Tema 5: Integridad de mensajes, firmas y certificados

Docente : Héctor Xavier Limón Riaño
email: xavier120@hotmail.com

Introducción

Introducción

En este tema se verá:

- "keyed hashes"
- Uso de criptografía asimétrica para integridad y autenticidad de mensajes
- Certificados y su diferencia con las llaves

Message Authentication Code (MAC)

- Como ya se vio antes, aunque un mensaje esté cifrado, éste puede ser alterado de formas que tienen sentido
- El cifrado no es suficiente para establecer integridad de datos
- Un MAC es cualquier código o dato transmitido junto con el mensaje que puede ser evaluado para determinar si el mensaje fue alterado
- Usualmente se utiliza un hash cómo código, osea que el código es el hash del mensaje

Message Authentication Code (MAC)

Proceso básico (no seguro):

- El emisor computa un código C_1 usando una función $f(M_1)$ para un mensaje dado M_1
- El emisor transmite C_1 y M_1 al receptor
- El receptor recibe los datos como C y M pero no sabe si han sido alterados
- El receptor recomputa $f(M)$ y compara la salida con C para verificar que no hubo alteraciones

Message Authentication Code (MAC)

- El proceso anterior no es seguro porque un atacante que intercepta M_1 y lo modifica para obtener M_2 podría computar un nuevo C_2 y mandar M_2 y C_2 sin que el receptor se dé cuenta
- Se requiere también de una llave que el atacante no debe saber
- Más adelante se verá el proceso completo

MAC, HMAC y CBC-MAC

MAC

- Un MAC seguro no es sólo el código (hash) también debe incluir una llave
- Una llave MAC no está relacionada con el cifrado
- Sirve para asegurar que el código de autenticación de mensaje sólo puede se computado por las entidades que conocen la llave
- No se debe asumir que el atacante no conoce la función de hash, recuerda que una buena seguridad significa que el atacante puede saber todo, excepto la llave (principio de Kerckhoffs)

MAC

- El MAC protege la integridad del mensaje.
- Un atacante sin la llave no debe poder alterar el mensaje sin que se pueda detectar
- Mientras la llave se mantenga privada, el MAC provee autenticidad: el receptor sabe que sólo el emisor real pudo enviar el mensaje porque sólo así puede generarse un MAC correcto

MAC

- Para fines del curso se verán dos algoritmos MAC fáciles de entender: HMAC y CBC-MAC
- A pesar de ser simples, se utilizan en la práctica e ilustran porque los MAC funcionan

HMAC

- Hash-based Message Authentication Code
- Es básicamente un hash keyed (no hay un término apropiado en español)
- Recordar que las funciones de hash son deterministas: dada la misma entrada, siempre se produce la misma salida
- Cuando un algoritmo es keyed, la salida no depende sólo del contenido de entrada, sino también de la llave

HMAC

- Para hacer que una función de hash esté keyed es en esencia sencillo
- Recuerda que las funciones de hash tienen la propiedad de avalancha, basta con agregar la llave al mensaje original para que produzca una salida muy diferente
- Como con cualquier llave criptográfica, ésta debe ser suficientemente larga y aleatoria para prevenir ataques de diccionario o fuerza bruta

HMAC

- Sólo agregar al inicio o al final del mensaje la llave no es suficientemente seguro
- HMAC utiliza una aproximación más segura
- HMAC está definido en el estándar RFC 2104
- `https://tools.ietf.org/html/rfc2104`

HMAC

- De acuerdo al RFC, una vez se decide una función de hash H, el HMAC sobre un texto de entrada se computa de la siguiente manera:
- $H(K \oplus opad, H((K \oplus ipad), text))$
- Un detalle con el que se debe tener cuidado es que en esta definición las comas no significan separación de parámetros de la función, sino más bien se pueden interpretar como concatenaciones (las funciones de hash sólo reciben un parámetro)

HMAC

$H(K \oplus opad, H((K \oplus ipad), text))$, donde:

- H es la función de hash
- K es la llave que debe cumplir ciertos requisitos
- *text* es cualquier binario arbitrario que representa el contenido de entrada
- *opad* e *ipad* son dos valores estáticos que se presentarán más adelante

HMAC

- La llave K debe cumplir con varios requisitos, principalmente relacionados con el tamaño de bloque de H
- ¿Tamaño de bloque en una función de hash?
- Este término ya se había usado en cifrado, por ejemplo AES tiene un tamaño de bloque de 16 bytes, lo que quiere decir que se cifran 16 bytes a la vez
- Pero el hash se aplica a toda la entrada de golpe, ¿Cierto?

HMAC

- De hecho las funciones de hash también trabajan por bloque, por eso funciona la función de update, cada vez que se llama se aplica a cierto número de bloques
- Una función de hash opera un bloque a la vez y la salida de un bloque se encadena con el cálculo del siguiente bloque
- Por ejemplo, SHA-1 es de 64 bytes de bloque y SHA-256 es de 128 bytes de bloque

HMAC

En el RFC el tamaño de bloque de H se define como B, estás son las reglas de restricción de K:

1. Si K es más corto que B, entonces a K se le tienen que agregar ceros de padding hasta ser de longitud B
2. Si K es más largo que B, entonces K se reduce aplicando H a K, osea $H(K)$
3. Si se aplicó 2 y la llave quedó muy corta, entonces se aplica 1

HMAC

- Si se aplican las reglas anteriores, siempre se va a tener una llave de longitud B
- En el RFC *ipad* se define como el byte $0x36$ repetido B veces
- *opad* es el byte $0x5c$ repetido B veces
- Los pads parecen arbitrarios pero tienen un sentido de seguridad matemático que ayuda a mitigar problemas si los algoritmos de hash son vulnerados (como con MD5 o SHA-1)

HMAC

- Primero debe computarse $H((K \oplus ipad), text)$ que es el inner-hash
- Computar $K \oplus ipad$ es fácil porque son de la misma longitud
- El resultado anterior se concatena con *text* y a este resultado se le aplica H

HMAC

- Luego se computa $K \oplus opad$
- Para obtener el outer-hash final se concatena el resultado de arriba con el resultado del inner-hash y se aplica H
- El outer-hash es el HMAC de *text* keyed con K

Ejercicio

- Realiza una implementación manual de HMAC para SHA-256 de acuerdo a la explicación del RFC
- De acuerdo al tiempo que quede de clase el profesor podría implementar parte del ejercicio

HMAC

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes, hmac

key = b"CorrectHorseBatteryStaple"
h = hmac.HMAC(key, hashes.SHA256(), backend=default_backend())
h.update(b"hello world")
h.finalize().hex() # HMAC en hexadecimal
```

CBC-MAC

- Similar a AES CBC
- Recordar que CBC es Cipher Block Chaining
- Un MAC a menudo es llamado un **tag** (etiqueta) del mensaje
- Es como una etiqueta que se agrega a un regalo: es información extra que se asocia al contenido principal

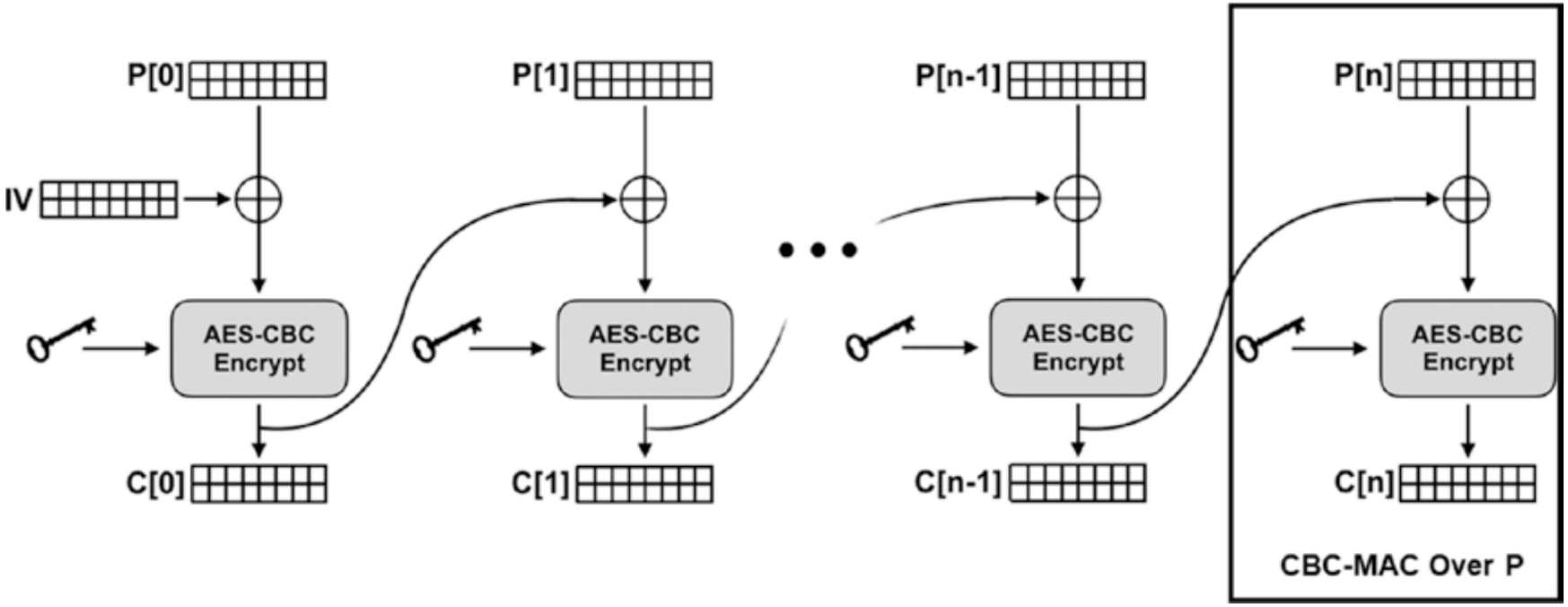
CBC-MAC

- En notación matemática un tag se denota a menudo como t
- Así, un MAC sobre un mensaje m_1 produce un tag t_1
- El par (m_1, t_1) es transmitido al receptor para ser verificado

CBC-MAC

- En el caso de AES, recuerda que se estaba limitado a cifrar 128 bits de bloque a la vez
- La idea de AES CBC es encadenar la salida de un bloque cifrado con el siguiente, para que no haya independencia entre bloques y sea más complejo alterar la integridad de los datos
- Si se alterara un bit en el primer bloque ese cambio crea una cascada de cambios en los bloques siguientes hasta llegar al último bloque
- Este efecto hace que el último bloque de CBC sea por si mismo un MAC

CBC-MAC


P[0]
P[1]
P[n-1]
P[n]
IV
AES-CBC Encrypt
AES-CBC Encrypt
AES-CBC Encrypt
AES-CBC Encrypt
C[0]
C[1]
C[n-1]
C[n]
CBC-MAC Over P

CBC-MAC

- En esencia CBC-MAC es AES CBC, recuperando el último bloque
- Pero se deben tomar en cuenta varias consideraciones de seguridad
- Veamos una implementación no segura

CBC-MAC: ataque antepuesto

- MAC prepend attack
- Este ataque funciona con la implementación insegura antes vista
- La idea básica es que si se tienen dos mensajes M_1 y M_2 y sus tags correspondientes t_1 y t_2 generados con la misma llave k, es posible generar un mensaje nuevo manipulado M_3 cuyo tag t_3 sea igual a t_1 o t_2
- Osea que es posible mandar un mensaje manipulado y hacer que parezca que es el mensaje M_1 o M_2

CBC-MAC: ataque antepuesto

- En el ataque M_1 es agregado al inicio de M_2 (o vise-versa), osea es antepuesto
- M_3 es una mezcla de M_1 y M_2
- Si el atacante realiza bien el ataque puede crear información falsa, de forma limitada pero con consecuencias prácticas para el atacante
- El atacante no necesita conocer la llave de MAC

CBC-MAC: ataque antepuesto

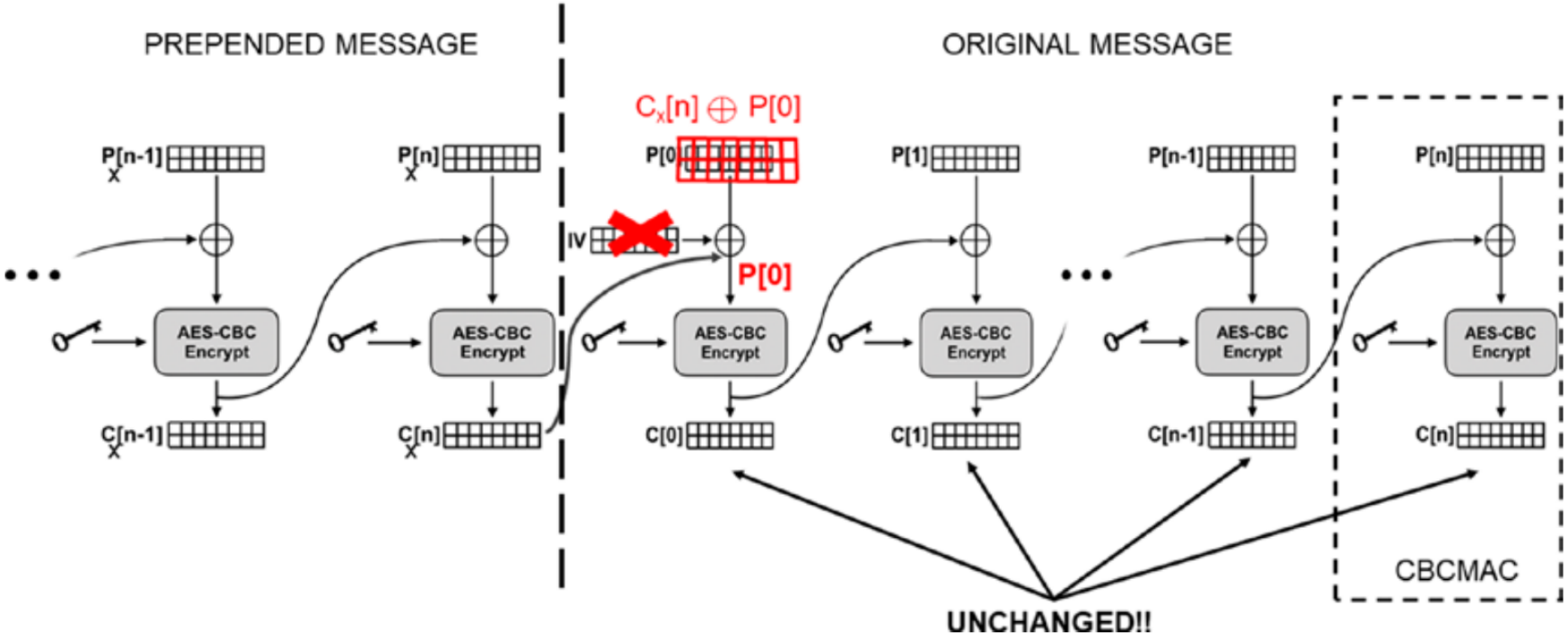
El proceso del ataque es el siguiente (anteponiendo M_1 a M_2):

- Recuperar el primer bloque de M_2, denominado M_{2_1} (como es AES se toman los primeros 16 bytes)
- Recuperar el resto de bloques de M_2, denominado M_{2_r}
- Obtener $M_2' = t_1 \oplus M_{2_1}$
- $M_3 = M_1 + M_2' + M_{2_r}$ ($+$ significa concatenación)
- Con esto $t_3 = t_2$

CBC-MAC: ataque antepuesto

- Básicamente lo que hace el ataque es remplazar el primer bloque de M_2 por uno manipulado
- La víctima verá todo el mensaje M_1 completo (posiblemente con su padding), un bloque que se verá posiblemente corrupto, y el mensaje M_2 sin su primer bloque

CBC-MAC: ataque antepuesto



CBC-MAC: ataque antepuesto

- Esto funciona por la aplicación de XOR que hace CBC, esto puede expresarse como (E es cifrado AES):
- $t_3 = E(M_{2_n} \oplus E(M_{2_{n-1}} \oplus ...E(M_{2_1} \oplus t_1 \oplus C(M_1, k), k)..., k), k)$
- El pedazo $M_{2_1} \oplus t_1$ es el bloque manipulado M_2'
- $C(M_1, k)$ es el último bloque cifrado de M_1, osea, es lo mismo que t_1
- Dado lo anterior, los dos t_1 se cancelan por XOR, esto es:
- $E(M_{2_1} \oplus t_1 \oplus C(M_1, k), k) = E(M_{2_1}, k)$
- La manipulación no afecta el cálculo, por lo que $t_3 = t_2$

CBC-MAC: ataque antepuesto

- Existen muchas soluciones a este ataque
- Una de ellas es por ejemplo agregar siempre la longitud del mensaje al inicio de éste, de esta manera puede revisarse si se agregó información extra
- Para hacer una implementación segura se requiere además:
  - Si además los datos se están cifrando con AES-CBC, no se debe usar la misma llave para cifrar y obtener el MAC
  - El IV debe ser diferente de cero
- Lo mejor es utilizar una versión segura estable, por ejemplo CMAC de la biblioteca cryptography que es una versión mejorada de CBC-MAC descrita en el RFC 4493

CBC-MAC

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import cmac
from cryptography.hazmat.primitives.ciphers import algorithms

c = cmac.CMAC(algorithms.AES(key), backend=default_backend())
c.update(b"message to authenticate")
mac = c.finalize()
```

HMAC y CBC-MAC

- Tanto HMAC como CBC-MAC son buenas opciones para un algoritmo MAC
- HMAC puede ser más eficiente en la mayoría de sistemas, con excepción de aquellos que tienen hardware especializado para AES
- Ante la duda, utiliza HMAC

Cifrado y MAC

- En muchas circunstancias un mensaje necesita ser cifrado y protegido contra modificaciones
- Si lo analizan, en lo que se vio anteriormente no se especifica si los mensajes para los cuales se derivaba el MAC estaban cifrados o no, en realidad podría ser cualquiera de las dos, los métodos funcionarían igual
- Pero, si se va a cifrar ¿En qué momento hay que derivar el MAC? ¿Antes o después de cifrar?

Cifrado y MAC

Existen tres aproximaciones:

- Cifra luego aplica MAC (Encrypt-Then-MAC): el MAC se aplica al texto cifrado
- Aplica MAC luego cifra (MAC-Then-Encrypt) (usado por versiones iniciales de TLS): se aplica el MAC al texto plano, luego se cifra el texto plano junto con el MAC
- Cifra y aplica MAC (Encrypt-And-MAC) (usado por SSH): se aplica el MAC sobre el texto plano, pero no se cifra el MAC, sólo el texto plano

Cifrado y MAC

- Lo que recomienda la mayoría de los criptógrafos (aunque no todos), es utilizar cifra luego aplica MAC (Encrypt-Then-MAC)
- Algunas vulnerabilidades han sido encontradas para aplica MAC luego cifra (MAC-Then-Encrypt), de hecho por eso ya no se usa en TLS. Por ejemplo el ataque de padding oráculo a CBC sólo funciona con este esquema
- Cifra luego aplica MAC parece contra-intuitivo, puesto que lo que queremos saber es si los datos originales (texto plano) llegaron bien

Cifrado y MAC

- Pero considera que tampoco queremos que los datos cifrados sean alterados, de hecho ya hemos visto varios ataques en este sentido
- Cosas malas pasan si los atacantes pueden cambiar el texto cifrado sin que los receptores se den cuenta
- Siempre prefiere el esquema de cifra luego aplica MAC
- En realidad hay algo mejor llamado AEAD (authenticated encryption with additional data) que deberías usar en vez de MAC
- Este esquema integra autenticación e integridad en un cifrado simétrico, se verá en el tema 7

Tarea

- Esta tarea se hará en equipo de dos personas, contarán con un poco más de una semana para hacerla
- Tiene un valor triple
- Dada una implementación para crear un chat basado en sockets TCP provista por el profesor, modifica el código para que todas las transmisiones estén cifradas y protegidas con un MAC
- Utiliza cualquier variante segura de AES vista (CBC o CTR)
- Utiliza HMAC o CMAC
- Utiliza el esquema de cifra luego aplica MAC
- Considera que el servidor y cada cliente deben conocer la llave de cifrado simétrico y la llave MAC, para simplificar, dichas llaves estarán en archivos, al ejecutar los clientes simplemente se pasan la llaves como parámetro
- Cada cliente tiene su llave MAC y su llave de cifrado propia, éstas se le pasan al servidor de forma no segura al inicio

Firmas digitales

Introducción

- Al igual que los MAC, las firmas digitales brindan autenticación e integridad
- Un problema de los MAC es que al utilizar llaves simétricas la llave puede comprometerse en cualquiera de los dos extremos de la comunicación
- De comprometerse esa llave, un atacante puede enviar mensajes falsos haciéndose pasar por la entidad original (spoofing)

Introducción

- La autenticidad es al menos tan importante como la confidencialidad
- Si las llaves se comprometen es posible que pase un tiempo para que las entidades se den cuenta, además, el proceso de revocación puede ser tardado
- Las firmas digitales mitigan este problema mediante criptografía asimétrica

Introducción

- Dado que las firmas digitales usan llaves asimétricas, no existen llaves compartidas, lo que mitiga problemas de seguridad
- Por ejemplo, si diez personas de una organización se comunicaran entre si con llaves simétricas y cada quien tuviera su llave personal de autenticación, entonces las diez personas necesitarían las diez llaves.
- Si a uno de ellos lo capturan, entonces las diez llaves se comprometerían, el atacante podría hacerse pasar por cualquiera de las diez personas
- Con llaves asimétricas el problema sería mucho menos grave, si capturan a una persona, se obtendrían diez llaves públicas y una privada, el atacante sólo podría hacerse pasar por el capturado

Criptografía asimétrica

- Se tienen dos llaves: una pública y privada
- La llave pública se puede derivar de la privada, pero no al revés
- Si se cifra con la llave pública sólo se puede descifrar con la llave privada y vise-versa
- La llave privada siempre debe permanecer privada, mientras que la pública está pensada para ser compartida

Criptografía asimétrica

- Cualquiera que mandé un mensaje cifrado con la llave pública sabe que sólo el poseedor de la llave privada podrá descifrarlo
- Esto brinda autenticidad en reversa: el receptor no sabrá de quien es el mensaje pero el emisor sabe que sólo podrá ser procesada por la entidad autorizada
- Este principio es muy importante en la navegación web: los usuarios esperan que la información privada que generan sólo pueda ser procesada por el sitio en el que confían

Criptografía asimétrica

- Si se cifra con la llave privada en cambio, el mensaje puede ser descifrado con la llave pública, lo cual no ayuda a la confidencialidad
- Pero el que recibe el mensaje sabe que sólo puede provenir de alguien con la llave privada, esto brinda autenticidad al mensaje

Criptografía asimétrica

- Todo esto se ve muy bien, puedes tener autenticidad normal y en reversa con un solo par de llaves pública y privada, sin tener que compartir la llave privada
- Sin embargo, hay varios problemas importantes

Criptografía asimétrica

- ¿Cómo sabes al recibir una llave pública que proviene de la entidad oficial que esperas?, un atacante podría darte su llave pública para engañarte (más sobre este problema más adelante)
- El cifrado asimétrico es ineficiente, muy ineficiente para documentos largos
- Descifrar un documento largo sólo para saber si proviene de la entidad correcta no es viable
- En el caso de RSA (y muchos otros cifradores asimétricos), no tiene forma de revisar la integridad de los datos, además de que sólo puede procesar documentos tan grandes como el tamaño de modulo (2048 bits tradicionalmente)

Firmas digitales

- Los problemas de integridad y eficiencia son fáciles de solucionar
- En el modelo explicado, no se está cifrando por confidencialidad, sino por autenticidad
- En vez de cifrar el mensaje, se puede cifrar un hash del mensaje
- Esto da tanto integridad como autenticidad
- Esto es esencialmente una firma digital sobre un mensaje

Firmas digitales

Siguen dos pasas básicos:

- Calcular el hash del mensaje
- Cifrar el hash mediante la llave privada

Firmas digitales

- El hash cifrado es la firma digital del emisor
- La firma es transmitida junto con el mensaje (cifrado o no)
- El receptor calcula el hash del mensaje, descifra la firma digital mediante la llave pública y comprueba que los dos hashes coincidan

Firmas digitales

Un poco de notación matemática

- h es un valor de hash
- M es un mensaje
- H es una función de hashc
- P es un texto plano
- K es una llave
- $K_{A,B}$ es una llave simétrica compartida por A y por B
- K_A es la llave pública de A
- K_A^{-1} es la llave privada de A
- $\{P\}_K$ es el texto plano cifrado mediante K
- t_M es la firma digital de un mensaje M

Firmas digitales

- Dada la notación anterior:
  - $t_M = \{H(M)\}_{K^{-1}}$
- El receptor recibe M', t_M
- El receptor calcula $h' = H(M')$ y recupera h al descifrar t_M
- Tiene que cumplirse que $h' = h$

Firmas digitales

- Una vez más: el cifrado con la llave pública es para confidencialidad, mientras que el cifrado con la llave privada es para autenticidad
- A continuación se verá el uso de firmas digitales RSA con la biblioteca cryptography

Firmas digitales

- Como pudo verse, las firmas digitales también requieren de padding
- En este caso el esquema recomendado para firmas RSA es PSS
- No son iguales las necesidades de seguridad de padding cuando se cifran mensajes que cuando se crean firmas digitales, por eso se requiere de otro esquema
- MGF1 es una función de generación de mascara, actualmente sólo existe una
- También puede verse que se necesitaba de una función de hash, en este caso SHA-256

Cifrar y firmar

- Si se quiere tanto cifrar y firmar, ¿En qué orden debería hacerse?
- Esto es diferente a los MAC
- Por lo general no debe usarse cifra luego firma (como se debe hacer con un MAC)

Cifrar y firmar

- La meta de una firma no es sólo integridad del mensaje, sino también autenticidad del emisor
- Si se cifra y luego se aplica la firma, entonces un atacante podría separar el mensaje de la firma, recalcular el hash del mensaje cifrado y generar una firma con su propia llave privada, haciendo que parezca que el mensaje es de su parte
- Aunque dicho ataque no parece ser muy práctico, muestra que es fácil alterar la autenticidad

Cifrar y firmar

- Lo que quiere el receptor es saber si los datos que descifró (el texto plano) vienen de la entidad correcta
- Si se cifra y luego firma, la firma no tiene relación directa con el texto plano
- La firma debe aplicarse siempre al texto plano

Cifrar y firmar

- Una debilidad de las firmas digitales es que no pueden prevenir que se altere el texto cifrado, a diferencia de los MAC
- Pero si se adopta cifra y luego firma no se resuelve el problema (como si lo hacía con MAC), porque los mensajes cifrados pueden ser alterados y se puede generar una nueva firma a partir de éstos
- Al crear nuevamente la firma tendría que ser con alguna llave privada, probablemente no del emisor original, pero no necesariamente tiene que ser la del atacante (en la cual no confía el receptor), el atacante pudo haber robado una firma digital de alguien en quien si confía el receptor

Cifrar y firmar

- La diferencia fundamental entre un MAC y una firma son las llaves
- Las llaves de MAC son simétricas, posiblemente sólo compartidas entre emisor y receptor, esto crea una ligadura entre ambos
- Nadie puede remplazar de forma válida el MAC sin conocer la llave
- Pero en una firma digital, la llave privada no es compartida y no liga a dos entidades entre si

Cifrar y firmar

- A pesar de la debilidad de las firmas digitales contra alteraciones del texto cifrado, esto no suele ser un problema para la mayoría de protocolos criptográficos
- Lo común es sólo utilizar técnicas asimétricas (como firmas digitales) al inicio de la comunicación, para el establecimiento de confianza e intercambio de llaves simétricas (más sobre esto en el siguiente tema)
- Después todo se realiza mediante llaves simétricas, pudiéndose utilizar MACs
- Así mismo, alterar mensajes cifrados de forma significativa con un esquema de padding adecuado como OAEP es extremadamente difícil

Cifrar y firmar

- Si tienes que utilizar firmas digitales para un mensaje, lo correcto es firmar luego cifrar, esto es, se firma el texto plano y luego se cifra tanto el mensaje como la firma
- Aunque no existen ataques conocidos para este esquema, los más paranoicos pueden utilizar un esquema: firma luego cifra luego vuelve a firmar
- Lo anterior te da una firma para la información original y otra para los datos cifrados, con doble autenticación
- Existen otras técnicas conocidas como singcryption que no se verán en el curso

Tarea

- Tomando nuevamente el ejemplo del chat y tu tarea anterior
- Implementa firmas digitales RSA, sólo en las transferencias que van del servidor a los clientes: el servidor firma cada mensaje y el cliente verifica que sea válido
- Los clientes tienen la llave pública del servidor (genera los archivos en formato PEM)
- Utiliza firma luego cifra
- No es necesario que cambies el esquema de cifrado para los mensajes (AES CBC o CTR)
- Los mensajes del cliente al servidor pueden seguir siendo verificados con la MAC

Curvas elípticas

- Son una alternativa moderna para RSA
- Hasta ahora sólo se ha hablado de criptografía asimétrica en términos de RSA
- La realidad es que la mayoría de algoritmos asimétricos ni siquiera soportan el cifrado de datos, ni cifran para generar firmas

Curvas elípticas

- Algunos algoritmos asimétricos generan la firma sin ningún cifrado y la verificación de la firma no tiene nada que ver con descifrar
- Por ejemplo, DSA (digital signing algorithm) es un algoritmo asimétrico que sólo se usa para firmas digitales
- RSA es un algoritmo histórico que se ha utilizado por décadas, pero se conocen varias debilidades del mismo y poco a poco se está dejando de usar

Curvas elípticas

- Las curvas elípticas se han utilizado para firmas digitales y para intercambio de llaves
- El fundamento matemático está fuera de los alcances de este curso
- Para firmar datos con curvas elípticas se utiliza el algoritmo **ECDSA**
- Tal como deben escogerse parámetros en RSA (tamaño de módulo, parámetro e), se deben escoger parámetros en operaciones basadas en curvas elípticas

Curvas elípticas

- El parámetro más importante a escoger es el tipo de curva
- Sin entrar en detalles, existen diferentes tipos de curvas que pueden ser utilizadas en diversas operaciones
- Para ECDSA, el módulo cryptography provee varias curvas aprobadas por NIST (que es una organización del gobierno de EU)
- A algunos criptografos les preocupa que dichas curvas hayan sido aprobadas por saberse que tienen debilidades que el gobierno de EU puede aprovechar

Curvas elípticas

- Dado que sólo se tiene ese tipo de curvas en cryptography, de usarse en un sistema en producción, se recomienda que se revise constantemente si aparecen vulnerabilidades conocidas
- En el siguiente ejemplo se muestra como firmar datos mediante la curva P-384 de NIST, la cual es referida como SECP384r1 en cryptography

Curvas elípticas

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import ec

private_key = ec.generate_private_key(
                    ec.SECP384R1(), default_backend())
public_key = private_key.public_key()

data = b"this is some data I'd like to sign"
signature = private_key.sign(data,
                             ec.ECDSA(hashes.SHA256()))

public_key.verify(signature, data, ec.ECDSA(hashes.SHA256()))
```

Certificados

Introducción

- Sirven para probar la pertenencia de llaves públicas
- Un problema de las llaves públicas es que pueden venir de cualquiera, incluso de entidades que se hacen pasar por otras
- Una posibilidad es que una "Entidad de confianza" sea quien entrega las llaves públicas
- Esta era la idea básica a principios de los noventas con el modelo PKI (Public Key Infrastructure)

Introducción

- En el modelo PKI inicial, se contaba con un registro global donde se almacenaban llaves públicas asociadas a su correspondiente entidad
- La llave pública de dicho registro se compartía de manera abierta
- Cualquiera con una copia válida de la llave pública del registro podía verificar otras llaves públicas
- El problema de la época con este modelo es que no era escalable, el registro podía crecer mucho y no ser manejable para las capacidades computacionales y de red de ese tiempo

Certificados

- Los certificados también se basan en la idea de tener una entidad de confianza
- Un certificado de llave pública son simplemente datos
- Suelen incluir una llave pública, metadatos relacionados con el propietario de la llave y una firma digital sobre todo el contenido del certificado, firma por la entidad de confianza (issuer)

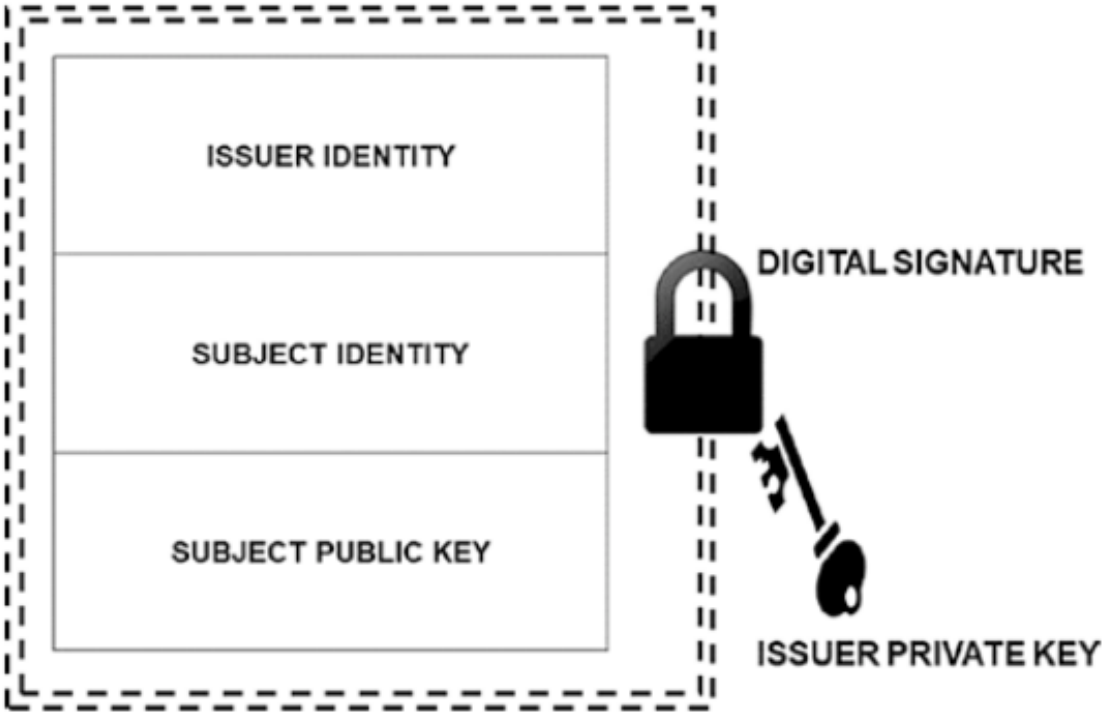
Certificados

- Los metadatos incluyen información como la identidad del propietario de la llave, la identidad del issuer, fecha de expiración, número de serie, entre otros
- La idea es ligar los metadatos, especialmente los de identidad, a la llave pública
- La identidad puede ser un nombre, email, una URL, u otro identificador acordado

Certificados

- En vez de simplemente compartir llaves públicas, lo que se comparte son certificados
- Si una entidad quiere un certificado:
  - Genera su par de llaves (la llave privada nunca debe compartirlaa nadie, ni a la entidad de confianza)
  - La entidad de confianza recibe la llave pública y empieza a generar el certificado, incluyendo los metadatos
  - Para completar el certificado, la entidad de confianza firma el certificado con su llave privada

Certificados



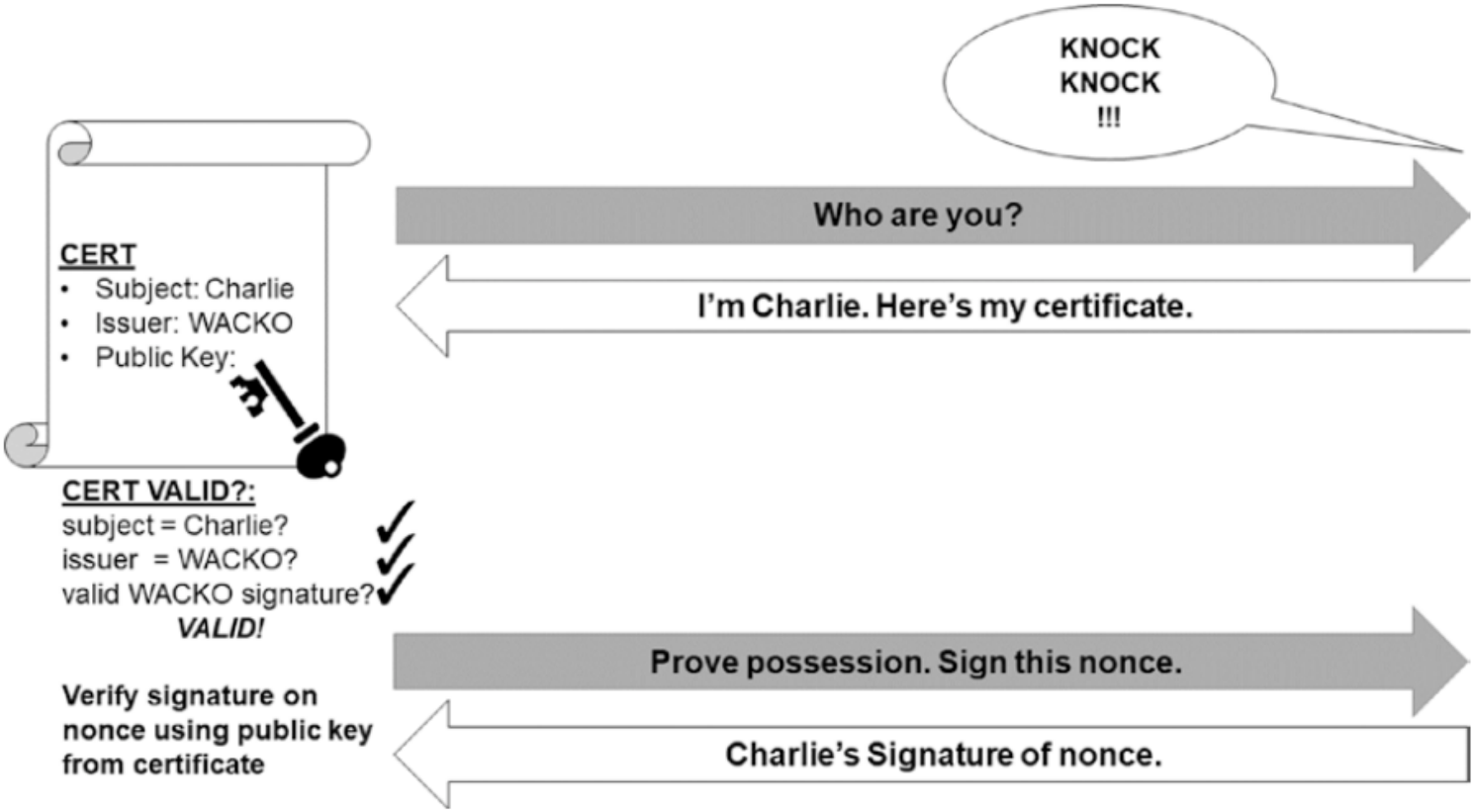
Certificados

- Existe un problema con el modelo visto hasta ahora
- Imagina que un compañero al que no conoces que se hace llar Charlie llega y te presenta su certificado, firmado por el profesor en el que confías
- Podrás, mediante la llave pública del maestro, la cual ya tenías, verificar la firma y sacar la información de identidad, que diga que es de Charlie
- Pero, ¿Cómo puedes saber que la persona que está ahí realmente es Charlie?
- Las llaves públicas cualquiera las tiene, alguien pudo haber generado un certificado falso engañando a la entidad de confianza

Certificados

- Esto se resuelve mediante un proceso reto-respuesta (challenge-response)
- Puedes pedirle al supuesto Charlie que te firme un mensaje de reto (llamado nonce), el cual debe firmar con su llave privada
- Lo siguiente es que verifiques dicho mensaje con la llave pública del certificado, si pasa la verificación entonces puedes confiar en que realmente sea Charlie (considerando que no se comprometió su llave privada)

Certificados



Certificados

- A continuación se verá una implementación no segura del ejemplo visto antes (no usar este código en producción)
- En este ejemplo los certificados tienen sólo 3 datos:
  - Nombre del issuer
  - Nombre del subject (propietario)
  - Llave pública
- Se realizarán 3 scripts:
  - script de issuer para generar un certificado dado el nombre de subject, la llave pública del subject y la llave privada del issuer
  - script de validador (en el ejemplo tú) que verifica el certificado, recibe el certificado, el nombre del subject y la llave pública del issuer
  - script del subject (Charlie) para responder al reto del validador, recibe la llave privada del subject y el reto enviado por el validador

Certificados y confianza

- A una entidad de confianza (issuer) se le suele llamar autoridad certificada (certificate authority CA)
- Los validadores de certificados deben decidir en que CAs confiarán
- De hecho, los CAs también tienen sus propios certificados con su nombre de identidad y su llave pública
- De esta forma, el campo issuer de un certificado, debe ser la misma identidad que el subject en el certificado del CA

Certificados y confianza

- Si el CA tienen un certificado ¿Quién lo firma?x
- Existe un concepto conocido como CA intermediario
- Un CA intermediario tiene su certificado firmado por un CA mayor
- Por ejemplo, si tu CA es tu maestro de criptografía, que en este caso sería un CA intermediario, el CA del maestro puede ser la facultad donde trabaja

Certificados y confianza

- Esta relación jerárquica puede continuar hasta llegar a un CA raíz con el certificado raíz
- ¿Quién firma el certificado del CA raíz?
- La respuesta es: él mismo
- Esto se conoce como un certificado auto-firmado (self-signed)

Certificados y confianza

- Nota que cualquiera puede generar un certificado auto-firmado
- Se debe tener mucho cuidado en qué certificados raíz confiar
- Básicamente si tu confías en un certificado que tiene alguna relación de descendencia con un certificado raíz, entonces también estás confiando en todos los certificados que se encuentran en la cadena de CAs, hasta llegar al CA raíz

Certificados y confianza

- El tener un certificado raíz confiable también facilita las cosas
- Sólo se necesita tener el certificado raíz, cualquier cadena de certificados generados a partir de ese certificado raíz puede ser verificado siguiendo la cadena en reversa

Certificados y confianza

- Por ejemplo, si tu tienes el certificado de la UV (CA raíz) en el que confías, puedes verificar que el certificado del maestro haya sido generado a través del certificado de la UV
- Esta revisión se haría viendo que el issuer del certificado del maestro es la facultad de Estadística e Informática y el issuer del certificado de la facultad es la UV
- En el último tema del curso se hablará de certificado s X.509, usados en TLS y se verá como CAs reales operan

Revocación de certificados

- Los certificados y las llaves públicas que contienen son muy poderosos
- Pero tienen una debilidad: ¿Cómo desactivas un certificado si la llave privada asociada se compromete?
- Revocar un certificado significa revertir la aprobación del issuer

Revocación de certificados

- Si la llave privada de un certificado generado por un CA se compromete, el CA debe determinar una forma de hacer saber a quienes tienen el certificado ya no es confiable
- Esto no es tan sencillo
- Una de las razones para dejar de utilizar un registro global de llaves públicas y usar certificados era poder hacer verificaciones offline
- ¿Cómo puede un sistema de verificación offline tener información de revocación?

Revocación de certificados

- La respuesta corta a la pregunta anterior es: no se puede
- Existen dos opciones:
  - El proceso de verificación debe tener un componente en tiempo real (online)
  - La revocación no puede ser actualizada en tiempo real

Revocación de certificados

- Ambas opciones están disponibles para certificados modernos con el protocolo OCSP (Online Certificate Status Protocol)
- Este protocolo revisa el estatus de un certificado en tiempo real
- Además el protocolo revisa CRLs (Certificate Revocation Lists) que son listas de certificados revocados, las cuales se van actualizando

Protección de llaves privadas

- Debido a lo complicado que es revocar certificados, lo mejor es proteger lo mejor posible las llaves privadas
- Si no se requieren firmas en tiempo real, las llaves privadas deberían estar en un ambiente seguro offline
- Si se requieren firmas en tiempo real y que la llave privada esté en un servidor, entonces deben aplicarse permisos mínimos y de preferencia ser cifrados (con cifradores simétricos) con contraseñas fuertes

Protección de llaves privadas

- Ten especial cuidado de no crear respaldos de las llaves privadas, pues el manejo de respaldos crea otro posible punto débil de seguridad
- También es buena idea mantener certificados con un tiempo de vida relativamente corto y rotarlos cuando sea necesario

Ataques de repetición

Introducción

- Replay attacks
- Este es un problema de seguridad que aplica tanto a MACs como a firmas
- Este ataque ocurre cuando un mensaje legitimo de una comunicación previa es utilizado por un atacante en un tiempo posterior, cuando el mensaje ya no debería ser válido

Ejemplo

- Considerando un contexto de guerra, y el mensaje "Atacaremos al amanecer"
- Los que generaron dicho mensaje lo pudieron proteger con un MAC o una firma digital
- Pero, ¿Qué evita que el bando enemigo intercepte el mensaje y lo reenvié días después?

Ejemplo

- Aunque el enemigo no pueda alterar el mensaje ni leerlo, puede reenviarlo
- Esto crea confusión y puede hacer que el enemigo adelante una estrategia

Prevención

- Todos los mensajes criptográficamente seguros tipicamente necesitan de algún tipo de componente único que lo distinga de los demás mensajes
- Este pedazo de datos usualmente se le llama **nonce**
- El nonce es un número aleatorio que le permite al receptor hacer un seguimiento de mensajes recibidos, de tal forma que si ve un nonce repetido, rechace el mensaje

Prevención

- El seguimiento de nonces puede ser problemático si existen miles o millones de mensajes
- ¿Cuál es la mejor estrategia? ¿Guardar los últimos N nonces? ¿Guardar cada nonce sólo por N minutos?
- Si eliges un patrón de comportamiento y el atacante logra determinarlo, entonces puede burlar la seguridad
- Por ejemplo, si decides guardar el nonce por 5 minutos y el atacante se da cuenta de esto, podría mandar mensajes que se enviaron hace 6 minutos

Prevención

- Algunos sistemas usan un timestamp en vez de nonces aleatorios
- Con un timestamp el receptor puede rechazar un menaje que sea demasiado viejo
- El problema de esta aproximación es que las partes en comunicación deben mantener sus relojes sincronizados, lo cual es más complejo de lo que parece
- Dado que no es posible tener una sincronización exacta, debido a retraso introducido por diferentes fuentes de latencia (velocidad de red, hops, etc.) el receptor debe aceptar mensajes dentro de una ventana de tiempo para lidiar con el error de tiempo
- ¿De qué tamaño debe ser la ventana de tiempo? otra vez el atacante puede encontrar patrones explotables

Prevención

- Una mejor solución es combinar nonces aleatorios y timestamps
- El timestamp ayuda a identificar mensajes realmente viejos
- Mientras que el nonce ayuda a identificar repeticiones dentro de la ventana de tiempo permitida
- Obviamente los nonces deben mantenerse el mismo tiempo que la ventana y no es necesario mantenerlos más tiempo, lo que ayuda a su manejo
- Esta aproximación también relaja los problemas de sincronización de relojes entre sistemas, pues no se necesita tanta exactitud