Secure File Sharing on Cloud

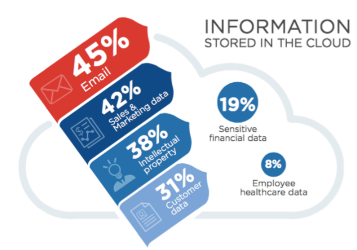
Attribute-Based Access Control and Keyword Search over Encrypted Data

Qiuxiang Dong
Supriya (Team Leader)
Rachita Gupta

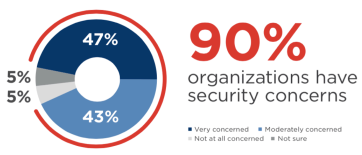
Sunday 27^{th} November, 2016

Cloud Security

- Separation of Data Ownership and Management
- Threats
  - Malicious Outsiders
  - Malicious Insiders
- Data Leakage: e.g., iCloud 2014, Dropbox 2016



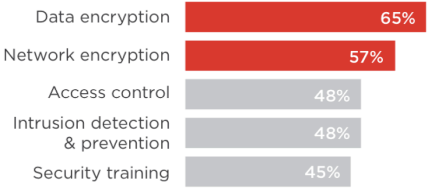
(a) Data Stored on Cloud Servers



(b) Security Concerns

Encrypted Cloud Storage

Data Leakage Prevention $\rightarrow$ Encrypted Cloud Storage



(c) Tech for Data Leakage Prevention

(d) Encrypted Cloud

- Conventional Encryption Schemes: Confidentiality & ~~Utility~~ & ~~Flexibility~~
- New Encryption Schemes: Confidentiality &Utility & Flexibility

Attribute-Based Encryption

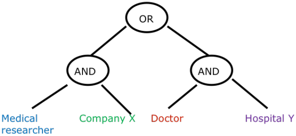
Public Parameters

MSK

Authority

Functionality: output message if attributes satisfy the access formulas

CT: associated with access formulas



Key: associated with attributes

SK

{Nurse, Hospital Y} ✗

{M R, Company X} ✓

Attribute-Based Keyword Search

Public
Parameters

MSK

Authority

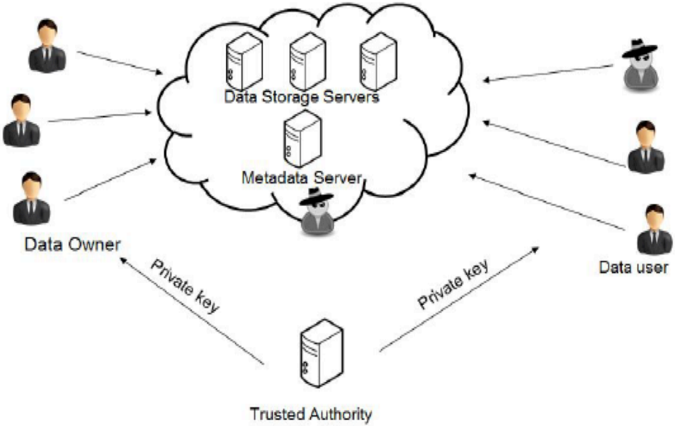
Functionality: could search over the encrypted file if attributes satisfy the access formulas

CT: associated with access formulas

Trapdoor: associated with attributes

SK

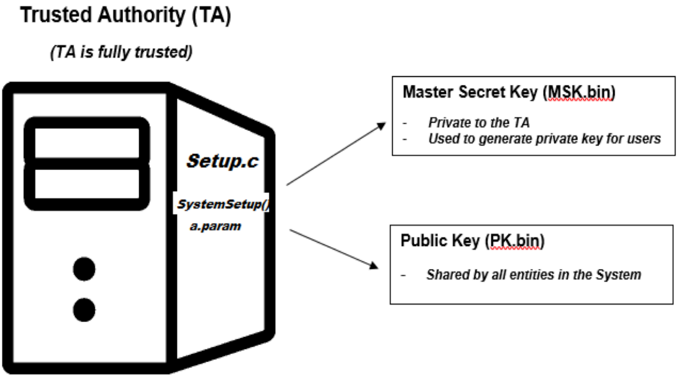
System Architecture



Security Model

- Trusted Authority: fully trusted
- Cloud Server (Data Storage Server and Metadata Server): honest but curious
- Users (Data Owners and Data Users): want to obtain access privilege beyond their private keys

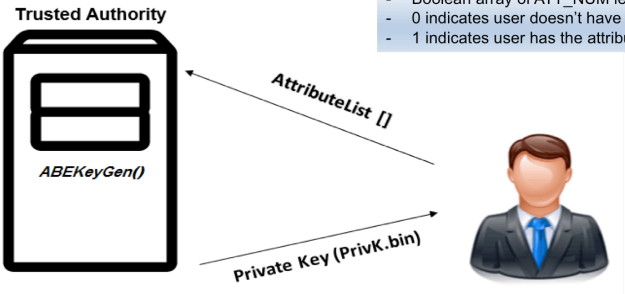
System Setup



Private Key Generation

AttributeList[]

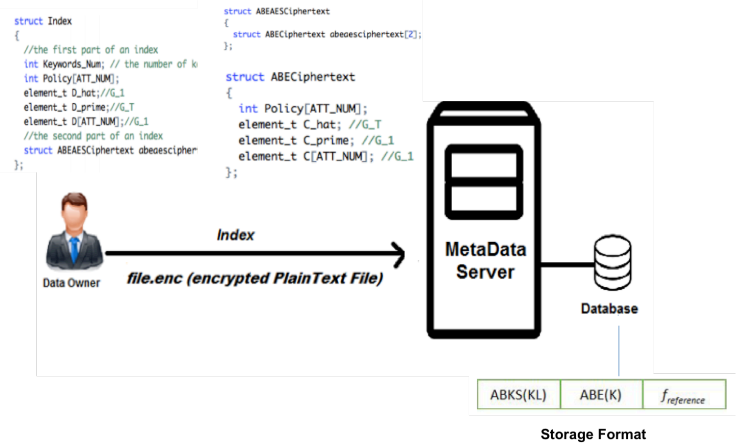
- Defines the user's access privileges
- Boolean array of ATT_NUM length
- 0 indicates user doesn't have that attribute
- 1 indicates user has the attribute




```
struct PrivateKey
{
  element_t K_hat;
  element_t K[ATT_NUM];
  element_t F[ATT_NUM];
};
```

Data owner

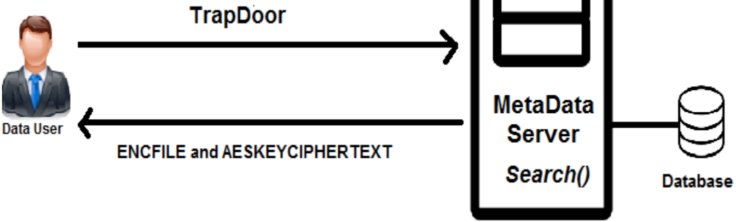
File Upload


struct Index
{
//the first part of an index
int Keywords_Num; // the number of k
int Policy[ATT_NUM];
element_t D_hat;//G_1
element_t D_prime;//G_T
element_t D[ATT_NUM];//G_1
//the second part of an index
struct ABEAESCiphertext abeaescipher
};
struct ABEAESCiphertext
{
struct ABECiphertext abeaesciphertext[2];
};
struct ABECiphertext
{
int Policy[ATT_NUM];
element_t C_hat; //G_T
element_t C_prime; //G_1
element_t C[ATT_NUM]; //G_1
};
Index
Data Owner
file.enc (encrypted PlainText File)
MetaData Server
Database
ABKS(KL)
ABE(K)
$f_{reference}$
Storage Format

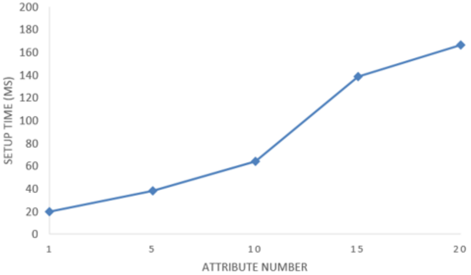
Search Request and Response

```
struct Trapdoor
{
  int AttributeList[ATT_NUM];
  element_t Q_hat; //G1
  element_t Q_prime; //Z_r
  element_t Q[ATT_NUM];//G_1
  element_t Qf[ATT_NUM];//G_1
};
```

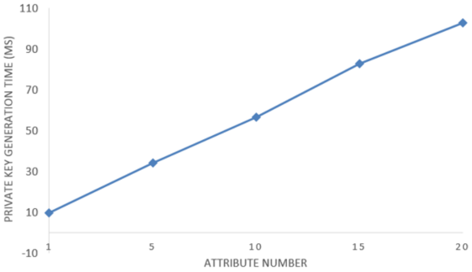
TrapDoor Generation using
PrivK and Keywords



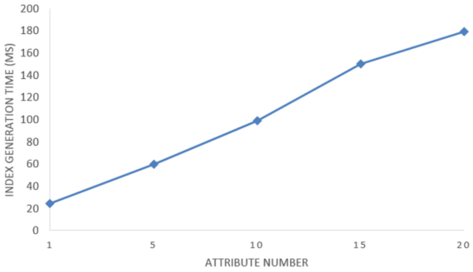
System Setup



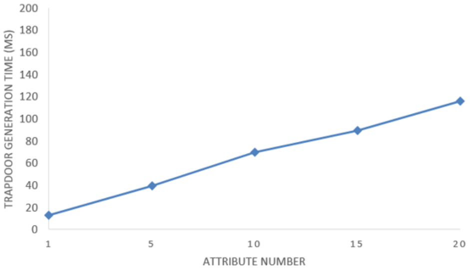
Private Key Generation



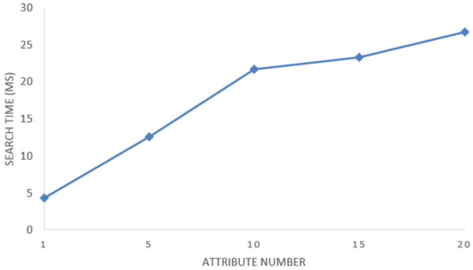
Secure Index Generation



Trapdoor Generation



Search



Future Work

- User Revocation
- Search over numerical values, e.g., range search
- efficiency enhancement, e.g., resource-constrained devices

System Setup

$\textbf{SystemSetup}() \rightarrow (\textbf{PK}, \textbf{MK})$: It defines a bilinear group $\mathbb{G}_1$ of prime order p with a generator g. Thus, a bilinear map is defined as $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$, which has the properties of bilinearity, computability and non-degeneracy. It selects random elements $t_1, \cdots, t_{3n}$. Define a collision-resistant hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$. Let $T_k = g^{t_k}$ for each $k \in \{1, \cdots, 3n\}$ such that for $1 \leq i \leq n$, T_i are referred to as positive attributes, T_{n+i} are for negative ones, and T_{2n+i} are thought of as don't care. Let $Y = e(g, g)^y$. The public key is $PK = (e, g, Y, T_1, \cdots, T_{3n})$ and the master key is $MK = (y, t_1, \cdots, t_{3n})$.

Data Structure

The Public Key data structure in C:

```c
struct PublicKey
{
  element_t g;//G_1
  element_t Y;//G_T
  element_t T[PARAM_NUM];//G_1
};
```

```c
struct MasterKey
{
  element_t y;//Z_r
  element_t t[PARAM_NUM];//Z_r
};
```

Key Generation

$\mathbf{ABEKeyGen(intAttributeList[], structPrivateKey * PrivK)}$: For every attribute, TA selects random number r_i from $\mathbb{Z}_p$ hence $r = \sum_{i=1}^{n} r_i$. K_hat is set as g^{y-r}. For $AttributeList[i] = 1$, set $K_i = g^{\frac{r_i}{t_i}}$ and $K_i = g^{\frac{r_i}{t_{n+i}}}$ otherwise. Finally, let F_i be $g^{\frac{r_i}{t_{2n+i}}}$. The secret key $PrivK = (K\_hat, \{K_i, F_i\})_{i \in [1, \cdots, ATT\_NUM]}$

Data Structure in C:

```
struct PrivateKey
{
  element_t K_hat;
  element_t K[ATT_NUM];
  element_t F[ATT_NUM];
};
```

Data Upload

$\textbf{SecureIndexGeneration}(\textbf{int Policy}[\,], \ \textbf{char} * \ \textbf{keywords}[\,])$:
Generate a random $s \in \mathbb{Z}_p$, set $D\_hat = g^s$, $D\_prime = Y^s$. Given the policy $\textbf{Policy} = \{\textbf{Policy}[\textbf{i}]_{\textbf{i} \in [\textbf{1}, \cdots, \textbf{ATT\_NUM}]}\}$, for each $i \in [1, \cdots, ATT\_NUM]$. If $Policy[i] = 1$, $D_i = T_i^s$, if $Policy[i] = 2$ $D_i = T_{n+i}^s$, else $D_i = T_{2n+i}^s$. The generated keyword ciphertext is (D_hat, D_prime, D).

Data Structure in C:

```
struct Index
{
  //the first part of an index
  int Keywords_Num; // the number of keywords in the file
  int Policy[ATT_NUM];
  element_t D_hat;//G_1
  element_t D_prime;//G_T
  element_t D[ATT_NUM];//G_1
  //the second part of an index
  struct ABEAESCiphertext abeaesciphertext;
};
```

Data Upload

$\textbf{ABEEncrypt}(\textbf{int Policy}[\ ],\ \textbf{element\_t plaintext},$ $\textbf{struct ABECiphertext}* \textbf{ abeciphertext})$: Generate a random $s \in \mathbb{Z}_p$, set $C\_prime = g^s$, $C\_hat = Y^s$. Given the policy $\textbf{Policy} = \{\textbf{Policy}[\textbf{i}]_{\textbf{i} \in [1,\cdots,\textbf{ATT\_NUM}]}\}$, for each $i \in [1, \cdots, ATT\_NUM]$. If $Policy[i] = 1$, $C_i = T_i^s$, if $Policy[i] = 2$ $C_i = T_{n+i}^s$, else $C_i = T_{2n+i}^s$. The generated keyword ciphertext is (C_hat, C_prime, C).

Data Structure in C:

```
struct ABEAESCiphertext
{
  struct ABECiphertext abeaesciphertext[2];
};

struct ABECiphertext
{
  int Policy[ATT_NUM];
  element_t C_hat; //G_T
  element_t C_prime; //G_1
  element_t C[ATT_NUM]; //G_1
};
```

Search Request

TrapdoorGeneration(struct PrivateKey PrivK, char ∗ keyword):
Generate a random $u \in \mathbb{Z}_p$. Let $Q\_hat = K\_hat^u$ and $Q_i = K_i^u$, $Qf_i = F_i^u$. For the same i', $Q_{i'} = K_{i'}^{H(w')\cdot u}$, where w' is the keyword of interest and $Qf_{i'} = F_{i'}^{H(w')\cdot u}$. The trapdoor is (Q_hat, Q_prime, Q, Qf).

Data Structure in C:

```
struct Trapdoor
{
  int AttributeList[ATT_NUM];
  element_t Q_hat; //G1
  element_t Q_prime; //Z_r
  element_t Q[ATT_NUM];//G_1
  element_t Qf[ATT_NUM];//G_1
};
```

Search Response

SearchIndex(struct Index ind, struct Trapdoor trapdoor): First compare the Policy[ ] in the index and the AttributeList[ ] in trapdoor. If data user's AttributeList[ ] does not satisfy data owner's Policy[ ], then skip this index, since the data user does not have the search and access privilege of this file. Or else, the following algorithm is run: check whether $D\_prime^{Q\_prime} = e(D\_hat, Q\_hat) \cdot \Pi_{i=1}^{ATT\_NUM} e(D_i, Q_i^*)$ holds, where $Q_i^* = Q_i$ if data user has that attribute, $Q_i^* = Qf_i$ otherwise.

System Demonstration

https://www.youtube.com/watch?v=t7rhjdX03OY

http:

//gitlab.thothlab.org/rgupta36/SecureFileStorageCloud

Thanks!