1

Introduction to Cryptography

1.1. The encryption function

The encryption function is the mechanism used to provide a confidentiality service. It enables the modification of a string of bytes (the data being transmitted) in order to make it incomprehensible to anyone who is not authorized to know its content.

Encryption is done using two types of algorithm:

– symmetrical or secret-key algorithms. The same (secret) key is used for encryption and decryption;

– asymmetrical or public key algorithms. Different keys are used for encryption and decryption. The public key (or, conversely, the private key) is used for encryption. The private key (or, conversely, the public key) is used for decryption.

Symmetrical algorithms are grouped into two categories:

– stream cipher algorithms act on bits. A stream cipher generally consists of an exclusive OR or XOR (eXclusive OR) operation between data issued by a pseudo-random number-generator and the data being transmitted;

– block cipher algorithms act on blocks of between 32 and 512 bits in size.

Asymmetrical algorithms are based on modular exponentiation or an elliptical curve.

Asymmetrical algorithms based on modular exponentiation are not adapted to data encryption because they have the disadvantage of being slow. This is due to the size of the keys used. They are used mainly in the following two scenarios:

– secret-key transport. The secret key is encrypted by the public key and decrypted by the private key. Only the holder of the private key can recover the secret key, which guarantees confidentiality;

– signature. The data digest, calculated using a hash function, is encrypted by the private key and decrypted by the public key. The data source is the only holder of the private key, which guarantees the integrity control of the data received, the authentication of the data source, and the non-repudiation by the client.

Table 1.1 shows a comparison of key sizes depending on whether symmetrical and asymmetrical algorithms are used to obtain an equivalent level of security.

Symmetrical algorithm	Asymmetrical algorithm Modular exponentiation	Asymmetrical algorithm Elliptical curve
80	1,024	160
112	2,048	224
128	3,072	256
192	7,680	384
256	15,360	512

Table 1.1. *Comparison of key size depending on algorithm*

1.1.1. *3DES algorithm*

The triple data encryption standard (3DES) algorithm is a symmetrical algorithm by blocks. It strings three successive operations of the DES algorithm on a single 64-bit data block.

Three keys (key1, key2 and key3) are used for encryption, one key per operation. Each 64-bit key contains 56 randomly generated bits and 8 odd parity check bits. Two options are defined for the composition of keys:

– option 1: the three keys (key1, key2 and key3) are different;

– option 2: two keys (key1 and key2) are different; two keys (key1 and key3) are identical.

The 3DES algorithm encryption operation consists of sequencing a DES encryption (E) with the key1, a DES decryption (D) with the key2, and a DES encryption (E) with the key3.

$$\text{Output data} = E_{key3}(D_{key2}(E_{key1}(\text{input data}))).$$

The DES encryption algorithm is shown in Figure 1.1.

Each 64-bit data block is submitted to an initial permutation function. The resulting 64-bit block is then cut into two 32-bit blocks (left block L_0 and right block R_0).

The DES algorithm is constructed using 16 successive iterations. The first iteration generates, from the two blocks L_0 and R_0, two blocks L_1 and R_1 in the following manner:

$$L_1 = R_0$$

$$R_1 = L_0 \oplus f(R_0, K_1)$$

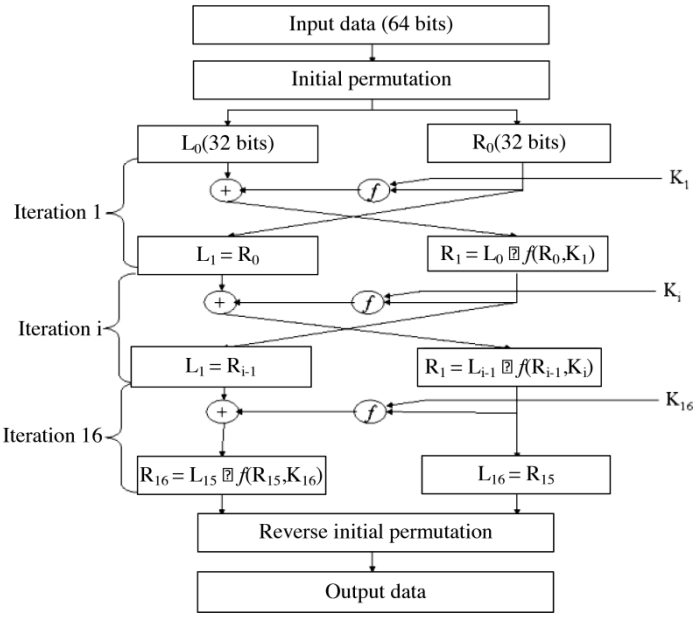


Figure 1.1. *DES algorithm*

At each iteration, the same operation is executed:

– left block L_i corresponds to right block R_{i-1};

– right block R_i is an exclusive OR of left block L_{i-1} and of the result of an *f* function applied to right block R_{i-1} and to K_i deducted from the initial 64-bit key (key1, key2 or key3).

The output of the last iteration is composed of left block L_{16} and right block R_{16}. Block $R_{16}L_{16}$ is subjected to a reverse initial permutation.

The *f* function is described in Figure 1.2.

The E function is used to generate 48 bits from right block R. The exclusive OR operation is applied to the previous result and to key K.

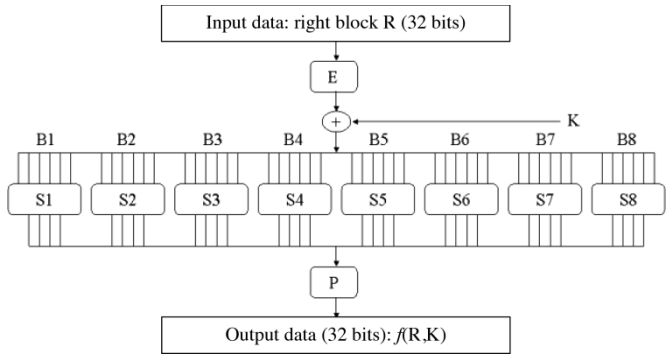


Figure 1.2. *f function*

The 48 resulting bits are cut into eight blocks (B1 to B8) of six bits, to which substitution functions are applied (S1 to S8), resulting in an output of a four-bit block for each substitution function.

The permutation function P operates on the 32 bits produced by the eight substitution functions S1 to S8.

The *f* function is written as follows:

$$f(K,R) = P(S1(B1)S2(B2)...S8(B8)), \text{ with } B1B2...B8 = K \oplus E(R)$$

The generation of keys K_i is shown in Figure 1.3.

A permutation function P1 is applied to the 64 bits of the initial key (key1, key2, or key3). The resulting 56 bits are cut into two blocks C_0 and D_0.

Blocks C_i and D_i are deduced from the previous blocks C_{i-1} and D_{i-1} by creating a shift to the left of one or two bits depending on the number of the block.

A P2 permutation is applied to this C_iD_i block to obtain the 16 48-bit keys K_i used for the 16 iterations.

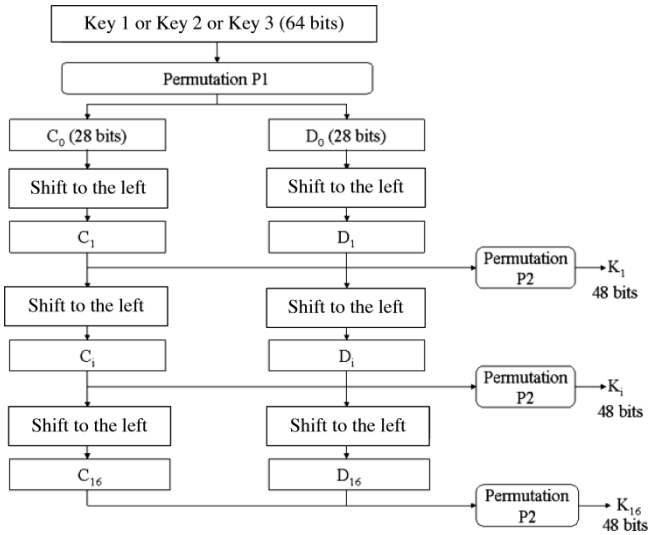


Figure 1.3. *Generation of keys K_i*

1.1.2. *AES algorithm*

The advanced encryption standard (AES) algorithm is a symmetrical block algorithm. It operates on 128-bit data blocks and uses keys 128 bits, 192 bits, or 256 bits in length.

The 16-byte data input block is represented by a four-line by four-column table, with each cell containing a byte.

Input data: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

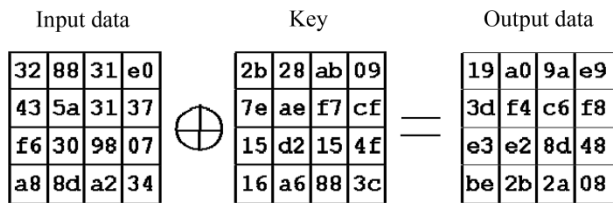
32	88	31	e0
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

The 16-byte key is represented by a four-line by four-column table, with each cell containing a byte.

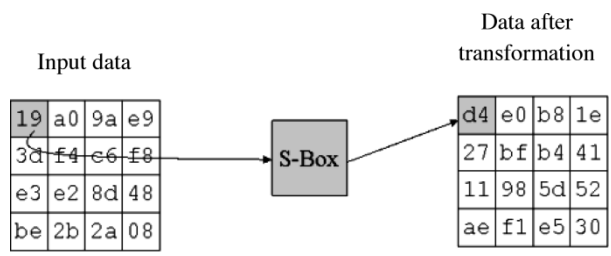
Initial 128-bit key: `2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c`

2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

An exclusive OR is executed between the input data and the initial key. The output data makes up the entry data of the first iteration.



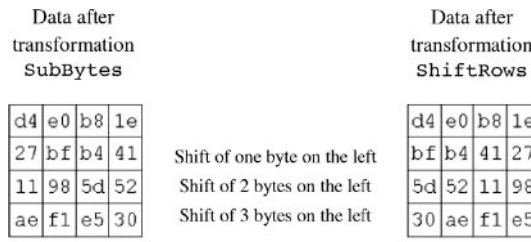
The `SubBytes` transformation applies the S-box substitution table to each byte of the input data table of the iteration. For example, if the byte has a hexadecimal value of 19, the substitution value corresponds to the value of the first line and ninth column in the S-box table.



Iteration 1: transformation `SubBytes`

The `ShiftRows` transformation consists of applying a shift to each line for the three last lines of the table resulting from the `SubBytes` transformation, with the first line being completely copied without modification. The second line

undergoes a shift of one byte on the left; the third line a shift of two bytes; and the fourth line a shift of three bytes.



Iteration 1: transformation `ShiftRows`

The `MixColumns` transformation operates on the columns of the table resulting from the `ShiftRows` transformation.

The transformation of the first column is executed as follows:

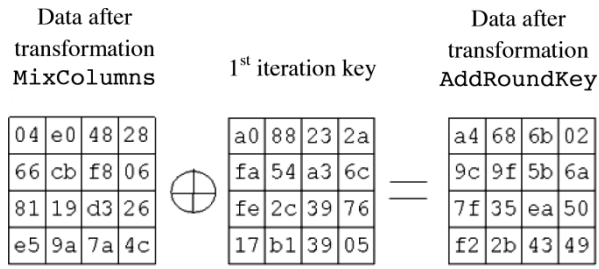
$$\{04\} = (\{02\} \bullet \{d4\}) \oplus (\{03\} \bullet \{bf\}) \oplus \{5d\} \oplus \{30\}$$
$$\{66\} = \{d4\} \oplus (\{02\} \bullet \{bf\}) \oplus (\{03\} \bullet \{5d\}) \oplus \{30\}$$
$$\{81\} = \{d4\} \oplus \{bf\} \oplus (\{02\} \bullet \{5d\}) \oplus (\{03\} \bullet \{30\})$$
$$\{e5\} = (\{03\} \bullet \{d4\}) \oplus \{bf\} \oplus \{5d\} \oplus (\{02\} \bullet \{30\})$$

The symbol • represents a multiplication of bytes in the form of polynomials, modulo the first polynomial $x^8 + x^4 + x^3 + x + 1$.

Iteration 1: transformation `MixColumns`

The first iteration is ended by an exclusive OR (`AddRoundKey` transformation) between the output data of

the MixColumns transformation and the key of the first iteration. This key is derived from the initial key.



Iteration 1: transformation AddRoundKey

The output data of the first iteration constitutes the input data of the second iteration. The number of iterations is equal to 10 for a 128-bit key, to 12 for a 192-bit key, and to 14 for a 256-bit key.

Table 1.2 shows the derivation of the initial key for the first iteration.

Key derivation starts with the cutting of the initial 128-bit key into four blocks, w_0, w_1, w_2 and w_3.

w_0 = 2b7e1516

w_1 = 28aed2a6

w_2 = abf71588

w_3 = 09cf4f3c

The key of the first iteration is made up of four blocks, w_4, w_5, w_6 and w_7. The RotWord transformation consists of the shift of a byte on the left. The SubWord transformation applies the S-box substitution table to each byte. Block w_i is the result of an exclusive OR between blocks w_{i-1} (having possibly undergone transformations) and w_{i-4}.

i	4	5	6	7
w_{i-1}	09cf4f3c	a0fafe17	88542cb1	23a33939
After RotWord	cf4f3c09			
After SubWord	8a84eb01			
Rcon	01000000			
After exclusive OU	8b84eb01			
w_{i-4}	2b7e1516	28aed2a6	abf71588	09cf4f3c
w_i	a0fafe17	88542cb1	23a33939	2a6c7605

Table 1.2. *Derivation of the initial key for the first iteration*

1.1.3. *RSA algorithm*

The RSA algorithm (named after the initials of its three inventors, Rivest, Shamir and Adleman) is a public key asymmetrical algorithm based on modular exponentiation.

Alice wants to receive a block of confidential data M from Bob. Alice generates a public key that she transmits to Bob and a private key that she keeps. Bob encrypts the data block M with the public key. Only Alice can recover the data block M using her private key.

The RSA algorithm is implemented using the following procedure:

– Alice generates two prime numbers, p and q: $p = 59$ and $q = 71$;

– Alice calculates the product $w = (p-1)(q-1)$: $w = 58 \times 70 = 4060$;

– Alice chooses a prime number e with w: $e = 671$. It is relatively easy to determine the value e using Euclid's algorithm to verify whether it is prime or not with w;

– Alice calculates the product $n = p \times q$: $n = 59 \times 71 = 4189$;

– Alice calculates the number d such that $d \times e \equiv 1 \ [mod\ w]$: $d = 1791$. The calculation of the value d uses an extended Euclidean algorithm, or Bézout algorithm. Since e is prime with w, there are two integer numbers d and k such that $(e \times d) + (k \times w) = 1$ (Bachet–Bézout theorem);

– Alice transmits to Bob the values n and e, which constitute the public key, and keeps the values n and d, which represent the private key;

– Bob encrypts his message M with the public key: $M \mapsto M^{671}$ [mod 4189]M = 01010010 01010011 01000001 M is cut into blocks of length k such that $2^k < n < 2^{k+1}$ M is therefore cut into blocks of 12 bits: M = 010100100101 001101000001The 12 bits 010100100101 converted into decimals give 1317. The 12 bits 001101000001 converted into decimals give 833. The encrypted message C transmitted by Bob is as follows:

- Bob calculates 1317^{671} [mod 4189], or 3530,

- Bob calculates 833^{671} [mod 4189], or 3050;

– Alice decrypts the received message C with her private key: $C \mapsto C^{1791}$ [mod 4189]:

- Alice calculates 3530^{1791} [mod 4189] and recovers the value 1317,

- Alice calculates 3050^{1791} [mod 4189] and recovers the value 833.

The difficulty of the RSA algorithm lies in the determination by Alice of two prime numbers p and q and in the calculation of M^e [mod n] and C^d [mod n]. The generation of large prime numbers constitutes a relatively complex problem. There are various methods for the calculation of modular exponentiation.

1.1.4. *ECC algorithm*

The Elliptic Curve Cryptography (ECC) algorithm is a public key asymmetrical algorithm based on elliptic curves.

An elliptic curve is a set of points (x,y) satisfying the conditions below:

$$y^2 = x^3 + ax + b$$

with $4\,a^3 + 27\,b^2$ different than zero

When E is an elliptic curve defined by values a and b, we can assign a switching operator + to this set. We define the sum of two points $(A, B) = A + B = C'$ on an elliptic curve as follows:

– a secant line passing through the two points A and B cuts the curve at a third point C;

– the point C' is the symmetric of C in relation to the axis of abscissas;

– if the two points A and B are identical, we consider the line connecting them is the tangent to the elliptic curve;

– if the line is parallel to the access of ordinates (the two points are P and $-P$), the point of intersection is considered as being infinite. This point is represented by point O such that $P + O = P$ or $P + -P = O$.

The procedure takes place as follows:

– Alice receives from Bob the public key composed of the values of a, b, P, and (d_bP):

- a and b are the values of the elliptic curve,

- P is a point on the elliptic curve,

- d_b is Bob's private key,

- (d_bP) is the point on the elliptic curve resulting from the sum of $d_b + P$;

– Alice chooses a secret number n;

– Alice sends Bob the two points nP and $[M + n(d_bP)]$, with M being the data block to be transmitted;

– Bob calculates $d_b(nP) = n(d_bP)$, then $[M + n(d_bP)] - n(d_bP) = M$, which enables him to recover the data block M.

1.2. Hash function

The hash function is the mechanism that converts a string of bytes (the data being transmitted) into a smaller string called a digest. When the data to be transmitted is associated with a secret key, the hash function enables the sealing of this data to provide authentication of the data source and integrity control of the data. The non-repudiation service is not provided.

1.2.1. *MD5 algorithm*

The Message Digest 5 (MD5) algorithm calculates a 128-bit digest from a block of data cut into 512-bit data blocks.

The first stage consists of filling the data block. The data block is completed so as to be congruent to 448 bits modulo 512. Filling is always executed, even if the message is already congruent to 448 bits modulo 512. The filling value therefore varies from 1 to 512 bits. Filling is executed as follows:

– the final filling bit is set at one;

– the other filling bits are set at zero.

The length of the data block is coded on 64 bits. This field is added to the structure previously defined so as to obtain 512-bit data blocks.

Calculation of the digest is based on registers A, B, C and D, which are 32 bits in length. For the first 512-bit block, these registers are initialized with the following values (initialization vector) in hexadecimals (Figure 1.4):

A = 67452301

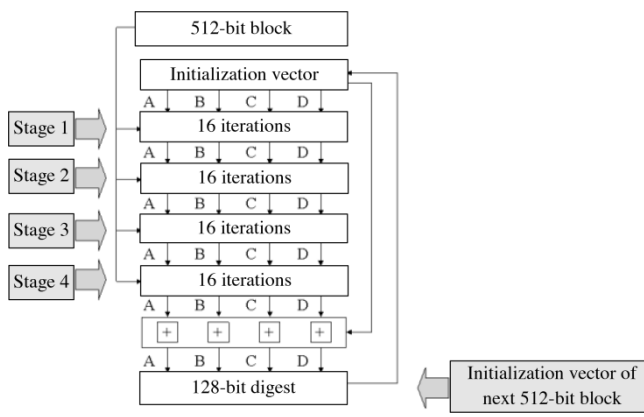
B = EFCDAB89

C = 98BADCFE

D = 10325476

For the following 512-bit blocks, these registers are initialized from the values obtained for the previous 512-bit block. When all of the 512-bit blocks have been processed, the final digest corresponds to the concatenation of the values of registers A, B, C and D (Figure 1.4).

Operations on a 512-bit block are broken down into four stages, which are themselves subdivided into 16 primary iterations based on a function that varies according to stage, on an addition and a shift to the left. When the four stages are finished, the values of registers A, B, C and D are added to those of the initialization vector (Figure 1.4).



Note: The + sign symbolizes an addition of modulo 2^{32}

Figure 1.4. *MD5 algorithm*

The four functions defined are:

Stage one: F(X, Y, Z) = (X AND Y) OR ((NO X) AND Z)

Stage two: F(X, Y, Z) = (X AND Y) OR (Y AND (NO Z))

Stage three: F(X, Y, Z) = X XOR Y XOR Z

Stage four: F(X, Y, Z) = Y XOR (X OR (NO Z))

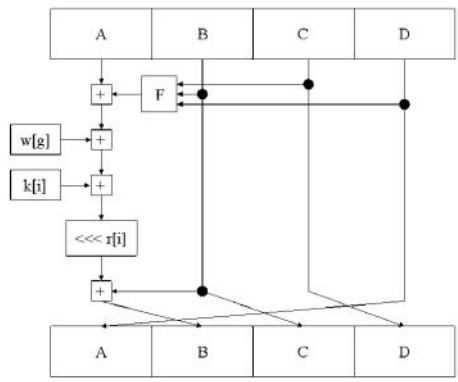
The primary iteration is shown in Figure 1.5. The new values of registers A, B, C and D are supplied, for stage one, by the following equations:

newA = previousD

newB = ((previousA + F(previousB, previousC, previousD) + k[i] + w[g]) <<< r[i]) + previousB.

newC = previousB

newD = previousC



Note: The + sign symbolizes an addition of modulo 2^{32}

Figure 1.5. *Primary iteration of MD5 algorithm*

Each 512-bit data block is subdivided into 16 words w[i] of 32 bits with $0 \leq i \leq 15$. The rank of the stage determines the index value [g]:

$g = i$, with $0 \leq i \leq 15$ for stage one

$g = (5 \times i + 1)$ [mod 16], with $16 \leq i \leq 31$ for stage two

$g = (3 \times i + 5)$ [mod 16], with $32 \leq i \leq 47$ for stage three

$g = (7 \times i)$ [mod 16], with $48 \leq i \leq 63$ for stage four

The symbol <<< indicates that a rotation to the left of r[i] bits must be executed. The value of r[i] is supplied by the following table, for each stage:

r[0..15] = {7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22}

r[16..31] = {5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20}

r[32..47] = {4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23}

r[48..63] = {6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21}

The value of k[i] is determined by the following formula:

$k[i] = E(abs(sin(i + 1)) \times 2^{32})$, where E designates the integer part and (i+1) is expressed in radians.

1.2.2. *SHA algorithm*

The Secure Hash Algorithm (SHA) is associated with multiple hash algorithms: SHA-0, SHA-1 and SHA-2. Algorithm SHA-0 is inspired by the MD5 algorithm. It is not recommended for using for security questions. Algorithm SHA-1 is a modified version of algorithm SHA-0. Algorithm SHA-2 is composed of two algorithms:

– algorithm SHA-256, including versions SHA-224 and SHA-256;

– algorithm SHA-512, including versions SHA-384, SHA-512, SHA-512/256, and SHA-512/224.

1.2.2.1. *SHA-1 algorithm*

The functioning of the SHA-1 algorithm is similar to that of the MD5 algorithm, for which the principal differences introduced are:

– there are 5 registers (A, B, C, D and E) used to produce the 160-bit digest;

– the number of primary iterations increases to 80.

Figure 1.6 shows the primary iteration of the SHA-1 algorithm.

The SHA-1 algorithm uses a Boolean function F among four and changes function every 20 iterations. Likewise, the value of k[i] remains constant for 20 iterations.

The number of words w[g] is equal to 80, one word per iteration. The first 16 words belong to the 512-bit block. The next 64 words are obtained using a rotation applied to the result of an exclusive OR of 4 words calculated during previous iterations.

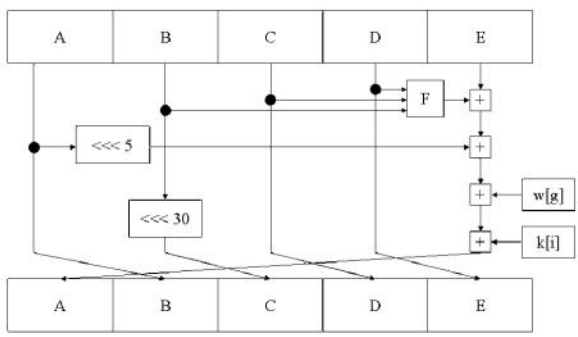


Figure 1.6. *Primary iteration of the SHA-1 algorithm*

1.2.2.2. *SHA-2 algorithm*

Algorithm SHA-226 and its truncated version, SHA-224, are structured on the basis of 32-bit words and a cutting into 512-bit blocks.

Algorithm SHA-512 and its truncated versions, SHA-384, SHA-512/256, and SHA-512/224, use 64-bit words and a cutting into 1024-bit blocks.

The SHA-2 algorithm has the following specific characteristics:

– there are eight registers (A, B, C, D, E, F, G and H). The capacity of each register is 32 bits (as for SHA-1 or MD5) in the case of the SHA-256 algorithm, or 64 bits in the case of the SHA-512 algorithm;

– there are 64 primary iterations in the case of SHA-256 (as for MD5) and 80 in the case of SHA-512 (as for SHA-1);

– the length of the data block is coded at the end of filling on 64 bits in the case of SHA-256 (as for SHA-1 or MD5) and on 128 bits in the case of SHA-512.

The digest of each version is indicated by the last suffix of the algorithm concerned:

– the SHA-256 algorithm produces a 256-bit digest generated by the content of eight registers (A, B, C, D, E, F, G and H);

– the SHA-224 algorithm uses the SHA-256 algorithm and produces a 224-bit digest generated by the content of seven registers (A, B, C, D, E, F and G);

– the SHA-512 algorithm produces a 512-bit digest generated by the content of eight registers (A, B, C, D, E, F, G and H);

– the SHA-384 algorithm uses the SHA-512 algorithm and produces a 384-bit digest generated by the content of six registers (A, B, C, D, E and F);

– the SHA-512/256 algorithm uses the SHA-512 algorithm and produces a 256-bit digest generated by the content of four registers (A, B, C and D;

– the SHA-512/224 algorithm uses the SHA-512 algorithm and produces a 224-bit digest generated by the content of three registers (A, B and C) and the first 32 bits of register D.

Figure 1.7 shows the primary iteration of the SHA-2 algorithm.

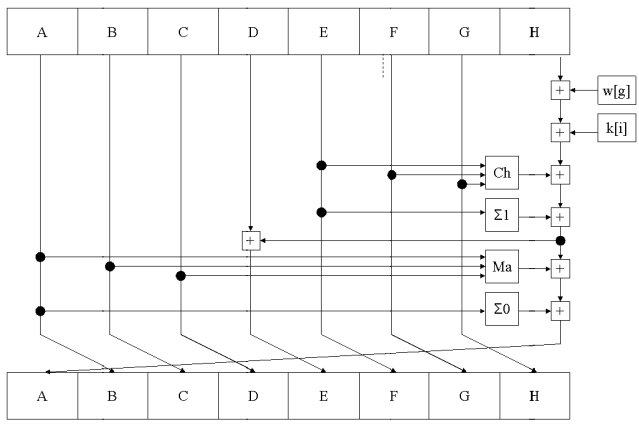


Figure 1.7. *Primary iteration of the SHA-2 algorithm*

The operating principle pertaining to each iteration is identical for all of the algorithms in the SHA-2 family. However, these algorithms differ in the following ways:

– the SHA-256 and SHA-224 algorithms use a specific initialization vector;

– the SHA-512, SHA-384 and SHA-512/256 algorithms use a specific initialization vector;

– the SHA-256 and SHA-224 algorithms, on the one hand, and the SHA-512, SHA-384 and SHA-512/256 algorithms, on the other hand, use Boolean functions (Ch, Ma), rotation operations (Σ0, Σ1), and specific k[i] constants. Likewise, the calculation of the word w[g] is specific for $g \geq 16$.

1.2.3. *HMAC mechanism*

The keyed-Hash Message Authentication Code (HMAC) mechanism defines the association of secret key K and a data block D, used for the calculation of the MAC seal, using a hash function. The digest obtained, which acts as an MAC seal, enables the checking of the authenticity (origin and integrity) of data block D.

The HMAC mechanism is also used to calculate derived keys.

The MAC seal is calculated using the formula below:

$$MAC(D) = HMAC(K, D) = H((K0 \oplus opad) || H((K0 \oplus ipad) || D))$$

H is the hash function (MD5 or SHA).

K0 is the key calculated from secret key K.

Outer padding (opad) is the filling composed of byte 5c (in hexadecimals) repeated B times.

Inner padding (ipad) is the filling composed of bite 36 (in hexadecimals) repeated B times.

B is the size of the data block to which the hash function is applied. If the block size of the hash function is 512 bits, ipad and opad are 64 repetitions of bytes 5c and 36 (in hexadecimals).

The sign || designates the concatenation of bytes.

The calculation of the MAC seal takes place in the following stages:

– Stage 1: the length of key K is equal to B.

K0 = K

Go to stage 4

– Stage 2: the length of key K is greater than B.

Apply the hash function H to key K to obtain a digest of length L.

Complete the digest calculated with zeros to obtain size B.

K0 = H(K) || 00....00

Go to stage 4

– Stage 3: the length of key K is smaller than B.

Complete key K with zeros to obtain size B.

– Stage 4: execute an exclusive OR between key K0 and ipad to obtain the structure K0 ⊕ ipad.

– Stage 5: concatenate data to authenticate D with the structure generated in stage 4 to obtain the structure K0 ⊕ ipad) || D.

– Stage 6: apply hash function H to the structure generated in stage 5 to obtain the structure H((K0 ⊕ ipad) || D).

– Stage 7: execute an exclusive OR between key K0 and opad to obtain the structure K0 ⊕ opad.

– Stage 8: concatenate the structures generated in stages 6 and 7 to obtain the structure (K0 ⊕ opad) || H((K0 ⊕ ipad) || D).

– Stage 9: apply hash function H to the structure generated in stage 8 to obtain the MAC seal enabling authentication of data D.

It is possible to truncate the structure generated in stage 9 to retain a reduced number of left t bits as an MAC seal. For security questions, the number t must be greater than or equal to max(L/2, 80), with L being the length of the digest. The MAC seal is designated in the form of HMAC-H-t. For example, notation HMAC-SHA1-80 designates the calculation of an MAC seal using hash function SHA-1, the digest of which is truncated to 80 bits.

1.3. Key exchange

1.3.1. *Secret-key generation*

The two methods of establishing a secret key are key transport and generation. Key transport generally uses a public key to encrypt the secret key.

The Diffie–Hellman mechanism is used to generate a secret key based on modular exponentiation. The procedure occurs as follows:

– Alice and Bob choose a prime number p (*prime*) with a primitive root g (*generator*) modulo p;

– Alice chooses a secret number a such that $1 \leq a \leq p$-1;

– Bob chooses a secret number b. such that $1 \leq b \leq p$-1;

– Alice calculates the value $A = g^a$ *[mod p]* and sends it to Bob;

– Bob calculates the value $B = g^b$ *[mod p]* and sends it to Alice;

– Alice generates the secret key equal to: $(B\ [mod\ p])^a = B^a\ [mod\ p] = (g^b)^a\ [mod\ p] = g^{ba}\ [mod\ p]$;

– Bob generates the same secret key equal to: $(A\ [mod\ p])^b = A^b\ [mod\ p] = (g^a)^b\ [mod\ p] = g^{ab}\ [mod\ p]$.

A third party can recover the values of p, g, $A = g^a\ [mod\ p]$ and $B = g^b\ [mod\ p]$, which circulate unscrambled in the network. This does not allow the third party to calculate $g^{ab}\ [mod\ p]$ if the numbers (p, a, b) are large enough to avoid an attack through exhaustive search.

Table 1.3 supplies the formula used to calculate the value of the prime number p (*prime*), with the value of g (*generator*) being fixed at 2, for MODP (MODular exponential modulus P) groups.

Group no.	Designation	Value of p
1	768-bit MODP	$p = 2^{768} - 2^{704} - 1 + 2^{64} \times \{ E[2^{638} \times pi] + 149686\}$
2	1024-bit MODP	$p = 2^{1024} - 2^{960} - 1 + 2^{64} \times \{ E[2^{894} \times pi] + 129093 \}$
5	1536-bit MODP	$p = 2^{1536} - 2^{1472} - 1 + 2^{64} \times \{ E[2^{1406} \times pi] + 741804 \}$
14	2048-bit MODP	$p = 2^{2048} - 2^{1984} - 1 + 2^{64} \times \{ E[2^{1918} \times pi] + 741804 \}$
15	3072-bit MODP	$p = 2^{3072} - 2^{3008} - 1 + 2^{64} \times \{ E[2^{2942} \times pi] + 169031 \}$
16	4096-bit MODP	$p = 2^{4096} - 2^{4032} - 1 + 2^{64} \times \{ E[2^{3966} \times pi] + 240904 \}$
17	6144-bit MODP	$p = 2^{6144} - 2^{6080} - 1 + 2^{64} \times \{ E[2^{6014} \times pi] + 929484 \}$
18	8192bit MODP	$p = 2^{8192} - 2^{8128} - 1 + 2^{64} \times \{ E[2^{8062} \times pi] + 4743158 \}$

Table 1.3. *MODP Diffie–Hellmann groups*

A second secret-key generation technique is based on the elliptic curve. The procedure occurs as follows:

– Alice and Bob agree on the two public values a and b of an elliptic curve $y^2 = x^3 + ax + b$;

– Alice and Bob agree on a public point P located on the elliptic curve;

– Alice chooses a point on the curve d_a considered to be Alice's secret;

– Alice sends Bob the point constructed using $(d_aP) = d_a + P$;

– Bob chooses a point on the curve d_b considered to be Bob's secret;

– Bob sends Alice the point constructed using $(d_bP) = d_b + P$;

– Alice calculates point $d_a(d_bP) = d_a + d_b + P$, which constitutes the secret key;

– Bob calculates point $d_b(d_aP) = d_b + d_a + P$, which constitutes the same secret key.

1.3.2. *Public key distribution*

Asymmetrical encryption algorithms are based on the diffusion of the public key by the holder of the private key. Conversely, nothing guarantees the identity of the holder of the private key. The certificate, signed by a trusted authority, enables a public key to be associated with a party. All the holders of the public key possess the trusted authority's public key, which enables them to ensure the certificate's authenticity.

Certificates are managed by a public key infrastructure (PKI), which ensures the following functions:

– the user registration;

– the generation of certificates;

– the renewal of certificates;

– the revocation of certificates;

– the publication of certificates;

– the publication of revocation lists.

PKI is divided into several entities:

– the certification authority, whose mission is to sign Certificate Signing Requests (CSR) and Certificate Revocation Lists (CRL);

– the registration authority, whose mission is to generate certificates and to conduct usage verifications of the identity of the final user;

– the repository authority, whose mission is to store digital certificates and revocation lists;

– the key escrow authority, which provides authorities with the means of decrypting data for a user.

A certificate is composed of two parts:

– the data being certified, which contains the base fields and extensions;

– the certificate signature.

The base fields of a certificate provide the following information:

– the structure version of the certificate (value = 3);

– the series number;

– the information about the certificate signature (algorithms and parameters);

– the name of the certificate issuer;

– the validity period of the certificate;

– the name of the certificate holder;

– the public key (value of the public key, algorithm and parameters).

Certificate extensions enable a more precise specification of the following characteristics:

– information about keys;

– information on certification policies;

– additional information on the issuer and holder of the certificate;

– limitations on the certification process.