**Experiment No. 4**

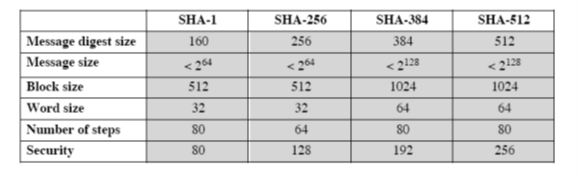
**Aim:** Write a program in C++ or Java to implement SHA-1 (Secure Hash Algorithm) using Libraries

**Objectives:** To study

* Hashing algorithm
* Working of SHA-1 algorithm
* Different security API/Library

**Theory:**

The National Institute of Standard and Technology(NIST) along with NSA developed the Secure Hash Algorithm(SHA).SHA works with any input message that is less than 2 64bits in length. The output of SHA-1 is a message digest which is 160 bits in length. Following table shows SHA parameters.



**Figure 4.1** Comparison of SHA parameters



The word secure is SHA was decided based on two features. SHA is designed to be computationally infeasible to:

1. Obtain the original message, given its message digest

2. Find two messages producing the same message digest

Following are the important steps in execution of SHA

**Step 1. Padding**

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 multiple of 512.Padding is always added even if message is already 64 bit short of 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 block**

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 initializes. Each chaining variable is 32 bits in length. Following table shows their values

|  |  |
| --- | --- |
| **Variable Name** | **Value(in Hexadecimal)** |
| A | 01 23 45 67 |
| B | 89 AB CD EF |
| C | FE DC BA 98 |
| D | 76 54 32 10 |
| E | C3 D2 E1 F0 |

**Fig. 4.2** Values of chaining variable

**Step 5. Process Block**

This process is divided into following sub steps

**Step 5.1**

Copy the chaining variables –E into variables a-e. The combination of a-e called 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 4 rounds each consisting of 20 steps or iteration. 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 SHA algorithm steps. We have only four constants defined for K[t],one used in each of four rounds .The values of K[t] are given in following table

|  |  |  |
| --- | --- | --- |
| Round | Value of t between | K[t](in Hexadecimal) |
| 1 | 1 and 19 | 5A 92 79 99 |
| 2 | 20 and 39 | 6E D9 EB A1 |
| 3 | 40 and 59 | 9F 1B BC DC |
| 4 | 60 and 79 | CA 62 C1 D6 |

**Fig. 4.3** Values of K[t]

**Step 5.4**

SHA consist of four rounds, each round containing 20 iterations. This makes a total of 80 iterations. The logical operation of SHA-1 is shown in following fig.

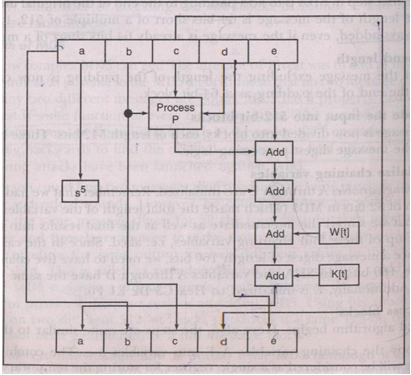


Fig. 4.4 Single SHA-1 iteration

Mathematically iteration consists of following operations

abcde =(e + Process P +s5 (a) + W[t] + K[t]),a,s30  (b),c, d

where

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

process P=The logical operation ,which is given in following table

st=Circular left shift of 32 bit sub block by t bits.

W[t]=A 32 bit value derived from current 32 bit sub block

K[t]=one of the constant defined earlier

Process P is given as

|  |  |
| --- | --- |
| 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 4.5** Process P in four rounds

Value of W[t] is calculated as follow:

For the first 16 blocks of W(i,e t=0 to 15) the contents of the input message sub block M[t] become the

content of W[t]. Remaining values are calculated as

|  |  |
| --- | --- |
| Value of t | Value of W[t] |
| 0 to 15 | W[t]=Same as M[t] |
| 16 to 63 | W[t]= (W[t -16] **XOR** W[t-14] **XOR** W[t-8] **XOR** W[t-3]) |

**Fig 4.6** Values of W

**INPUT:** Plaintext (preferably read a input file)

**OUTPUT: Message** Digest

**Conclusion**: We have successfully implemented SHA1 using OpenSSL library.

**Source Code:**

#include<bits/stdc++.h>

#include <openssl/sha.h>

#include<fstream>

using namespace std;

int main()

{

string fname;

cout << "Enter file name: ";

getline(cin, fname);

ifstream is (fname, std::ifstream::binary);

if (is)

{

// get length of file:

is.seekg (0, is.end);

int length = is.tellg();

is.seekg (0, is.beg);

char \* ibuffer = new char [length];

cout << "Reading " << length << " characters... ";

// read data as a block:

is.read (ibuffer,length);

if (is)

cout << "all characters read successfully.\n";

else

cout << "error: only " << is.gcount() << " could be read";

is.close();

// ...buffer contains the entire file...

unsigned char tmphash[SHA\_DIGEST\_LENGTH]; // == 20 bytes aka 160 bits

unsigned char\* buffer = reinterpret\_cast<unsigned char\*>(ibuffer);

cout << "\nBuffer contents:" << buffer << "\n";

SHA1(buffer, length - 1, tmphash);

int i = 0;

for(; i < SHA\_DIGEST\_LENGTH; i++)

printf("%02x", tmphash[i]);

cout << "\n";

delete[] buffer;

}

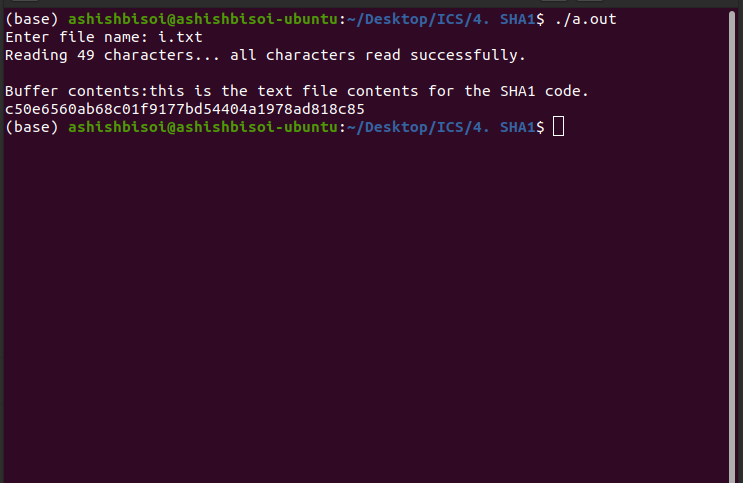
else

cout << "File not found";

return 0;

}

**Snapshot :**

****