An Efficient and Secure Data Sharing Framework using Homomorphic Encryption in the Cloud

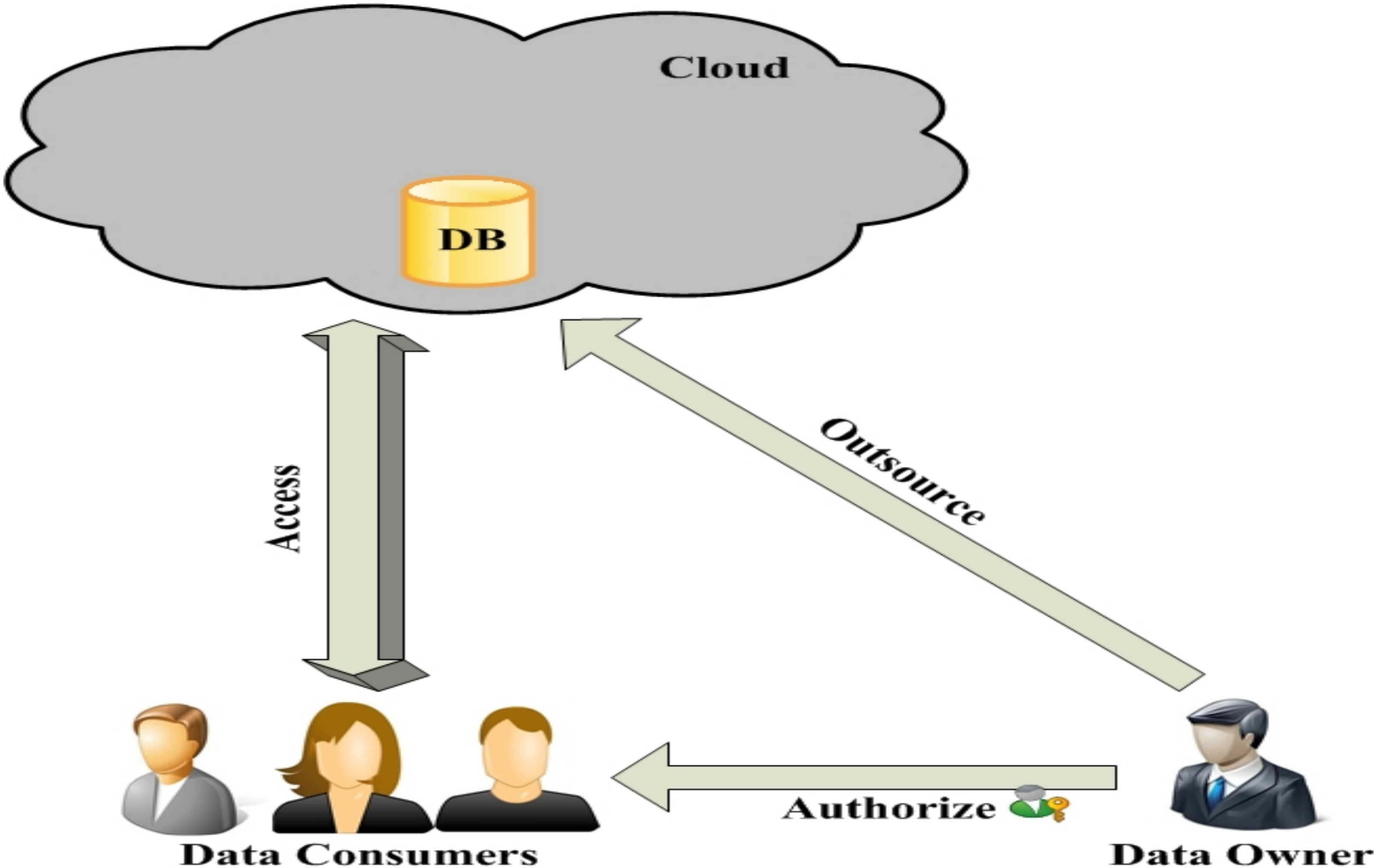
Sanjay Madria
Professor and Site Director for NSF I/UCRC Center on Net-Centric Software and Systems
Missouri University of Science & Technology, Rolla, MO 65401, USA
madrias@mst.edu

Joint work with Bharath K. Samanthula, Gerry Howser, Yousef Elmehdwi
Missouri University of Science & Technology, Rolla, MO 65401, USA

Outline

- Motivation
- Problem Statement
- Related work
- Main Contribution
- Preliminaries
- Proposed Solutions
  - SDS Framework
    - Correctness proof
    - Example
  - Modified-SDS Framework
- Conclusion / Future Work

SYSTEM MODEL



PROBLEM STATEMENT

- Data owner Alice outsources data to the cloud after encryption

- Goal: To provide a fine-grained access control to various users authorized by Alice

MOTIVATION

- Data is outsourced to the cloud
  - Cost-efficiency and flexibility

- For privacy issues – encrypting the data seems to be a better choice

- *Access Control on Encrypted Data in the Cloud*
  - Relies heavily upon encrypted data in the cloud
  - One of the reasons in using encrypted data in the cloud is protecting the data from the cloud itself
  - However, encrypted data on the cloud places limitations upon data searches and queries

Cont..

- Some important issues to be addressed in Access Control
  - Fine-grained access control with efficient user revocation
  - Rejoin of revoked users
  - Collusion between users
  - Collusion between a user and the cloud
  - Efficient modification of user access privileges

RELATED WORK

- Yang et al. [1] proposed a new fine-grained access control protocol using Symmetric encryption and Proxy Re-encryption schemes.

- Disadvantages:
  - Symmetric encryption provides weaker security guarantees
  - Possibility of Information leakage:
    - Rejoin of revoked user
    - Collusion of revoked user with authorized user Bob
    - Collusion between Bob and the cloud

OUR CONTRIBUTION

- Developed a new Secure Data Sharing (SDS) framework to achieve fine-grained data sharing/access control over data outsourced to the cloud that provides following features:
  - Efficient user revocation
  - Efficient and secure re-join of a previously revoked user
  - Prevention of collusion between a user and the CSP
  - Prevention of collusion between a revoked user and an authorized user.
  - Generic Approach

Preliminaries

- SDS uses two specific encryption techniques: additive homomorphic encryption + proxy re-encryption

- *Additive homomorphic (Probabilistic) encryption*:
  - $E_{pk}(x + y) = E_{pk}(x) \cdot E_{pk}(y) \bmod N^2$
  - $E_{pk}(c \cdot x) = E_{pk}(x)^c \bmod N^2$
  - The encryption scheme is semantically secure

where N is the RSA modulus which is also a part of the public key pk.

CONTD…

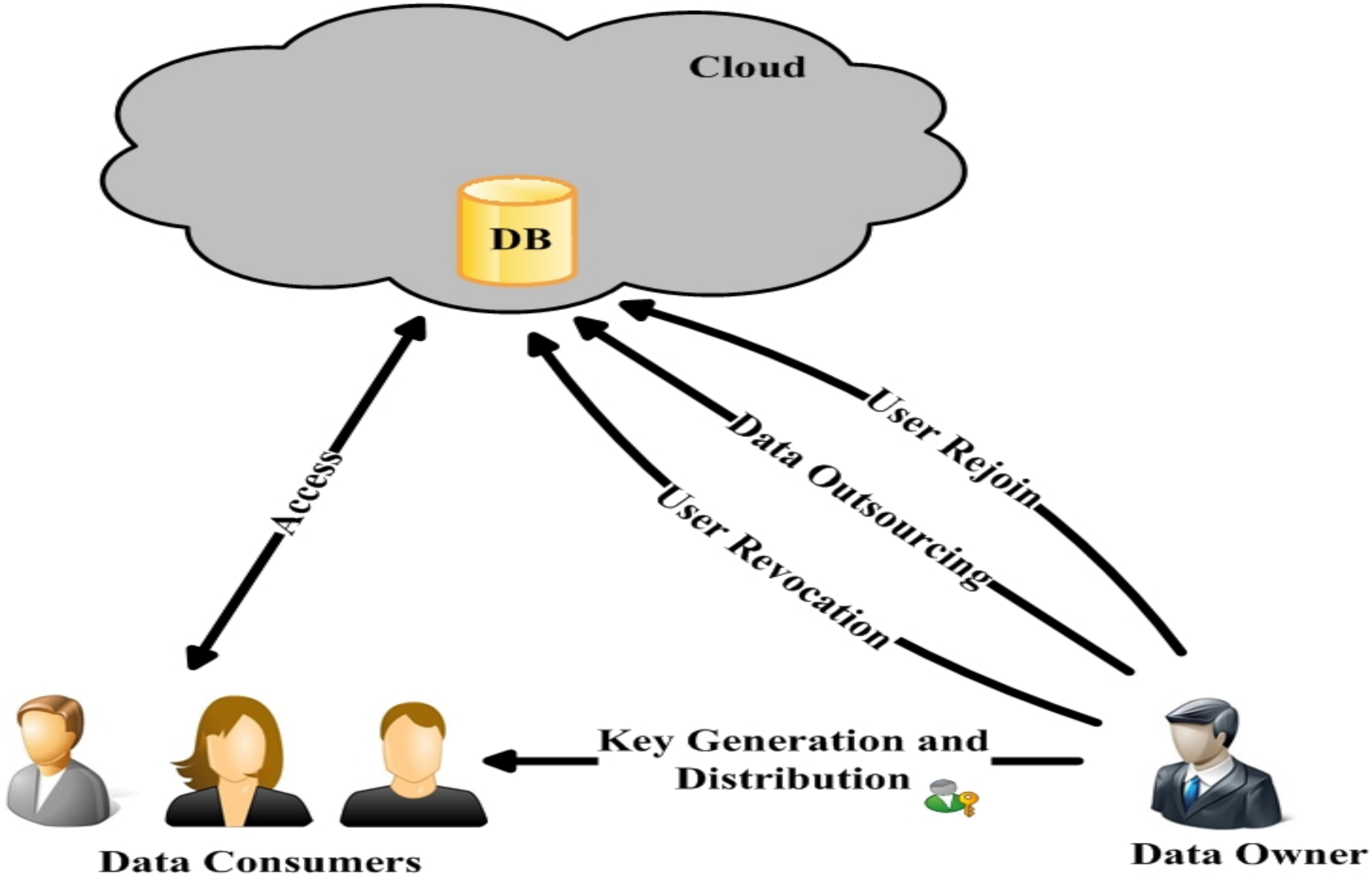
- Proxy Re-encryption:
  - Allows a "semi-trusted" proxy *T* to convert ciphertext under Alice's public key into one encrypting the same plaintext under Bob's public key:

    $$PRE(E_{pk_a}(x), rk_{pk_a \rightarrow pk_b}) \rightarrow E_{pk_b}(x)$$

    where pk_a and pk_b are the public keys of Alice and Bob respectively.
  - Proxy only knows the re-encryption key $rk_{pk_a \rightarrow pk_b}$
  - Nothing is revealed about the plaintext x to *T*.

Proposed SDS Framework

- Utilizes additive homomorphic encryption and proxy re-encryption schemes as underlying sub-routines
- Our Secure Data Sharing (SDS) framework consists of five stages:
  1) Key Generation and Distribution
  2) Data Outsourcing
  3) Data Access
  4) User Revocation
  5) User Rejoin



Proposed SDS Framework

Key Generation and Distribution

- Acts as an initialization step
- The data owner (Alice) generates two kinds of key pairs

  - Master key pair – (pk_a , pr_a). Where, pk_a and pr_a are the public and private keys of Alice.

  - For each authorized user, say Bob, Alice creates a public/ private key pair (pk_b, pr_b) and sends it to Bob.

Data Outsourcing

- For each data record d, Alice proceeds as follows:
  - Let $d_1, \ldots, d_n$ denote the attribute values of d
  - Picks n+m number of random numbers - $r_1, \ldots, r_{n+m}$
  - $d' = < d_1 + r_{n+1}, \ldots, d_n + r_n, r_{n+1}, \ldots, r_{n+m} >$
    $= < d'_1, \ldots, d'_{n+m} >$

    where r_i is a random number chosen from Z_N
  - Assume $E_{pk_a}(d') = < E_{pk_a}(d'_1), \ldots, E_{pk_a}(d'_{n+m}) >$
  - For a particular user, say Bob, we have the following two cases:
    - Case 1: Bob has access to a set of attributes (S) in d
    - Case 2: Bob is not authorized to access d

Data Outsourcing (contd…)

- For each authorized user Bob on d, Alice creates authorization token T^d_b
- ***Case 1***:
  - $T^d_b = \{Bob, rk_{pk_a \to pk_b}, \langle E_{pk_b}(\alpha_1), \ldots, E_{pk_b}(\alpha_{n+m}) \rangle\}$
  - For, $1 \leq i \leq n+m$:
    - If $1 \leq i \leq n$ and $d_i \in S$, $\alpha_i = - r_i$
    - Otherwise, $\alpha_i = - d'_i$
- ***Case 2***:
  - Alice sets $T^d_b = \textit{null}$

Data Outsourcing (contd…)

- Similarly, Alice generates the authorization list for all authorized users – T^d
- Note that if T^d_b is null, it is not included in T^d
- Now Alice exports the new data (T^d, E_{pk_a} (d')) to the cloud

Data Access

- Upon a request from Bob, for each data record d, the cloud checks whether there is a token for Bob
- If there is no entry – the cloud simply aborts the request
- If there exists an entry ($T^d{}_b$) for Bob, the cloud proceeds as follows:
  - $E_{pk_b}(d') \leftarrow \{E_{pk_b}(d'_1), \ldots, E_{pk_b}(d'_{n+m})\}$ using $rk_{pk_a \text{-}> pk_b}$
  - For all i, computes $E_{pk_b}(d'_i + \alpha_i) \leftarrow E_{pk_b}(d'_i) + E_{pk_b}(\alpha_i)$
  - Sends $< E_{pk_b}(d'_1 + \alpha_1), \ldots\ldots, E_{pk_b}(d'_{n+m} + \alpha_{n+m}) >$ to Bob

Data Access

- Bob decrypts each entry and gets $d'_i + \alpha_i$ ($1 \le i \le n+m$)
- Note that Bob will successfully decrypt to only those attribute values he is authorized to access
  - That is, $d'_i + \alpha_i = d_i$ only if Bob is authorized to access attribute i.
- Other attribute values will yield a value of zero upon decryption.

User Revocation & Rejoin

- ***User Revocation:*** Whenever Alice wish to revoke user Bob for a data record d, Alice simply asks the cloud to remove T^d_b from T^d

- ***User Rejoin*:** Bob can have following two scenarios for d.
  - Scenario 1: Authorized to the same set (S) of attributes
  - Scenario 2: Authorized to different set of attributes (U)
  - In any case, Alice uses corresponding set (either S or U) and creates T^d_b and sends it to the cloud. Then the cloud adds T^d_b to T^d

Correctness (proof)

- **Theorem**: For any data record d, Bob can only retrieve the set of attributes (*S*) he is authorized to access. On the other hand, if Bob is not an authorized user then he does not get access to d on the cloud (assuming no collusion).

- *Proof*: If Bob is an authorized user, then
  - The final values retrieved by Bob after decryption are $< d'_1 + \alpha_1, \ldots, d'_{n+m} + \alpha_{n+m} >$.
  - For $n+1 \leq i \leq n+m$, $d'_i + \alpha_i = -r_i + r_i = 0$
  - For $1 \leq i \leq n$:
    - If $d_i \in S$, then $d'_i + \alpha_i = d_i + r_i - r_i = d_i$
    - Otherwise, $d'_i + \alpha_i = 0$

Example

Table 1: Sample Patient's Medical data

NAME	AGE	SSN	ROOM	DISEASE
Tom	36	821	63	Miagraine
Cherry	27	163	65	Diabetes
David	45	557	94	Thyroid
Alex	43	923	20	Diabetes
Richard	25	629	34	Skin Cancer
Smith	54	338	55	Cholesterol

- Alice: Data Owner
- Consider Cherry data record as d
- Suppose Bob (Supervisor) is authorized to access <NAME, AGE, ROOM, DISEASE> attribute values of d
- Whereas Charles (Friend) is authorized to access only <NAME, ROOM> attribute values of d

Example (Data Outsource)

- First, Alice masks the data record d and proceeds as follows:
  - Let d' = $<$Cherry + r_1, 27+ r_2, 163+ r_3, 65+ r_4, Diabetes+ r_5, $r_6$$>$, here m=1
  - E_{pk_a} (d') = $<$ E_{pk_a}(Cherry + r_1), E_{pk_a}(27+ r_2), E_{pk_a}(163+ r_3), E_{pk_a}(65+ r_4), E_{pk_a}(Diabetes+ r_5), E_{pk_a}(r_6)$>$
  - $T^d{}_b$ = {Bob, $rk_{pk_a \to pk_b}$, $<$$E_{pk_b}$($-r_1$), E_{pk_b}($-r_2$), E_{pk_b}($-r_3$-163), E_{pk_b}($-r_4$), E_{pk_b}($-r_5$), E_{pk_b}($-r_6$)$>$}
  - $T^d{}_c$ = {Charles, $rk_{pk_a \to pk_c}$, $<$$E_{pk_c}$($-r_1$), E_{pk_c}($-r_2$-27), E_{pk_c}($-r_3$-163), E_{pk_c}($-r_4$), E_{pk_c}($-r_5$-Diabetes), E_{pk_c}($-r_6$)$>$}
  - $T^d = < T^d{}_b , T^d{}_c >$
  - Sends (T^d, E_{pk_a} (d')) to the cloud

Example (Data Access by Bob)

- The cloud computes $< E_{pk_b}(\text{Cherry} + r_1), E_{pk_b}(27+ r_2), E_{pk_b}(163+ r_3), E_{pk_b}(65+ r_4), E_{pk_b}(\text{Diabetes}+ r_5), E_{pk_b}(r_6)>$

Cloud

E_{pk_b}(Cherry)
E_{pk_b}(27)
E_{pk_b}(0)
E_{pk_b}(65)
E_{pk_b}(Diabetes)
E_{pk_b}(0)

Bob decrypts using pr_b

Cherry
27
0
65
Diabetes
0

Example (Data Access by Charles)

- The cloud computes $< E_{pk_c}(\text{Cherry} + r_1), E_{pk_c}(27 + r_2), E_{pk_c}(163 + r_3), E_{pk_c}(65 + r_4), E_{pk_c}(\text{Diabetes} + r_5), E_{pk_c}(r_6) >$

Charles decrypts using pr_c

Cloud

E_{pk_c}(Cherry)
$E_{pk_c}(0)$
$E_{pk_c}(0)$
$E_{pk_c}(65)$
$E_{pk_c}(0)$
$E_{pk_c}(0)$

→

Cherry
0
0
65
0
0

Modified SDS Framework

- Collusion between a user and the cloud might keep the owner's data at risk
- To address this issue, we modify the proposed protocol:

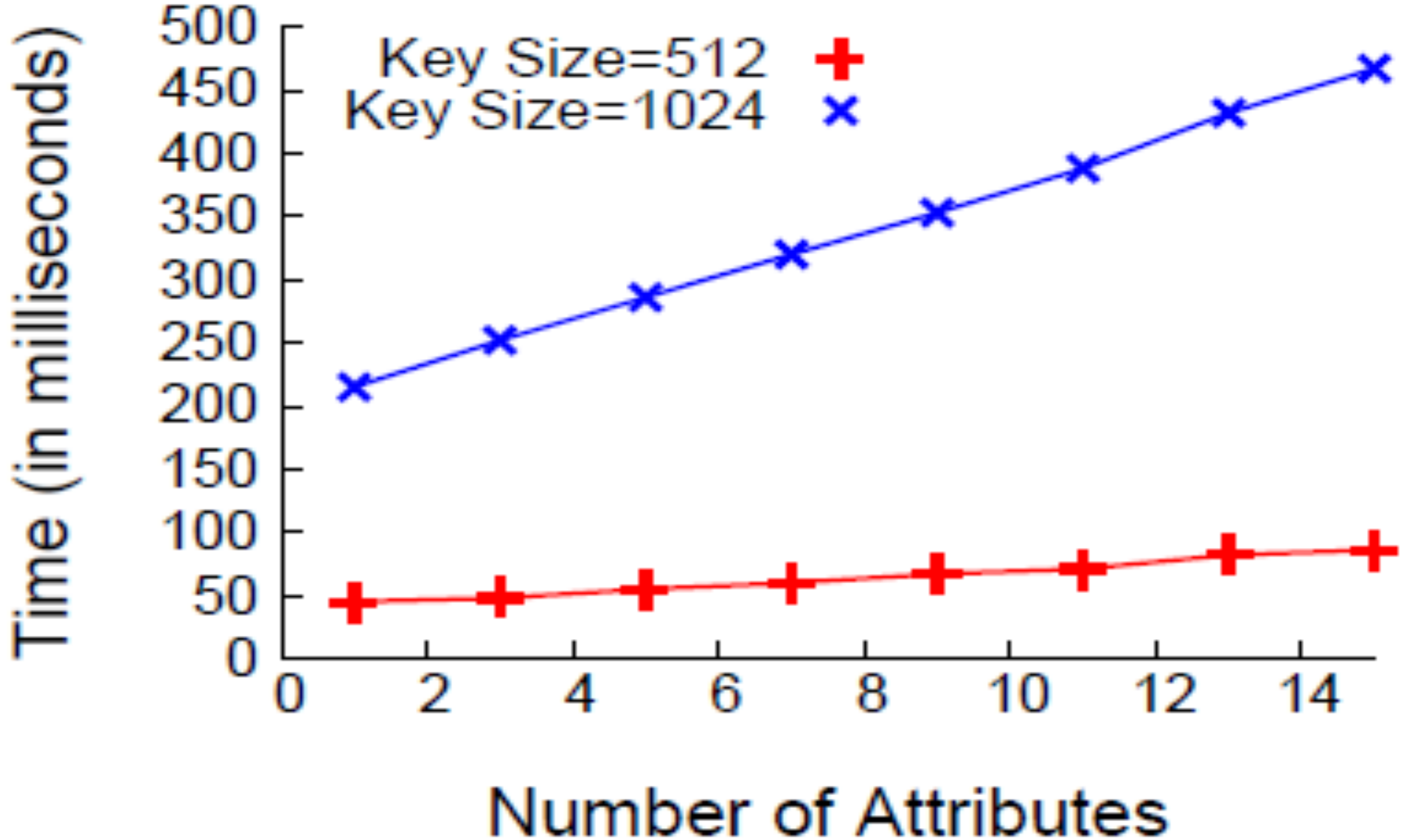
Data Distribution

- Instead of storing the data (T^d, E_{pk_a} (d')) on one cloud, we distribute it to two clouds (Federated cloud).
- Alice will outsource (ID_list, E_{pk_a} (d')) to the primary cloud and (ID_list, T^d) to the secondary cloud
- A collusion between a user and one of the clouds will not provide any meaning full information to either of the parties.

Preliminary Experimental Results

- Platform Description: Linux machine with an Intel 3.0GHz CORE 2 DUO with 3GB memory.
- Randomly generated the number of attributes for a data record d (i.e., n).
- Tested the computational time for Alice for generating a token and encrypting d' based on varying number of attributes for key sizes 512 and 1024 bits.

Alice computational time (m=10)



Conclusion/ Future Work

- Proposed an efficient and secure data sharing (SDS) framework that prevents information leakage when user rejoins the system
- In addition, modified the SDS framework, to prevent the information leakage in the case of collusion between a user and the cloud by distributing the data among two clouds.

- Alternative approach: To distribute private key of user Bob among multiple clouds and Bob.
- Hybrid approach – Key + Data Distribution
- Currently, implementing the SDS framework in a cloud environment

Reference

[1]Y. Yang and Y. Zhang. A generic scheme for secure data sharing in cloud. In *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on, pages 145 –153, sept.* 2011.

Questions ☺