Cryptography And Network Security Lab

Assignment submission

PRN No: 2019BTECS00017

Full name: Muskan Raju Attar

Batch: B5

Assignment: 13

Title of assignment: Implementation of SHA – 512 (Secured Hash Algorithm)

Title:

Implementation of SHA – 512 (Secured Hash Algorithm)

Aim:

To develop and implement the SHA – 512 (Secured Hash Algorithm)

Theory:

- SHA-2 (Secure Hash Algorithm 2) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001.
- They are built using the Merkle–Damgård construction, from a one-way compression function itself built using the Davies–Meyer structure from a specialized block cipher.
- SHA-2 includes significant changes from its predecessor, SHA-1. The SHA-2 family consists of six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits.

- SHA-512, or Secure Hash Algorithm 512, is a hashing algorithm used to convert text of any length into a fixed-size string. Each output produces a SHA-512 length of 512 bits (64 bytes). This algorithm is commonly used for email addresses hashing, password hashing, and digital record verification.

Implementation of SHA 512 (Secured Hash Algorithm)

Code:

```
#include<bits/stdc++.h>

#define ull unsigned long long

#define SHA_512_INPUT_REPRESENTATION_LENGTH 128
#define BLOCK_SIZE 1024

#define BUFFER_COUNT 8
#define WORD_LENGTH 64
#define ROUND_COUNT 80

using namespace std;

void initialiseBuffersAndConstants(vector<ull>& buffers, vector<ull>&
constants)
{
        buffers = {
                0x6a09e667f3bcc908, 0xbb67ae8584caa73b,
0x3c6ef372fe94f82b, 0xa54ff53a5f1d36f1,
     0x510e527fade682d1, 0x9b05688c2b3e6c1f, 0x1f83d9abfb41bd6b,
0x5be0cd19137e2179
   };

        constants = {
```

```
        0x428a2f98d728ae22, 0x7137449123ef65cd,
0xb5c0fbcfec4d3b2f, 0xe9b5dba58189dbbc, 0x3956c25bf348b538,
        0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118,
0xd807aa98a3030242, 0x12835b0145706fbe,
        0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2, 0x72be5d74f27b896f,
0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
        0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3,
0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
        0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4,
0x76f988da831153b5, 0x983e5152ee66dfab,
        0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4,
0xc6e00bf33da88fc2, 0xd5a79147930aa725,
        0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc,
0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed,
        0x53380d139d95b3df, 0x650a73548baf63de,
0x766a0abb3c77b2a8, 0x81c2c92e47edaee6, 0x92722c851482353b,
        0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791,
0xc76c51a30654be30, 0xd192e819d6ef5218,
        0xd69906245565a910, 0xf40e35855771202a,
0x106aa07032bbd1b8, 0x19a4c116b8d2d0c8, 0x1e376c085141ab53,
        0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63,
0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373,
        0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc, 0x78a5636f43172f60,
0x84c87814a1f0ab72, 0x8cc702081a6439ec,
        0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915,
0xc67178f2e372532b, 0xca273eceea26619c,
        0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178,
0x06f067aa72176fba, 0x0a637dc5a2c898a6,
        0x113f9804bef90dae, 0x1b710b35131c471b, 0x28db77f523047d84,
0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
        0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a,
0x5fcb6fab3ad6faec, 0x6c44198c4a475817
    };
}
```

```
string sha512Padding(string input)
{
	string finalPlainText = "";

	for(int i=0 ; i<input.size() ; ++i)
	{
		finalPlainText += bitset<8>((int)input[i]).to_string();
	}

	finalPlainText += '1';

	int plainTextSize = input.size() * 8;
	int numberOfZeros = BLOCK_SIZE - ((plainTextSize +
SHA_512_INPUT_REPRESENTATION_LENGTH + 1) % BLOCK_SIZE);

	while(numberOfZeros--)
	{
		finalPlainText += '0';
	}

	finalPlainText +=
bitset<SHA_512_INPUT_REPRESENTATION_LENGTH>(plainTextSize).to_s
tring();

	cout<<"Plain text length = "<<plainTextSize<<endl;
	cout<<"Plain text length after padding =
"<<finalPlainText.length()<<endl<<endl;

	return finalPlainText;
}

ull getUllFromString(string str)
{
	bitset<WORD_LENGTH> word(str);
	return word.to_ullong();
```

```
}

static inline ull rotr64(ull n, ull c)
{
	const unsigned int mask = (CHAR_BIT * sizeof(n) - 1);
	c &= mask;
	return (n>>c) | (n<<((-c)&mask ));
}

int main()
{

	vector<ull> buffers(BUFFER_COUNT);
	vector<ull> constants(ROUND_COUNT);

	initialiseBuffersAndConstants(buffers, constants);

	cout<<"Enter Text: ";
	string input;
	getline(cin, input);

	cout<<"Input: "<<input<<endl;
	string paddedInput = sha512Padding(input);

	cout<<"Padded Input:"<<" "<<paddedInput<<endl<<endl;

	for(int i=0 ; i<paddedInput.size() ; i+=BLOCK_SIZE)
	{
		string currentBlock = paddedInput.substr(i, BLOCK_SIZE);

		vector<ull> w(ROUND_COUNT);
		for(int j=0 ; j<16 ; ++j)
		{
			w[j] = getUllFromString(currentBlock.substr(j,
WORD_LENGTH));
```

```
                }

                for(int j=16 ; j<80 ; ++j)
                {
                        ull sigma1 = (rotr64(w[j-15], 1)) ^ (rotr64(w[j-15], 8))
^ (w[j-15] >> 7);
                        ull sigma2 = (rotr64(w[j-2], 19)) ^ (rotr64(w[j-2], 61))
^ (w[j-2] >> 6);

        w[j] = w[j-16] + sigma1 + w[j-7] + sigma2;
                }

                ull a = buffers[0], b = buffers[1], c = buffers[2], d =
buffers[3];
                ull e = buffers[4], f = buffers[5], g = buffers[6], h =
buffers[7];

                for(int j=0 ; j<ROUND_COUNT ; ++j)
                {

                        ull sum0 = (rotr64(a, 28)) ^ (rotr64(a, 34)) ^ (rotr64(a,
39));
        ull sum1 = (rotr64(e, 14)) ^ (rotr64(e, 18)) ^ (rotr64(e, 41));

        ull ch = (e && f) ^ ((!e) && g);
        ull temp1 = h + sum1 + ch + constants[i] + w[i];

        ull majorityFunction = (a && b) ^ (a && c) ^ (b && c);
        ull temp2 = sum0 + majorityFunction;

        h = g;
        g = f;
        f = e;
        e = d + temp1;
        d = c;
```

```
            c = b;
            b = a;
            a = temp1 + temp2;
        }

        buffers[0] += a;
        buffers[1] += b;
        buffers[2] += c;
        buffers[3] += d;
        buffers[4] += e;
        buffers[5] += f;
        buffers[6] += g;
        buffers[7] += h;
            }
    cout<<"Output of SHA-512 Algorithm: "<<endl;
            for(int i=0 ; i<BUFFER_COUNT ; ++i)
            {
                    cout << setfill('0') << setw(16) << right << hex << buffers[i];
            }

            return 0;
}
```

Output:

```
input
Enter Text: This is SHA 512 algorithm which is used for encryption of the user names and passwords and is used in authentication
 process
Input: This is SHA 512 algorithm which is used for encryption of the user names and passwords and is used in authentication proc
ess
Plain text length = 1000
Plain text length after padding = 2048

Padded Input: 010101000110100001101001011100110010000001101001011100110010000001010011010010000100000100100000001101010011000100
11001000100000011000010110110001100111011011110111001001101001011101000110100001101101001000000111011101101000011010010110001101
10100000100000011010010111001100100000011101010111001101100101011001000010000001100110011011110111001000100000011001010110111001
10001101110010011110010111000001110100011010010110111101101110001000000110111101100110001000000111010001101000011001010010000001
11010101110011011001010111001000100000011011100110000101101101011001010111001100100000011000010110111001100100001000000111000001
10000101110011011100110111011101101111011100100110010001110011001000000110000101101110011001000010000001101001011100110010000001
11010101110011011001010110010000100000011010010110111000100000011000010111010101110100011010000110010101101110011101000110100101
10001101100001011101000110100101101111011011100010000001110000011100100110111101100011011001010111001101110011001000001000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00001111101000

Output of SHA-512 Algorithm:
d805bf7bfcd3c913e7db484cd00a1c14dd410a82bc83b849f283bde2d36a3149f186ee6a908b032e355fc1f3ce0a5d46d028e1597ef98d93b07806f4ad02ef65
```

Conclusion:

The SHA 512 can be used for authentication process.