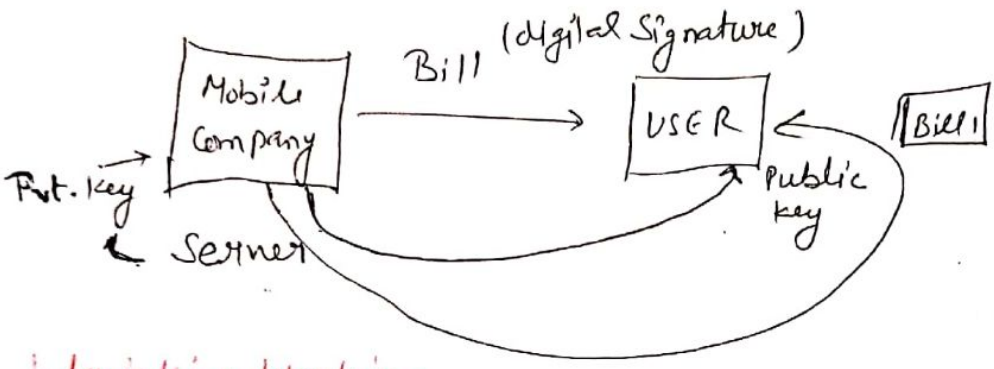
Digital Signature [E-commerce]

Signature :- proof of identity (Is it from correct sender/not)

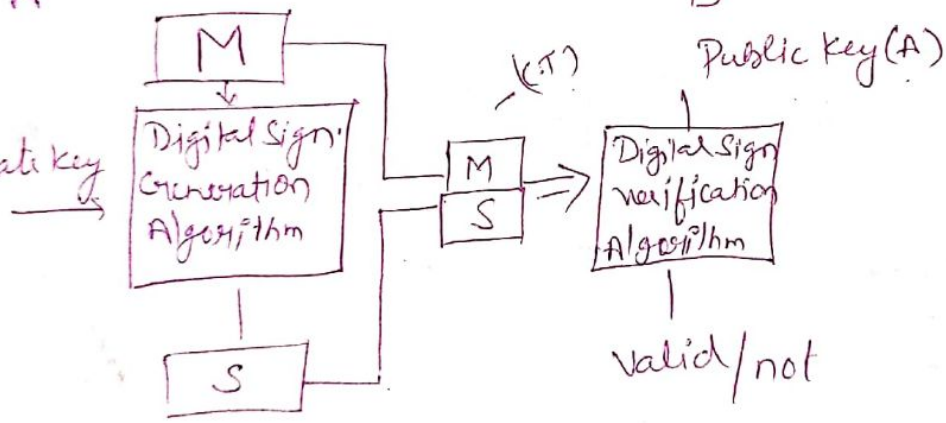
↳ Asymmetric key cryptographic

↳ Encryption with private key and Decryption with public key

↳ used for Authentication (correct person) and Non-Repudiation (Cannot deny)



Working



Sender
↓
Plaintext
↓
Encrypt with Sender's Pvt. key
↓
C.T — N/W
↓
C.T
↓
Decrypt with Sender's Public key
↓
Plaintext
↓
Receiver

ELGAMAL Digital Signature

- digital signature Scheme
- Encryption – Public key
- Decryption – Private key

Working :- (1) Select a Prime number (q)

(2) Select a primitive root (α) of q.

3. Generate a random integer (X_A)

$$1 < X_A < q-1$$

(4) Compute $Y_A = (\alpha)^{X_A} \bmod q$

(5) Generate keys for user (A)

Private key $\rightarrow X_A$

Public key $\Rightarrow \{q, \alpha, Y_A\}$

(6) Generate hash code (m) for the plain text (M)

$$m = H(M) \quad 0 \leq m \leq q-1$$

(7) Generate a random integer K

$$1 \leq K \leq q-1 \quad \text{and} \quad \gcd(K, q-1) = 1$$

(8) Now calculate S_1 and S_2 $\quad S_1 = \alpha^K \bmod q$

$$S_2 = K^{-1}(m - X_A S_1) \bmod q-1$$

(9) Now we got the signature pair (S_1, S_2)

Now at user B's side,

calculate V_1 and V_2

$$V_1 = \alpha^m \bmod q$$

$$V_2 = (Y_A)^{S_1} \cdot (S_1)^{S_2} \bmod q$$

if $V_1 = V_2 \rightarrow$ Signature is valid

if $V_1 \neq V_2 \Rightarrow$ Not valid

Example:

Let $q = 19$ and $\alpha = 10$

Now, Random integer X_A $(1 < X_A < q-1)$

$1 < X_A < 18$

$\Rightarrow$ $\boxed{X_A = 16}$

$$Y_A = \alpha^{X_A} \bmod q = (10)^{16} \bmod 19$$

$$= 4$$

$\boxed{Y_A = 4}$

Sender

(5) Keys:- private key $\Rightarrow X_A \Rightarrow 16$

Public key $\Rightarrow \{q, \alpha, Y_A\} = (19, 10, 4)$

Now generate hash code (m)

(6) $m = h(M)$ $\quad 0 \le m \le q-1$

$0 \le m \le 18$

$\therefore$ $\boxed{m = 14}$

(7) Generate (K) $\quad 0 \le K \le q-1$ and $\gcd(K, q-1) = 1$

$0 \le K \le 18$ and $\gcd(K, 18) = 1$

$\therefore$ $\boxed{K = 5}$

Calculate $S_1 = \alpha^K \bmod q = (10)^5 \bmod 19 = 3$

$\boxed{S_1 = 3}$

$$S_2 = k^{-1}(m - X_A S_1) \bmod q-1$$

$k^{-1} \Rightarrow K^{-1} \bmod q-1$

$\Rightarrow 5^{-1} \bmod q-1$

$\Rightarrow 5 \times ? = 1 \pmod{18}$

$\frac{5 \times ?}{18} = 1$

$\frac{5 \times 11}{18} = 1$

$\therefore \boxed{k^{-1} = 11}$

$S_2 = k^{-1}(m - X_A S_1) \bmod q-1$

$= 11(14 - 16 \times 3) \bmod 18$

$11(14 - 48) \bmod 18$

$= -374 \bmod 18$

$\therefore \boxed{S_2 = 4}$

$\therefore S_1, S_2 = (3, 4)$

At B's end:

$v_1 = \alpha^m \bmod q \Rightarrow = 10^{14} \bmod 19 = 16$

$\therefore \boxed{v_1 = 16}$

$v_2 = (Y_A)^{S_1} \cdot (S_1)^{S_2} \bmod q$

$4^3 \times 3^4 \bmod 19$

$= 64 \times 81 \bmod 19$

$= 5184 \bmod 19$

$\boxed{v_2 = 16}$

Now, $v_1 = v_2$

$\therefore$ signature is valid

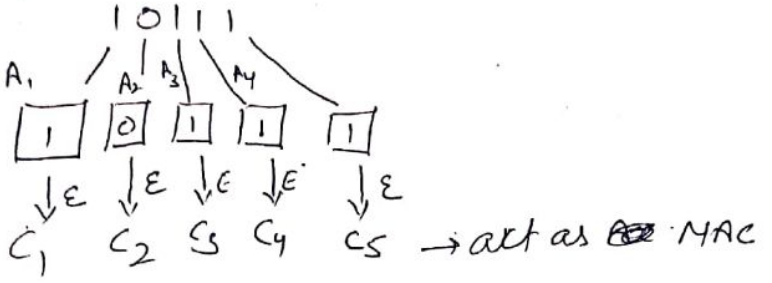
Cipher Based MAC (CMAC)

It has message size limit

→ Based on block cipher

↳ given message is divided into equal no. of blocks and each block is encrypted separately

Example:-



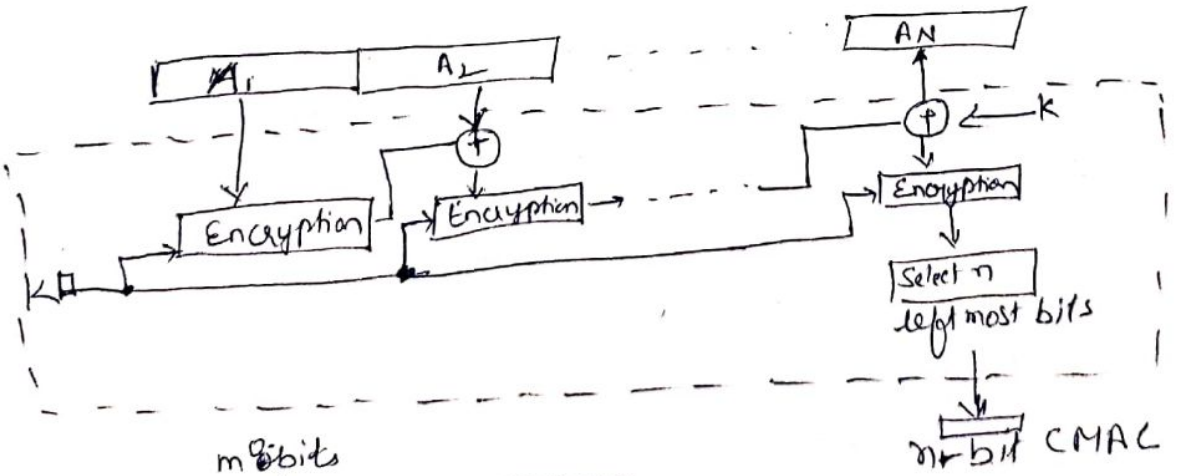
$C_1 = E(K, A_1)$

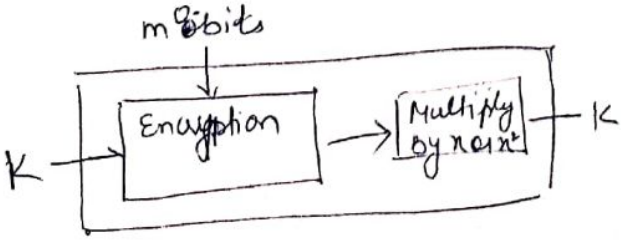
$C_2 = E(K, (A_2 \oplus C_1))$

$C_3 = E(K, [A_3 \oplus C_2])$

$\vdots$

$C_n = E(K, (A_n \oplus C_{n-1}))$





Schnorr Digital Signature Scheme

① The scheme is based on using a prime modulus p, p-1 having a prime factor q of ~~appropriate~~ appropriate size; that is $p-1 \equiv (\text{mod } q)$

We use $p = 2^{1024}$ and $q = 2^{160}$

p is a 1024 bit number, q is a 160-bit number, which

Step 1.

Key Generation:-

1. Choose primes p and q such that q is prime factor of p-1
2. Choose an integer a such that $a^q \equiv 1 \bmod p$. The values a, p and q comprise a global public key that can be common to group of users.
3. Choose a random integer s with $0 < s < q$. This is the user's private key.
4. Calculate $v = a^{-s} \bmod p$. This is the user's public key

Step 2:-

Signature Generation:-

1. Choose a random integer r with $0 < r < q$ and compute $x = a^r \bmod p$. This computation is a preprocessing stage independent of the message M to be signed.
2. Concatenate the message with x and Hash the result to

Schnorr Digital Signature Scheme

① The scheme is based on using a prime modulus p, $p-1$ having a prime factor q of ~~appropriate~~ appropriate size; that is $p-1 \equiv (\text{mod } q)$

We use $p = 2^{1024}$ and $q = 2^{160}$

p is a 1024 bit number, q is a 160-bit number, which

Step 1.

Key Generation :-

1. Choose primes p and q such that q is prime factor of $p-1$
2. Choose an integer a such that $a^q \equiv 1 \bmod p$. The values a, p and q comprise a global public key that can be common to group of users.
3. Choose a random integer s with $0 < s < q$. This is the user's private key.
4. Calculate $v = a^{-s} \bmod p$. This is the user's public key

Step 2:-

Signature Generation :-

1. Choose a random integer r with $0 < r < q$ and compute $x = a^r \bmod p$. This computation is a preprocessing stage independent of the message M to be signed.
2. Concatenate the message with x and hash the result to

Compute the value e:-

$$e = H(M \| r)$$

3. Compute $y = (r + se) \bmod q$

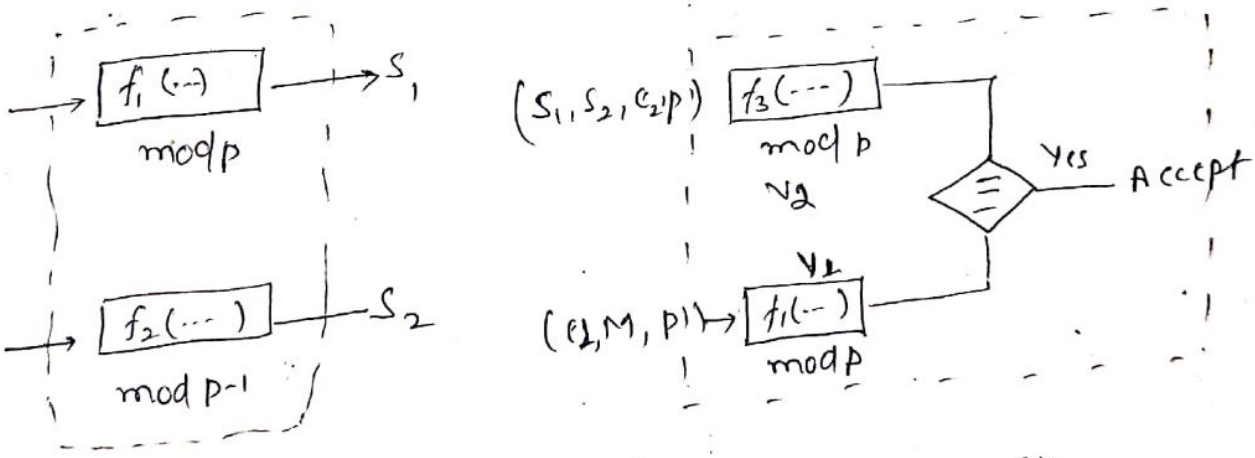
The signature consists of the pair (e, y)

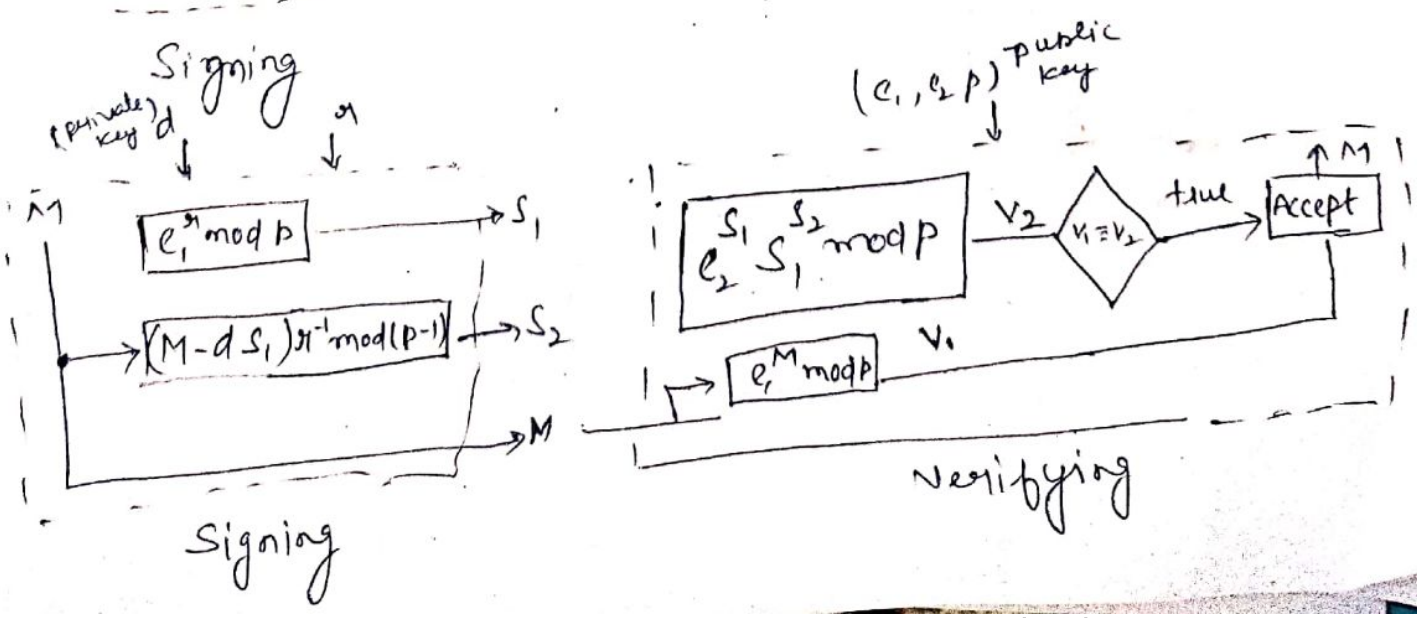
Step3 Signature Verification

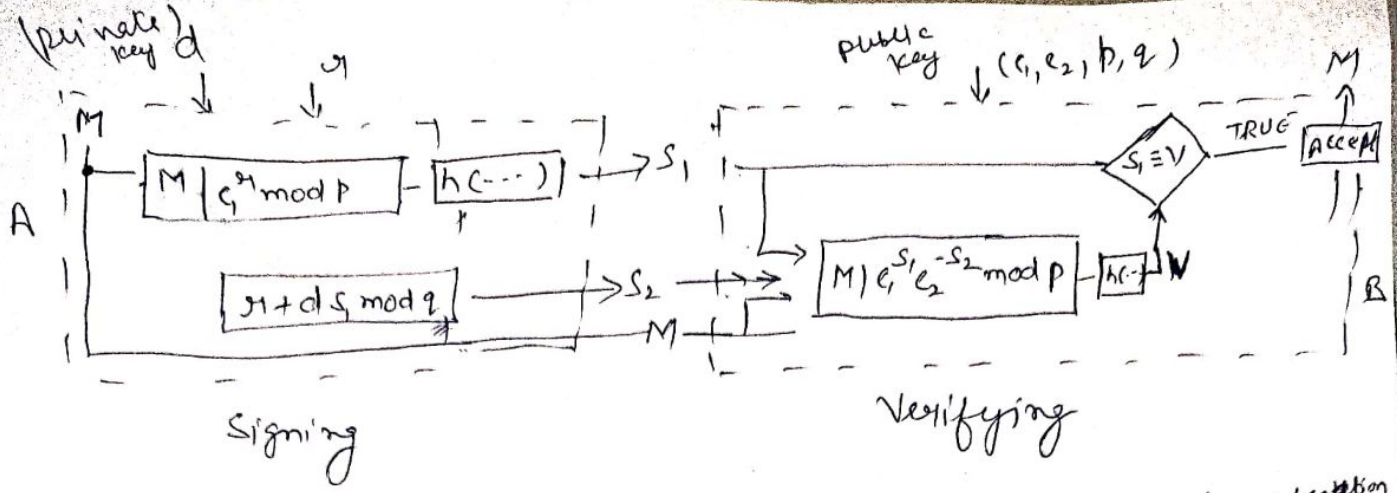
① Compute $x' = a^y v^e \bmod p$.

② Verify that $e = H(M \| x')$. To see that the verification works, observe that

ELGAMAL Digital signature







M:- Message

S_1 S_2 :- Signature

V: Verification

r: Random secret

(d) user's private key

(c_1, c_2, p, q): user's public key

|: ~~Concats~~ |: concatenation

h(---): Hash algorithm

Schnorr Digital Signature Step:

Signing

1. Alice choose a random number r.
2. Alice calculates $S_1 = h(M \mid c_1^r \bmod p)$
3. Alice calculates $S_2 = r + d * S_1 \bmod q$
4. Alice sends M, S_1 and S_2

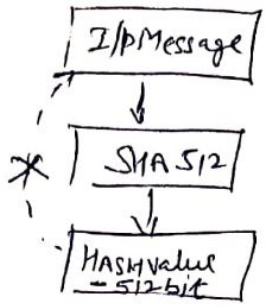
Verifying Message

(5) Bob calculates $V = h(M \mid c_1^{S_2} c_2^{-S_1} \bmod p)$.

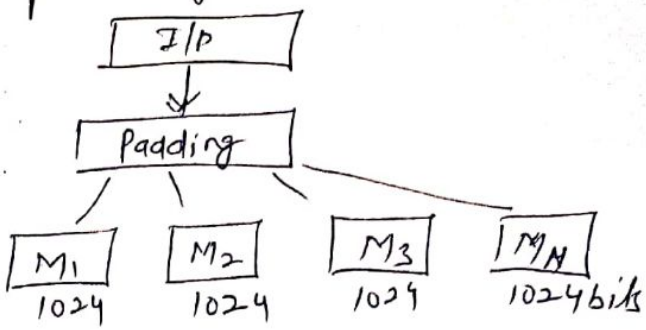
(6) if S_1 is congruent to V modulo p, the message is accepted;

otherwise rejected.

Overview SHA 512



Input Message



Padding Example:-

Consider Input Message "abc" (ASCII code a-97, b-98, c-99)

Represent in binary

01100001 01100010 01100011 1 871 zero

Message_Length = 24 bits

24 ≡ 896 mod 1024

24 mod 1024 ≡ 896

24 ≠ 896

9 1 1 4
1024
-128
896

Needed

Message_Length ≡ 896 mod 1024

Message_length mod 1024 = 896

24 + 872 mod 1024 = 896

↳ Pad 872 bit to message such that Message_length mod 1024 = 896

872 bits to be padded - 1 bit followed by 871 zeros

01100001 01100010 01100011 1000 (871 zeros)

6 1 6 2 6 3 8

Pad the original length of the message for 128 bits at the end

01100001 01100010 01100011

Message_length = 24 bits

Hexadecimal value of 24 is 18

16 | 24
1 | 8
(18)

0000000000000000 0000000000000018

How many bits you will pad for input message length of 2348 bits?

Messag. length $\equiv 896 \bmod 1024$

$2348 \bmod 1024 = 300 + 596$

$2348 + 596 \bmod 1024 = 896$

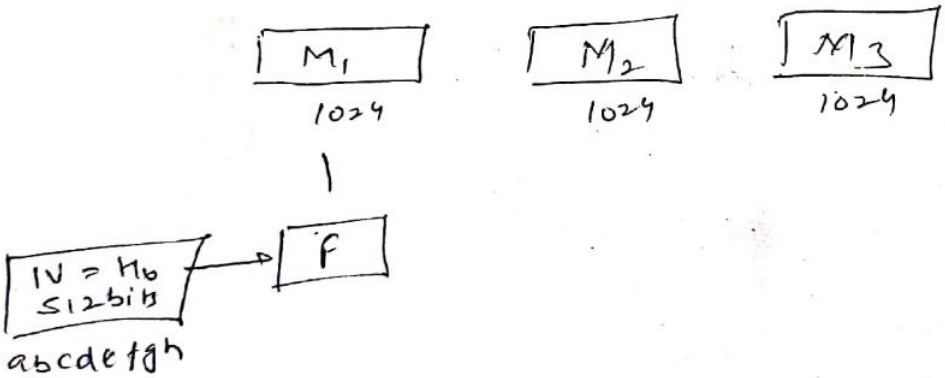
pad 596 bits where in 1 followed by 595 zeros

Message length is 2944

Add the message-length (2348) as 128 bits at the end

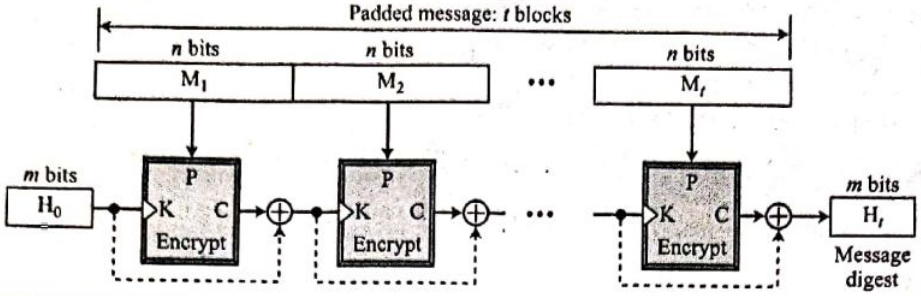
Total bits will be $2944 + 128 = 3072 / 1024 = 3$

This is nothing but three M blocks of size 1024 bits



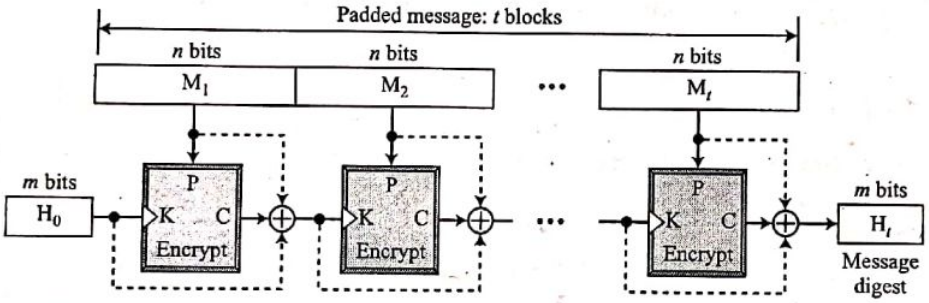
example, AES is a good candidate for this purpose. Figure 12.4 shows the Matyas-Meyer-Oseas scheme.

Figure 12.4 *Matyas-Meyer-Oseas scheme*



Miyaguchi-Preneel Scheme The **Miyaguchi-Preneel scheme** is an extended version of Matyas-Meyer-Oseas. To make the algorithm stronger against attack, the plaintext, the cipher key, and the ciphertext are all exclusive-ored together to create the new digest. This is the scheme used by the Whirlpool hash function. Figure 12.5 shows the Miyaguchi-Preneel scheme.

Figure 12.5 *Miyaguchi-Preneel scheme*



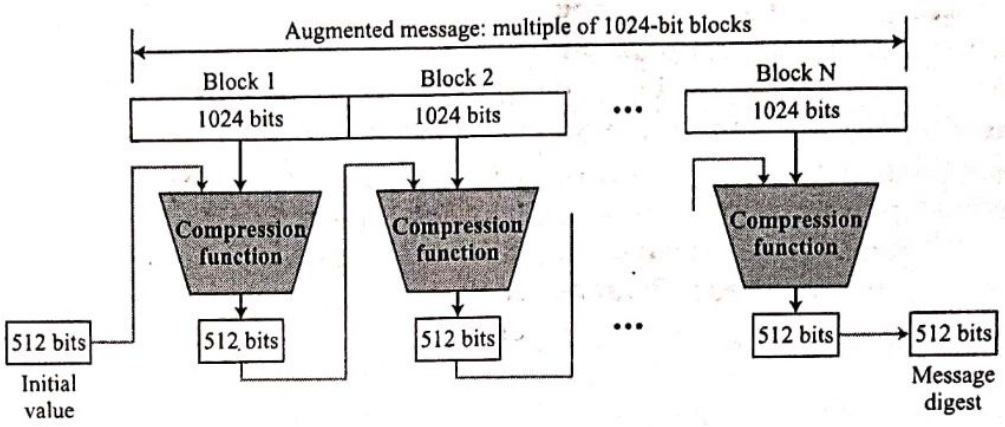
12.2 SHA-512

SHA-512 is the version of SHA with a 512-bit message digest. This version, like the others in the SHA family of algorithms, is based on the Merkle-Damgard scheme. We have chosen this particular version for discussion because it is the latest version, it has a more complex structure than the others, and its message digest is the longest. Once the structure of this version is understood, it should not be difficult to understand the structures of the other versions. For characteristics of SHA-512 see Table 12.1.

Introduction

SHA-512 creates a digest of 512 bits from a multiple-block message. Each block is 1024 bits in length, as shown in Figure 12.6.

Figure 12.6 *Message digest creation SHA-512*



The digest is initialized to a predetermined value of 512 bits. The algorithm mixes this initial value with the first block of the message to create the first intermediate message digest of 512 bits. This digest is then mixed with the second block to create the second intermediate digest. Finally, the $(N - 1)$th digest is mixed with the Nth block to create the Nth digest. When the last block is processed, the resulting digest is the message digest for the entire message.

Message Preparation

SHA-512 insists that the length of the original message be less than 2^{128} bits. This means that if the length of a message is equal to or greater than 2^{128}, it will not be processed by SHA-512. This is not usually a problem because 2^{128} bits is probably larger than the total storage capacity of any system.

SHA-512 creates a 512-bit message digest out of a message less than 2^{128}.

Example 12.1

This example shows that the message length limitation of SHA-512 is not a serious problem. Suppose we need to send a message that is 2^{128} bits in length. How long does it take for a communications network with a data rate of 2^{64} bits per second to send this message?

Solution

A communications network that can send 2^{64} bits per second is not yet available. Even if it were, it would take many years to send this message. This tells us that we do not need to worry about the SHA-512 message length restriction.

Example 12.2

This example also concerns the message length in SHA-512. How many pages are occupied by a message of 2^{128} bits?

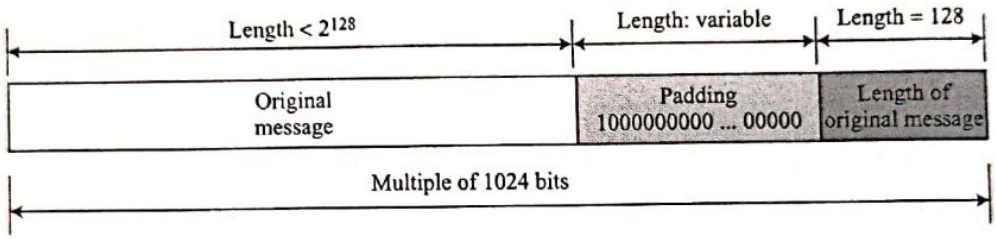
Solution

Suppose that a character is 64, or 2^6, bits. Each page is less than 2048, or approximately 2^{12}, characters. So 2^{128} bits need at least $2^{128} / 2^{18}$, or 2^{110}, pages. This again shows that we need not worry about the message length restriction.

Length Field and Padding

Before the message digest can be created, SHA-512 requires the addition of a 128-bit unsigned-integer length field to the message that defines the length of the message in bits. This is the length of the original message before padding. An unsigned integer field of 128 bits can define a number between 0 and $2^{128} - 1$, which is the maximum length of the message allowed in SHA-512. The length field defines the length of the original message before adding the length field or the padding (Figure 12.7).

Figure 12.7 *Padding and length field in SHA-512*



Before the addition of the length field, we need to pad the original message to make the length a multiple of 1024. We reserve 128 bits for the length field, as shown in Figure 12.7. The length of the padding field can be calculated as follows. Let |M| be the length of the original message and |P| be the length of the padding field.

$$(|M| + |P| + 128) = 0 \bmod 1024 \quad \rightarrow \quad |P| = (-|M| - 128) \bmod 1024$$

The format of the padding is one 1 followed by the necessary number of 0s.

Example 12.3

What is the number of padding bits if the length of the original message is 2590 bits?

Solution

We can calculate the number of padding bits as follows:

$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

The padding consists of one 1 followed by 353 0's.

Example 12.4

Do we need padding if the length of the original message is already a multiple of 1024 bits?

Solution

Yes we do, because we need to add the length field. So padding is needed to make the new block a multiple of 1024 bits.

Example 12.5

What is the minimum and maximum number of padding bits that can be added to a message?

Solution

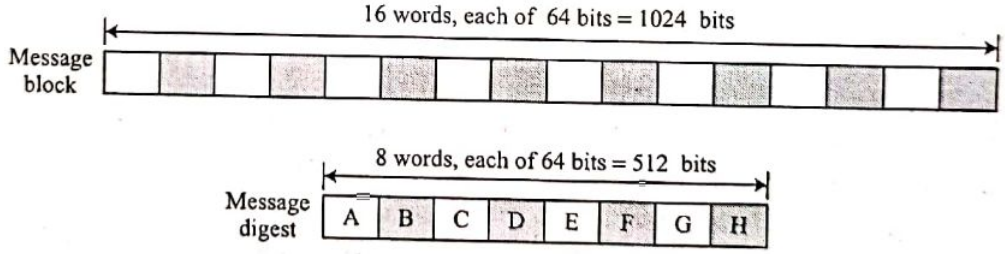
a. The minimum length of padding is 0 and it happens when $(-M - 128) \bmod 1024$ is 0. This means that $|M| = -128 \bmod 1024 = 896 \bmod 1024$ bits. In other words, the last block in the original message is 896 bits. We add a 128-bit length field to make the block complete.

b. The maximum length of padding is 1023 and it happens when $(-|M| - 128) = 1023 \bmod 1024$. This means that the length of the original message is $|M| = (-128 - 1023) \bmod 1024$ or the length is $|M| = 897 \bmod 1024$. In this case, we cannot just add the length field because the length of the last block exceeds one bit more than 1024. So we need to add 897 bits to complete this block and create a second block of 896 bits. Now the length can be added to make this block complete.

Words

SHA-512 operates on words; it is **word oriented.** A word is defined as 64 bits. This means that, after the padding and the length field are added to the message, each block of the message consists of sixteen 64-bit words. The message digest is also made of 64-bit words, but the message digest is only eight words and the words are named A, B, C, D, E, F, G, and H, as shown in Figure 12.8.

Figure 12.8 *A message block and the digest as words*

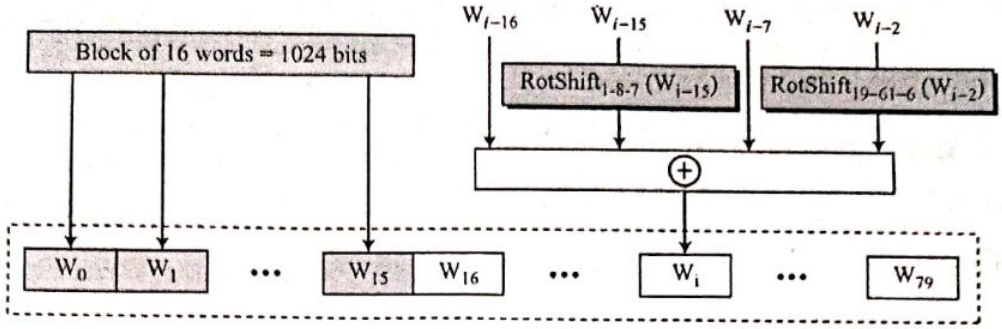


SHA-252 is word-oriented. Each block is 16 words; the digest is only 8 words.

Word Expansion

Before processing, each message block must be expanded. A block is made of 1024 bits, or sixteen 64-bit words. As we will see later, we need 80 words in the processing phase. So the 16-word block needs to be expanded to 80 words, from W_0 to W_{79}. Figure 12.9 shows the **word-expansion** process. The 1024-bit block becomes the first 16 words; the rest of the words come from already-made words according to the operation shown in the figure.

Figure 12.9 *Word expansion in SHA-512*



RotShift$_{l\text{-}m\text{-}n}$ (x): $RotR_l(x) \oplus RotR_m(x) \oplus ShL_n(x)$

$RotR_i(x)$: Right-rotation of the argument x by i bits
$ShL_i(x)$: Shift-left of the argument x by i bits and padding the left by 0's.

Example 12.6

Show how W_{60} is made.

Solution

Each word in the range W_{16} to W_{79} is made from four previously-made words. W_{60} is made as

$$W_{60} = W_{44} \oplus \text{RotShift}_{1\text{-}8\text{-}7}(W_{45}) \oplus W_{53} \oplus \text{RotShift}_{19\text{-}61\text{-}6}(W_{58})$$

Message Digest Initialization

The algorithm uses eight constants for message digest initialization. We call these constants A_0 to H_0 to match with the word naming used for the digest. Table 12.2 shows the value of these constants.

Table 12.2 *Values of constants in message digest initialization of SHA-512*

Buffer	*Value (in hexadecimal)*	*Buffer*	*Value (in hexadecimal)*
A_0	6A09E667F3BCC908	E_0	510E527FADE682D1
B_0	BB67AE8584CAA73B	F_0	9B05688C2B3E6C1F
C_0	3C6EF372EF94F82B	G_0	1F83D9ABFB41BD6B
D_0	A54FE53A5F1D36F1	H_0	5BE0CD19137E2179

The reader may wonder where these values come from. The values are calculated from the first eight prime numbers (2, 3, 5, 7, 11, 13, 17, and 19). Each value is the fraction part of the square root of the corresponding prime number after converting to binary and keeping only the first 64 bits. For example, the eighth prime is 19, with the square root $(19)^{1/2} = 4.35889894354$. Converting the number to binary with only 64 bits in the fraction part, we get

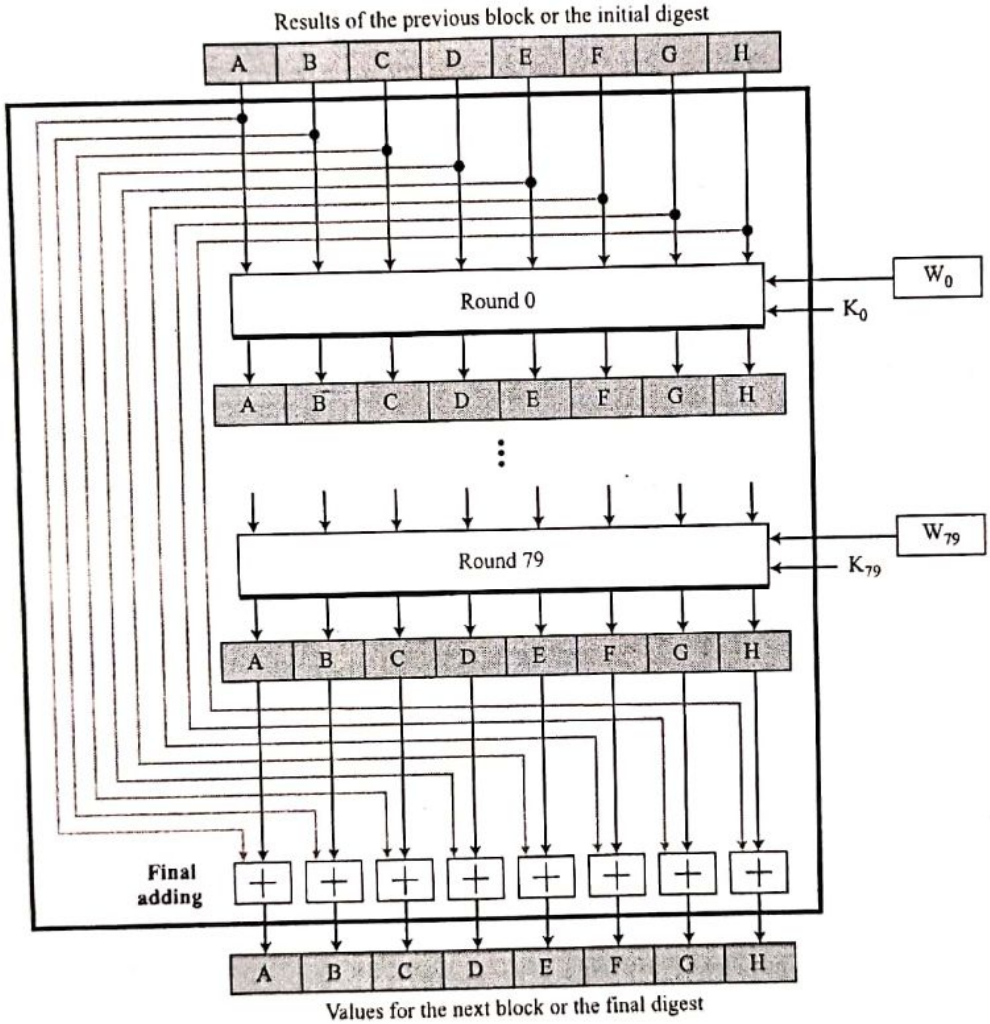
$$(100.0101\ 1011\ 1110 \ldots 1001)_2 \rightarrow (4.5BE0CD19137E2179)_{16}$$

SHA-512 keeps the fraction part, $(5BE0CD19137E2179)_{16}$, as an unsigned integer.

Compression Function

SHA-512 creates a 512-bit (eight 64-bit words) message digest from a multiple-block message where each block is 1024 bits. The processing of each block of data in SHA-512 involves 80 rounds. Figure 12.10 shows the general outline for the compression function. In each round, the contents of eight previous buffers, one word from the expanded block (W_i), and one 64-bit constant (K_i) are mixed together and then operated on to create a new set of eight buffers. At the beginning of processing, the values of the eight buffers are saved into eight temporary variables. At the end of the processing (after step 79), these values are added to the values created from step 79. We call this last operation the *final adding*, as shown in the figure.

Figure 12.10 *Compression function in SHA-512*



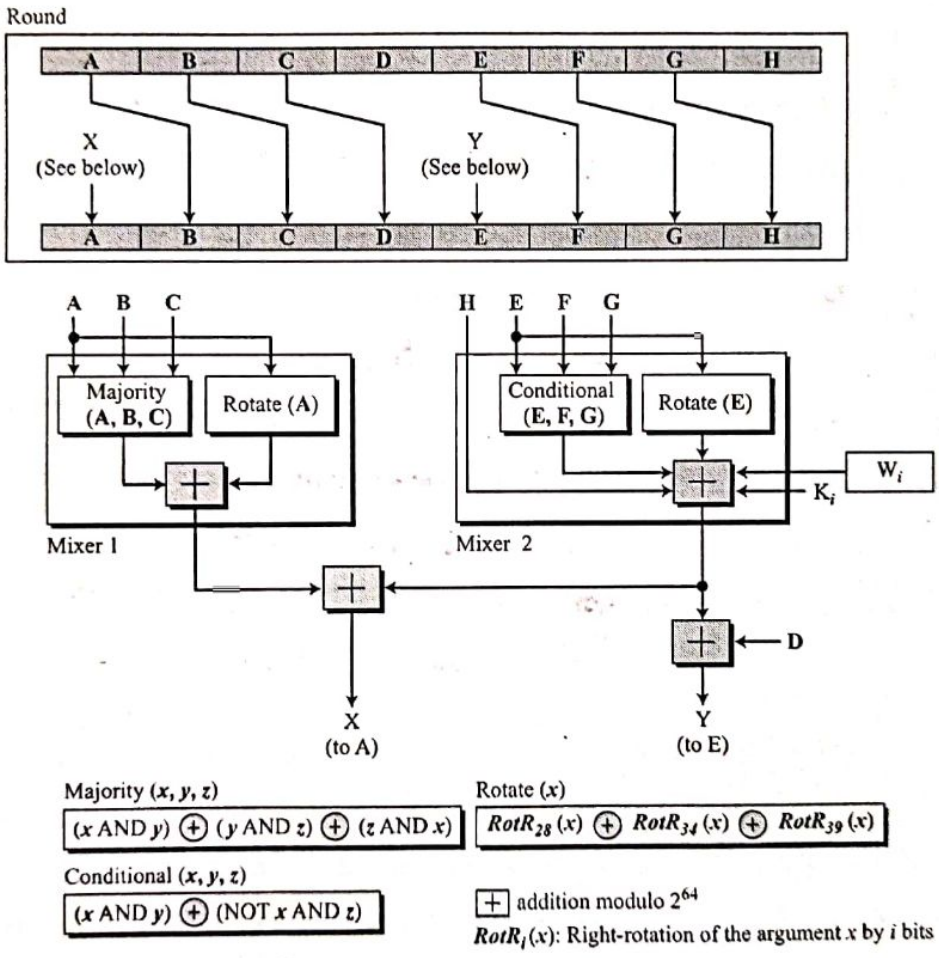
Structure of Each Round

In each round, eight new values for the 64-bit buffers are created from the values of the buffers in the previous round. As Figure 12.11 shows, six buffers are the exact copies of one of the buffers in the previous round as shown below:

$$A \rightarrow B \quad B \rightarrow C \quad C \rightarrow D \quad E \rightarrow F \quad F \rightarrow G \quad G \rightarrow H$$

Two of the new buffers, A and E, receive their inputs from some complex functions that involve some of the previous buffers, the corresponding word for this round (W_i), and the corresponding constant for this round (K_i). Figure 12.11 shows the structure of each round.

Figure 12.11 *Structure of each round in SHA-512*



There are two mixers, three functions, and several operators. Each mixer combines two functions. The description of the functions and operators follows:

1. The Majority function, as we call it, is a bitwise function. It takes three corresponding bits in three buffers (A, B, and C) and calculates

$$(A_j \text{ AND } B_j) \oplus (B_j \text{ AND } C_j) \oplus (C_j \text{ AND } A_j)$$

   The resulting bit is the majority of three bits. If two or three bits are 1's, the resulting bit is 1; otherwise it is 0.

2. The Conditional function, as we call it, is also a bitwise function. It takes three corresponding bits in three buffers (E, F, and G) and calculates

$$(E_j \text{ AND } F_j) \oplus (\text{NOT } E_j \text{ AND } G_j)$$

   The resulting bit is the logic "If E_j then F_j; else G_j".

3. The Rotate function, as we call it, right-rotates the three instances of the same buffer (A or E) and applies the exclusive-or operation on the results.

$$\textbf{Rotate (A): } \text{RotR}_{28}(A) \oplus \text{RotR}_{34}(A) \oplus \text{RotR}_{29}(A)$$

$$\textbf{Rotate (E): } \text{RotR}_{28}(E) \oplus \text{RotR}_{34}(E) \oplus \text{RotR}_{29}(E)$$

4. The right-rotation function, $\text{RotR}_i(x)$, is the same as the one we used in the word-expansion process. It right-rotates its argument i bits; it is actually a circular shift-right operation.
5. The addition operator used in the process is addition modulo 2^{64}. This means that the result of adding two or more buffers is always a 64-bit word.
6. There are 80 constants, K_0 to K_{79}, each of 64 bits as shown in Table 12.3 in hexadecimal format (four in a row). Similar to the initial values for the eight digest buffers, these values are calculated from the first 80 prime numbers (2, 3,..., 409).

Table 12.3 *Eighty constants used for eighty rounds in SHA-512*

428A2F98D728AE22	7137449123EF65CD	B5C0FBCFEC4D3B2F	E9B5DBA58189DBBC
3956C25BF348B538	59F111F1B605D019	923F82A4AF194F9B	AB1C5ED5DA6D8118
D807AA98A3030242	12835B0145706FBE	243185BE4EE4B28C	550C7DC3D5FFB4E2
72BE5D74F27B896F	80DEB1FE3B1696B1	9BDC06A725C71235	C19BF174CF692694
E49B69C19EF14AD2	EFBE4786384F25E3	0FC19DC68B8CD5B5	240CA1CC77AC9C65
2DE92C6F592B0275	4A7484AA6EA6E483	5CB0A9DCBD41FBD4	76F988DA831153B5
983E5152EE66DFAB	A831C66D2DB43210	B00327C898FB213F	BF597FC7BEEF0EE4
C6E00BF33DA88FC2	D5A79147930AA725	06CA6351E003826F	142929670A0E6E70
27B70A8546D22FFC	2E1B21385C26C926	4D2C6DFC5AC42AED	53380D139D95B3DF
650A73548BAF63DE	766A0ABB3C77B2A8	81C2C92E47EDAEE6	92722C851482353B
A2BFE8A14CF10364	A81A664BBC423001	C24B8B70D0F89791	C76C51A30654BE30
D192E819D6EF5218	D69906245565A910	F40E35855771202A	106AA07032BBD1B8
19A4C116B8D2D0C8	1E376C085141AB53	2748774CDF8EEB99	34B0BCB5E19B48A8
391C0CB3C5C95A63	4ED8AA4AE3418ACB	5B9CCA4F7763E373	682E6FF3D6B2B8A3
748F82EE5DEFB2FC	78A5636F43172F60	84C87814A1F0AB72	8CC702081A6439EC
90BEFFFA23631E28	A4506CEBDE82BDE9	BEF9A3F7B2C67915	C67178F2E372532B
CA273ECEEA26619C	D186B8C721C0C207	EADA7DD6CDE0EB1E	F57D4F7FEE6ED178
06F067AA72176FBA	0A637DC5A2C898A6	113F9804BEF90DAE	1B710B35131C471B
28DB77F523047D84	32CAAB7B40C72493	3C9EBE0A15C9BEBC	431D67C49C100D4C
4CC5D4BECB3E42B6	4597F299CFC657E2	5FCB6FAB3AD6FAEC	6C44198C4A475817

Each value is the fraction part of the cubic root of the corresponding prime number after converting it to binary and keeping only the first 64 bits. For example, the 80th prime is 409, with the cubic root $(409)^{1/3} = 7.42291412044$. Converting this number to binary with only 64 bits in the fraction part, we get

$$(111.0110\ 1100\ 0100\ 0100\ \ldots\ 0111)_2 \rightarrow (7.6C44198C4A475817)_{16}$$

SHA-512 keeps the fraction part, $(6C44198C4A475817)_{16}$, as an unsigned integer.

Example 12.7

We apply the Majority function on buffers A, B, and C. If the leftmost hexadecimal digits of these buffers are 0x7, 0xA, and 0xE, respectively, what is the leftmost digit of the result?

Solution

The digits in binary are 0111, 1010, and 1110.

a. The first bits are 0, 1, and 1. The majority is 1. We can also prove it using the definition of the Majority function:

$$(0 \text{ AND } 1) \oplus (1 \text{ AND } 1) \oplus (1 \text{ AND } 0) = 0 \oplus 1 \oplus 0 = 1$$

b. The second bits are 1, 0, and 1. The majority is 1.

c. The third bits are 1, 1, and 1. The majority is 1.

d. The fourth bits are 1, 0, and 0. The majority is 0.

The result is 1110, or 0xE in hexadecimal.

Example 12.8

We apply the Conditional function on E, F, and G buffers. If the leftmost hexadecimal digits of these buffers are 0x9, 0xA, and 0xF respectively, what is the leftmost digit of the result?

Solution

The digits in binary are 1001, 1010, and 1111.

a. The first bits are 1, 1, and 1. Since $E_1 = 1$, the result is F_1, which is 1. We can also use the definition of the Condition function to prove the result:

$$(1 \text{ AND } 1) \oplus (\text{NOT } 1 \text{ AND } 1) = 1 \oplus 0 = 1$$

b. The second bits are 0, 0, and 1. Since E_2 is 0, the result is G_2, which is 1.

c. The third bits are 0, 1, and 1. Since E_3 is 0, the result is G_3, which is 1.

d. The fourth bits are 1, 0, and 1. Since E_4 is 1, the result is F_4, which is 0.

The result is 1110, or 0xE in hexadecimal.

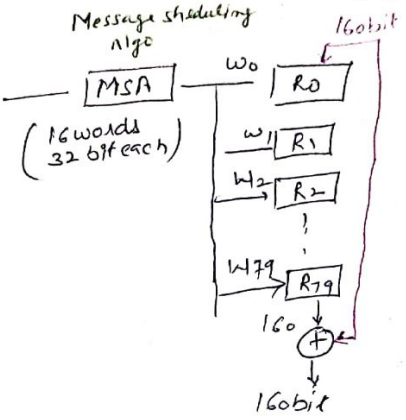
Analysis

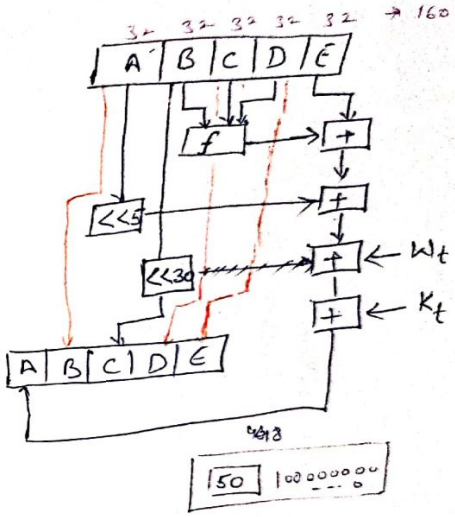
With a message digest of 512 bits, SHA-512 expected to be resistant to all attacks, including collision attacks. It has been claimed that this version's improved design makes it more efficient and more secure than the previous versions. However, more research and testing are needed to confirm this claim.

Variable length I/P(M) → SHA → HASH value (160 bits)

• $M(m_1, m_2, m_3 \cdots m_n)$

448 + 64 bit = 512 bits (length)





$A = H_0 = 67452301$

$B = H_1 = EFCDAB89$

$C = H_2 = 98BADCFE$

$D = H_3 = 10325476$

$E = H_4 = C3D2E1F0$

$K_{1t} = 5A827999 \quad 0 \le t \le 19 \rightarrow B \cdot C \wedge \bar{B} \cdot D$

$K_{2t} = 6ED9EBA1 \quad 20 \le t \le 39 \rightarrow B \oplus C \oplus D$

$K_{3t} = 8F1BBCDC \quad 40 \le t \le 59 \rightarrow B \cdot C \wedge B \cdot D \wedge C \cdot D$

$K_{4t} = CA62C1D6 \quad 60 \le t \le 79 \rightarrow B \oplus C \oplus D$

• → AND
∧ → OR
⊕ → XOR

⊞ → addition modulo 2^{32}

$W_t \rightarrow W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$

$W_{16} \rightarrow W_0 \oplus W_2 + W_8 \oplus W_{13}$