OAuth and ABE based Authorization in Semi-Trusted Cloud Computing

Anuchart Tassanaviboon and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
CANADA
<http://comsec.uwaterloo.ca/~atassana, ggong>
Outline

- **Security** requirements in cloud environment
- Solutions & **challenges** in semi-trusted cloud computing (STCC)
- **Review** of OAuth and ABE based schemes
- **AAuth**: A new authenticated **authorization** scheme for securing STCC
- **Performance** evaluation and simulation
- Conclusions and remarks
Vulnerabilities

- **Web-interface flaws**, XML signature wrapping, legacy same origin policy, unsecured browser authentication
- **Leak virtual isolation**, side/covert channel, cross-tenant data access
- **Image insanity**, malicious/illegal images
- Limited **network control**, under-provisioning, limited QoS, new form of DoS
- **Weak access control**, weak credentials, weak tokens, coarse authorization
- **Lack of standards**, APIs, inter-operations
**Typical** models
- Centralized server
- Client-server: Kerberos/Active Directory
- HTTP: OpenID/OAuth

**Cloud Computing?**

**Cloud** problems and challenges
- Trust boundary is expanded to CSPs
- CSPs are untrusted or semi-trusted
- A shared trusted domain doesn’t present
- A single trusted domain is unscalable
An authorizer arbitrarily grants accesses.

Cloud servers reveal sensitive data.

Cloud servers disobey the access policies.

Weak tokens cause fabrication, replay attacks, etc.

Lock-in vendors.
Kerberos

Key Distribution Center (KDC)

Owner

Client

Resource Server (RS)

Authentication Server (AS)

Database

Ticket Granting Server (TGS)

1. \((\text{ID}_{\text{CLI}}, \text{ID}_{\text{TGS}}, \text{LT})\)

2. \((\text{ID}_{\text{TGS}}, \text{TS}, \text{LT}, \text{SK}_{\text{TGS}}) K_{\text{Owner}} || \{\text{TGT}\} K_{\text{TGS}}\)

3. \((\text{ID}_{\text{RS}}, \text{LT}) || \{\text{Auth}\} \text{SK}_{\text{TGS}} || \{\text{TGT}\} K_{\text{TGS}}\)

4. \((\text{ID}_{\text{RS}}, \text{TS}, \text{LT}, \text{SK}_{\text{RS}}) \text{SK}_{\text{TGS}} || \{\text{ST}\} K_{\text{RS}}\)

5. \{\text{Auth}\} \text{SK}_{\text{RS}} || \{\text{ST}\} K_{\text{RS}}\)

6. Resource

**Variables:**

- \(\text{ID}_{\text{CLI}}\): Client ID
- \(\text{ID}_{\text{RS}}\): Resource Server ID
- \(\text{ID}_{\text{TGS}}\): TGS ID
- \(\text{TS}\): Timestamp
- \(\text{LT}\): Lifetime
- \(K_A\): A’s Secret key in KDC
- \(\text{SK}_{\text{B}}\): CLI – B Session key between
- \(\text{TGT} = (\text{ID}_{\text{CLI}}, \text{ID}_{\text{TGS}}, \text{TS}, \text{LT}, \text{SK}_{\text{TGS}})\)
- \(\text{Auth} = (\text{ID}_{\text{CLI}}, \text{TS})\)
- \(\text{ST} = (\text{ID}_{\text{CLI}}, \text{ID}_{\text{SR}}, \text{TS}, \text{LT}, \text{SK}_{\text{RS}}) K_{\text{Owner}} = \text{Hash}(\text{password}, \text{salt})\)
OAuth

1. Client ID, Redirection URI
3. Authorization Code

2. Owner Authentication, Authorization Decision

Owner User-Agent (Browser)

OAuth Provider (Authorization Server)

1. 
3. Authorization Code

Third-party Client (Web-Application)

Resource Server (HTTP Service)

5. Access Token, Expiry date, Scope

6. Access Token
7. Resource

OAuth Provider (Authorization Server)

Owner User-Agent (Browser)

Third-party Client (Web-Application)

Resource Server (HTTP Service)
ABE (Sahai-Waters 05), KP-ABE (Goyal, et. al. 06), and CP-ABE (Bethencourt-Sahai-Waters 07). In order to adapt to our scheme, a modified CP-ABE will be introduced later.
**Design Goals**

- **Data owners** contribute to token generation.
- Data is encrypted in an **end-to-end** fashion.
- **Policies** are enforced by cryptographic functions.
- **Token** knowledge is distributed among CSPs for reducing risks.
- Scheme is integrated with existing standards and cloud entities.
System Model

- **Data owner** (*O*): (owners for short) entities, i.e., end-users or software applications, who have resource ownerships and the right to grant access to protected data.

- **Cloud server** (*S*): (servers for short) cloud-storage or cloud-database providers that host protected data and provide basic data-services, i.e., read, write, and delete.

- **Consumers** (*C*): web or traditional applications service provider (ASPs) that use owners’ data to provide services to the owners.

- **Authority** (*AA*): trusted organizations or agencies who legitimately define descriptive attributes to eligible consumers.

- **Authorizer** (*AZ*): the server who runs AAuth protocol, then issues ABE-based tokens to eligible consumers.
System Model (Example)

Authority
'authority.org'

Consumer
'print.com'

Authorizer
'mail.net'

Server
'photos.com'

Owner
'Jane'

# Confined attributes
AND [OWNER=Jane@photos.net]
AND [SEC-CLASS=3]
AND [PERMIS=r]
AND [TIMESLOT=2011/06/27/13/**]
AND # Descriptive attributes
[(OWNERe@mail.net=Jane@mail.net) OR
 (NAME@authority.org=printer.com) AND
 (SERVICE@authority.org = print) AND
 (LOCAT@authority.org = canada) OR
 (TRUST-LEV@authority.org = 3)].
Pre-conditions and Adversary Model

- **Servers** are trusted to provide data-services properly but may be curious about sensitive information and prone to reveal data to ineligible parties.
- **The authorizer** may disobey owners’ orders to issue tokens, or issue any arbitrary tokens to its conspirators.
- **Consumers** may try to get unauthorized files from honest servers by fabricating tokens to obtain unauthorized accesses, resubmitting previous tokens (replay attacks).
- **Internet users** may launch general network attacks on encrypted data or tokens. However, we assume that the communications among CSPs are secure and authentic under SSL/TLS secure channel.
- Adversaries do not have enough computing power to break cryptographic primitives.
AAuth Components

Defined Attributes

FILE-LOC = URI
OWNER = ownerId
PERMIS = ⟨r|w⟩
SEC-CLASS = ⟨1 – 5⟩
TIMESLOT = yyyy/mm/dd/hh/nn

Access Policy \( \mathcal{A} \)

\[ \mathcal{A} = \text{[FILE-LOC] AND [OWNER] AND [SEC-CLASS] AND [PERMIS] AND [TIMESLOT] AND [(OWNER@AUTHZ) OR (Descriptive Boolean Algebra)]} \]
AAuth Components (Cont.): Access Tree \( \tau \)

\[ s = q_R(0) = q_{FL}(0) + q_{OW}(0) + q_{SC}(0) + q_{PM}(0) + q_{TS}(0) + q_{DA}(0) \]

\( q_{DA}(7) = q_{ow@AZ}(0) \)

\( q_{DA}(8) = q_{AA}(0) \)

R: Root  FL: File Location  OW: Owner  SC: Security Class  PM: Permission  TS: Timeslot  DA: Descriptive Attributes  OW@AZ: Owner at Authorizer  AA: Authority

(6,6) threshold,  \( q_R(0) = s \)

(1,2) threshold  

\[ q_{R}(6) = q_{DA}(0) \]

\[ q_{DA}(0) = q_{ow@AZ}(0) = q_{AA}(0) \]
AAuth Components (Cont.): Archive File

- File Desc
- Access Policy
- Encry. Meth.
- Encry. Key
- Integ. Meth.
- Integ. Key

{T}Header\_ABE\{\}Data File\_KE\{\}Tail

Access Policy  Integ. Tag
Modified CP-ABE

**Setup**(*k*)

**Authorizer**

- **System parameters**
  - Bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.
  - Generator $g$ of group $\mathbb{G}_1$.
  - Hash function $H : \{0, 1\}^* \rightarrow \mathbb{G}_1$.

- Randomly selects $\beta \in \mathbb{Z}_p$.
- Master Secret Key: $MSK = \langle \beta \rangle$.
- Master Public Key: $MPK = \langle \mathbb{G}_1, g, h = g^\beta, f = g^{1/\beta} \rangle$.

**Owner**

- Randomly selects $\alpha \in \mathbb{Z}_p$.
- Owner Secret Key: $OSK = \langle g^\alpha \rangle$.
- Owner Public Key: $OPK = \langle e(g, g)^\alpha \rangle$. 
Modified CP-ABE: $Encrypt(\text{MPK}, m, \tau)$

- Randomly selects $s \in \mathbb{Z}_p$.
- Construct access tree $\tau$ according to $q_R(0) = s$ and an access policy $A$.
- Let $Y$ be the leaf nodes in $\tau$:

  **Ciphertext:** $CT = \langle \tau, \tilde{C} = m \cdot e(g, g)^{\alpha s}, C = h^s, \forall y \in Y: C_y = g^{q_y(0)}, C'_y = H(\text{att}(y))^{q_y(0)} \rangle$. 
Modified CP-ABE: $\text{KeyGen}(\text{MSK}, \text{OSK}, \omega)$

Assume that an attribute set $\omega = \omega' \cup \omega''$ where $\omega' =$ confined attributes, and $\omega'' =$ descriptive attribute.

- **Authorizer:** $r \in R \mathbb{Z}_p$, and selects a set $\{ r_i \in R \mathbb{Z}_p \mid i \in \omega' \}$, i.e., responds for confined attributes.
- **Authority:** selects $\{ r_j \in R \mathbb{Z}_p \mid j \in \omega'' \}$, descriptive attributes
- **Owner:** $a \in R \mathbb{Z}_p$.

With ElGamal-like masking, the authorizer, the authority, and the owner jointly compute a private key for the consumer

**Private key:**

$$SK = \langle D = g^{(\alpha + ra)/\beta}, D_k = g^{ra} \cdot H(k)^{r_k}, D'_k = g^{r_k}, \forall k \in \omega \rangle.$$
Modified CP-ABE: \( \text{Delegate}(SK, \tilde{\omega}) \)

- Given a secret key \( SK \) for an attribute set \( \omega \).
- Let \( \tilde{\omega} \supseteq \omega \) denote a new attribute set.
- Random value \( \tilde{r} \) and random set \( \{ \tilde{r}_l \mid \forall l \in \tilde{\omega} \} \).
- A consumer creates a new private key \( \tilde{SK} \) for the attribute set \( \tilde{\omega} \):
  \[
  \tilde{SK} = \langle \tilde{D} = D \cdot f^{\tilde{r}}, \forall l \in \tilde{\omega} : \tilde{D}_l = D_l \cdot g^{\tilde{r}} \cdot H(l)^{\tilde{r}_l}, \tilde{D}'_l = D'_l \cdot g^{\tilde{r}_l} \rangle.
  \]
Modified CP-ABE: \( Decrypt(CT, SK) \)

- Recursively computes from the root node \( R \) of access tree \( \tau \) by using node algorithm \( DecryptNode(CT, SK, x) \):

\[
F_R = DecryptNode(CT, SK, R) = e(g, g)^{ra \cdot q_R(0)} = e(g, g)^{ras}
\]

- If the tree \( \tau \) is satisfied by \( \omega \) then decryption can be computed by:

\[
Decrypt(CT, SK) = \tilde{C} / (e(C, D) / F_R) \tilde{C} / (e(h^s, g^{(\alpha + ra) / \beta}) / e(g, g)^{ras}) = m
\]
A Diagram of $\text{DecryptNode}(CT, SK, x)$

$$F_R = e(g, g)^{ras}$$

$$F_x = \prod_{z \in \omega_x} F^\Delta_{k, \hat{\omega}_x}(0)$$

$$F_x = \frac{e(D_k, C_x)}{e(D'_k, C'_x)}$$
AAuth in a Nutshell

- **AAuth** extends OAuth to a cryptographic token system that its ABE-token is a private key associated with a set of attributes, and its protected resource (data file) is encrypted with an access policy constructed from an access structure over a public key.

- Due to the inapplicability of a single authority in large scale systems, our scheme divides attribute universe in two disjointed sets: **confined attributes** defined by owners to limit the lifetime and scope of tokens, and descriptive attributes defined by authority(s) to certify the characteristic of consumers.

- To allow **owners to contribute** to private-key generation, we separate the master key of CP-ABE to two parts: $g^\alpha$ for owners and $\beta$ for a authorizer, then add another level of ElGamal-liked masks to conceal the master keys from each other during key generation.

- Resource servers have no affiliation in authorization but provide basic data-services: read, write, delete.
AAuth in a Nutshell (cont.)

- AAuth consists of **three off-line procedures**:
  - **Setup**: authorizer chooses a security parameter $k$ and run CP-ABE algorithm $\text{Setup}(k)$.
  - **File encapsulation**: the owner encrypts and encapsulate data files into archives file and send to the sever by converting $\mathbb{A}$ to an access tree $\tau$ and using the CP-ABE Enc algorithm.
  - **File decapsulation**: consumer verifies the integrity of the archive file, then performs decapsulation of an archive file.
- **four on-line protocols**: service request, token request, file access, and timeslot synchronization, which will be presented in details.
- Optionally, AAuth can provide key delegation, policy change, and data update.
AAuth: Service Request Protocol

1. REQ-PRT

2. REQ-POL

3. [A]s

4. RED
AAuth: Token Request Protocol

1. RED = \([ID_C, RED-URI]_C, [\mathcal{A}]_S\)
2. HTTP Form
3. Login/password
4. REQ-DES1
5. \([\{\hat{D}_j}\}]_AA
6. \{\hat{D}_j\}, \{\{D'_j\}\}_AZ, \{\{\hat{D}_j\}\}_AA, g^r
7. \(g^{(\alpha + ra)}\)
8. RED
9. \{\hat{D}_j\}, \{\{D'_j\}\}_AZ, \{D_j\}
10. REQ-DES2
11. \{D'_j\}

1. RED = RED([ID_C, RED-URI]_C, [\mathcal{A}]_S).

Tassana-Gong (University of Waterloo) AAuth
AAuth: File Access Protocol

Server S

1. REQ-FILE

2. Chall

3. Resp

4. Archive

Consumer C
## AAuth: Time Slot Synchronization

<table>
<thead>
<tr>
<th>Timeslot</th>
<th>0</th>
<th>1</th>
<th>( \ldots )</th>
<th>( n - 1 )</th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random value, ( \tilde{s} )</td>
<td>( \tilde{s}(0) )</td>
<td>( \tilde{s}(1) )</td>
<td>( \ldots )</td>
<td>( \tilde{s}(n - 1) )</td>
<td>( \tilde{s}(n) )</td>
</tr>
<tr>
<td>Share, ( q_{TS}(0) )</td>
<td>( q_{TS}(0, 0) )</td>
<td>( q_{TS}(0, 1) = q_{TS}(0, 0) + \tilde{s}(1) )</td>
<td>( \ldots )</td>
<td>( q_{TS}(0, n - 1) )</td>
<td>( q_{TS}(0, n) = q_{TS}(0, n - 1) + \tilde{s}(n) )</td>
</tr>
<tr>
<td>Component, ( C_{ST} )</td>
<td>( C_{ST}(0) )</td>
<td>( C_{ST}(1) = g^{q_{TS}(0,1)} )</td>
<td>( \ldots )</td>
<td>( C_{ST}(n - 1) )</td>
<td>( C_{ST}(n) = g^{q_{TS}(0,n)} )</td>
</tr>
<tr>
<td>Component, ( C'_{ST} )</td>
<td>( C_{ST}(0) )</td>
<td>( C'<em>{ST}(1) = H(\text{Att}</em>{ST}(1))^{q_{TS}(0,1)} )</td>
<td>( \ldots )</td>
<td>( C'_{ST}(n - 1) )</td>
<td>( C'<em>{ST}(n) = H(\text{Att}</em>{ST}(n))^{q_{TS}(0,n)} )</td>
</tr>
<tr>
<td>Component, ( C )</td>
<td>( C(0) )</td>
<td>( C(1) = C(0) \cdot h^{\tilde{s}(1)} )</td>
<td>( \ldots )</td>
<td>( C(n - 1) )</td>
<td>( C(n) = C(n - 1) \cdot h^{\tilde{s}(n)} )</td>
</tr>
<tr>
<td>Component, ( \tilde{C} )</td>
<td>( \tilde{C}(0) )</td>
<td>( \tilde{C}(1) = \tilde{C}(0) \cdot e(g, g)^{\alpha \tilde{s}(1)} )</td>
<td>( \ldots )</td>
<td>( \tilde{C}(n - 1) )</td>
<td>( \tilde{C}(n) = \tilde{C}(n - 1) \cdot e(g, g)^{\alpha \tilde{s}(n)} )</td>
</tr>
<tr>
<td>Secret mask, ( s )</td>
<td>( s(0) )</td>
<td>( s(1) = s(0) + \tilde{s}(1) )</td>
<td>( \ldots )</td>
<td>( s(n - 1) )</td>
<td>( s(n) = s(n - 1) + \tilde{s}(n) )</td>
</tr>
</tbody>
</table>
The web site ‘printer.com’ can ask the website ‘poster.com’ to print a poster for a file ‘pic-1’ in the time slot ‘2011|06|27|13|**’

‘printer.example.com’

FILE-LOC = http://photos.com/2010/brunce/pic-1,
FILE-LOC = http://photos.com/2010/brunce/pic-2,
SEC-CLASS = 3, PERMIS=r,
/* current time slot */
TIMESLOT = 2011|06|27|13|**, 
/* future time slot(s)*/
TIMESLOT = 2011|06|27|14|**.

‘poster.com’

FILE-LOC = http://photos.com/2010/brunce/pic-1,
SEC-CLASS = 3, PERMIS = r,
/* current time slot */
TIMESLOT = 2011|06|27|13|**.
Recap: The procedures and protocols in AAuth

<table>
<thead>
<tr>
<th>AAuth</th>
<th>Procedures/Protocols</th>
<th>Outputs</th>
</tr>
</thead>
</table>
| Setup procedure | 1. A bilinear group $G_1, G_2$  
2. A bilinear map $e$  
3. A generator $g$ of $G_1$  
4. hash function $H$ |
| File encapsulation procedure | 1. An access policy $A$ from both confined and descriptive attributes  
2. An access tree $\tau$  
3. An archive file |
| Service request protocol | An access policy $A$ |
| Token request protocol | An ABE-token |
| File access protocol | An archive file |
| File decapsulation procedure | 1. A header in plaintext form  
2. An integrity tag  
3. A data file in plaintext form |
| Time slot synchronization protocol | 1. Two ciphertext components  
2. Two update values  
3. A new time slot header |
A Block Diagram of AAuth Authorization Scheme

- Authority
- Consumer
- Autorizer
- ID
- Owner
- User-Agent (Browser)
- Server
- Description Att
- Confined Att

Symbols:
- $C$: Common Lock
- $D$: Common Key
- $C_i$: Confined Lock
- $D_i$: Confined Key
- $C_j$: Descriptive Lock
- $D_j$: Descriptive Key
- $D_i^*$: Descriptive Key

Logic:
AND

Confined Att
Description Att
Security Analysis

i. With **end-to-end encryption** and signature, a cloud server cannot subvert the confidentiality and integrity of the data it is hosting.

ii. With **end-to-end authorization**, the access policy is enforced by the encryption algorithm, not by a cloud server.

iii. Without **cooperation** between owners and the authority, none of them can individually generates ABE-tokens.

iv. Since owners can verify confined keys before combining, the **authorizer** cannot faked keys to owners.

v. Separating keys to two parts, each of which is individually sent to consumer, to fabricate keys, owners face DLP while consumers face DBDH problems.

vi. The scheme can prevent eavesdropping, active, MITM, off-line attacks form external adversaries.
## On-line Cryptographic Cost

<table>
<thead>
<tr>
<th>Role</th>
<th>Signing</th>
<th>Verify</th>
<th>Exponent</th>
<th>Paring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Owner</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consumer</td>
<td>2</td>
<td></td>
<td></td>
<td>$2(</td>
</tr>
<tr>
<td>Authorizer</td>
<td>2</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authority</td>
<td>1</td>
<td></td>
<td>$2</td>
<td>I - 5</td>
</tr>
<tr>
<td>Server</td>
<td>1</td>
<td></td>
<td>$2</td>
<td>L</td>
</tr>
</tbody>
</table>
Additional Communication Cost

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Additional messages</th>
<th>Message flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service request</td>
<td>2</td>
<td>$C \rightarrow S$</td>
</tr>
<tr>
<td>Token request</td>
<td>2</td>
<td>$AZ \rightarrow AA$</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>$O \rightarrow AZ$</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>$C \rightarrow O$</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>$C \rightarrow AA$</td>
</tr>
<tr>
<td>File access</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>
Simulations

- Tool: OMNet++
- Settings: the cloud network has a bandwidth at 400 packets/s, each owner continuously requests services in exponential distribution, each service request transfers three 256 KB-files as a dummy load, the number of owners (users) starts from 100 to 700.

### OAuth-AAuth

![Latency vs Number of Owners Graph]

- **Latency (Second)**
- **Number of Owners**
- **Graph Legend**:
  - Oauth
  - Aauth

<table>
<thead>
<tr>
<th>Number of Owners</th>
<th>Latency (Second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>6.6515116</td>
</tr>
<tr>
<td>200</td>
<td>6.7936554</td>
</tr>
<tr>
<td>300</td>
<td>10.169864</td>
</tr>
<tr>
<td>400</td>
<td>16.5296745</td>
</tr>
<tr>
<td>500</td>
<td>22.9078254</td>
</tr>
<tr>
<td>600</td>
<td>29.271162</td>
</tr>
<tr>
<td>700</td>
<td>35.6323535714</td>
</tr>
</tbody>
</table>
Related Work

- Work on a cryptographic **storage** system
- Proof of Retrievability (POR) (Bowers, 09): a framework on archival or backup files in cloud storage
- **Proxy** re-encryption and **lazy re-encryption** (Wang, et. al. 2010): Fine-grained and scalable access control in cloud computing that exploits KP-ABE to reduce complexity in key management and key distribution
- **K2C** (Zarandioon, 2011), a **scalable ABE-based** access hierarchies by combining KP-ABC and key-updating scheme and combining KP-ABE and signature scheme
Conclusions & Remarks

1. **ABE-tokens** for each authorization grant.
2. A **user-centric** system in which an owner controls the authorization system to protect her resources.
3. **End-to-end** cryptographic functions from an owner to a consumer.
4. A light-weight encryption for time slot **synchronization**.
5. No significant computation cost for users.
6. AAuth’s cost is independent of the number of users in the system.
7. An acceptable increasing cost is compensated by achieving better security than OAuth.
8. **AAuth** is as secure as the original CP-ABE scheme and can resist both internal and external adversaries.
### The comparison of Kerberos, OAuth, and AAuth

<table>
<thead>
<tr>
<th></th>
<th>Kerberos</th>
<th>OAuth</th>
<th>AAuth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trust platform</td>
<td>Client</td>
<td>Browser</td>
<td>Browser</td>
</tr>
<tr>
<td>SSO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Key management</td>
<td>No</td>
<td>No</td>
<td>Integrated &amp; distributed</td>
</tr>
<tr>
<td>Data-at-rest</td>
<td>Plaintext</td>
<td>Plaintext</td>
<td>Ciphertext</td>
</tr>
<tr>
<td>Policy mechanism</td>
<td>ACL / capabilities</td>
<td>ACL / capabilities</td>
<td>ABE attributes</td>
</tr>
<tr>
<td>Policy enforced by</td>
<td>server</td>
<td>server</td>
<td>ABE decryption</td>
</tr>
<tr>
<td>Token generation</td>
<td>AS &amp; TGS</td>
<td>OAuth provider</td>
<td>Owner, Authorizer, and Authority(s)</td>
</tr>
<tr>
<td>Ext. attacks resisted by</td>
<td>Time synch.</td>
<td>SSL/TLS</td>
<td>multi SSL/TLS</td>
</tr>
<tr>
<td>Int. attacks resisted by</td>
<td>No</td>
<td>No</td>
<td>modified CP-ABE</td>
</tr>
</tbody>
</table>