Tema 3: Esquemas, protocolos y mecanismos de soporte

Índice

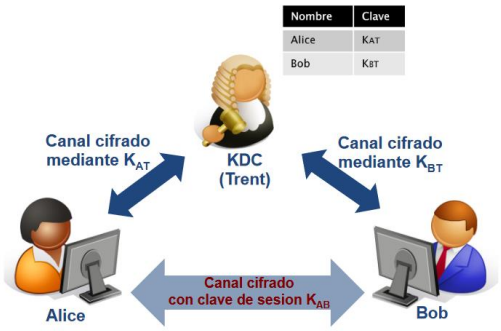
Gestión de las claves

Protocolos de distribución de claves simétricas

Hay escenarios donde la utilización de la criptografía de clave pública para el intercambio de una clave de sesión K_{AB} no es posible, o simplemente no es conveniente. A pesar de esto, Alice y Bob van a seguir necesitando de alguna solución que les permita, aún estando geográficamente lejanos, decidir esa clave de sesión K_{AB}.

En estos casos, la solución pasa por algún protocolo de **distribución centralizada de claves** para los usuarios del sistema. Consiste en hacer uso de una **tercera parte confiable (TTP)** que en este caso se denomina **Centro de Distribución de Claves KDC** (*Key Distribution Center*).

En el modus operandi general de este tipo de protocolos, cada usuario del sistema comparte, desde el inicio, una **clave secreta con el KDC** (K_{AT}, K_{BT}). Esta fue asignada mediante algún proceso de registro o inscripción del usuario ante el KDC.

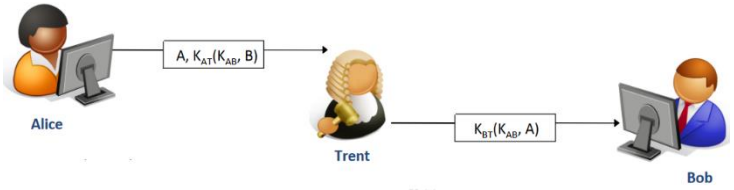


El uso de un *KDC* se basa en el uso de claves jerárquicas, de manera que se requieren al menos dos niveles de claves (canal cifrado mediante K_{AT}, K_{BT}, canal cifrado mediante K_{AB}). La mayoría de las técnicas de distribución de claves se adaptan a situaciones, escenarios y aplicaciones específicas, de manera que **son diversos los esquemas** que se integran a entornos locales donde todos los usuarios tienen acceso a un servidor común de confianza. Hay muchos modelos de distribución de claves:

- Modelos simples
- Modelos genéricos, y dentro de los genéricos nos podemos encontrar:
    - Modelos PULL.
    - Modelos PUSH.
    - Combinaciones entre ambos.

Modelo simple

El protocolo **"La rana de la boca grande"** es un ejemplo de modelo simple para la distribución de claves. Se resume en la siguiente imagen:



Funcionamiento general:

1. *A* genera una clave de sesión K_{AB} y se la envía a *KDC*. El mensaje incluye la identidad de A, la identidad de B y la clave de sesión cifrada con el K_{AT}.

2. El *KDC* verifica la identidad de *A* y reenvía K_{AB} a *B* cifrado con K_{BT}.
3. *B* verifica la identidad de *KDC* y por la K_{BT} y obtiene la **clave de sesión**.

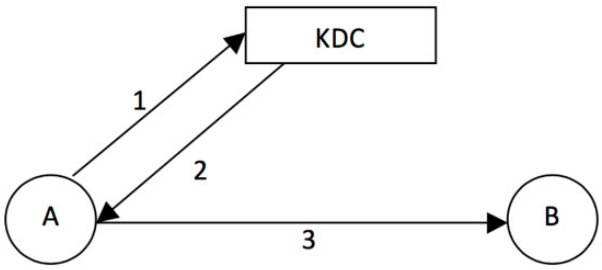
Se puede observar que existe **validación de identidad**, las claves con el *KDC* son secretas, por lo que nadie más habría sido capaz de cifrar la clave secreta K_{AB}, además existe **autenticación** de cada parte involucrada.

Sin embargo existe un **fallo de seguridad**. Si *Mallory* intercepta el canal y captura todos los mensajes de *KDC* a *B*, entonces es posible que *Mallory* cause un **ataque de repetición** (*reply*), y, por consiguiente un **ataque de denegación de servicio** (DoS) sin necesidad de que este derive K_{AB} o K_{BT}. Para resolver lo anterior se puede hacer uso de alguno de los mecanismos existentes:

- Marca de tiempo: incluir en cada mensaje una marca de tiempo (un sello, un entero) de forma que pueda descartar mensajes obsoletos. Problema: los relojes nunca están perfectamente sincronizados en toda una red.
- Nonce/núnico: incluir un número aleatorio único para cada mensaje enviado, de forma que cada parte de la comunicación debe siempre recordar todos los núnicos enviados o recibidos, y rechazar cualquier mensaje que contenga un núnico previamente usado. Problema: si una de las partes pierde la lista de nonce/núnicos, es susceptible a ataques reply.
- Combinación de ambas estrategias: para limitar el tiempo que pueden recordarse los núnicos, pero el protocolo se volverá más complicado.

Modelos genéricos

Modelo PULL



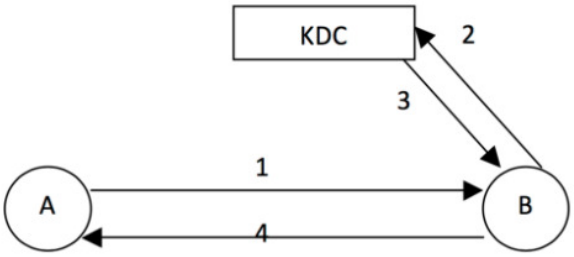
La entidad *A* desea comunicación segura con la entidad *B* por lo que contacta primero con el *KDC* antes de comunicarse con la entidad *B*. El funcionamiento general es:

1. *A* solicita una clave de sesión K_{AB} al *KDC.* El mensaje incluye la identidad de *A*, la identidad de *B* y un valor *N1* (sello de tiempo, valor aleatorio).
2. El *KDC* le contesta a *A* con un mensaje cifrado mediante la clave maestra K_{AT}, de manera que solamente *A* puede leer dicho mensaje y con ello sabe que el *KDC* es el único que pudo haberlo generado. El mensaje contiene la clave K_{AB}, *N1* y un mensaje cifrado para *B* con el K_{AB}.
3. *A* obtiene la información recibida y reenvía le mensaje a *B* para que pueda obtener el K_{AB} también.

A continuación es conveniente establecer un **desafío-respuesta**:

4. *B* utiliza la K_{AB} para cifrar un valor núnico N2 y se lo envía a *A*.
5. *A* recibe el valor N2, le aplica una transformación $f(N2)$, lo cifra con K_{AB} y lo transmite a *B*.

Modelo PUSH

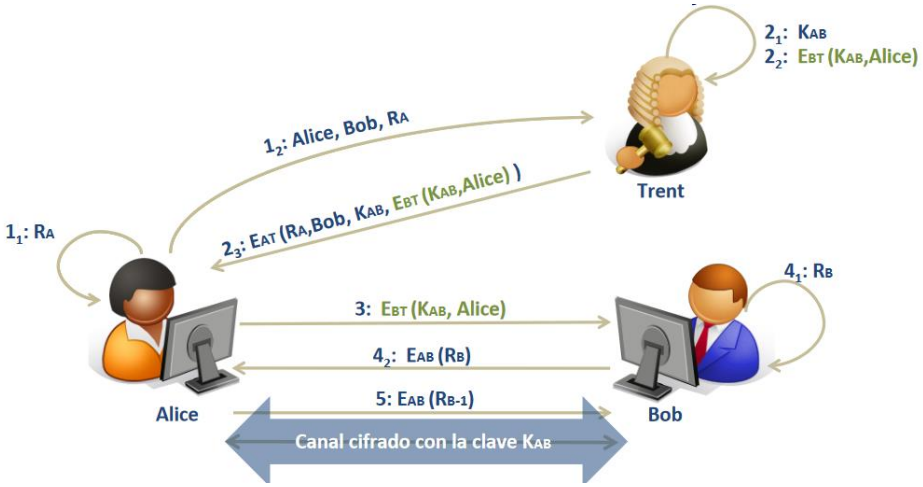


La entidad *A* contacta primero con la entidad *B* a fin de que esta última sea la encargada de solicitar al *KDC* la clave correspondiente. Funcionamiento general:

1. *A* solicita conexión segura con *B*, a *B*. Le manda, como mínimo, su identidad, la identidad de *B* y un nonce.
2. *B* reenvía dicha solicitud a *KDC* para que este genere la K_{AB}.
3. *KDC* verifica las identidades y el *freshnesses* (frescura) de los mensajes, genera la K_{AB}, y dicha información se reenvía a *B* cifrada con la correspondiente K_{BT}.
4. *B* reenvía dicha solicitud a *A* para que obtenga K_{AB}.
5. (opcional) *A* y *B* establecen un desafío respuesta.

Protocolo de Needham-Schroeder

Funcionamiento



El funcionamiento es el siguiente:

1. $A \rightarrow T: A, B, N_A$. *Alice* genera el valor aleatorio N_A y se lo envía a *Trent*.
2. $T \rightarrow A: E_{K_{AT}}\left(N_A, B, K_{AB}, E_{K_{BT}}(K_{AB}, A)\right)$. *Trent* genera la clave de sesión K_{AB}, y se la envía a *Alice*, junto con un mensaje para *Bob*.
3. $A \rightarrow B: E_{K_{BT}}(K_{AB}, A)$. *Alice* envía a *Bob* el mensaje que ella ha recibido de *Trent*.
4. $B \rightarrow A: E_{K_{AB}}(N_B)$. *Bob* genera el valor aleatorio N_B y se lo envía a *Alice* usando la clave K_{AB}. Proceso de **desafío-respuesta** (*challenge-response*).
5. $A \rightarrow B: E_{K_{AB}}(N_B - 1)$. *Alice* responde a *Bob* cifrando el resultado de restar 1 al valor aleatorio que este generó. Proceso de **desafío-respuesta**.

Como podemos observar se trata de un protocolo **tipo PULL**.

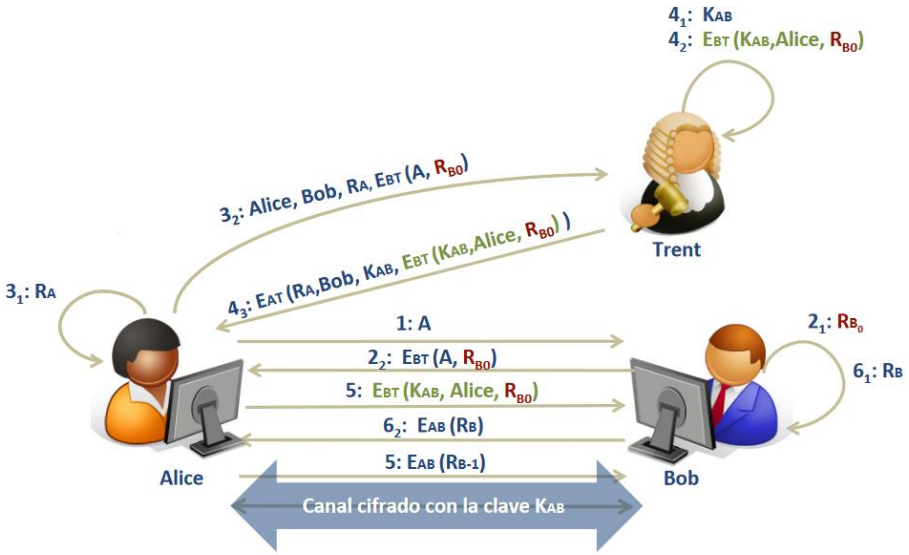
Fallos de seguridad

El protocolo presenta los siguientes fallos de seguridad:

1. *Mallory*, el atacante, puede **suplantar la identidad de *Alice*** si este consigue derivar K_{AT}. *Mallory* copia cualquier mensaje de *Alice* hacia *Trent* en el pasado y **deriva la clave** K_{AT} (suponemos que tiene éxito en su intento)**.** A partir de aquí todos los mensajes quedan comprometidos.
2. *Mallory* puede **suplantar la identidad de *Alice*** si consigue derivar la clave K_{AB}. *Mallory* copia el mensaje de *Alice* hacia *Bob* ($E_{K_{BT}}(K_{AB}, A)$) e intenta descifrar K_{AB} (supongamos que lo consigue). A continuación enviará el mensaje $E_{K_{BT}}(K_{AB}, A)$ a Bob, a lo que Bob responderá con un valor aleatorio (desafío-respuesta). *Mallory* descifrará el valor aleatorio (porque conoce K_{AB}) y responderá al desafío-respuesta. En este punto *Bob* piensa que habla con *Alice*.
3. *Mallory* puede **producir un ataque de DoS** (denegación de servicio) debido a un ataque de repetición (*replay*), especialmente en las últimas fases del protocolo. Esto se debe a que el *freshness* (N_A) solo se encuentra en los mensajes 1 y 2 del protocolo, pero no en el resto. Mallory copiaría el mensaje enviado por *Alice* en el paso 3 y lo enviaría repetidamente a Bob. Solución: extender el uso del nonce a los pasos 3, 4 y 5.

Protocolo Amended Needham-Schroeder

Soluciona el fallo anterior en relación a los ataques de repetición.



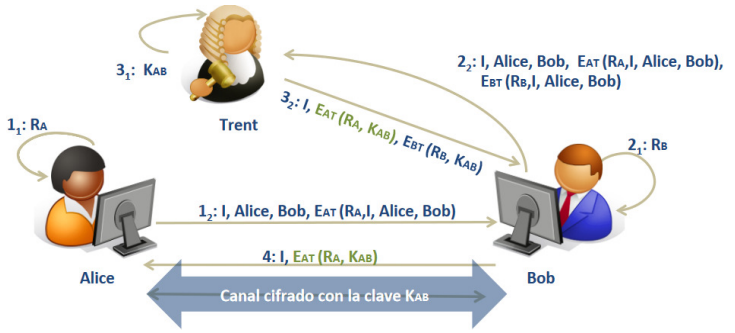
El funcionamiento es el siguiente:

1. $A \rightarrow B: A$. *Alice*, que se quiere comunicar con *Bob*, le envía su dirección.
2. $B \rightarrow A: E_{K_{BT}}(A, N_B)$. *Bob* le envía un mensaje cifrado para *Trent* que contiene la dirección de *Alice* y un núnico.
3. $A \rightarrow T: A, B, N_A, E_{K_{BT}}(A, N_B)$. *Alice* envía a *Trent* su dirección, la dirección de quien quiere comunicar (*Bob*), un núnico y el mensaje cifrado por *Bob* para *Trent*.
4. $T \rightarrow A: E_{K_{AT}}(A, N_A, K_{AB}, E_{K_{BT}}(A, K_{AB}, N_B))$. *Trent* le envía a *Alice* (cifrado con la clave entre ambos) el núnico de *Alice*, la clave de sesión y un mensaje para *Bob*.
5. $A \rightarrow B: E_{K_{BT}}(A, K_{AB}, N_B)$. *Alice* envía a *Bob* en mensaje que le envió *Trent*.
6. $B \rightarrow A: E_{K_{AB}}(N_B)$. *Bob* comienza el desafío-respuesta.
7. $A \rightarrow B: E_{K_{AB}}(N_B - 1)$. *Alice* le responde el desafío-respuesta a *Bob*.

Protocolo Otway-Rees

Soluciona también el fallo de *Needham-Schroeder* aunque con un diseño diferente.

Funcionamiento



El funcionamiento es el siguiente:

1. $A \rightarrow B: I, A, B, E_{K_{AT}}(N_A, I, A, B)$. *Alice* genera un valor aleatorio N_A y se lo envía hacia *Bob*, dentro de un mensaje cifrado con las claves que comparte con *Trent*.
2. $B \rightarrow T: I, A, B, E_{K_{AT}}(N_A, I, A, B), E_{K_{BT}}(N_B, I, A, B)$. *Bob* genera un valor aleatorio N_B y se lo envía a *Trent* usando la clave que comparte con este. También le envía el mensaje que recibió de *Alice*.
3. $T \rightarrow B: I, E_{K_{AT}}(K_{AB}, N_A), E_{K_{BT}}(K_{AB}, N_B)$. *Trent* descifra el mensaje cifrado con la clave que comparte con *Alice*, genera la clave de sesión K_{AB} y se la envía a *Bob* cifrada, junto con un mensaje para *Alice*.
4. $B \rightarrow A: I, E_{K_{AT}}(K_{AB}, N_A)$. *Bob* envía a *Alice* el mensaje que recibió de *Trent* para ella.

Se representa con I la i-ésima sesión establecida entre A y B.

Fallos de seguridad

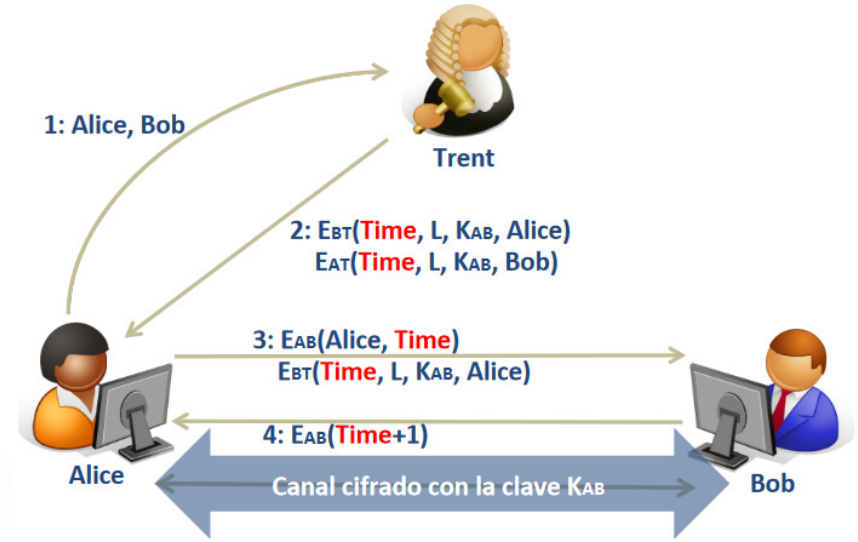
Este protocolo presenta dos fallos de seguridad si la longitud de K_{AB} coincide con la longitud de la concatenación $I|A|B$.

$$length(K_{AB}) = length(I|A|B)$$

1. En el paso 1, si *Alice* en lugar de contactar con *Bob* contacta con *Mallory* (le envía $I, A, B, E_{K_{AT}}(N_A, I, A, B)$), este último podría simular un paso 4 enviando a *Alice* el mismo mensaje (en lugar de $E_{K_{AT}}(N_A, K_{AB})$). En este caso *Alice* pensaría que está hablando con *Bob*, utilizando la clave de sesión $K_{AB} = I|A|B$, que *Mallory* conoce.
2. En el paso 2, si *Bob* en lugar de contactar con *Trent* contacta con *Mallory* (enviándole ambos $E_{K_{AT}}(N_A, I, A, B), E_{K_{BT}}(N_B, I, A, B)$), *Mallory* podría reenviarle los mismos mensajes a *Bob* y *Alice* (a través de *Bob*), que pensarían que se corresponden con: $E_{K_{AT}}(K_{AB}, N_A), E_{K_{BT}}(K_{AB}, N_B)$. De esta forma *Alice* y *Bob* piensan que están hablando por un canal seguro (protegido por K_{AB}), pero esto no es así porque *Mallory* conoce que $K_{AB} = I|A|B$.

Protocolo Kerberos

Funcionamiento



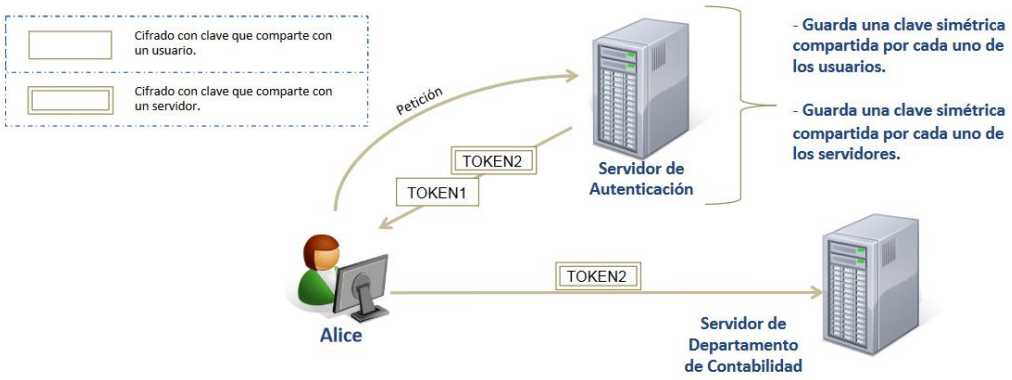
El funcionamiento es el siguiente:

1. $A \rightarrow T$: A, B. *Alice* le envía a *Trent* un mensaje con su identidad y la de *Bob*.
2. $T \rightarrow A$: $E_{K_{BT}}(time, L, K_{AB}, A), E_{K_{AT}}(time, L, K_{AB}, B)$. *Trent* genera un mensaje con un *timestamp* (*time*), un tiempo de vida (*L*), una clave de sesión aleatoria, y la identidad de *Alice*. Lo cifra con la clave compartida con *Bob.* Prepara un mensaje similar para *Alice*. Envía ambos mensajes cifrados a *Alice*.
3. $A \rightarrow B$: $E_{K_{AB}}(A, time), E_{K_{BT}}(time, L, K_{AB}, A)$. *Alice* obtiene K_{AB}, genera un mensaje con su identidad y el *timestamp*, y lo cifra con K_{AB} para enviárselo a *Bob*. También le envía el mensaje cifrado que recibió de *Trent*.
4. $B \rightarrow A$: $E_{AB}(time + 1)$. *Bob* genera un mensaje que costa del *timestamp* más uno, lo cifra con K_{AB} y se lo envía a *Alice*.

Este protocolo asume que los relojes de todos los sistemas están sincronizados con el reloj de *Trent*. En la práctica se sincronizan en el rango de unos pocos minutos. Por fallos del sistema o por sabotaje los relojes podrían desincronizarse.

Ejemplo de uso

Un ejemplo de uso podría ser en un campus universitario o en una empresa privada, donde se usa Kerberos para evitar que cada usuario tenga una cuenta en cada servidor con el que va a contactar. Dispondríamos de un servidor central de autenticación, que guarda una **clave simétrica compartida por cada uno de los usuarios**, y una **clave simétrica compartida por cada uno de los servidores.**



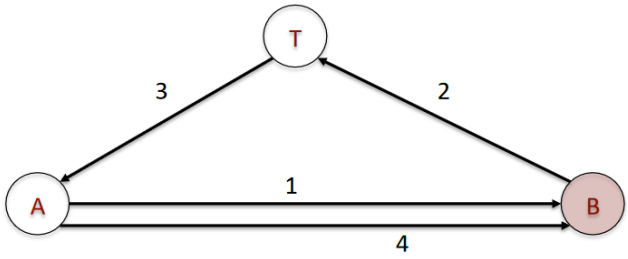
Otros protocolos

A parte de los estudiados existen otros que **combinan diferentes tipos de estrategias**, como los siguientes.

Yahalom

Tiene el objetivo de que *Trent* genere la clave de sesión K_{AB}, y la reenvíe directamente a *Alice* e indirectamente a *Bob*, formando así un modelo **push extendido**.

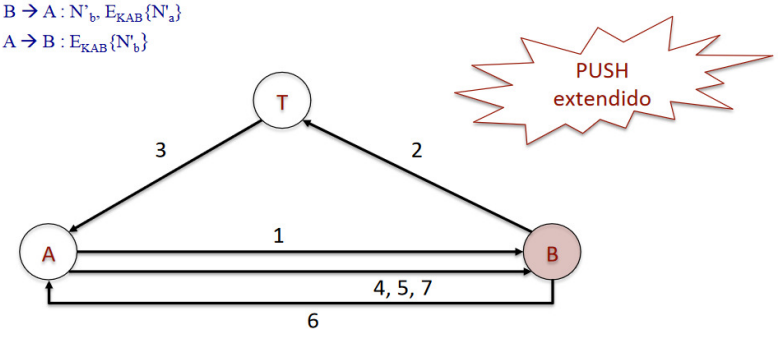
$A \rightarrow B : A, N_a$
$B \rightarrow T : B, E_{KBT}\{A, N_a, N_b\}$
$T \rightarrow A : E_{KAT}\{B, K_{AB}, N_a, N_b\}, E_{KBT}\{A, K_{AB}\}$
$A \rightarrow B : E_{KBT}\{A, K_{AB}\}, E_{KAB}\{N_b\}$



Neuman Stubblebine

Combina diferentes formas de verificar la autenticidad, *timestamps*, núnicos…

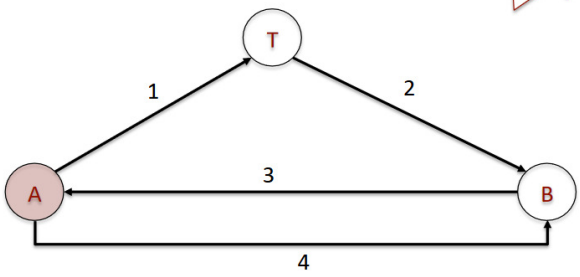
$A \rightarrow B : A, N_a$
$B \rightarrow T : B, E_{KBT}\{A, N_a, \text{time-stamp}_b\}, N_b$
$T \rightarrow A : E_{KAT}\{B, K_{AB}, N_a, \text{time-stamp}_b\}, E_{KBT}\{A, K_{AB}, \text{time-stamp}_b\}, N_b$
$A \rightarrow B : E_{KBT}\{A, K_{AB}, \text{time-stamp}_b\}, E_{KAB}\{N_b\}$
$A \rightarrow B : N'_a, E_{KBT}\{A, K_{AB}, \text{time-stamp}_b\}$
$B \rightarrow A : N'_b, E_{KAB}\{N'_a\}$
$A \rightarrow B : E_{KAB}\{N'_b\}$



Kao Chow

$A \rightarrow T : A, B, N_a$
$T \rightarrow B : E_{KAT}\{N_a, K_{AB}, A, B\}, E_{KBT}\{N_a, K_{AB}, A, B\}$
$B \rightarrow A : E_{KAT}\{N_a, K_{AB}, A, B\}, E_{KAB}\{N_a\}, N_b$
$A \rightarrow B : E_{KAB}\{N_b\}$

PULL extendido



Problemas

Hemos visto que existen diferentes protocolos para solucionar el mismo problema de administración/intercambio de claves. El protocolo a elegir **depende de la arquitectura de comunicaciones subyacente**, es decir de las respuestas a las preguntas:

- ¿Es necesario minimizar el tamaño de los mensajes?
- ¿Es necesario minimizar el número de mensajes?
- ¿Quién ha de contactar primero con el KDC, *Alice* o *Bob*?
- ¿Debe el KDC contactar directamente con ambos, o es suficiente que lo haga con solo uno de ellos?

Además, este sistema no está exento de potenciales problemas:

1. El KDC posee información para **suplantar a cualquier usuario**. Si un intruso llega hasta él, todos los documentos cifrados que circulan por la red se vuelven vulnerables.
2. El KDC representa un **único punto de fallos**. Esto significa que si queda inutilizado nadie podrá establecer comunicaciones seguras dentro de la red.
3. El **rendimiento** de todo el sistema **puede bajar** cuando el KDC se convierte en cuello de botella. Lo cual no es difícil porque todos los usuarios necesitan comunicar con él de forma frecuente con objetivo de obtener las claves.

Conclusiones

Estas conclusiones se extraen de desarrolladores en *Information Security Stack Exchange Q&A.*

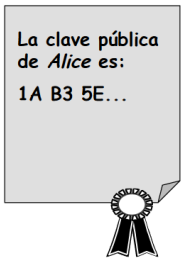
- Utilizar un KDC es más conveniente para **pequeñas infraestructuras**, donde hay confianza en cada persona o nodo que encripta.
- Este método **no escala más allá de infraestructuras locales**, y no es muy **tolerante a fallos**, por ejemplo de denegación de servicios. Por esta razón los navegadores utilizan infraestructura de clave pública y un framework de autoridades de certificación mutuamente confiables.
- Criptográficamente, KDCs y **las claves simétricas son más fuertes que las asimétricas** claves públicas, ya que no hay peligro de que algún día algún matemático loco encuentre una forma de factorizar rápidamente el producto de dos primos elevados.
- Por otro lado, KDCs tienen problemas inherentes con la distribución de las claves, fiabilidad y continua confianza que no pueden ser fácilmente resolubles y por tanto hacen que los KDCs no sean adecuados más allá de infraestructuras locales donde es fácil de asegurar la confianza.

- Por otro lado, SSL utiliza criptografía asimétrica y distribución de claves públicas con certificados, aún así hay entidades que pueden traicionar a otras, las autoridades de certificación. Pero al menos pueden operar offline, lo que evita problemas con la escalabilidad y permite defenderse más fácilmente contra la subversión externa.
- SSL utiliza **criptografía asimétrica porque es más fácil aplicarla genéricamente** (especialmente a gran escala) y también más fácil de mantener segura.

Mecanismos e infraestructuras de administración de claves públicas

En una infraestructura de clave pública el problema de: ¿cómo sabe *Bob* si la clave pública de *Alice* es **genuina** (que conserva autenticidad)? ¿cómo sabe *Alice* si la clave pública de *Bob* es genuina? Estas preguntas son equivalentes a preguntarse: ¿cómo garantizar que las claves públicas de *Alice* y *Bob* son auténticas?

Veamos como ejemplo el caso en que *Bob* necesita la clave pública de *Alice*; *Bob* necesitaría algún documento digital con algún **sello de garantía**, o sea, algo equivalente a:



En realidad la **información relevante** que debería contener ese documento digital sería:

- La **identidad del usuario** al respecto del cual se ofrece información.
- El valor de la **clave pública** de Alice.
- Algo que identifique unívocamente a ese documento entre otros muchos (por ejemplo un **número de serie**). En este caso la identidad del usuario no serviría, porque puede tener más de un par <clave pública, clave privada>.
- Algo que indique "desde" y "hasta" cuándo es válido el documento digital, una fecha de **emisión** y una fecha de **expiración**.
- La identidad de **quien emite** el documento.
- La **firma digital** de quien emite el documento.

El documento digital con esa información se denomina **certificado digital**, o **certificado de clave pública**. Es la **firma digital** de un documento (que contiene la información antes mencionada) que garantiza que cierta clave pública pertenece a un determinado usuario.

Se denomina **Autoridad de Certificación (CA** *Certificate Authority***)** a la tercera parte confiable (TTP) que emite y administra los certificados digitales de los usuarios de un sistema. Garantiza que una clave pública pertenece a cierto usuario inequívocamente identificado.

Estándar Certificado X-509

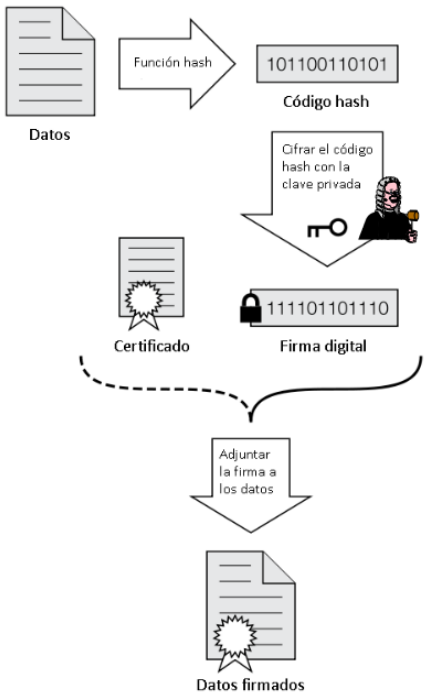
La ITU-T ha definido una estructura estándar de certificado digital adoptada mundialmente: certificado X.509. Un certificado digital debe especificar:

1. **Versión**: versión de X.509, o sea 1, 2 o 3.
2. **Número de serie**: número de identificación único para este certificado digital, asignado por la CA.

3. **Algoritmo de firma**: identificador del algoritmo de firma digital usado por la CA para firmar el certificado.
4. **Emisor**: nombre X.500 de la CA emisora.
5. **Periodo de validez**: fecha desde que el certificado comienza a ser válido, y día y hora de la expiración.
6. **Sujeto**: nombre X.500 del usuario cuya clave pública se está certificando.
7. **Algoritmo de clave pública**: identificador del algoritmo de clave pública con el que se ha de utilizar la clave pública.
8. **Clave pública**: valor de la clave pública.
9. **Identificador único de emisor**: opcional para que el nombre de la CA no sea ambiguo en caso de que pueda ocurrir (versión 2).
10. **Identificador único de usuario**: opcional para que el nombre del usuario no sea ambiguo en caso de que pueda ocurrir (versión 2).
11. **Extensiones**: campo opcional para almacenar información de distinto tipo (versión 3).
12. **Firma**: firma digital de la CA sobre el valor hash del conjunto de los demás campos del certificado.

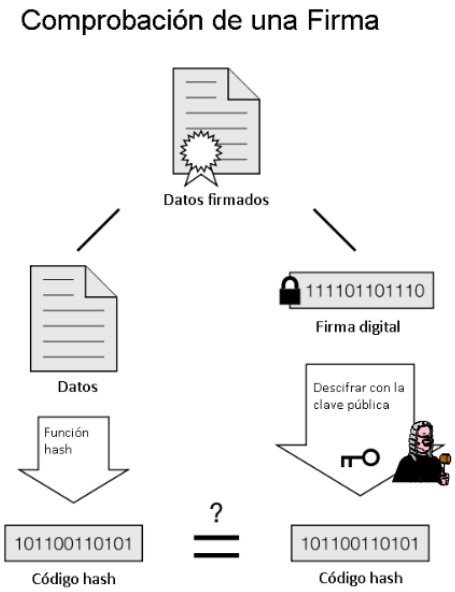
Proceso de firma y comprobación

El proceso de **firma** es el siguiente:



A los datos se le aplica una función hash, ese hash se firma con la clave privada de la CA y se obtiene una firma digital. Al certificado se le **adjunta** la firma del hash y ya disponemos de un **certificado digital**.

El proceso de **comprobación** es el siguiente:

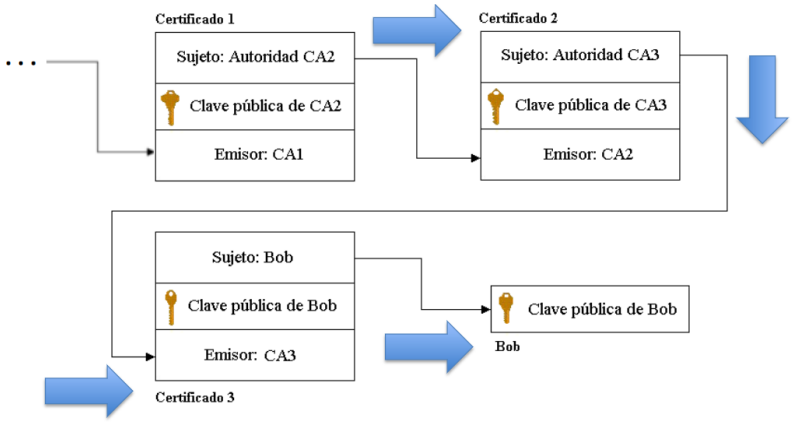


Del certificado digital se separan los datos y la firma digital. A los datos se le aplica el mismo algoritmo de hash que en la firma y se obtiene un código hash. La firma digital se descifra con la clave pública de la CA y se obtiene el hash proporcionado por la CA. Si ambos valores coinciden la firma es válida.

Infraestructura de clave pública

La situación ideal sería que una única CA pudiera certificar a todos los usuarios de Internet. Sin embargo, la situación real es bien distinta, dado que existe una gran multiplicidad de grupos de usuarios en Internet, y distribuidos geográficamente, lo que implica la necesidad de múltiples CAs.

Entre las Cas se utiliza de forma recursiva el esquema de certificación, creándose **cadenas de confianza** (o **cambios** de **certificación**).



Estas cadenas de confianza se forman gracias a la infraestructura de CAs, denominada **Infraestructura de Clave Pública** (**PKI** *Public Key Infraestructure*). Una PKI proporciona el **marco subyacente** que permite la implantación de la tecnología de clave pública. Servicios ofrecidos por una PKI:

1. Emisión de certificados.
2. Distribución de certificados.
3. Obtención de certificados.
4. Certificación cruzada.
5. Generación de claves.
6. Actualización de claves.

7. Salvaguarda y recuperación de claves.
8. Revocación y suspensión de certificados.

Revocación de certificados

A veces, puede ser recomendable **invalidar** (revocar) un certificado **antes de la fecha de expiración**, cuando:

- La clave pública deja de ser válida.
- El usuario identificado en el certificado no se considera por más tiempo un usuario con potestad sobre la clave privada correspondiente.
- Varía la información dentro del certificado.

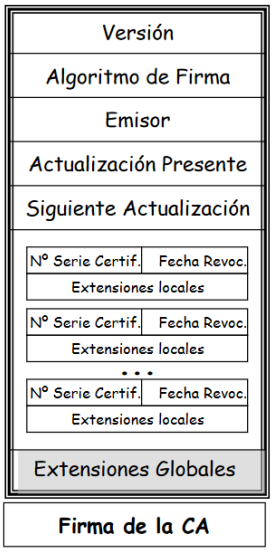
La **CA se encarga de hacer la revocación**, bajo petición del usuario; por esto ha de **publicar** esa información acerca del estado el certificado para que el resto de usuarios puedan realizar la comprobación antes de usarlo.

La comunidad Internet y la ITU-T han desarrollado el concepto de **Lista de Revocación de Certificados, (CRL** *Certificate Revocation List*), como mecanismo de revocación. Una CRL es una lista (con *timestamping*) de certificados revocados, **firmada por la autoridad que emitió los certificados.**

Un **escenario típico de uso** sería: para que *Bob* verifique la firma de *Alice* sobre un documento digital, no solo ha de verificar el certificado de *Alice* y su validez, además ha de comprobar que ese certificado no está en la CRL. O sea, ha de adquirir la versión más reciente de la CRL y confirmar que el número de serie del certificado de *Alice* no está en tal CRL.

Una CA emite CRLs regularmente (cada hora, día, semana…) con independencia de que se hayan producido nuevas revocaciones. El intervalo de emisión de CRLs depende de la política de certificación de la CA. El certificado **se borra de la CRL cuando alcanza la fecha de expiración** (en su caducidad natural).

La estructura de una CRL según el estándar X.509 es la siguiente:



X.509 CRL versión 2

El protocolo ***Online Certificate Status Protocol*** **(OCSP)** es otra solución a la revocación. Define un **formato estándar** para **peticiones y respuestas.** Su funcionamiento se basa en que **un usuario puede confirmar online el status de un certificado** mediante la ejecución de una transacción con un servidor (*responder*) OCSP asociado a la CA. La CA debe poner a disposición de todos los usuarios potenciales

un **servicio online de alta disponibilidad**, y además, el servicio ha de proporcionarse dentro de un entorno seguro. El *OCSP responder* puede ser o bien la misma CA o alguna entidad autorizada por ella.

Tarjetas inteligentes

Una **tarjeta inteligente** o *smartcard* es una tarjeta que incluye un chip cuya función puede ser variada, desde simplemente **almacenar** cierta información en su memoria interna (con o sin medidas de protección), hasta **realizar complejos cálculos criptográficos** y encargarse de proteger el acceso a las claves que almacena. Su uso se extiende hoy a muchos sectores: tarjetas de fidelización, tarjetas bancarias, tarjetas de parking, documentos de identificación (DNI electrónico o pasaporte electrónico).

Si atendemos al **método de comunicación** o interfaz con el circuito integrado, las *smartcards* se clasifican en: tarjetas **de contacto**, tarjetas **sin contacto**.

Si atendemos a **capacidades del chip**, se clasifican en:

- **Tarjetas de memoria**, solo contienen datos y no albergan aplicaciones. Usadas para identificación y control de acceso sin altos requisitos de seguridad.
- **Tarjetas microprocesadas,** albergan datos y aplicaciones. Usadas para pago con monederos electrónicos.
- **Tarjetas criptográficas:** tarjetas microprocesadas avanzadas que incluyen módulos hardware para la ejecución de cifrados y firmas digitales. Usadas para almacenar de forma segura un certificado digital (y su clave privada), así como firmar documentos o autenticarse. El procesador de la tarjeta realiza la firma.

DNI Electrónico (DNI-e)

El **DNI** electrónico, a través de las capacidades criptográficas que aporta permite:

- **Identificación** en medios telemáticos.
- **Firmar** electrónicamente.

Chip

Está dotado con el chip ST19WL34 (STMicroelectronics), compuesto por:

- Microprocesador seguro de 8 bits.
- 6Kb de memoria RAM.
- 224KB de memoria ROM para el almacenamiento del sistema operativo y código de programas.
- 34KB de memoria EEPROM para el **almacenamiento de datos personales** con tecnología de almacenamiento fiable y código de corrección de errores.

Este chip ofrece una retención de datos de al menos 10 años, y una resistencia de 500.000 ciclos de borrado y escritura. También se caracteriza por incorporar:

- Procesador aritmético modular (MAP) de 1088 bits para criptografía de clave pública.
- Motor de aceleración por hardware de los algoritmos DES y triple-DES.
- Módulo para el cálculo de funciones CRC.
- Interfaz de entrada/salida serie.
- Generador de números aleatorios.
- Bus de interconexión interno.
- 3 timers de 8 bits.
- Reloj interno.

Algunas medidas del tiempo de ejecución de operaciones criptográficas son las siguientes:

Function	Speed[1]
RSA 1024 bits signature with CRT[2]	85 ms
RSA 1024 bits signature without CRT[2]	282 ms
RSA 1024 bits verification (e='$10001')	5.5 ms
RSA 1024 bits key generation	2.5 s
RSA 2048 bits signature with CRT[2]	570 ms
RSA 2048 bits verification (e='$10001')	91 ms
Triple DES (with enhanced security)	58.0 μs
Single DES (with enhanced security)	43.0 μs

El **sistema operativo** que gestiona el chip se denomina DNIe v3.0, desarrollado por la FNMT a partir de las especificaciones funcionales de la Dirección General de Policía. Este sistema operativo ha sido sometido con posterioridad a los perfiles de protección de la certificación *Common Criteria*.

CAN (*Card Access Number*) es un número que aparece en la parte inferior del DNI 3.0, y corresponde con el número de la tarjeta como medida de seguridad.

Componentes

El DNI-e contiene dos certificados digitales asociados al titular:

- **Certificado de autenticación**: asegura que la comunicación electrónica se realiza con el titular del DNI, pero no demuestra voluntad de firma. Restringido a operaciones para **confirmar la identidad y acceso seguro a sistemas** remotos.
- **Certificado de firma digital**: para la firma de documentos, garantizando la integridad del documento y el no repudio de origen.

Cuenta también con un **certificado de componente**, emitido para autenticar al propio chip y cifrar la comunicación con él, de forma similar a como se utiliza un certificado TLS en un servicio Web.

El generador interno de números aleatorios origina el par de claves de cada certificado, en presencia del ciudadano: se garantiza que solo existirá una copia de cada clave privada, y que esta residirá siempre en el interior del chip.

La información de la **memoria EEPROM** del chip está distribuida en tres zonas, con diferentes niveles y condiciones de acceso. Los tres son solo accesibles para realizar **operaciones de lectura**, no siendo posible para el ciudadano escribir o grabar datos.

- **Zona pública** accesible sin restricciones.
    - Certificado CA emisora
    - Claves Diffie-Hellman
    - Certificado X.509 de componente
- **Zona privada** accesible por el ciudadano mediante la utilización de su PIN
    - Certificado de autenticación (identificación).
    - Certificado de firma.
- **Zona de seguridad** accesible por el ciudadano de forma exclusiva en los puntos de actualización del DNI-e (comisarías).
    - Datos de filiación del ciudadano
    - Fotografía del titular
    - Imagen de la firma manuscrita

Claves criptográficas

Cualquier operación criptográfica que requiera el uso de una de las claves privada debe ser ejecutada en el interior del chip. Las claves públicas se envían, tras su generación en el acto de expedición del

DNI-e, a la CA para su inclusión en los correspondientes **certificados digitales**. Una vez emitidos los certificados, estos se incorporan a la tarjeta para ser empleados en operaciones posteriores. Los certificados digitales pueden ser leídos para su proceso de forma externa al chip.

Revocación

En el ámbito del DNI-e se utiliza **OCSP** para las revocaciones. Cuando una aplicación requiere el estado actual de un certificado, envía una petición OCSP (mediante HTTP), a la URL del servicio de validación. Una vez recibida la petición, el *OCSP Responder* accede a las CRLs, y averigua si dicho certificado se encuentra ahí incluido.

En la PKI adoptada para el DNI-e se ha optado por asignar las funciones de **Autoridad de Validación** a entidades diferentes de la **Autoridad de Certificación**, con el fin de aislar la comprobación de la vigencia de un certificado. Existen tres autoridades: FNMT, Ministerio de Administraciones Públicas, Ministerio de Industria.

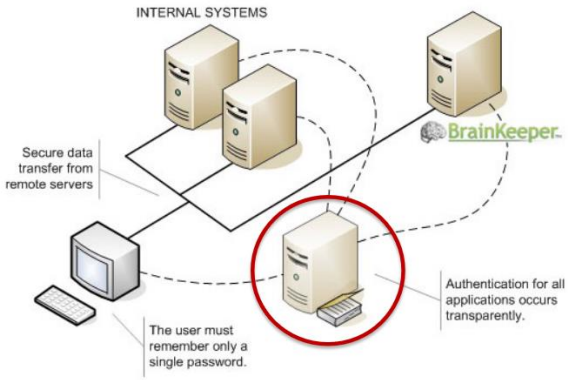
Mecanismos de *Single Sign-On* para Autenticación

El ***Single Sign-On*** es un mecanismo que permite a un usuario **autenticarse una sola vez** para acceder a todos los sistemas, independientes pero relacionados, a los que tiene acceso. Una vez autenticado, **el usuario puede ir cambiando de un sistema a otro** sin necesidad de autenticarse de nuevo.

Existen diferentes **ventajas**:

- **Usabilidad**: el usuario solo ha de recordar una contraseña, o usar un solo token, o un solo certificado… Reduce la probabilidad de error humano.
- **Seguridad**: reduce el riesgo de los ataques de intercepción.
- **Productividad**: reduce el tiempo de autenticación.

Aun así presenta una **desventaja**, y es que hay un único punto de ataques, el servidor SSO. Además, el intruso podrá entrar en todos los sistemas si su ataque tiene éxito aunque sea una vez.



Mecanismos de control de acceso

El **control de acceso** es un elemento central, uno de los esenciales, de la **Seguridad de Ordenadores**. El RFC-2828 define la **Seguridad en Ordenadores** como: "Medidas que implementan y aseguran los servicios de seguridad en un sistema informático, particularmente aquellas que aseguran el servicio de control de acceso". Los objetivos principales del control de acceso son:

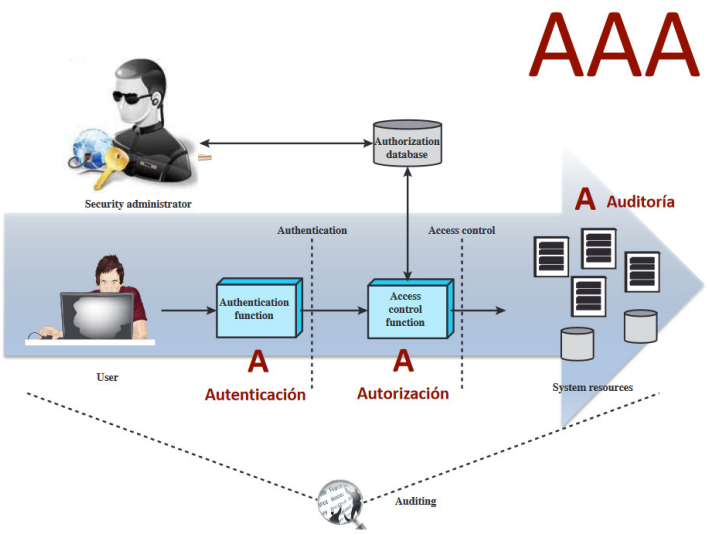
- Prevenir los accesos a los recursos por parte de usuarios no autorizados.
- Prevenir que los usuarios legítimos accedan a los recursos de forma no autorizada.
- Permitir a los usuarios legítimos acceder a los recursos de una forma autorizada.

Por tanto, el control de acceso implementa una **política de control de acceso**, que especifica:

- Quién o qué puede tener acceso a cada recurso del sistema.
- El tipo de acceso que se permite.

Existe una relación clara entre el control de acceso y otros servicios de seguridad, concretamente con los servicios de **autenticación, autorización y auditoría** (AAA).

- **Autorización**: concesión de un derecho o un permiso a una entidad para acceder a un recurso.
- **Auditoría:** revisión de los registros y actividades del sistema para:
    - Garantizar el cumplimiento de la política y los procedimientos operacionales.
    - Recomendar cambios en la política y en los procedimientos.
    - Comprobar la adecuación de los sistemas de control.
    - Detectar problemas de seguridad.

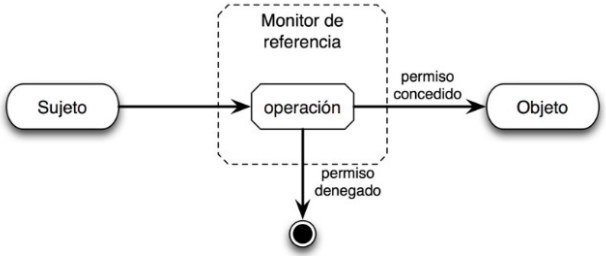


Como se puede observar en la figura anterior, el mecanismo de control de acceso hace de **mediador entre un usuario** (o un proceso) y **los recursos del sistema** (aplicaciones, sistemas operativos, firewalls, routers, ficheros, bases de datos, dispositivos concretos…). La figura anterior muestra un modelo simple de control de acceso, pero en la práctica puede haber muchos componentes que, de forma cooperativa, comparten la función de control de acceso.

Los elementos básicos de un control de acceso son:

- **Objeto**: recurso al cual se controla el acceso. Ejemplos: registros, páginas, segmentos, ficheros…

- **Sujeto**: entidad que potencialmente accede a los objetos. Generalmente el concepto de sujeto se asimila al de **proceso**; cualquier usuario o aplicación consigue el acceso a un objeto a través de un proceso que lo representa.
- **Derecho de acceso**: describe la forma en que el sujeto podría acceder al objeto: *read, write, execute, delete, create...*



Los esquemas de control de acceso se dividen principalmente en varias categorías:

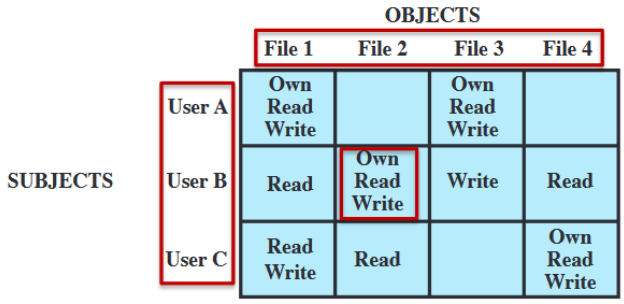
- **DAC** (*Discretionry Access Control*), se basa en:
  - Identidad del solicitante.
  - Reglas de acceso (solicitantes autorizados o no).
- **MAC** (*Mandatory Access Control*), se basa en comparar:
  - Etiquetas de seguridad (que indican criticidad de recursos).
  - Autorizaciones de seguridad (que indican entidades que pueden acceder a ciertos recursos).
- **RBAC** (*Role-Based Access Control*), se basa en:
  - Rol que tiene cada usuario dentro del sistema.
  - Reglas que indican qué accesos están permitidos a quien posee un determinado rol.
- **ABAC** (*Attribute-Based Access Control*), se basa en:
  - Atributos asociados con el usuario y que dependiendo del atributo se permite o no el acceso a un sistema.

Estas políticas **no son mutuamente exclusivas**, un mecanismo de control de acceso puede usar dos, tres, o incluso todos los mecanismos para cubrir diferentes tipos de recursos del sistema.

DAC (*Discretionary Access Control*)

DAC se basa en la **identidad del solicitante** y en las **reglas de acceso** que indican qué solicitantes están autorizados.

La **matriz de acceso** es una solución general para DAC, tal y como ocurren en los sistemas operativos y en sistemas de operación de bases de datos. Una dimensión está formada por los **sujetos** (usuarios individuales, grupos de usuarios, equipos de la red, hosts…) que potencialmente acceden a los recursos; la otra dimensión de la matriz está formada por los **objetos** (campos individuales de datos, registros, ficheros, bases de datos…) que se podrían acceder. Cada entrada de la matriz indica los **derechos de acceso** del sujeto para ese objeto.



En la práctica la matriz de acceso se descompone en dos partes:

1. **Access Control List** (*ACL*): resultado de la descomposición por columnas. Por cada objeto una ACL lista los usuarios y sus correspondientes derechos de acceso.

$$L_{bar.txt} = \{(pepe, \{r\}), (paco, -), (luis, \{r, d\})\}$$

$$L_{foo.txt} = \{(pepe, -), (paco, \{x, d\}), (luis, \{x\})\}$$

   - Ventajas:
     - Fácil ver los permisos de acceso de un determinado objeto.
     - Fácil revocar todos los permisos sobre un objeto, poniendo $L_{obj} = \{\}$.
     - Fácil eliminar los permisos asociados a un objeto que ya no existe, por simplemente eliminar L_{ob}.
   - Desventajas:
     - Comprobar permisos de acceso de un determinado sujeto, usabilidad
   - Uso:
     - Se suelen implementar en sistemas orientados a la gestión de recursos, como los sistemas operativos.

2. **Ticket de capacidades** (o perfil de acceso): descomposición por filas. Especifica los objetos autorizados y las operaciones para cada usuario.

$$L_{pepe} = \{(bar.txt, \{r\}), (foo.exe, -)\}$$

$$L_{paco} = \{(bar.txt, -), (foo.exe, \{x, d\})\}$$

$$L_{luis} = \{(bar.txt, \{r, d\}), (foo.exe, \{x\})\}$$

   - Ventajas:
     - Fácil ver los permisos de acceso de un sujeto.
     - Fácil revocar todos los permisos de un sujeto, poniendo $L_{sj} = \{\}$.
     - Fácil eliminar los permisos asociados a un sujeto que ya no existe, por simplemente eliminar L_{sj}.
   - Desventajas:
     - Comprobar permisos de acceso de un determinado objeto, usabilidad
   - Uso:
     - Se suelen implementar en sistemas orientados al usuario, como bases de datos o sistemas distribuidos.

La **Tabla de autorización** es una alternativa a la matriz de acceso. Contiene una fila por cada derecho de acceso a un recurso. Es de uso más ágil, en comparación con las ACLs o los tickets de capacidades.

MAC (*Mandatory Access Control*)

Se basa en comparar **etiquetas de seguridad** (que indican la criticidad de los recursos) con las **autorizaciones de seguridad** (que indican las entidades que pueden acceder a ciertos recursos). Por tanto, a cada recurso **se le asigna una etiqueta de seguridad,** que de alguna forma lo clasifica (ejemplos: *top secret, secret, confidential, restricted, unmarked, unclassified*).

RBAC (*Role Based Access Control*)

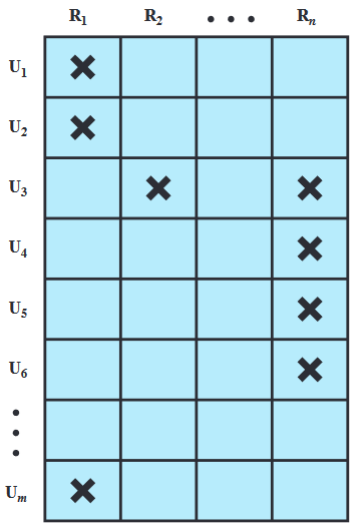
En este modelo se considera que un **rol** es una función o tarea que se lleva a cabo dentro de una empresa u organización. **No se basa en la identidad**, sino en los roles que asumen los usuarios; por esto **los derechos se asignan a los roles**, y luego se asignan usuarios a dichos roles.

El **funcionamiento** se fundamenta en que:

- El conjunto de roles de un sistema, aún pudiendo ser complejo, es relativamente **estático** en la mayoría de los escenarios.
- El conjunto de recursos y los derechos de acceso específicos asociados a un rol particular **tampoco cambian con frecuencia**.

RBAC tiene un **uso comercial** bastante amplio hoy en día, y por ello ha sido estandarizado por el NIST en el documento *Security requirements for cryptographic modules.*

Podemos utilizar la representación de **matriz de acceso** para una representación simple de los elementos clave en un sistema RBAC. En primer lugar tendremos una matriz que indiqué a qué rol pertenece cada usuario:



La matriz de acceso tendría la forma:

	OBJECTS								
ROLES	R_1	R_2	R_n	F_1	F_1	P_1	P_2	D_1	D_2
R_1	control	owner	owner control	read *	read owner	wakeup	wakeup	seek	owner
R_2		control		write *	execute			owner	seek *
. . .									
R_n			control		write	stop			

Este modelo consta de **4 tipos de entidades:** usuario, permiso, sesión y rol. Cada **sesión** es un mapeo de cada usuario a los posibles roles. Las relaciones *many-to-many* entre usuarios y roles y entre roles y permisos proporcionan una **flexibilidad y granularidad** de asignación no proporcionada por DAC. Además, RBAC permite definir:

- **Roles mutuamente exclusivos:** es una restricción de tal forma que un usuario solo se puede asignar a **uno** de los roles del conjunto. Esta limitación puede ser estática o dinámica en una sesión.
- **Cardinalidad**: establecimiento de un **número máximo** con respecto a los roles. Ejemplos:
    - Número máximo de usuarios que se pueden asignar a un rol.
    - Número de roles asignados a un usuario.
    - Número de roles que un usuario puede tener en una sesión.
    - Número de roles que se puede conceder a un permiso particular.
- **Prerrequisitos:** ejemplo: a un usuario solo se le puede asignar un rol si ya está asignado en otro específico.

El modelo RBAC estándar de NIST introduce dos extensiones:

- **SSD *Static Separation of Duties***: para definir roles mutuamente excluyentes.
- **DSS *Dynamic Separation of Duties*:** para definir restricciones sobre los roles que un usuario puede activar en una sesión.

ABAC (*Attribute Based Access Control*)

El acceso no está basado en permisos del usuario, sino en los **atributos del usuario**. Esto permite incluso el **acceso anónimo**, si la identificación y autenticación no es estrictamente necesaria.

Otros

Existen otros mecanismos de control de acceso menos utilizados o relevantes.

CapBAC (*Capability-Based Access Control*)

CapBAC basa su modelo en un conjunto de roles y atributos, en donde el acceso solo es posible si el usuario recibe del proveedor del recurso un **token** ("certificado de autorización") que demuestra su "capability" para realizar determinadas acciones sobre dicho recurso. Por tanto, si un usuario desea acceder a un recurso, solo debe mostrar su certificado de autorización al proveedor antes de solicitar la operación. La principal desventaja es que **requiere mantener** todos los certificados de autorización.

Risk-Based Access Control

Fue diseñado para escenarios **heterogéneos** compuestos de múltiples tipos de organizaciones funcionando con múltiples y diversas políticas de seguridad, y en donde **no es posible predecir el número real de usuarios y recursos**. Para gestionar esto, el modelo debe requerir de gestores y/o algoritmos funcionando en tiempo real.

El acceso depende del riesgo, que se evalúa:

$$R = V \cdot P$$

V es el **valor de información** (criticidad o susceptibilidad del recurso demandado) que se puede computar de acuerdo al grado de disponibilidad, nivel de confidencialidad, integridad…

P representa la **probabilidad del acceso** cuyo valor puede ser determinado por estudiar previamente un conjunto de escenarios amenazantes, las políticas de seguridad, los niveles de seguridad…

OrBAC (*Organizational-Based Access Control*)

Extiende y mejora el uso de RBAC de la siguiente forma:

- **Sujetos:** entidades con roles predefinidos y conteniendo específicos permisos de seguridad.
- **Actividades:** conjunto de acciones a realizar en una organización bajo una misma política de seguridad.
- **Vistas:** relacionados con los objetos y su acceso, todos ellos funcionando sobre políticas de seguridad preestablecidas por la organización.

Sin embargo, este modelo depende de las políticas de seguridad implantadas y del nivel de confianza de cada entidad implicada.

Protocolos criptográficos avanzados

Un **protocolo criptográfico** es un algoritmo **distribuido** definido para **alcanzar un objetivo** específico de seguridad. En estos algoritmos los objetivos no serán los de antes (confidencialidad, integridad, control de acceso…) sino otros, por lo que no se aplicará la criptografía como la conocemos hasta ahora. Consta de una secuencia de pasos que especifican, de forma precisa, las acciones a llevar a cabo por parte de dos o más entidades. Se utilizarán las **primitivas criptográficas** que sustentan el diseño de protocolos criptográficos.

En realidad ya hemos comentado algunos, por ejemplo en la administración/intercambio de claves o autenticación de entidades.

División de secretos

Se usa cuando hay que dividir un mensaje M en n trozos. Cada trozo, por sí mismo, no tiene valor, pero cuando se ponen en conjunto se recupera el mensaje original M. Es necesario utilizar una tercera parte confiable, TTP.

Entre 2 usuarios

La división la realiza Trent que genera dos trozos, uno para Alice, otro para Bob.

1. Trent genera una cadena aleatoria de bits R de la misma longitud que el mensaje.
2. Hace XOR de R con M: $R \oplus M = S$.
3. Trent distribuye R a Alice y S a Bob.

Para reconstruir el mensaje Alice y Bob tienen que ponerse de acuerdo y realizar $R \oplus S = M$. En realidad Trent está cifrando el mensaje con un **OTP (One Time Pad)** y proporciona el texto cifrado a una persona y el pad a otra.

Entre más de 2 usuarios

En este caso supondremos n usuarios, por lo que Trent genera n trozos.

1. Trent genera $(n-1)$ cadenas aleatorias de bits R_i de la misma longitud que el mensaje.
2. Hace XOR de R_i con M: $R_1 \oplus R_2 \oplus \ldots \oplus R_{n-1} \oplus M = S$.
3. Trent distribuye a los n usuarios las cadenas R_i y S.

Para reconstruir el mensaje M todas las partes se deberán poner de acuerdo y proporcionar su trozo de información. El **problema** de división de secretos es que si una parte se pierde, entonces el mensaje no se puede recuperar.

Compartición de secretos

Este protocolo se basa en el concepto de esquema umbral (k-n). Consiste en que se divide un mensaje M en n trozos, llamados **sombras**, de forma que con k de ellos se puede reconstruir el mensaje original. Los esquemas umbrales son incluso más versátiles.

El fundamento matemático es una **interpolación lineal**: dados n puntos hay uno y solo un polinomio $g(x)$ de grado $k-1$ tal que: $\forall i: q(x_i) = y_i$.

El **funcionamiento** es:

- Se divide el dato D (mensaje M como un número) en n trozos de forma que D es fácilmente reconstruible a partir de k trozos cualesquiera.
- Incluso el conocimiento de $k-1$ trozos no revela ninguna información sobre D.

Más concretamente:

- Se eligen aleatoriamente los coeficientes de un polinomio de grado $k-1$:
$$q(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$
- Al término independiente a_0 se le asigna el valor del bloque de mensaje D.
$$a_0 = D_0 = D$$
- Se evalúa el polinomio para cada usuario y se le envían los valores D_i.
$$D_1 = q(1); D_2 = q(2); D_3 = q(3); \ldots D_n = q(n)$$
- A partir de esos k valores D_i, se podrían encontrar por interpolación los coeficientes de $q(x)$ (una ecuación de k incógnitas con k ecuaciones) y entonces evaluar $D = q(0)$.

Como se ha mencionado, el conocimiento de $k-1$ de esos valores no es suficiente.

Bit-commitment

El problema general de este tipo de protocolos es: Alice quiere **realizar una predicción** (apuesta), pero **no revelarla hasta después de producirse el hecho**. Bob, por otro lado, quiere asegurarse de que Alice no puede cambiar su predicción después de que el hecho se ha producido.

Solución criptografía simétrica

1. $B \rightarrow A: R$. Bob genera aleatoriamente una cadena R de bits y la envía a Alice.
2. $A \rightarrow B: E_K(R, b)$. Alice crea un mensaje con el bit b a dejar en compromiso y la cadena aleatoria de Bob. Lo cifra con una clave simétrica K y envía el resultado a Bob.

Alice ha realizado el compromiso (apuesta) y Bob no puede descifrar el mensaje, así que no puede conocer el bit. Cuando llega la hora de que Alice revele el bit el protocolo continúa.

3. $A \rightarrow B: K$. Alice le envía a Bob la clave K con que cifró el mensaje.
4. B: $D_K\big(E_K(R, b)\big)$. Bob descifra el mensaje para obtener el bit y chequea su cadena aleatoria para verificar la validez.

Funciones hash

1. $A: (R_1, R_2, b)$. Alice genera aleatoriamente dos cadenas de bits R_1, R_2 y crea un mensaje que consta de las dos cadenas y el bit de compromiso.
2. $A \rightarrow B: H(R_1, R_2, b), R_1$. A continuación computa la función hash de ese mensaje y envía el resultado a Bob, junto con una de las cadenas aleatorias.

Esta transmisión es la evidencia del compromiso. La función hash en el paso 3 previene que Bob pueda invertir la función y determinar el bit. Cuando Alice tiene que revelar el bit el protocolo continúa.

3. $A \rightarrow B: R_1, R_2, b$. Alice envía a Bob el mensaje original.
4. $B: H(R_1, R_2, b)$. Bob computa la función hash del mensaje y compara esta y R_1 con el valor y la cadena aleatoria recibida en el paso 3. Si coincide es válido.

Protocolo de lanzamiento de moneda

Supongamos que Alice y Bob desean hacer un "cara o cruz" sin encontrarse presencialmente. Uno de ellos hace el lanzamiento de moneda, pero el otro no lo ve. En general necesitamos un protocolo con las propiedades:

- Alice debe lanzar la moneda antes deue Bob se pronuncie.
- Alice no debe ser capaz de re-lanzar la moneda después del pronunciamiento de Bob.
- Bob no debe saber de qué lado cayó la moneda antes de pronunciarse.

Podemos solucionar este problema utilizando un protocolo de **compromiso de bit**, también se puede solucionar utilizando criptografía de clave pública.

Para poder realizar este protocolo ha de cumplirse: $D_{K1}(E_{K2}(E_{K1}(M))) = E_{K2}(M)$.

1. $A: K_{Apub}, K_{Apriv}$; $B: K_{Bpub}, K_{Bpriv}$. Alice y Bob generan cada uno su par clave pública y privada.
2. $A \rightarrow B: E_{Apub}(M_1), E_{Apub}(M_2)$. Alice genera dos mensajes, uno indicando "cara" y otro "cruz". Estos mensajes contienen además alguna cadena aleatoria para verificar posteriormente su autenticidad.
3. $B \rightarrow A: E_{Bpub}(E_{Apub}(M))$. Bob, que no puede leer los mensajes elige uno y lo cifra con su clave pública y lo devuelve a Alice.
4. $A: D_{Apriv}(E_{Bpub}(E_{Apub}(M))) = E_{Bpub}(M)$; $A \rightarrow B: E_{pubB}(M)$. Alice, que no puede leer el mensaje que Bob le ha devuelto, lo descifra con su clave privada y lo devuelve a Bob.
5. $B: D_{Bpri}(E_{Bpub}(M))$; $B \rightarrow A: M$. Bob descifra el mensaje con su clave privada para averiguar el lanzamiento de la moneda, y envía el mensaje descifrado a Alice.
6. Alice lee el resultado del lanzamiento y verifica que la cadena aleatoria es correcta.

No hace falta una tercera parte. Nótese que en el quinto paso, cuando Bob envía a Alice la "cara de la moneda" no podría cambiar el resultado obtenido, pues solo conoce el valor de la cara que le descifró Alice en el paso 4.

Protocolo de póquer mental

Es un protocolo similar al del lanzamiento de monedas, en que también se usa criptografía de clave pública.

1. $A: K_{Apub}, K_{Apriv}$; $B: K_{Bpub}, K_{Bpriv}$. Alice y Bob generan cada uno su par clave pública y privada.
2. $A \rightarrow B: E_{Apub}(M_i) \mid 1 \leq i \leq 52$. Alice genera 52 mensajes, uno para cada carta de la baraja. Estos mensajes contienen además alguna cadena aleatoria para verificar posteriormente su autenticidad. Los envía a Bob en **orden aleatorio**.
3. $B \rightarrow A: E_{Bpub}(E_{Apub}({M_j}^B))$. Bob, que no puede leer los mensajes elige aleatoriamente cinco de ellos; los cifra con su clave pública y se los devuelve a Alice. ${M_j}^B | 1 \leq j \leq 5$ son las cinco cartas que Bob ha elegido.
4. $A: D_{Apriv}(E_{Bpub}(E_{Apub}({M_j}^B))) = E_{Bpub}(M)$; $A \rightarrow B: E_{Bpub}({M_j}^B)$. Alice, que no puede leer los mensajes que Bob le ha devuelto, los descifra con su clave privada y se los devuelve a Bob.
5. $B: D_{Bpri}(E_{Bpub}({M_j}^B))$. Bob descifra los mensajes con su clave privada para averiguar su mano, y envía el mensaje descifrado a Alice.
6. $B \rightarrow A: E_{Apub}({M_k}^A)$. De los 47 mensajes que restan de los 52 que recibió en el paso 2, Bob elige aleatoriamente cinco de ellos y se los envía a Alice. ${M_k}^A | 1 \leq k \leq 5$ son las cinco cartas que Bob ha elegido para Alice.
7. $D_{Apri}(E_{Apub}({M_k}^A) = {M_k}^A$. Alice descifra estos cinco mensajes y ve cuáles son las cartas que le han tocado.