Pontificia Universidad Católica de Chile

Departamento de Ciencia de la Computación

Apunte IIC3253

Criptografía y Seguridad Computacional

Autor
Cristóbal Rojas

En base a apuntes de
Prof. Marcelo Arenas
Prof. Martín Ugarte

En base al texto
Introduction to Modern Cryptography
J. Katz, Y. Lindell

3 de junio de 2023

Índice

1. Introducción

1.1. Dos problemas fundamentales: cifrado y autentificación

La criptografía moderna implica *el estudio de técnicas matemáticas para proteger la información digital, los sistemas y los cálculos distribuidos contra ataques de adversarios.*

La seguridad de los **esquemas criptográficos modernos** (hablaremos de ellos más adelante) recae en una **llave secreta** compartida con anterioridad entre el emisor (A, de *Alice*) y receptor del mensaje (B, de *Bob*), que debe ser desconocida para algún adversario (E, de *Eavesdropper*). Este escenario, en donde las partes que se comunican comparten alguna información secreta por adelantado, es conocida como **encriptación de llave privada** (*private-key encryption*).

En este contexto, Alice y Bob comparten una llave privada cuando se quieren comunicar en secreto. Uno de ellos, por ejemplo, Alice, puede mandar un **mensaje** (*plain text*, texto plano) hacia el otro usando la llave para **encriptar** el mensaje y así obtener un **texto cifrado** o *cyphertext* que es trasmitido al receptor. Bob, el receptor, usa la misma llave para **decriptar** el texto cifrado y recuperar el mensaje original.

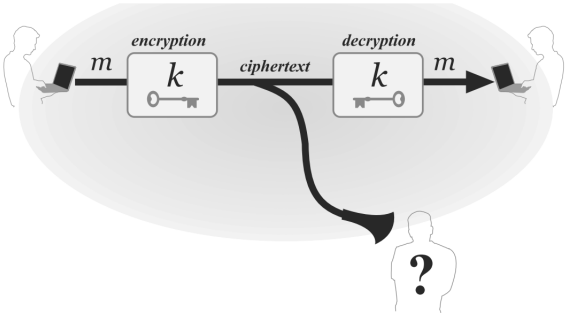


Figura 1: Alice manda un mensaje m a Bob usando una llave privada k que compartieron anteriormente

La **autentificación**, en cambio, es el proceso en el que nos interesa garantizar la **integridad del mensaje** contra un adversario que podría modificar o mandar otros mensajes por el canal de comunicación.

Por ejemplo, Alice y Bob, que desean comunicarse de forma autenticada, empiezan por generar y compartir una llave secreta k antes de su comunicación. Cuando una de las partes desea enviar un mensaje m a la otra, calcula una etiqueta (*tag*) t basada en el mensaje y la llave compartida, y envía el mensaje m junto con t a la otra parte. El *tag* se calcula utilizando un *algoritmo de generación de tags* denotado por `Mac`; así, reformulando lo que acabamos de decir, el emisor de un mensaje m calcula $t \leftarrow \texttt{Mac}_k(m)$ y transmite (m, t) al receptor. Al recibir (m, t), la segunda parte verifica si t es un *tag* válido en el mensaje m (con respecto a la llave compartida) o no. Esto se hace ejecutando un *algoritmo de verificación* `Vrfy` que toma como entrada la llave compartida, así como un mensaje m y un *tag* t, e indica si el *tag* dado es válido. Profundizaremos más sobre este tema en el capítulo X.

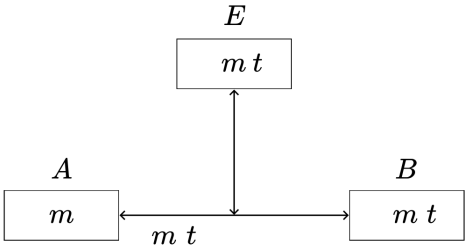


Figura 2: Alice manda (m, t) a Bob, generando t con `Mac` y Bob debe verificar t con `Vrfy`

1.2. Sintaxis de encriptación

Formalmente, un esquema de encriptación de llave privada es definido por un **espacio de mensajes** $\mathcal{M}$, un **espacio de llaves** $\mathcal{K}$ y un **espacio de textos cifrados** $\mathcal{C}$, junto con tres funciones: una para generar llaves (`Gen`), una para encriptar (`Enc`) y otra para decriptar (`Dec`). El espacio de mensajes $\mathcal{M}$ define el conjunto "legal" de mensajes, es decir, los soportados por el esquema. Las funciones del esquema tienen la siguiente funcionalidad:

- `Gen` es un algoritmo de probabilidades que retorna una llave k elegida acorde a una distribución. Es decir, $\texttt{Gen} : \mathcal{K} \to [0, 1]$ tal que
$$\sum_{k \in \mathcal{K}} \texttt{Gen}(k) = 1$$
- `Enc` es una familia de algoritmos de encriptación que toma como input la llave k y el mensaje m, y retorna un texto cifrado c. Denotamos $\texttt{Enc}_k(m)$ la encriptación del texto plano m usando la llave k.
$$\texttt{Enc} = \{\texttt{Enc}_k \mid k \in \mathcal{K}\}, \quad \text{con } \texttt{Enc}_k : \mathcal{M} \to \mathcal{C},\ \forall k \in \mathcal{K}$$
- `Dec` es una familia de algoritmos de decriptación que toma como input la llave k y el texto cifrado c, y retorna un texto plano m. Denotamos $\texttt{Dec}_k(c)$ la decriptación del texto cifrado c usando la llave k.
$$\texttt{Dec} = \{\texttt{Dec}_k \mid k \in \mathcal{K}\}, \quad \text{con } \texttt{Dec}_k : \mathcal{C} \to \mathcal{M},\ \forall k \in \mathcal{K}$$

Un esquema de encriptación debe satisfacer el siguiente requisito de correctitud: para cada llave k generada por `Gen` y para cada mensaje $m \in \mathcal{M}$, se tiene que

$$\boxed{\texttt{Dec}_k(\texttt{Enc}_k(m)) = m}$$

En otras palabras, encriptar un mensaje y luego decriptar el texto cifrado resultante usando la misma llave devuelve el mensaje original.

1.3. Principio de Kerckhoffs

Si un adversario conoce el algoritmo `Dec` y la llave k compartida por las dos partes comunicantes, podrá descifrar cualquier texto cifrado transmitido por dichas partes. Por este motivo, las partes comunicantes deben compartir la llave k de forma segura y mantener k completamente oculta para los demás. ¿Quizás también deberían mantener en secreto el algoritmo de descifrado `Dec`? ¿No sería mejor que mantuvieran en secreto todos los detalles del esquema de cifrado?

Lo anterior condujo a Auguste Kerckhoffs en 1983 a declarar el siguiente principio:

*"La seguridad de un sistema criptográfico **no** debe depender de que los algoritmos de cifrado y descifrado sean secretos, solo debe depender de que las claves sean secretas".*

Es decir, un esquema de encriptación debe diseñarse para que sea seguro *incluso* si un espía conoce todos los detalles del esquema, siempre que el atacante no conozca la clave utilizada. Dicho de otro modo, la seguridad no debe depender de que el esquema de cifrado sea secreto; en cambio, el principio de Kerckhoffs exige que la seguridad dependa únicamente del secreto de la clave.

Seguimos este principio por tres simples motivos:

- Es más fácil mantener la privacidad de una llave que la de un algoritmo.
- Si la seguridad se ve comprometida es más fácil cambiar una llave que un algoritmo.
- Es mejor usar algoritmos públicos que hayan sido ampliamente verificados.

1.4. Principios de la criptografía moderna

Podemos mencionar tres grandes principios para la criptografía moderna:

1. **Definiciones formales:** Es importante definir formalmente los sistemas criptográficos y nociones de seguridad usados.

   "Si no sabes lo que quieres conseguir, ¿cómo vas a saber si lo has conseguido?"

   Las definiciones formales facilitan esa comprensión al ofrecer una descripción clara de las amenazas que entran en juego y de las garantías de seguridad deseadas. Como tales, las definiciones pueden ayudar a guiar el diseño de esquemas criptográficos. De hecho, es mucho mejor formalizar lo que se necesita *antes* de que comience el proceso de diseño, en lugar de llegar a una definición *post facto* —posterior a los hechos— una vez que el diseño se ha completado.

2. **Supuestos precisos:** Es importante que los supuestos detrás del funcionamiento de un sistema criptográfico tengan una **formulación precisa** y sean **conocidos**.

   La mayoría de los esquemas criptográficos modernos no pueden demostrarse seguros de forma incondicional; para ello habría que resolver cuestiones de la teoría de la complejidad computacional que hoy en día parecen lejos de tener respuesta. El resultado de esta desafortunada situación es que las pruebas de seguridad suelen basarse en suposiciones. La criptografía moderna exige que estas suposiciones sean explícitas y precisas desde el punto de vista matemático. En el nivel más básico, esto se debe a que las pruebas de seguridad así lo exigen.

3. **Pruebas de seguridad:** Es importante construir **demostraciones formales de seguridad** (basadas en las definiciones y supuestos).

   Los dos principios anteriores que acabamos de describir nos permiten alcanzar nuestro objetivo de proporcionar pruebas rigurosas de que una construcción satisface una definición dada bajo ciertos supuestos. Estas pruebas son especialmente importantes en el contexto de la criptografía, donde hay un atacante que intenta activamente "romper" algún esquema. Las pruebas de seguridad ofrecen una garantía irrefutable —relativa a la definición y los supuestos— de que ningún atacante tendrá éxito; esto es mucho mejor que adoptar un enfoque sin principios o heurístico del problema. Sin una prueba de que ningún adversario con los recursos especificados puede romper un esquema, sólo nos queda nuestra intuición de que es así. La experiencia ha demostrado que la intuición en criptografía y seguridad informática es desastrosa. Hay innumerables ejemplos de esquemas no probados que se rompieron, a veces inmediatamente y otras años después de ser desarrollados.

1.5. Seguridad, noción de adversario y tipos de ataque

En general, una definición de seguridad tiene dos componentes: una **garantía de seguridad** (o, desde el punto de vista del adversario, lo que constituye un ataque exitoso) y un **modelo de amenaza** (tipos de ataque). La garantía de seguridad define lo que se pretende que el esquema impida al atacante hacer, mientras que el modelo de amenaza describe el poder del adversario, es decir, qué acciones se supone que el atacante puede llevar a cabo.

Los siguientes puntos engloban lo que un esquema de encriptación debería **garantizar**:

- *Debe ser imposible para un adversario recuperar la llave:* aunque la seguridad criptográfica depende de que el adversario no pueda encontrar la llave, este requisito por sí solo no es suficiente para garantizar la seguridad del esquema.
- *Debe ser imposible para un adversario recuperar el mensaje (texto plano) a partir del texto cifrado:* este requisito es fundamental para la seguridad del cifrado. Si un adversario puede recuperar el mensaje original a partir del texto cifrado, entonces el esquema de cifrado no es seguro.

- *Debe ser imposible para un adversario recuperar cualquier carácter del mensaje a partir del texto cifrado:* este requisito es aún más riguroso, ya que no sólo busca proteger el mensaje completo, sino también la información contenida en él.
- *La "respuesta correcta": independientemente de la información que tenga el adversario, el texto cifrado no debe filtrar información adicional sobre el mensaje.* Esta definición amplia garantiza que el esquema de cifrado protege la información sin importar cuánta información tenga el adversario. El objetivo del cifrado es evitar que el adversario obtenga cualquier información adicional sobre el mensaje, lo que se logra si el texto cifrado no contiene información adicional más allá de lo que ya se sabe.

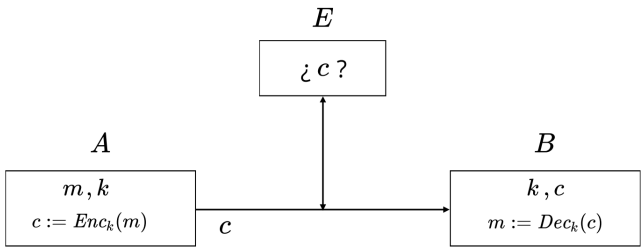
El **modelo de amenaza** (los tipos de ataque) especifica qué "poder" se supone que tiene el atacante, pero no impone ninguna restricción en la *estrategia* del adversario. Esta es una distinción importante: especificamos lo que asumimos sobre las habilidades del adversario, pero **no** asumimos nada acerca de cómo usa esas habilidades. Es imposible prever qué estrategias podrían ser utilizadas en un ataque, y la historia ha demostrado que los intentos de hacerlo están destinados al fracaso. Algunos de los tipos de ataque son:

- **Ciphertext-only attack:** es el ataque más básico, en el que el adversario solo observa el texto cifrado y trata de determinar información sobre el texto plano subyacente. Este es el modelo de amenaza que hemos estado asumiendo implícitamente al discutir el esquema de encriptación definido anteriormente.
- **Known-plaintext attack:** aquí, el adversario puede aprender uno o más pares de texto plano/cifrado generados utilizando alguna clave. El objetivo del adversario es deducir información sobre el texto plano subyacente de otro texto cifrado producido utilizando la misma clave.
- **Chosen-plaintext attack:** en este ataque, el adversario puede obtener pares de texto plano/cifrado para textos planos de su elección.
- **Chosen-ciphertext attack:** el último tipo de ataque es aquel en el que el adversario es capaz de obtener (alguna información sobre) el descifrado de textos cifrados de su elección. El objetivo del adversario, una vez más, es aprender información sobre el texto plano subyacente de algún otro texto cifrado (cuyo descifrado el adversario no puede obtener directamente) generado utilizando la misma clave.

Aunque los modelos de amenaza están listados en orden de creciente fuerza, ninguno es inherentemente mejor que otro; el adecuado a utilizar depende del entorno en el que se despliega un esquema de encriptación.

2. Criptografía simétrica o de clave privada

Hasta ahora, un ejemplo de criptografía simétrica se ve así:



Alice desea mandar un mensaje m a Bob, por lo que usa $\texttt{Enc}_k$ para encriptar el mensaje con la llave k que compartieron anteriomente, enviando así el texto cifrado c por un canal de comunicación. Bob, por su lado, puede desencriptar c usando $\texttt{Dec}_k$ con la llave k y así obtener el mensaje m. Frente a lo anterior, puede existir un adversario E que quiera conocer la información del mensaje m capturando el texto cifrado c.

2.1. Cifrado del César

Julio César cifró desplazando (*shift*) las letras del alfabeto 3 lugares hacia delante: `A` se sustituyó por `D`, `B` por `E`, y así sucesivamente. Al final del alfabeto, las letras se envuelven alrededor y así `Z` fue reemplazado con `C`, y con `B`, y `X` con `A`. Por ejemplo, el cifrado del mensaje `BEGIN THE ATTACK NOW`, con espacios eliminados, da:

`EHJLQWKHDWWDFNQRZ`

Un problema inmediato de este cifrado es que el método de encriptación es fijo; **no hay llave**. Así, **cualquiera que supiera cómo encriptó César sus mensajes podría desencriptarlos sin esfuerzo**.

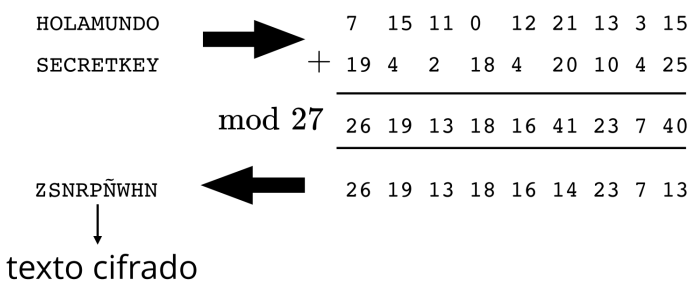
A pesar de que es poco seguro, este cifrado nos servirá como base para entender OTP, un esquema de encriptación *perfectamente secreto* (ya veremos que significa esto).

2.2. Una primera versión de One-Time Pad (OTP)

Veamos una construcción de OTP basada en el cifrado del César. Partimos enumerando las 27 letras del abecedario:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

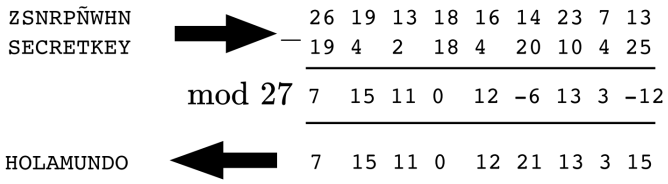
Para enviar un mensaje de largo ℓ, necesitaremos una llave de largo ℓ. Por ejemplo, probemos mandar el mensaje `HOLAMUNDO` con la llave `SECRETKEY`:



En el ejemplo anterior, a cada letra del mensaje y de la llave, se convierten a un número según la tabla del abecedario, luego se suman y les aplica la operación módulo 27 (ya que son 27 letras). Así, se obtiene un número que representa el cifrado de la letra correspondiente. Es decir:

$$\texttt{Enc}_{\texttt{SECRETKEY}}(\texttt{HOLAMUNDO}) = \texttt{ZSNRPÑWHN}$$

Para desencriptar, hacemos el mismo proceso, solo que en vez de sumar, restamos.



Y tenemos entonces que

$$\texttt{Dec}_{\texttt{SECRETKEY}}(\texttt{ZSNRPÑWHN}) = \texttt{HOLAMUNDO}$$

Para **formalizar** este algoritmo, necesitamos convertir mensajes, llaves y textos cifrados en arreglos de enteros:

$$m = \texttt{HOLAMUNDO} \quad \bar{m} = (7, 15, 11, 0, 12, 21, 13, 3, 15)$$
$$k = \texttt{SECRETKEY} \quad \bar{k} = (19, 4, 2, 18, 4, 20, 10, 4, 25)$$
$$c = \texttt{ZSNRPNWHN} \quad \bar{c} = (26, 19, 13, 18, 16, 14, 23, 7, 13)$$

De la misma forma necesitamos hacer la conversión en la otra dirección:

$$a = (4, 9, 4, 12, 16, 11, 15) \quad \bar{a} = \texttt{EJEMPLO}$$

Naturalmente, vemos que siempre se cumple que $\bar{\bar{s}} = s$. Con esto, definimos OTP en base a

$$\boxed{\begin{aligned}\texttt{Enc}_k(m) &= \overline{(\bar{m} + \bar{k}) \bmod 27} \\ \texttt{Dec}_k(m) &= \overline{(\bar{c} - \bar{k}) \bmod 27}\end{aligned}}$$

Desde ahora supondremos que nuestros mensajes y llaves **son arreglos de números**. Con esta suposición en mente, definiremos OTP simplemente usando:

$$\boxed{\begin{aligned}\texttt{Enc}_k(m) &= (m + k) \bmod 27 \\ \texttt{Dec}_k(m) &= (c - k) \bmod 27\end{aligned}}$$

Con esta definición, vemos entonces que

$$\texttt{Dec}_k(\texttt{Enc}_k(m)) = ((m + k) \bmod 27 - k) \bmod 27 = (m + k - k) \bmod 27 = m \bmod 27 = m$$

Generalizando, podemos describir $\text{OTP}^{N,\ell}$ sobre $\{0, 1, \ldots, N-1\}$ y mensajes de largo ℓ, con:

$$\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0, 1, \ldots, N-1\}$$

$\texttt{Gen}$ es la **distribución de probabilidad uniforme** sobre $\{0, 1, \ldots, N-1\}$, y

$$\texttt{Enc}_k(m) = (m + k) \bmod N$$
$$\texttt{Dec}_k(c) = (c - k) \bmod N$$

Esperamos que para un esquema criptográfico $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ se cumpla que

$$\boxed{\forall k \in \mathcal{K}.\ \forall m \in \mathcal{M}: \quad \texttt{Dec}_k\left(\texttt{Enc}_k(m)\right) = m}$$

En este caso diremos que el esquema es *perfectamente correcto*.

2.3. Perfect secrecy

Recordemos que un esquema criptográfico está compuesto por una tupla $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$, con un espacio de mensajes $\mathcal{M}$, un espacio de llaves $\mathcal{K}$ y un espacio de textos cifrados $\mathcal{C}$. ¿Cuándo podemos decir que un esquema así es *perfectamente secreto*? Podríamos decir algo como lo siguiente: *"Al ver un texto cifrado c_0 pasar, para el atacante el mensaje original m podría haber sido cualquiera"*.

¿Cómo formalizamos lo anterior? Dado un texto cifrado c_0, se cumple que:

$$\forall m_0 \in \mathcal{M}. \quad \Pr_{k\sim\texttt{Gen}}[\texttt{Enc}_k(m_0) = c_0] = \frac{1}{|\mathcal{M}|}$$

La probabilidad anterior se calcula viendo la probabilidad de haber elegido una llave que encripte m_0 como c_0:

$$\sum_{k\in\mathcal{K}:\ \texttt{Enc}_k(m_0)=c_0} \texttt{Gen}(k)$$

Ahora, ¿qué pasa si el atacante tiene información sobre el mensaje? En otras palabras, el atacante posee una distribución de probabilidad $\mathbb{D}$ sobre $\mathcal{M}$ (conoce la probabilidad de un mensaje). Para cada distribución de probabilidad $\mathbb{D}$ sobre $\mathcal{M}$ y cada texto cifrado $c_0 \in \mathcal{C}$ se cumple que:

$$\forall m_0 \in \mathcal{M}: \quad \Pr_{\substack{m\sim\mathbb{D}\\ k\sim\texttt{Gen}}}[m = m_0 \mid \texttt{Enc}_k(m) = c_0] = \Pr_{m\sim\mathbb{D}}[m = m_0]$$

Recordemos que $\Pr(A \mid B) = \frac{\Pr(A\cap B)}{\Pr(B)}$. Luego, ¿se cumple la siguiente ecuación?

$$\frac{\mathbb{D}(m_0)\sum_{k\in\mathcal{K}:\texttt{Enc}_k(m_0)=c_0}\texttt{Gen}(k)}{\sum_{m\in\mathcal{M}}\mathbb{D}(m)\sum_{k\in\mathcal{K}:\texttt{Enc}_k(m)=c_0}\texttt{Gen}(k)} \stackrel{?}{=} \mathbb{D}(m_0)$$

Para responder la pregunta anterior, volvamos a $\text{OTP}^{N,\ell}$ y analicemos si es *perfectamente secreto*.

1. $\texttt{Gen}$ es la distribución uniforme $1/N^\ell$.
2. Para cada c_0 y cada m_0 existe una única llave k tal que $\texttt{Enc}_k(m_0) = c_0$

Sea $c_0 \in \mathcal{C}$ un texto cifrado y m_0 un mensaje, entonces:

$$\frac{\mathbb{D}(m_0)\sum_{k\in\mathcal{K}:\texttt{Enc}_k(m_0)=c_0}\texttt{Gen}(k)}{\sum_{m\in\mathcal{M}}\mathbb{D}(m)\sum_{k\in\mathcal{K}:\texttt{Enc}_k(m)=c_0}\texttt{Gen}(k)} = \frac{\mathbb{D}(m_0)\ \cdot\ \cancel{1/N^\ell}}{\cancel{\sum_{m\in\mathcal{M}}\mathbb{D}(m)}^{\,1}\ \cdot\ \cancel{1/N^\ell}} = \mathbb{D}(m_0)$$

Por lo tanto, **OTP es un esquema perfectamente secreto** y luego, la ecuación anterior sí se cumple.

Como hemos podido ver, *perfect secrecy* es una condición muy fuerte. Lamentablemente, pareciera ser molesto y/o poco razonable que la llave tenga que ser tan larga como el mensaje. ¿Hay alguna forma de modificar OTP para tener $|\mathcal{K}| < |\mathcal{M}|$ y seguir teniendo un esquema criptográfico *perfectamente secreto*? Spoiler, no hay tal forma :(

Teorema 1

Sean $\mathcal{M}$, $\mathcal{K}$, $\mathcal{C}$ espacios de mensajes, llaves y textos cifrados, respectivamente. Si $|\mathcal{K}| < |\mathcal{M}|$, entonces no existe un esquema $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ que sea perfectamente secreto.

Dado su nombre, podemos decir que OTP solo es seguro solo si **no** se reutilizan las llaves cada vez que se encripta un mensaje. Si se llega a utilizar la misma llave con la que se cifró el mensaje, OTP **no es seguro bajo ningún ataque**.

Demostración teorema. Supongamos que $|\mathcal{K}| < |\mathcal{M}| \leq \mathcal{C}$ y sea $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ un esquema criptográfico. Sea $\mathbb{D}$ una distribución sobre $\mathcal{M}$ y $m_0 \in \mathcal{M}$ un mensaje tal que $\mathbb{D}(m_0) > 0$. Como $|\mathcal{K}| < |\mathcal{M}| \leq \mathcal{C}$, debe existir $c_0 \in \mathcal{C}$ para el cual **ninguna** llave $k \in \mathcal{K}$ satisface $\texttt{Enc}_k(m_0) = c_0$, por tanto

$$\Pr_{\substack{m\sim\mathbb{D}\\ k\sim\texttt{Gen}}}[m = m_0 \mid \texttt{Enc}_k(m_0) = c_0] \quad < \quad \mathbb{D}(m_0) = \Pr_{m\sim\mathbb{D}}[m = m_0]$$

(la probabilidad condicional es 0)

■

Con todo lo anterior, podemos formalizar la definición de *perfect secrecy* para cualquier esquema criptográfico desde otra perspectiva.

Definición formal de perfect secrecy. Un esquema criptográfico $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ con un espacio de mensajes $\mathcal{M}$ es **perfectamente secreto** si para cada distribución de probabilidad para M, para cada mensaje $m_0 \in \mathcal{M}$ y para cada texto cifrado $c \in \mathcal{C}$ para el cual $\Pr[C = c_0] > 0$:

$$\Pr[M = m \mid C = c] = Pr[M = m]$$

con M y C las variables aleatorias para el mensaje y el texto cifrado.

Lema. Un esquema de encriptación $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ con un espacio de mensajes $\mathcal{M}$ es perfectamente secreto si, y sólo si se cumple la siguiente ecuación para todo $m, m' \in \mathcal{M}$ y para todo $c \in \mathcal{C}$:

$$\Pr[\texttt{Enc}_k(m) = c] = \Pr[\texttt{Enc}_k(m') = c]$$

En otras palabras, la distribución de probabilidad del texto cifrado cuando ciframos m debe ser idéntica a la distribución de probabilidad del texto cifrado cuando se cifra m'.

2.4. Ahora sí, One-Time Pad

Primero, recordemos que $a \oplus b$ denota el OR-exclusivo (XOR) de dos *strings* binarios de igual longitud a y b (es decir, si $a = a_1 \cdots a_n$ y $b = b_1 \cdots b_n$ son *strings* de n bits, entonces $a \oplus b$ es un *string* de n bits dado por $(a_1 \oplus b_1) \cdots (a_n \oplus b_n)$). En OTP, **la llave** es un *string* uniforme de la **misma longitud** que el mensaje, y el texto cifrado se calcula simplemente haciendo XORing entre la llave y el mensaje.

Antes de hablar de la seguridad de OTP, verifiquemos su **corrección**: Para cada clave k y cada mensaje m se cumple que $\texttt{Dec}_k(\texttt{Enc}_k(m)) = \texttt{Dec}_k(k \oplus m) = k \oplus k \oplus m = m$, por lo que OTP constituye un esquema de cifrado válido.

Construcción del esquema. Fijemos un entero $n > 0$. El espacio de mensajes $\mathcal{M}$, el espacio de llaves $\mathcal{K}$ y el espacio de textos cifrados $\mathcal{C}$ son todos equivalentes a $\{0,1\}^n$ (el conjunto de todos los *strings* binarios de largo n).

1. `Gen`: el algoritmo de generación de llaves elige una desde $\mathcal{K} = \{0,1\}^n$ de acuerdo a la distribución de probabilidad **uniforme** (es decir, cada uno de los 2^n *strings* en el espacio es elegido como llave con probabilidad $1/2^n$).
2. `Enc`: dada una llave $k \in \{0,1\}^n$ y un mensaje $m \in \{0,1\}^n$, el algoritmo de encriptación retorna el texto cifrado $c = k \oplus m$.
3. `Dec`: dada una llave $k \in \{0,1\}^n$ y un texto cifrado $c \in \{0,1\}^n$, el algoritmo de decriptación retorna un mensaje $m = k \oplus c$.

Demostración de la construcción. Calculamos $\Pr[C = c \mid M = m]$ para un $c \in \mathcal{C}$ arbitrario y un $m \in \mathcal{M}$ con $\Pr[M = m] > 0$. Por OTP, tenemos que

$$\Pr[C = c \mid M = m] = \Pr[K \oplus m = c \mid M = m] = \Pr[K = m \oplus c \mid M = m] = \frac{1}{2^n}$$

donde M, K y C son las variables aleatorias para el mensaje, la llave y el texto cifrado, que son **independientes** entre sí. Aquí, la primera ecuación es por definición del esquema y el hecho que condicionamos el evento de que $M = m$, y la última ecuación es válida porque la llave K es un n-bit uniforme que es independiente de M. Fijemos cualquier distribución de probabilidad sobre $\mathcal{M}$. Usando el resultado de arriba, vemos que para cualquier $c \in \mathcal{C}$ tenemos:

$$\Pr[C = c] = \sum_{m \in \mathcal{M}} \Pr[C = c \mid M = m] \cdot \Pr[M = m] = \frac{1}{2^n} \cdot \sum_{m \in \mathcal{M}} \Pr[M = m] = \frac{1}{2^n}$$

donde la suma es sobre $m \in \mathcal{M}$ con $\Pr[M = m] \neq 0$. Por el Teorema de Bayes:

$$\Pr[M = m \mid C = c] = \frac{\Pr[C = c \mid M = m] \cdot \Pr[M = m]}{\Pr[C = c]} = \frac{2^{-n} \cdot \Pr[M = m]}{2^{-n}} = \Pr[M = m]$$

Concluimos entonces que OTP es *perfectamente secreto.* ■

2.5. Permutaciones pseudo-aleatorias (PRP)

Recordemos que una noción de seguridad debe incluir:

- Un modelo de amenaza (tipos de ataque), que define las capacidades de un **adversario**.
- Una garantía de seguridad, lo cual normalmente se traduce en definir qué significa que el adversario no tenga éxito en su ataque.

Vamos a definir una primera noción de seguridad que nos va a permitir mostrar que OTP no es seguro si la llave es **reutilizada**, como mencionamos anteriormente.

Consideremos un largo n fijo tal que los espacios de mensajes, llaves y textos cifrados cumplen que $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$ y $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ es un esquema criptográfico. Veamos un **juego** para definir una **pseudo-random permutation** (PRP):

1. Un **verificador** elige $b \in \{0,1\}$ con distribución uniforme (por ejemplo, tirar una moneda).
   1.1. Si $b = 0$, entonces elige una clave $k \in \mathcal{K}$ según la distribución $\texttt{Gen}$ y define $f(x) = \texttt{Enc}_k(x)$.
   1.2. Si $b = 1$, entonces elige una permutación π con distribución uniforme y define $f(x) = \pi(x)$
2. El **adversario** elige una palabra $y \in \{0,1\}^n$ y el verificador responde con $f(y)$.
3. El paso 2. es repetido q veces.
4. El adversario indica si $b = 0$ o $b = 1$, y gana si su elección es correcta.

La probabilidad de que el adversario gane depende de la cantidad de rondas q. Si el adversario "tira una moneda" en el paso 4., entonces su probabilidad de ganar es $\frac{1}{2}$.

Definición de PRP. Decimos que $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ es una permutación pseudo-aleatoria si **no existe** un adversario que pueda ganar el juego con una probabilidad significativamente mayor a $\frac{1}{2}$.

No imponemos restricciones en las capacidades computacionales del adversario, podríamos tener una noción más débil donde el adversario puede realizar una cantidad de operaciones que es polinomial en n. Por ejemplo, solo puede realizar n^2 operaciones. Debemos entonces definir qué significa que *la probabilidad de ganar el juego sea significativamente mayor a* $\frac{1}{2}$ (podría ser $\frac{3}{4}$, por ejemplo). También, tenemos que indicar cuál es el número de rondas q.

Un primer ejemplo. Considere un esquema $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ tal que:

- $\texttt{Gen}(k_0) = 1$ para una llave fija $k_0 \in \mathcal{K}$.
- $\texttt{Gen}(k) = 0$ para todo $k \in \mathcal{K}$ tal que $k \neq k_0$.

Vamos a demostrar que este esquema criptográfico no es una PRP con una ronda ($q = 1$). La estrategia del adversario es la siguiente:

- En el paso 2. toma $y = 0^n$ y recibe $f(y)$ como respuesta del verificador.
- Si $f(y) = \texttt{Enc}_{k_0}(y)$ entonces indica que $b = 0$, si no, indica que $b = 1$.

¿Puede el adversario equivocarse al indicar el valor de b? ¿Cuál es la probabilidad de que gane el juego? Veamos. Definamos A como el evento "Adversario gana el juego", entonces:

$$\begin{aligned}\Pr[A] &= \Pr[A \mid b = 0] \cdot \Pr[b = 0] + \Pr[A \mid b = 1] \cdot \Pr[b = 1] \\ &= \Pr[A \mid b = 0] \cdot \frac{1}{2} + \Pr[A \mid b = 1] \cdot \frac{1}{2}\end{aligned}$$

Vemos que:

- $\Pr[A \mid b = 0] = 1$
- $\Pr[A \mid b = 1] = \Pr[\pi(y) \neq \texttt{Enc}_{k_0}(y)]$

  En este caso el verificador elige una permutación $\pi : \{0,1\}^n \to \{0,1\}^n$ con **distribución uniforme**. Si $\pi(y) = \texttt{Enc}_{k_0}(y)$, entonces el adversario da la respuesta equivocada. Luego,

$$\begin{aligned}\Pr[\pi(y) \neq \texttt{Enc}_{k_0}(y)] &= 1 - \Pr[\pi(y) = \texttt{Enc}_{k_0}(y)] \\ &= 1 - \frac{\text{casos favorables}}{\text{casos totales}} \\ &= 1 - \frac{(2^n - 1)!}{(2^n)!} \\ &= 1 - \frac{1}{2^n}\end{aligned}$$

De la ecuación anterior:

- **Casos totales:** número de permutaciones $\pi : \{0,1\}^n \to \{0,1\}^n$.
- **Casos favorables:** número de permutaciones $\pi : \{0,1\}^n \to \{0,1\}^n$ tales que $\pi(0^n)$ es igual al valor fijo $\texttt{Enc}_{k_0}(0^n)$.

Finalmente, tenemos que

$$\begin{aligned}\Pr[A] &= \Pr[A \mid b = 0] \cdot \frac{1}{2} + \Pr[A \mid b = 1] \cdot \frac{1}{2} \\ &= 1 \cdot \frac{1}{2} + \left(1 - \frac{1}{2^n}\right) \cdot \frac{1}{2} \\ &= 1 - \frac{1}{2^{n+1}}\end{aligned}$$

Por ende, el adversario **sí** gana el juego con probabilidad significativamente mayor a $\frac{1}{2}$. Puede verificar este hecho tomando $n = 10$ o $n = 100$.

Un segundo ejemplo. Dado que $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$, debemos realizar las operaciones de OTP en módulo 2:

$$\begin{array}{rll} & 0\,1\,1\,0\,1\,0\,0\,0\,1\,1 & \longleftarrow\ m \\ + & 1\,0\,0\,1\,1\,0\,1\,1\,0\,1 & \longleftarrow\ k \\ \hline \mod 2 & 1\,1\,1\,1\,2\,0\,1\,1\,1\,2 & \\ \hline & 1\,1\,1\,1\,0\,0\,1\,1\,1\,0 & \longleftarrow\ c \end{array}$$

Veremos que OTP no es una PRP con dos rondas. La estrategia del adversario es la siguiente:

- En el paso 2. toma $y_1 = 0^n$ e $y_2 = 1^n$, y recibe $f(y_1)$ y $f(y_2)$ como respuesta del verificador.
- Si $y_2 + f(y_1) = f(y_2)$ entonces indica que $b = 0$, sino indica que $b = 1$.

Si $b = 0$, el verificador está usando OTP y existe una clave k tal que:

$$\begin{aligned} y_1 + k &= f(y_1) \\ y_2 + k &= f(y_2) \end{aligned}$$

Como $y_1 = 0^n$, se tiene que $k = f(y_1)$ (sumamos 0 en binario), y se concluye que $y_2 + f(y_1) = f(y_2)$. Luego, consideremos nuevamente el evento A como "Adversario gana el juego", entonces, la probabilidad de ganar el juego con dos rondas es la siguiente:

$$\begin{aligned} \Pr[A] &= \Pr[A \mid b = 0] \cdot \Pr[b = 0] + \Pr[A \mid b = 1] \cdot \Pr[b = 1] \\ &= \Pr[A \mid b = 0] \cdot \frac{1}{2} + \Pr[A \mid b = 1] \cdot \frac{1}{2} \end{aligned}$$

Vemos que:

- $\Pr[A \mid b = 0] = 1$
- $\Pr[A \mid b = 1] = \Pr[\pi(y_2) \neq y_2 + \pi(y_1)]$

  En este caso, el verificador elige una permutación $\pi : \{0,1\}^n \to \{0,1\}^n$ con **distribución uniforme**. Si $\pi(y_2) = y_2 + \pi(y_1)$, entonces el adversario da la respuesta equivocada. Así, vemos que:

$$\begin{aligned} \Pr[\pi(y_2) \neq y_2 + \pi(y_1)] &= 1 - \Pr[\pi(y_2) = y_2 + \pi(y_1)] \\ &= 1 - \frac{\text{casos favorables}}{\text{casos totales}} \\ &= 1 - \frac{2^n \cdot (2^n - 2)!}{(2^n)!} \\ &= 1 - \frac{1}{2^n - 1} \end{aligned}$$

De la ecuación anterior:

- **Casos totales:** número de permutaciones $\pi : \{0,1\}^n \to \{0,1\}^n$.
- **Casos favorables:** número de permutaciones $\pi : \{0,1\}^n \to \{0,1\}^n$ tales que $\pi(1^n) = 1^n + \pi(0^n)$.

Finalmente, tenemos que

$$\begin{aligned} \Pr[A] &= \Pr[A \mid b = 0] \cdot \frac{1}{2} + \Pr[A \mid b = 1] \cdot \frac{1}{2} \\ &= 1 \cdot \frac{1}{2} + \left(1 - \frac{1}{2^n - 1}\right) \cdot \frac{1}{2} \\ &= 1 - \frac{1}{2 \cdot (2^n - 1)} \end{aligned}$$

Por ende, el adversario gana el juego con una probabilidad significativamente mayor a $\frac{1}{2}$, y entonces OTP no es una PRP con dos rondas.

3. Definiciones y herramientas para la autentificación

3.1. Funciones de hash

Definición. Las funciones de hash **criptográficas** son funciones de un espacio de posibles mensajes a un espacio de mensajes de largo fijo:

$$h : \mathcal{M} \to \mathcal{H}$$

$\mathcal{M}$ es el espacio de mensajes y $\mathcal{H}$ es el espacio de posibles valores de la función de hash. Por ejemplo:

$$\mathcal{M} = \{0,1\}^n \quad \text{y} \quad \mathcal{H} = \{0,1\}^{128}$$

Decimos que $h(m)$ es el hash de un mensaje m.

Resistencia a preimágenes. Una propiedad básica de las funciones de hash es que debe existir un algoritmo **eficiente**, tal que dado un mensaje $m \in \mathcal{M}$, calcula $h(m)$. Además, no debe existir un algoritmo eficiente que, dado $x \in \mathcal{H}$, encuentra $m \in \mathcal{M}$ tal que $h(m) = x$. Esta propiedad se denota como ser resistente a preimagen.

¿Funciones criptográficas? Insistimos en el adjetivo criptográficas ya que estas funciones deben ser construidas para ser seguras y soportar ataques de adversarios, en la medida que sea posible. Por ejemplo, consideremos la siguiente función de hash:

$$h(m) = (A \cdot m + B) \bmod C$$

Suponemos que los mensajes son números naturales y que A, B y C son constantes, con C un número primo. ¿Es esta función resistente a preimagen? Digamos que $h(m) = (13 \cdot m + 97) \bmod 641$. ¿Podemos encontrar un mensaje m tal que $h(m) = 200$? Una combinación de herramientas de aritmética modular nos pueden dar una respuesta rápida:

$$h(501) = (13 \cdot 501 + 97) \bmod 641 = 200$$

Concluimos entonces que h **no** es resistente a preimágenes.

Resistencia a colisiones. Otra propiedad de las funciones de hash es que no debe existir un algoritmo eficiente que pueda encontrar $m_1, m_2 \in \mathcal{M}$ tales que $m_1 \neq m_2$ y $h(m_1) = h(m_2)$. Esta propiedad se denota como ser resistente colisiones.

Definición formal. Una función de hash es un par $(\texttt{Gen}, h)$ tal que:

- `Gen` es un algoritmo aleatorizado de tiempo polinomial. `Gen` toma como entrada un parámetro de seguridad 1^n y genera una llave s.
- h es un algoritmo de tiempo polinomial, que toma como entrada s y un mensaje $m \in \{0,1\}^*$, y retorna un hash $h^s(m) \in \{0,1\}^{\ell(n)}$, donde ℓ es un polinomio fijo.

Si $m \in \{0,1\}^{\ell'(n)}$ para un polinomio fijo ℓ' tal que $\ell'(n) > \ell(n)$, entonces $(\texttt{Gen}, h)$ es una función de hash de **largo fijo**.

Funciones despreciables. Sea $\mathbb{R}^+$ el conjunto de los números reales positivos, y $\mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$. Una función $f : \mathbb{N} \to \mathbb{R}_0^+$ es **despreciable** si:

$$\forall \text{ polinomio } p : \mathbb{N} \to \mathbb{R}.\ \exists n_0 \in \mathbb{N}.\ \forall n \geq n_0.\ f(n) < \frac{1}{p(n)}$$

- Por ejemplo, las funciones $f(n) = 2^{-n}$ y $f(n) = n^{-\log(n)}$ son funciones despreciables.
- Si f y g son funciones despreciables y p es un polinomio, entonces $f+g$ y $f \cdot p$ son funciones despreciables.

Formalizando la resistencia a colisiones. Considere una función de hash $(\texttt{Gen}, h)$. Definimos el juego $\texttt{Hash-Col}(n)$:

1. El verificador genera $s = \texttt{Gen}(1^n)$ y se lo entrega al adversario.
2. El adversario elige mensajes m_1 y m_2 con $m_1 \neq m_2$.
3. El adversario gana el juego si $h^s(m_1) = h^s(m_2)$, en caso contrario, pierde.

Una función de hash $(\texttt{Gen}, h)$ se dice resistente a colisiones si **para todo adversario que funciona como un algoritmo aleatorizado de tiempo polinomial**, existe una función despreciable $f(n)$ tal que:

$$\Pr[\text{Adversario gane } \texttt{Hash-Col}(n)] \leq f(n)$$

Como colorario de esta definición, si una función de hash es resistente a colisiones, entonces es resistente a preimágen.

3.2. Códigos de autentificación de mensajes (MAC)

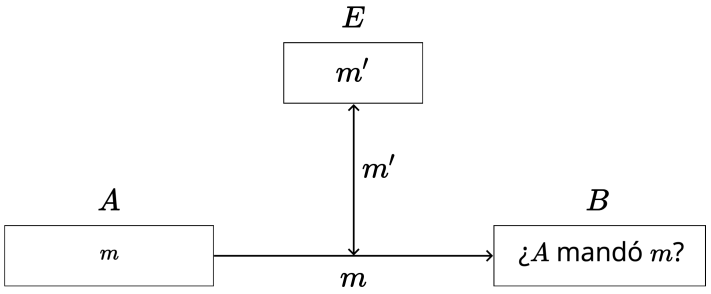


Figura 3: Escenario de autentificación de mensajes

Un objetivo básico de la criptografía es permitir que las partes se comuniquen de manera segura. Pero, ¿qué implica que "la comunicación sea segura"? Antes, mostramos cómo es posible lograr *perfect secrecy*; es decir, cómo se puede utilizar la encriptación para evitar que un adversario aprenda algo sobre los mensajes enviados a través de un canal público. Sin embargo, no toda la encriptación se relaciona con *perfect secrecy*, y no todos los adversarios se limitan a solo leer mensajes encriptados que pasan por el canal. En muchos casos, es de igual o mayor importancia garantizar la integridad del mensaje (o la autenticación del mensaje) contra un adversario que puede introducir mensajes en el canal o modificar mensajes en tránsito.

Encriptación vs. autentificación. Mantener un mensaje secreto y mantener la integridad de un mensaje son objetivos distintos, y por ende también lo son las técnicas y herramientas para lograr cada una de esas tareas. Desafortunadamente, mantener un mensaje secreto y la integridad del mismo a menudo se confunden y se entrelazan innecesariamente, así que seamos claros desde el principio: la encriptación no proporciona (en general) ninguna integridad, y **no se debe suponer que la encriptación garantiza la autenticación del mensaje** a menos que esté específicamente diseñada con ese propósito en mente.

Definiciones para MAC. Dado el problema anterior, necesitamos un mecanismo que ayude a las partes comunicantes a saber si el mensaje que se ha enviado a sido manipulado o no. La herramienta para esto son los *códigos de autentificación de mensajes* (MAC). Ahora, tendremos un espacio de llaves $\mathcal{K}$, un espacio de mensajes $\mathcal{M}$ y un espacio de **tags** $\mathcal{T}$, con los que definimos el esquema de autentificación de mensajes $(\texttt{Gen}, \texttt{Mac}, \texttt{Vrfy})$ tal que:

- $\texttt{Gen}$ es un algoritmo de generación de llaves que recibe un **parámetro de seguridad** 1^n y retorna una llave k, con $|k| \geq n$.

- $\texttt{Mac}$ es una familia de funciones:

$$\texttt{Mac} = \{\texttt{Mac}_k \mid k \in \mathcal{K}\}, \quad \text{con } \texttt{Mac}_k : \mathcal{M} \to \mathcal{T}, \forall k \in \mathcal{K}$$

  Es decir, es un algoritmo de generación de tags. Recibe un mensaje m y una llave k y retorna un tag t. Denotamos esto como $t = \texttt{Mac}_k(m)$

- $\texttt{Vrfy}$ es una familia de funciones:

$$\texttt{Vrfy} = \{\texttt{Vrfy}_k \mid k \in \mathcal{K}\}, \quad \text{con } \texttt{Vrfy}_k : \mathcal{M} \times \mathcal{T} \to \{0,1\}, \forall k \in \mathcal{K}$$

  Es decir, es un algoritmo de verificación. Recibe una llave k, un mensaje m y un tag t. Retorna un bit b, con $b = 1$ un mensaje válido y $b = 0$ un mensaje inválido. Denotamos esto como $b = \texttt{Vrfy}_k(m, t)$.

Requisito para MAC. En un esquema de autentificación de mensajes $(\texttt{Gen}, \texttt{Mac}, \texttt{Vrfy})$ es necesario que para todo n, para toda llave $k \in \mathcal{K}$ que retorna $\texttt{Gen}(1^n)$ y para cada mensaje $m \in \mathcal{M}$, se cumpla que:

$$\boxed{\texttt{Vrfy}_k(m, \texttt{Mac}_k(m)) = 1}$$

Juego para MAC. Un atacante puede romper el esquema de autentificación si logra generar una un tag falso, es decir, si produce un mensaje m junto con una etiqueta t tal que:

1. t es una etiqueta válida para el mensaje m (es decir, $\texttt{Vrfy}_k(m, t) = 1$), y
2. las partes comunicantes no habían autenticado previamente m.

Estas condiciones implican que si el adversario enviara (m, t) a una de las partes comunicantes, entonces esa parte sería engañada erróneamente pensando que m se originó en la otra parte legítima (ya que $\texttt{Vrfy}_k(m, t) = 1$) aunque no fue así.

Decimos que MAC es *inmune a falsificación (existentially unforgeable)* cuando el adversario es incapaz de falsificar un tag válido para cualquier mensaje. En base a esto, podemos definir el juego $\texttt{Forge}_{\texttt{Mac}}(n)$:

1. El verificador invoca $\texttt{Gen}(1^n)$ para generar una llave k, con n el parámetro de seguridad.
2. El adversario envía $m_0 \in \mathcal{M}$ al verificador.
3. El verificador responde $\texttt{Mac}_k(m_0)$.
4. Los pasos 2 y 3 se repiten tantas veces como quiera el adversario.
5. El adversario envía (m, t), siendo m un mensaje que no había enviado antes.

Decimos que el adversario **gana** el juego si se cumple que $\texttt{Vrfy}_k(m, t) = 1$, y pierde en caso contrario.

Seguridad de MAC. Decimos que un código de autentificación de mensajes MAC es inmune a falsificación, o en otras palabras, es **seguro**, si todo adversario A que juega $\texttt{Forge}_{\texttt{Mac}}(n)$ en tiempo polinomial en n tiene una probabilidad despreciable de ganar el juego, es decir:

$$\boxed{\Pr[A \text{ gane } \texttt{Forge}_{\texttt{Mac}}(n)] \leq f(n)}$$

con $f(n)$ una función despreciable.

Replay attacks. Los códigos de autenticación de mensajes no ofrecen protección contra *replay attacks* (ataques de repetición), en los que un atacante reenvía un mensaje previamente autenticado con su etiqueta válida. Aunque son una amenaza importante, un MAC no puede proteger contra estos ataques, ya que **no está diseñado para resistirlos**. La protección debe ser manejada a nivel de aplicación, ya que la decisión de si un mensaje repetido debe ser tratado como "válido" puede depender de la misma aplicación como tal.

4. Cifrado simétrico en la práctica

4.1. Redes de sustitución y permutación

Las redes de sustitución y permutación (SP-networks) son una estructura común utilizada en muchos algoritmos de cifrado por bloques, incluido AES, que profundizaremos un poco más adelante. Consisten en una serie de rondas de sustitución (S) y permutación (P), que juntas proporcionan una mezcla fuerte de los bits de entrada.

- **Sustitución (S):** En esta etapa, los bloques de bits de entrada se transforman mediante una operación no lineal. En el caso de AES, esto se realiza en la etapa SubBytes, donde cada byte del estado se sustituye por otro byte según una tabla de sustitución predefinida llamada S-box. Esta tabla ha sido diseñada para ser resistente a varios tipos de ataques criptográficos, y proporciona la propiedad de **confusión** en el cifrado, lo que significa que los bits de salida deben depender de manera compleja y no lineal de los bits de la clave.
- **Permutación (P)**: Después de la sustitución, los bits se reorganizan (o se permutan). Esto esencialmente dispersa las correlaciones estadísticas introducidas por la etapa de sustitución. En AES, esto se realiza en las etapas ShiftRows y MixColumns. ShiftRows es una operación de permutación simple que reordena los bytes dentro de cada bloque. MixColumns es una operación de permutación más compleja que mezcla los bytes entre los bloques. Estas operaciones proporcionan la propiedad de **difusión** en el cifrado, lo que significa que si un solo bit de entrada cambia, entonces los bits de salida deberían cambiar de manera aparentemente aleatoria e impredecible.

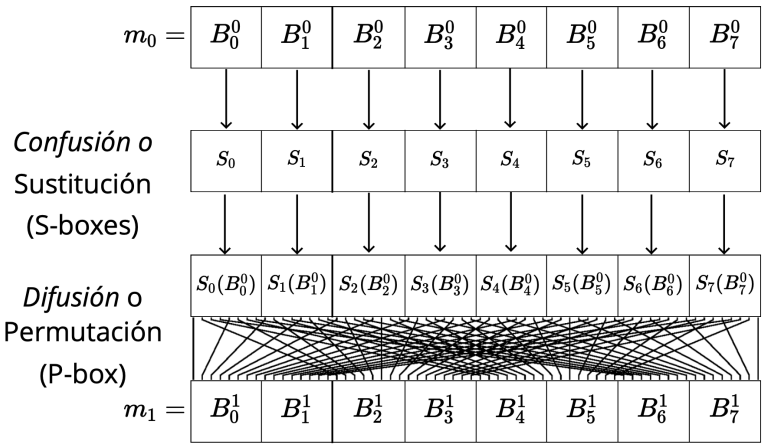


Figura 4: Representación de las sustituciones y permutaciones

Las SP-networks son poderosas porque combinan la confusión (alterando los valores de los datos) y la difusión (alterando la ubicación de los datos). Estas dos propiedades juntas hacen que el cifrado sea muy difícil de romper sin conocer la clave, porque cualquier cambio pequeño en la clave o en el texto plano de entrada debería producir cambios grandes y aparentemente aleatorios en el texto cifrado de salida.

El diseño de la S-box y las operaciones de permutación en AES tiene una base sólida en la teoría de números y la aritmética de campos finitos. Esta base teórica es una de las razones por las que AES es considerado seguro contra muchos tipos de ataques criptográficos.

4.2. Advanced Encryption Standard (AES)

AES es un algoritmo de cifrado por bloques que fue establecido por el Instituto Nacional de Estándares y Tecnología (NIST) de los Estados Unidos en 2001. Es un algoritmo de cifrado simétrico, lo que significa que utiliza la misma clave para cifrar y descifrar los datos. Esta clave puede tener una longitud de 128, 192 o 256 bits, lo que da lugar a las variantes AES-128, AES-192 y AES-256, respectivamente.

AES opera en bloques de datos de 128 bits, independientemente de la longitud de la clave. Si los datos que se van a cifrar no son múltiplos de 128 bits, deben rellenarse hasta el tamaño correcto. Además, AES es una **red de sustitución**, con 10 rondas para AES-128, 12 rondas para AES-192 y 14 rondas para AES-256.

Durante el cálculo del algoritmo AES, una matriz de 4x4 bytes llamada **matriz de estado** se modifica en una serie de rondas. La matriz de estado se establece inicialmente igual a la entrada del cifrado (notar que la entrada es de 128 bits, que son exactamente 16 bytes).

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Figura 5: Matriz de estado

Estudiemos el caso de AES-128. En cada ronda, las siguientes operaciones se aplican a la matriz de estado:

1. **AddRoundKey:** Se deriva, desde el *key schedule*, una subllave de 128 bits de la llave principal y se ve como una matriz de 4x4 bytes. La matriz de estado se actualiza mediante un XOR con esta nueva llave.

k_0^0	k_4^0	k_8^0	k_{12}^0
k_1^0	k_5^0	k_9^0	k_{13}^0
k_2^0	k_6^0	k_{10}^0	k_{14}^0
k_3^0	k_7^0	k_{11}^0	k_{15}^0

Figura 6: Representación de la matriz de una llave

El *key schedule* que genera las subllaves de todas las rondas funciona, a grandes rangos, de la siguienta manera:

- A la última columna de la matriz de 4x4 de la llave principal se realiza una operación **RotWord** que rota el primer byte de la columna al final de la misma.

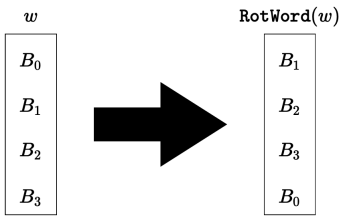


Figura 7: Representación de RotWord

- Se realiza una operación **SubWord** en el resultado de la operación RotWord. Esta operación aplica la S-box de AES a cada byte de la palabra clave.

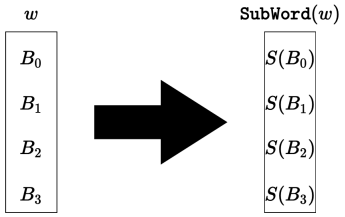


Figura 8: Representación de RotWord

- Se realiza una operación XOR entre el resultado de Subword, la columna que está 3 posiciones detrás (la primera columna de la llave principal), y una columna que es única por cada llave por ronda denominada **Rcon**.

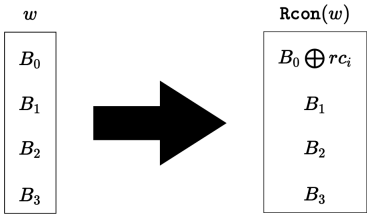


Figura 9: Representación de Rcon

- La primera iteración de RotWord, SubWord y Rcon dan origen a la primera columna de la primera subllave, para luego seguir con el proceso de hacer XOR con la columna 3 posiciones detrás hasta obtener las 4 columnas de la primera subllave.
  - La primera columna de la subllave hace XOR con la segunda de la llave principal, obteniendo la segunda columna de la subllave.
  - La segunda columna de la subllave hace XOR con la tercera de la llave principal, obteniendo la tercera columna de la subllave.
  - Finalmente, la tercera columna de la subllave hace XOR con la cuarta (última) columna de la llave principal, obteniendo la cuarta columna de la subllave, y completando así la matriz.
- Este proceso se repite hasta obtener las 10 subllaves para cada ronda, recordando que Rcon es un vector único en cada una de las rondas.

2. **SubBytes:** En este paso, cada byte de la matriz de estado se reemplaza por otro byte de acuerdo a una única tabla de búsqueda fija S. Esta tabla de sustitución (o S-box) es una permutación en $\{0,1\}^8$.

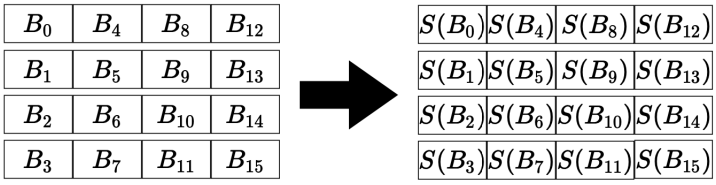


Figura 10: Representación de SubBytes

3. **ShiftRows:** A continuación, los bytes en cada fila de la matriz de estado se mezclan de la siguiente manera: la primera fila de la matriz no se toca, cada byte de la segunda fila se desplaza un lugar a la izquierda, la tercera fila se desplaza dos lugares a la izquierda, y la cuarta fila se desplaza tres lugares a la izquierda.

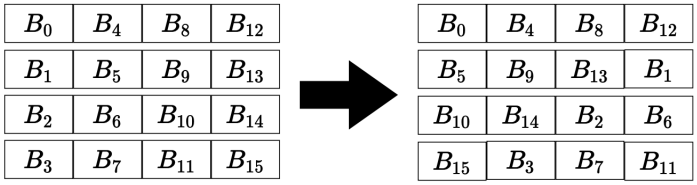


Figura 11: Representación de ShiftRows

4. **MixColumns:** Finalmente, se aplica una transformación lineal invertible a los cuatro bytes en cada columna. Esta transformación tiene la propiedad de que si dos entradas difieren en $b > 0$ bytes, entonces las salidas resultantes difieren en al menos $5 - b$ bytes.

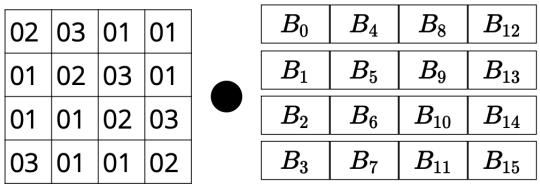
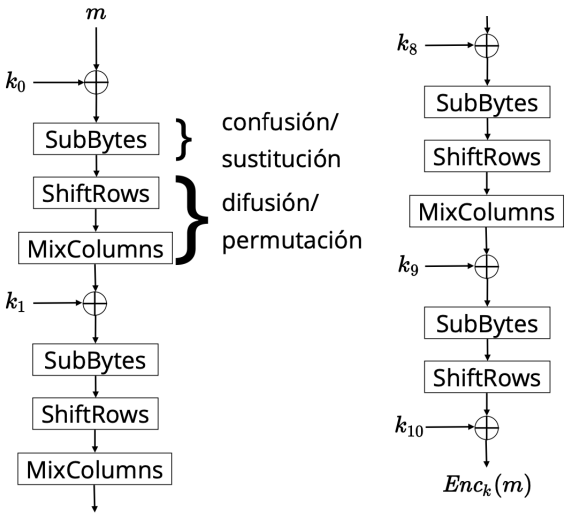


Figura 12: Representación de MixColumns

En la última ronda, MixColumns se reemplaza con AddRoundKey. Esto evita que un adversario simplemente invierta las últimas tres etapas, que no dependen de la clave.



	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 13: Ejemplo de AES-128 y la S-box

5. Funciones de hash y autentificación en la práctica

Una función de hash se construye de la siguiente manera:

- Se define una función de hash para mensajes de largo fijo, la cual es llamada **función de compresión**.
- Se define un método que permite utilizar de manera **iterativa** la función de compresión para construir una función de hash para mensajes de **largo arbitrario**.

Por ejemplo, digamos que queremos construir una función de hash de largo fijo (de compresión) tal que:

$$h : \{0,1\}^{2n} \to \{0,1\}^n$$

Para la construcción supondremos que tenemos un esquema criptográfico $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ sobre los espacios $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^n$, donde:

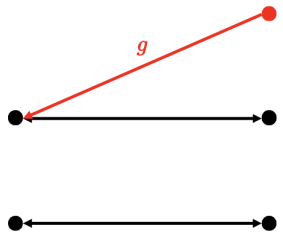
$$\texttt{Enc}_k : \{0,1\}^n \to \{0,1\}^n \quad \text{para cada } k \in \{0,1\}^n$$

En un primer intento, podemos definir, dados $u, v \in \{0,1\}^n$, la función $h(u||v) = \texttt{Enc}_u(v)$, es decir, nuestra función de hash encripta v usando a u como llave. El problema de esta implementación es que existe un algoritmo eficiente, en términos del parámetro de seguridad 1^n, que puede construir preimágenes de esta función h.

En términos más generales, fijemos una llave $k \in \{0,1\}^n$ y definamos las siguientes funciones de $\{0,1\}^n$ a $\{0,1\}^n$:

$$f(x) = \texttt{Enc}_k(x) \qquad g(x) = \texttt{Dec}_k(x)$$

Ya hemos demostrado anteriormente que $\forall x \in \{0,1\}^n : g(f(x)) = x$, pero, ¿se cumple también $f(g(x)) = x$? Esto depende del conjunto en el que se muevan las funciones f y g, por ejemplo:



Si hay alguna preimagen con dos preimágenes, no se cumple siempre que $f(g(x)) = x$, puesto que podríamos llegar a dos valores diferentes para x. Sin embargo, si nos encontramos en el caso en que $\mathcal{M} = \mathcal{C} = \{0,1\}^n$, tenemos que:

$$\forall x \in \{0,1\}^n : f(g(x)) = x$$

y por ende podemos concluir que:

$$\forall k \in \mathcal{K}. \forall m \in \mathcal{M} : \texttt{Enc}_k(\texttt{Dec}_k(m)) = m$$

Con lo anterior, podemos demostrar que nuestro primer intento de función de hash no es resistente a preimagen. Consideremos $u' \in \{0,1\}^n$ arbitrario, y definimos $v' = \texttt{Dec}_{u'}(x)$. Tenemos entonces que

$$h(u'||v') = \texttt{Enc}_{u'}(v') = \texttt{Enc}_{u'}(\texttt{Dec}_{u'}(x)) = x$$

Necesitamos una construcción mejor para funciones de hash.

5.1. Construcción de Davies-Meyer

La función de hash de Davies-Meyer se define como:

$$\boxed{h(u||v) = \texttt{Enc}_u(v) \oplus v}$$

Recordando que $\oplus$ representa la operación XOR, o en el caso de trabajar con bits, la suma en módulo 2.

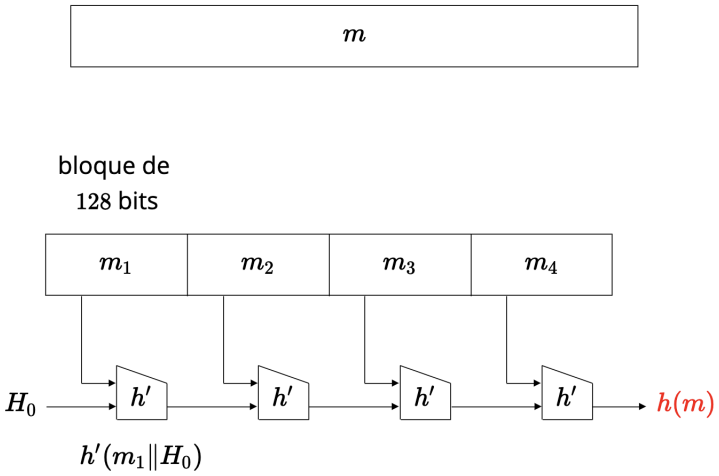
Formalización. Sea un esquema criptográfico $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ definido sobre los espacios $\mathcal{M} = \mathcal{K} = \mathcal{C} = \{0,1\}^*$. Definimos una función de hash $(\texttt{Gen}', h')$ de largo fijo tal que:

- $\texttt{Gen}'(1^n) = n$ para un parámetro de seguridad 1^n.
- $(h')^n : \{0,1\}^{2n} \to \{0,1\}^n$ tal que para cada $u, v \in \{0,1\}^n$:

$$(h')^n(u||v) = \texttt{Enc}_u(v) \oplus v$$

Propiedad fundamental. Si $(\texttt{Gen}, \texttt{Enc}, \texttt{Dec})$ es un esquema criptográfico *ideal*, entonces $(\texttt{Gen}', h')$ es resistente a colisiones. Por ende, $(\texttt{Gen}', h')$ es una buena alternativa para una función de compresión.

Extensión a largo arbitrario. Suponemos que tenemos una función de compresión $h' : \{0,1\}^{256} \to \{0,1\}^{128}$ y un mensaje m de largo arbitrario. Podemos dividir el mensaje en bloques de 128 bits e ir aplicando la función de hash de la siguiente manera:



H_0 es un **vector de inicialización** de la función de hash, y se define de manera previa. Para analizar la intuición de esta idea, es posible demostrar que si h' es resistente a colisiones, y no se conoce una preimagen de H_0, entonces h es resistente a colisiones. La idea de esta demostración se vuelve más sencilla en el caso de mensajes cuyo largo es divisible por 128, y donde los mensajes en una colisión tienen el mismo largo. Esta demostración queda propuesta para el lector.

El vector de inicialización H_0. ¿Podríamos pedir que H_0 sea un número sacado al azar? Esto es peligroso ya que un adversario puede sacar un mensaje $m_0 \in \{0,1\}^{256}$ y decir que $H_0 = h'(m_0)$. Este H_0 se ve como un número sacado al azar, pero el adversario conoce una preimagen, que es m_0.

Entonces, ¿cómo podemos estar seguros de que nadie conoce una preimagen de H_0? Lo que necesitamos es un **nothing-up-my-sleeve number**, es decir, un número que ha sido escogido de tal manera que evita la posibilidad de que alguien pueda argumentar que el número ha sido seleccionado con una intención oculta de manipular el sistema. Estos números ayudan a demostrar que no hay "trucos ocultos" en la elección de estas constantes, es decir, no hay nada escondido bajo la manga. Un ejemplo podría ser los primeros 128 bits de la representación de π en binario, ¿alguien podría conocer una preimagen de este valor de H_0?

Padding. En el caso en que los mensajes no sean divisibles por un valor n, debemos hacer **padding**, por ejemplo, agregar 0's al final del mensaje:

Sin embargo, esta no es una buena idea ya que es fácil encontrar colisiones: $h(m) = h(m0)$ si $|m|$ no es divisible por 128. Entonces, ¿qué debe cumplir una buena función de padding? Consideremos una función $Pad(\cdot)$ para bloques de largo n, por ejemplo, $n = 128$. Dado un mensaje $m \in \{0,1\}^*$, se debe tener que

$$|Pad(m)| \geq n \quad \text{y} |Pad(m)| \quad \text{es divisible por } n$$

Los axiomas fundamentales del padding son:

- m es un prefijo de $Pad(m)$.
- si $|m_1| = |m_2|$, entonces $|Pad(m_1)| = |Pad(m_2)|$.
- si $|m_1| \neq |m_2|$, entonces el último bloque de $Pad(m_1)$ es distinto del último bloque de $Pad(m_2)/$

Si un padding cumple con estos axiomas, entonces Pad es una **función inyectiva**. Si suponemos que $m_1 \neq m_2$, entonces:

- si $|m_1| \neq |m_2|$, entonces $Pad(m_1) \neq Pad(m_2)$ ya que el último bloque de $Pad(m_1)$ es distinto del último bloque de $Pad(m_2)$.
- si $|m_1| = |m_2|$, entonces $Pad(m_1) \neq Pad(m_2)$ ya que m_1 es prefijo de $Pad(m_1)$ y m_2 es prefijo de $Pad(m_2)$.

5.2. Construcción de Merkle-Damgård

Juntando todo lo anterior, supongamos dados:

- La función de compresión $(\texttt{Gen}, h')$ de Davies-Meyer.
- Para cada $n \in \mathbb{N}$, una función de padding Pad_n, que considera bloques con n elementos y satisface los axiomas fundamentales.

Vamos a definir una función de hash $(\texttt{Gen}, h)$ para mensajes de largo arbitrario. Consideremos un parámetro de seguridad 1^n, entonces, tenemos que $\texttt{Gen}(1^n) = s$ y luego $h^s : \{0,1\}^* \to \{0,1\}^n$.

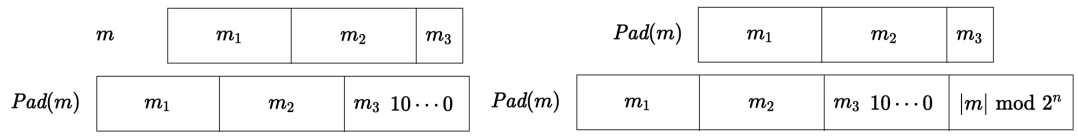
El valor del vector de inicialización H_0 está contenido en s, ya que s puede ser definido como (n, H_0). Luego, dado un $m \in \{0,1\}^*$, calculamos $h^s(m)$ de la siguiente forma:

1. Supongamos que $Pad_n(m) = m_1 m_2 \cdots m_\ell$, donde el largo de cada bloque m_i es n.
2. Para cada $i \in \{1, \ldots, \ell\}$: $H_i = (h')^n(m_1 || H_{i-1})$
3. $h^s(m) = H_\ell$

Esta construcción $(\texttt{Gen}, h)$ es resistente a colisiones. Queda como ejercicio propuesto al lector demostrar esta afirmación, dado que $(\texttt{Gen}', h')$ es resistente a colsiones y cada función de padding Pad_n satisface los axiomas fundamentales. Notar que:

- Podemos reemplazar la construcción de Davies-Meyer por cualquier función de compresión resistente a colisiones.
- Podemos considerar funciones de compresión de la forma $h' : \{0,1\}^{p(n)} \to \{0,1\}^n$, con $p(n)$ un polinomio tal que $p(n) > n$.

Pero, ¿qué función de padding usamos? Consideramos bloques de largo n:



Esta función de padding satisface los dos primeros axiomas fundamentales, pero pueden existir mensajes m_1 y m_2 tales que $|m_1| \neq |m_2|$ y los últimos bloques de m_1 y m_2 son iguales. Se debe tener que: $|m_1| \equiv |m_2| \bmod 2^n$, ya que si $|m_1| < |m_2|$, entonces $|m_2| > 2^n$.

Si $n = 128$, entonces $|m_2| > 2^{128}$, y m_2 debe tener al menos 10^{38} dígitos. ¿Es posible escribir un número de este tamaño? **No.** Cisco estima que el tráfico de Internet en 2025 será de 175 zettabytes, vale decir, $175 \cdot 10^{21}$ bytes.

5.3. Secure Hash Algorithm 2 (SHA-2)

SHA-2 es una familia de funciones de funciones de hash.

- Se definen utilizando la construcción de Merkle-Damgård
- Las funcoines de compresión son definidas utilizando la construcción de Davies-Meyer sobre un block cipher propio (no el de AES).

Por ejemplo, en SHA-256, el número se refiere al largo del hash. Esta función es una función de $\{0,1\}^*$ a $\{0,1\}^{256}$. SHA-256 puede considerarse como el resultado de instanciar el parámetro de seguridad el el valor 256 (de hecho, se usa en Bitcoin). También son utilizados otros ejemplos como SHA-224, SHA-384 y SHA-512.

SHA-256. Considera bloques de 512 bits, y sus estados internos H_i son de 256 bits. La función de compresión es de la forma:

$$h' : \{0,1\}^{512} \times \{0,1\}^{256} \to \{0,1\}^{256}$$

La función de padding se define utilizando las ideas descritas en las secciones anteriores, pero reservando los últimos 64 bits para el largo del mensaje m. Se realizan los siguientes pasos sobre el mensaje m:

1. Se agrega un símbolo 1.
2. Se agregan ℓ símbolos 0, donde ℓ es el menor número natural tal que $|m| + 1 + \ell \equiv 448 \bmod 512$.
3. Se agrega $|m| \bmod 2^{64}$.

El vector de inicialización H_0 se define como $H_0^1 H_0^2 H_0^3 H_0^4 H_0^5 1 H_0^6 1 H_0^7 H_0^8$, donde H_0^i tiene los primeros 32 bits de la parte decimal de la raíz cuadrada del i-ésimo número primo. Por ejemplo, H_0^1 tiene los primeros 32 bits de la parte decimal de $\sqrt{2}$:

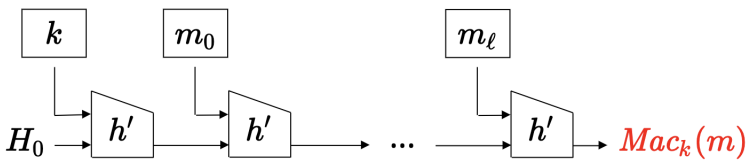
$$H_0^1 = 01101010000010011110011001100111$$

5.4. HMAC

HMAC, siglas en inglés correspondientes a *Hash-based message authentication code*, corresponde a la construcción de un código de autentificación de mensajes $\texttt{Mac}$ en base a una función de hash h. ¿Qué pasa si definimos $\texttt{Mac}_k(m) = h(k||m)$? Recordemos el juego que definía un buen MAC:

1. El verificador genera una llave k.
2. El adversario envía $m_0 \in \mathcal{M}$.
3. El verificador responde $\texttt{Mac}_k(m_0)$.
4. Los pasos 2 y 3 se repiten tantas veces como quiera el adversario.
5. El adversario envía (m, t), siendo m un mensaje que no se había enviado antes.

El adversario gana si $\texttt{Vrfy}_k(m,t) = 1$. Ahora, pensemos que estamos usando SHA-2, ¿puede el adversario ganar el juego? Hagamos una simplificación, el largo de k es un bloque:



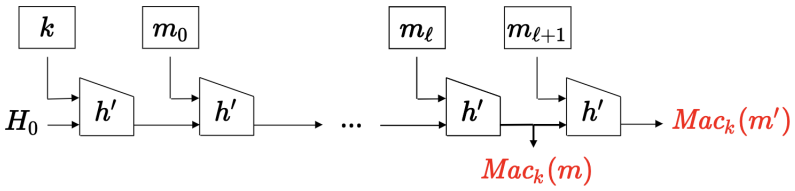
Donde:

- $k\, m_0 \cdots m_\ell$ es en realidad $Pad(k||m)$.
- En particular, $m \neq m_0 \cdots m_\ell$.

Entonces, ¿cómo se ve $Pad(Pad(k||m))$?

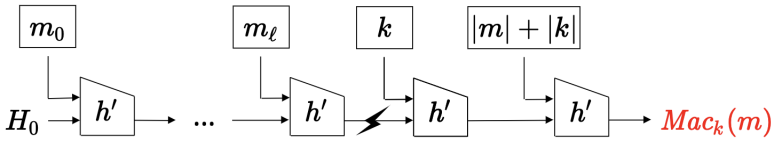
$$Pad(Pad(k||m)) = k\, m_0 \cdots m_\ell\, m_{\ell+1}$$

Por lo tanto, el adversario puede calcular y ganar el juego:



Donde $m' = m_0 \cdots m_\ell$. Este tipo de ataques se conoce como *length extension attacks*.

Si tratamos con $\texttt{Mac}_k(m) = h(m||k)$ (al revés del ejemplo anterior), no podemos hacer ataques de extensión de largo, pero si puede ocurrir una colisión de dos mensajes del mismo largo, que termina rompiendo el MAC.



Podemos seguir tratando varias funciones de hash, pero lo que podemos *intuir* que es seguro es

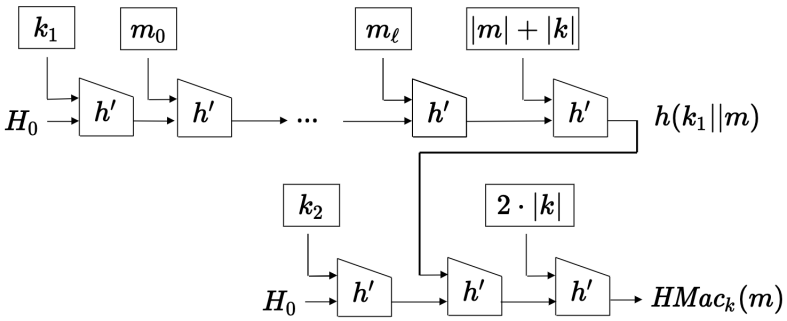
$$\texttt{Mac}_k(m) = h(k_2 \,||\, h(k_1 \,||\, m))$$

donde k_1 y k_2 ocupan exclusivamente un bloque, son distintas, se derivan de forma determinista a partir de k y no se pueden obtener sin k. El estándar se define de la siguiente forma:

$$k' = \begin{cases} h(k) & k \text{ usa más de un bloque} \\ k & \text{e.o.c.} \end{cases}$$

Así, podemos definir a `HMac` como:

$$\texttt{HMac}_k(m) = h(k_2 \,||\, h(k_1 \,||\, m))$$



6. Criptografía asimétrica o de clave pública