Criptografía: Función SHA-256

Javier Domínguez Gómez
jdg@member.fsf.org
Fingerprint: 94AD 19F4 9005 EEB2 3384 C20F 5BDC C668 D664 8E2B

v0.01 - Agosto 2018

Índice

1. Introducción

Este documento describe en detalle las características y el funcionamiento logico-matemático de la función *hash* criptográfica o algoritmo SHA-256.

Se trata de una función *hash* iterativa y unidireccional que puede procesar datos de entrada, como un cadena de texto o un archivo, para producir una representación condensada de longitud fija llamada *digest*. Este algoritmo determina de la integridad de los datos de entrada, es decir, cualquier cambio en los datos de entrada producirá un *digest* diferente. Esta propiedad es útil en la generación y verificación de firmas digitales y códigos de autenticación de mensajes, así como la generación de números aleatorios o *bits*.

Los puntos que vienen a continuación detallan cada uno de los elementos que forman parte del algoritmo empleado en la función *hash* SHA-256, tales como variables, constantes y funciones, y también el desarrollo y explicación de las operaciones de lógica proposicional, álgebra y operaciones con *bits* que se utilizan para obtener el mensaje *digest* adecuado.

2. Mensaje M o datos de entrada

Es la información que procesará la función SHA-256 para calcular el *digest* o *hash* correspondiente. M puede tener longitud variable[1], puede ser un archivo, una cadena de texto como *"Hola mundo"* e incluso una cadena vacía, obteniendo como resultado una cadena ω de 256 bits de longitud expresada en 64 caracteres hexadecimales o base 16.

$$sha256(M) = |\omega|_{16}^{256}$$

$$sha256('Hola\ mundo') \quad = \quad \begin{cases} ca8f60b2cc7f05837d98b208b57fb648 \\ 1553fc5f1219d59618fd025002a66f5c \end{cases}$$

La cadena hexadecimal generada se obtiene mediante una serie de cálculos en los que se emplean entre otros datos los bits de entrada, es decir, para un mismo mensaje M siempre se obtendrá el mismo *hash* criptográfico. Por el contrario, si se modifica un solo *bit*, por ejemplo cambiando o añadiendo un caracter (los espacios y los saltos de línea también son caracteres), se obtendrá un *hash* distinto.

[1] El esquema de relleno que utiliza SHA-256 requiere que el tamaño de la entrada se exprese como un número de 64 bits, es decir: $(2^{64} - 1)/8 \approx 2{,}091{,}752$ terabytes.

$$sha256('Hola\ mundo.') = \begin{cases} 8a3b7da2428acbc74623fb5a7b306a83 \\ b62b513371171e78c048fc12fbdb6ddf \end{cases}$$

$$sha256('abc') = \begin{cases} ba7816bf8f01cfea414140de5dae2223 \\ b00361a396177a9cb410ff61f20015ad \end{cases}$$

$$sha256('Abc') = \begin{cases} 06d90109c8cce34ec0c776950465421e \\ 176f08b831a938b3c6e76cb7bee8790b \end{cases}$$

De este modo, se garantiza la integridad de los datos. Si estos cambian también lo hará el *hash*.

2.1. Conversión a hexadecimal

Para procesar el mensaje de entrada M hay que realizar en primer lugar una conversión a formato hexadecimal de la misma. En este proceso se ha de sustituir cada caracter ASCII por su número hexadecimal equivalente. Esta información se puede consultar en una tabla ASCII[2]. Véase el siguiente ejemplo:

$\underline{Char}$		$\underline{Dec}$		$\underline{Hex}$
H	$\Rightarrow$	72	$\Rightarrow$	48
o	$\Rightarrow$	111	$\Rightarrow$	$6f$
l	$\Rightarrow$	108	$\Rightarrow$	$6c$
a	$\Rightarrow$	97	$\Rightarrow$	61
	$\Rightarrow$	32	$\Rightarrow$	20
m	$\Rightarrow$	109	$\Rightarrow$	$6d$
u	$\Rightarrow$	117	$\Rightarrow$	75
n	$\Rightarrow$	110	$\Rightarrow$	$6e$
d	$\Rightarrow$	100	$\Rightarrow$	64
o	$\Rightarrow$	111	$\Rightarrow$	$6f$

Uniendo cada código hexadecimal correspondiente a cada caracter se obtiene la conversión del mensaje de entrada a hexadecimal o base 16. Nótese que los espacios en blanco, o los signos de puntuación si los hubiere, también son caracteres ASCII, por lo tanto también tienen una equivalencia numérica.

$$\begin{aligned} Hola\ mundo &= 486f6c61206d756e646f \\ abc &= 616263 \\ Te\ gusta\ cifrar? &= 546520677573746120636966726172 3f \end{aligned}$$

Una vez se obtiene el mensaje de entrada en formato hexadecimal se ha de reservar el resultado a parte para utilizarlo mas adelante en otras funciones.

[2] https://www.ieee.li/computer/ascii.htm

2.2. Cálculo de la longitud de M

En el proceso para hallar el *digest* o *hash*, será necesario calcular la longitud del mensaje de entrada $|M|$. Primero hay que calcular la longitud en *bits*. Lo mas sencillo es multiplicar el número total de caracteres ASCII del mensaje de entrada por 8, ya que cada caracter ASCII equivale a 8 bits.

M		$Bits \times Chars$		Dec		Hex	$Longitud$
$Hola\ mundo$	=	8×10	=	80	$\Rightarrow$	50	$\|M\|_{16} = 50$
abc	=	8×3	=	24	$\Rightarrow$	18	$\|M\|_{16} = 18$
$Te\ gusta\ cifrar?$	=	8×16	=	128	$\Rightarrow$	80	$\|M\|_{16} = 80$

La longitud del mensaje de entrada $|M|$ en formato hexadecimal es un dato que se ha de reservar, ya que se utilizará para algunos cálculos en otras funciones.

3. Construcción de la variable W_t

La variable W_t es un array de 64 elementos que contiene palabras hexadecimales de 32 bits. Tiene un tamaño o longitud de 2048 bits (256 bytes) y se obtiene mediante la siguiente función recursiva definida por intervalos.

$$W_t = \begin{cases} M_i & si \quad 0 \leq i < 16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & si \quad 16 \leq i < 64 \end{cases}$$

Las funciones σ_0 y σ_1 realizan las siguientes operaciones lógicas de compresión.

$$\sigma_0(x) = ROT\ R^7(x) \oplus ROT\ R^{18}(x) \oplus SHR^3(x)$$
$$\sigma_1(x) = ROT\ R^{17}(x) \oplus ROT\ R^{19}(x) \oplus SHR^{10}(x)$$

Tal y como se indica en la función anterior, el primer intervalo es el que abarca los 16 primeros registros, o sea desde W_0 hasta W_{15}. El segundo intervalo es el esquema de los 48 registros restantes, es decir, desde W_{16} hasta W_{65}. Ambos intervalos tienen sus propias reglas que se explicarán en detalle en los siguientes puntos.

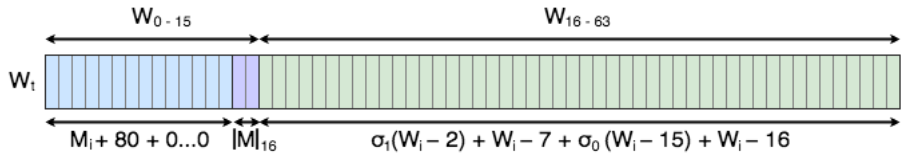


Figura 1: Representación gráfica de W_t

3.1. Los primeros 16 registros de W_t

En el primer intervalo de W_t se reservan los 16 primeros registros para almacenar el mensaje de entrada M en formato hexadecimal dividido en bloques de 32 bits. Tiene un tamaño o longitud de 512 bits. En el punto 2 de este documento se explica que el mensaje de entrada M puede tener una longitud variable, e incluso no tener longitud (cadena vacía). Sea cual sea la longitud, se ha de utilizar su representación hexadecimal, y se ha de dividir en palabras de 32 bits de izquierda a derecha. Si alguno de los bloques no llega a ocupar los 32 bits, los bits restantes se han de dejar vacíos para completarlos mas adelante.

$$Hola\ mundo = \overbrace{486f6c61}^{32\ bits}\ \overbrace{206d756e}^{32\ bits}\ \overbrace{646f}^{32\ bits}$$

A continuación hay que tomar un bit que represente el número 1 decimal o base 10, es decir 00000001, este se desplaza al bit más alto del byte, con lo que se obtiene 10000000 y finalmente se calcula el valor hexadecimal, que es 80.

$$10000000_2 = 80_{16}$$

Independientemente de la longitud de cadena hexadecimal de la palabra de entrada, se añade 80 por la derecha.

$$486f6c61 + 206d756e + 646f + 80$$

Ahora hay que añadir a la cadena una cantidad de bits con valor 0 hasta llegar a 448 bits en total, que es la longitud que abarca todos los intervalos que van desde W_0 hasta W_{13}.

$$\left.\begin{array}{l} \overbrace{486f6c61}^{32\ bits} + \overbrace{206d756e}^{32\ bits} + \overbrace{646f + 80 + 00}^{32\ bits} + \\ 00000000 + 00000000 + 00000000 + \\ 00000000 + 00000000 + 00000000 + \\ 00000000 + 00000000 + 00000000 + \\ 00000000 + 00000000 \end{array}\right\} = 448\ bits$$

Para terminar de completar W_t solo queda rellenar los últimos dos bloques de 32 bits W_{14} y W_{15} con la longitud del mensaje de entrada $|M|$ en hexadecimal, con tantos ceros por la izquierda como sean necesario para alcanzar 64 bits de longitud. La obtención de este dato se explica en el punto 2.2 de este documento.

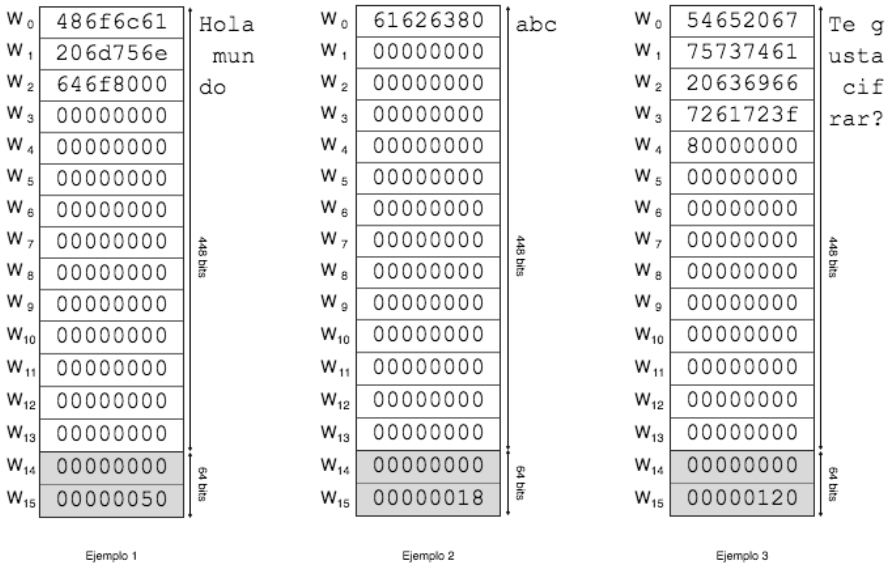


Figura 2: Tres ejemplos distintos de esquemas de relleno para las 16 primeras posiciones de W_t.

4. Constante K_t

La constante K_t se compone de 64 palabras hexadecimales. Cada una de esas palabras representa los 32 primeros bits (en hexadecimal) de la parte fraccionaria de la raíz cúbica de cada uno de los primeros 64 números primos. Estos números primos son:

2	3	5	7	11	13	17	19	23	29	31	37	41
43	47	53	59	61	67	71	73	79	83	89	97	101
103	107	109	113	127	131	137	139	149	151	157	163	167
173	179	181	191	193	197	199	211	223	227	229	233	239
241	251	257	263	269	271	277	281	283	293	307	311	

Se puede representar en código C la constante de tipo array de 64 elementos con su valor inicial de la siguiente forma:

```
static const unsigned int k[64] = {
        0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5,
        0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
        0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3,
        0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
        0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc,
        0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
        0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7,
        0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
        0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13,
        0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
        0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3,
        0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
        0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5,
        0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
        0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208,
        0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
};
```

4.1. Cálculo de las 64 palabras de K_t

Para obtener cada una de las palabras hexadecimales se han de seguir las siguientes operaciones:

1. En primer lugar se toma el primer número primo de lista de 64 números primos, el número 2, y se obtiene la parte fraccionaria de la raíz cúbica del mismo.

   $$\lfloor \sqrt[3]{2} \rfloor = 0{,}259921049894873$$

   Una vez se obtiene el resultado se reserva a parte para utilizarlo mas adelante.

2. Por otro lado se toma del array K la primera palabra K_0, es decir $0x428a2f98$, y se convierte a decimal o base 10.

   $$0x428a2f98 = \frac{4}{16^1} + \frac{2}{16^2} + \frac{8}{16^3} + \frac{10(a)}{16^4} + \frac{2}{16^5} + \frac{15(f)}{16^6} + \frac{9}{16^7} + \frac{8}{16^8}$$

$$\left.\begin{array}{rcll}
\frac{4}{16^1} & = \frac{4}{16} & = & 0{,}25 \\
\frac{2}{16^2} & = \frac{2}{256} & = & 0{,}0078125 \\
\frac{8}{16^3} & = \frac{8}{4096} & = & 0{,}001953125 \\
\frac{10}{16^4} & = \frac{10}{65536} & = & 0{,}00015258789063 \\
\frac{2}{16^5} & = \frac{2}{1048576} & = & 0{,}00000190734863 \\
\frac{15}{16^6} & = \frac{15}{16777216} & = & 0{,}00000089406967 \\
\frac{9}{16^7} & = \frac{9}{268435456} & = & 0{,}00000003352761 \\
\frac{8}{16^8} & = \frac{8}{4294967296} & = & 0{,}00000000186265
\end{array}\right\} \Sigma = 0{,}25992104969919$$

3. Se puede observar que los resultados obtenidos en los puntos 1 y 2 de esta sección se aproximan.

$$0{,}259921049894873 \approx 0{,}25992104969919$$

4. Por último hay que representar estos dos números decimales o base 10 en sistema hexadecimal o base 16, y tomar solo los primeros 32 bits, en cuyo caso ambos son iguales.

$$\begin{array}{rcl}
0{,}25992104969919_{10} & = & 0.\underbrace{428a2f98}_{32\ bits}0000c4163f52_{16} \\
0{,}259921049894873_{10} & = & 0.\underbrace{428a2f98}_{32\ bits}d728a242d2b4_{16}
\end{array}$$

5. Por lo tanto se puede decir que $0x428a2f98 = \lfloor \sqrt[3]{2} \rfloor$.

Para comprobar las otras 63 palabras hexadecimales solo hay que repetir los cálculos de los puntos anteriores con cada uno de los 63 números primos restantes, y comprobar su equivalencia respectiva con las palabras de la constante K_t:

$$\begin{array}{rcl}
K_0 \quad 0x428a2f98 & = & \lfloor \sqrt[3]{2} \rfloor \\
K_1 \quad 0x71374491 & = & \lfloor \sqrt[3]{3} \rfloor \\
K_2 \quad 0xb5c0fbcf & = & \lfloor \sqrt[3]{5} \rfloor \\
K_3 \quad 0xe9b5dba5 & = & \lfloor \sqrt[3]{7} \rfloor \\
\vdots & & \\
K_{63} \quad 0xc67178f2 & = & \lfloor \sqrt[3]{311} \rfloor
\end{array}$$

5. Las 8 palabras iniciales

La función SHA-256 utiliza una grupo inicial de 8 palabras correlacionadas una a una con las variables A, B, C, D, E, F, G y H. Cada una de estas palabras representa los 32 primeros bits en hexadecimal o base 16 de la parte fraccionaria de las raíces cuadradas de los primeros 8 números primos. Se trata de los 8 valores iniciales que se reciben en la primera ronda criptográfica y que se asignarán a un array de 8 elementos denominado P_t, donde:

$$\begin{array}{cccc} P_0 = A & P_1 = B & P_2 = C & P_3 = D \\ P_4 = E & P_5 = F & P_6 = G & P_7 = H \end{array}$$

Se puede representar en código C el array de 8 elementos con su valor inicial de la siguiente forma:

```
unsigned int p[8] = {
        0x6a09e667, 0xbb67ae85, 0x3c6ef372,
        0xa54ff53a, 0x510e527f, 0x9b05688c,
        0x1f83d9ab, 0x5be0cd19
};
```

Los 8 valores iniciales son siempre los mismos únicamente en la primera ronda criptográfica. En las siguientes rondas estas palabras irán cambiando cíclicamente su valor de entrada.

5.1. Cálculo de las 8 palabras iniciales

Para obtener estas 8 palabras de la primera ronda criptográfica se deben seguir las siguientes operaciones:

1. Se toman los 8 primeros números primos, es decir, 2, 3, 5, 7, 11, 13, 17 y 19. A continuación se obtiene la raíz cuadrada de cada uno de ellos y se reserva únicamente la parte fraccionaria.

$$\begin{array}{lcl} \sqrt[2]{2} \approx 1{,}4142135623730950 & \rightarrow & \lfloor \sqrt[2]{2} \rfloor \approx 0{,}4142135623730950 \\ \sqrt[2]{3} \approx 1{,}7320508075688770 & \rightarrow & \lfloor \sqrt[2]{3} \rfloor \approx 0{,}7320508075688770 \\ \sqrt[2]{5} \approx 2{,}2360679774997900 & \rightarrow & \lfloor \sqrt[2]{5} \rfloor \approx 0{,}2360679774997900 \\ \sqrt[2]{7} \approx 2{,}6457513110645910 & \rightarrow & \lfloor \sqrt[2]{7} \rfloor \approx 0{,}6457513110645910 \\ \sqrt[2]{11} \approx 3{,}316624790355400 & \rightarrow & \lfloor \sqrt[2]{11} \rfloor \approx 0{,}316624790355400 \\ \sqrt[2]{13} \approx 3{,}605551275463989 & \rightarrow & \lfloor \sqrt[2]{13} \rfloor \approx 0{,}605551275463989 \\ \sqrt[2]{17} \approx 4{,}123105625617661 & \rightarrow & \lfloor \sqrt[2]{17} \rfloor \approx 0{,}123105625617661 \\ \sqrt[2]{19} \approx 4{,}358898943540674 & \rightarrow & \lfloor \sqrt[2]{19} \rfloor \approx 0{,}358898943540674 \end{array}$$

2. Se convierte la parte fraccionaria de sistema decimal o base 10 a hexadecimal o base 16.

$$
\begin{aligned}
0{,}414213562373095_{10} &= 0{,}6A09E667_{16} \\
0{,}732050807568877_{10} &= 0.BB67AE85_{16} \\
0{,}236067977499790_{10} &= 0{,}3C6EF372_{16} \\
0{,}645751311064591_{10} &= 0.A54FF53A_{16} \\
0{,}316624790355400_{10} &= 0{,}510E527F_{16} \\
0{,}605551275463989_{10} &= 0{,}9B05688C_{16} \\
0{,}123105625617661_{10} &= 0{,}1F83D9AB_{16} \\
0{,}358898943540674_{10} &= 0{,}5BE0CD19_{16}
\end{aligned}
$$

3. Finalmente se almacena únicamente la parte fraccionaria hexadecimal como las 8 palabras de entrada A, B, C, D, E, F, G y H.

$$
\begin{aligned}
A &= 0x6A09E667 & E &= 0x510E527F \\
B &= 0xBB67AE85 & F &= 0x9B05688C \\
C &= 0x3C6EF372 & G &= 0x1F83D9AB \\
D &= 0xA54FF53A & H &= 0x5BE0CD19
\end{aligned}
$$

6. Primera ronda criptográfica

Habiendo calculado anteriormente las 8 palabras iniciales A, B, C, D, E, F, G y H, las 64 palabras variables de W_t y conociendo las 64 palabras constantes de K_t, ya se tienen los elementos necesarios para realizar la primera ronda criptográfica. Para ello es necesario seguir estrictamente unas reglas y funciones que se detallan en los siguientes puntos. El siguiente diagrama muestra los movimientos y funciones por los que ha de pasar cada una de las 8 palabras iniciales en cada ronda, hasta completar 64 rondas.

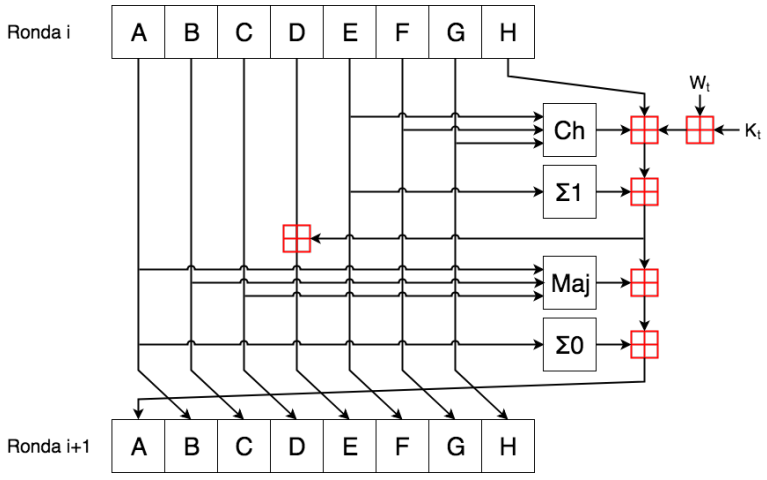


Figura 3: Diagrama del funcionamiento cíclico de SHA-256

Los recuadros de color rojo del diagrama representan una suma $mod\ 2^{32}$, o lo que es lo mismo, una suma binaria de los inputs de 32 bits que reciba sin tener en cuanta las unidades que se deban llevar al siguiente nivel.

6.1. Operaciones con las 8 palabras

En la primera ronda criptográfica las palabras A, B, C, D, E, F, G y H tendrán siempre el mismo valor inicial, tal y como se explica en el punto 5 de este documento. Hay que hacer una serie de operaciones y movimientos con los valores de cada una de las palabras, de modo que se obtenga un nuevo valor de las mismas para la siguiente ronda. Es decir, en la primera ronda A_0 siempre tendrá como valor $6A09E667$, pero en la segunda ronda A_1 tendrá un valor muy distinto, dependerá del resultado de una serie de movimientos y operaciones, y esto sucederá con el todas y cada una de las palabras.

6.1.1. Operaciones con A

La palabra A se va a utilizar en tres casos por cada ronda:

- El valor actual de A_i será el nuevo valor que tendrá la palabra B_{i+1} en la siguiente ronda. En este caso no se hace ningún cálculo, solo se transfiere el valor.

$$A_i \Rightarrow B_{i+1}$$

- Se pasa como argumento a la función Maj y se utilizará para obtener el nuevo valor de A_{i+1} en la siguiente ronda.
- Se pasa como argumento a la función $\Sigma 0$ y también se utilizará como elemento para calcular el nuevo valor de A_{i+1} en la siguiente ronda.

6.1.2. Operaciones con B y C

Las palabras B y C se van a utilizar cada una en dos casos por cada ronda:

- El valor actual de B_i será el nuevo valor que tendrá la palabra C_{i+1}, y el valor de C_i será el nuevo valor que tendrá D_{i+1} en la siguiente ronda. En este caso no se hace ningún cálculo, solo se transfiere el valor.

$$B_i \Rightarrow C_{i+1} \qquad C_i \Rightarrow D_{i+1}$$

- Sendos valores B_i y C_i se pasan como argumento a la función Maj para obtener el nuevo valor de A_{i+1} en la siguiente ronda.

6.1.3. Operaciones con D

La palabra D se va a utilizar en un único caso por cada ronda. Su valor actual D_i será un elemento más en la operación $mod\ 2^{32}$ junto con los valores de W_t, K_t, H_i, el resultado de la función Ch y el resultado de la función $\Sigma 1$ para obtener el valor de E_{i+1}.

$$mod\ 2^{32}(W_t, K_t, H_i, Ch, \Sigma 1, D_i) = E_{i+1}$$

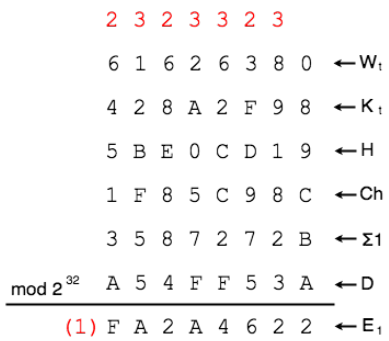


Figura 4: La operación $mod\ 2^{32}$ con W_t, K_t, H, Ch, $\Sigma 1$ y D.

El ejemplo anterior muestra los datos con los que se ha de operar en la primera ronda criptográfica para el mensaje de entrada $M =$ "abc". El resultado que se obtiene es el valor que tendrá E_1 (la palabra E en la segunda ronda). Los números en color rojo de la parte superior son el acarreo de la suma de cada caracter hexadecimal, y el número rojo de la parte inferior izquierda es el número hexadecimal que es excluido.

6.1.4. Operaciones con E

La palabra E se va a utilizar en tres casos por cada ronda:

- El valor actual de E_i será el nuevo valor que tendrá la palabra F_{i+1} en la siguiente ronda. En este caso no se hace ningún cálculo, solo se transfiere el valor.

$$E_i \Rightarrow F_{i+1}$$

- Se pasa como argumento a la función Ch para obtener el nuevo valor de A_{i+1} en la siguiente ronda.
- Se pasa como argumento a la función $\Sigma 1$ y también se utilizará como elemento para calcular el nuevo valor de A_{i+1} en la siguiente ronda.

6.1.5. Operaciones con F y G

Las palabras F y G se van a utilizar cada una en dos casos por cada ronda:

- El valor actual de F_i será el nuevo valor que tendrá la palabra G_{i+1}, y el valor de G_i será el nuevo valor que tendrá H_{i+1} en la siguiente ronda. En este caso no se hace ningún cálculo, solo se transfiere el valor.

$$F_i \Rightarrow G_{i+1} \qquad G_i \Rightarrow H_{i+1}$$

- Sendos valores F_i y G_i se pasan como argumento a la función Ch para obtener el nuevo valor de A_{i+1} en la siguiente ronda.

6.1.6. Operaciones con H

La palabra H se va a utilizar en un único caso por cada ronda. Su valor actual será un elemento más en la operación $mod\ 2^{32}$ junto con los valores de W_t, K_t, el resultado de las funciónes Ch, $\Sigma 1$, Maj y $\Sigma 0$ para obtener el valor de A_{i+1}

6.2. Función Ch

Se trata de una función booleana que realizará operaciones lógicas tomando como datos de entrada las palabras E, F y G. Se opera con cada bit de la palabra hexadecimal aplicando la siguiente fórmula de lógica proposicional:

$$Ch(E, F, G) = (E \wedge F) \vee (\neg E \wedge G)$$

E	F	G	$(E \wedge F)$	$\vee$	$(\neg E \wedge G)$
1	1	1	1	**1**	0
1	1	0	1	**1**	0
1	0	1	0	**0**	0
1	0	0	0	**0**	0
0	1	1	0	**1**	1
0	1	0	0	**0**	0
0	0	1	0	**1**	1
0	0	0	0	**0**	0
1	1	1	1	**1**	0
1	1	0	1	**1**	0
1	0	1	0	**0**	0
1	0	0	0	**0**	0
0	1	1	0	**1**	1
0	1	0	0	**0**	0
0	0	1	0	**1**	1
0	0	0	0	**0**	0

Cuadro 1: *Tabla de la verdad de la función Ch*

En el *álgebra de Boole*[3] la anterior tabla se denomina *tabla de la verdad*, y en este caso representa las 16 posibilidades binarias existentes en el caso de un input de tres entradas δ_x, δ_y y δ_z. Además muestra el resultado en cada una de las posibilidades para la operación lógica $(\delta_x \wedge \delta_y) \vee (\neg\delta_x \wedge \delta_z)$.

En la primera ronda criptográfica los valores de las palabras E, F y G son constantes, tal y como se indica en el punto 5.1 de este documento. El resultado de la función $Ch(E_0, F_0, G_0)$ es el siguiente:

$E_0 \rightarrow$	5	1	0	E	5	2	7	F
	0101	0001	0000	1110	0101	0010	0111	1111
$F_0 \rightarrow$	9	B	0	5	6	8	8	C
	1001	1011	0000	0101	0110	1000	1000	1100
$G_0 \rightarrow$	1	F	8	3	D	9	A	B
	0001	1111	1000	0011	1101	1001	1010	1011
$Ch \rightarrow$	1	F	8	5	C	9	8	C
	0001	1111	1000	0101	1100	1001	1000	1100

Figura 5: Cálculo bit a bit para $(E_0 \wedge F_0) \vee (\neg E_0 \wedge G_0)$

Se puede comprobar en la *tabla de la verdad* los resultados de las operaciones de 3 bits en cada caso. El resultado siempre se expresará en hexadecimal.

$$Ch_0 = 0x1F85C98C$$

6.3. Función $\Sigma 1$

Esta función realiza una serie de operaciones binarias rotativas ROT y operaciones lógicas XOR teniendo como input el valor de E_i en cada ronda criptográfica. La fórmula para cada bit x es la siguiente:

$$\Sigma 1(x) = ROT\ R^6(x) \oplus ROT\ R^{11}(x) \oplus ROT\ R^{25}(x)$$

Tal y como se explica en el punto 5.1 de este documento, la palabra E_0 es una constante con valor hexadecimal $0x510E527F$ únicamente en la primera ronda. A continuación un ejemplo del cálculo de la función $\Sigma 1_0$.

[3] Álgebra de Boole: https://en.wikipedia.org/wiki/Boolean_algebra

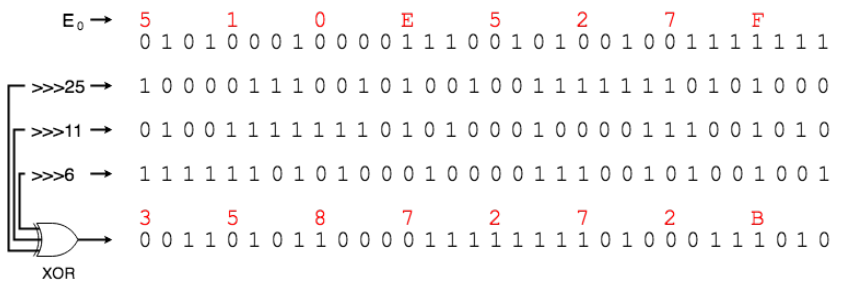


Figura 6: Cálculo bit a bit para $ROT\ R^6(x) \oplus ROT\ R^{11}(x) \oplus ROT\ R^{25}(x)$

En el ejemplo anterior se realiza una operación XOR de tres inputs, que son cada uno de los bits de E_0 tras realizar una operación de rotación de 25 bits a la derecha, 11 bits a la derecha y 6 bits a la derecha también. El resultado siempre se expresará en hexadecimal.

$$\Sigma 1_0 = 0x3587272B$$

6.4. Función Maj

Se trata de una función booleana que realizará operaciones lógicas tomando como datos de entrada las palabras A, B y C. Se opera con cada bit de la palabra hexadecimal aplicando la siguiente fórmula de lógica proposicional:

$$Maj(A, B, C) = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$$

A	B	C	$(A \wedge B)$	$\vee$	$(A \wedge C)$	$\vee$	$(B \wedge C)$
1	1	1	1	1	1	**1**	1
1	1	0	1	1	0	**1**	0
1	0	1	0	1	1	**1**	0
1	0	0	0	0	0	**0**	0
0	1	1	0	0	0	**1**	1
0	1	0	0	0	0	**0**	0
0	0	1	0	0	0	**0**	0
0	0	0	0	0	0	**0**	0
1	1	1	1	1	1	**1**	1
1	1	0	1	1	0	**1**	0
1	0	1	0	1	1	**1**	0
1	0	0	0	0	0	**0**	0
0	1	1	0	0	0	**1**	1
0	1	0	0	0	0	**0**	0
0	0	1	0	0	0	**0**	0
0	0	0	0	0	0	**0**	0

Cuadro 2: *Tabla de la verdad de la función Maj*

La tabla anterior muestra el resultado en cada una de las 16 posibilidades binarias para la operación lógica $(\delta_x \wedge \delta_y) \vee (\delta_x \wedge \delta_z) \vee (\delta_y \wedge \delta_z)$.

En la primera ronda criptográfica los valores de las palabras A, B y C son constantes, tal y como se indica en el punto 5.1 de este documento. El resultado de la función $Maj(A_0, B_0, C_0)$ es el siguiente:

Figura 7: Cálculo bit a bit para $(A_0 \wedge B_0) \vee (A_0 \wedge C_0) \vee (B_0 \wedge C_0)$

El resultado siempre se expresará en hexadecimal.

$$Maj_0 = 0x3A6FE667$$

6.5. Función $\Sigma 0$

Al igual que la función $\Sigma 1$ esta función realiza una serie de operaciones binarias rotativas ROT y operaciones lógicas XOR teniendo como input el valor de A_i en cada ronda criptográfica. La fórmula para cada bit x es la siguiente:

$$\Sigma 0(x) = ROT\ R^2(x) \oplus ROT\ R^{13}(x) \oplus ROT\ R^{22}(x)$$

Tal y como se explica en el punto 5.1 de este documento, la palabra A_0 es una constante con valor hexadecimal $0x6A09E667$ únicamente en la primera ronda. A continuación un ejemplo del cálculo de la función $\Sigma 0_0$.

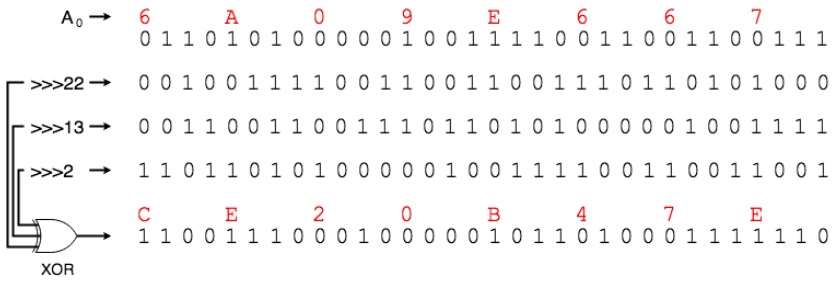


Figura 8: Cálculo bit a bit para $ROT\ R^2(x) \oplus ROT\ R^{13}(x) \oplus ROT\ R^{22}(x)$

En el ejemplo anterior se realiza una operación XOR de tres inputs, que son cada uno de los bits de A_0 tras realizar una operación de rotación de 2 bits

a la derecha, 13 bits a la derecha y 22 bits a la derecha también. El resultado siempre se expresará en hexadecimal.

$$\Sigma 1_0 = 0xCE20B47E$$

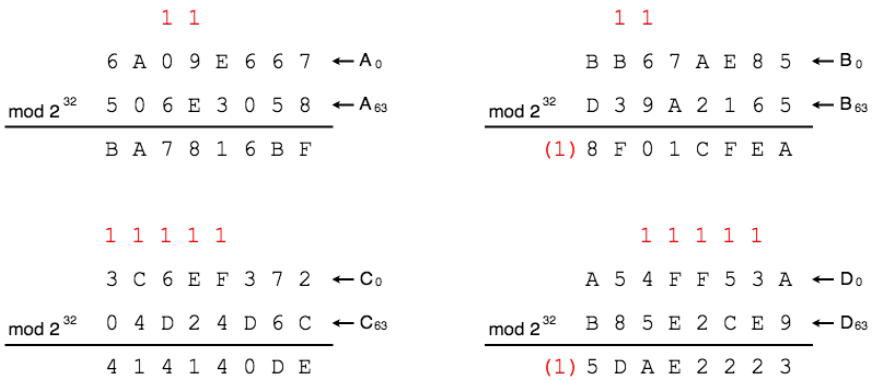
6.6. Las 8 nuevas palabras resultantes

Como resultado de las anteriores operaciones se obtienen los nuevos valores de las 8 palabras A, B, C, D, E, F, G y H para las siguient ronda.

$$\begin{aligned}
A_{i+1} &= mod\ 2^{32}(W_t, K_t, Ch, \Sigma 1, Maj, \Sigma 0) \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i \\
D_{i+1} &= C_i \\
E_{i+1} &= mod\ 2^{32}(W_t, K_t, H_i, Ch, \Sigma 1, D_i) \\
F_{i+1} &= E_i \\
G_{i+1} &= F_i \\
H_{i+1} &= G_i
\end{aligned}$$

7. Siguientes rondas

En total se han de realizar 64 rondas, y en cada ronda las 8 palabras A, B, C, D, E, F, G y H irán tomando nuevos valores. Una vez terminadas todas las rondas criptográficas, antes de obtener la cadena *digest* del mensaje original M primero se han de sumar los valores de la palabra correspondiente a la primera ronda P_0 y la última P_{63} en una operación $mod\ 2^{32}$, por ejemplo, para los datos de entrada $M =$ "abc":



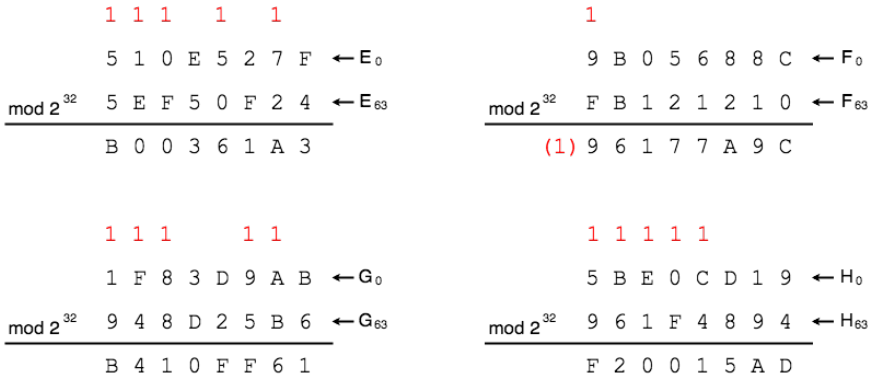


Figura 9: Cálculo bit a bit *mod* 2^{32} de las palabras P_0 y P_{63}.

Finalmente se han de concatenar los resultados de las operaciones anteriores de la siguiente manera:

$$A+B+C+D+E+F+G+H = \begin{cases} ba7816bf8f01cfea414140de5dae2223 \\ b00361a396177a9cb410ff61f20015ad \end{cases}$$

Así pues, el *hash* o *digest* resultante para la cadena o datos de entrada "*abc*"será siempre la misma cadena de 256 bits de longitud expresada en 64 caracteres hexadecimales o base 16.

$$sha256('abc') = \begin{cases} ba7816bf8f01cfea414140de5dae2223 \\ b00361a396177a9cb410ff61f20015ad \end{cases}$$