

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
Autenticado

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): $\text{Enc}_k(m)$
 - **Dec** (descifrado): $\text{Dec}_k(c)$
- 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: $\text{Dec}_k(\text{Enc}_k(m))=m$

Ataques de texto plano elegido

- Chosen Plain Text indistinguishability: $\text{CPA}_{A,\Pi}$
- 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 $f(x) = \text{Enc}_k(x)$ y emite (m_0, m_1)
- 3) Se genera $b \leftarrow \{0, 1\}$
- 4) Se calculan $c_i \leftarrow \text{Enc}_k(m_b)$ y se le envían a A
- 5) A emite $b' = \{0, 1\}$

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

Si $\Pr[\text{CPA}_{A,\Pi}=1] < \frac{1}{2} + \text{neg}(n) \Rightarrow \Pi$ es indisting.

Criptosistemas CPA-Secure (Rep.)

- Criptosistemas de flujo con tweaks

- Utilizan un parámetro extra, llamado IV para evitar reutilizar exactamente la misma semilla en el generador

La idea es que el IV logra que nuestro criptosistema deje de ser determinístico

- 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	0xEA26969AA3F61CDD9EC68498DF031CA950
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

Texto plano:
2678 \$12,345

Termina el juego

- Ciframos 'empleado||sueldo'

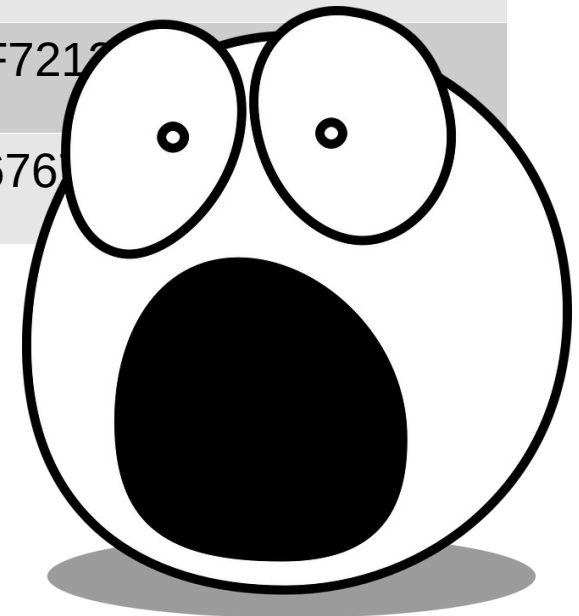
Una solución que puede surgir es también cifrar el ID de empleado.

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 cifrarían de manera distinta (al ser no determinista).

Una mejor solución es hacer que el campo de la columna cifrada también contenga al ID adecuado de empleado por ejemplo.

empleado	sueldo
2542	0xEA26969AA3F61CDD9EC68498DF031CA935EFA9C
2678	0xF4933E107178B88D8EE00F40E43A9A3C9D2F69F0
2789	0x155BBBE7A5442CBBCAB8F57DACF7212
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 $c' = c \oplus x$

- $Dec_k(c') = m \oplus x$

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.

- A contar bits:

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 descrypte el mensaje cifrado resultante (2F69F0), va a ver que mi sueldo es 100 000.

Ataques de texto cifrado escogido

- Chosen Ciphertext Attack: $\text{CCA}_{A,\Pi}$
- Dado un nivel de seguridad n , un adversario A , y un Criptosistema $\Pi(n)$:

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.

El atacante va a tener una función 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 función G). Ninguno de los criptosistemas vistos hasta ahora pasa esta prueba.

1) Se genera una clave $k \leftarrow K$

2) A obtiene $f(x) = \text{Enc}_k(x)$ y $g(x) = \text{Dec}_k(x)$

3) A emite (m_0, m_1)

4) Se genera $b \leftarrow \{0, 1\}$ y se envía $c \leftarrow \text{Enc}_k(m_{b_i})$

5) A emite $b' \in \{0, 1\}$ (A no puede ver $g(c)$)

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

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

Si $\Pr[\text{CCA}_{A,\Pi}=1] < \frac{1}{2} + \text{neg}(n) \Rightarrow \Pi$ 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 $m_0 = (0...0)$ y $m_1 = (1...1)$ 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 función de descifrado sobre c' (porque la prueba me permite llamar a la función 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): $t \leftarrow \text{Mac}_k(m)$

- Vrfy (verificación): $b = \text{Vrfy}_k(m, t)$

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: $\text{Vrfy}_k(m, \text{Mac}_k(m)) = 1$

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 Authentication Experiment: **Mac-forge**_{A,Π}
- 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 $f(x) = \text{Mac}_k(x)$
- 3) A realiza las evaluaciones que quiera de $f(x)$
(Llamese Q al conjunto de las evaluaciones)
- 4) A emite (m, t)

- **Mac-Forge**_{A,Π} = 1 si $\text{Vrfy}_k(m, t) = 1$ 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 $\Pr[\text{Mac-Forge} = 1] < \text{neg}(n) \Rightarrow$
 - Π es infalsificable ante un ataque de mensaje escogido
- Observaciones:
 - 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:
 - $\text{Mac}_k(m) = G(k) \oplus m$ ($G(-)$ gen. Pseudoaleat.)
Definimos la funcion de etiquetado como tomar el mensaje original, y hacer un XOR con una secuencia pseudoaleatoria.
Esta MAC no es segura, ya que
 - $\text{Mac}_k(m) = k \oplus \text{first_k_bits}(m)$
 - $\text{Mac}_k(m) = \text{Enc}_k(|m|)$ (Enc, CPA-Secure)

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

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

CBC-MAC

- Extensiones seguras para mensajes arbitrarios:
 - Computar $k' = F_k(|m|)$
Calcular $T_i = F_{k'}(t_{i-1} \oplus m_i)$. $MAC_k(m) = t_j$
 - Computar $m' = |m| || m$
 $Mac_k(m) = CBC-MAC_k(m')$
 - Generar dos claves k_1 y k_2
 - Calcular $t' = CBC-MAC_{k_1}(m)$
 - $t = F_{k_2}(t')$

CBC-MAC

- Ejercicio:

Demostrar que este MAC no es seguro

$$\text{Mac}_k(m) = \text{CBC-MAC}_k(m')$$

$$m' = m \parallel |m|$$

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

CBC-MAC

- Ejercicio:

$$\text{Mac}_k(m) = \text{CBC-MAC}_k(m')$$

$$m' = m \parallel |m|$$

Considerar:

$$m_1 = \text{AAA} \quad \rightarrow \quad m_1' = \text{AAA3}$$

$$m_2 = \text{BBB} \quad \rightarrow \quad m_2' = \text{BBB3}$$

$$m_3 = \text{AAA3CC} \quad \rightarrow \quad m_3' = \text{AAA3CC6}$$

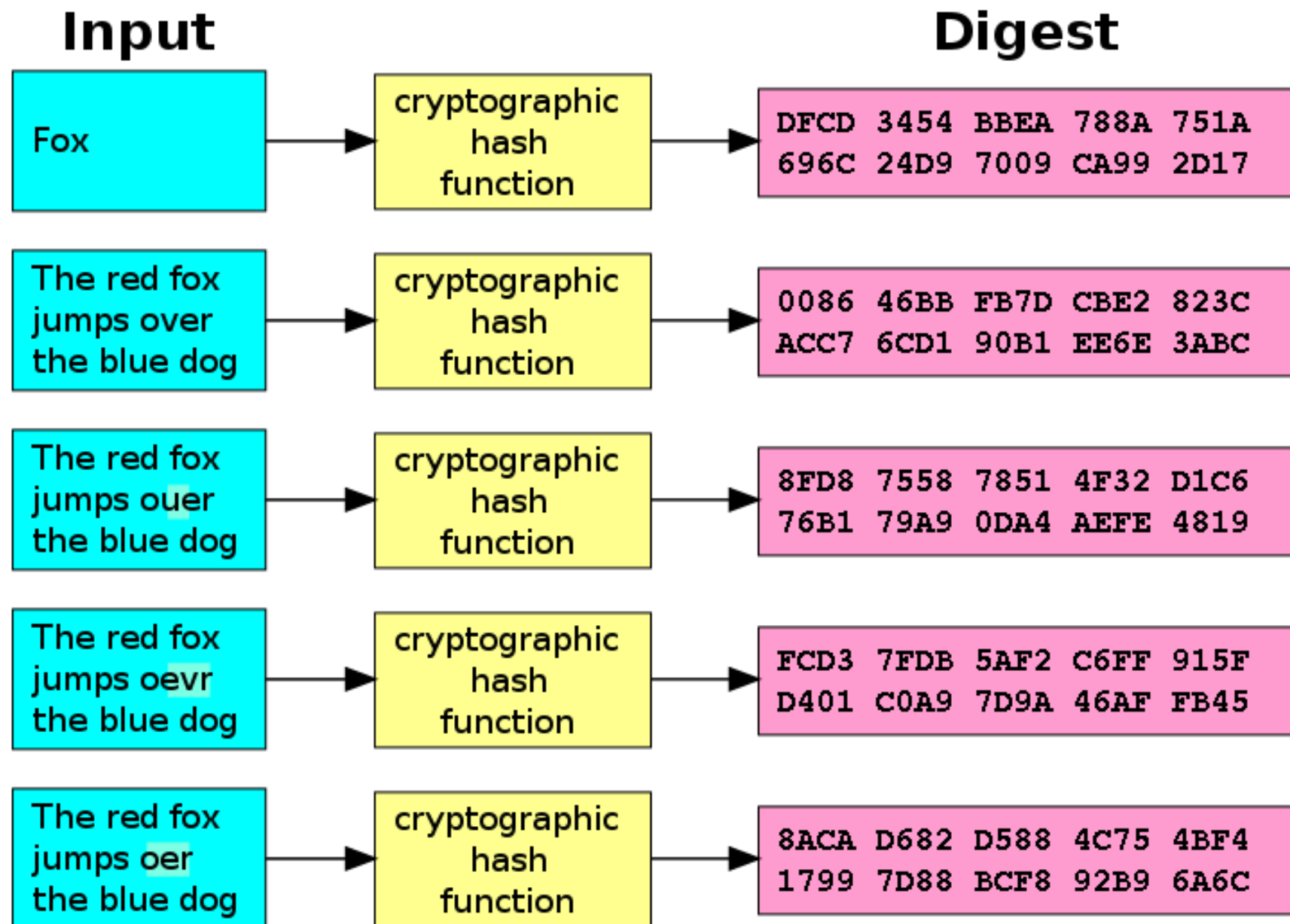
Intentar $m = \text{BBB3XC}$

donde $X = t_1 \text{ xor } t_2 \text{ xor } C$

Funciones de hash criptograficas

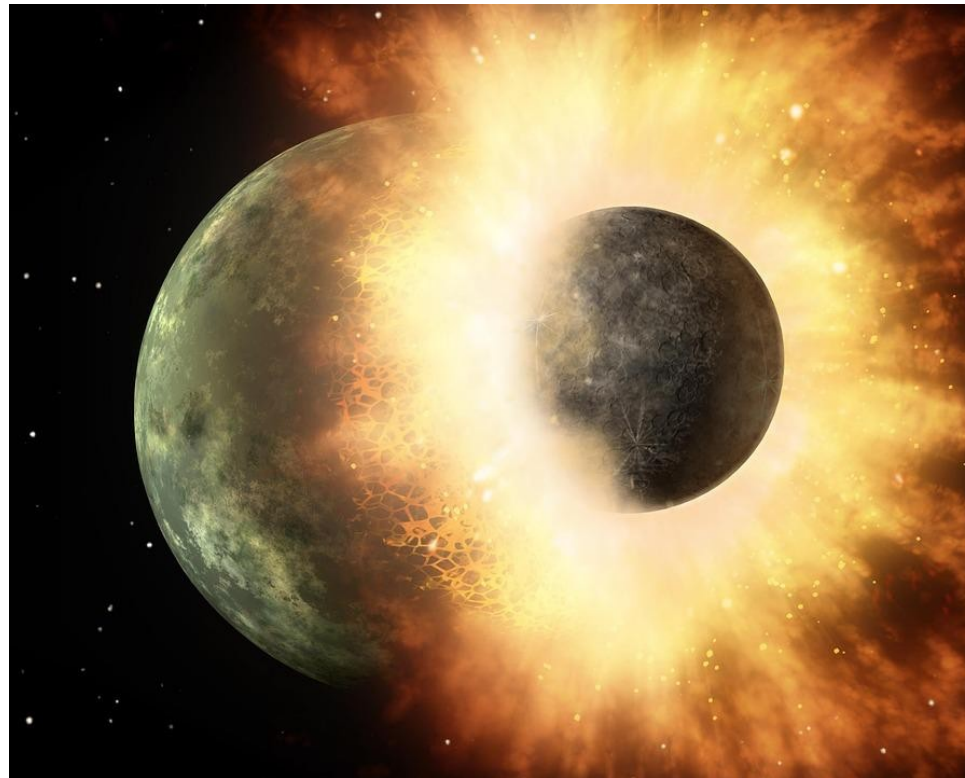
- Son pares de algoritmos:
 - Gen: $s \leftarrow S$
 - Hash: $h = H^s(m) \in \{0,1\}^L$
 - L es la longitud del hash
- Diferencia importante:
 - s no es una clave. Simplemente es un selector.
 - En muchas implementaciones $S = \{s_0\}$
 - Solo hay una función de hash en la familia
- Se las llama funciones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

Etiquetadores universales



Colisiones

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



Propiedades (definición informal)

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



Propiedades (definición informal)

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



Propiedades (definición informal)

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



Resistencia a colisiones

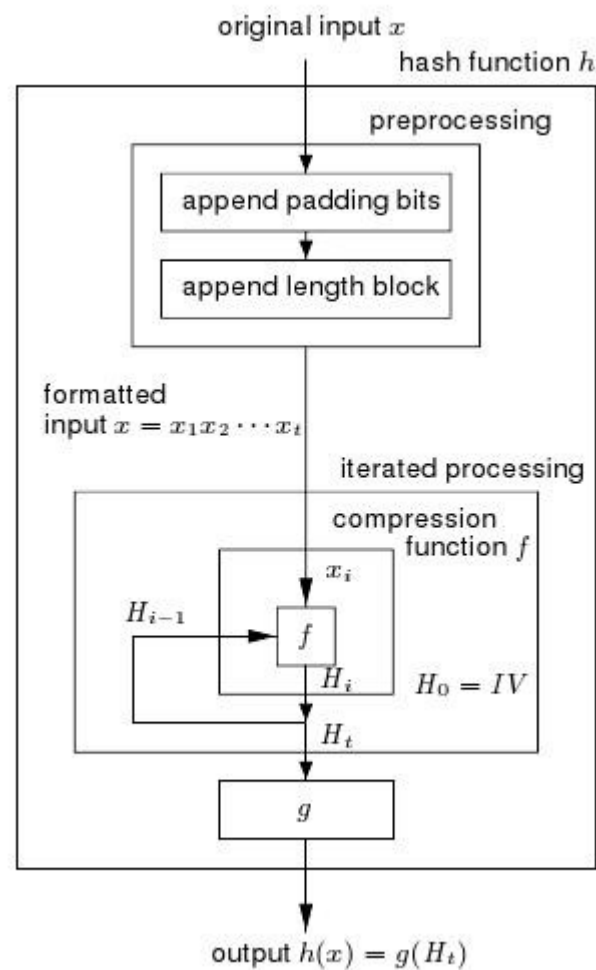
- Collision resistance: $\text{Hash-Coll}_{A,H}$
- Dado un nivel de seguridad n , un adversario A , y una familia de funciones de hash $H^x(n)$:

- 1) Se selecciona una función $s \leftarrow S$
- 2) A obtiene s , y por lo tanto $H(x) = H^s(x)$
- 3) A emite x, x'

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

Modelo general iterativo

- Propuesto por Merkle en 1989
- Utilizado por muchas funciones
- Preprocesamiento
 - Ajustar el tamaño del mensaje
 - Agregar bloque con tamaño
- Función de compresión
 - Similar al hash pero opera sobre bloques pequeños
 - La seguridad está dada por la función de compresión



MD5

- Entrada: secuencia de hasta 2^{64} bits
- Salida: secuencia de 128 bits
- Aplicación iterativa
- Ejemplos:
 - "" -> d41d8cd98f00b204e9800998ecf8427e
 - "a" -> 0cc175b9c0f1b6a831c399e269772661
 - "abc" -> 900150983cd24fb0d6963f7d28e17f72

SHA-1

- Entrada: secuencia de hasta 2^{64} bits
- Salida: secuencia de 160 bits
- 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:
 - “” -> a7ffc6f8bf1ed76651c14756a061d662
f580ff4de43b49fa82d80a4b80f8434a
 - “a” -> 80084bf2fba02475726feb2cab2d8215
eab14bc6bdd8bfb2c8151257032ecd8b
 - “abc” → 3a985da74fe225b2045c172d6bd390bd
855f086e3e9d525b46bfe24511431532

Funciones de Hash y MACs

- Es posible construir un MAC a partir de una función de hash
- HMAC:
 - Dado $H(x)$ función de hash libre de colisiones
 - $\text{HMAC}_K(x) = \{$
 - GEN: $k \leftarrow K, s \leftarrow S$
 - MAC: $t = H^s((k \oplus \text{opad}) \parallel H^s((k \oplus \text{ipad}) \parallel m))$
 - $\text{opad} = 0x36\dots36 \quad \text{ipad} = 0x5c5c\dots5c$
 - $\}$
 - Es infalsificable (MAC-Forge)

Seguridad de funciones de Hash

- Objetivos del atacante. Sea $h: A \rightarrow B$
 - Preimágenes: dado $y \in B$, hallar $x / h(x) = y$
 - *Búsqueda por fuerza bruta: requiere $|B|$ intentos*
 - Segundas imágenes: 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: requiere $|B|^{1/2}$*
 - *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
 - SHA-1 → 160bits
 - SHA-2/256 → 256 bits
 - SHA-3/256-512: Recientemente seleccionada
 - Antes conocida como Kekkak
 - Estandar recomendado para nuevos proyectos

Privacidad e integridad

- Tres formas:

- Cifrar y autenticar:

- $c \leftarrow \text{Enc}_{k1}(m), t \leftarrow \text{Mac}_{k2}(m)$

t puede brindar información de m

- Autenticar, luego cifrar

- $c \leftarrow \text{Enc}_{k1}(m \parallel \text{Mac}_{k2}(m)),$

Puede ser seguro, requiere prueba de seguridad

- Cifrar, luego Autenticar

- $c \leftarrow \text{Enc}_{k1}(m), t \leftarrow \text{Mac}_{k2}(c)$

Siempre es seguro

Cifrado Autenticado

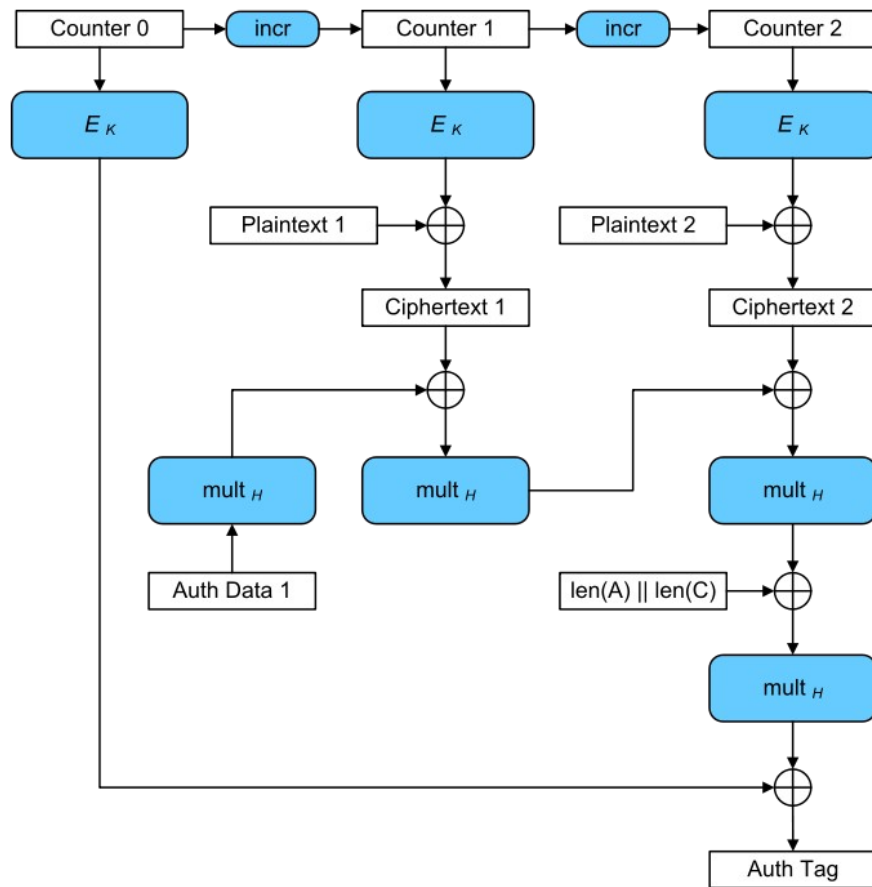
- Combina cifrado y control de integridad
- Sean $\Pi_e(\text{Gen}_e, \text{Enc}, \text{Dec})$ un criptosistema CPA-Secure y $\Pi_m(\text{Gen}_m, \text{mac}, \text{vrfy})$ un mac infalsificable.
- Se define el criptosistema:
 - Gen: $k_1 \leftarrow \text{Gen}_e, k_2 \leftarrow \text{Gen}_m$
 - Enc: $c \leftarrow \text{Enc}_{k_1}(m), t \leftarrow \text{Mac}_{k_2}(c)$
 - Dec: Si $\text{vrfy}_{k_2}(c, t) = 1 \rightarrow m = \text{Dec}_{k_1}(c)$
Sino, $m = /$ (fallo)
- El criptosistema resultante es CCA-Secure

Encadenamiento CCM

- Significa Counter with CBC-MAC
 - “Authenticate-then-encrypt”
 - $e_k(\text{cbc-mac}_k(M) || m)$
- Existe una prueba de seguridad específica
 - 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

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



Cifrado modo counter

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

$H = E_k(0000\dots0000)$

$\text{Mult}_h(x) = \text{Mult}(x, h)$

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

Aplicaciones prácticas

- En términos generales, solo utilizar cifrado autenticado 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 esto.

Lectura Recomendada

Capítulo 4

Introduction to Modern Cryptography
Katz & Lindell