Ceci est un document qui complète le support de cours vue ensemble

# Les caractéristiques de SHA-1

* Taille du message : 2^64 bits maximum
* Taille des blocs : 512 bits
* Taille des mots : 32 bits
* Taille du condensé : 160 bits

Complexité de l’attaque sur SHA-1 :

En se basant sur le théorème de « paradoxe des anniversaires », la complexité est calculée par la formule suivante :

C = 2(n/2) [ où n= taille empreinte ]  
🡺 C = 2(160/2)

En théorie, et sous des conditions spécifique la complexité a diminué à :

🡺 C = 263

# Algorithme SHA-1

Commence par un prétraitement du message, cette opération se déroule en trois étapes : initialiser les valeurs de hachage, compléter le message M (msg) et découper en blocs.  
Après le prétraitement vient les itérations de hachage.

## Définition des constantes

// constant K

const K = [ 0x5a827999, 0x6ed9eba1, 0x8f1bbcdc, 0xca62c1d6 ];

// Vecteur d’initialisation (IV)

const H = [ 0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476, 0xc3d2e1f0 ];

## Compléter le message M (msg) et découpage en blocs (512bits)

Il s'agit ici d'ajouter des informations à M (msg) pour qu'il soit d'une taille multiple de 512 bits.  
pour ce faire, l'on ajoute un bit «1» à la fin du message M (msg), puis (Z) zéros «0»     
où (Z) est la plus petite solution non négative de l'équation :   
S + 1 + Z = (512-64) mod 512, [avec S = taille de M(msg) en bits (msg.length)]  
les 64 bits restants sont réservés pour mettre dedans la valeur de la taille S en bits.

msg += String.fromCharCode(0x80); // add trailing '1' bit (+ 0's padding) to string

// convert string msg into 512-bit/16-integer blocks arrays of ints

const l = msg.length/4 + 2; // length (in 32-bit integers) of msg + ‘1’ + appended length

const N = Math.ceil(l/16); // number of 16-integer-blocks required to hold 'l' ints

const M = new Array(N);

for (let i=0; i<N; i++) {

M[i] = new Array(16); // block(i) of 512bits = 16\*32

for (let j=0; j<16; j++) { // encode 4 chars per integer, big-endian encoding

M[i][j] = (msg.charCodeAt(i\*64+j\*4+0)<<24) | (msg.charCodeAt(i\*64+j\*4+1)<<16)

| (msg.charCodeAt(i\*64+j\*4+2)<< 8) | (msg.charCodeAt(i\*64+j\*4+3)<< 0);

} // note running off the end of msg is ok 'cos bitwise ops on NaN return 0

}

// les 64 bits restants sont réservés pour mettre dedans la valeur de la taille bits.

// add length (in bits) into final pair of 32-bit integers (big-endian)

// note: most significant word would be (len-1)\*8 >>> 32, but since JS converts

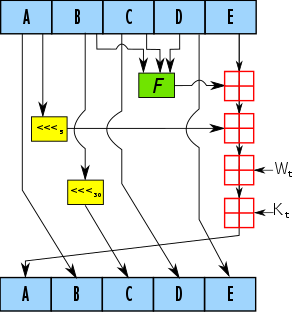
// bitwise-op args to 32 bits, we need to simulate this by arithmetic operators

M[N-1][14] = ((msg.length-1)\*8) / Math.pow(2, 32); M[N-1][14] = Math.floor(M[N-1][14]);

M[N-1][15] = ((msg.length-1)\*8) & 0xffffffff;

## Procéder au calcul des hachés itérativement

* Un tour de calcul SHA-1:



// HASH COMPUTATION

for (let i=0; i<N; i++) {

const W = new Array(80);

// 1 - prepare message schedule 'W'

for (let t=0; t<16; t++) W[t] = M[i][t]; //mettre 16 sous-block de 32 bits dans W

// une procédure d'expansion de W[] pour compléter le reste des 80 mots de 32 bits

for (let t=16; t<80; t++) W[t] = Sha1.ROTL(W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16], 1);

// 2 - initialize five working variables a, b, c, d, e with previous hash value

let a = H[0], b = H[1], c = H[2], d = H[3], e = H[4];

// 3 - main loop (use JavaScript '>>> 0' to emulate UInt32 variables)

for (let t=0; t<80; t++) {

// s détermine l’intervalle de calcul pour la fonction (f) et la const.(k)

const s = Math.floor(t/20);

const T = (Sha1.ROTL(a, 5) + Sha1.f(s, b, c, d) + e + K[s] + W[t]) >>> 0;

e = d;

d = c;

c = Sha1.ROTL(b, 30) >>> 0;

b = a;

a = T;

}

// 4 - compute the new intermediate hash value (note 'addition modulo 2^32' – JavaScript

// '>>> 0' coerces to unsigned UInt32 which achieves modulo 2^32 addition)

H[0] = (H[0]+a) >>> 0;

H[1] = (H[1]+b) >>> 0;

H[2] = (H[2]+c) >>> 0;

H[3] = (H[3]+d) >>> 0;

H[4] = (H[4]+e) >>> 0;

}

# Références et liens:

pour plus de détail consulter la page : <https://www.movable-type.co.uk/scripts/sha1.html>  
Implémentation en Javascript.

Un autre lien utile pour approfondir vos connaissances : <https://zestedesavoir.com/tutoriels/1895/les-fonctions-de-hachage-cryptographiques/>