Instituto Politécnico Nacional
Escuela Superior de Cómputo
Cryptography

Practice 7: RSA (Digital Sign and Verification)

By:
Moreno Zárate Víctor Gibrán
Leonardo Manzano Salazar

Professor:
M. en C. NIDIA ASUNCIÓN CORTEZ DUARTE
December 2016

Contents

Problem

How the RSA works?

What are the main characteristics of the RSA?

How does this method helps us to share information between two entities?

How can we determinate if the real sender of a message is who claims to be?

RSA

RSA is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, had developed an equivalent system in 1973, but it was not declassified until 1997.

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message. Breaking RSA encryption is known as the RSA problem; whether it is as hard as the factoring problem remains an open question. [1]

Formulas

Parameter Generation

$$n_x = p_x q_x \qquad p \text{ and } q \text{ primes}$$

$$e_x \text{ coprime with } \varphi(n_x)$$

$$d_x = e_x^{-1} \bmod \varphi(n_x)$$

$$Public\ key = \{e_x, n_x\}$$

$$Private\ key = \{d_x\}$$

Usage

To cipher a message with addressee “x”. Use their public parameters given this formula:

$$c = m^{e_x} \bmod n_x$$

User X will use their private parameters to decrypt the received message, using this formula:

$$m' = c^{d_x} \, mod \, n_x$$
$$m' = {m^{e_x}}^{d_x} \, mod \, n_x$$
$$m' = m^{e_x d_x} \, mod \, n_x$$

Hash Functions

A cryptographic hash function is a special class of hash function that has certain properties which make it suitable for use in cryptography. It is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash function) which is designed to also be a one-way function, that is, a function which is infeasible to invert. The only way to recreate the input data from an ideal cryptographic hash function's output is to attempt a brute-force search of possible inputs to see if they produce a match. [2]

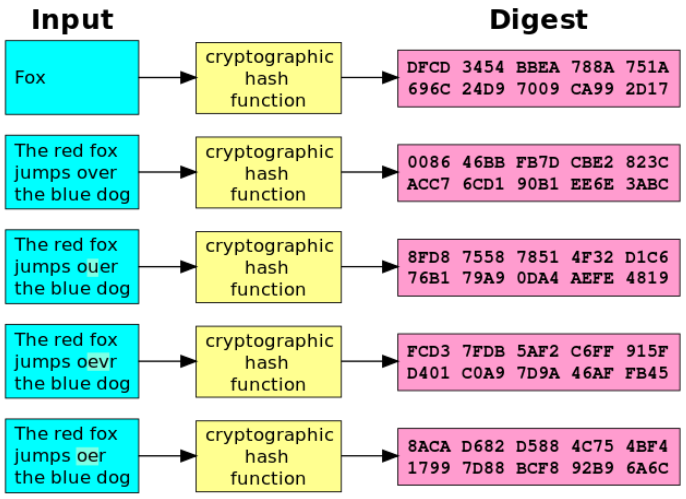

Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C

Illustration 1: *Example of a cryptographic hash function (SHA-1)*

SHA-1

In cryptography, SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function designed by the United States National Security Agency and is a U.S. Federal Information Processing Standard published by the United States NIST. SHA-1 produces a 160-bit (20-byte) hash value known as a message digest. A SHA-1 hash value is typically rendered as a hexadecimal number, 40 digits long. [3]

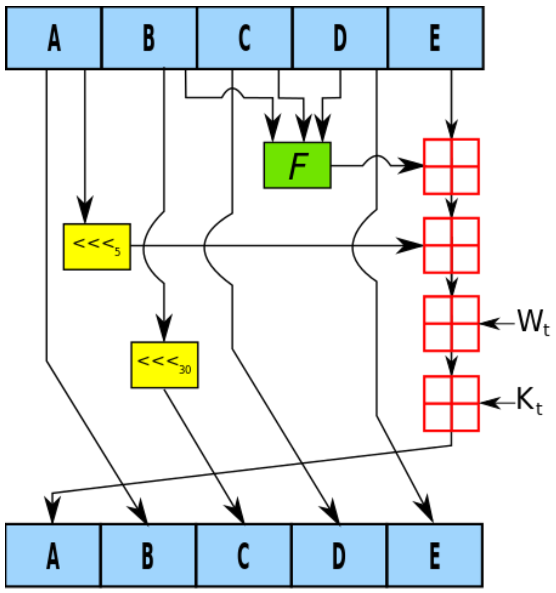
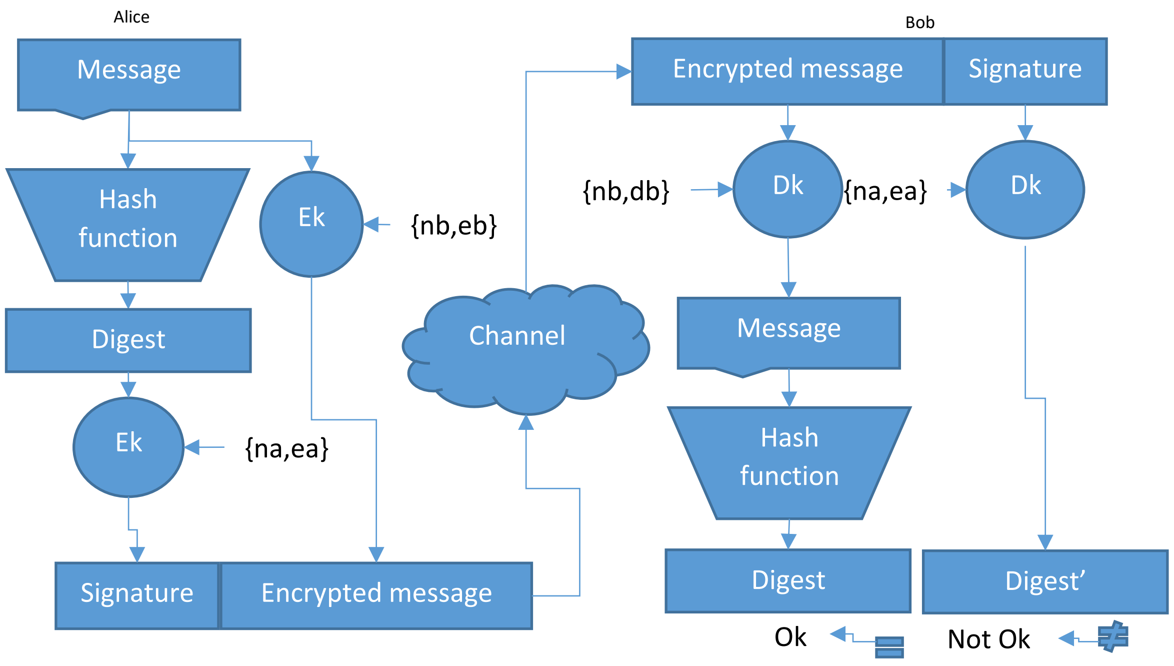


Illustration 2: *Diagram of the SHA-1 hash function.*

The Illustration 2 show one iteration within the SHA-1 compression function:
A, B, C, D and E are 32-bit words of the state;
F is a nonlinear function that varies;
⋘n denotes a left bit rotation by n places;
n varies for each operation;
Wt is the expanded message word of round t;
Kt is the round constant of round t;
⊞ denotes addition modulo 232.

Digital Signature and Verification Scheme



Hypothesis

A possible solution to the problem is to write a computer program to analyse the behaviour of the RSA, this program will be able to encrypt and decrypt a text using this technique. It also will work over a network.

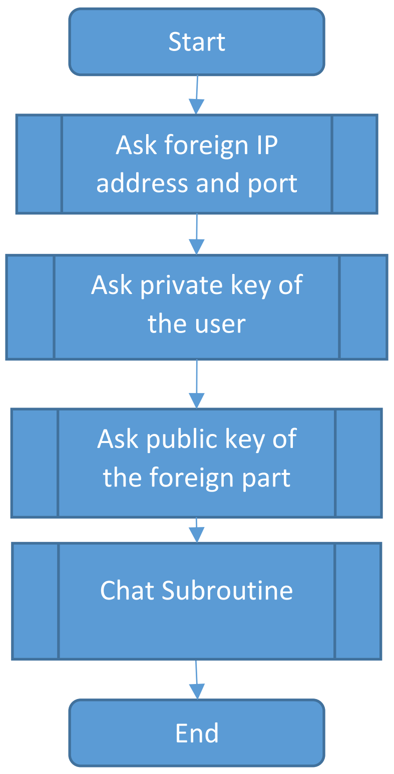
If we program this software correctly, we will see how the character data is send and decrypted using this scheme, also it will provide the digital sign.

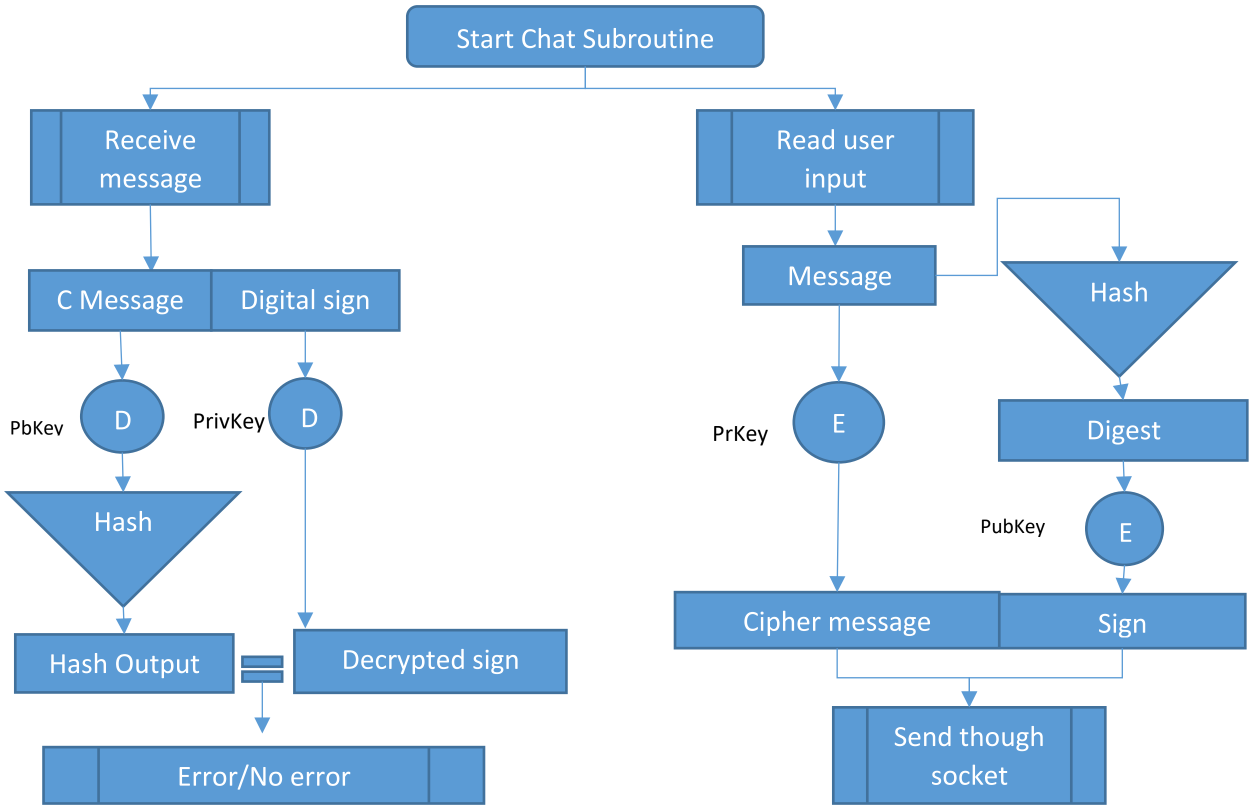
Software (libraries, package, tools)

To do this practice, we used:

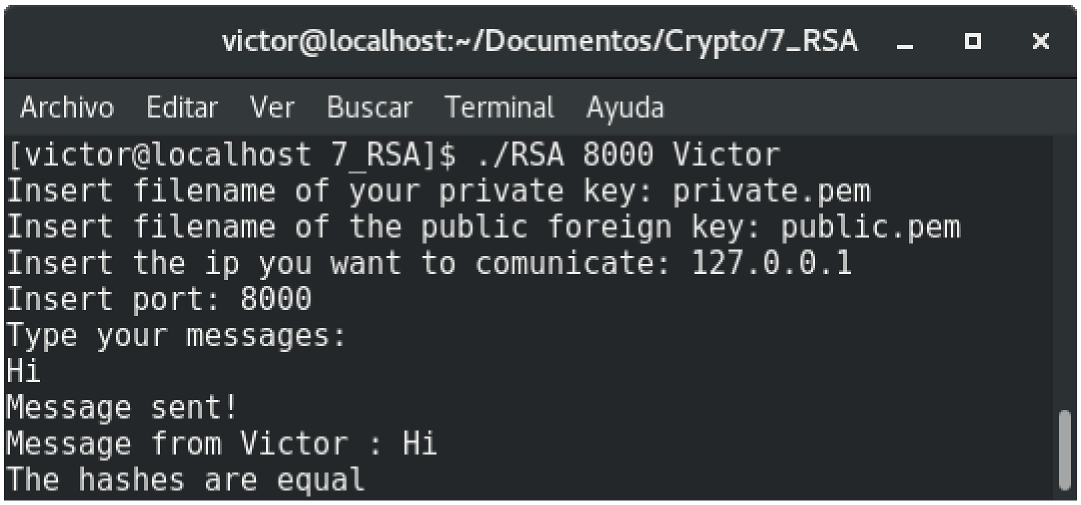
- Personal Computer
- Linux Operating System
- Text Editor
- GNU C++ Compiler
- Access to a computer network

Procedure


Start
Ask foreign IP address and port
Ask private key of the user
Ask public key of the foreign part
Chat Subroutine
End

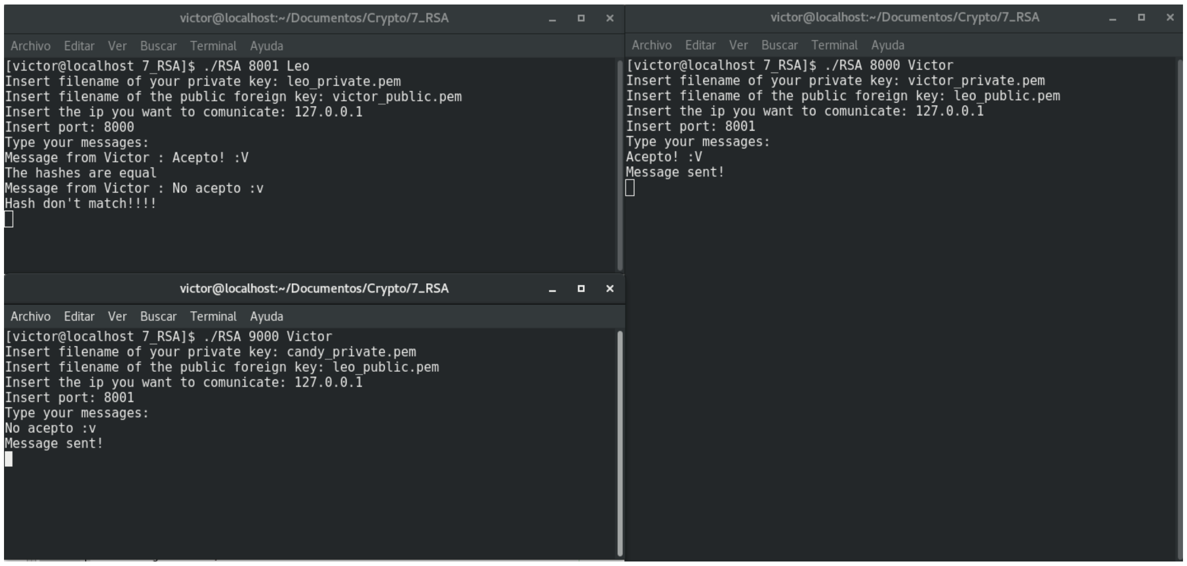


Results (Data)


victor@localhost:~/Documentos/Crypto/7_RSA
Archivo Editar Ver Buscar Terminal Ayuda

```
[victor@localhost 7_RSA]$ ./RSA 8000 Victor
Insert filename of your private key: private.pem
Insert filename of the public foreign key: public.pem
Insert the ip you want to comunicate: 127.0.0.1
Insert port: 8000
Type your messages:
Hi
Message sent!
Message from Victor : Hi
The hashes are equal
```

Figure 1: Starting a Chat Room and receiving a message


victor@localhost:~/Documentos/Crypto/7_RSA
Archivo Editar Ver Buscar Terminal Ayuda

```
[victor@localhost 7_RSA]$ ./RSA 8001 Leo
Insert filename of your private key: leo_private.pem
Insert filename of the public foreign key: victor_public.pem
Insert the ip you want to comunicate: 127.0.0.1
Insert port: 8000
Type your messages:
Message from Victor : Acepto! :V
The hashes are equal
Message from Victor : No acepto :v
Hash don't match!!!!
```

```
[victor@localhost 7_RSA]$ ./RSA 8000 Victor
Insert filename of your private key: victor_private.pem
Insert filename of the public foreign key: leo_public.pem
Insert the ip you want to comunicate: 127.0.0.1
Insert port: 8001
Type your messages:
Acepto! :V
Message sent!
```

```
[victor@localhost 7_RSA]$ ./RSA 9000 Victor
Insert filename of your private key: candy_private.pem
Insert filename of the public foreign key: leo_public.pem
Insert the ip you want to comunicate: 127.0.0.1
Insert port: 8001
Type your messages:
No acepto :v
Message sent!
```

Figure 2: Simulating a phishing attack by Candy.

Conclusions

In this practice, we accepted the hypothesis because we could send messages and detect when the person wasn't who claims to be, using the RSA and SHA-1 for authentication and digital sign.

We could send and receive data encrypted with this scheme and this is useful because we used real RSA key and real Hash functions, making this chat cryptographically useful.

Reference

[1] "RSA", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/RSA_(cryptosystem) . [Accessed: 14- Dec- 2016].

[2] "Hash function", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/Hash_function . [Accessed: 14- Dec- 2016].

[3] "SHA-1", *Wikipedia*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/SHA-1 . [Accessed: 14- Dec- 2016].

Code

```
//C++
#include <iostream>
#include <cstring>
#include <fstream>
//POSIX
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

std::string getFileData(string filename);
```

```cpp

int main(int argc, char *argv[]){
    // Uso: ./RSA local_port name
    if(argc!=3){
        cout<<"Usage: ./RSA local_port your_name"<<endl;
        exit(0);
    }
    string name=string(argv[2]);
    SocketDatagrama socket(atoi(argv[1]));
    string filename="";
    cout<<"Insert filename of your private key: ";
    getline(cin,filename);
    string myprivateKey=getFileData(filename);
    cout<<"Insert filename of the public foreign key: ";
    getline(cin,filename);
    string foreignpublicKey=getFileData(filename);
    cout<<"Insert the ip you want to comunicate: ";
    string ip;
    getline(cin,ip);
    cout<<"Insert port: ";
    string port;
    getline(cin,port);

    pid_t pid;
    pid=fork();
    if(pid>0){//Main process, sender
        cout<<"Type your messages: "<<endl;
        string out_message="";
        struct RSAmessage out_c;
        int encrypted_length=0;
        unsigned char digest[SHA_DIGEST_LENGTH];
        PaqueteDatagrama *pack;

        while(true){
            getline(cin,out_message);
            //Encrypt message
            if((encrypted_length=public_encrypt((unsigned char *)out_message.c_str(),out_message.size(),(unsigned char *)foreignpublicKey.c_str(),out_c.data))== -1){
                printLastError("Public Encrypt failed");
                exit(0);
            }
            //Hash function
            SHA((unsigned char *)out_message.c_str(), out_message.size(),digest);
            //Sing Hash
            if((encrypted_length=private_encrypt(digest,strlen((const char *)digest),(unsigned char *)myprivateKey.c_str(),out_c.sign)) == -1){
                printLastError("Private Encrypt failed");
                exit(0);
            }
            //Send message to peer
            strcpy(out_c.name,name.c_str());
            out_c.bytes=out_message.size();
            pack=new PaqueteDatagrama((char*)&out_c, sizeof out_c, ip,stoi(port));
            socket.envia(*pack);
            delete pack;
            cout<<"Message sent!"<<endl;
        }

    }else if(pid==0){//Child process, receiver
        struct RSAmessage in_c;
        PaqueteDatagrama pack=PaqueteDatagrama(sizeof in_c);
        unsigned char decrypted[2048/8];
        unsigned char decrypted_hash[2048/8];
        char plain_text[2048/8];
        unsigned char digest[SHA_DIGEST_LENGTH];
        while(true){
            socket.recibe(pack);
            memcpy(&in_c,pack.obtieneDatos(),sizeof in_c);
            //Message
            int decrypted_length = private_decrypt(in_c.data,2048/8,(unsigned char *)myprivateKey.c_str(), decrypted);
            if(decrypted_length == -1){
                //printLastError("Private Decrypt failed ");
            }
            memset(plain_text,'\0',2048/8);
            memcpy(plain_text,decrypted,in_c.bytes);
            cout<<"Message from "<<in_c.name<<" : "<<plain_text<<endl;

            //Sign
            SHA((const unsigned char*)plain_text,in_c.bytes,digest);
            decrypted_length = public_decrypt(in_c.sign,2048/8,(unsigned char*)foreignpublicKey.c_str(), decrypted_hash);
            if(decrypted_length == -1){
                //printLastError("Public Decrypt failed");
            }
            memset(decrypted_hash+SHA_DIGEST_LENGTH,'\0',2048/8 - SHA_DIGEST_LENGTH);
            //Comparing hashes
            if(strcmp((const char*)decrypted_hash,(const char*)digest)!=0){
                cout<<"Hash don't match!!!!"<<endl;
            }else{
                cout<<"The hashes are equal"<<endl;
            }
               }

       }else{//Error
           cerr << "Error in fork: "<<strerror(errno) << endl;
           exit(0);
       }

       wait(0);
       return 0;

   }


   std::string getFileData(string filename){
       string data="";
       streampos size;
       ifstream file (filename, ios::in|ios::binary|ios::ate);
       size = file.tellg();
       char *buffer=new char [size];
   file.seekg (0, ios::beg);
   file.read (buffer, size);
   file.close();
   data=string(buffer);
   delete[] buffer;
       return data;
   }
```