

PRACTICAL-9

AIM: Write a program to implement SHA1 Algorithm.

INTRODUCTION:

- SHA-1 or Secure Hash Algorithm 1 is a cryptographic hash function which takes an input and produces a 160-bit (20-byte) hash value. This hash value is known as a message digest. This message digest is usually then rendered as a hexadecimal number which is 40 digits long. It is a U.S. Federal Information Processing Standard and was designed by the United States National Security Agency.
- SHA-1 is now considered insecure since 2005. Major tech giants browsers like Microsoft, Google, Apple and Mozilla have stopped accepting SHA-1 SSL certificates by 2017.
- The different SHA hash functions are explained below.
 - **SHA256** : This hash function belong to hash class SHA-2, the internal block size of it is 32 bits.
 - **SHA384** : This hash function belong to hash class SHA-2, the internal block size of it is 32 bits. This is one of the truncated version.
 - **SHA224** : This hash function belong to hash class SHA-2, the internal block size of it is 32 bits. This is one of the truncated version.
 - **SHA512** : This hash function belong to hash class SHA-2, the internal block size of it is 64 bits.
 - **SHA1** : The 160 bit hash function that resembles MD5 hash in working and was discontinued to be used seeing its security vulnerabilities.
- Hashing transforms a string of characters into a shorter fixed length value which is sufficient to represent the original string. The hash obtained at the end of hashing process is combined with the original message and is used to ensure the integrity of the message. If the hash calculated at receiver end does not matches with the attached hash, the integrity of message has been compromised. Also, a layer of encryption over the packet (which has both Message + HMAC) can help to ensure message confidentiality, because if adversary somehow manages to get his hands on this packet, he still cannot make any sense out of it.

CODE:

```
#include <iostream>

#include <string>

#include <sstream>

#include <iomanip>

#include <fstream>

#define SHA1_ROL(value, bits) (((value) << (bits)) | (((value) & 0xffffffff) >> (32 - (bits))))

#define SHA1_BLK(i) (block[i&15] = SHA1_ROL(block[(i+13)&15] ^ block[(i+8)&15] ^ block[(i+2)&15] ^ block[i&15],1))
```

```

#define SHA1_R0(v,w,x,y,z,i) z += ((w&(x^y))^y) + block[i] + 0x5a827999 +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R1(v,w,x,y,z,i) z += ((w&(x^y))^y) + SHA1_BLK(i) + 0x5a827999 +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R2(v,w,x,y,z,i) z += (w^x^y) + SHA1_BLK(i) + 0x6ed9eba1 +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R3(v,w,x,y,z,i) z += (((w|x)&y)|(w&x)) + SHA1_BLK(i) + 0x8f1bbcdc +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

#define SHA1_R4(v,w,x,y,z,i) z += (w^x^y) + SHA1_BLK(i) + 0xca62c1d6 +
SHA1_ROL(v,5); w=SHA1_ROL(w,30);

class SHA1{
public:
    SHA1();

    void update(const std::string &s);
    void update(std::istream &is);

    std::string final();

    static std::string from_file(const std::string &filename);

private:
    typedef unsigned long int uint32;
    typedef unsigned long long uint64;

    static const unsigned int DIGEST_INTS = 5;
    static const unsigned int BLOCK_INTS = 16;
    static const unsigned int BLOCK_BYTES = BLOCK_INTS * 4;

    uint32 digest[DIGEST_INTS];

    std::string buffer;

    uint64 transforms;

    void reset();

    void transform(uint32 block[BLOCK_BYTES]);

    static void buffer_to_block(const std::string &buffer, uint32
block[BLOCK_BYTES]);

    static void read(std::istream &is, std::string &s, int max);
};

```

```
std::string sha1(const std::string &string);
using namespace std;
int main(int argc, char *argv[])
{
    string str;
    cout<<"Enter word: ";
    cin>>str;
    cout << sha1(str) << endl;
    return 0;
}
SHA1::SHA1()
{
    reset();
}
void SHA1::update(const std::string &s)
{
    std::istringstream is(s);
    update(is);
}
void SHA1::update(std::istream &is)
{
    std::string rest_of_buffer;
    read(is, rest_of_buffer, BLOCK_BYTES - buffer.size());
    buffer += rest_of_buffer;
    while (is)
    {
        uint32 block[BLOCK_INTS];
        buffer_to_block(buffer, block);
        transform(block);
        read(is, buffer, BLOCK_BYTES);
    }
}
```

```
    }  
}  
std::string SHA1::final()  
{  
    uint64 total_bits = (transforms*BLOCK_BYTES + buffer.size()) * 8;  
    buffer += 0x80;  
    unsigned int orig_size = buffer.size();  
    while (buffer.size() < BLOCK_BYTES)  
    {  
        buffer += (char)0x00;  
    }  
    uint32 block[BLOCK_INTS];  
    buffer_to_block(buffer, block);  
    if (orig_size > BLOCK_BYTES - 8)  
    {  
        transform(block);  
        for (unsigned int i = 0; i < BLOCK_INTS - 2; i++)  
        {  
            block[i] = 0; }  
    }  
    block[BLOCK_INTS - 1] = total_bits;  
    block[BLOCK_INTS - 2] = (total_bits >> 32);  
    transform(block);  
    std::ostringstream result;  
    for (unsigned int i = 0; i < DIGEST_INTS; i++)  
    {  
        result << std::hex << std::setfill('0') << std::setw(8);  
        result << (digest[i] & 0xffffffff);  
    }  
    reset();  
}
```

```
        return result.str();
    }

    std::string SHA1::from_file(const std::string &filename)
    {
        std::ifstream stream(filename.c_str(), std::ios::binary);
        SHA1 checksum;
        checksum.update(stream);
        return checksum.final();
    }

    void SHA1::reset()
    {
        digest[0] = 0x67452301;
        digest[1] = 0xefcdab89;
        digest[2] = 0x98badcfe;
        digest[3] = 0x10325476;
        digest[4] = 0xc3d2e1f0;
        transforms = 0;
        buffer = "";
    }

    void SHA1::transform(uint32 block[BLOCK_BYTES])
    {
        uint32 a = digest[0];
        uint32 b = digest[1];
        uint32 c = digest[2];
        uint32 d = digest[3];
        uint32 e = digest[4];

        SHA1_R0(a,b,c,d,e, 0);
        SHA1_R0(e,a,b,c,d, 1);
        SHA1_R0(d,e,a,b,c, 2);
        SHA1_R0(c,d,e,a,b, 3);
```

SHA1_R0(b,c,d,e,a, 4);
SHA1_R0(a,b,c,d,e, 5);
SHA1_R0(e,a,b,c,d, 6);
SHA1_R0(d,e,a,b,c, 7);
SHA1_R0(c,d,e,a,b, 8);
SHA1_R0(b,c,d,e,a, 9);
SHA1_R0(a,b,c,d,e,10);
SHA1_R0(e,a,b,c,d,11);
SHA1_R0(d,e,a,b,c,12);
SHA1_R0(c,d,e,a,b,13);
SHA1_R0(b,c,d,e,a,14);
SHA1_R0(a,b,c,d,e,15);
SHA1_R1(e,a,b,c,d,16);
SHA1_R1(d,e,a,b,c,17);
SHA1_R1(c,d,e,a,b,18);
SHA1_R1(b,c,d,e,a,19);
SHA1_R2(a,b,c,d,e,20);
SHA1_R2(e,a,b,c,d,21);
SHA1_R2(d,e,a,b,c,22);
SHA1_R2(c,d,e,a,b,23);
SHA1_R2(b,c,d,e,a,24);
SHA1_R2(a,b,c,d,e,25);
SHA1_R2(e,a,b,c,d,26);
SHA1_R2(d,e,a,b,c,27);
SHA1_R2(c,d,e,a,b,28);
SHA1_R2(b,c,d,e,a,29);
SHA1_R2(a,b,c,d,e,30);
SHA1_R2(e,a,b,c,d,31);
SHA1_R2(d,e,a,b,c,32);
SHA1_R2(c,d,e,a,b,33);

SHA1_R2(b,c,d,e,a,34);
SHA1_R2(a,b,c,d,e,35);
SHA1_R2(e,a,b,c,d,36);
SHA1_R2(d,e,a,b,c,37);
SHA1_R2(c,d,e,a,b,38);
SHA1_R2(b,c,d,e,a,39);
SHA1_R3(a,b,c,d,e,40);
SHA1_R3(e,a,b,c,d,41);
SHA1_R3(d,e,a,b,c,42);
SHA1_R3(c,d,e,a,b,43);
SHA1_R3(b,c,d,e,a,44);
SHA1_R3(a,b,c,d,e,45);
SHA1_R3(e,a,b,c,d,46);
SHA1_R3(d,e,a,b,c,47);
SHA1_R3(c,d,e,a,b,48);
SHA1_R3(b,c,d,e,a,49);
SHA1_R3(a,b,c,d,e,50);
SHA1_R3(e,a,b,c,d,51);
SHA1_R3(d,e,a,b,c,52);
SHA1_R3(c,d,e,a,b,53);
SHA1_R3(b,c,d,e,a,54);
SHA1_R3(a,b,c,d,e,55);
SHA1_R3(e,a,b,c,d,56);
SHA1_R3(d,e,a,b,c,57);
SHA1_R3(c,d,e,a,b,58);
SHA1_R3(b,c,d,e,a,59);
SHA1_R4(a,b,c,d,e,60);
SHA1_R4(e,a,b,c,d,61);
SHA1_R4(d,e,a,b,c,62);
SHA1_R4(c,d,e,a,b,63);

```

    SHA1_R4(b,c,d,e,a,64);
    SHA1_R4(a,b,c,d,e,65);
    SHA1_R4(e,a,b,c,d,66);
    SHA1_R4(d,e,a,b,c,67);
    SHA1_R4(c,d,e,a,b,68);
    SHA1_R4(b,c,d,e,a,69);
    SHA1_R4(a,b,c,d,e,70);
    SHA1_R4(e,a,b,c,d,71);
    SHA1_R4(d,e,a,b,c,72);
    SHA1_R4(c,d,e,a,b,73);
    SHA1_R4(b,c,d,e,a,74);
    SHA1_R4(a,b,c,d,e,75);
    SHA1_R4(e,a,b,c,d,76);
    SHA1_R4(d,e,a,b,c,77);
    SHA1_R4(c,d,e,a,b,78);
    SHA1_R4(b,c,d,e,a,79);
    digest[0] += a;
    digest[1] += b;
    digest[2] += c;
    digest[3] += d;
    digest[4] += e;
    transforms++;
}

void SHA1::buffer_to_block(const std::string &buffer, uint32 block[BLOCK_BYTES])
{
    for (unsigned int i = 0; i < BLOCK_INTS; i++)
    {
        block[i] = (buffer[4*i+3] & 0xff)
            | (buffer[4*i+2] & 0xff)<<8
            | (buffer[4*i+1] & 0xff)<<16

```

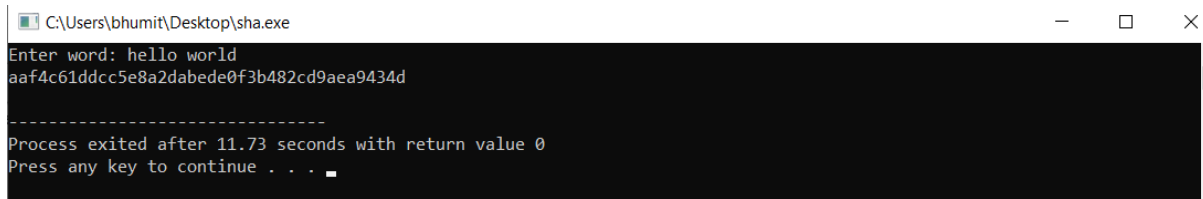


```
        | (buffer[4*i+0] & 0xff)<<24;
    }
}

void SHA1::read(std::istream &is, std::string &s, int max)
{
    char sbuf[max];
    is.read(sbuf, max);
    s.assign(sbuf, is.gcount());
}

std::string sha1(const std::string &string)
{
    SHA1 checksum;
    checksum.update(string);
    return checksum.final();
}
```

OUTPUT:



```
C:\Users\bhumit\Desktop\sha.exe
Enter word: hello world
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
-----
Process exited after 11.73 seconds with return value 0
Press any key to continue . . .
```