**Cryptography, Network and Security**

Assignment 10

Implement the SIGNATURE SCHEME – Digital Signature Standard

Code:

#include <iostream>

#include <openssl/evp.h>

#include <openssl/sha.h>

#include <openssl/dsa.h>

#include <openssl/pem.h>

#include <openssl/err.h>

#include <cstring>

using namespace std;

*// Function to generate a DSA key pair*

DSA *\**generateDSAKeyPair()

{

    DSA *\**dsa = DSA\_new();

    if (!DSA\_generate\_parameters\_ex(dsa, 1024, NULL, 0, NULL, NULL, NULL))

    {

        cerr << "Error generating DSA parameters." << endl;

        DSA\_free(dsa);

        return nullptr;

    }

    if (!DSA\_generate\_key(dsa))

    {

        cerr << "Error generating DSA key pair." << endl;

        DSA\_free(dsa);

        return nullptr;

    }

    return dsa;

}

*// Function to sign a message using the DSA private key*

bool signMessage(DSA *\**dsa, *const* string *&*message, unsigned char *\*\**signature, unsigned int *\**signature\_len)

{

    EVP\_MD\_CTX *\**mdctx = EVP\_MD\_CTX\_new();

    if (!mdctx)

    {

        cerr << "Error creating EVP\_MD\_CTX." << endl;

        return false;

    }

*// Initialize the signing process with DSA private key*

    if (EVP\_SignInit(mdctx, EVP\_sha1()) <= 0)

    {

        cerr << "Error initializing signing." << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

*// Add the message to be signed*

    if (EVP\_SignUpdate(mdctx, message.c\_str(), message.length()) <= 0)

    {

        cerr << "Error during signing." << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

*// Allocate memory for the signature*

    \*signature = (unsigned char \*)OPENSSL\_malloc(EVP\_PKEY\_size(EVP\_PKEY\_new()));

    if (!\*signature)

    {

        cerr << "Error allocating memory for signature." << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

*// Finalize the signing process*

    if (EVP\_SignFinal(mdctx, \*signature, signature\_len, EVP\_PKEY\_new()))

    {

        EVP\_MD\_CTX\_free(mdctx);

        return true;

    }

    else

    {

        cerr << "Error finalizing signature." << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

}

*// Function to verify a signature using the DSA public key*

bool verifySignature(DSA *\**dsa, *const* string *&*message, unsigned char *\**signature, unsigned int signature\_len)

{

    EVP\_MD\_CTX *\**mdctx = EVP\_MD\_CTX\_new();

    if (!mdctx)

    {

        cerr << "Error creating EVP\_MD\_CTX for verification." << endl;

        return false;

    }

*// Initialize the verification process*

    if (EVP\_VerifyInit(mdctx, EVP\_sha1()) <= 0)

    {

        cerr << "Error initializing verification." << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

*// Add the message to be verified*

    if (EVP\_VerifyUpdate(mdctx, message.c\_str(), message.length()) <= 0)

    {

        cerr << "Error during verification." << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

*// Verify the signature*

    if (EVP\_VerifyFinal(mdctx, signature, signature\_len, EVP\_PKEY\_new()) == 1)

    {

        cout << "Signature verification successful!" << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return true;

    }

    else

    {

        cerr << "Signature verification failed!" << endl;

        EVP\_MD\_CTX\_free(mdctx);

        return false;

    }

}

int main()

{

*// Generate DSA key pair*

    DSA *\**dsa = generateDSAKeyPair();

    if (!dsa)

    {

        cerr << "Key generation failed." << endl;

        return 1;

    }

*// Input message to be signed*

    string message = "This is a test message for Digital Signature Standard.";

    cout << "Message to be signed: " << message << endl;

*// Sign the message*

    unsigned char *\**signature = nullptr;

    unsigned int signature\_len = 0;

    if (!signMessage(dsa, message, &signature, &signature\_len))

    {

        cerr << "Error signing the message." << endl;

        DSA\_free(dsa);

        return 1;

    }

    cout << "Message signed successfully." << endl;

*// Verify the signature*

    if (!verifySignature(dsa, message, signature, signature\_len))

    {

        cerr << "Signature verification failed." << endl;

        DSA\_free(dsa);

        return 1;

    }

*// Clean up*

    DSA\_free(dsa);

    if (signature)

    {

        OPENSSL\_free(signature);

    }

    return 0;

}