.pid AND
salary > 100k

Summary

- Proposed solution
  - encrypts data, creates “coarse indexes” and stores the data at server
  - allows only data owner to decrypt the data

- With query decomposition
  - most of query execution performed at server side
  - client only performs filtering

Search on Encrypted Data

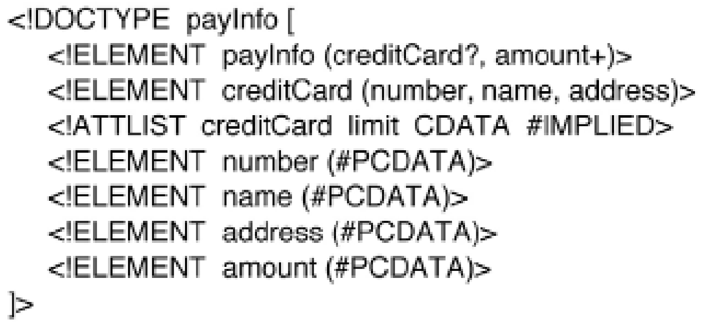
- Motivation
- Some Solutions
  - Executing SQL over Encrypted Data in the Database-Service-Provider Model (H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra**)**
  - Preparations for Encrypted XML Metadata Querying (Ling Feng and Willem Jonker)
  - Practical Techniques for Searches on Encrypted Data (D. Song, D. Wagner, and A. Perrig)
- Challenges Ahead

Solution 2: Preparation for Encrypted XML Metadata Query

- Problem Statement
  - Given a collection of encrypted XML documents and an X-path query, find relevant documents while guaranteeing efficient processing

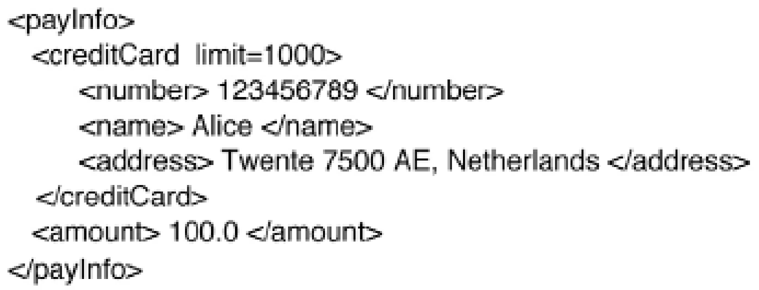
The Proposal

- Discard non-candidate XML data and decrypt the remaining data set
- How to distinguish candidate XML data from non-candidate data? ***Exploit DTDs***



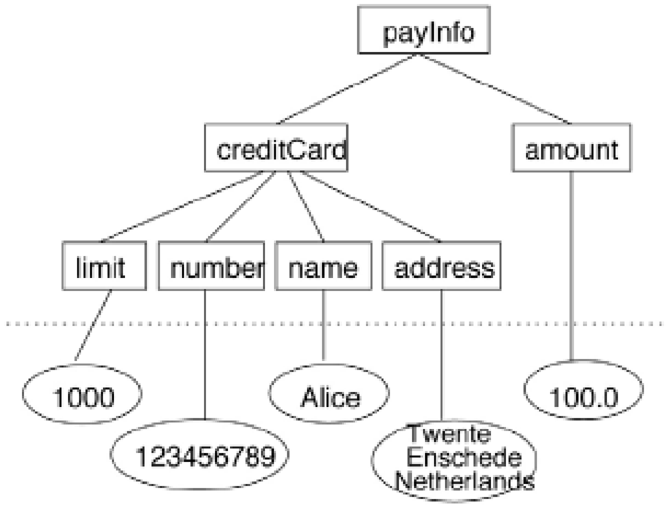
```
<!DOCTYPE  payInfo [
    <!ELEMENT  payInfo (creditCard?, amount+)>
    <!ELEMENT  creditCard (number, name, address)>
    <!ATTLIST  creditCard  limit  CDATA  #IMPLIED>
    <!ELEMENT  number (#PCDATA)>
    <!ELEMENT  name (#PCDATA)>
    <!ELEMENT  address (#PCDATA)>
    <!ELEMENT  amount (#PCDATA)>
]>
```

(a) An XML DTD exmple – DTD1



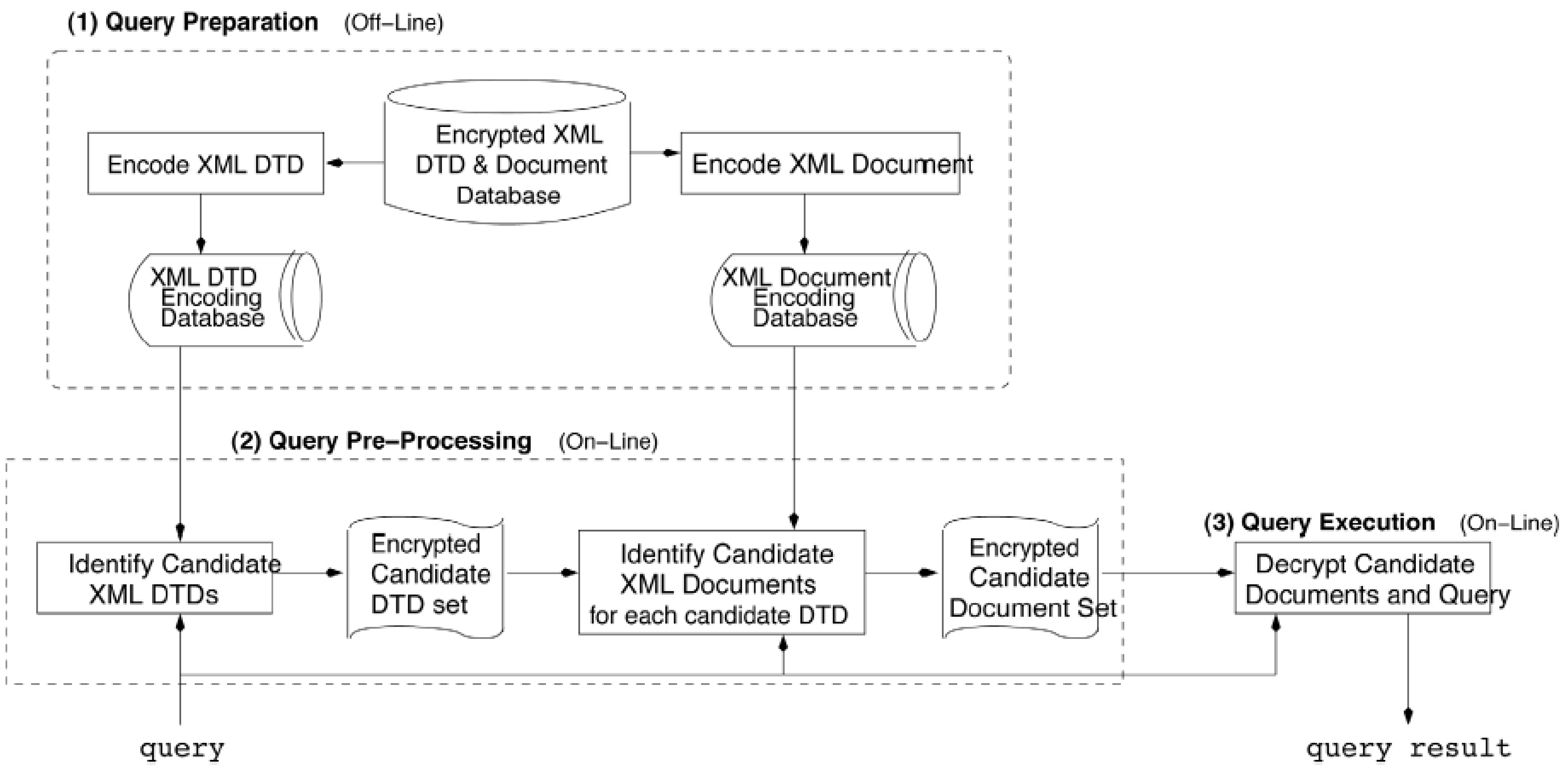
```
<payInfo>
  <creditCard  limit=1000>
      <number> 123456789 </number>
      <name> Alice </name>
      <address> Twente 7500 AE, Netherlands </address>
  </creditCard>
  <amount> 100.0 </amount>
</payInfo>
```

(b) An XML document exmple that conforms to DTD1



(c) A graphical representation of the DOM tree structure of DTD1 with the example document

The Framework


(1) Query Preparation (Off–Line)
Encode XML DTD
Encrypted XML DTD & Document Database
Encode XML Document
XML DTD Encoding Database
XML Document Encoding Database
(2) Query Pre–Processing (On–Line)
Identify Candidate XML DTDs
Encrypted Candidate DTD set
Identify Candidate XML Documents for each candidate DTD
Encrypted Candidate Document Set
(3) Query Execution (On–Line)
Decrypt Candidate Documents and Query
query
query result

Phase-1: Query Preparation (off-line)

- Effective encoding schemas
  - It can facilitate fast identification of candidate DTDs and documents
- Safe encodings
  - Encodings do not leak any information to externals
- Condensed encoding size
  - The space used to store encodings of DTDs and documents is small.

Phase-2: Query Pre-Processing (on-line)

- Completeness
  - The pre-selected candidate DTDs and documents must be supersets of the real query target DTDs and documents.
- High selectivity
  - Candidate DTDs and documents should have a high possibility of being real query targets.
- Efficiency
  - The candidate pre-selection process must be fast enough
- Hidden query support
  - Besides encrypted XML data, query itself could be encrypted.

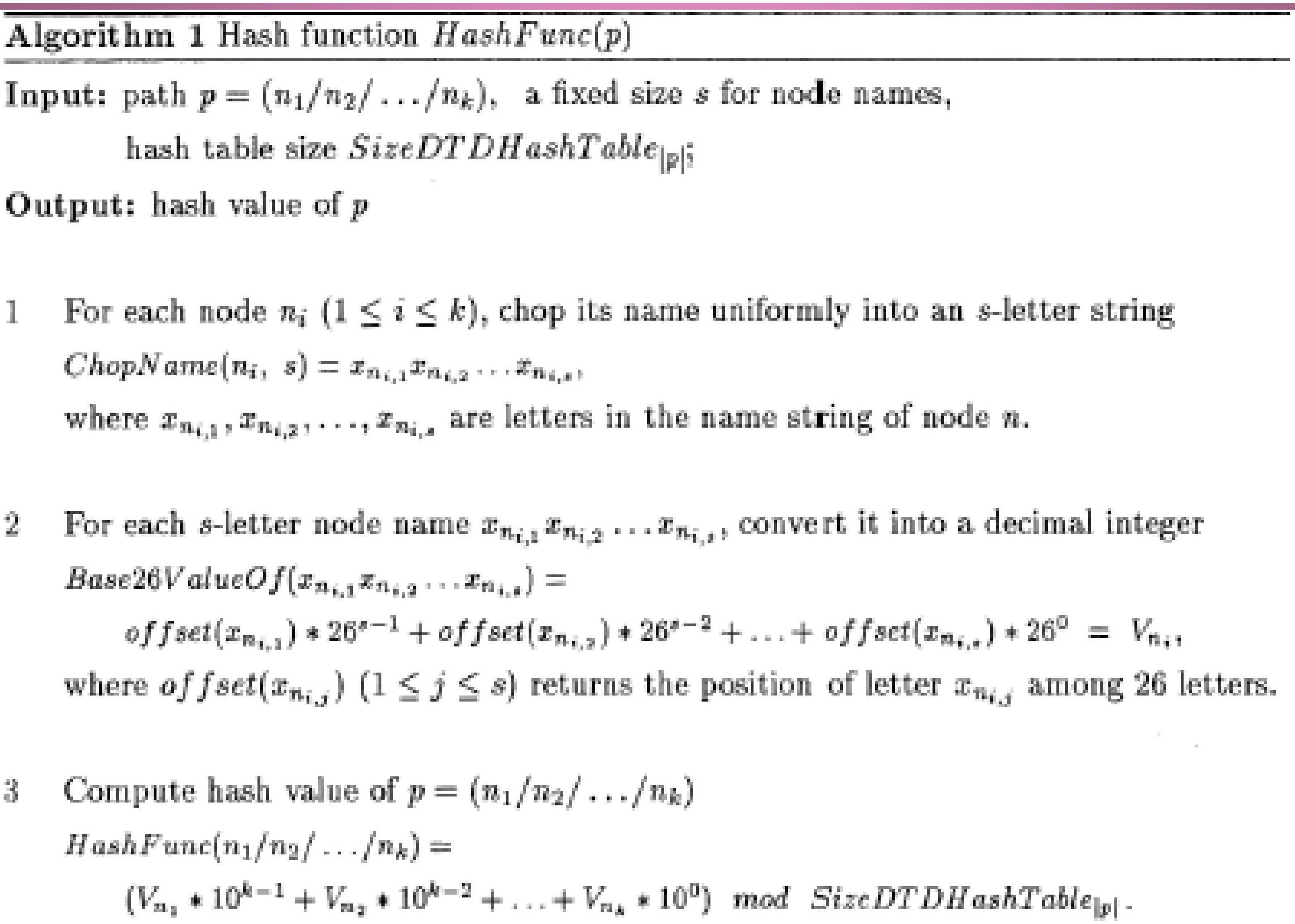
Encoding DTDs based on Paths

- A path p is hashed to a value $H(p)$
- There are different hash tables for each path length $T_1 \ldots T_N$, where N is the max. path length of all paths in the collection of DTDs.
- A reference to DTD_i is put into hash table $T_k[H(p)]$ where $k = \text{length}(p)$

Table 1 Paths extracted from the example XML DTD

Path length	Path
2	p_1 = (payInfo/creditCard/limit)
	p_2 = (payInfo/creditCard/number)
	p_3 = (payInfo/creditCard/name)
	p_4 = (payInfo/creditCard/address)
1	p_5 = (payInfo/creditCard)
	p_6 = (payInfo/amount)
	p_7 = (creditCard/limit)
	p_8 = (creditCard/number)
	p_9 = (creditCard/name)
	p_{10} = (creditCard/address)
0	p_{11} = (payInfo)
	p_{12} = (creditCard)
	p_{13} = (amount)
	p_{14} = (limit)
	p_{15} = (number)
	p_{16} = (name)
	p_{17} = (address)

A Possible hash function



Algorithm 1 Hash function $HashFunc(p)$

Input: path $p = (n_1/n_2/\ldots/n_k)$, a fixed size s for node names,
hash table size $SizeDTDHashTable_{|p|}$;

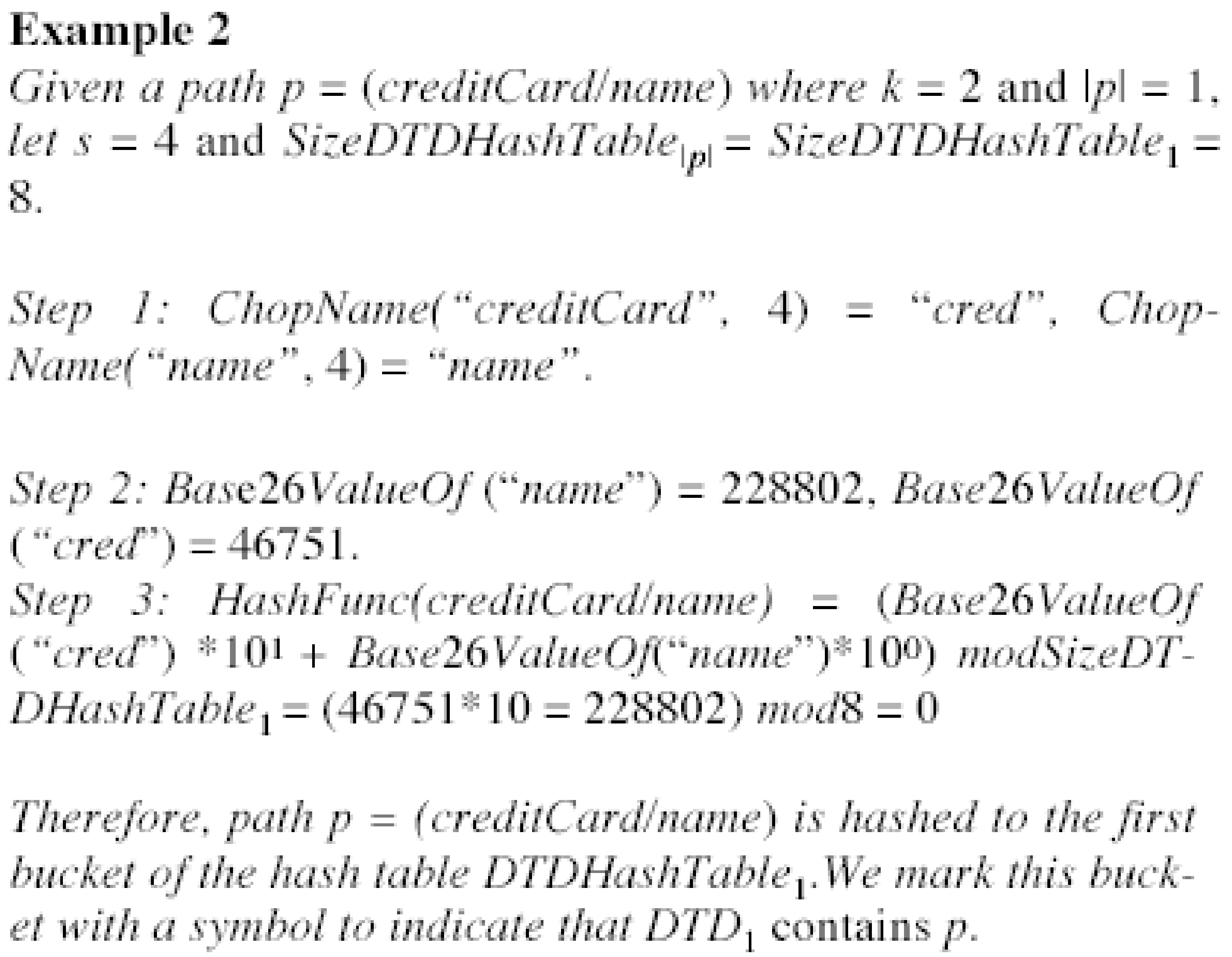
Output: hash value of p

```
1   For each node $n_i$ $(1 \leq i \leq k)$, chop its name uniformly into an $s$-letter string
    $ChopName(n_i,\ s) = x_{n_{i,1}} x_{n_{i,2}} \ldots x_{n_{i,s}}$,
    where $x_{n_{i,1}}, x_{n_{i,2}}, \ldots, x_{n_{i,s}}$ are letters in the name string of node $n$.

2   For each $s$-letter node name $x_{n_{i,1}} x_{n_{i,2}} \ldots x_{n_{i,s}}$, convert it into a decimal integer
    $Base26ValueOf(x_{n_{i,1}} x_{n_{i,2}} \ldots x_{n_{i,s}}) =$
        $offset(x_{n_{i,1}}) * 26^{s-1} + offset(x_{n_{i,2}}) * 26^{s-2} + \ldots + offset(x_{n_{i,s}}) * 26^0 = V_{n_i}$,
    where $offset(x_{n_{i,j}})$ $(1 \leq j \leq s)$ returns the position of letter $x_{n_{i,j}}$ among 26 letters.

3   Compute hash value of $p = (n_1/n_2/\ldots/n_k)$
    $HashFunc(n_1/n_2/\ldots/n_k) =$
        $(V_{n_1} * 10^{k-1} + V_{n_2} * 10^{k-2} + \ldots + V_{n_k} * 10^0)$ $mod$ $SizeDTDHashTable_{|p|}$.
```

Example path hashing



Example 2

Given a path p = (*creditCard/name*) *where* $k = 2$ and $|p| = 1$, *let* $s = 4$ and $SizeDTDHashTable_{|p|} = SizeDTDHashTable_1 = 8$.

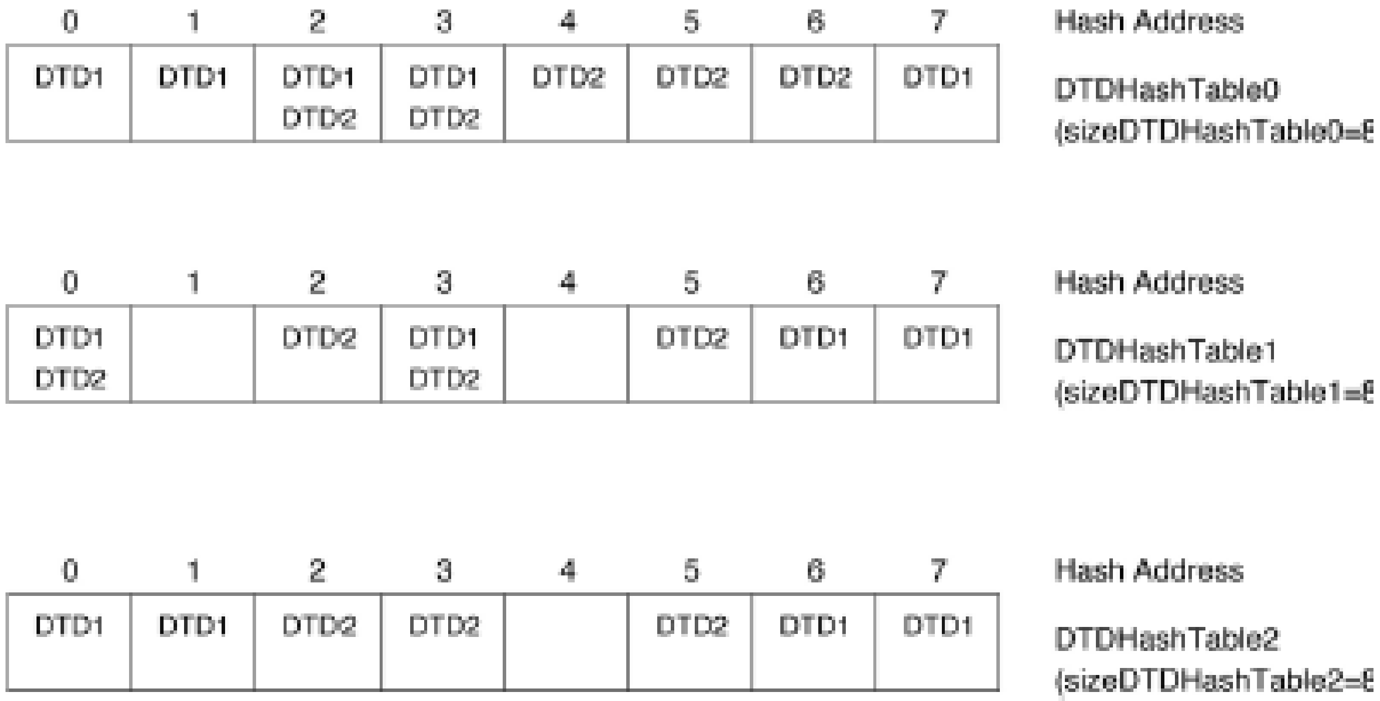
Step 1: ChopName(*"creditCard"*, 4) = *"cred"*, *ChopName*(*"name"*, 4) = *"name"*.

Step 2: Base26ValueOf (*"name"*) = 228802, *Base26ValueOf* (*"cred"*) = 46751.

Step 3: HashFunc(*creditCard/name*) = (*Base26ValueOf* (*"cred"*) $*10^1$ + *Base26ValueOf*(*"name"*)$*10^0$) *modSizeDTDHashTable*$_1$ = $(46751*10 = 228802)$ *mod*$8 = 0$

Therefore, path p = (*creditCard/name*) *is hashed to the first bucket of the hash table DTDHashTable*$_1$.*We mark this bucket with a symbol to indicate that DTD*$_1$ contains p.

Encoding of XML DTDs



0	1	2	3	4	5	6	7	Hash Address
DTD1	DTD1	DTD1 DTD2	DTD1 DTD2	DTD2	DTD2	DTD2	DTD1	DTDHashTable0 (sizeDTDHashTable0=8

0	1	2	3	4	5	6	7	Hash Address
DTD1 DTD2		DTD2	DTD1 DTD2		DTD2	DTD1	DTD1	DTDHashTable1 (sizeDTDHashTable1=8

0	1	2	3	4	5	6	7	Hash Address
DTD1	DTD1	DTD2	DTD2		DTD2	DTD1	DTD1	DTDHashTable2 (sizeDTDHashTable2=8

Figure 4 Encodings of the example DTD_1 and DTD_2 (DTD$HashTable_0$, $DTDHashTable_1$ and $DTDHashTable_2$)

Selecting the DTD for a Query

- Let query q be a path expression

- Then all matching DTDs are in $T_{\mathrm{length}(q)}[H(q)]$

Solution 3: Keyword-based Search

"Practical Techniques for Searches on Encrypted Data"

-- *D. Song, D. Wagner, and A. Perrig*

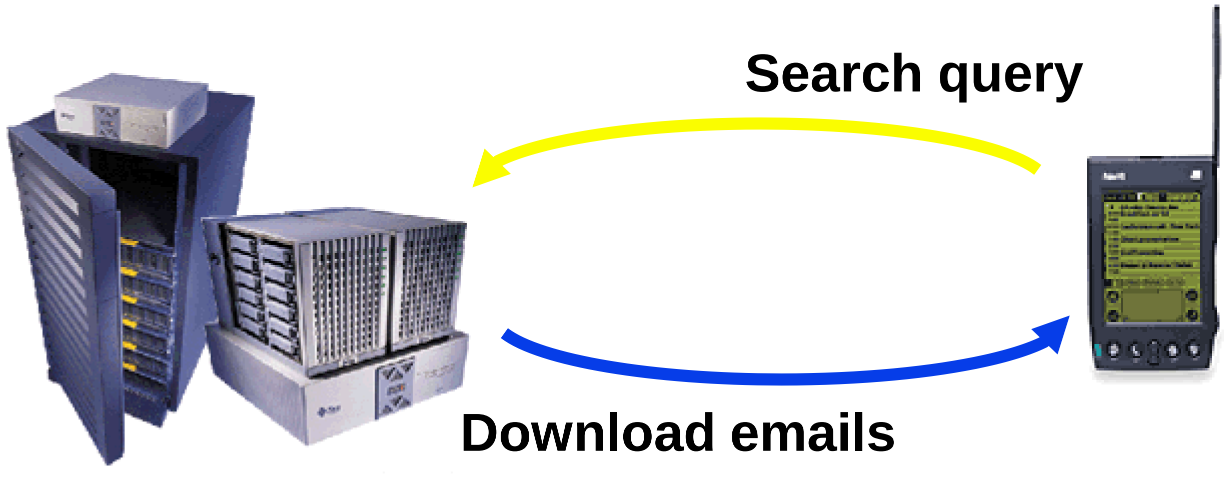
Gmail Example

- Gmail scans e-mail to insert suitable advertisements.
- Many people have raised privacy concerns about this scanning.

- Idea: Encrypt all e-mails and search in the encrypted e-mails using keywords.

A Naïve Solution

- Download all documents, decrypt, and then search on local machine



- Problem: exchange efficiency for security

Design Goals

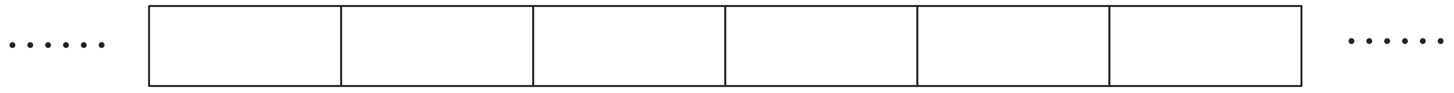
- Provable security
  - Provable secrecy
    - *encryption scheme is provable secure*
  - Controlled search
    - *server cannot search for an arbitrary word without the user's authorization*
  - Query isolation
    - *search for one word cannot learn anything more about the plaintext than the search result*
  - Hidden query
    - *does not reveal the search words*

Design Goals (*cont.*)

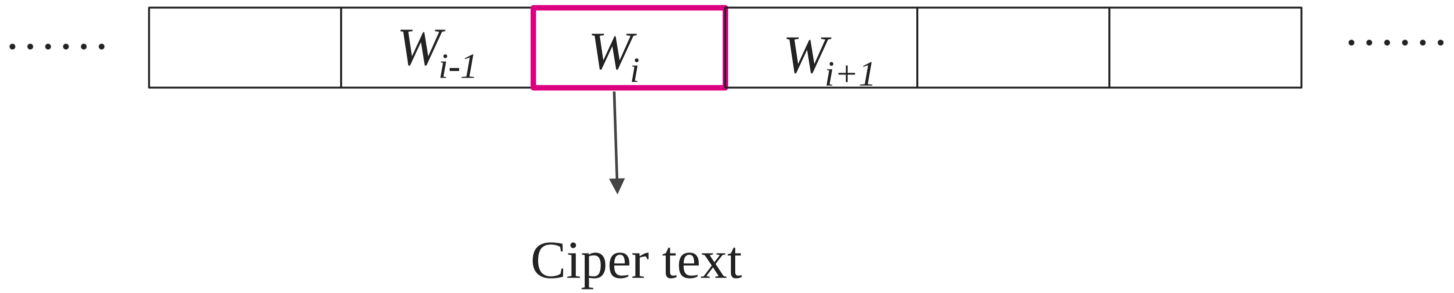
- Efficiency
  - Low computation overhead
  - Low space and communication overhead
  - Low management overhead

Schema I: Basic Scheme Encryption

- Treat text as a series of keyword bloacks



- To encrypt,



Schema I: Basic Scheme Encryption

We want to encrypt words $W_1, W_2, \ldots, W_l$

$W_1, W_2, \ldots, W_l$ n bits each

$S_1, S_2, \ldots, S_l$ n - m bits each

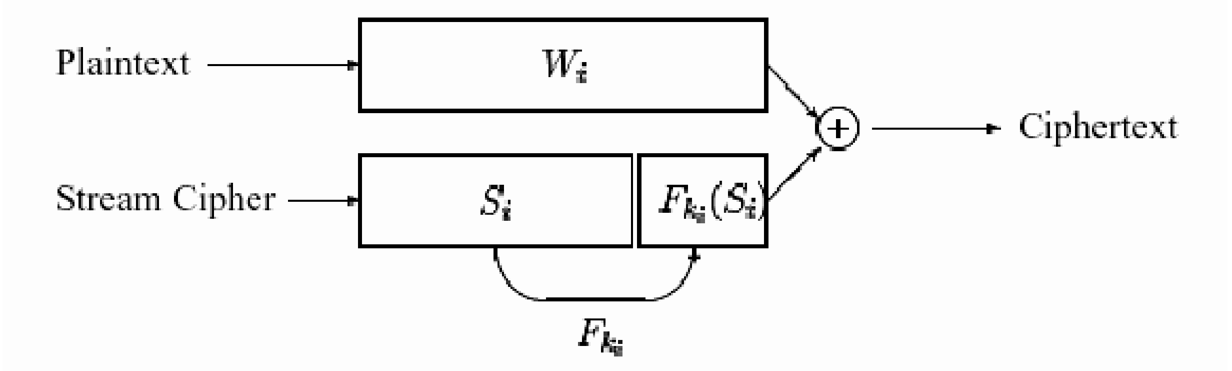
S_i are pseudo random values generated using stream cipher

$T_i = < S_i, F_{k_i}(S_i) >$

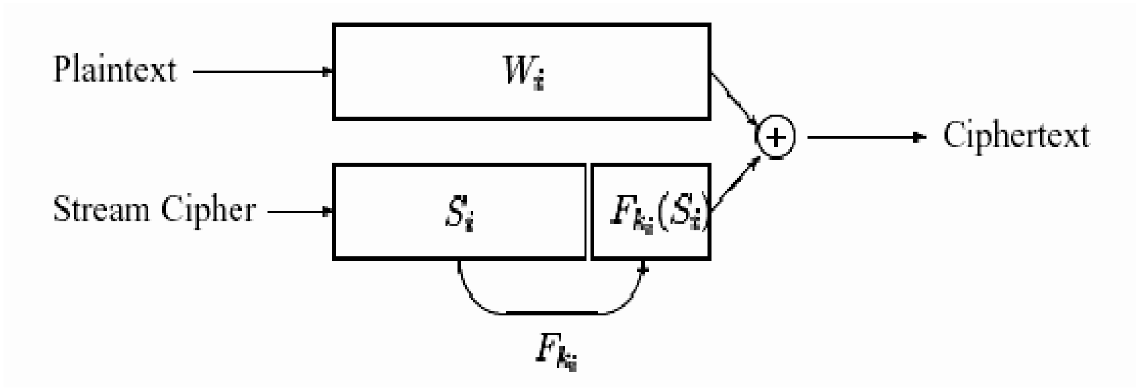
F is the pseudo random function with the range of m bits

k_i is some secret key stored on a trusted server

$C_i = W_i \oplus T_i$



Basic Scheme I - Search and Decryption



- To Search:

  Send $< W, k_i >$ to the unstrusted server

  For each entry the server computes $C_i \oplus W = T_i$

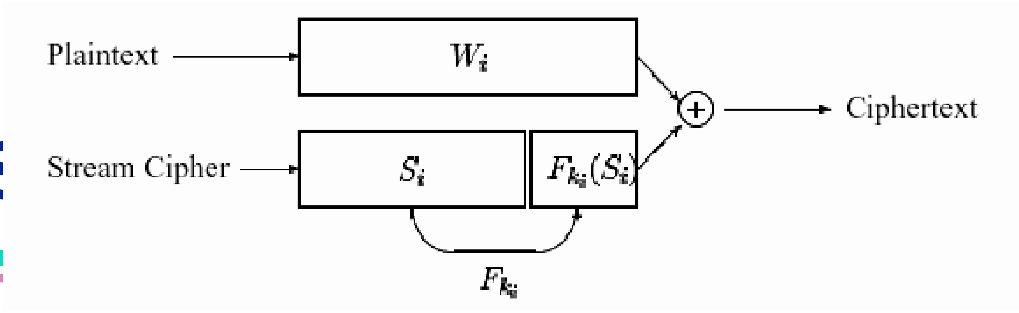
  and checks whether $F_{k_i}(T_i^{1,n-m}) = T_i^{n-m+1,n}$

  If they are equal, then the match occurs and the document is sent to the requester. Number of false positives are possible, but can be reduced by increasing m.

- To Decrypt:

  Determine S_i, compute $F_{k_i}(S_i)$, and $W_i = C_i \oplus < S_i, F_{k_i}(S_i) >$

Basic Scheme Iss



Bad

1. The problem with the basic scheme lies in k_i, giving the untrusted server an opportunit y to search for any keyword, violating the controlled search criteria.

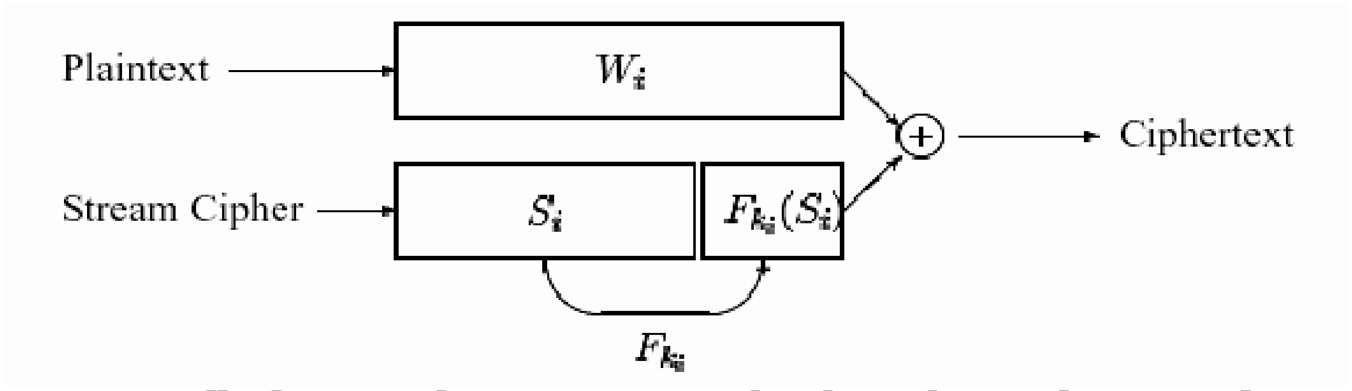
2. The untrusted server knows the search query.

Good

3. The time to perform the search is linear in the number of keywords, so it requires $O(n)$ stream cipher and block cipher operations .

4. At the positions where the untrusted server does not know k_i, it learns nothing about the keyword.

Schema II: Controlled Search



To perform controlled searching, we tie the key k_i to the word W_i.

To do that, we introduce a new pseudorandom function $f : K_F \times \{0,1\}^* \to K_F$ keyed with a secret key chosen uniformly at random. Now, $k_i = f_{k'}(W_i)$.

To search : the untrusted server is given W and $f_{k'}(W)$

(where k' is secret, never revealed).

For each entry the server computes $C_i \oplus W = T_i$ and checks whether $F_{f_{k'}(W)}(T_i^{1,n-m}) = T_i^{n-m+1,n}$. If they are equal, then the match occurs and the document is sent to the requester.

Schema II Issues

- To decrypt

  Determine $\mathrm{S_i}$, compute $F_{fk'}(S_i)$, and $W_i = C_i \oplus < S_i, F_{fk'}(S_i) >$

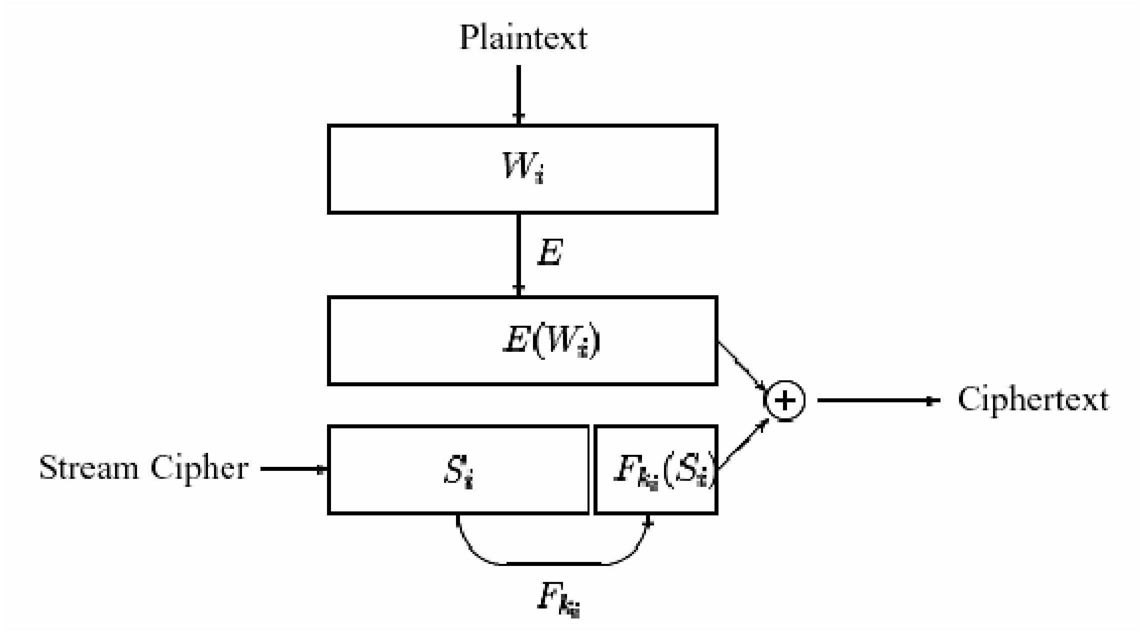
- The issue of hiding search queries is still unresolved.

Schema III: Hidden Search

To allow for hidden searches, we encrypt the word W.

$C_i = E_{k''}(W_i) \oplus T_i$, where $T_i = < S_i, F_{k_i}(S_i) >$

To search: Send $E_{k''}(W_i)$ and $f_{k'}(E_{k''}(W_i))$ to the untrusted server.



Decryption Problem

If Alice generates keys $k_i = f_{k'}(E_{k''}(W_i))$, then Alice cannot recover the plaintext from just the ciphertext, because she would need to know $E_{k''}(W_i)$ before she can decrypt.

This defeats the purpose of an encryption schema, because even legitimate principals with access to the decryption keys will be unable to decrypt.

Schema IV: Solving Decryption Problem

To solve the decryption problem, $X_i = E_{k''}(\mathrm{W_i})$ is broken into two parts.

The first part L_i has $n-m$ bits and the second R_i has m bits.

Then, compute the key k_i only as the function of the first part.

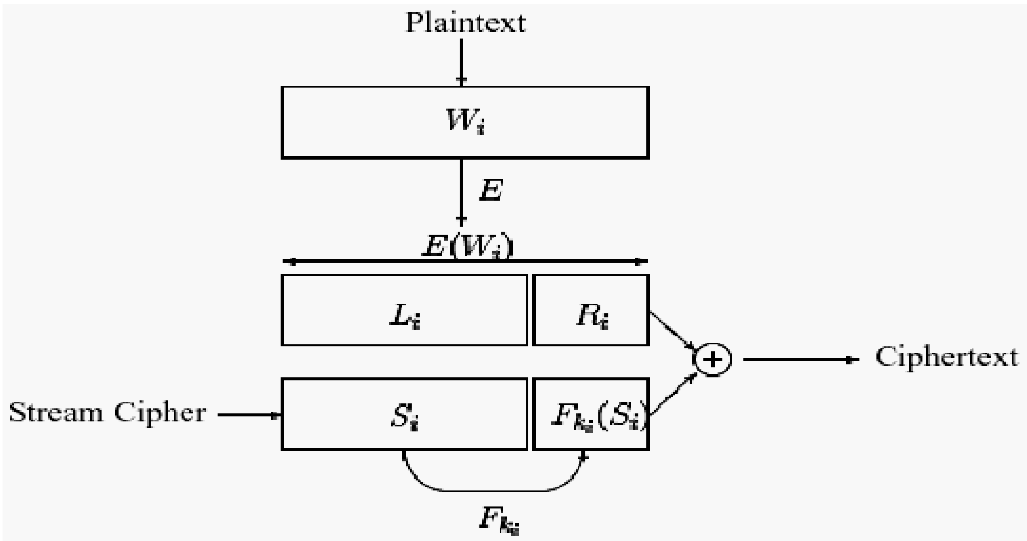
Making the above changes does not reduce the security of the scheme, but allows for an easy decryption because we can find S_i, XOR it with th e ciphertext to retrieve L_i and compute $k_i = f_{k'}(L_i)$.

To search, send :

$$E_{k''}(W_i) = < L_i, R_i >, \underline{k_i = f_{k'}(L_i)}.$$

To decrypt : Determine S_i, compute

$$L_i = C_i^{1,n-m} \oplus S_i \text{ and } k_i = f_{k'}(L_i).$$



Advanced Search Queries

- Building blocks for advanced search queries, like
  - Boolean operations (W and W’)
  - Proximity queries (W near W’)
  - Phrase searches (W immediately precedes W’)

Dealing With Variable Length Words

- Pick a long enough fixed-size block
  - A fixed padding is required
  - Inefficient in space
- Support variable length word with word length
  - Instead of W, use $< l_W, W>$
  - Move pointer bit by bit
  - Longer scan time, but efficient space

Index-based Search

- For large database applications
- Index contains a list of keywords
  - each keyword points to documents containing it
- Methods
  - Encrypt keyword and leave pointers unencrypted
  - Encrypt pointers also
    - Alice queries encrypted keyword, and Bob returns encrypted pointers
    - Alice needs to spend extra round
- Update cost is expensive

Summary

- “Efficient” encryption, decryption, search that take O(n) number of block cipher and stream cipher operations
- Provable security with controlled searching, hidden queries, and query isolation
- Possible support for composed queries
- Possible support for varied-length words
  - Padding with fixed length blocks
  - Variable length words (store the length)

Search on Encrypted Data

- Motivation
- Some Solutions
  - Practical Techniques for Searches on Encrypted Data (D. Song, D. Wagner, and A. Perrig)
  - Executing SQL over Encrypted Data in the Database-Service-Provider Model (H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra)
  - Preparations for Encrypted XML Metadata Querying (Ling Feng and Willem Jonker)
- Challenges Ahead

Challenges Ahead

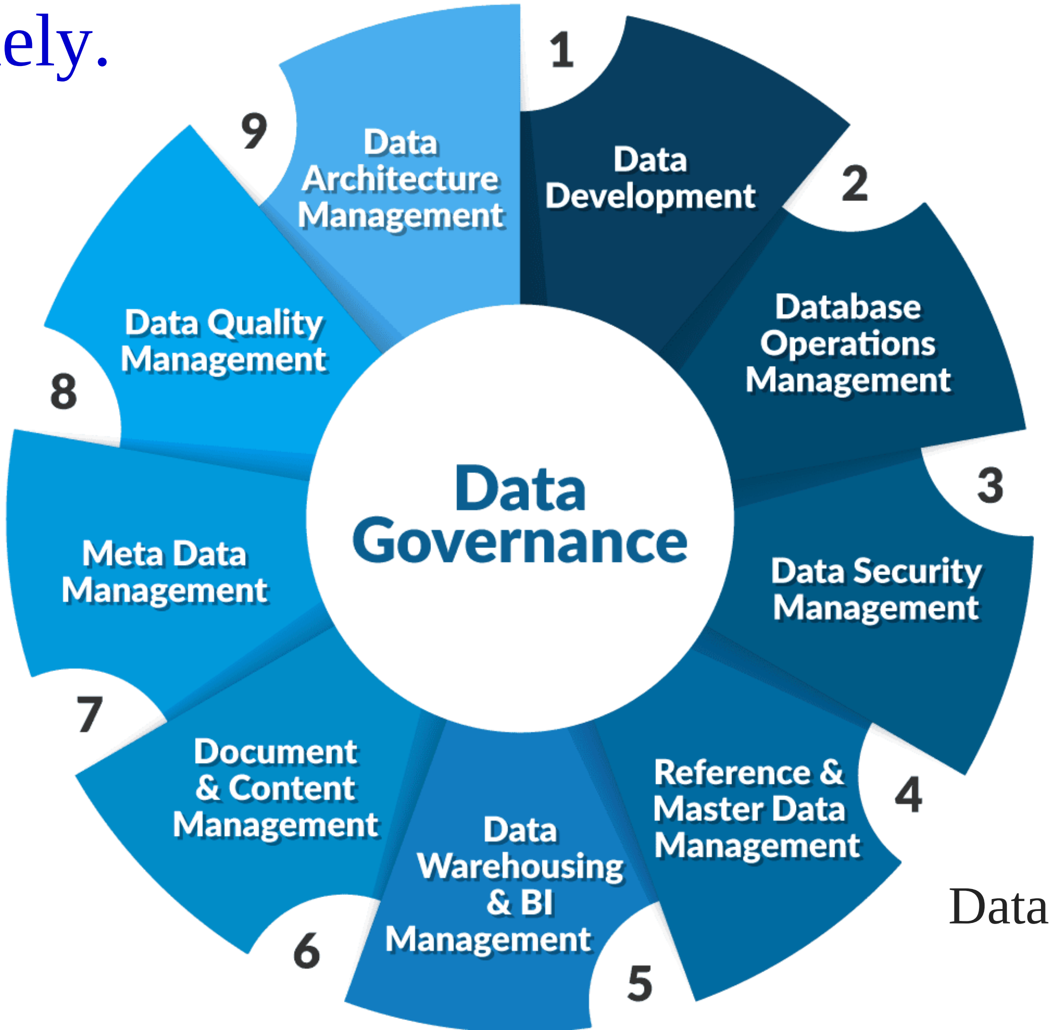
- Data security concerns are evolving
- Data becomes more and more valuable
- We have massive amounts of data
- The democratization of ubiquitous computing has led to requirements to access data anywhere, anytime, and anyhow
- New computing paradigms and applications like cloud-computing are emerging
- Security and privacy policies are becoming more and more complex.

Big data brings new challenges to information security

- Increase the risk of privacy leakage
  - Collection and storage of large amounts of data
  - The ownership and use rights of some sensitive data are not clearly defined
- Challenge currently existing storage and security measures
  - Bringing complex data together may cause non-compliance of corporate security management
  - Vulnerabilities in the update and upgrade of security protection methods
- Used in attack methods
  - Hackers can collect more useful information, and big data analysis makes attacks more accurate
  - Big data provides more opportunities for hackers to launch attacks

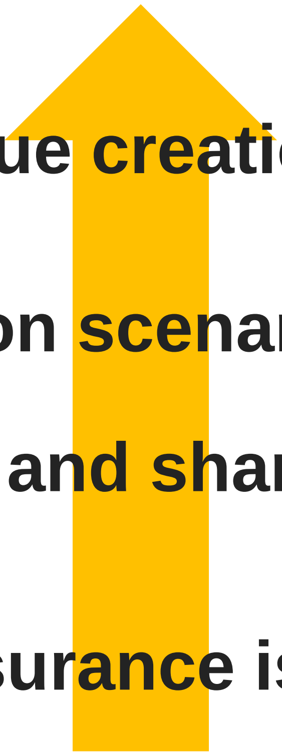
Data Governance

- Data is ubiquitous, spreads abnormally fast, and effects vary widely.


Data Governance
1
Data Development
2
Database Operations Management
3
Data Security Management
4
Reference & Master Data Management
5
Data Warehousing & BI Management
6
Document & Content Management
7
Meta Data Management
8
Data Quality Management
9
Data Architecture Management

Data quality assurance is the founda

Data Governance



Data asset value creation is the goal

Data application scenarios are carriers

Data integration and sharing are the means

Data quality assurance is the foundation

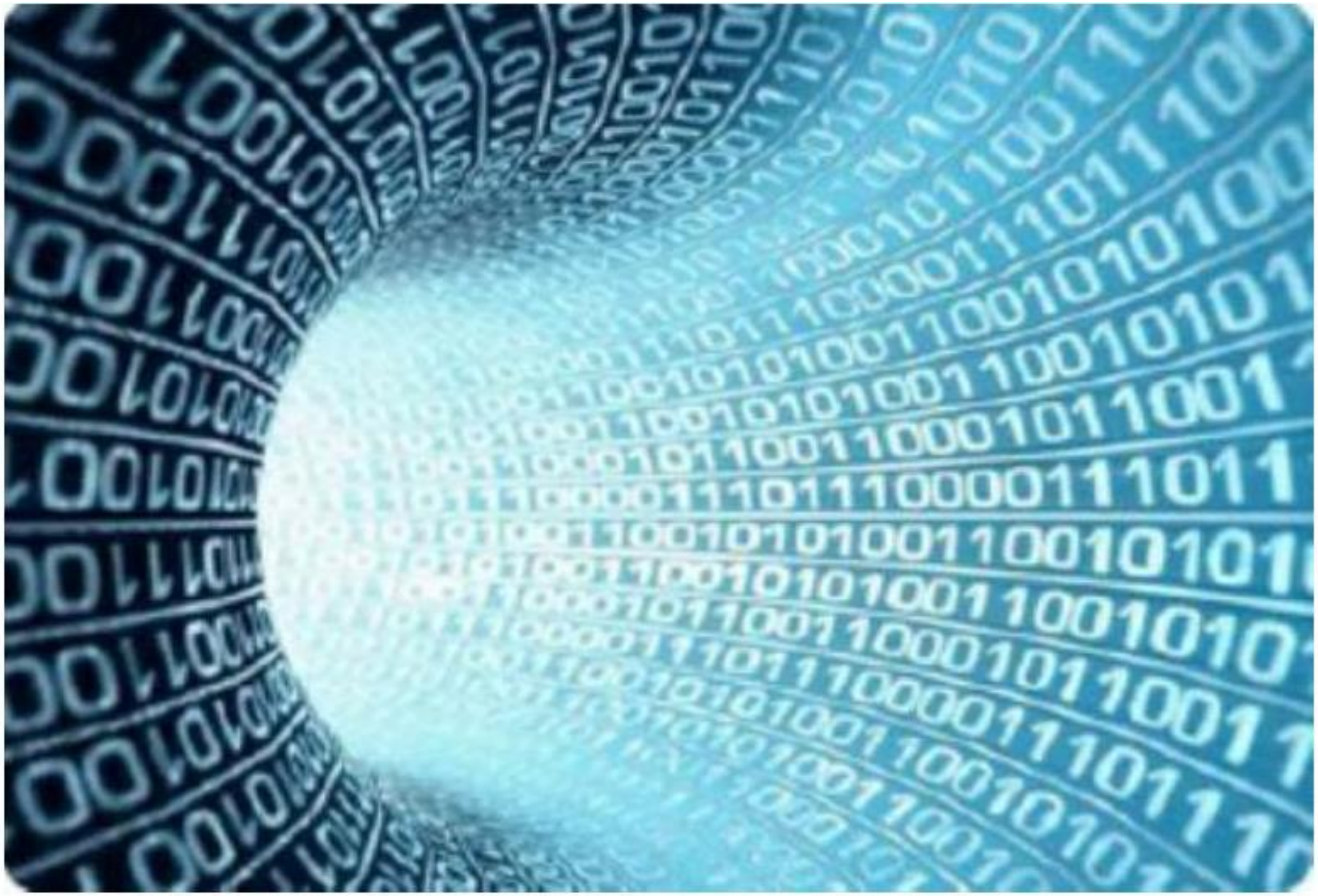
Issues in Data Governance

- The more data, the greater the responsibility
- Collecting data must conform to the logical relationship behind the application
- When aggregating data, design data granularity reasonably
- Moderately dispersed data storage is safer than highly centralized data

Digital Country

- Break data barriers, break through data islands, dispel data fears, and combat data crimes
- Break data barriers, break through data islands, dispel data fears, and combat data crimes
- Establish data thinking, improve data systems, build data government, and improve data society
- New legal system, new public order and good customs, new values

Data space-time tunnel



Question & Answer