

# CADAT: control de acceso basado en tokens y cadenas hash delegables

Guillermo Navarro, Joaquín García

Dept. de Informàtica,  
Universitat Autònoma de Barcelona,  
{gnavarro, jgarcia}@ccd.uab.es,

**Resumen** En este artículo presentamos una introducción a un sistema de control de acceso basado en tokens y cadenas hash delegables. Este sistema permite aprovechar las ventajas de las cadenas hash de la misma manera que los sistemas de micropago. Por otra parte utiliza una infraestructura de autorización que permite establecer delegaciones mediante certificados entre las entidades del sistema. El sistema recibe el nombre de CADAT (*Chained and Delegable Authorization Tokens*).

## 1. Introducción

En control de acceso en sistemas informáticos, es actualmente un campo de investigación, desarrollo y aplicación muy activo. Recientemente se han propuesto modelos, técnicas y sistemas novedosos sobre el control de acceso a sistemas distribuidos. Generalmente se busca diseñar sistemas flexibles, escalables y fáciles de utilizar por los usuarios sin necesidad de sacrificar la seguridad. Mediante delegación de permisos o autorizaciones, por ejemplo, podemos introducir flexibilidad a un sistema, donde los usuarios pueden delegar tareas, autorizaciones o permisos de manera fácil a otros usuarios. Algunos de los primeros sistemas en introducir la delegación de autorizaciones fueron KeyNote[3] y SPKI/SDSI (*Simple Public Key Infrastructure/Simple Distributed Secure Infrastructure*)[4]. Algunos autores también se refieren a ellos como sistemas de gestión de confianza (*trust management*).

Otro campo interesante y más tradicional en el control de acceso, es la utilización de *tokens* de acceso. En dichos sistemas un usuario recibe un token o ticket que le permite acceder a cierto recurso o servicio. Por ejemplo, Kerberos[8], es un sistema muy popular que utiliza tickets de acceso.

Una aplicación, en cierta medida, similar a los tokens de acceso son los micropagos. Un sistema de micropagos se caracteriza por permitir pagos de pequeñas cantidades de manera ágil. Aunque los sistemas de micropagos existen desde hace mucho tiempo y han recibido algunas críticas, siguen siendo un campo de investigación interesante [7]. Como prueba de ello, recientemente han aparecido varias empresas como Peppercoin (<http://www.peppercoin.com>) o Bitpass (<http://www.bitpass.com>), que ofrecen servicios de micropago.

Una aportación interesante de los sistemas de micropagos ha sido la introducción de cadenas hash [2,9,10]. El uso de cadenas hash permite agilizar mucho la emisión

de micropagos, al sustituir operaciones criptográficas costosas por cálculos sencillos (funciones hash).

Recientemente han aparecido propuestas para utilizar sistemas de gestión de autorizaciones con delegación para implementar micropagos. Nosotros nos hemos basado en una de estas propuestas [5], para proponer un sistema de tokens de acceso. Este sistema recoge ventajas de los sistemas de micropagos y de la delegación de autorizaciones. Nuestra propuesta recibe el nombre de CADAT (*Chained and Delegable Authorization Tokens*). En CADAT los tokens de autorización se forman a partir de cadenas hash, esto hace que la emisión y verificación de los tokens sea rápida y eficiente. Por otra parte introducimos la posibilidad de delegar cadenas de tokens entre usuarios de diversas maneras lo que añade flexibilidad al sistema.

En este artículo presentamos una introducción general al sistema CADAT, sus bases teóricas y comentarios sobre su implementación, sin no entramos en detalle en su aplicación concreta. Esta aplicación se realiza sobre una plataforma de agentes móviles segura, donde las características de CADAT cobran una importancia relevante. Cabe decir que la posibilidad de que agentes móviles puedan entregar tokens de acceso a recursos sin necesidad de realizar ninguna operación criptográfica es una de las ventajas más destacables del sistema.

En la sección 2 introducimos las bases del sistema CADAT. En la sección 3 introducimos SPKI/SDSI y su utilización en CADAT. En la sección 4 discutimos como codificar los permisos basados en elementos de cadenas hash en SPKI/SDSI. A continuación mostramos el funcionamiento básico de CADAT en la sección 5. Finalmente la sección 6 presenta las conclusiones y líneas de trabajo futuro.

## 2. Delegación y cadenas hash

En esta sección mostramos como se pueden introducir las cadenas hash en la delegación de autorizaciones. Para ello nos basamos principalmente en la notación y el modelo presentado en [5], simplificándolo para mayor claridad. La idea principal es considerar los elementos de una cadena hash como permisos.

Adoptamos la siguiente notación y definiciones:

- $K_A$ : clave pública del usuario  $A$ .
- $sK_A$ : clave privada del usuario  $A$ .
- $\{m\}_{sK_A}$ : mensaje  $m$  firmado por el usuario  $A$  con la clave privada  $sK_A$ .
- $(|K_B, p|)_{K_A}$  relación de delegación directa en la que el usuario  $A$  delega el permiso  $p$  al usuario  $B$ . Esta relación se puede establecer mediante un *certificado de autorización*. De momento denotaremos dicho certificado como  $\{|K_B, p|\}_{sK_A}$ .
- $h(m)$ : función hash criptográfica unidireccional aplicada a  $m$ .
- $[h^n(m), h^{n-1}(m), \dots, h^1(m), h^0(m)]$ : cadena hash sobre el valor inicial  $m$ . Generalmente  $m$  incluye información diversa como algún componente aleatorio suficientemente grande como para asegurar que sea único.
- $\mathcal{P}$ : conjunto de todos los permisos del sistema. El conjunto de permisos se estructura como un *lattice* donde la relación de orden queda determinada por la inclusión de un permiso por otro ( $\preceq$ ). Y la cota inferior máxima queda definida por la intersección ( $\wedge$ ).

- $\mathcal{P}_{\mathcal{H}}$ : conjunto de permisos, donde cada permiso es un elemento de una cadena hash. En el caso de  $\mathcal{P}_{\mathcal{H}}$ , es necesario redefinir o refinar la relación de orden. Dados  $x, y \in \mathcal{P}_{\mathcal{H}}$ ,  $x \preceq y$  respecto a principal  $P$ , si y solo si  $P$ , conociendo  $y$ , puede determinar de una manera factible algún  $x$  e  $i$  tal que  $h^i(x) = y$  [5]. En este caso, determinar de manera factible, quiere decir que  $P$  puede encontrar  $x$  e  $i$  sin necesidad de invertir la función hash.

Siguiendo esta notación podemos expresar las reglas de reducción de certificados de SPKI/SDSI [4], considerando el conjunto de permisos  $\mathcal{P}_{\mathcal{H}}$  de la siguiente manera:

$$\frac{(|K_B, x|)_{K_A}; y \preceq x}{(|K_B, y|)_{K_A}} \quad (1)$$

$$\frac{(|K_C, x|)_{K_B}; (|K_B, y|)_{K_A}}{(|K_C, x \wedge y|)_{K_A}} \quad (2)$$

La introducción de cadenas hash como permisos presenta novedades importantes, como la posibilidad de delegar autorizaciones específicas sin necesidad de emitir ningún certificado, y la posibilidad de transferir (delegar) partes de la cadena a otros principales.

Para aclarar el uso de las cadenas hash con delegación consideramos el siguiente ejemplo. Tenemos tres usuarios Alice, Bob, y Carol, con sus respectivas claves públicas  $K_A$ ,  $K_B$ , y  $K_C$ . Supongamos que Alice es una determinada autoridad de autorización. Alice genera la siguiente cadena hash a partir de un mensaje  $m$ :  $[h^5(m), h^4(m), h^3(m), h^2(m), h^1(m)]$  donde cada elemento de la cadena se corresponde a un permiso determinado.

Alice puede emitir el siguiente certificado:

$$\{|K_B, h^5(m)|\}_{sK_A} \quad (3)$$

Con este certificado delega  $h^5(m)$  a Bob, que pasa a tener dicho permiso. Si Alice quiere ahora delegar el permiso  $h^4(m)$  a Bob, no es necesario que emita ningún certificado, simplemente tiene que hacer público el valor  $h^4(m)$ . Una vez hecho público, Bob pasa a tener el permiso de manera automática. Bob puede demostrar que es el receptor del permiso  $h^4(m)$  porque puede demostrar que tiene el permiso  $h^5(m)$ . De la misma manera, Alice podría ir haciendo públicos los sucesivos permisos, por ejemplo  $h^3(m)$ .

De manera más general, si Bob tiene un certificado que le otorga el permiso  $h^i(m)$  de manera directa, y Alice hace público el valor  $h^j(m)$  ( $i > j$ ), Bob pasa a tener los permisos  $h^i(m), \dots, h^j(m)$ . Por la regla (1), tenemos que  $(|K_B, h^i(m)|)_{sK_A}$ , y  $h^j(m) \preceq h^i(m)$ , por lo que  $(|K_B, h^j(m)|)_{sK_A}$ .

Otro aspecto importante que introduce la delegación, es que Bob podría transferir (delegar) parte de los permisos a otro principal. Siguiendo el ejemplo, Bob esta en posesión de  $h^5(m)$ ,  $h^4(m)$  y  $h^3(m)$ . Bob puede delegar, este último permiso a Carol emitiendo el siguiente certificado:

$$\{|K_C, h^3(m)|\}_{sK_B} \quad (4)$$

Bob delega  $h^3(m)$  a Carol. Si Alice hace público  $h^2(m)$ , Carol puede demostrar que esta en posesión de dicho permiso mediante los certificados (3), y (4). Es importante

remarcar que en ningún caso Carol puede demostrar que tiene los permisos  $h^5(m)$  y  $h^4(m)$ . Solo puede recibir los permisos de la cadena iguales o inferiores a  $h^3(m)$ .

En la siguiente sección mostramos como se puede utilizar SPKI/SDSI para implementar este modelo. En [6] se muestra como se puede utilizar KeyNote para codificar el un modelo similar.

### 3. Certificados de autorización SPKI/SDSI

En SPKI/SDSI cada principal posee un par de claves criptográficas, y esta representado por la clave pública. Dicho de otra manera, en SPKI/SDSI cada principal es su clave pública. Un certificado de autorización permite establecer un vínculo entre una clave pública y una autorización o permiso. De esta manera se evita la indirección que presentan PKIs tradicionales donde es necesario establecer un vínculo entre una clave pública y un identificador global, y otro entre el identificador y la autorización. Denotamos un certificado de autorización como:

$$(I, S, tag, p, V) \quad (5)$$

Donde:

- $I$ : es el emisor, el principal que otorga la autorización.
- $S$ : es el sujeto, el principal que recibe la autorización.
- $tag$ : es la autorización específica que otorgada mediante el certificado.
- $p$ : es el bit de delegación o propagación. Si este bit esta activo el sujeto del certificado podrá delegar la autorización recibida a otro principal.
- $V$ : es la especificación de validez, que incluye el rango de tiempo durante el cual el certificado es valido y otras posibles condiciones.
- Comentario: aunque no lo mostramos en la notación, el certificado incluye un campo con información arbitraria que recibe el nombre de comentario.

Como se puede observar los certificados de autorización de SPKI/SDSI nos permitirán expresar de manera sencilla los certificados de autorización comentados en la sección 2.

### 4. Cadenas hash como permisos SPKI/SDSI

En SPKI/SDSI los permisos se expresan con el elemento  $tag$ . El formato utilizado por SPKI/SDSI para representar toda la información son las S-expressions. Para facilitar el procesamiento de las autorizaciones, incluimos el índice del componente de la cadena hash ( $i$ ) y un identificador de cadena ( $cid$ ) en la autorización. De esta manera un permiso  $p \in \mathcal{P}_{\mathcal{H}}$ , tendrá la forma:

$$p = (cid, i, h^i(m)) \quad (6)$$

Donde el hash sobre  $m$  incluye  $i$  y  $cid$  en cada paso para evitar que puedan ser falsados. El identificador de cadena  $cid$ , es una cadena de bits aleatoria suficientemente grande para asegurar su unicidad. Finalmente el mensaje  $m$  original incluirá información adicional como la clave pública del principal que la ha generado, etc. No entramos

en detalle en la información exacta de  $m$ , ya que dependerá en gran medida de la aplicación concreta del permiso.

A la hora de codificar el permiso (6) en un tag SPKI/SDSI consideramos dos alternativas. Estas se basan en la necesidad de verificar el hash del elemento a la hora de hacer la intersección<sup>1</sup>. Es decir, al hacer la intersección de dos permisos con un hash  $h^i(m)$  y  $h^j(m)$  respectivamente, si  $i \geq j$ , el resultado será  $h^i(m)$ , la intersección puede además, verificar que  $(h^j)^x(m) = h^i(m)$  para algún  $x$ . Esta verificación permite descartar inmediatamente permisos erróneos o falseados. Si la comprobación no se hace en la intersección, es importante remarcar que se tendría que hacer a posteriori para poder verificar la validez de una prueba de autorización.

#### 4.1. Intersección de tags sin verificación de hash

En este caso podemos codificar el permiso utilizando las estructuras de SPKI/SDSI. Por ejemplo, consideramos un tag que se corresponde a un permiso con identificador de cadena (h-chain-id) 123456789, índice del elemento en la cadena (h-chain-index) igual a 7, y  $h^7(m)$  (h-val) igual a 899b786bf7dfad58aa3844f2489aa5bf. El elemento más importante es el índice, para el que podemos utilizar un rango numérico que hará intersección con rangos mayores o iguales a 7.

A la hora de hacer la intersección nos encontramos con un problema, y es que si incluimos  $h^7(m)$  en el tag, no interesectará de manera correcta. Lo que hacemos es introducir una pequeña modificación. El valor *h-val* se tiene que tratar de manera diferente. Para ello, ponemos el valor *h-val* como comentario en el certificado. El tag finalmente quedaría:

```
(tag
  (h-chain-id |123456789|)
  (h-chain-index (* range numeric ge 7)))
(comment (h-val (hash md5 |899b786bf7dfad58aa3844f2489aa5bf|)))
```

Esto permite utilizar el motor de decisión de SPKI/SDSI directamente, sin perder ningún tipo de información. La principal desventaja de este método es que al verificar una prueba de autorización, el verificador necesita realizar una operación adicional: comprobar la consistencia de la cadena (o subcadena) hash.

#### 4.2. Intersección de tags con verificación de hash

En este caso, es necesario redefinir la intersección de tags para las autorizaciones que se corresponden a elementos de una cadena hash. Para ello, introducimos un nuevo tipo de tag, el <tag-hash-auth>. La definición en BNF del nuevo <tag-hash-auth> se muestra en la Figura 1.

También introducimos una nueva operación de intersección: *HCAIntersect* (*Hash Chained Authorization Intersection*). La intersección de dos tags de  $\mathcal{P}_{\mathcal{H}}$  será igual al tag que presente un índice de cadena mayor, siempre que el identificador de cadena sea el mismo, y se pueda verificar el hash. Por ejemplo, dados los siguientes tags:

<sup>1</sup> La intersección de tags de autorización es una de las operaciones clave, para la reducción de cadenas de certificados y verificación de pruebas de autorización

```

<tag>:: <tag-star> | "(" "tag" <tag-expr> ")" ;
<tag-star>:: "(" "tag" "(" "*" ")" ;
<tag-expr>:: <simple-tag> | <tag-set> | <tag-string> | <tag-hash-auth>;
<tag-hash-auth>:: "(" "hash-auth" <chain-id> <chain-index> <hash>";
<chain-index>:: "(" "chain-index" <decimal> ")";
<chain-id>:: "(" "chain-id" <byte-string> ")";

```

**Figura 1.** Definición de <tag-hash-auth>.

```

(tag
  (hash-auth
    (hchain-id |lksjfsDFIsdfkj0sndKIShfoMSKJSD|)
    (hchain-index 14)
    (hash md5 |899b786bf7dfad58aa3844f2489aa5bf|)))
(tag
  (hash-auth
    (hchain-id |lksjfsDFIsdfkj0sndKIShfoMSKJSD|)
    (hchain-index 15)
    (hash md5 |d52885e0c4bc097f6ba3b4622e147c30|)))

```

Su intersección ( $HCAIntersect$ ) será igual a el segundo tag, porque el identificador es igual y el índice del segundo es mayor. Pero además se verifica la validez del hash. A continuación mostramos un algoritmo simplificado para  $HCAIntersect$  con verificación de hash (*HCAIntersect full algorithm*).

---

**Algoritmo 1:**  $HCAIntersect$  full algorithm

---

```

input   :  $p = (id_p, i, h^i(m)_p)$ ,  $q = (id_q, j, h^j(m)_q)$ , tal que  $p, q \in \mathcal{P}_H$ 
output  :  $r$  tal que  $r = HCAIntersect(p, q)$ 
begin
  if  $id_p \neq id_q$  then  $r \leftarrow NULL$ ;
  if  $i \geq j$  then
    if  $verifyHashSubChain(p, q)$  then  $r \leftarrow p$ ;
    else  $r \leftarrow NULL$ ;
  end
  else
    if  $verifyHashSubChain(q, p)$  then  $r \leftarrow q$ ;
    else  $r \leftarrow NULL$ ;
  end
end

```

---

La implementación de elementos de cadenas hash como autorizaciones y el algoritmo  $HCAIntersect$  en SPKI/SDSI la hemos realizado en Java con la librería JSDSI [1]. JSDSI es una librería de código libre, que actualmente esta siendo activamente desarrollada. La implementación de este nuevo *tag* ha resultado sencilla, siendo solo necesario implementar el algoritmo  $HCAIntersect$  y algunas sencillas funciones de conversión.

---

**Algoritmo 2:** verifyHashSubChain function

---

**input** :  $p = (id_p, i, h^i(m_p)), q = (id_q, j, h^j(m_q))$ , donde  $i \geq j$   
**output** :  $res = true$  si  $h^i(m)$  y  $h^j(m)$  pertenecen a la misma cadena hash,  $res = false$  en caso contrario  
**begin**  
   $res \leftarrow false$  ;  
   $aux \leftarrow h^j(m_p)$  ;  
  **for**  $x \in [(j-1)..i]$  **do**  
    **if**  $aux = h^i(m_q)$  **then**  $res \leftarrow true$  ;  
     $aux \leftarrow h(aux)$  ;  
  **end**  
**end**

---

Para ello hemos colaborado activamente con miembros del proyecto JSDSI resultando en la definición de una interficie estándar para implementación de nuevos *tags* con su respectivo algoritmo de intersección.

## 5. Aplicación a tokens de acceso: CADAT

La principal motivación del diseño de CADAT es su aplicación en el control de acceso basado en tokens. En esta sección mostramos el funcionamiento básico de CADAT mediante un ejemplo. Consideramos un escenario donde agencias de noticias ofrecen acceso a sus bases de datos a usuarios. El acceso se controla por tokens, es decir cada vez que un usuario accede a (o lee) una noticia necesita entregar un token.

Por ejemplo, la central de noticias *AcmeNews* quiere permitir 9 accesos a la usuaria Alice a todas sus agencias de noticias repartidas por todo el mundo (*AcmeNews-SudAfrica*, *AcmeNews-India*, *AcmeNews-Antartida*, etc.). Para ello, *AcmeNews* emite un contrato mediante un certificado de autorización a Alice en el que le autoriza a utilizar 10 *tokens* de acceso *acme* durante un periodo de tiempo  $V_o$  (el primer token no se utilizará como token de acceso).

$$(AcmeNews, Alice, auth_{star}, p = true, V_0) \quad (7)$$

Donde  $auth_{star}$  se corresponde a:  $(acmeID, 10, *)$ . Este es un  $\langle tag-hash-auth \rangle$ , con el valor hash igual a “\*”. Este es un símbolo especial en SPKI/SDSI que permite hacer intersección con cualquier cadena de caracteres.

Denotamos este primer certificado como *chain-contract-cert* o certificado de contrato de cadena, ya que establece un contrato que permitirá a Alice demostrar que ha sido autorizada por *AcmeNews* a utilizar 9 *tokens acmeID*.

Ahora Alice puede generar la cadena hash de 10 elementos:

$$[(acmeID, 1, h^1(m)), (acmeID, 2, h^2(m)), \dots, (acmeID, 10, h^{10}(m))] \quad (8)$$

El mensaje inicial (o semilla)  $m$  generalmente incluirá información del certificado (7), o información específica compartida por *AcmeNews* y Alice.

Supongamos que Alice viaja a la agencia *AcmeNews-Antartida* para buscar noticias sobre las costumbres del *Aptenodytes forsteri*<sup>2</sup>. Para ello, establece un contrato inicial mediante el siguiente certificado:

$$(Alice, AcmeNews - Antartida, auth_{10}, p = true, V_1) \quad (9)$$

Donde  $auth_{10} = (acmeID, 10, h^{10}(m))$ . Este certificado recibe el nombre de *token-contract-cert* o certificado de contrato de tokens. En él, Alice establece ante *AcmeNews-Antartida* que esta en posición de gastar 10 tokens *acme*. Por su parte *AcmeNews-Antartida* puede verificarlo mediante los certificados (7) y (9) con el motor de decisión de SPKI/SDSI<sup>3</sup>.

Una vez emitido el *token-contract-cert*, Alice puede empezar a utilizar tokens para acceder a los recursos de *AcmeNews-Antartida*, simplemente enviando los *tokens* a *AcmeNews-Antartida*. Por ejemplo, Alice envía el valor  $auth_9 = (acmeID, 9, h^9(m))$  y  $auth_8 = (acmeID, 8, h^8(m))$ . *AcmeNews-Antartida* puede verificar que  $h(h^9(m)) = h^{10}(m)$  y que  $h(h^8(m)) = h^9(m)$  y junto a los certificados (7) y (9) puede permitir dos accesos a Alice. De esta manera, Alice, al hacer públicos los elementos de la cadena hash, a delegado implícitamente los tokens a *AcmeNews-Antartida* en base al contrato inicial *token-contract-cert* (9).

### 5.1. Delegación de *token-contract-cert*

Supongamos ahora, que *AcmeNews-Antartida* decide transferir el contrato inicial con Alice *token-contract-cert*(9) a *AcmeNewsZoology* ya que Alice necesita hacer las siguientes consultas al departamento de zoología de la agencia. Para ello, utiliza el último token que ha recibido de Alice como un nuevo *token-contract-cert* que emite a *AcmeNewsZoology*:

$$(AcmeNews - Antartida, AcmeNewsZoology, auth_8, p = true, V_2) \quad (10)$$

*AcmeNewsZoology* puede verificar junto con el *token-contract-cert* (9) y el *chain-contract-cert* (7), que Alice esta autorizada a utilizar 7 tokens *acme*. Ahora Alice emite el siguiente token  $auth_7$  que puede ser aceptado por *AcmeNewsZoology*, quien realizará las verificaciones pertinentes.

Mediante el *token-contract-cert* (10), *AcmeNews-Antartida* ha podido delegar parte de su contrato inicial con Alice a *AcmeNewsZoology*. Generalmente esta delegación puede ser transparente a Alice, permitiendo a *AcmeNews* subcontratar servicios fácilmente.

La Figura 2 muestra gráficamente los dos *token-contract-cert* emitidos, así como los tokens entregados por Alice.

<sup>2</sup> También conocido como pingüino emperador.

<sup>3</sup> Motor que simplemente implementa los algoritmos de reducción y descubrimiento de cadenas de SPKI/SDSI

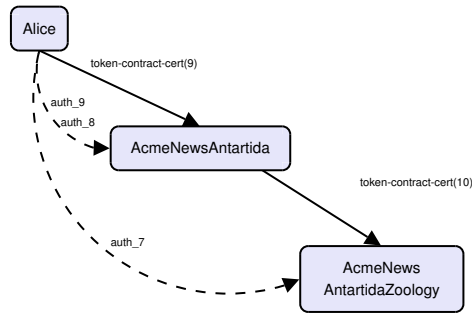


Figura 2. Ejemplo de delegación del *token-contract-cert*.

## 5.2. Delegación de *chain-contract-cert*

Otro posible caso de delegación sería la delegación del *chain-contract-cert* por parte de Alice. Supongamos que Alice decide transferir el resto de tokens que le quedan a su colega Bob, para que él los pueda utilizar. Para ello, simplemente tiene que delegarle el *chain-contract-cert* (7) que recibió directamente de *AcmeNews* emitiendo el siguiente certificado:

$$(Alice, Bob, auth_7, p = true, V_3) \quad (11)$$

Con él está autorizando a Bob a utilizar los 6 tokens restantes de la cadena. Es importante remarcar que también tiene que hacer llegar a Bob de alguna manera la cadena hash (8), o el valor inicial  $m$  para que Bob pueda seguir emitiendo los siguientes elementos de la cadena. Este valor podría ir cifrado como comentario del certificado por ejemplo.

Ahora Bob puede emitir el valor  $auth_6$ . Este token será aceptado por *AcmeNews-Zoology*, quien podrá verificar que Bob está autorizado a emitir el token gracias a los *chain-contract-cert* (7) y (11).

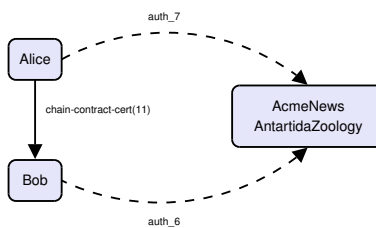


Figura 3. Ejemplo de delegación del *chain-contract-cert*.

### 5.3. Comentarios

En las secciones anteriores hemos introducido tanto el *chain-contract-cert*, como el *token-contract-cert*. Aunque ambos certificados tiene un forma muy similar la principal diferencia viene dada por el uso que se hace de ellos. De manera resumida:

- *chain-contract-cert*: permite delegar una cadena hash o parte de ella a un principal. Éste será el que utilice los tokens para acceder o utiliza un servicio. Como caso especial se contempla el primer contrato que establece la confianza depositada por una autoridad (*AcmeNews*) en un principal determinado.
- *token-contract-cert*: este certificado permite delegar la cadena hash o parte de ella a un principal, que será el consumidor de los tokens. Esta situación es especialmente interesante ya que podría permitir la subcontratación de servicios por parte de una empresa de manera transparente al cliente.

## 6. Conclusiones y trabajo futuro

En este artículo hemos presentado de manera general las ideas que hay detrás del sistema CADAT, basado principalmente en [5]. Hemos descrito sus bases, su utilización y funcionamiento general.

En esta descripción hemos dejado de lado algunos detalles que seria necesario aclarar en futuros trabajos. Por ejemplo en el caso de SPKI/SDSI no hemos discutido las implicaciones de por ejemplo certificados umbral, o el uso de certificados de nombres. Generalmente la introducción de estos elementos no presenta cambios importantes al sistema por lo que hemos preferido centrarnos en la idea central para mayor claridad.

Actualmente estamos trabajando sobre un prototipo que ha de resultar en el refinamiento del sistema, funcionalidad adicional y asentamiento de las principales ideas. Así como técnicas para evitar la reutilización de tokens y sistemas de auditoría. Debido a la falta de espacio no hemos entrado en la descripción de dicho prototipo y su aplicación. Esta se realiza sobre una plataforma de agentes móviles segura, donde las características de CADAT cobran un especial interés. El hecho de agentes móviles puedan entregar tokens de acceso a recursos sin necesidad de realizar ninguna operación criptográfica es una de las ventajas más destacables.

## Referencias

1. JSDSI: A Java implementation of the SPKI/SDSI standard. <http://jsdsi.sourceforge.net>.
2. R. Anderson, H. Manifavas, and C. Shutherland. Netcard - a practical electronic cash system. In *Cambridge Workshop on Security Protocols*, 1995.
3. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust Management System. RFC 2704, IETF, September 1999.
4. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, IETF, September 1999.
5. S. Foley. Using trust management to support transferable hash-based micropayments. In *Financial Cryptography 2003*, pages 1–14, 2003.

6. S. Foley and T. B. Quillinan. Using trust management to support micropayments. In *Annual Conference on Information Technology and Telecommunications*, October 2002.
7. M. Lesk. Micropayments: An Idea Whose Time Has Passed Twice? *IEEE Security & Privacy*, January/February 2004.
8. B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, pages 33–38, 1994.
9. T. P. Pedersen. Electronic payments of small amounts. In *Proc. 4th International Security Protocols Conference*, pages 59–68, 1996.
10. R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In *Proc. 4th International Security Protocols Conference*, pages 69–87, 1996.