Basic Temporal Access Control without User Revocation and proxy for Cloud Data

Ayan Das*, Sushmita Ruj†
* R.C. Bose Center for Cryptology and Security, Indian Statistical Institute, Kolkata, India –ayandas.infy@gmail.com
† R.C. Bose Center for Cryptology and Security, Indian Statistical Institute, Kolkata, India – sush@isical.ac.in

***Abstract*—Here We propose a basic temporal access control scheme to protect and selectively access data in clouds. This is our first scheme on temporal access control. Our this scheme encrypts and stores data in clouds in such a way that only authorized users are able to decrypt it within a specified time period. This scheme is implemented Without any proxy for decryption and without the facility of revocation. Later on we will see more complicated schemes with extra facilities and extra ramdomization on this basic scheme.**

Keywords: Access Control, Cloud Storage, CP-ABE, Temporal Access Control, Revocation

A. *Framework*

A brief description of each of these algorithms is as follows:

- Setup$(1^\kappa) \to (MK, PK)$: Takes a security parameter κ as input, outputs the master secret key MK and the public key PK.
- KeyGen$(MK, u_k, \mathcal{L}) \to SK_\mathcal{L}$: Takes the user ID u_k, the access privilege $\mathcal{L}$, and MK as input, outputs the users private key $SK_\mathcal{L}$.
- Encrypt$(PK, M, \mathcal{P}) \to (\mathcal{H}_\mathcal{P}, C')$: Takes a comparable access policy $\mathcal{P}$, public key PK and message M as input, outputs the ciphertext header $\mathcal{H}_\mathcal{P}$ and ciphertext C'.
- Decrypt$(SK_\mathcal{L}, \mathcal{H}_\mathcal{P}, C') \to M$. Takes the users private key $SK_\mathcal{L}$, Ciphertext header $\mathcal{H}_\mathcal{P}$, and the ciphertext C' as input, outputs the message M.

I. Construction

We provide detailed implementations of the algorithms mentioned in section -A and present our novel construction to enforce user revocation.

- *Setup*$(1^\kappa) \to (MK, PK)$
  This is run by the system manager. Let κ be the security parameter. The setup algorithm takes κ (which depends on the application) and outputs the master secret key MK and public key PK. We use bilinear map $\mathbb{S} = (\mathbb{G}, \mathbb{G}_T, e(\cdot,\cdot))$ of composite order $n = s'n'$ with two subgroups $\mathbb{G}_{s'}$ and $\mathbb{G}_{n'}$ of $\mathbb{G}$. Random generators $w \in \mathbb{G}$, $g \in \mathbb{G}_{s'}$, and $\varphi, \bar{\varphi} \in \mathbb{G}_{n'}$ are chosen. Two random numbers $\lambda, \mu \in \mathbb{Z}_n^*$ are selected, to implement forward and backward derivation functions. By Proposition 1, we have $e(g,\varphi) = e(g,\bar{\varphi}) = 1$, but $e(g,w) \neq 1$. A hash function $H : \{0,1\}^* \to \mathbb{G}$ is used to map an attribute to a binary string which represents a random group element. A randomly generated polynomial P of degree c (the maximum number of revoked users) over $\mathbb{Z}_n$ will be employed to enforce revocation. Next, this algorithm chooses two random exponents $\alpha, \beta \in \mathbb{Z}_n^*$ and outputs the public key
$$PK = (\mathbb{S}, n, g, h, \zeta, \eta, w, \varphi, \bar{\varphi}, \lambda, \mu, H(\cdot)),$$
  where,
$$h = w^\beta, \zeta = e(g,w)^\alpha, \eta = g^{\frac{1}{\beta}},$$
  and the master key
$$MK = (g^\alpha, \beta, n', P).$$
- *KeyGen*$(MK, u_k, \mathcal{L}) \to SK_\mathcal{L}$

  Given a user with ID u_k[1] and access structure $\mathcal{L}$ on a set of attributes $S = \{A_t\} \subseteq \mathcal{A}$, this algorithm chooses a unique τ_k for each user u_k. Assume that the user u_k is assigned a temporal attribute $A_t \in \mathcal{L}$ with the constraint $A_t[t_a, t_b]$ and non-temporal attribute $B_t \in \mathcal{L}$. This algorithm chooses a random $r \in \mathbb{Z}$ and sets the user's attribute key as
$$(D_t, D^{'}_{t_a}, \bar{D}^{'}_{t_b}, D^{''}_{t},)_{A_t[t_a,t_b]\in\mathcal{L}} \text{and}$$
$$(DB_t)_{B_t\in\mathcal{L}}$$
  where,
$$\begin{aligned} D_t &= g^{\tau_k} H(A_t)^r, \\ D^{'}_{t_a} &= (v_{t_a})^r, \\ \bar{D}^{'}_{t_b} &= (\bar{v}_{t_b})^r, \\ D^{''}_{t} &= w^r, \\ DB_t &= g^{\tau_k} H(B_t)^r, \end{aligned}$$
  and,
$$v_{t_a} = \varphi^{\lambda^{t_a}}, \bar{v}_{t_b} = \bar{\varphi}^{\mu^{Z-t_b}} \in \mathbb{G}_{n'},$$
  Finally, this algorithm outputs the user's private key
$$\begin{aligned} SK_\mathcal{L} = (D = g^{\frac{(\alpha+\tau_k)}{\beta}}, & \\ \{(D_t, D^{'}_{t_a}, \bar{D}^{'}_{t_b}, D^{''}_{t})\}_{A_t[t_a,t_b]\in\mathcal{L}}, & \\ \{(DB_t)\}_{B_t\in\mathcal{L}}). & \end{aligned}$$
- *Encrypt*$(PK, \mathcal{P}, M) \to (\mathcal{H}_\mathcal{P}, C')$
  Let $\mathcal{T}$ be the access tree for the access policy $\mathcal{P}$. Let s be a secret in $\mathbb{Z}_N$ for the access tree $\mathcal{T}$, and x, y are chosen such that $x + y = q_{A_t}(0)$, which is the secret share at leaf node corresponding to attribute A_t. Please refer to Section 4, [**?**] for a detailed meaning of $q_{A_t}(0)$. Given an access tree $\mathcal{T}$, the ciphertext is composed of a ciphertext header,

[1]We assume that each user ID $u_k \in \mathbb{Z}_l$ where $l = min(s, p', q')$, so that, for every pair $(i,j), i \neq j, u_i - u_j \in \mathbb{Z}_n^*$. As a result, its inverse $(u_i - u_j)^{-1}$, which is needed in the *ProxyRekey* phase will exist and the secret can be successfully recovered.

$$\mathcal{H}_{\mathcal{P}} = (\mathcal{T}, C = h^s,\\ \{(\bar{E}_{t_i}, E'_{t_i}, E_{t_j}, E'_{t_j})\}_{A_t[t_i,t_j]\in\mathcal{P}},\\ \{(EB'_t)\}_{B_t\in\mathcal{P}}).$$

and a ciphertext $C' = Me(g,w)^{s\alpha}$, where,

$(\bar{E}_{t_i}, E'_{t_i}, E_{t_j}, E'_{t_j}) = ((\bar{v}_{t_i}w)^x, H(A_t)^x, (v_{t_j}w)^y, H(A_t)^y)$ and $(EB'_t, \bar{EB}_t) = (H(B_t)^{q_{B_t}(0)}, w^{q_{B_t}(0)})$.

- *Decrypt*$(SK_{\mathcal{L}}, \mathcal{L}') \rightarrow M$

Given the private key $SK_{\mathcal{L}}$ and a specified $\mathcal{L}'$, this algorithm checks whether, for every $A_t[t_a,t_b] \in \mathcal{L}$ and $A_t[t_i,t_j] \in \mathcal{L}'$, $t_a \le t_j$ and $t_b \ge t_i$ is true for each attribute $A_t \in \mathcal{L}'$.
If true, the user computes,

$$D'_{t_j} \leftarrow f_{t_a \le t_j}(D'_{t_a}) \cdot D''_t = (v_{t_j}w)^r$$
$$\bar{D}'_{t_i} \leftarrow \bar{f}_{t_b \ge t_i}(\bar{D}'_{t_b}) \cdot D''_t = (\bar{v}_{t_i}w)^r$$

Finally, it outputs $\widetilde{SK}_{\mathcal{L}'}$ as the derivation key for $\mathcal{L}'$.
On receiving the private key $\widetilde{SK}_{\mathcal{L}'}$, and the ciphertext header $\mathcal{H}_{\mathcal{P}}$, this algorithm checks whether each range attribute $A_t[t_i,t_j] \in \mathcal{L}'$ is consistent with $A_t[t_i,t_j] \in \mathcal{P}$. If true, the secret share $q_{A_t}(0)$ over $\mathbb{G}_T$ is reconstructed

$$\begin{aligned} F_1 &\longleftarrow \frac{e(D_t, E_{t_j})}{e(D'_{t_j}, E'_{t_j})} \\ &= \frac{e(g^{\tau_k}H(A_t)^r, (v_{t_j}w)^y)}{e((v_{t_j}w)^r, H(A_t)^y)} \\ &= \frac{e(g^{\tau_k}, (v_{t_j}w)^y)\cdot e(H(A_t)^r, (v_{t_j}w)^y)}{e((v_{t_j}w)^r, H(A_t)^y)} \\ &= e(g^{\tau_k}, (v_{t_j}w)^y) \\ &= e(g^{\tau_k}, v_{t_j}^y)\cdot e(g^{\tau_k}, w^y) \\ &= e(g^{\tau_k}, w)^y \end{aligned}$$

Similarly,

$$\begin{aligned} F_2 &\longleftarrow \frac{e(D_t, \bar{E}_{t_i})}{e(\bar{D}'{}_{t_i}, E'_{t_i})} \\ &= \frac{e(g^{\tau_k}H(A_t)^r, (\bar{v}_{t_i}w)^x)}{e((\bar{v}_{t_i}w)^r, H(A_t)^x)} \\ &= \frac{e(g^{\tau_k}, (\bar{v}_{t_i}w)^x)\cdot e(H(A_t)^r, (\bar{v}_{t_i}w)^x)}{e((\bar{v}_{t_i}w)^r, H(A_t)^x)} \\ &= e(g^{\tau_k}, (\bar{v}_{t_i}w)^x) \\ &= e(g^{\tau_k}, \bar{v}_{t_i}^x)\cdot e(g^{\tau_k}, w^x) \\ &= e(g^{\tau_k}, w)^x \end{aligned}$$

For $B_t \in \mathcal{L}'$ is consistent with $B_t \in \mathcal{P}$, then the secret share $q_{B_t}(0)$ of s over $\mathbb{G}_T$ is reconstructed as,

$$\begin{aligned} F_3 &\longleftarrow \frac{e(DB_t, \bar{EB}_t)}{e(D''{}_t, EB'_t)} \\ &= \frac{e(g^{\tau_k}H(B_t)^r, (w^{q_{B_t}(0)})}{e(w^r, H(B_t)^{q_{B_t}(0)})} \\ &= \frac{e(g^{\tau_k}, w^{q_{B_t}(0)})\cdot e(H(B_t)^r, w^{q_{B_t}(0)})}{e(w^r, H(B_t)^{q_{B_t}(0)})} \\ &= e(g^{\tau_k}, w)^{q_{B_t}(0)} \end{aligned}$$

$$F_t = F_1 \cdot F_2 = e(g^{\tau_k}, w)^{q_{A_t}(0)}$$

where, $e(g^{\tau_k}, v_{t_j}^y) = e(g^{\tau_k}, \bar{v}_{t_i}^x) = 1$ because $g^{\tau_k} \in \mathbb{G}_{s'}$ and $v_{t_j}^y, \bar{v}_{t_i}^x \in \mathbb{G}_{n'}$. Next, the value of $T = e(g^{\tau_k}, w)^s$ is computed from $\{e(g^{\tau_k}, w)^{q_{A_i}(0)}\}_{A_i\in\mathcal{P}}$ and $\{e(g^{\tau_k}, w)^{q_{B_i}(0)}\}_{B_i\in\mathcal{P}}$ by using the recursive DecryptNode algorithm described in Bethencourt et al. [**?**]. Finally, the new ciphertext header $\widetilde{\mathcal{H}}_{\mathcal{P}} = (C,T)$ is returned.

Once the ciphertext header $\widetilde{\mathcal{H}}_{\mathcal{P}} = (C,T) = (w^{\beta s}, e(g^{\tau_k}, w)^s)$ is formed, the secret δ is used to compute $D' = D \cdot \eta = g^{\frac{(\alpha+\tau_k)}{\beta}} g^{\frac{1`}{\beta}} = g^{\frac{(\alpha+\tau_k)}{\beta}}$.
Let $ek = \dfrac{e(C,D')}{T} = \dfrac{e(g^{\frac{(\alpha+\tau_k)}{\beta}}, w^{\beta s})}{e(g^{\tau_k}, w)^s} = e(g^\alpha, w)^s$.
Finally, the plaintext message is computed by
$M = C'/ek$.