Cryptography and Network Security

Name: Piyush Mhaske Batch: B3
PRN : 2019BTECS00089

Assignment 10 : SHA512

Problem statement : Implement SHA512

Theory : SHA 512 is encryption algorithm which encrypts the the given message to 512 bytes

```
#include <iostream>
#include <cstring>
#include <sstream>
#include <iomanip>
#pragma warning( push )
#pragma warning( disable : 4101)
// Your function
#pragma warning( pop )
#define BYTE8 (int64)0xFF
#define COUNT_WORDS 80
#define BLOCK_SIZE 1024
#define MESSAGE_LENGTH 128
#define ONE_BYTE 0x80

using namespace std;

class SHA512CryptoServiceProvider {
private:
    typedef unsigned long long int64;
    int64 _H[8]{};
    int64 _K[80] = {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};
    int64 *message{};
    static void InitialState(int64 H[]);
    int _word{}, _byte{};
    void AppendByte(unsigned char byte);
```

```
    void AppendWord(int64 word);
    static int64 CircularRightRotate(int64 num, int val);
    void ProcessBlock(const int64 *M, int64 *H);
    static int64 CH(int64 x, int64 y, int64 z);
    static int64 MAJ(int64 x, int64 y, int64 z);
    static int64 BSIG1(int64 x);
    static int64 BSIG0(int64 x);
    static int64 SSIG0(int64 x);
    static int64 SSIG1(int64 x);

public:
    SHA512CryptoServiceProvider();
    std::string Hashing(std::string message);
};

SHA512CryptoServiceProvider::SHA512CryptoServiceProvider()
{
    InitialState(_H);
}

void SHA512CryptoServiceProvider::InitialState(int64 H[])
{
    H[0] = 0x6a09e667f3bcc908;
    H[1] = 0xbb67ae8584caa73b;
    H[2] = 0x3c6ef372fe94f82b;
    H[3] = 0xa54ff53a5f1d36f1,
    H[4] = 0x510e527fade682d1;
    H[5] = 0x9b05688c2b3e6c1f;
    H[6] = 0x1f83d9abfb41bd6b;
    H[7] = 0x5be0cd19137e2179;
}
/*
 * CH, MAJ, SSIG0, SSIG1, BSIG0, BSIG1 - logical functions, each function
 * operates on 64-bit words, which are represented as x, y, and z.
 * The result of each function is a new 64-bit word.
*/
SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::CH(int64 x, int64 y, int64 z)
{
    return (x & y) ^ (~x & z);
}

SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::MAJ(int64 x, int64 y, int64 z)
{
    return (x & (y | z)) | (y & z);
}

SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::BSIG1(int64 x)
{
    return CircularRightRotate(x, 14) ^ CircularRightRotate(x, 18) ^ CircularRightRotate(x, 41);
}

SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::BSIG0(int64 x)
{
    return CircularRightRotate(x, 28) ^ CircularRightRotate(x, 34) ^ CircularRightRotate(x, 39);
}

SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::SSIG0(int64 x)
{
    return CircularRightRotate(x, 1) ^ CircularRightRotate(x, 8) ^ (x >> 7);
}

SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::SSIG1(int64 x)
{
    return CircularRightRotate(x, 19) ^ CircularRightRotate(x, 61) ^ (x >> 6);
}

void SHA512CryptoServiceProvider::AppendByte(unsigned char byte)
{
    message[_word] &= ~(BYTE8 << ((8 - 1 - _byte) * 8) );
    message[_word] |= ((int64)byte << ((8 - 1 - _byte) * 8) );
```

```
    _byte = _byte + 1;
    _word += _byte / 8;
    _byte = _byte % 8;
}

void SHA512CryptoServiceProvider::AppendWord(int64 word)
{
    message[_word++] = word;
}

SHA512CryptoServiceProvider::int64 SHA512CryptoServiceProvider::CircularRightRotate(int64 x, int n)
{
    return (x >> n) | (x << (64 - n));
}

void SHA512CryptoServiceProvider::ProcessBlock(const int64 *Message, int64 *H)
{
    int64 words[COUNT_WORDS];
    int64 state[8];

    for (int64 i = 0; i < 16; i++)
    {
        words[i] = Message[i];
    }

    for (int64 i = 16; i < COUNT_WORDS; i++)
    {
        words[i] = SSIG1(words[i - 2]) + words[i - 7] + SSIG0(words[i - 15]) + words[i - 16];
    }

    for(int64 i = 0 ; i < 8 ; i++)
    {
        state[i] = H[i];
    }

    for (int64 i = 0; i < COUNT_WORDS; i++)
    {
        int64 majRes   = MAJ(state[0], state[1], state[2]);
        int64 resFunc  = words[i] + _K[i] + state[7] + CH(state[4], state[5], state[6]) + BSIG1(state[4]);

        state[7] = state[6];
        state[6] = state[5];
        state[5] = state[4];
        state[4] = state[3] + resFunc;
        state[3] = state[2];
        state[2] = state[1];
        state[1] = state[0];
        state[0] = BSIG0(state[0]) + majRes + resFunc;
    }

    for(uint8_t i = 0 ; i < 8 ; i++)
    {
        H[i] += state[i];
    }
}
/*
 * Increases the total length of the padded message multiple of 1024.
 * Append one byte to message (0x80).
 * Add message length at the end of the block (128 bits).
 * Process each blocks and output final hash.
*/
std::string SHA512CryptoServiceProvider::Hashing(std::string inputMessage)
{
    const char* mess;
    int inputMessageLength = inputMessage.length();
    for (int i = 0; i < inputMessageLength; i++)
    {
        mess += inputMessage[i];
    }
```

```
    int64 intermediateLength, K, messageLength;

    intermediateLength = (int64)inputMessageLength * 8;
    messageLength = intermediateLength + 1 + MESSAGE_LENGTH;
    K = ((~messageLength + 1) % BLOCK_SIZE + BLOCK_SIZE) % BLOCK_SIZE;
    messageLength += K;

    message = (int64 *)malloc(messageLength / 8);
    _word = _byte = 0;

    for (int i = 0; i < inputMessageLength; i++)
        AppendByte(inputMessage[i]);
    AppendByte(ONE_BYTE);                                                    //Append one byte
    for (int i = 0; i < K / 8; i++)
        AppendByte(0);
    AppendWord(0);
    AppendWord(intermediateLength);

    for (int i = 0; i < (int)(messageLength / 64); i += 16)
    {
        ProcessBlock(message + i, _H);
    }

    std::stringstream is;
    is << std::setfill('0') << std::hex;

    for (int64 x : _H) {
        for (uint8_t i = 0; i < 64; i += 8)
        {
            is << std::setw(2) << (unsigned int) (((*(((int64 *) & x))) >> (64 - 8 - i)) & BYTE8);
        }
    }

    return is.str();                                                         //Return final hash
}

int main(){
    cout<<"Enter the message\n";
    string str; cin>>str;

    SHA512CryptoServiceProvider s;
    string hash = s.Hashing(str);

    cout<<hash;
}
```

Output :

Test case 1:

```
PS D:\Academics\Fourth Year\CNS Lab\cns lab> cd "d:\Academics\Fourth Year\CNS Lab\cns lab"
PS D:\Academics\Fourth Year\CNS Lab\cns lab> & .\"assignment10.exe"
Enter the message
this is the code
cc92838cf882ff6fe36570ab5eb859c88b6e9b5ca163eab4c83fdc52ce20d4322d187d74810a048d8413896115d2c74de29b6d9f05c4beb834e56fd372f6708f
PS D:\Academics\Fourth Year\CNS Lab\cns lab>
```

Test Case 2:

```
PS D:\Academics\Fourth Year\CNS Lab\cns lab> cd "d:\Academics\Fourth Year\CNS Lab\cns lab"
PS D:\Academics\Fourth Year\CNS Lab\cns lab> & .\"assignment10.exe"
Enter the message
walchand college of engineering sangli
01f732513bc8950d6bc06532d87248e6e81e8e946f039f67a827e24a1e933dd2eee9434d0ea61e772b153296ea7a00ac7da088145d2ea36cae1193385d054fff
```