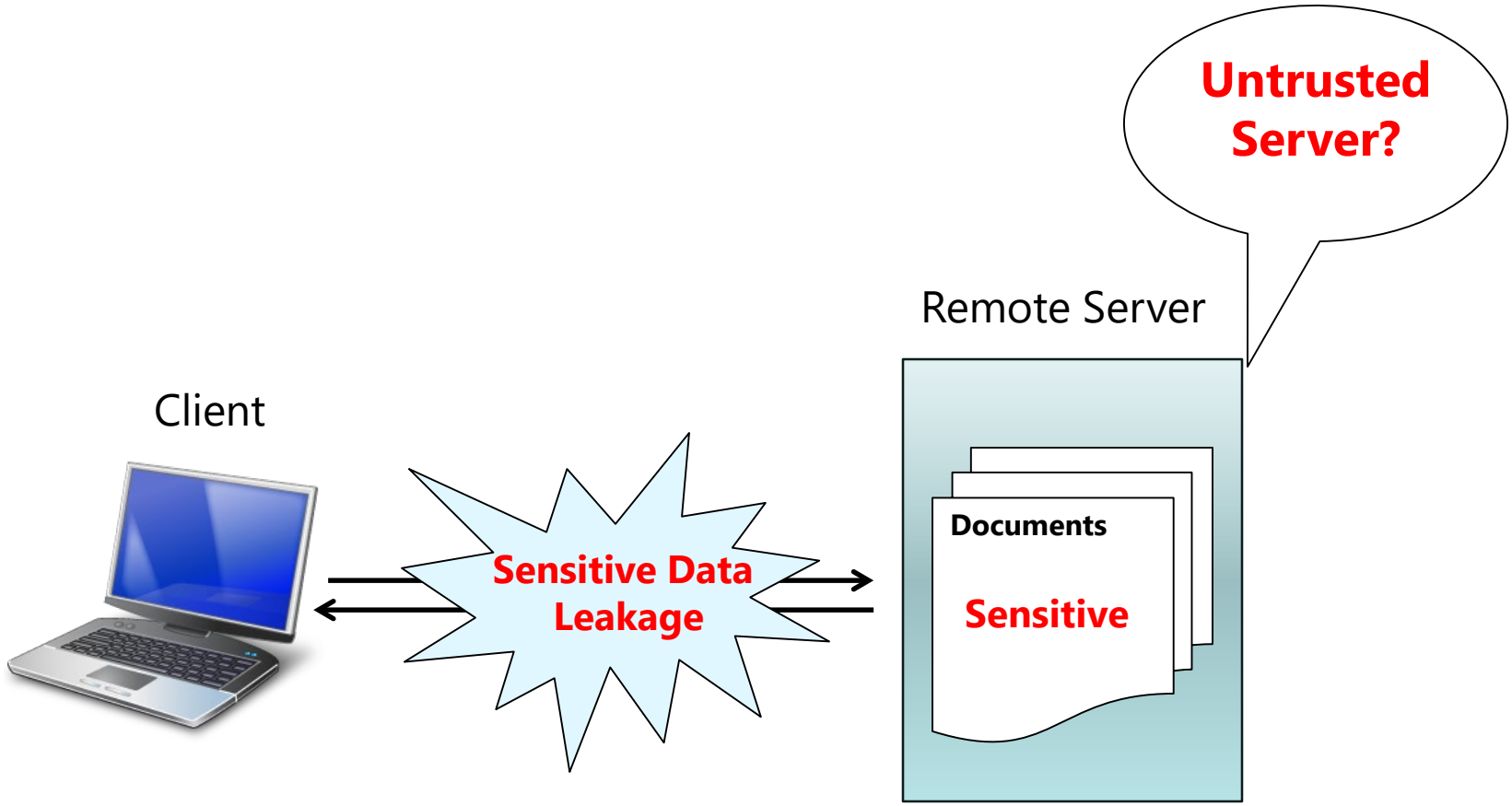
Supporting Complex Data Querying over Encrypted Data

- Murat Kantarcioglu

Outsourcing Data to Cloud

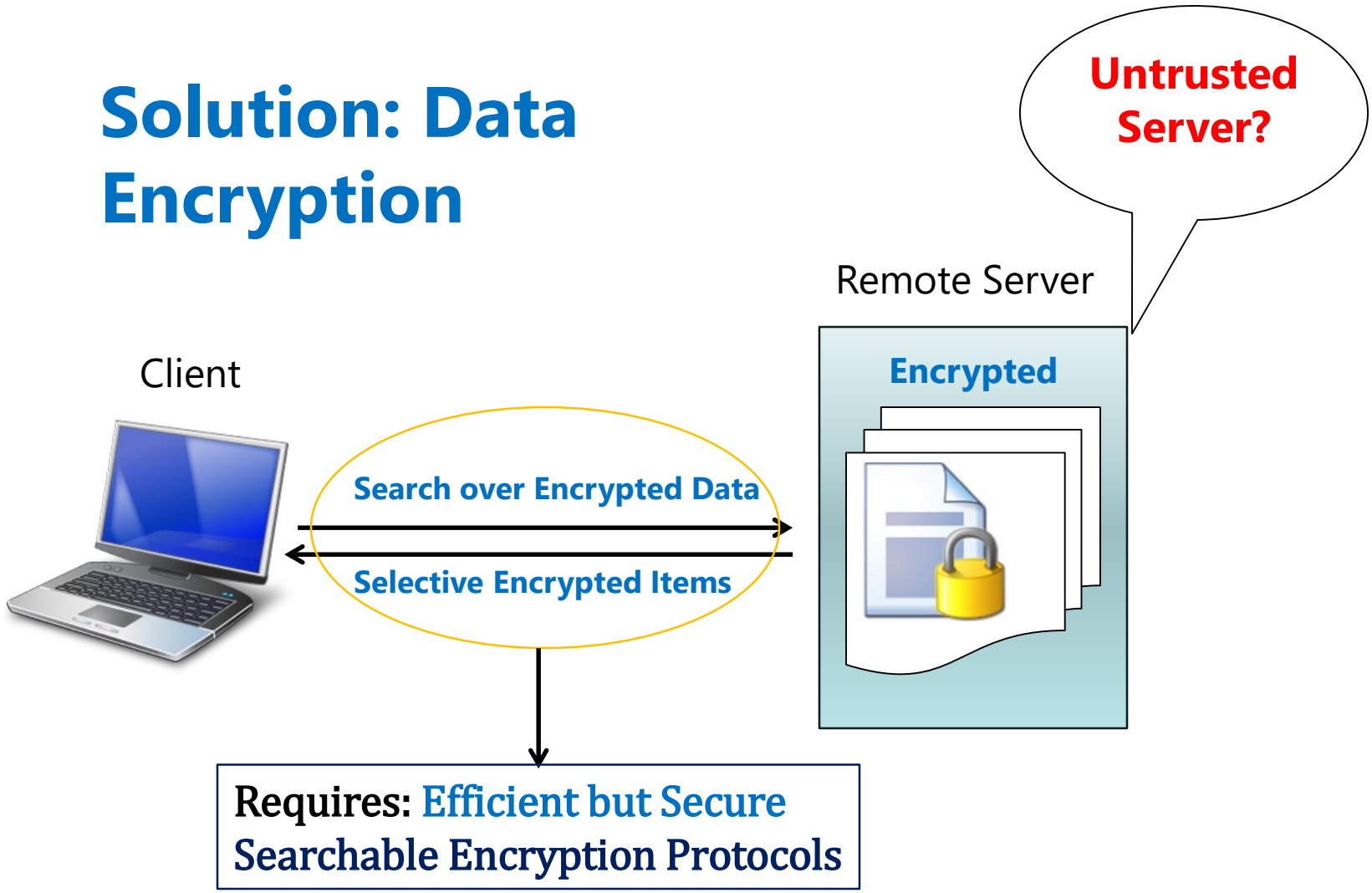
All primary cloud vendors offer RDBMS in cloud.

Introduction

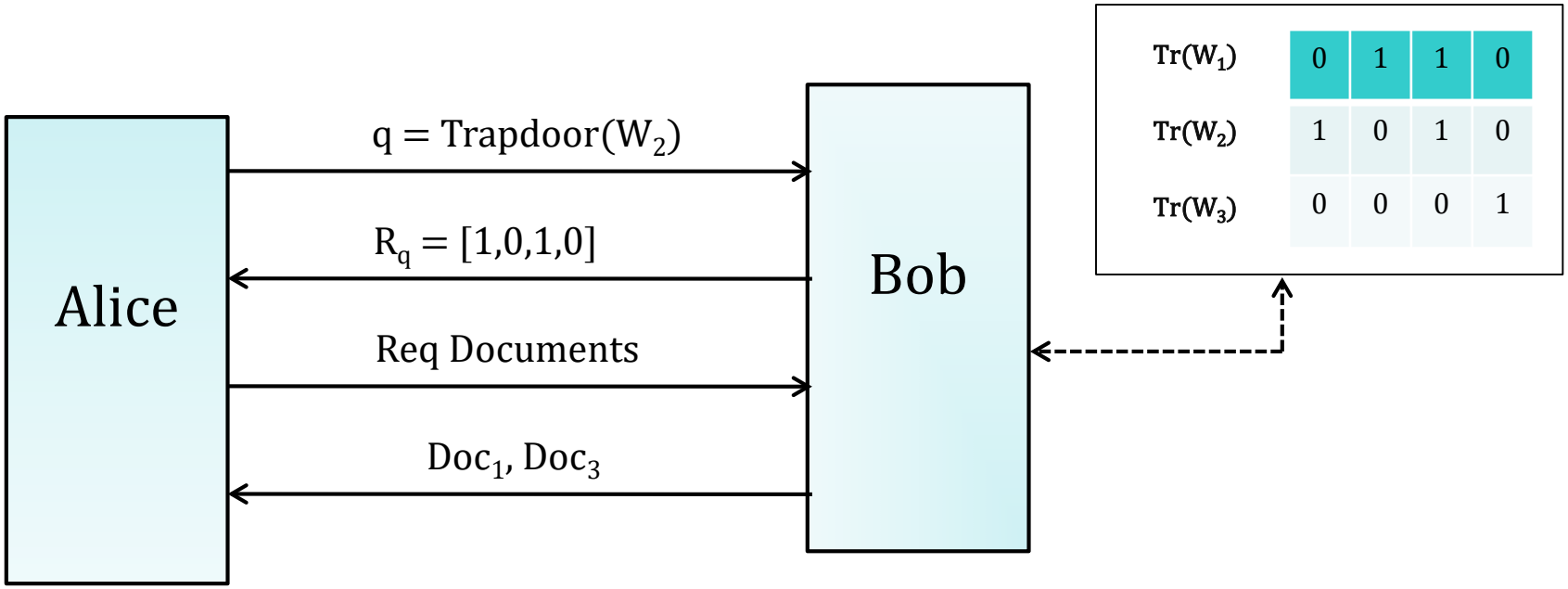


Introduction

Solution: Data Encryption



Simplified Searchable Encryption



Challenges*

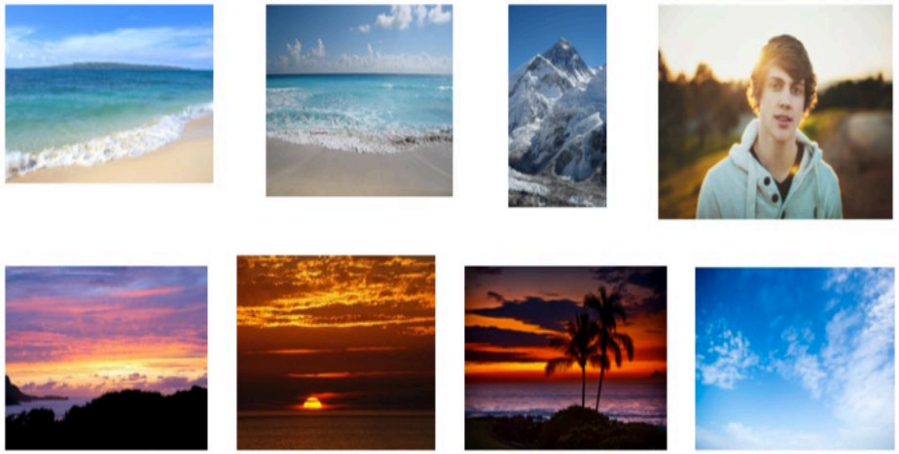
- Many general cloud storage services do not support complex crypto operations.
  - Most of the popular cloud storage Dropbox, GoogleDrive, Box, etc. doesn't support such computation.
  - People keep sensitive data on those services

- Simple encrypted keyword query supported only
  - Multimedia queries are complex in nature

* DBSEC 2016

GOALS

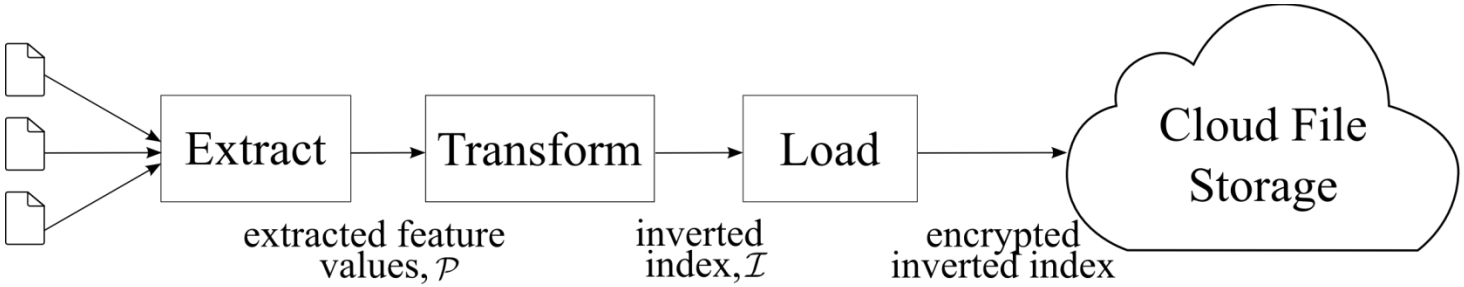
- Query encrypted multimedia data
  - Answer queries like "*Find photos of John taken last summer in Hawaii during sunset*"



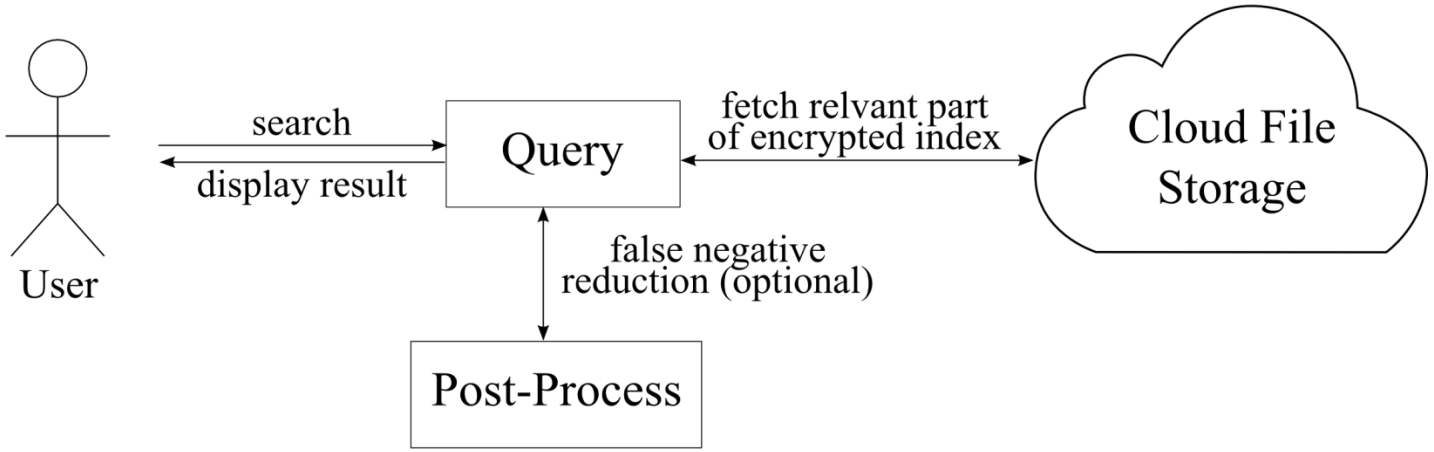
- **No special requirement from server**
  - Use existing cloud storage fileservers.

- Question: What can be achieved if we do not have any support from the server?

Phases



(a) Index creation, encryption and upload



(b) Query and post-process phase to search content

Extract output abstract example

Document ID	Feature Value pairs
$id(D_1)$	$(f_a, v_\alpha), (f_b, v_\beta), (f_b, v_\gamma)$
$id(D_2)$	$(f_a, v_\sigma), (f_b, v_\beta)$
$id(D_3)$	$(f_a, v_\alpha), (f_b, v_\gamma)$
$id(D_4)$	$(f_a, v_\delta), (f_b, v_\beta), (f_b, v_\gamma)$

(c) Extracted feature-values, $\mathcal{P}$

Transform

- In this phase extract data are converted into simpler and general form.
- Core idea is generate signature value based on feature, value combination.
- **Example**: Location
  - Input: <document_id, (longitude, latitude)>
  - We look up the address of the geo location value and generate search signatures based on country, state, city, address, etc.
    - S_1 = H("Location" || "Country "|| Country_Value)
    - S_2 = H("Location" || "State"|| State_Value)
  - Output: < S_1, document_id>, < S_2 , document_id>

Load – Overview

- Here we encrypt and load the inverted index to cloud file server.
- We observe that distribution of the length of the document list of search signatures can be approximated with **Pareto distribution.**
- Based on that we further block the document list (details in full version)
- Then we generate search signatures of the blocked document list.
- And keep certain information in a cache.

Load - Algorithm

Algorithm 1 Load encrypted index

1: **Require:** K = Master key, $\mathcal{I}$ = Inverted index of search signatures, $\mathcal{C}$ = Synchronized cache, K_C = encryption key for cache, $\mathcal{Z}$ = File storage server.
2: $b \leftarrow optimize(\mathcal{I})$
3: **for all** signature s in $\mathcal{I}$ **do**
4: $\quad blocks_s \leftarrow \lceil \frac{|\mathcal{I}[s]|}{b} \rceil$
5: $\quad$ **for** $j = 1 \rightarrow blocks_s$ **do**
6: $\quad\quad T_j^s \leftarrow H(K, s \mid\mid j \mid\mid C_1),\ K_j^s \leftarrow H(K, s \mid\mid j \mid\mid C_2)$
7: $\quad\quad sub \leftarrow \mathcal{I}[s].slice((j-1) \times b, j \times b)$
8: $\quad\quad \mathcal{E}[T_j^s] \leftarrow \varphi(K_j^s, pad(sub))$
9: $\quad$ **end for**
10: $\quad \mathcal{C}.freq[s] \leftarrow |\mathcal{I}[s]|$
11: **end for**
12: **for all** trapdoor t in $\mathcal{E}$ **do**
13: $\quad \mathcal{Z}.write(t, \mathcal{E}[t])$
14: **end for**
15: $C_{sig} \leftarrow H(K_C \mid\mid C_3, 1)$
16: $\mathcal{Z}.write(C_{sig}, \varphi(K_C, \mathcal{C}))$

Query and Post Process – Overview

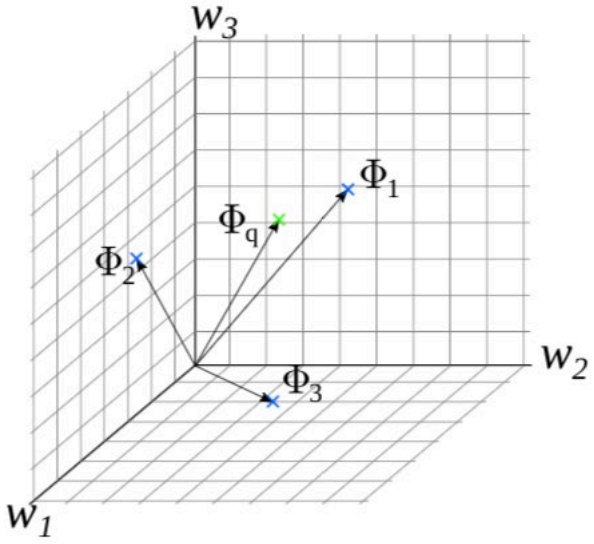
- Given a query we first extract and transform it
- Next we generate search signatures
- Generate trapdoors
- Get those trapdoor related information
- Then decrypt the document ids
- Finally, remove false positives (if necessary)

Complex Feature – Face recognition

- An example of complex query: face recognition.
- Interesting applications in homeland security!
- We adopted Eigenface mechanism to support face recognition.

Eigenface – Review – Finding Eigen Vectors

- We adopted EigenFace recognition method
- We start with M faces of size $N \times N$
- Let, $\{\Gamma_1, \dots, \Gamma_M\}$ be the $N^2 \times 1$ (vector) representation of the square faces
- $\Psi = \frac{1}{M}\sum_{i=0}^{M} \Gamma_i$ be the average of face vectors.
- Subtract average from each face, $\phi_i = \Gamma_i - \Psi$
- Find M eigen vectors u_i of $\boldsymbol{A^T A}$, where $\mathrm{A} = [\phi_1 \phi_2 \dots \phi_M]$
- We take top K of these Eigen vectors.
- Use projection for matching



Faces in Eigen Space

Encrypted Eigenface Recognition - ETL

- **Extract:** Find face locations in image
  - *id(D_1):<"Face", (X:10px, Y:12px, H: 120px, W: 120px)>*
- **Transform:**
  - Convert face to point in EigenFace Plane ω
  - Define Euclidian LSH function
  - *bucket_id* = Find LSH bucket ids of ω
  - *search_signatures = generate_signatures(bucket_ids)*
- **Load:**
  - Upload *search_signature*s and document assignments

Encrypted Eigenface Recognition - QP

- **Query**:
  - Given a new Face
  - Convert to a point in eigen plane point
  - Create *bucket_ids* of previously defined LSH schema.
  - Create *search_signatures* of the *bucket_ids*
  - Now search the search *search_signatures* in the encrypted index
- **Post Process**:
  - Remove the false positives due to LSH

Experiments – Dataset Generation

- Randomly selected **20,109** images from *Yahoo Flickr Creative Commons 100 Million Dataset (YFCC100M)*
- Size **42.3GB**
- Average file size **2.15MB**
- Number of faces detected **7027**
- Image with latitude and longitude embedded in EXIF data **4102**

Experiment – Features

- Our prototype image storage system can handle 4 types of features
  - Location
    - Find images based on location
  - Time
    - Find images that are taken on a specific time or in a time range
  - Texture and Color
    - Find images that are similar, e.g., images of sunset, sky, etc.
  - Face
    - Find images of a particular person.

Experiments – Index Size

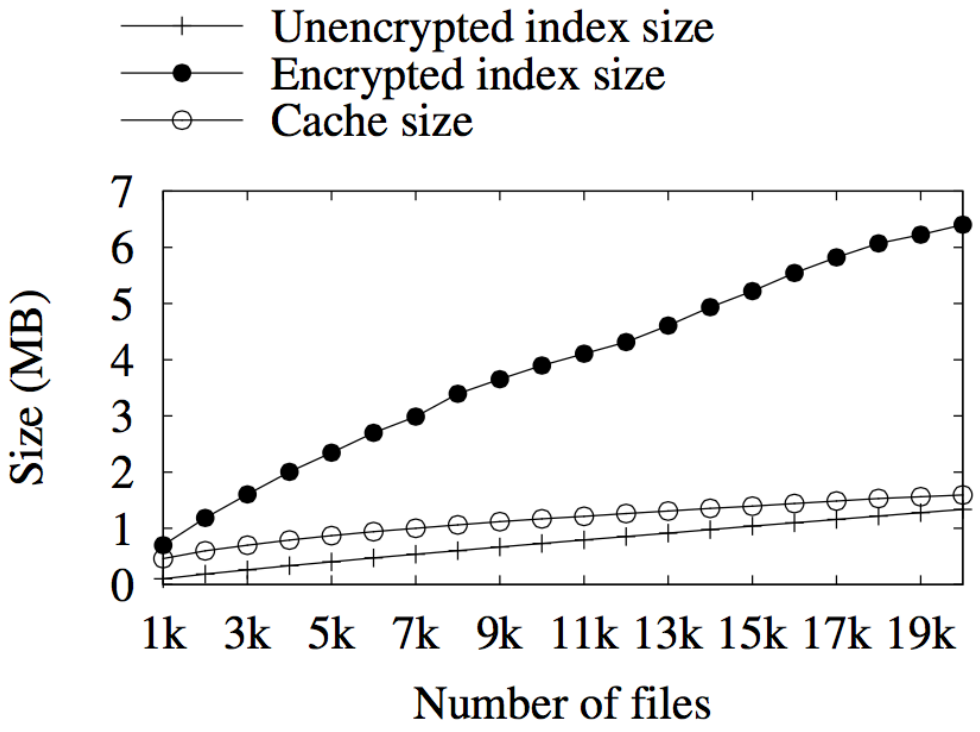
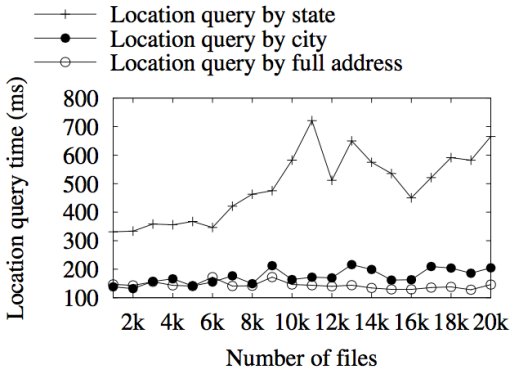
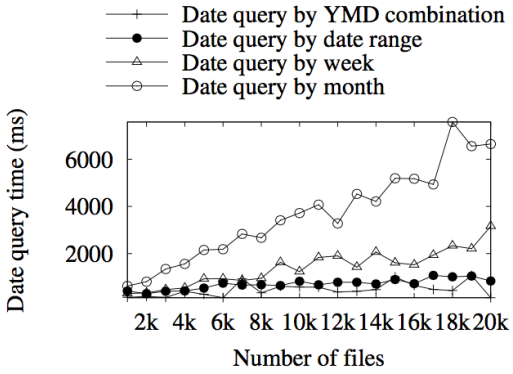


Fig. 2. Index and cache sizes

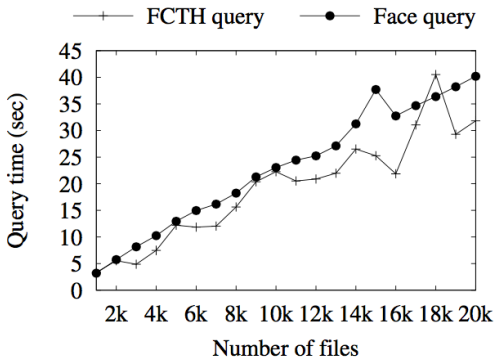
Experiment Query Time



(a) Location Query



(b) Date Query



(c) FCTH and Face Query

Fig. 4. Time required for different type of queries vs number of files.

Conclusion

- We have proposed a practical framework for performing complex queries over encrypted data.
- Uses series of simple encrypted key-word queries to answer complex queries
  - This leaks access pattern and some similarity info. about queries

How do we protect against access pattern leakage attacks ??

- Almost all practical searchable encryption schemes leak data access pattern for efficiency which is subject to statistical attacks.

- Do we need the optimal protection of oblivious ram to ensure individual privacy?

Differential Privacy

- Minimize the risk by bounding the probability of disclosure caused by participating in a dataset
  - T_1, T_2 are sibling datasets

$$\frac{Pr[F(T_1)=U]}{Pr[F(T_2)=U]} \leq e^{\epsilon}$$

- Add random noise to query responses
  - Laplace noise (μ, λ), where $\lambda = S(Q) / \epsilon$
  - S(Q): sensitivity, ϵ: privacy parameter

$$S(\mathcal{Q}) = \max_{\forall T_1, T_2} \sum_{i=1}^{q} |Q_i^{T_1} - Q_i^{T_2}|.$$

Differentially Private Access Pattern Leakage

- *Access Pattern:* Memory addresses of the encrypted records that are accessed against queries

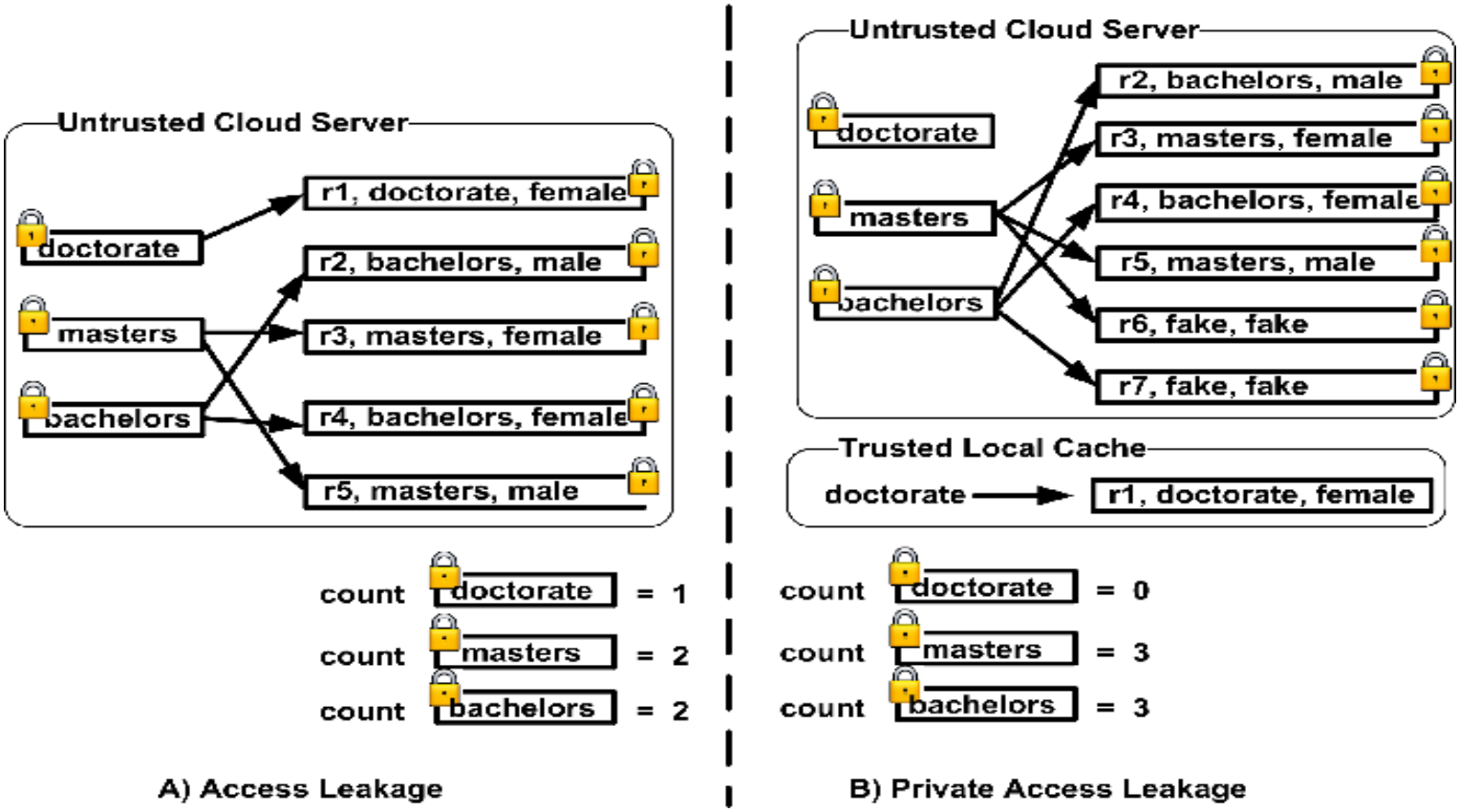
- Differentially private access pattern statistics corresponds leaking diff. private count queries in the form of:

  select count(*) from Database where Predicate is true

- Given query set Q = $\{q_1, \ldots, q_n\}$, DP adds Laplace noise with magnitude $S(Q)/\epsilon$ to the true response
  - S(Q): query set sensitivity
  - ϵ: privacy parameter

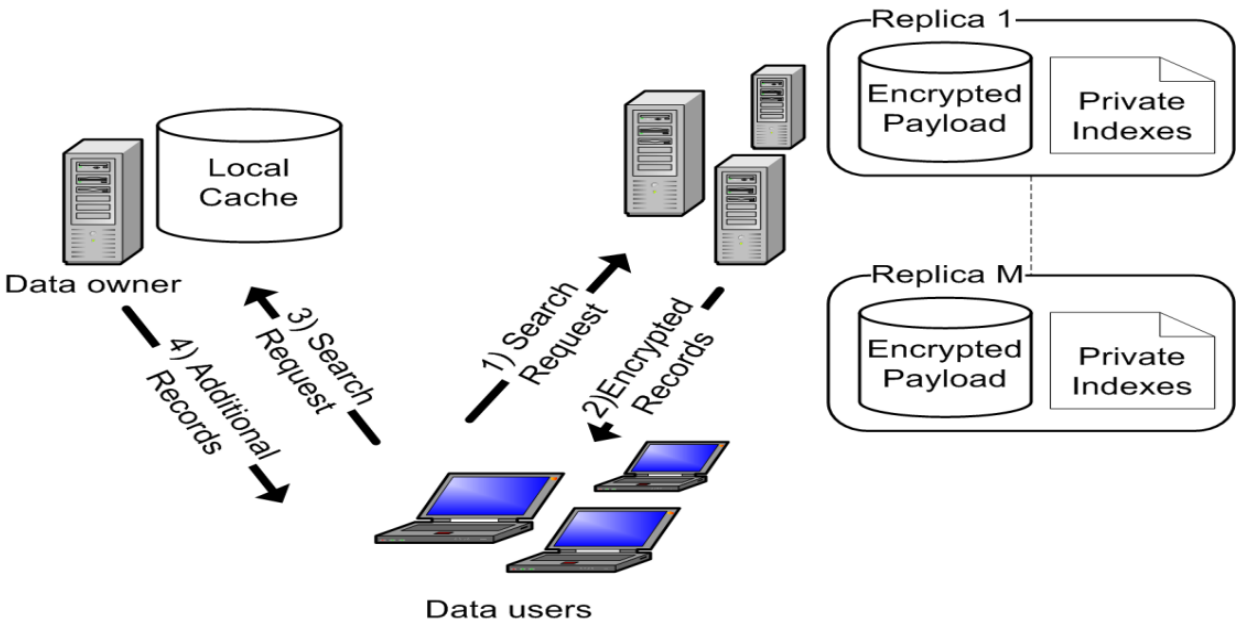
Privacy-Aware Searchable Encryption

- Privacy-aware searchable encryption protects access statistics with differential privacy.



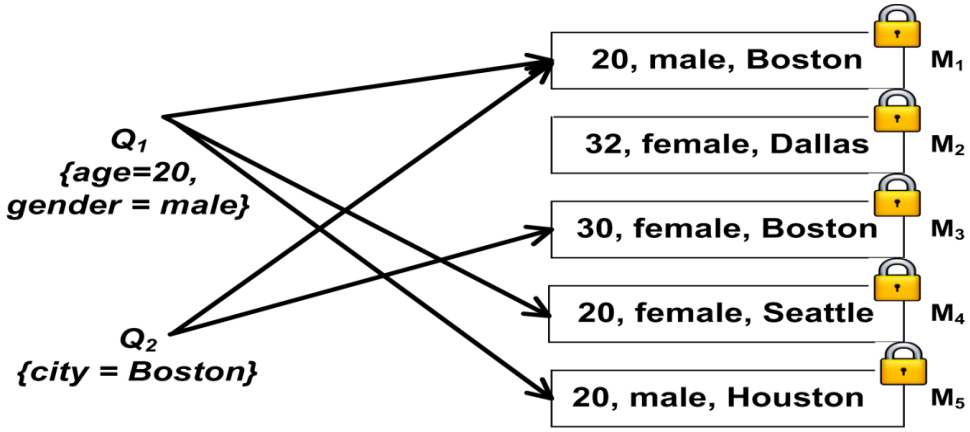
Private Search Scheme

- Data owner builds private indexes on the desired subsets of the attributes (e.g., {age}, {age, gender})
- To satisfy differential privacy (DP), owner keeps limited amount of records in local cache and injects some fake records into the outsourced set



Differentially Private Access Pattern

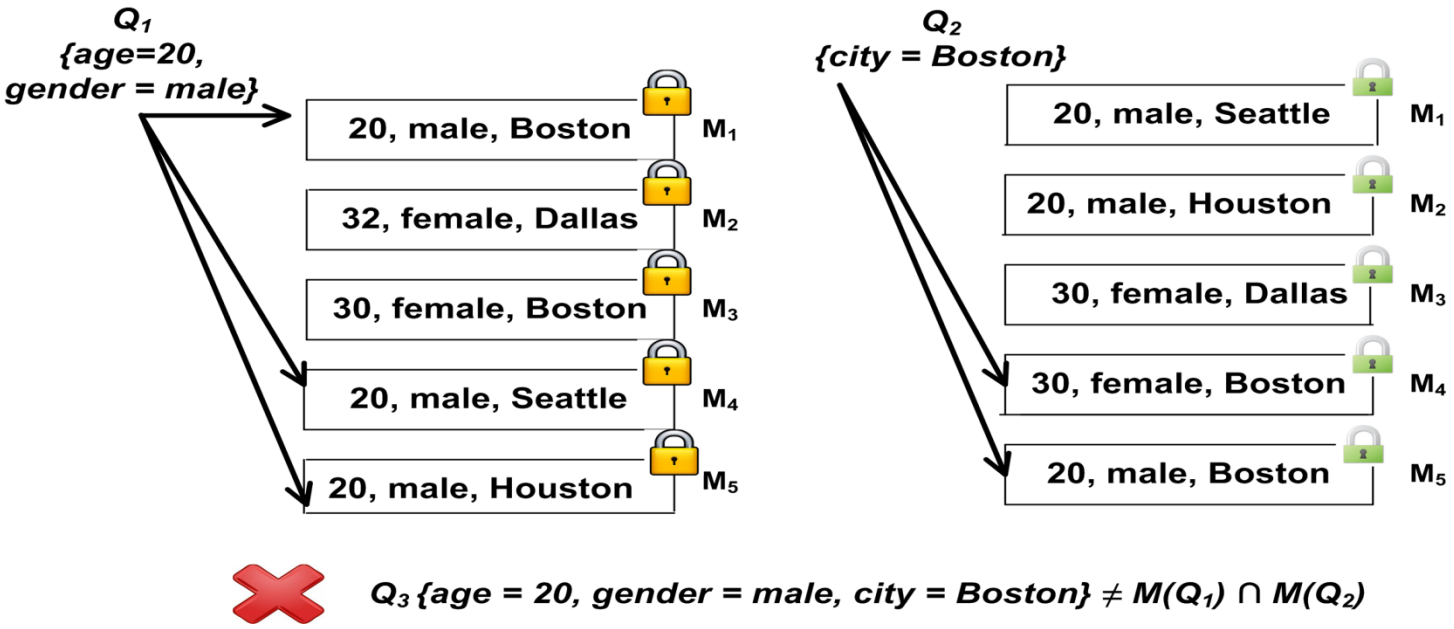
- Query set sensitivity is equal to the number of observable query interfaces.
- Owner provides initial query interfaces (e.g., {age, gender}, {city})
- Interactions among initial interfaces may lead to new observable interfaces (e.g., {age, gender, city})



Q_3 {age = 20, gender = male, city = Boston} = $M(Q_1) \cap M(Q_2) = M_1$

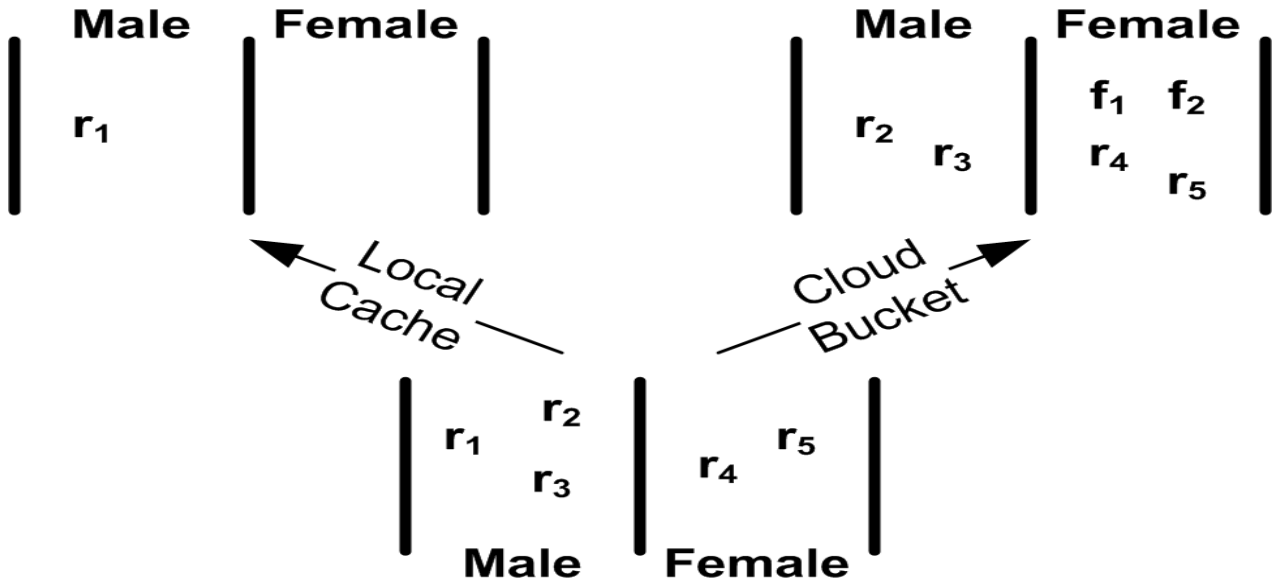
Data Replication

- Replication of data for distinct initial query interfaces prevent additional interfaces due to interactions
- For each replication, data source is subject to random permutation and encryption with distinct keys



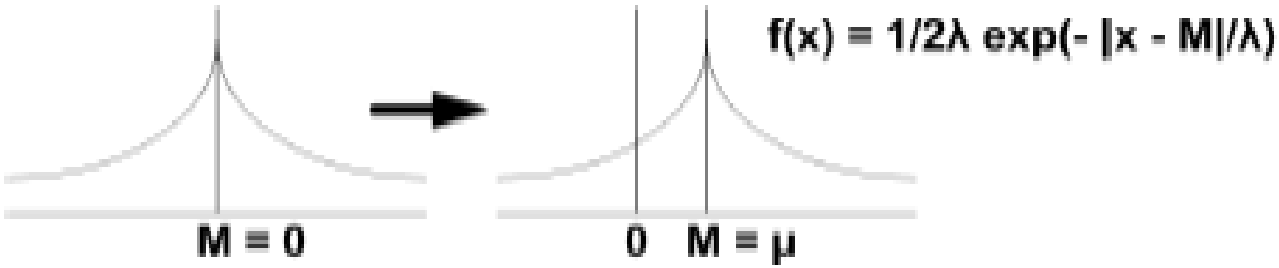
Private Index Construction

- Private indexes enforce obfuscation on the access pattern in addition to the content protection
- Positive noise is incorporated by fake record injection while negative noise requires local cache placement



Private Index Construction

- Amount of negative noise should be limited to satisfy the capacity constraint of the local cache
- Mean shift on the positive axis of Laplace distribution enables capacity enforcement with the cost of more fake record injection to the cloud buckets.

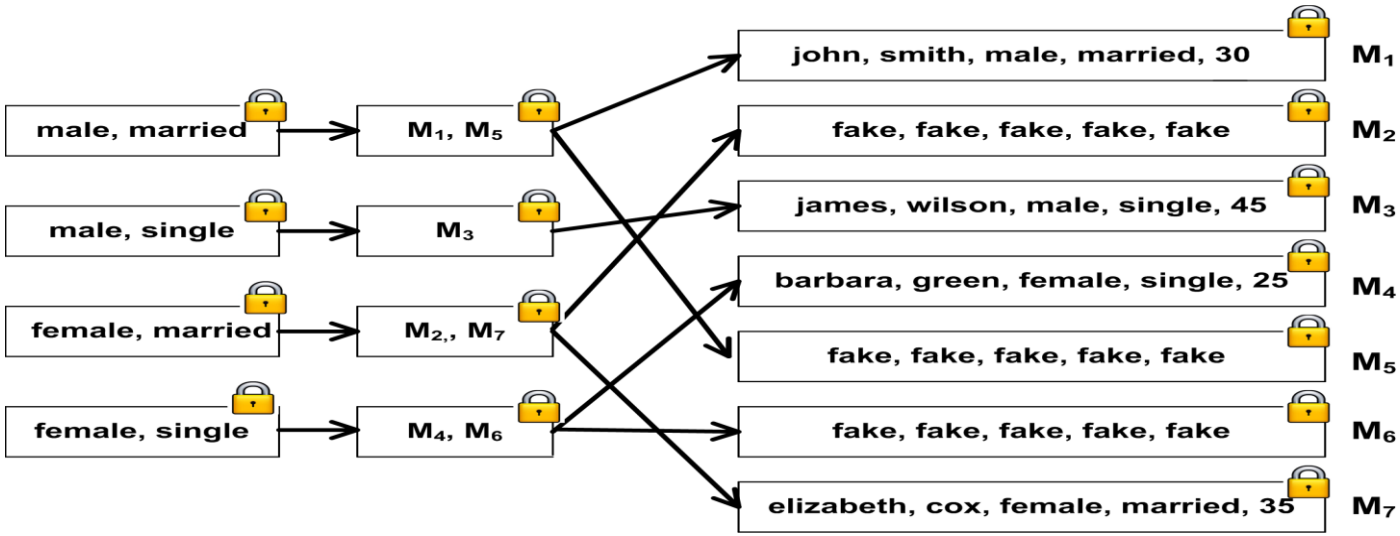


Record Encryption

- Both index cells and records that are sent to the cloud are subject to encryption
- Memory addresses are encrypted through a random oracle to satisfy adaptive semantic security model

$$\pi_{c_k} = F_{K_{cell}}(c_k),\ key(c_k) = F_{K_{loc}}(c_k)$$

$$\pi_{loc(c_k^i)} = (H_{key(c_k)}(\alpha_i) \oplus loc(c_k^i),\ \alpha_i)$$



Experimental Setup

- We selected a publicly available dataset of real personnel identifiers, namely Census-Income dataset
- Dataset consists of 48,842 individual records, each with 8 categorical and 4 numerical attributes
- We selected random query interfaces from all possible interfaces that can be generated on categorical attributes
- Default protocol parameters:
  - $\epsilon = 0.5$, $|\Delta| = 7$, $C = 2500$
- 1000 random queries are issued against the server using selected query interfaces
- Bandwidth consumption of the local cache and cloud server is utilized as the main evaluation metric

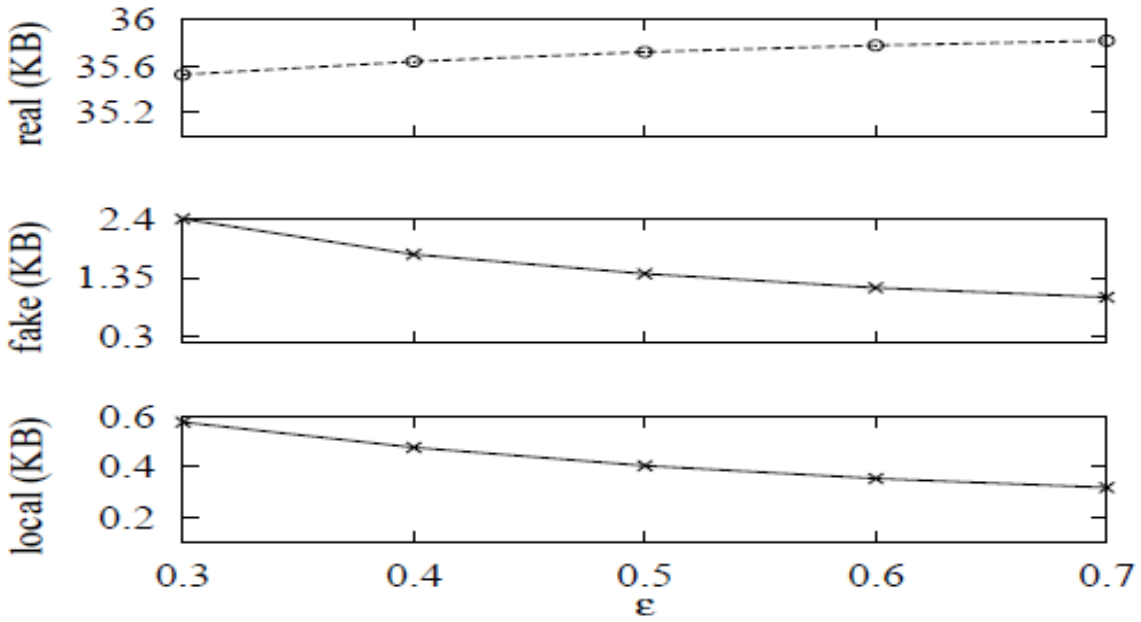
Experiments - 1

- Overhead of the proposed scheme varied between 1.01 ~ 2X faster than typical ORAM implementation

ε	Server Overhead (%)	Cache Overhead (%)
0.3	**6.62**	**1.60**
0.4	**4.88**	**1.32**
0.5	**3.93**	**1.12**
0.6	**3.25**	**0.97**
0.7	**2.78**	**0.87**

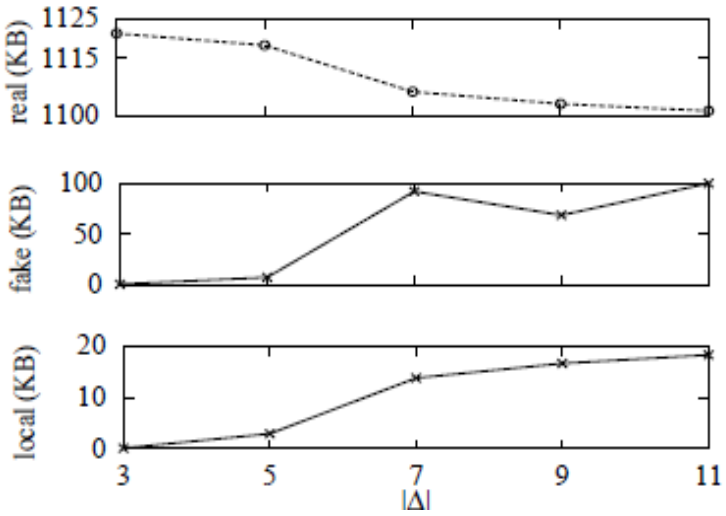
Experiments - 2

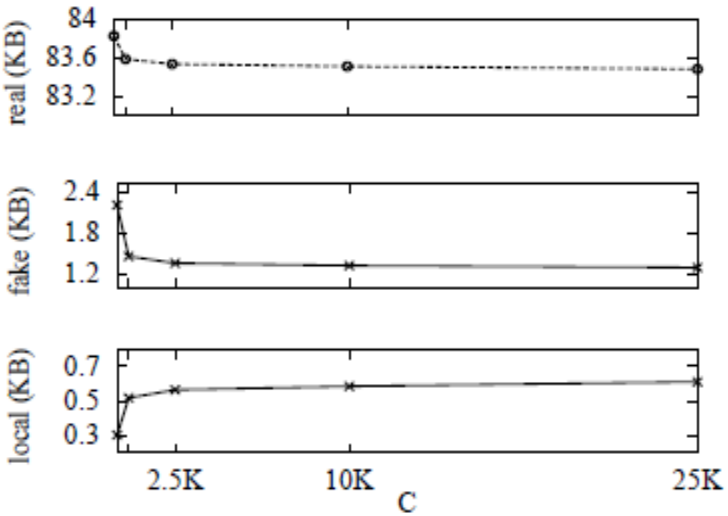
- With increasing ϵ, both fake record and local cache retrieval decreases due to less noise for DP



Experiments - 3

- More query interfaces (|Δ|) leads to increase in the number of fake records and local cache placements since sensitivity is proportional to |Δ|
- With increasing local cache capacity, fake record retrieval from the server decreases



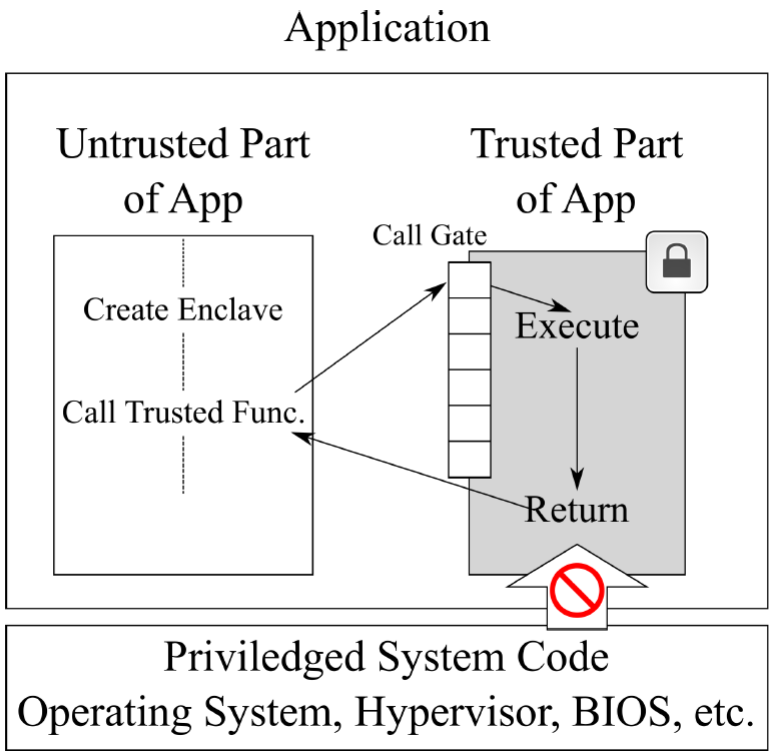


Use Hardware Support for Efficient Oblivious Complex Data Analysis **

- ***Querying is not enough for many cloud applications.***
- ***Need to build complex ML models***
- ***Data scientists are not crypto experts***
  - ***Like to use PLs such Python and use libraries like Pandas etc.***
- ***Need to protect*** *the secrecy and integrity* of big data and the ML models using encryption
- Need to enable **general programming language** for data processing while satisfying data obliviousness
- **Make it efficient and practical enough for general use**

** ACM CCS 2017

Intel SGX ??


Application
Untrusted Part of App
Trusted Part of App
Call Gate
Create Enclave
Call Trusted Func.
Execute
Return
Priviledged System Code
Operating System, Hypervisor, BIOS, etc.

How to Support Data Obliviousness Efficiently ??

- Idea 1: Use generic ORAM construction and do not care about the specific data analytics workload
  - May be too costly in many cases for big data
- Idea 2: Create specific but oblivious data analytics functions
  - Matrix multiplication is oblivious !!
  - Challenge: many tasks require non-obvious algorithms to satisfy ORAM security definition
  - Challenge: many users cannot be trusted to write oblivious functions by default

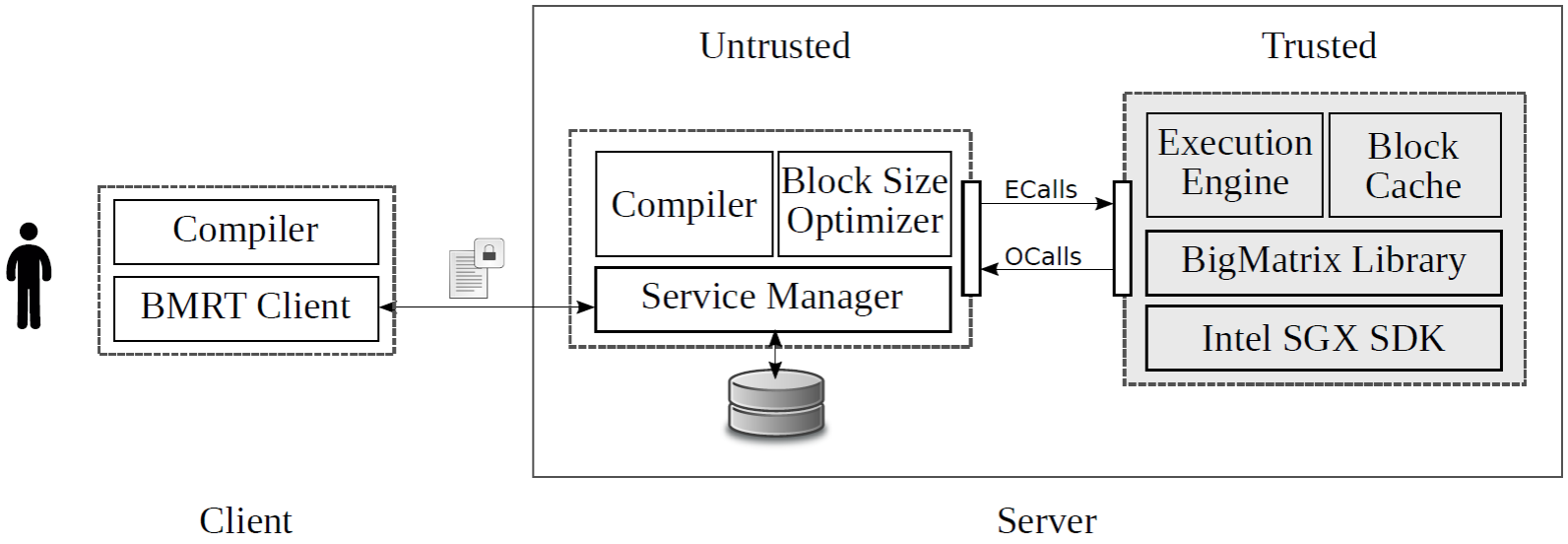
How to Support Data Obliviousness Efficiently ??

- Idea, remove **If statements using vectorization**

```
sum = 0, count = 0
for i = 0 to Person.length:
    if Person.age >= 50:
        count++
        sum += P.income
print sum / count
```

```
S = where(Person, "Person['age'] >= 50")
print (S .* Person['income'] ) / sum(S)
```

SGX- BigMatrix Architecture



SGX BigMatrix

Compiler

- Compiles our python like language into **basic commands**
- Data obliviousness using **data oblivious building blocks** and **operation vectorizations**

Input

```
x = load('path/to/X_Matrix')
y = load('path/to/Y_Matrix')
xt = transpose(x)
theta = inverse(xt * x) * xt * y
publish(theta)
```

Compiler-Output

Output

```
x = load(X_Matrix_ID)
y = load(Y_Matrix_ID)
xt = transpose(x)
t1 = multiply(xt, x)
unset(x)
t2 = inverse(t1)
unset(t1)
t3 = multiply(t2, xt)
unset(xt)
unset(t2)
theta = multiply(t3, y)
unset(y)
unset(t3)
publish(theta)
```

Support for Basic Data Science

- E.g., SQL, Matrix Operations etc.

Input

```
I = sql('SELECT *
FROM person p
JOIN person_income pi (1)
ON p.id = pi.id
WHERE p.age > 50
AND pi.income > 100000')
```

Other Important Features

- **Automatic Sensitivity Analysis** for flagging sensitive information disclosure
  - I.e., using sensitive output for allocating a new array
- **Cost based and secure optimization** for optimizing blocking
  - Sgx do not support efficient data buffering

Experimental Evaluation

- Performed linear regression on two popular datasets

Data Set	Rows	BigMatrix Encrypted
USCensus1990	2,458,285	3m 5s 460ms
OnlineNewsPopularity	39,644	2s 250ms

Table: Time results of linear regression on real datasets

- Performed Page Rank on three popular datasets

Data Set	Nodes	BigMatrix Encrypted
Wiki-Vote	7,115	97s 560ms
Astro-Physics	18,772	6m 41s 200ms
Enron Email	36,692	23m 19s 700ms

Comparison with ObliVM

Matrix Dimension	ObliVM	BigMatrix SGX Enc.	BigMatrix SGX Unenc.
100	28s 660ms	10ms	10ms
250	7m 0s 90ms	93ms	88ms
500	53m 48s 910ms	706.66ms	675.66ms
750	2h 59m 40s 990ms	2s 310ms	2s 260ms
1,000	6h 34m 17s 900ms	10s 450ms	10s 330ms

Table: Two-party matrix multiplication time in ObliVM vs BigMatrix

Current Work: TEE + Searchable Encryption

- Building searchable encryption index requires storage and memory on the client side
  - Require complex processing for images etc.
- Securely outsource the index construction to SGX
  - Send encrypted doc-id, token-id pairs to SGX
  - Use SGX to securely build the index

Questions?

- **This work is supported by the following grants:**
  - Air Force Office of Scientific Research Grant FA9550-12-1-0082, National Institutes of Health Grants 1R01LM009989 and 1R01HG006844, National Science Foundation (NSF) Grants Career-CNS-0845803, CNS-0964350, CNS-1016343, CNS-1111529, CNS-1228198, Army Research Office Grant W911NF-12-1-0558.

- **This talk is based on the following papers:**
  - M. S. Islam, M. Kuzu, M. Kantarcioglu: Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. NDSS 2012
  - M. S. Islam, M. Kuzu, M. Kantarcioglu: Inference attack against encrypted range queries on outsourced databases. ACM CODASPY 2014: 235-246
  - M. Kuzu, M. S. Islam, M. Kantarcioglu: Efficient privacy-aware search over encrypted databases. ACM CODASPY 2014: 249-256
  - Fahad Shaon, Murat Kantarcioglu: A Practical Framework for Executing Complex Queries over Encrypted Multimedia Data. DBSec 2016: 179-195
  - Fahad Shaon, Murat Kantarcioglu, Zhiqiang Lin, Latifur Khan: SGX-BigMatrix: A Practical Encrypted Data Analytic Framework With Trusted Processors. ACM Conference on Computer and Communications Security 2017: 1211-1228