Table 4.10 *Comparison of MD5 and SHA-1*

Point of discussion	*MD5*	*SHA-1*
Message digest length in bits	128	160
Attack to try and find the original message given a message digest	Requires 2^{128} operations to break in	Requires 2^{160} operations to break in, therefore more secure
Attack to try and find two messages producing the same message digest	Requires 2^{64} operations to break in	Requires 2^{80} operations to break in
Successful attacks so far	There have been reported attempts to some extent (as we discussed earlier)	No such claims so far
Speed	Faster (64 iterations and 128-bit buffer)	Slower (80 iterations and 160-bit buffer)
Software implementation	Simple, does not need any large programs or complex tables	Simple, does not need any large programs or complex tables

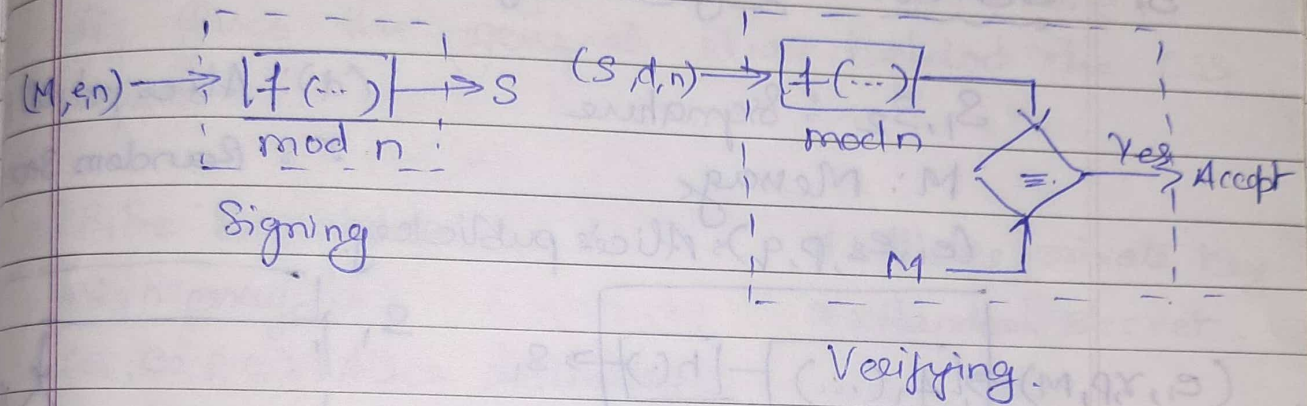
Security of SHA-1 In 2005, a possible vulnerability was discovered in SHA-1. Just before that, NIST had announced its intentions to seek more secure versions of SHA by 2010. Hence, various options are being discussed as follows: In 2002, NIST had come up with a new version of SHA in standard document FIPS 1802, called as SHA-256, SHA-384 and SHA-512; with the number after the word SHA indicating the length of the message digest in bits. Table 4.11 summarizes the various SHA versions.

Table 4.11 *Parameters for the Various Versions of SHA*

Digital Signature Scheme

① RSA Digital Signature Scheme

M: Message
S: Signature
(e,n): Alice's public key
d: Alice's private key

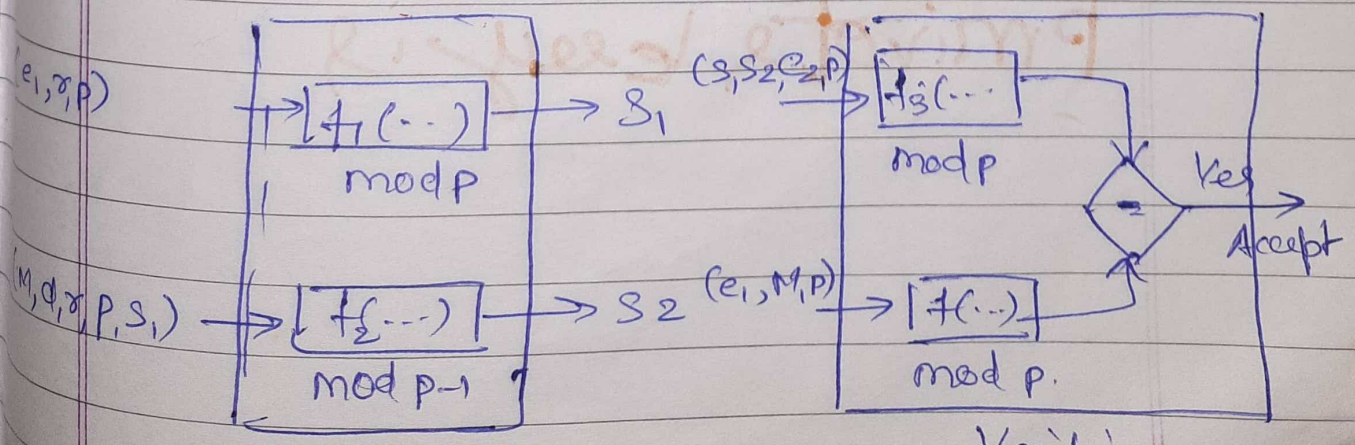


In RSA, d is private; e & n are public.

Attacks
- Key Only Attack
- Known-Message Attack.
- Chosen-Message Attack.

② ElGamal Digital Signature Scheme

S_1, S_2: Signature
M: Message
(e_1, e_2, p): Alice's public key.
d: Alice's private key.
r: Random Secret.



In (e_1, e_2, p) is Alice's public key; d is her private key

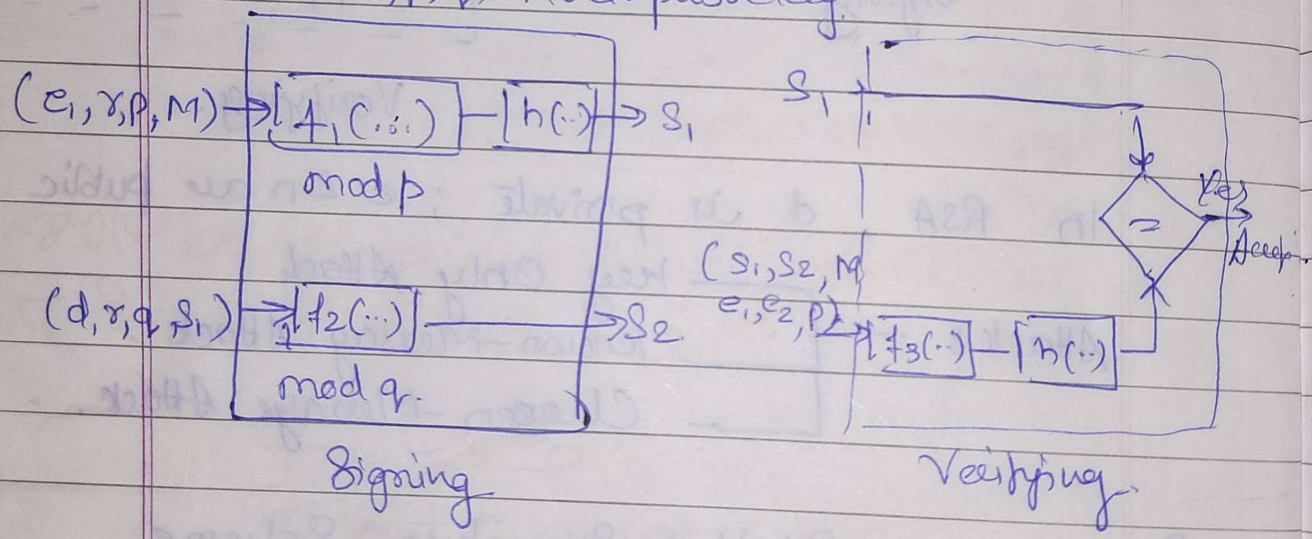
Forgery in the ElGamal Digital Signature Scheme.

Key Only Forgery
Known Message Forgery

3. Schnorr Digital Signature Scheme

S_1, S_2 : Signature
M: Message
(e_1, e_2, P, q): Alice's public key

(d): Alice private key
r: Random Secret.



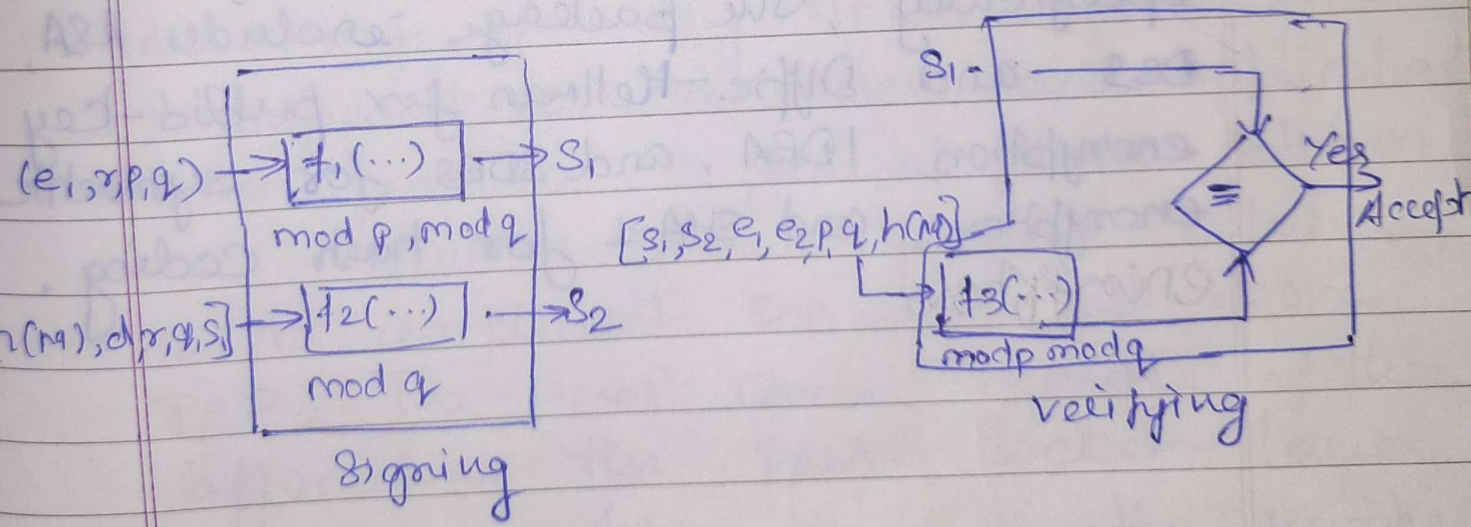
In this scheme, Alice's public key is (e_1, e_2, P, q); her private key is (d).

Private key is

4. Digital Signature Standard (DSS)

The DSS was adopted by the National Institute of Standards and Technology (NIST) in 1994. DSS uses a digital signature algorithm (DSA) based on the ElGamal scheme with some ideas from the Schnorr scheme. Fig. gives the general idea behind the DSS scheme.

S_1, S_2 : Signatures
M : Message
(e_1, e_2, p, q) : Alice public key
d : Alice's private key
r : Random secret.



Secure Hash Algorithm (SHA-1)

The national Institute of standards and Technology (NIST) along with NSA developed the Secure Hash Algorithm (SHA). In 1993, SHA was published as a Federal Information Processing standard (FIPS PUB 180). It was revised to FIPS PUB 180-I in 1995 and the name was changed to SHA-1. SHA is a modified version of MD5 and its design closely resembles MD5.

SHA works with any input message that is less than 2^{64} bits in length. The output of SHA is a message digest which is 160 bits in length. The word secure in SHA was decided based on two features. SHA is designed to be computationally infeasible to:

(a) Obtain the original message, given its message digest

(b) Find two messages produced the same message digest.

SHA working: SHA is closely modeled after MD5. It is similar to MD5.

Step-1 **Padding :-** Like MD5, the first step in SHA is to add padding to the end of the original message in such a way

that the length of the message is 64 bits short of a multiple of 512. Like MD5, the padding is always added, even if the message is already 64 bits short of a multiple of 512.

Step 2. Append length :- The length of the message excluding the length of the padding is now calculated and appended to the end of the padding as a 64-bit block.

Step 3. Divide the input into 512-bit blocks. :- The input message is now divided into blocks, each of length 512 bits. These blocks become the input to the message digest processing logic.

Step 4. Initialize chaining variables :- Five chaining variables A through E are initialized. We had four chaining variable, each of 32 bits in MD5. We stored the intermediates as well as the final results in the combined register made up of these four chaining variables, i.e. abcd. Since in the case of SHA, we want to produce a message digest of length 160 bits, we need to have five chaining variable here (5×32 = 160 bits). In SHA, the variables A through D have the same values as they had in MD5. Additionally, E is initialized to

Hex C3 D2 E1 F0.

Step 5: Process blocks :- Now the actual algorithm begins. Here also, the steps are quite similar to those in MD5.

Step 5.1: Copy the chaining variable A - E into variables a-e. The combination of a-e, called as abcde will be considered as a single register for storing the temporary intermediate as well as the final results.

Step 5.2. Now, divide the current 512-bit block into 16 sub-blocks, each consisting of 32 bits

Step 5.3 SHA has four rounds, each round consisting of 20 steps. Each round takes the current 512-bit block, the register abcde and a constant K(t) (where $t = 0$ to 79) as the three inputs. It then updates the contents of the register abcde using the SHA algorithm steps.

Step 5.4: SHA Consists of four rounds, each round containing 20 iterations. This makes it a total of 80 iterations. The logical operation of a single SHA iteration looks as figure.

Mathematically, an Iteration consists of the following operation.

$abcde = (e + \text{Process P} + S^5(a) + W[t] + K[t]), a, S^{30}(b), c, d.$

where.

abcde = The register made up of the five variables a, b, c, d & e.

Process P = The Logical operation, ~~which~~

S^t = Circular-left shift of the 32-bit sub-block by t bits.

W[t] = A 32-bit derived from the current 32-bit sub block,

K[t] = One of the five additive constants

Process P in each SHA-1 round

Round	Process P
1	(b AND c) OR ((NOT b) AND (d))
2	B XOR C XOR d
3	(b AND C) OR (b and D) OR (C AND d)
4.	B XOR C XOR d.

EIEven

2. den Boer and Bosselaers showed that the execution of MD5 on a single block of 512 bits will produce the same output for two different values in the chaining variable register abcd. This is called as **pseudocollision**. However, they could not extend this to a full MD5 consisting of four rounds, each containing 16 steps.
3. Dobbertin provided the most serious attack on MD5. Using his attack, the operation of MD5 on two different 512-bit blocks produces the same 128-bit output. However, this has not been generalized to a full message block.

The general recommendation now is not to trust MD5 (although it is not practically broken into, as yet). Consequently, the quest for better message algorithms has led to the possible alternative, called as **SHA-1**. We will study it now.

4.6.4 Secure Hash Algorithm (SHA)

Introduction The National Institute of Standards and Technology (NIST) along with NSA developed the **Secure Hash Algorithm (SHA)**. In 1993, SHA was published as a Federal Information Processing Standard (FIPS PUB 180). It was revised to FIPS PUB 180-1 in 1995 and the name was changed to SHA-1. As we shall study, SHA is a modified version of MD5 and its design closely resembles MD5.

SHA works with any input message that is less than 2^{64} bits in length. The output of SHA is a message digest, which is 160 bits in length (32 bits more than the message digest produced by MD5). The word *Secure* in SHA was decided based on two features. SHA is designed to be computationally infeasible to:

(a) Obtain the original message, given its message digest and
(b) Find two messages producing the same message digest

How SHA Works? As we have mentioned before, SHA is closely modeled after MD5. Therefore, we shall not discuss in detail those features of SHA, which are similar to MD5. Instead, we shall simply mention them and point out the differences. The reader can go back to the appropriate descriptions of MD5 to find out more details.

Step 1: Padding Like MD5, the first step in SHA is to add padding to the end of the original message in such a way that the length of the message is 64 bits short of a multiple of 512. Like MD5, the padding is always added, even if the message is already 64 bits short of a multiple of 512.

Step 2: Append length The length of the message excluding the length of the padding is now calculated and appended to the end of the padding as a 64-bit block.

Step 3: Divide the input into 512-bit blocks The input message is now divided into blocks, each of length 512 bits. These blocks become the input to the message digest processing logic.

Step 4: Initialize chaining variables Now, five *chaining variables* A through E are initialized. Remember that we had four chaining variables, each of 32 bits in MD5 (which made the total length of the variables $4 \times 32 = 128$ bits). Recall that we stored the intermediate as well as the final results into the combined register made up of these four chaining variables, i.e. abcd. Since in the case of SHA, we want to produce a message digest of length 160 bits, we need to have five chaining variables here ($5 \times 32 = 160$ bits). In SHA, the variables A through D have the same values as they had in MD5. Additionally, E is initialized to Hex C3 D2 E1 F0.

Step 5: Process blocks Now the actual algorithm begins. Here also, the steps are quite similar to those in MD5.

Step 5.1: Copy the chaining variables A-E into variables a-e. The combination of a-e, called as abcde will be considered as a single register for storing the temporary intermediate as well as the final results.

Step 5.2: Now, divide the current 512-bit block into 16 sub-blocks, each consisting of 32 bits.

Step 5.3: SHA has four rounds, each round consisting of 20 steps. Each round takes the current 512-bit block, the register abcde and a constant K[t] (where t = 0 to 79) as the three inputs. It then updates the contents of the register abcde using the SHA algorithm steps. Also notable is the fact that we had 64 constants defined as t in MD5. Here, we have only four constants defined for K[t], one used in each of the four rounds. The values of K[t] are as shown in Table 4.7.

Table 4.7 *Values of K[t]*

Round	*Value of t between*	*K[t] in hexadecimal*	*K[t] in decimal (Only integer portion of the value shown)*
1	0 and 19	5A 92 79 99	$2^{30} \times \sqrt{2}$
2	20 and 39	6E D9 EB A1	$2^{30} \times \sqrt{3}$
3	40 and 59	9F 1B BC DC	$2^{30} \times \sqrt{5}$
4	60 and 79	CA 62 C1 D6	$2^{30} \times \sqrt{10}$

Step 5.4: SHA consists of four rounds, each round containing 20 iterations. This makes it a total of 80 iterations. The logical operation of a single SHA iteration looks as shown in Fig. 4.32.

Mathematically, an iteration consists of the following operations:

$$abcde = (e + \text{Process P} + s^5 (a) + W[t] + K[t]), a, s^{30} (b), c, d$$

Where,

abcde = The register made up of the five variables a, b, c, d and e

Process P = The logical operation, which we shall study later

s^t = Circular-left shift of the 32-bit sub-block by t bits

W[t] = A 32-bit derived from the current 32-bit sub block, as we shall study later

K[t] = One of the five additive constants, as defined earlier

We will notice that this is very similar to MD5, with a few variations (which are induced to try and make SHA more complicated in comparison with MD5). We have to now see what are the meanings of the process P and W[t] in the above equation.

Table 4.8 illustrates the process P.

Table 4.8 *Process P in each SHA-1 round*

Round	*Process P*
1	(b AND c) OR ((NOT b) AND (d))
2	B XOR c XOR d
3	(b AND c) OR (b and D) OR (c AND d)
4	B XOR c XOR d

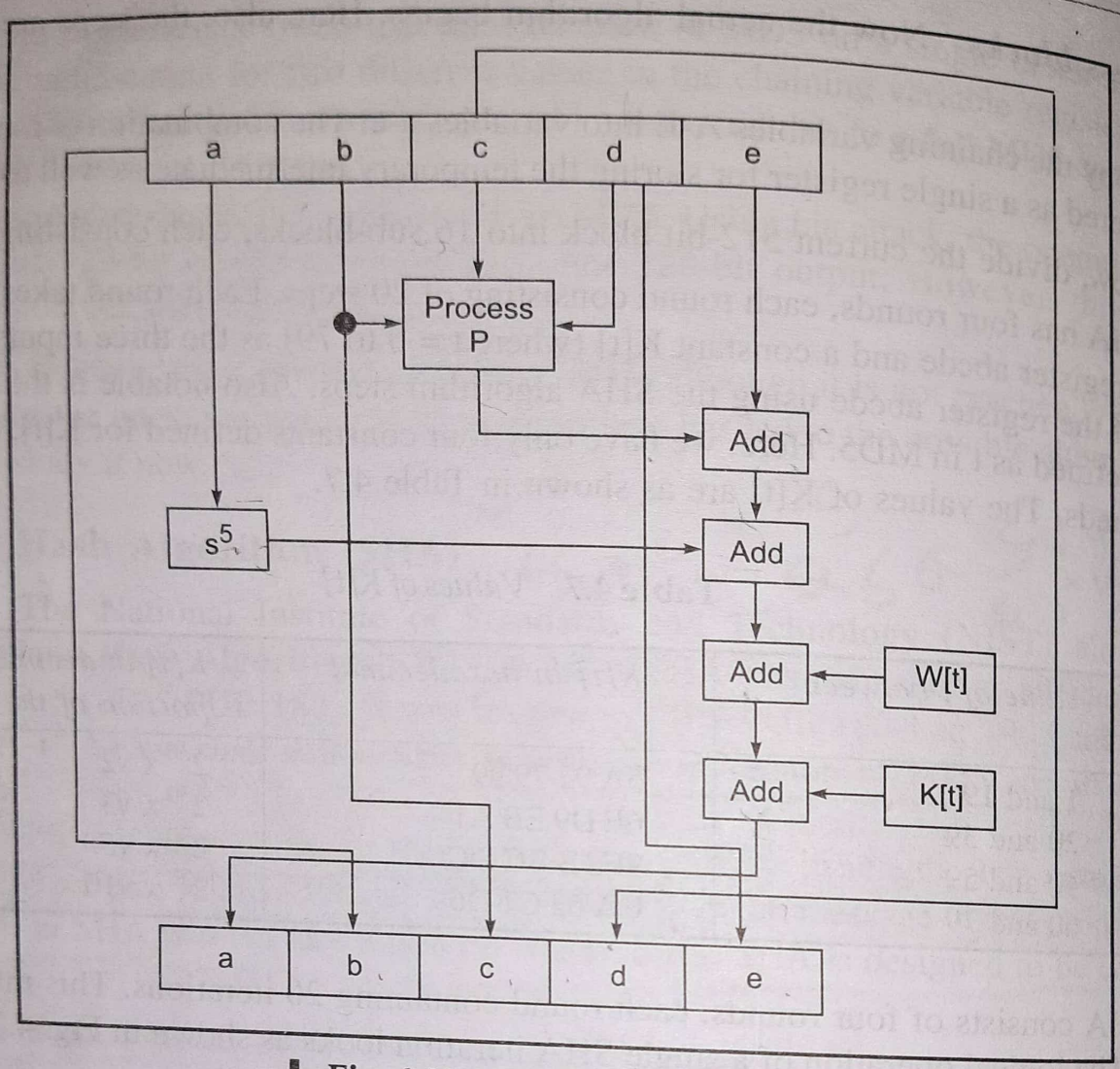


Fig. 4.32 *Single SHA-1 iteration*

The values of W[t] are calculated as follows:

For the first 16 words of W (i.e. t = 0 to 15), the contents of the input message sub-block M become the contents of W[t] straightaway. That is, the first 16 blocks of the input message M copied to W.

The remaining 64 values of W are derived using the equation:

$$W[t] = s^1 (W [t - 16] \text{ XOR } W[t - 14] \text{ XOR } W[t - 8] \text{ XOR } W[t - 3])$$

As before, s^1 indicates a circular-left shift (i.e. rotation) by 1 bit position.

Thus, we can summarize the values of W as shown in Table 4.9.

Table 4.9 *Values of W in SHA-1*

For t = 0 to 15	*Value of W[t]*
W[t] =	Same as M[t]
W[t] =	s^1 (W [t – 16] XOR W[t – 14] XOR W[t – 8] XOR W[t – 3])