SEGURIDAD Y PROTECCIÓN

DE LA INFORMACIÓN II

(II55)

Apuntes de Teoría

Pascual Bayo Estéller

Curso 2008/2009

ÍNDICE

1. Introducción

El contenido de esta asignatura se basa en una ampliación de lo que se explicaba en Seguridad y Protección de la Información I (II38). Por esto, en primer lugar se verá un resumen de los conceptos referentes al modelo X509 explicados en el curso anterior, para profundizar después en las PKI (infraestructuras de llave pública). También se verá el funcionamiento de algunas arquitecturas criptográficas, más concretamente CryptoApi y Pkcs11.

Por último se presentarán problemas de criptografía de la vida real junto sus consecuencias y soluciones.

2. Infraestructuras de llave pública

2.1 Funciones Hash

Una función *hash* (o función de resumen) es un algoritmo que, dada una información de la longitud que sea, devuelve una cadena de caracteres alfanuméricos de longitud constante, de forma que cualquier variación, por pequeña que sea, en la información de entrada, provoca un cambio radical en la salida de la función.

Las particularidades de las funciones hash son las siguientes:

- Todos los números resumen generados con un mismo método tienen el mismo tamaño sea cual sea el texto utilizado como base.
- Dado un texto base, es fácil y rápido (para un ordenador) calcular su número resumen.

- Es imposible reconstruir el texto base a partir del número resumen.
- Es imposible que dos textos base diferentes tengan el mismo número resumen.

Pero no todo podían ser ventajas. Un inconveniente de la función Hash radica en el hecho de que la información de entrada se empieza a leer desde la izquierda. Por tanto, si para una información obtenemos un código H1, si añadimos información por la derecha a dicha información, el hash de esta nueva entrada será también H1.

Para solucionar este problema se obtiene el md5 y el sha1 del texto base y se juntan ambos resultados.

Las funciones hash más importantes son el md5 y sha, teniendo este último mayor interés por ser el utilizado en las firmas digitales.

Vamos a ver ahora los pasos que se siguen para la firma y cifrado de documentos digitales.

Firmar: creamos un paquete con el mensaje que queremos enviar. Le añadimos el hash de dicho mensaje, cifrando este último con la llave privada del emisor.

El destinatario, utilizando la llave pública del emisor, extrae el hash y el mensaje, y calcula el hash del mensaje que acaba de extraer. Si este valor coincide con el hash que se ha enviado en el paquete, el mensaje no ha sido alterado. En caso de no coincidir, se llega a la conclusión de que el mensaje ha sido modificado.

Tenemos dos tipos de firmas:

`Avanzada`: cuando se firma con un certificado que ha sido firmado por una CA reconocida por el estado.

`Reconocida`: cuando se firma utilizando un dispositivo hardware reconocido por el estado. El DNI electrónico tiene la firma reconocida. Por tanto, toda aplicación que lo utilice está obligada a aceptar la firma.

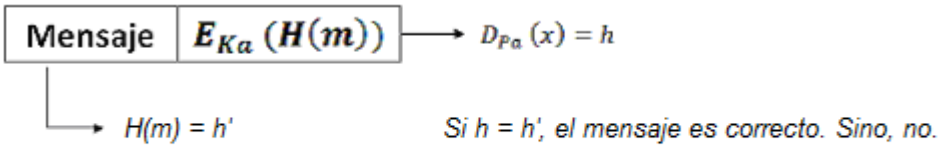


Figura 1: Firma de un documento

`Cifrar`: se crea una llave única, que será utilizada sólo en ese momento (denominada *`llave de sesión`*), con la que ciframos el mensaje. Además, incluimos la propia llave de sesión cifrada con la llave pública del emisor.

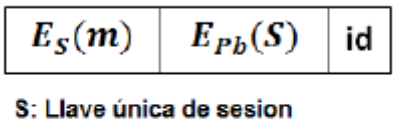


Figura 2: Cifrado de un documento

La ventaja de utilizar este método es que si queremos enviar los mismos datos cifrados a otro destinatario, basta con añadir al paquete la llave cifrada con la llave pública del nuevo destinatario.

A la hora de firmar hay que tener en cuenta que no es oro todo lo que reluce, y por tanto, hay que “aceptar” unas condiciones.

- **Confianza en el Software**: Microsoft incluye opciones por defecto en sus productos y nosotros debemos confiar en que ninguna de ellas es maliciosa para nosotros.
- **Confianza en la autenticidad**: la *confianza directa* consiste en obtener la llave pública directamente de la otra persona. La *confianza autoadquirida* consiste en aceptar a priori una llave (por ejemplo, al hacer un *ssh*) y a medida que vamos trabajando, nos vamos dando cuenta que la máquina es realmente la que nosotros pretendíamos conectarnos.

  Además existe la *confianza indirecta*, que puede ser vertical (cuando se confía en una tercera persona) y horizontal ("*los amigos de mis amigos, son mis amigos*").

Un concepto importante a tener en cuenta es la *brecha digital*, consistente en la distancia existente (cada día mayor) entre los que dominan las tecnologías y los "analfabetos digitales". Esta es una de las razones por las que la tecnología no crece todavía más.

2.2 Dispositivos de firma

La firma se puede hacer por Software (Firefox) o por Hardware (dispositivos). La firma por hardware consiste en un chip del que no se puede sacar la llave. Es decir, es imposible de copiar. El principal problema de estos aparatos es el coste temporal a la hora de hacer la firma, ya que funcionan a muy pocos Mhz y eso hace que una simple firma requiera un tiempo de cómputo considerable.

Estos chips también pueden generar la llave, por lo que de esta forma la llave nunca ha estado fuera del chip, pudiendo estar seguros que no ha sido copiada por otras personas.

2.3 Repaso del modelo x509

Este modelo nos dice como debe ser el certificado y la CRL (*c*ertificate Revocation List). Como ya vimos, los campos del certificado digital son:

Issuer y Subject = DN (Distinguished Name)

- CN (Common Name)
- O (Organization)
- OU (Organizational Unit)
- L (Location)
- ST (State)
- C (Country)

Número de serie

CA emisora

Fecha inicio - fecha fin

Versión

Extensiones

Algoritmo utilizado

Llave pública

Firma del certificado

Propósito del certificado

La forma en que se estructuran estos campos se define mediante el lenguaje ASN1, y los formatos más usuales son el DER y el PEM.

Los campos CN, O, OU, L, ST y C identifican al usuario, y la firma de toda la información recogida en todos los campos es lo que se conoce como el **certificado**.

Además de estos campos, se pueden añadir más *extensiones*, que son campos adicionales que necesitan de su propio software para ser interpretados debido a que ningún software es capaz de reconocerlos todos puesto que el número de extensiones es variable.

¿Qué pasa si perdemos nuestro certificado? ¿y si nos lo roban?

En caso de no disponer del certificado por una de las razones anteriores (entre otras), las Autoridades Certificadoras (CA) disponen de una CRL (Certificate Revocation List) donde se especifican los números de serie de los certificados revocados. De esta forma, podemos revocar nuestro certificado y evitar que otras personas puedan utilizarlo.

Esta CRL aparece firmada por la propia CA. Por tanto, será reconocida por los navegadores siempre y cuando tengan confianza en esa CA.

Problema de la CRL: hay que descargarse la CRL cada X tiempo para poder disponer de la lista actualizada con todos los certificados revocados.

Solución: OCSP (Online Certificate Storage Provider). OCSP comprueba la validez de los certificados vía web, con lo que el trabajo es mucho más cómodo.

2.4 Autoridades de certificación x509

Toda Autoridad Certificadora debe tener un documento llamado CSP donde se explican las medidas de protección físicas de la CA. Además, hay una autoridad encargada de asegurar que la CA cumple con lo especificado en el CSP. Esta autoridad es WebTrust, de forma que Windows acepta la CA siempre y cuando WebTrust no diga lo contrario.

Las partes del documento CPS son las siguientes:

- Identificación: se identifica la CA.
- Comunidad y Aplicabilidad
  - Autoridad Certificadora
  - Autoridad Registradora: la autoridad no puede firmar, pero sí que puede identificar.
- Detalles de contacto
- Obligaciones
- Garantías
- Responsabilidades financieras: si hay un fraude, la CA expone en este apartado cuanto puede cubrir.
- Precio de acceso a los certificados, de creación de certificados, …
- Identificación y autenticación: cómo se identifica y autentica a los subscriptores.
- Registro Inicial.
- Tipos de certificados.
- Proceso de identificación y autenticación de los subscriptores para cada uno de los certificados.

Estos dos últimos puntos son los elementos más importantes del certificado.

2.5 Tipos de certificados habituales

Hasta ahora hemos visto los certificados pertenecientes a una persona (certificado personal), pero vamos a ver que hay más tipos, dependiendo del uso que se les vaya a dar. Los certificados personales se descomponen en presenciales, no presenciales, y de email.

Certificado SSL/TLS: para servidores. En el CN se pone el nombre del dominio.

Certificado de pertenencia a empresa: acredita que el titular del certificado está vinculado con la empresa que aparece en él.

Certificado de representante: como el anterior pero añadiendo información de los poderes del titular.

Certificado de persona jurídica: identifica una empresa cuando hace trámites legales.

Certificado de servidor seguro: utilizado en servidores donde el intercambio de información debe ser exclusivo entre el servidor y el cliente, evitando que terceros puedan acceder a los datos.

Certificado de firma de código: garantiza que un código no ha sido alterado.

Como en cada servidor puede haber varios dominios alojados, en principio no podríamos tener un certificado común a todos ellos, puesto que debe haber un certificado por dominio/ip.

La forma de que un certificado nos sirva para varios dominios (alojados en la misma IP) es utilizar un comodín en el nombre del dominio (recordemos que en un certificado de servidor, en el CN se pone el nombre de dominio para el que se crea el certificado).

Vamos a verlo más fácilmente con un ejemplo:

Supongamos que tenemos los dominios www.abc.es, www.abc.com y www.abc.net. Lo que habría que hacer es pedir un certificado para www.abc.*, y como el navegador solo se fija en la parte delantera, nos serviría el mismo certificado para todos los dominios.

Lo que no tenemos forma de cambiar es que un certificado está asociado a una IP.

2.6 Modelos de expedición de certificados

Vamos a ver de qué forma otorgan los certificados las Autoridades Certificadoras más importantes, como son La Generalitat Valenciana y la FNMT (Fábrica Nacional de la Moneda y Timbre).

Generalitat Valenciana:

1. Acudir al punto de registro para la obtención del certificado.
2. Autenticarse presentando el DNI
3. Para obtener las llaves, se pueden seguir dos métodos:
    - *Software*: pkcs12 de firma, pkcs12 de cifrado
    - *Hardware (tarjeta):* llave de firma (generada dentro de la tarjeta), llave de cifrado (se genera fuera de la tarjeta y se mete después).

La mayor ventaja que ofrece la tarjeta es que una vez se han metido las llaves en su interior, estas ya no se pueden sacar de la misma.

Fábrica Nacional de la Moneda y Timbre:

1) *Generación*: el usuario, desde su casa, hace una petición de certificado poniendo su DNI en un pkcs10. Lo envía y recibe un ID.
2) *Identificación*: el usuario acude al punto de registro con su DNI, una fotocopia del mismo y el ID para comprobar que ha sido él quien ha hecho la petición de certificado.
3) *Obtención*: pasadas 24 horas después de haber acudido al punto de registro, el usuario podrá descargarse desde su casa el certificado.

2.7 Firma digital en la práctica

1. Pkcs7

Es un formato para guardar documentos firmados, en su clase *SignedData*. Esta clase tiene dos formatos:

- *No detached*: Contiene los datos más la firma. Lo malo es que si no tienen la herramienta adecuada de lectura del pkcs7, no se puede leer.
- *Detached*: Genera un pkcs7 únicamente con la firma y aparte tenemos el documento. El único inconveniente es tener que ir arrastrando dos objetos en lugar de uno.

  Una forma de solucionar esto es crear un correo electrónico que contenga los dos objetos e ir moviéndonos con este.

2. PDF

El pdf se puede firmar, lo único que necesitaremos un lector que lo reconozca, como el Acrobat Reader.

3. Correo Electrónico

Se puede firmar mediante SMIME o PGP, ambos de confianza horizontal.

4. Documentos

Openoffice utiliza el *xml signature*, que consiste en calcular el Hash del documento y se añade al final. *MicrosoftWord* no acepta la firma.

5. Firma Xades

Es el formato más correcto para firma de formato *xml*. Este exige que la firma lleve una marca de tiempo (timestamp), conseguida de una CA de tiempo, para saber la fecha real en que fue firmado el documento. El problema de este método es su difícil implementación.

3. Librerías y arquitecturas criptográficas

3.1 Pkcs

En criptografía, PKCS se refiere a un conjunto de estándares criptográficos de llave pública publicados por los laboratorios de RSA. Cada uno de estos estándares define unas reglas que se deben seguir para que un documento sea considerado pkcs#X, siendo X un número del 1 al 15. En otras palabras, pkcs se refiere a cómo debe ser la interfaz para una arquitectura de seguridad, y es el fabricante del dispositivo quien construye el pkcs.

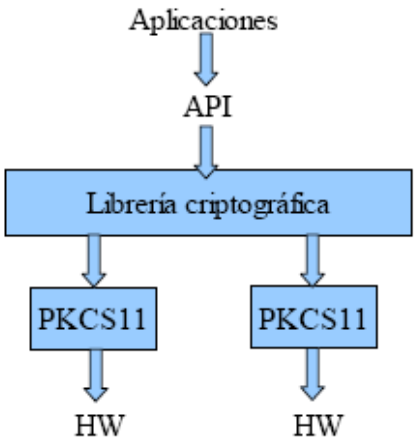
Los tipos de pkcs más comunes son:

- **pkcs#1** => define el formato del cifrado RSA.

- **Pkcs#7** => utilizado para firmar y cifrar mensajes en infraestructura de llave públic (PKI). Fue la base del SMIME.
- **Pkcs#10** => estandar de solicitud de certificado.
- **Pkcs#11** => define un API para acceder a dispositivos criptográficos.
- **Pkcs#12** => define el formato del fichero que almacena llaves privadas junto con el certificado de llave pública.

Firefox utiliza un pkcs#11 para firmar, cifrar, generar las llaves, etc.

Como se ha comentado, para cada dispositivo que haga uso de la criptografía se debe crear un nuevo pkcs#11. Es decir, se implementa a nivel de aplicación, por lo que si queremos utilizar un nuevo pkcs11, deberemos añadirlo a todas las aplicaciones que lo vayan a utilizar.



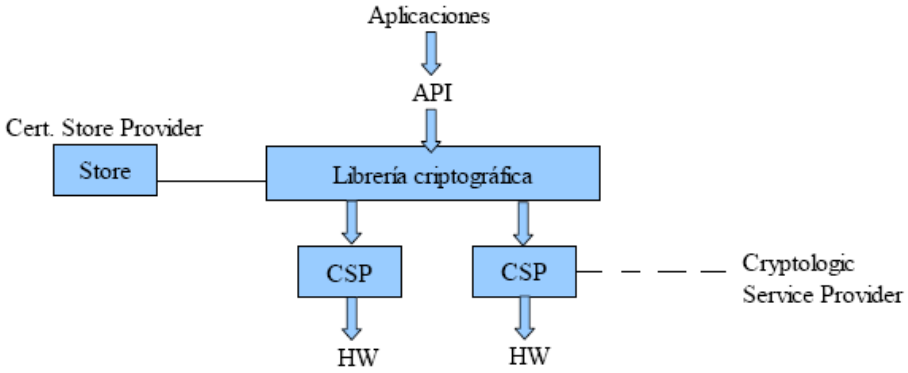
3.2 CryptoApi y CSP

Como la implantación del pkcs es bastante tediosa, existe también CryptoApi, consistente en una interfaz para la programación de aplicaciones incluida en los sistemas operativos de Microsoft que provee a los desarrolladores de servicios para utilizar la criptografía en aplicaciones basadas en Windows. Se

trata de un conjunto de librerías que proporcionan una capa de abstracción a los programadores entre la aplicación a desarrollar y el código utilizado para cifrar. Al tratarse de una arquitectura implementa a nivel de sistema operativo, todas las aplicaciones instaladas en el mismo podrán utilizarla.

Por otra parte, este conjunto de librerias trabajan sobre los CSP (Proveedor de Servicio Criptográfico), que son unos módulos donde se implementan las funciones de cifrado y descifrado de los dispositivos, por lo que hay que implementar un CSP por cada uno de estos.

Cada módulo que queramos añadir al CryptoApi debe llevar un CSP y un Store. El primero se encarga de la criptografía, y el segundo de gestionar los certificados.



Como se puede dar el caso de que hayan varios Store en el sistema (uno por dispositivo), existe un store MY por cada usuario y es ahí donde está lo propio de cada usuario. Este store MY es lógico, y de él se encargan los store físicos.

Windows utiliza un módulo criptográfico llamado CAPICOM para realizar las funciones del CryptoApi.

3.3 Dispositivos criptográficos

Un dispositivo criptográfico es, a grosso modo, una parte de un sistema que utiliza funciones de cifrado y descifrado.

Para que estas unidades puedan utilizar la criptografía, deben acceder a la funciones criptográficas, bien instaladas en la propia aplicación que utilicen(pkcs), bien instaladas en el sistema operativo (CryptoApi).

En el primer caso, basta con que accedan al pkcs implementado en el dispositivo para poder utilizar las funciones criptográficas.

En el segundo caso, deberán acceder a su CSP propio que és donde están implementadas las funciones de criptografía, sabiendo también cuál es el Store de certificados que están utilizando.

3.3.1. El clauer

El clauer es un proyecto de la UJI para incentivar el uso de la firma electrónica mediante la instalación de software especial en los ordenadores de forma que cuando se inserte el clauer, este software lo reconozca, además de cómo unidad de almacenamiento, como un almacén de certificados.

El clauer puede estar en dos formatos:

- `Disco clauer`: se utiliza un disco usb particionado en dos unidades, colocando en una de estas las herramientas de gestión del almacén de certificados y haciéndola semitransparente al usuario.
- `Fichero clauer`: se generan ficheros con las claves que se introducen en unidades de almacenamiento.

¿Cual es el problema del clauer? El problema está en que el dispositivo no tiene autonomía criptográfica puesto que simplemente es una unidad de almacenamiento. Por tanto, habrá que ir con cuidado y vigilar en qué ordenadores introducimos el USB. A poder ser, habrá que utilizar aquellos que sepamos que están administrados correctamente.

3.4. OpenSSL

SSL (Secure Socket Layer) es un protocolo para la conexión segura con clientes y servidores.

OpenSSL consiste en un robusto paquete de herramientas de administración y librerías relacionadas con la criptografía, que suministran funciones criptográficas a otros paquetes y navegadores web (para acceso seguro a sitios HTTPS).

Además, se divide en dos grandes partes:

1) Librería criptográfica: cifradores simétricos, asimétricos, RSA, funciones hash, librería de grandes números,…
2) Librería SSL: para montar clientes y servidores SSL.

3.5. Programación

Vamos a ver primero como se utiliza el RSA para funciones de firma, cifrado y verificación. A continuación veremos las funciones utilizadas a la hora de cifrar con EVP.

3.5.1 RSA

El algoritmo RSA es asimétrico (llave privada y llave pública), pero se comporta de forma simétrica, ya que se puede cifrar y descifrar tanto con la llave pública como con la llave privada. Aun así, se suele cifrar con la llave pública y se descifra con la llave privada.

Además está totalmente integrado en todas las aplicaciones. Las funciones de las que se compone son:

RSA publicencrypt(bytes,info,donde,llaveRSA, tipo_de_relleno)

Cifra la cantidad indicada en *bytes* de la información contenida en *info* y el resultado lo almacena en *donde*. Utilizará la llave RSA indicada y *tipo_de_relleno* indica los bytes de padding.

RSA privatedecrypt(*bytes,info,donde,llaveRSA, tipo_de_relleno*)

Al igual que en el anterior el tipo de relleno recomendado es RSA PKCS1 PADDING.

RSA sign(tipo Hash, fichero, tam fichero, buffer salida, tam buffer salida, llave RSA)

Hace un cifrado (firma) de *Hash* del fichero que se le pasa con la *llave RSA* proporcionado, almacenándolo en *buffersalida*.

RSA verify(tipo Hash, fichero, tam fichero, buffer salida, tam buffer salida, llave RSA)

Comprueba que la firma es válida.

3.5.2 EVP

Cuando se cifra con EVP, se obtiene un buffer con los datos cifrados y un array con las llaves de sesión cifradas para cada destinatario, además del tamaño del buffer y del array. En la práctica posteriormente se crea un *pkcs7* con el texto y todos los *recipe* y se envía a cada destinatario. Algunas funciones para el manejo del EVP son:

EVP signinit(contexto para hash, tipo Hash)

Inicializa la firma. En el *tipo de hash* le podemos pasar: EVP md4, EVP md5 o EVP sha1, si estamos trabajando con una llave RSA para firmar.

EVP verifyinit(contexto para hash, tipo Hash)

Se le pasan los mismos datos que a la función anterior y nos dice si esta es válida o no.

EVP signupdate(contexto para hash, buffer, tam buffer)

Va actualizando el *Hash* con los datos que le pasamos en el *buffer* (*longitud buffer* marca el tamaño del mismo). El *contexto para hash*, debe haber sido inicializado con *EVP signinit()* para que la función sea válida.

EVP verifyupdate(contexto para hash, buffer, tam buffer)

Se le pasan los mismos datos que a la función anterior y nos dice si esta es válida o no.

EVP signFinal(contexto para hash, buffer devolución, tam buffer, llave privada)

Para acabar la firma, le tenemos que pasar un puntero al buffer donde queremos tener la firma de los datos pasados a la función que será *buffer devolución*, así como la llave con la que queremos firmar(RSA o DSA). La función nos devolvería en *tamaño buffer*, el tamaño que tiene el mismo, esto también lo podemos conseguir con la función *EVP PKEY size()*. Como en el apartado anterior *contexto para hash*, debe haber sido inicializado con *EVP signinit()* para que la función sea válida.

EVP verifyFinal(contexto para hash, buffer devolucion, tam buffer, llave privada)

Se le pasan los mismos datos que a la función anterior y nos dice si esta es válida o no.

EVP sealinit(contexto para hash, tipo cifrado, buffer ini, buffer llaves, n llaves)

Inicializa los diferentes *recipes*, donde albergar cifrada la llave de sesión con la llave pública del usuario requerido.

EVP sealupdate(contexto para Hash, buffer salida, buffer tam salida, buffer entrada, buffer tam entrada)

Va actualizando los *buffer salida* con los datos que le pasamos en el *buffer* (*longitud buffer* marca el tamaño del mismo). El *contexto para hash* debe haber sido inicializado con *EVP sealinit()*, para que la función sea válida.

EVP sealfinal(contexto para Hash, buffer salida, buffer tam salida)

Finaliza el cifrado de las llaves de sesión y las devuelve junto con su tamaño en *buffer salida* y *buffer tam salida* respectivamente.

EVP openinit(contexto para Hash, tipo Hash, llave sesion cifrada, tam llave cifrada, buffer ini, llave privada RSA)

Inicializa el descifrado de una llave de sesión pasada en *llave sesión cifrada*, con el algoritmo *tipo Hash* y con la llave privada *llave privada RSA* .

EVP openupdate(contexto para Hash, buffer salida, buffer tam salida, buffer entrada, buffer tam entrada)

Va actualizando los *buffer salida* con los datos que le pasamos en el *buffer* (*longitud buffer* marca el tamaño del mismo). El *contexto para hash*, debe haber sido inicializado con *EVP openinit()* para que la función sea válida.

EVP openfinal(contexto para Hash, buffer salida, buffer tam salida)

Termina el descifrado de la llave de sesión, devolviéndola en *buffer salida*, así como su tamaño en *buffer tam salida*.

4. Protocolos criptográficos

4.1 La firma totalmente ciega

Lo que se pretende es que alguien nos firme algo a ciegas, es decir sin conocer nada de su contenido. El usuario (A) tiene m y quiere obtener $m^d \bmod n$, siendo (d,n) la llave privada del servidor (B). El esquema de Chaum es el siguiente:

A: genera k factores de opacidad (aleatorios, secretos y grandes).

Calcula $r = m \cdot k^e$

Envía r al servidor para que lo firme.

B: calcula $r^d \bmod n$ y se lo retorna al cliente.

A: calcula k^{-1}

Calcula $k^{-1} \cdot r^d = k^{-1}(m \cdot k^e)^d = k^{-1} m^d k^{ed} = m^d$ ➔ Firma.

El resultado que obtiene A es m^d, es decir, que B ha firmado m sin conocerlo.

4.2. Firma ciega de Chaum

El inconveniente del método anterior es que B puede llegar a firmar algo indebido, es decir que A presente un documento a su conveniencia y B lo firme creyendo que es otra cosa. Existen dos soluciones al problema. Una es vincular una llave privada a un objetivo de firma y administrativamente regular dicha vinculación. Es decir, "la llave X sirve para el tipo X de documento y la firma no tiene valor si no es un documento de ese tipo". El problema es que el servidor debe emplear una llave diferente y requiere una compleja gestión de llaves. Por ejemplo, en el caso de un sistema de encuestas, sería necesaria una llave distinta para cada encuesta, con la complejidad que ello implica.

La otra alternativa es emplear un protocolo estadístico, por el que B puede cerciorarse "aceptablemente" de la estructura del documento que está firmando. El protocolo es como sigue:
Perseguimos que el documento firmado tenga una determinada estructura y sólo pueda variar una componente aleatoria carente de significado.
El interesado, A, genera N documentos, d_i con i=1...N. Todos los documentos tendrán la misma estructura excepto el mencionado componente aleatorio.

Calcula los resúmenes (md5, por ejemplo) de cada uno m_i, genera los factores de opacidad r_i, y envía al servidor los N productos $x_i = r_i^e \bmod n$.

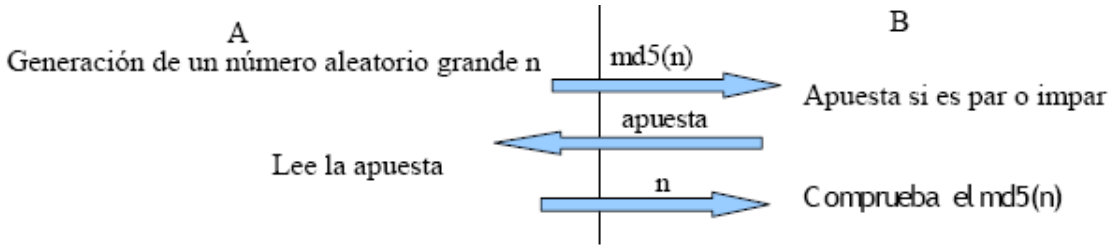
El firmante, B, elige un valor de i (sea j) y solicita a A los documentos d_i y los factores r_i con i distinto de j. Recibidos los documentos y los factores comprueba que los documentos son coherentes, es decir coinciden con el objetivo de la firma y sólo difieren en un componente aleatorio. Comprueba también que los x_i corresponden con los cálculos esperados y *presupone* que el documento d_j también tendrá la estructura de los demás, firmando de x_j.

El punto crítico es el valor de N. Si la penalización que A puede sufrir por hacer trampa es baja, A intentará conseguir un documento ilegal firmado, con lo que N debería ser grande. Si la penalización es alta, bastará con un valor bajo de N para que el sistema funcione correctamente.

4.3. Lanzar una moneda

Este problema consiste en cómo apostar al lanzamiento de una moneda estando seguros casi al 100% de que la persona que lanza la moneda no hace trampas.

Una solución podría ser utilizar un Tercero de Confianza, pero vamos a ver el proceso para que no necesitemos buscar una tercera persona.



A genera un número aleatorio grande, llamado n y le manda a B el resultado de aplicar el md5 sobre n. B, una vez recibe el md5, le dice a A su apuesta acerca si cree que n es par o impar. Hay que fijarse que B no ve n, sino su md5, y una propiedad de esta función es que su resultado no revela ningún tipo de información del número (en este caso) de entrada.

Así, una vez A recibe la apuesta (par o impar) de B, le manda a B el número n en claro.

Ahora, una vez B tiene n, calcula su md5 y lo compara con el que le había enviado A inicialmente.

Si coinciden, B sabe que A le ha enviado el número inicial, ya que B sabe que una propiedad importante de las funciones hash es que es casi imposible encontrar dos números cuyo md5 sea igual.

Así, sabiendo lo que ha apostado B y cual es el número n, se sabe a ciencia cierta qué parte ha ganado la apuesta.

4.4. Compartición de secretos

Imagina que tenemos un secreto pero somos algo olvidadizos y queremos que otra persona tenga el secreto, con la responsabilidad que eso conlleva. Por una parte, si le mandamos el secreto a la otra persona, debemos estar seguros

que confiamos en esa persona. Y esta persona, a su vez, debe asegurarnos que no lo va a difundir a terceras personas.

Esto, hoy en día, es difícil de asegurar al 100 %, por lo que lo que se hace es dividir el secreto en N partes y enviar cada parte a una persona diferente, de forma que la información que revele cada parte acerca del secreto esa 0. Si ahora queremos obtener el secreto, basta con juntar todas las partes.

El proceso que se sigue es el siguiente:

1. Supongamos que M es el secreto.
2. Genero una llave aleatoria X de la misma longitud que M.
3. Y(mensaje cifrado) = M $\oplus$ X
4. M(mensaje descifrado) = X $\oplus$ Y
5. Para dividir el secreto en varias partes:
    a. Generar las llaves X, Y, W
    b. Cifrar: Z = M $\oplus$ X $\oplus$ Y $\oplus$ W
    c. Descifrar: M = X $\oplus$ Y $\oplus$ W $\oplus$ Z

El inconveniente de este proceso es que se necesitan todas las partes para reconstruir el secreto, y no podemos garantizar que siempre vayan a estar las N partes disponibles para obtener el secreto.

Por esto aparece el método de Shamir, consistente en partir un secreto en varias partes y repartir cada una de estas a una persona diferente, de forma que podamos reconstruir el secreto sin necesidad de juntar todas las partes. Vamos a verlo con un ejemplo para que quede claro el proceso.

Supongamos que el secreto es el número de una tarjeta de crédito: 1234

(S = 1234).

Queremos dividir el secreto en seis partes (n = 6), de forma que cualquier subconjunto (k = 3) sea suficiente para reconstruir el secreto. Obtenemos dos números al azar: 166, 94.

(a1 = 166 ; a2 = 94)

El polinomio con el que operaremos será por lo tanto:

$$f(x) = 1234 + 166x + 94x^2$$

Calculamos ahora seis puntos (uno por parte) a partir del polinomio:

$$(1, 1494)\,;(2, 1942)\,;(3, 2578)\,;(4, 3402)\,;(5, 4414)\,;(6, 5614)$$

Para finalizar, damos a cada persona un único punto, formado por el valor $(x, f(x))$.

Si queremos ahora reconstruir el secreto, utilizaremos la Interpolación Polinómica de Lagrange[1] para obtener la ecuación inicial a partir de 3 de los 6 números que habíamos entregado. Y sabiendo que el coeficiente x^0 es el secreto, ya habremos acabado el proceso de obtención del secreto.

4.5. Dinero electrónico

El dinero electrónico es físicamente un número que se genera aleatoriamente, se le asigna un valor, se cifra y firma y se envía al banco, ahí el banco valida el número y certifica el valor, y lo regresa al usuario firmado por el banco. Entonces el usuario puede efectuar alguna transacción con ese billete electrónico.

Las principales propiedades del dinero electrónico son las siguientes:

[1] 1. Más información en Wikipedia.

1. Independencia:

La seguridad del dinero digital no debe depender de la el lugar físico donde se encuentre, por ejemplo en el disco duro de una PC.

2. Seguridad:

El dinero digital (el número) no debe de ser usado en dos diferentes transacciones.

3. Privacidad:

El dinero electrónico debe de proteger la privacidad de su usuario, de esta forma cuando se haga una transacción debe de poder cambiarse el número a otro usuario sin que el banco sepa que dueños tuvo antes.

4. Pagos fuera de línea:

El dinero electrónico no debe de depender de la conexión de la red y debe ser independiente al medio de transporte que use el propietario para llevarlo consigo.

5. Transferibilidad:

El dinero electrónico debe de ser transferible. Esto es, cuando un usuario transfiere dinero electrónico a otro usuario debe de borrarse la identidad del primero.

6. Divisibilidad:

El dinero electrónico debe de poder dividirse en valores fraccionarios según sea el uso que se da, por ejemplo en valor de 100, 50 y 25.

Es por esta última propiedad por la que el banco puede disponer de varios certificados (uno por valor fraccionario) y dependiendo del certificado con el que se firmó el "billete" se podrá pagar cantidades inferiores a lo que en él consta.

4.6. Voto electrónico

Vamos ahora a ver cómo se puede votar "desde casa". Primero se especifican los requisitos que debe cumplir el sistema que se implante y luego se irán desarrollando sistemas a partir de los fallos que se vayan encontrando a uno inicial.

Los requisitos para el voto por correo son:

1. Autenticidad

Solo deben poder votar las personas autorizadas.

2. Acotabilidad

Cada votante solo puede votar una vez. Se consigue mediante comprobación de IP's (votaciones anónimas) y mediante autenticación (votaciones privadas).

3. Anonimato

Imposibilidad de asociar voto y votante.

4. Imposibilidad de coacción

Que el votante no pueda demostrar lo que ha votado.

5. Verificabilidad individual

Que cada votante pueda comprobar que su voto se haya tenido en cuenta.

6. Verificación global

Que la mesa pueda comprobar la gente que ha votado, por ejemplo.

7. Fiabilidad

8. Que sea auditable

9. Neutralidad

Que no se conozca ningún resultado antes de terminar la votación

10. Movilidad de los votantes

11. Facilidad de uso

Que una persona no informática pueda votar con facilidad

12. Voto rápido

13. Posibilidad de hacer voto nulo

14. Código abierto

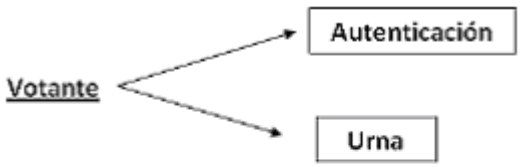
Esto no lo cumple nadie

15. Utilización en una red cerrada

16. Compatibilidad con mecanismos tradicionales

Es prácticamente imposible

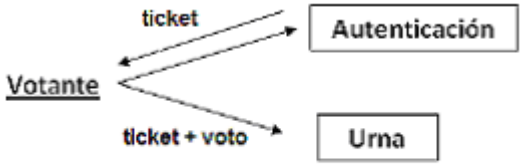
Vamos a ver cómo sería una primera idea para votar remotamente:



Sistema de votación 1

Una persona se identifica en una máquina remota, deposita su voto. Aunque es una solución bastante sencilla, no es factible porque el administrador del sistema remoto podría ver quién ha votado, a qué hora y a quién.

Por tanto, debemos buscar un poco más de seguridad para los votantes, de forma que no se pueda saber a quién han votado (incluso ni si lo han hecho).

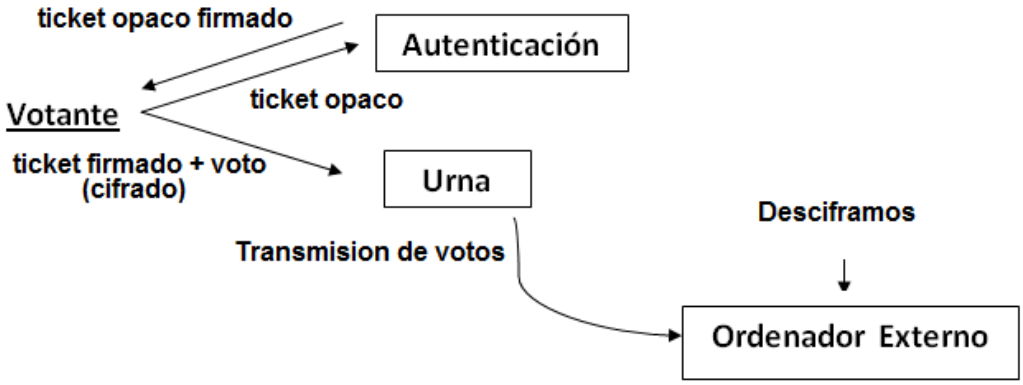


Sistema de votación 2

Con el nuevo sistema, una persona se autentica en la máquina remota y tras esto, recibe un ticket del servidor. Ahora, ya puede cerrar la sesión autenticada, de forma que enviará a la urna su *ticket+voto*.

Problema: si el servidor de autenticación y el de la urna se quedan los tickets, ambos pueden hacer un join con todos los tickets y tendrían la información referente a las votaciones.

Veamos una solución que esquiva estos problemas aunque luego se explicará por qué no se utiliza demasiado en la actualidad.



Sistema de votación 3

El votante envía un ticket opaco al servidor de autenticación, y este, utilizando firma ciega, se lo devuelve firmado al votante. Con esto ya conseguimos evitar el problema de relacionar votantes con sus respectivos votos.

Al cabo de un tiempo, el votante cifrará su ticket firmado + voto y enviará el resultado a la urna.

Cuando acabe el periodo de votación, se enviarán todos los votos a un ordenador externo donde se descifrarán y se contabilizarán.

Para que no se conozcan los votos durante la votación, el procedimiento es generar las llaves antes de la votación, utilizar *secret sharing*[2] para partir la llave y recomponer la llave privada al finalizar la votación.

El motivo por el que no se ha implantado al 100 % este sistema es porque la firma ciega está patentada, y por tanto, cada vez que se utilizaba había que pagar por ello, con lo que salía bastante caro.

[2] Más información en Wikipedia

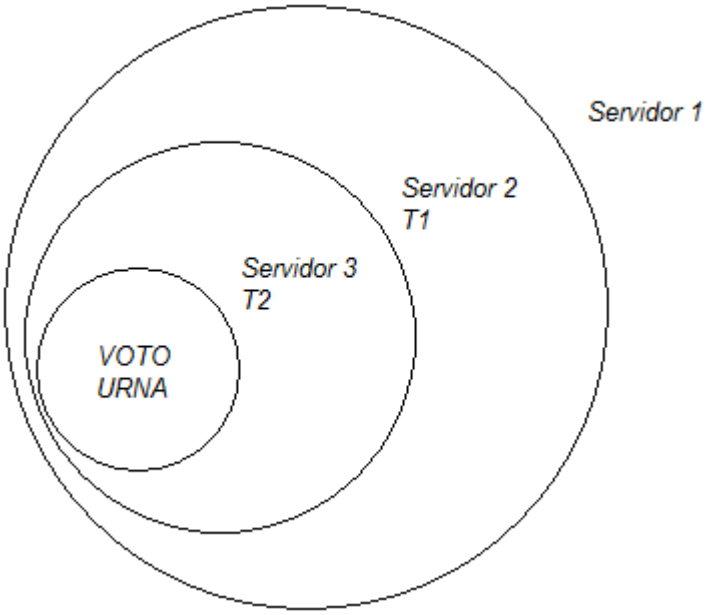
¿Qué pasa cuando nos autenticamos en un servidor? Que se crea una entrada en un log con nuestro usuario y la hora a la que nos hemos autenticado. Si a continuación enviamos el voto anónimo, el administrador verá que a las 10:05 se ha autenticado la persona X y a las 10:07 se ha enviado un voto anónimo.

Aunque no tiene por qué pertenecer el voto a dicha persona, la proximidad temporal sí que incita a pensar que la persona X habría enviado dicho voto, con lo que se podría establecer algún tipo de relación.

La solución consiste en enviar el voto a un buffer en lugar de a una urna, y cuando el buffer tenga un número considerable de votos, que los envíe todos a la urna desordenados. También se podría pensar en una cadena de servidores auxiliares en lugar de un buffer. Así, podríamos incluir en el paquete que enviamos a la urna el tiempo que queremos que se retrase el envío de un servidor a otro, eliminando el problema del rastreo temporal. Como se puede dar el caso que cayese uno de estos servidores, añadiendo varios caminos por los que llegar a la urna quedaría resuelto el problema.

¿Qué pasa ahora si el primer servidor auxiliar está asociado con el servidor de gestión de los usuarios? Pues que podría modificar el tiempo de permanencia del voto en este primer servidor, y ajustarlo al límite, para que al llegar el voto al servidor 2, al estar en el límite, lo reenviaría al servidor 3, y así sucesivamente, con lo que el tiempo de espera del voto en los servidores sería nulo.

La solución a este problema es utilizar una estructura que se conoce como *Onion Routing*, y quedaría una estructura como la siguiente:

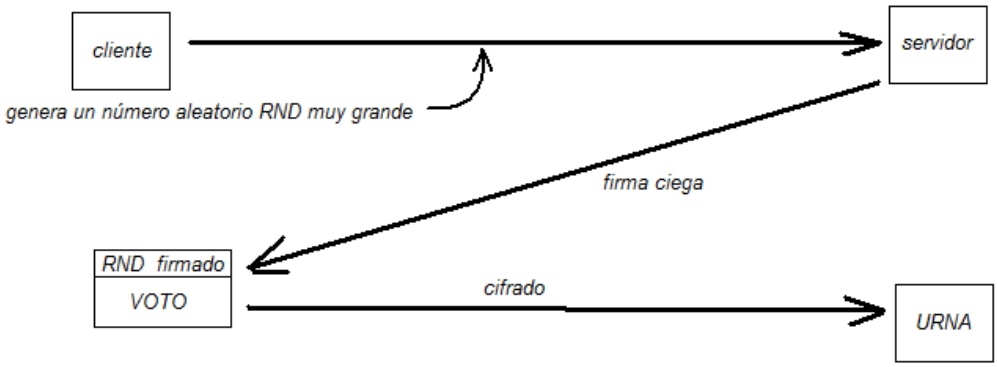


Estructura de capas en Onion Routing.

El cliente envía al servidor 1 dicha estructura, donde el servidor 1 extrae la dirección del siguiente servidor al que enviar el voto (Servidor 2) y el tiempo que debe permanecer el voto en el Servidor 1. A su vez, Servidor 2 recibirá su estructura y obtendrá la dirección del servidor 3 y el tiempo de permanencia en el servidor 2. Por último, el servidor 3 enviará el voto a la dirección de la urna.

4.7 Problemas del voto telemático

La clave aquí se basa en la firma ciega, pero vamos a ver que hay todavía problemas mediante los cuales se pueden establecer asociaciones en los votos.



Asociación de origen: el paquete se envía con una ip, con lo que el servidor podría asociar el paquete con dicha ip.

Rastreo temporal: en un espacio de tiempo grande, puede que en un día solo vote una persona, por lo que el servidor sabría quien ha votado.

Bibliografía

- Apuntes de Fernando Marco Sales
- Apuntes de Alejandro Llaves Arellano

- http://www.eumed.net/cursecon/ecoinet/seguridad/resumenes.htm
- http://en.wikipedia.org/wiki/CAPICOM
- http://es.wikipedia.org/wiki/Firma_digital_ciega
- http://es.wikipedia.org/wiki/Esquema_de_Shamir
- http://es.wikipedia.org/wiki/PKCS
- http://en.wikipedia.org/wiki/Cryptographic_Service_Provider
- http://en.wikipedia.org/wiki/Cryptographic_API