Nota principio de clase: la definicion del cifrado de bloque y de flujo es basicamente igual, salvo que el de flujo define mensajes de cifrado arbitrario, mientras que el de bloque define funciones de ENC/DEC de mensajes de tamano finito.

Criptografía y Seguridad

Criptografía: MACs y Cifrado Autentificado

Agenda

- Repaso CPA
- Ataques de Texto Cifrado Escogido
- MACs
- Funciones de hash criptograficas
- Cifrado autenticado

Criptosistema simétrico (repaso)

- Es una terna de algoritmos
  - Gen (algoritmo de generación de claves)
  - Enc (cifrado): $Enc_k(m)$
  - Dec (descifrado): $Dec_k(c)$
- Propiedades
  - La salida de Gen define K el espacio de claves.
  - La entrada de Enc define el espacio de mensajes
  - La entrada de Dec define el espacio de mensajes cifrados
  - Para todo m y k válidos: $Dec_k(Enc_k(m))=m$

Ataques de texto plano elegido

- Chosen Plain Text indistinguishability: $CPA_{A,\Pi}$
- Dado un nivel de seguridad n, un adversario A, y un Criptosistema $\Pi(n)$:

Medimos la seguridad de nuestras funciones criptograficas con este tipo de pruebas

1) Se genera una clave $k \leftarrow K$
2) A obtiene $f(x)=Enc_k(x)$ y emite (m_0, m_1)
3) Se genera $b \leftarrow \{ 0, 1 \}$
4) Se calculan $c_i \leftarrow Enc_k(m_b)$ y se le envían a A
5) A emite $b' = \{ 0, 1\}$

- $CPA_{A,\Pi} = 1$ si b = b' (A gana)

Si $Pr[CPA_{A,\Pi}=1] < ½ + neg(n) \Rightarrow \Pi$ es indisting.

Criptosistemas CPA-Secure (Rep.)

- Criptosistemas de flujo con tweaks
  - Utilizan un parámetro extra, llamado IV para evitar reutilizar exactamente la misma semila en el generador
    La idea es que el IV logra que nuestro criptosistema deje de ser deterministico
  - El IV es público pero se selecciona aleatoriamente.
- Criptosistemas de bloque
  - Con encadenamiento CBC, Counter, OFB, CFB.
    Son distintas formas de ir procesando serial o paralelamente nuestros bloques para cifrarlos de manera no determinista
  - Utilizando primitivas de cifrado de bloque seguras

Para reflexionar

- ¿Podríamos utilizarlo para cifrar los sueldos de empleados en una base de datos?
  - Supongamos un criptosistema de flujo

empleado	sueldo
2542	0xEA26969AA3F61CDD9EC68498DF031CA935EFA9C
2678	0xF4933E107178B88D8EE00F40E43A9A3C9D2EDF79
2789	0x155BBBE7A5442CBBCAB8F57DACF7212255F3641C
2890	0x4D9B47D518F2ED7DE04D601F1856767969A328E8

"Yo necesito que los database admin no puedan ver el sueldo de todas las personas".
En vez de guardar el sueldo, lo puedo cifrar. En este caso, lo unico que necesitaria proteger es la clave utilizada.
Sin embargo, desde un punto de vista de seguridad, el database admin podria copiar el sueldo de otro empleado.
Este sistema sigue la definicion de criptosistema seguro por la definicion de CPA, porque un atacante no puede obtener informacion a partir de los sueldos encriptados (como cual es mayor).
Sin embargo, un atacante podria conocer el sueldo de otro empleado, si es que tiene el mismo sueldo que es por ejemplo (porque los montos encriptados coinciden).

Para reflexionar

- ¿Podríamos utilizarlo para cifrar los sueldos de empleados en una base de datos?

empleado	sueldo
2542	0xEA26969AA3F61CDD9EC68498DF031CA93
2678	0x4D9B47D518F2ED7DE04D601F1856767969A328E8
2789	0x155BBBE7A5442CBBCAB8F57DACF7212255F3641C
2890	0x4D9B47D518F2ED7DE04D601F1856767969A328E8

¡Aumento de sueldo!

Supongamos que es un director

Comienza el juego

- Ciframos 'empleado||sueldo'

empleado	sueldo
2542	0xEA26969AA3F61CDD9EC68498DF031CA935EFA9C
2678	0xF4933E107178B88D8EE00F40E43A9A3C9D2EDF79
2789	0x155BBBE7A5442CBBCAB8F57DACF7212255F3641C
2890	0x4D9B47D518F2ED7DE04D601F1856767969A328E8

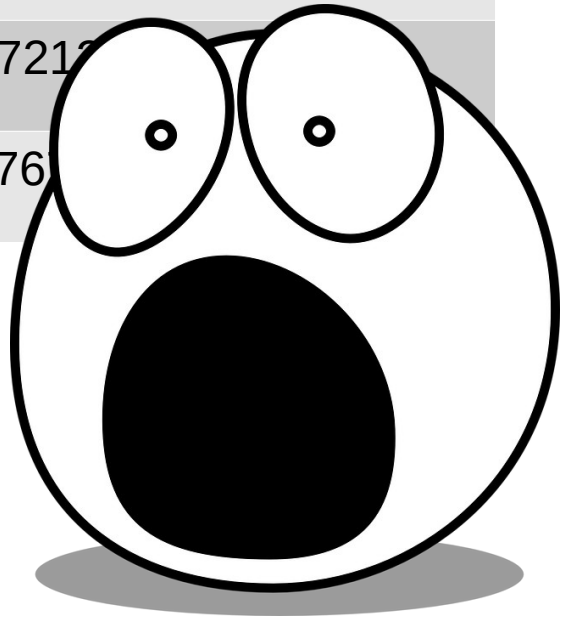
Termina el juego

- Ciframos 'empleado||sueldo'

Una solucion que puede surgir es tambien cifrar el ID de empleado.
Sin embargo, no es buena idea cifrar primary keys de una tabla, ya que va a haber problemas con los JOINs, ya que los mismos ID en distintas tablas se cifrarian de manera distinta (al ser no determinista).
Una mejor solucion es hacer que el campo de la columna cifrada tambien contenga al id adecuado de empleado por ejemplo.

empleado	sueldo
2542	0xEA26969AA3F61CDD9EC68498DF031CA935EFA9C
2678	0xF4933E107178B88D8EE00F40E43A9A3C9D2F69F0
2789	0x155BBBE7A5442CBBCAB8F57DACF721
2890	0x4D9B47D518F2ED7DE04D601F185676

Texto plano:
2678 $100,000



Backstage

- Se necesita conocer el formato
  - En este caso era IV || empl || sueldo
  - IV = byte[] (12 bytes)
  - empl = int (4 bytes)
  - sueldo = int (4 bytes)

Los primeros 12 bytes son el IV, porque estamos usando un criptosistema de flujo

0x F4933E107178B88D8EE00F40 E43A9A3C 9D2EDF79

El ataque de backstage consiste en que un atacante que conoce como se separan estos campos, puede ponerse a cambiar estos campos individualmente.

Backstage

- Considerar que:
  - $Enc_k(m) = G(k) \oplus m = c$
  - Definiendo $c' = c \oplus x$
    - $Dec_k(c') = m \oplus x$
- A contar bits:

Si elegimos un X cualquiera y le hacemos XOR al mensaje cifrado (c), obtenemos c'. Descifrando c', obtenemos el mensaje cifrado XOR X, por lo cual podemos alterar el mensaje original que se va a obtener cuando se quiera descifrar, solo alterando el mensaje cifrado.

2EDF79	00101110	11011111	01111001
xor			
12345	00000000	00110000	00111001
xor			
100000	00000001	10000110	10100000
=			
2F69F0	00101111	01101001	11110000

Basicamente lo que estamos haciendo aca es agarrar la parte del mensaje cifrado donde estaba el sueldo del empleado en la slide anterior (2EDF79).
Como yo se que mi sueldo es 12345, le hago XOR con eso al texto cifrado con mi sueldo.
Una vez que "limpie mi sueldo" del texto cifrado, hago XOR con el salario que quiero tener (100 000).
Cuando alguna persona desencripte el mensaje cifrado resultante (2F69F0), va a ver que mi sueldo es 100 000.

Ataques de texto cifrado escogido

- Chosen Ciphertext Attack: $CCA_{A,\Pi}$

Esta es la prueba de seguridad que toma en cuenta el tipo de ataque Backstage. En este ataque, en vez de inventar un texto plano, estamos inventando un texto cifrado.

- Dado un nivel de seguridad n, un adversario A, y un Criptosistema $\Pi(n)$:

El atacante va a tener una funcion G, que es la capacidad de descifrar cualquier mensaje arbitrario. El ataque consiste en que el atacante genere dos mensajes, y el sistema le devuelva un mensaje cifrado (parecido a antes, pero ahora tenemos la funcion G). Ninguno de los criptosistemas vistos hasta ahora pasa esta prueba.

1) Se genera una clave $k \leftarrow K$
2) A obtiene $f(x)=Enc_k(x)$ y $g(x) = Dec_k(x)$
3) A emite (m_0, m_1)
4) Se genera $b \leftarrow \{0, 1\}$ y se envía $c \leftarrow Enc_k(m_{bi})$
5) A emite $b' = \{ 0, 1\}$ (A no puede ver g(c))

A (el atacante) tiene la capacidad de tanto cifrar como descifrar mensajes. La unica limitacion del atacante es que, una vez que reciba el mensaje cifrado del sistema, no puede descifrarlo usando g(x).

- $CCA_{A,\Pi} = 1$ si b = b' (A gana)

Si $Pr[CCA_{A,\Pi}=1] < ½ + neg(n) \Rightarrow \Pi$ es indisting.

Ataques de texto cifrado escogido

- Ningún criptosistema visto hasta el momento es CCA-Secure

- Ejercicio: Demostrar que el cifrado de flujo en general no es CCA-Secure

- Ayuda: Considerar m0 = (0...0) y m1 = (1...1) y verificar que consultas se pueden hacer a g(x) al recibir c.

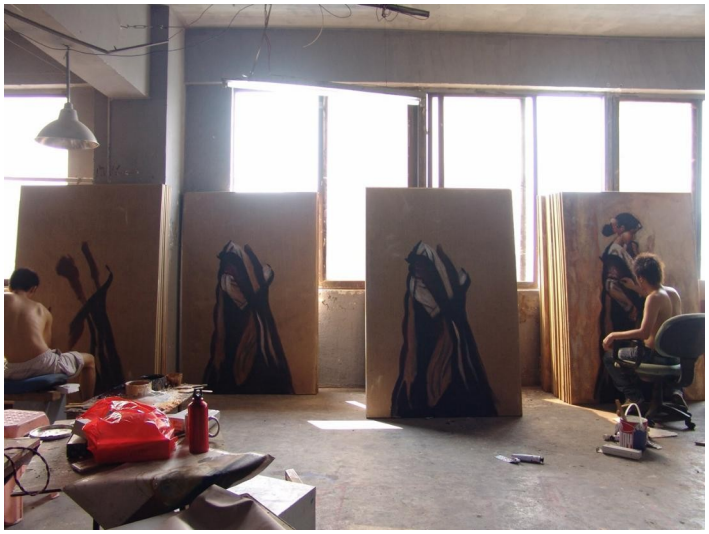
Yo puedo tomar c (el mensaje cifrado), y hacerle XOR con todos unos, obteniendo c'. Luego, llamo a la funcion de descifrado sobre c' (porque la prueba me permite llamar a la funcion de descifrado sobre cualquier mensaje menos sobre c), obteniendo m'.
Luego, haciendo m' XOR 1111...111111, entonces puedo obtener el mensaje original m.
Por este motivo, el cifrado de flujo no es CCA-Secure.

Ataques de texto cifrado escogido

- Es un problema no resuelto por el cifrado solamente.

Es una funcion que se especializa en detectar adulteraciones

- Requiere control de integridad
  - Consiste en identificar adulteraciones
  - Primitiva criptográfica: MAC
    - MAC = Message Authentication Code



MAC

- Es una terna de algoritmos
  - Gen (algoritmo de generación de claves)
  - Mac (etiquetado): $t \leftarrow Mac_k(m)$
  - Vrfy (verificación): $b = Vrfy_k(m, t)$

Funcion de MAC/etiquetado. El MAC funciona distinto a un encriptado. La idea es que a partir de una clave, genera una etiqueta.
La funcion de verificacion, nos dice a partir de una clave y una etiqueta, si matchean o no.

- Propiedades
  - La salida de Gen define K el espacio de claves.
  - Mac es una función no determinística
  - Vrfy retorna 0 (invalido) o 1 (valido)
  - Para todo m y k validos: $Vrfy_k(m, Mac_k(m)) = 1$

Dado un mensaje, no siempre voy a obtener el mismo MAC, pero siempre la funcion verify devolvera que hay match entre el MAC y el mensaje.

Seguridad de un MAC

- Message Authenticaiton Experiment: $\text{Mac-forge}_{A,\Pi}$
- Dado un nivel de seguridad n, un adversario A, y un Criptosistema $\Pi(n)$:

El atacante no conoce K, pero le damos un oraculo de etiquetado (para cualquier mensaje, puede conocer la etiqueta).
El atacante gana la prueba si logra emitir un mensaje y una etiqueta (para los cuales no haya usado el oraculo) tales que la funcion vrfy devuelva 1.

1) Se genera una clave $k \leftarrow K$
2) A obtiene $f(x)=Mac_k(x)$
3) A realiza las evaluaciones que quiera de f(x)

   (Llamese Q al conjunto de las evaluaciones)
4) A emite (m, t)

- $\text{Mac-Forge}_{A,\Pi} = 1$ si $Vrfy_k(m, t) = 1$ y m no pertenece a Q

Seguridad de un MAC

- $\text{Mac-Forge}_{A,\Pi} = 1$ si $\text{Vrfy}_k(m, t) = 1$ y m no pertenece a Q
- Si Pr[Mac-Forge = 1] < neg(n) =>
  - Π es infalsificable ante un ataque de mensaje escogido
- Obervaciones:
  - El adversario puede obtener un MAC para cualquier mensaje que elija
  - Se considera roto el MAC si el adversario puede falsificar cualquier mensaje, independientemente de si tiene sentido o no.

Ejercicio

- Considerar la seguridad de los siguientes Macs:

- $Mac_k(m) = G(k) \oplus m$ (G(-) gen. Pseudoaleat.)

El corolario que nos llevamos en el primer punto es que un MAC NO debe ser reversible. En este caso si lo era, y nos dejo obtener informacion adicional para lograr el MAC Forge

Definimos la funcion de etiquetado como tomar el mensaje original, y hacer un XOR con una secuencia pseudoaleatoria.
La pregunta entonces es si podemos usar un criptosistema de flujo como MAC.
En principio ya sabemos que no porque estos criptosistemas no pasan la prueba CCA.
Hacemos un MAC Forge de la siguiente forma:
1) Evaluo MAC(m=000...0) -> obtengo G(k) XOR m = G(k) XOR 000...0 = G(k)
2) Emito m' != m
3) En el paso anterior obtuve G(k), entonces lo uso para emitir un t' para m', haciendo m' XOR G(k)
4) Emito m' y t'. Estos van a pasar VRFY, por lo que logre hacer un MAC Forge

- $Mac_k(m) = k \oplus first\_k\_bits(m)$

El corolario de este segundo punto es que un MAC tiene que tener en cuenta TODOS los bits del mensaje. Si hay bits del mensaje que no se tienen en cuenta (como en este caso), entonces es facil emitir un mensaje m' que tenga el mismo MAC.

Este caso es casi igual al anterior.
1) Evaluo el MAC (m=000...0), esto nos devuelve un K
2) Emito el (m=000...01, K). Esto va a pasar la funcion VRFY, por lo que logre hacer un MAC Forge

- $Mac_k(m) = Enc_k(|m|)$ (Enc, CPA-Secure)

(cifrar con la clave k la longitud del mensaje, con una funcion ENC que pase la prueba CPA Secure)

El corolario del tercer caso es que cualquier funcion de MAC que ignore aunque sea un bit del contenido del mensaje es falsificable. En este caso se tiene en cuenta un atributo del mensaje (la longitud), pero no se tienen en cuenta los bits del mismo.

La dificultad que tenemos aca es que ENC no es deterministico (por ser CPA Secure). Lo bueno para el atacante es que la funcion VRFY si debe ser deterministica.
En este caso la funcion VRFY evaluaria que t sea igual a ENC de la longitud del mensaje que le llego

Como construir un MAC

- 1ra opción: A partir de una primitiva de cifrado de bloque
- Sea F una función pseudoaleatoria
- CBC-MAC (tamaño fijo de mensajes):
  - Gen: $k \leftarrow \{0, 1\}^n$
  - Mac:

1) Se divide el mensaje en partes iguales
2) Se define el estado inicial (todos 0)
3) Se define un nuevo estado ti usando el estado anterior y cifrando con el bloque de mensaje
4) El MAC resultante es el estado final, y su longitud es igual a la longitud de un bloque

Esta construccion de MAC es infalsificable si y solo si la definimos de mensaje de tamaño fijo. Si permitimos mensajes de tamaño arbitrario ya es falsificable.
Esto se probaria usando un ataque de extension (que consiste generar un mensaje que sea un prefijo de otro)

  - sea $m = m_1||m_2||m_3...||m_j$, Sea $t_0 = 00...0$
  - $t_i = F_k(t_{i-1} \oplus m_i)$.
  - $Mac_k(m) = t_j$
  - Vrfy: $Vrfy_k(m, t) = 1 \leftrightarrow t = mac_k(m)$

CBC-MAC

- La Construcción anterior es infalsificable SOLO si se permiten mensajes de una misma longitud

Demostrarlo a través de una prueba de seguridad

CBC-MAC

- La Construcción anterior es infalsificable SOLO si se permiten mensajes de una misma longitud

- Considerar: $m_1 = A||B$, $m_2 = A$

  (A y B tienen el tamaño de un bloque)

- Obtener t_1 y t_2

m2 es un prefijo de m1.
El calculo de MAC de m2 va a tener un solo bloque. Va a ser el resultado de cifrar con la clave inicial (todos 0) el primer bloque. Osea que la etiqueta t2 es el resultado de cifrar el bloque A con el estado inicial.
El calculo de MAC de m1 comienza cifrando A con la clave inicial, que es igual al resultado anterior, por lo que ya sabemos cuanto vale este estado intermedio. El calculo del MAC termina cifrando este estado con el bloque B, y ese es nuestro MAC final.

- Intentar: $m_3 = A||B||(A \text{ xor } t_1)$ $\quad t_3 = t_2$

  (t_2 = A, t_1 = B)

aca nos queda t1 XOR A XOR t1. La entrada a la funcion pseudoaleatoria va a ser A. Vamos a obtener un texto cifrado cualquiera. Lo importante es que la salida (t3) va a ser igual al estado intermedio del primer bloque (t2).

La seguridad de CBC-MAC radica en que yo solo tengo acceso al estado final, no a los estados intermedios.
En este caso, con m1 y m2 obtuve t1 y t2, que son todos los estados intermedios que se usan al calcular el MAC de m3.
Lo que paso en este caso es que yo averigue el MAC de A||B||(A xor t1) = t3, solo utilizando la funcion MAC con otros mensajes m1 y m2, por lo que logre hacer un MAC-FORGE.

CBC-MAC

Dado el tipo de ataque de la slide anterior, se busco "arreglar" el algoritmo. Se puede elegir cualquiera de estas 3 opciones.

- Extensiones seguras para mensajes arbitrarios:
  - Computar $k' = F_k(|m|)$

    Calcular $T_i = F_{k'}(t_{i-1} \oplus m_i)$. $MAC_k(m) = t_j$

    Si yo emito m1, m2 tal que m2 es prefijo de m1, no voy a poder acceder al estado interno de m1 usando solo MAC(m2).

  - Computar $m' = |m| \,||\, m$

    $Mac_k(m) = CBC\text{-}MAC_k(m')$

    Aplicar el mismo algoritmo, pero en vez de hacerlo sobre el mensaje, hacerlo sobre la longitud del mensaje concatenada con el mensaje. Como el primer bloque ya es distinto, el resto de los estados intermedios ya no se comparten con mensajes de otra longitud.

  - Generar dos claves k_1 y k_2
    - Calcular $t' = CBC\text{-}MAC_{k1}(m)$
    - $t = F_{k2}(t')$

    Calcular el CBC-MAC del mensaje como la veniamos calculando. En lugar de usar eso como etiqueta, encriptar el resultado y usar eso como etiqueta.
    Esto tambien funciona, pero la desventaja es que la clave tiene que tener el doble de longitud, por lo que se usa poco.

CBC-MAC

- Ejercicio:

Demostrar que este MAC no es seguro

$$Mac_k(m) = CBC\text{-}MAC_k(m')$$

$$m' = m \,||\, |m|$$

(CBC-MAC donde al mensaje se le agrega la longitud como sufijo en lugar de prefijo)

CBC-MAC

- Ejercicio:

En una de las soluciones anteriores, concatenabamos la longitud del mensaje con el mensaje. Sin embargo, si lo concatenamos al reves, descubrimos que este no es un MAC seguro.

$$Mac_k(m) = CBC\text{-}MAC_k(m')$$

$$m' = m \,||\, |m|$$

Considerar:

m_1=AAA → m_1'=AAA3

m_2=BBB → m_2'=BBB3

m_3=AAA3CC → m_3'=AAA3CC6

La etiqueta de m1' nos esta dando un estado intermedio de la etiqueta de m3'

Intentar m=BBB3XC

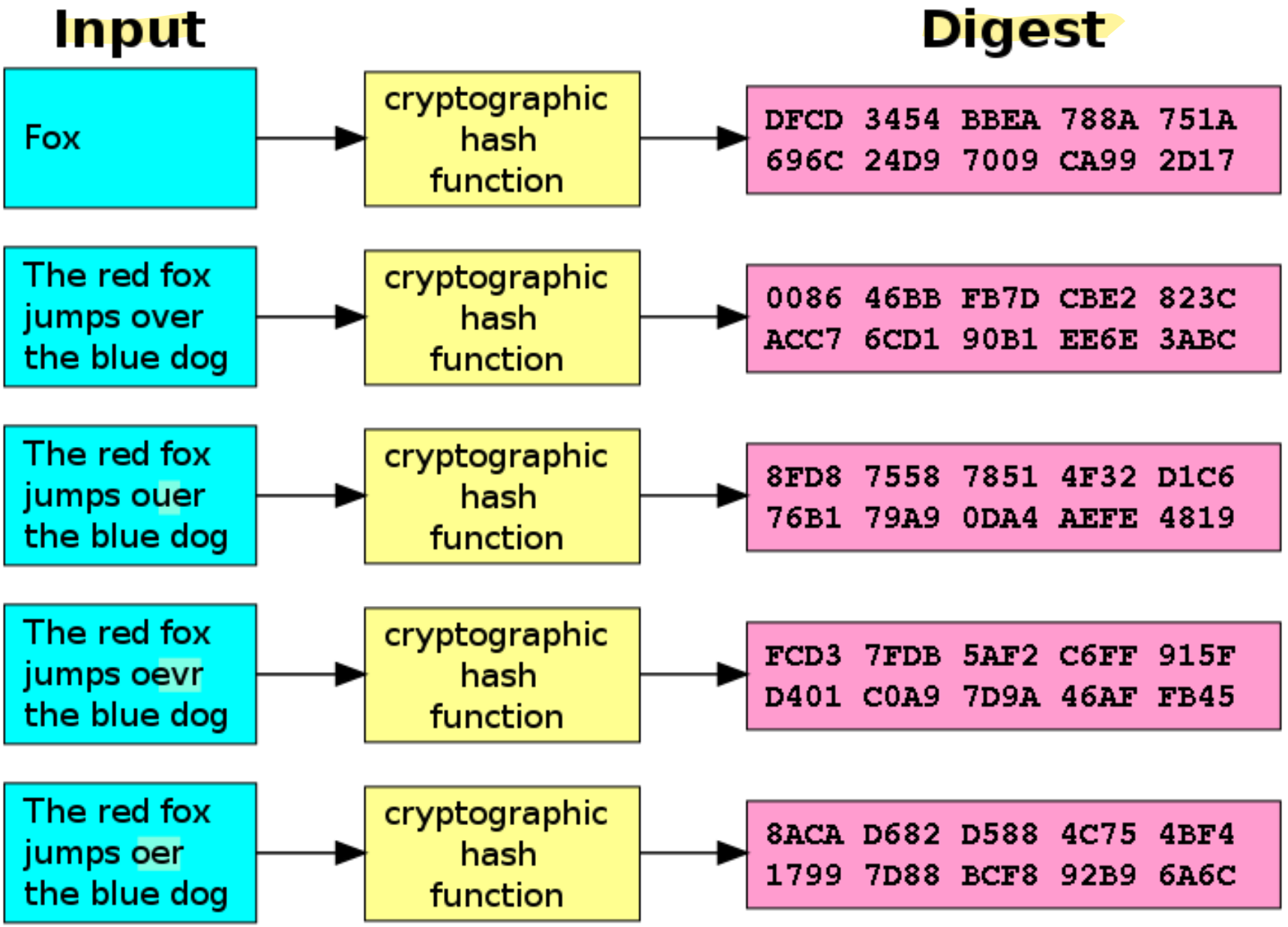
donde $X=t_1$ xor t_2 xor C

Funciones de hash criptograficas

- Son pares de algoritmos:
  - Gen: $s \leftarrow S$
  - Hash: $h = H^s(m) \varepsilon \{0,1\}^L$
    - L es la longitud del hash
- Diferencia importante:
  - s no es una clave. Simplemente es un selector.
  - En muchas implementaciones $S = \{ s_0 \}$
    - Solo hay una función de hash en la familia
- Se las llama funciones de resumen
- Analogas a los MACs, pero sin clave

Parten de una idea parecida a los MAC, la idea es dado un mensaje, generar una etiqueta.
La diferencia principal con los MAC, es que los hashes criptograficos NO usan ningun tipo de clave.
Se habla de una FAMILIA de funciones de hash (para pruebas matematicas), y se elige una a usar. En la practica, generalmente se usa una unica funcion de hash.
Algo bueno del hash es que es MUY dificil que haya dos cosas con las mismas etiquetas.

Etiquetadores universales



Colisiones

- Colision:
  - $x \neq x'$ y $h(x)=h(x')$
- Si hay n+1 mensajes y n valores de salida, existe al menos una colisión

Toda funcion de hash que se usa en la practica TIENE COLISIONES

Propiedades (definición informal)

- Resistencia a preimagenes
  - Para todo *y*, es computacionalmente imposible hallar *x* / *h*(*x*) = *y*

Para toda etiqueta, es computacionalmente imposible encontrar un mensaje que genere esa etiqueta.

Computacionalmente imposible significa que la unica forma de hacerlo es con fuerza bruta.

Propiedades (definición informal)

- Resistencia a segundas imagenes
  - Para todo x, es computacionalmente imposible hallar x' ≠ x / h(x')=h(x)

Dado un mensaje, es computacionalmente imposible producir un segundo mensaje con la misma etiqueta.

Propiedades (definición informal)

- Resistencia a colisiones
  - Es computacionalmente imposible hallar x, x' / h(x) = h(x') y x ≠ x'

No se puede hallar dos x y x' distintos que tengan el mismo hash.
Esta propiedad INCLUYE a las otras dos propiedades.

Resistencia a colisiones

- Collision resistance: $\text{Hash-Coll}_{A,H}$

La prueba consiste en que me dan una funcion de hash, y tengo que producir dos mensajes cuyo hash sea el mismo. Si puedo hacer esto, paso la prueba.

- Dado un nivel de seguridad n, un adversario A, y una familia de funciones de hash $H^x(n)$:

1) Se selecciona una funcíon s ← S
2) A obtiene s, y por lo tanto H(x) =H^s(x)
3) A emite x, x'

Los pasos 1 y 2 implican que "no vale que yo me ponga a buscar colisiones con anterioridad y emitir esas. Hay que empezar a buscar colisiones cuando me dan la funcion".
Esto se dice porque la prueba no mide que existan colisiones (porque ya se sabe que existen). Lo que evalua la prueba es como yo las puedo obtener.

- $\text{Hash-Coll}_{A,H}$ = 1 si x != x' y $H^s(x)=H^s(x')$ (A encuentra una colisión)
- Si Pr[$\text{Hash-Coll}_{A,H}$ = 1] < neg(n), la función H es libre de colisiones

Modelo general iterativo

Esta pensada para calcular hashes sobre un mensaje muy largo. La idea es que uno no sabe el largo del mensaje hasta que lo termina de procesar. Por eso se concatena la longitud al final.

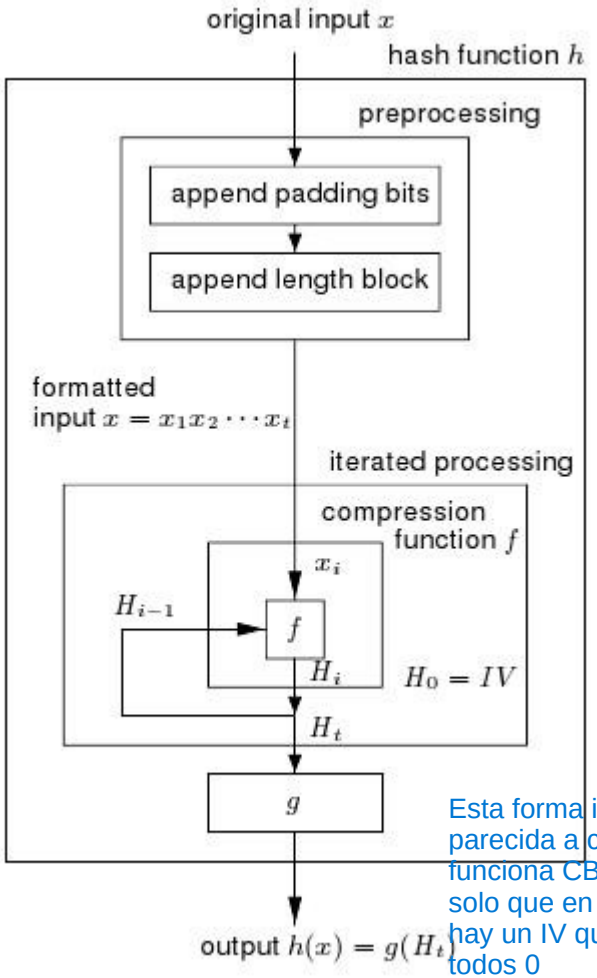
- Propuesto por Merkle en 1989
- Utilizado por muchas funciones

Usan muchas funciones de Hash para construir una "compresion"

- Preprocesamiento
  - Ajustar el tamaño del mensaje
  - Agregar bloque con tamaño

Primero se rellena el mensaje hasta cierta longitud estandar. Luego se concatena la ongitud del bloque al final

- Función de compresión
  - Similar al hash pero opera sobre bloques pequeños
  - La seguridad está dada por la función de compresión



Esta forma iterativa es parecida a como funciona CBC-MAC, solo que en este caso hay un IV que no son todos 0

MD5

Este tipo de funciones se usa mucho en protocolos P2P (peer to peer) como torrent. Entonces estos programas empezaron a usar los hashes para indexar los archivos en vez de usar los nombres de archivos (que son distintos para todas las compus de cada uno)

- Entrada: secuencia de hasta 2^{64} bits
- Salida: secuencia de 128 bits

MD5 actualmente esta roto, se pueden generar millones de colisiones por segundo.

- Aplicación iterativa
- Ejemplos:
  - “” -> `d41d8cd98f00b204e9800998ecf8427e`
  - “a” -> `0cc175b9c0f1b6a831c399e269772661`
  - “abc” -> `900150983cd24fb0d6963f7d28e17f72`

Ver que strings similares tienen hashes totalmente distintos

SHA-1

- Entrada: secuencia de hasta 2^{64} bits
- Salida: secuencia de 160 bits

Se descubrio un algoritmo para encontrar colisiones en SHA-1 que permite encontrar en 2^63 operaciones (que no es muy poco pero es computable)

- Aplicación iterativa
- Se construye sobre la base de MD5
- Ejemplos:
  - “” -> `da39a3ee5e6b4b0d3255bfef95601890afd80709`
  - “a” -> `86f7e437faa5a7fce15d1ddcb9eaeaea377667b8`
  - “abc” -> `a9993e364706816aba3e25717850c26c9cd0d89d`

SHA-3

- Entrada: secuencia de hasta 2^{64} bits
- Salida: secuencia de 224,256,384 o 512 bits
- Estandarizado por NIST en 2013
- Aplicación modelo esponja
- Ejemplos:
  - “” -> `a7ffc6f8bf1ed76651c14756a061d662f580ff4de43b49fa82d80a4b80f8434a`
  - “a” -> `80084bf2fba02475726feb2cab2d8215eab14bc6bdd8bfb2c8151257032ecd8b`
  - “abc” → `3a985da74fe225b2045c172d6bd390bd855f086e3e9d525b46bfe24511431532`

SHA-3 es la funcion de hash que se usa hoy en dia, es la que "arreglo" los problemas de SHA-1

Funciones de Hash y MACs

- Es posible construir un MAC a partir de una función de hash
- HMAC:

Generalmente cuando se necesita calcular MACs en la vida real, se usa HMAC en vez de CBC-MAC. El que usariamos es HMAC-SHA3.

  - Dado H(x) función de hash libre de colisiones
  - HMACk(x) = {

El MAC resultado es la funcion de hash aplicada a la clave XOR outer-pad concatenada a la clave XOR inner-pad concatenada al mensaje.

GEN: $k \leftarrow K, s \leftarrow S$

MAC: $t = H^s((k \oplus opad) \,||\, H^s((k \oplus ipad) \,||\, m))$

$opad = 0x36...36 \quad ipad = 0x5c5c...5c$

opad e ipad siempre valen estos valores (aunque el unico requisito es que sean dos valores distintos cualesquiera) la idea de fijarlos es que no sean un secreto estos numeros

  }

  Es infalsificable (MAC-Forge)

Seguridad de funciones de Hash

- Objetivos del atacante. Sea h: A → B
  - Preimagenes: dado $y \in B$, hallar x / $h(x) = y$
    - *Búsqueda por fuerza bruta: requiere |B| intentos*
  - *Segundas imagenes: dado (x,y) / h(x)=y, hallar x' / h(x')=y*
    - *Búsqueda por fuerza bruta: requiere |B| intentos*
  - *Colisiones: hallar x, x' / h(x)=h(x')*
    - *Búsqueda por fuerza bruta: requere $|B|^{1/2}$*
    - *Paradoja del cumpleaños*

Funciones de hash en la práctica

- Tamaño mínimo de salida: 160 bits
  - Complejidad de un ataque por fuerza bruta:
    - Imágenes: 2^{160} operaciones
    - Colisiones: 2^{80} operaciones
- Primitivas recomendadas
  - ~~MD5 → 128 bits~~ Quebrada

    Aun si MD5 no estuviese quebrada, el valor teorico para computar colisiones seria de 2^64, lo que es computable actualmente.
  - SHA-1 → 160bits

    SHA-1 tiene un valor teorico para calcular colisiones en 2^80 operaciones. Todavia no tenemos esta capacidad computacional, pero en 10 años podriamos tenerla.
  - SHA-2/256 → 256 bits
  - SHA-3/256-512: Recientemente seleccionada
    - Antes conocida como Kekkak

      Se recomienda usar SHA-3 para datos que deben ser guardados por mucho tiempo (30 años)
    - Estandard recomendado para nuevos proyectos

Privacidad e integridad

- Tres formas:

Formas de juntar un MAC con un criptosistema.
SE NECESITA SIEMPRE QUE K1 != K2

- Cifrar y autentificar:
  - $c \leftarrow Enc_{k1}(m), t \leftarrow Mac_{k2}(m)$

Opcion 1: cifro el mensaje y luego calculo el MAC sobre el mensaje no cifrado. No tiene garantias de seguridad, porque no hay ninguna propiedad que nos garantice que el calculo de etiqueta del mensaje no nos brinde informacion del mensaje.

t puede brindar información de m

- Autentificar, luego cifrar
  - $c \leftarrow Enc_{k1}(m \,||\, Mac_{k2}(m))$,

Opcion 2: es la que implementa ssh. Se cifra el mensaje junto al MAC del mismo. No tiene una demostracion matematica general.

Puede ser seguro, requiere prueba de seguridad

- Cifrar, luego Autentificar
  - $c \leftarrow Enc_{k1}(m), t \leftarrow Mac_{k2}(c)$

Opcion 3: hay una demostracion matematica general que garantiza que funciona siempre.
Primero cifro el mensaje, luego calculo el MAC sobre el mensaje cifrado. Se llama cifrado-autenticado.

Siempre es seguro

Cifrado Autentificado

- Combina cifrado y control de integridad
- Sean $\Pi_e(Gen_e, Enc, Dec)$ un criptosistema CPA-Secure y $\Pi_m(Gen_m, mac, vrfy)$ un mac infalsificable.
- Se define el criptosistema:
  - Gen: $k_1 \leftarrow Gen_e$, $k_2 \leftarrow Gen_m$ Se generan dos claves
  - Enc: $c \leftarrow Enc_{k1}(m)$, $t \leftarrow Mac_{k2}(c)$ Se encripta el mensaje. Se genera un MAC sobre el mensaje encriptado
  - Dec: Si $vrfy_{k2}(c, t) = 1 \rightarrow m = Dec_{k1}(c)$

    Sino, m = / (fallo)

    Primero se aplica la funcion VRFY. Luego se descifra. Si alguien modifico o la etiqueta o el mensaje, VRFY falla.
- El criptosistema resultante es CCA-Secure

Encadenamiento CCM

Trata de usar un CBC-MAC y cifrar usando modo Counter. La ventaja es que el CBC-MAC y el Counter pueden usar la misma clave, por lo que no hace falta generar dos claves.
La gran desventaja es que hay que procesar el mensaje dos veces. Primero recorremos todo el mensaje para calcular el CBC-MAC, y luego lo appeandeamos al mensaje y encriptamos todo eso.

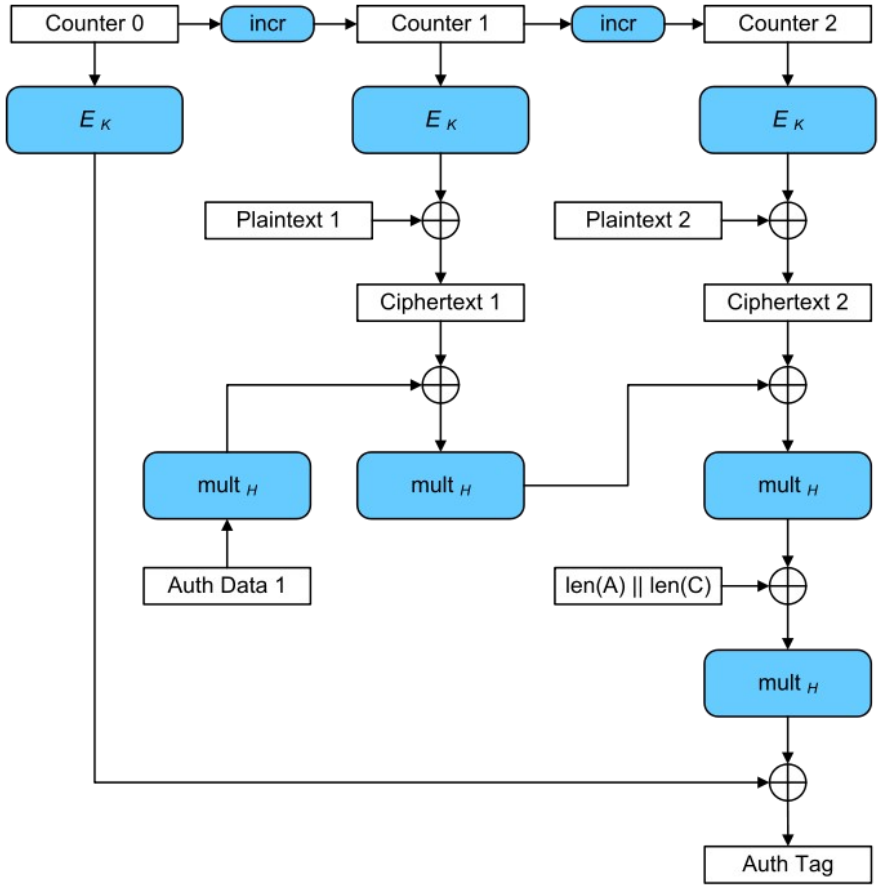
- Significa Counter with CBC-MAC
  - “Authenticate-then-encrypt”
  - $e_k(\text{cbc-mac}_k(M) \,||\, m)$
- Existe una prueba de seguridad especifica
  - Si el IV y el nonce no coinciden ni se reutilizan, la construcción es CCA-Secure usando LA MISMA CLAVE

    (ver material adicional en Campus).
- Ejercicio, esquematizar un cifrado utilizando AES-CCM

Encadenamiento GCM

Plantea una forma de resolver la desventaja del anterior, procesando una sola vez el mensaje

- Forma de encadenar un criptosistema de bloque
- Provee cifrado autenticado



Cifrado modo counter

Auth tag (MAC) GHASH: $GF(2^{128})$

$H = E_k(0000...0000)$

$Mult_h(x) = Mult(x, h)$

$Mult(x, y) = x * y \bmod x^{128}+x^7+x^2+x+1$

Convierto a los numeros en polinomios, los divido por otros polinomios y el resultado es un polinomio. Como estaba patentado, habia que pagar para usarlo, por lo que no lo uso nadie.
10 años despues, los autores liberaron la patente y hoy en dia es el modo de encadenamiento que todos usan.

Aplicaciones prácticas

- En términos generales, solo utilizar cifrado autentificado en sistemas reales
  - El cifrado “normal” puede ser manipulado
  - En aplicaciones reales, el requerimiento de privacidad lleva implícito el de integridad
  - Muy pocas veces se tiene control sobre el material que va a ser cifrado y descifrado. La solución debería funcionar independientemente de ésto.

Lectura Recomendada

Capítulo 4

Introduction to Modern Cryptography
Katz & Lindell