Network Security

Project Specification Document

Group 14 - Implementing Fiat-Shamir protocol

Group Members

IIT2019021	**Medha Balani**
IIT2019027	**Vidushi Pathak**
IIT2019032	**Aarushi**
IIT2019035	**Aaryan Bhardwaj**
IIT2019036	**Jyotika Bhatti**
IIT2019039	**Hardik Bajaj**

Instructor: Dr. J. Kokila Ma'am

Abstract

The zero-knowledge proof of knowledge is a process in which the prover demonstrates possession of knowledge without revealing any computational information. There are such notions for the identification schemes, in which the claimant proves the above in order to get access, without passwords.
In this paper we have discussed a way in which we can implement the fiat shamir protocol in c++ language. We have also included necessary steps along with the screenshots to help visualise how the functioning of fiat shamir protocol takes place.

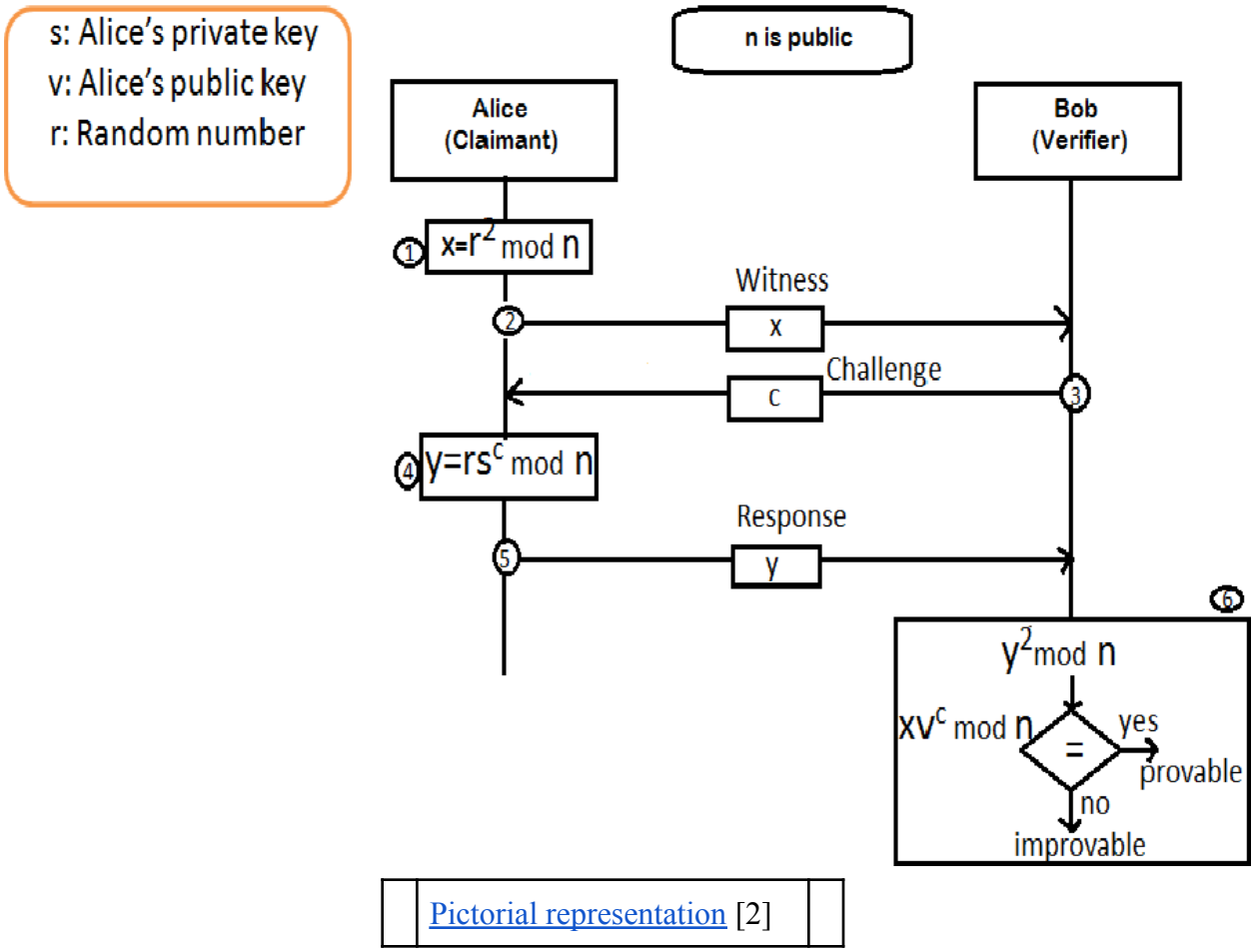
Introduction

Password authentication is very common but it is very much vulnerable to attacks, such as eavesdropping, guessing passwords, stolen passwords, etc. there are some ways in which the user can convince the verifier that he knows the password without actually telling him the exact password. Here the password could be anything , any real password or some numbers that only the real user might know. One of these techniques is fiat shamir protocol.
Fiat-shamir protocol is a type of zero knowledge protocol. Here the claimant need not disclose any secret keys, or anything that might endanger the confidentiality of the claimant. This is a better way of securing because here the eavesdropper would not know anything about the secret like password or anything, so they cannot steal it anyway. It is just that he has to prove to the verifier that he knows the secret every time the verifier asks to. If at any time the claimant is not able to prove himself the session would be terminated.

Working

This protocol works between claimant and verifier as :
Suppose the claimant is Alice and the verifier is Bob, Alice has a private key(s), a public key(v) and a random number as r.
N is a public number which is very large made by multiplying p and q which are other 2 large prime numbers.
Step 0: r is a random number from (0 to n-1) chosen by Alice, also known as commitment.
Step 1: Alice sends x , such that x = r^2%n as a witness.
Step 2: Now Bob sends a challenge c to Alice which is either 0 or 1 randomly.
Step 3: Alice now calculates y= r*s^c %n and sends y to Bob as a response.
Step 4: Now Bob calculates, y^2 mod n, if this is equal to xv^c mod n then Alice passed this test , otherwise Alice failed.

This process is repeated a certain number of times, and Alice has to pass all the times, if fails any test the session will not be established.



Pictorial representation [2]

Proof: x was r^2 initially, and s is Alice's private key, and v is Alice's public key

$$y^2 \%n = xv^c \%n$$
$$y^2 = xv^c$$
$$r^2 (s^2)^c = xv^c$$

The implementation details with screenshots wherever required:

Alice (Claimant Code)

```cpp
///-------------------------Alice---------------------

#include<bits/stdc++.h>
using namespace std;


int32_t GetRandomNumber(const int32_t min, const int32_t max)
{
    return (rand() % (max - min)) + min;
}
int32_t getCoprime(int32_t n)
{
    int32_t coprime;

    do {
        coprime = GetRandomNumber(1, n);
    }
    while (__gcd(n, coprime) != 1);

    return coprime;
}

int main()
{
    srand(time(NULL));
    std::cout<<"HI ! I AM ALICE (Claimant) \n\n";

    cout<<"Enter values of p and q: "<<"\n\n";

    int32_t p ;
    int32_t q ;
    std::cin>> p>>q;

    int32_t n = p * q;

    std::cout<<"n = " << p << "*" << q << " = " << n << " (public)"<<"\n\n";

    int32_t s = getCoprime(n);
    std::cout<<"s = " << s << " (your[Alice] secret)"<<"\n\n";

    int32_t v = (s * s) % n;
    std::cout<<"v = " << v << " give it to Bob (public: s*s)"<<"\n\n";

    int32_t r = GetRandomNumber(1, n);
    std::cout<<"r = " << r << " (A's private)"<<"\n\n";

    int32_t x = (r * r) % n;
    std::cout<<"x = " << x<<"\n\n";

    int32_t e ;
    std::cout<<"Enter e challenge from Bob "<<"\n\n";
    std::cin>>e;

    int32_t y ;
    y = r * pow(s, e) ;
    std::cout << "Calculated y = "<<y;

    return 0;
}
```

Bob (verifier's code):

```cpp
//---------------------BOB--------------------------

#include<bits/stdc++.h>
using namespace std;

#define USER_INPUT

int32_t GetRandomNumber(const int32_t min, const int32_t max)
{
    return (rand() % (max - min)) + min;
}

int32_t GetRandomSign()
{

    if((rand()%2)&1)
    return -1;
    else
    return 1;
}

class NumberGenerator
{
    int32_t min = 100;
    int32_t max = 10000;
    std::vector<int> primeNumbers;

    void GeneratePrimeNumbers(int32_t min, int32_t max)
    {
        primeNumbers.clear();

        for (int32_t i = min; i != max + 1; i++) {
            int32_t j;
            for (j = 2; j < i; j++) {
                if (i % j == 0) {
                    break;
                }
            }
            if (i == j) {
                primeNumbers.push_back(i);
            }
        }
    }


public:
    NumberGenerator(int32_t min_ = 100, int32_t max_ = 10000)
    {
        min = min_;
        max = max_;
    }
    int32_t GetLargePrimeNumber()
    {
        GeneratePrimeNumbers(min, max);

        int32_t index = GetRandomNumber(0, primeNumbers.size());
        return primeNumbers[ index ];
    }
};


```

```
int32_t getCoprime(int32_t n)
{
    int32_t coprime;

    do {
        coprime = GetRandomNumber(1, n);
    }
    while (__gcd(n, coprime) != 1);

    return coprime;
}

int main()
{
    srand(time(NULL));

    std::cout<<"HI ! I AM BOB(Verifier) \n\n";
    auto numGen = NumberGenerator(1, 30);

    int32_t p = numGen.GetLargePrimeNumber();
    int32_t q = numGen.GetLargePrimeNumber();

    int32_t n = p * q;
    cout<<"Prime Number 1 is p = "<<p<<"\n\n";
    cout<<"Prime Number 2 is q = "<<q<<"\n\n";
    std::cout<<"n = " << p << "*" << q << " = " << n << "  (public)"<<"\n\n";

    int32_t v ;
    std::cout<<"Enter v   (your public key)"<<"\n\n";
    std::cin>>v;


    int32_t x;
    std::cout<<"Enter value of x = "<<"\n\n";
    std::cin>>x;

    int32_t e = GetRandomNumber(0, 2);
    std::cout<<"e = " << e<<"\n\n";

    int32_t y = 0;

#ifdef USER_INPUT
    std::cout << "Give value of y, such that y = r * s^e: ";
    std::cin >> y;
#else
    y = r * pow(s, e);
    y = y % n;
    P("y = " << y);
#endif

    int32_t y_sq_mod_n = (y * y) % n;
    std::cout<<"y^2 mod n = " << y_sq_mod_n<<"\n\n";

    int32_t test = (x * (int)pow(v, e)) % n;
    std::cout<<"Tested Value = " << test<<"\n";

    if (y_sq_mod_n == test) {
        std::cout<<"YES ! ALICE IS AUTHENTICATED TO LOGIN"<<"\n";
    } else {
        std::cout<<"NO! ALICE IS NOT AUTHENTICATED TO LOGIN"<<"\n";
    }

    return 0;
}
```

Output compiled from both the codes:

```
HI ! I AM BOB(Verifier)

Prime Number 1 is p = 3

Prime Number 2 is q = 17

n = 3*17 = 51 (public)

Enter v  (your public key)

13
Enter value of x =

18
e = 1

Give value of y, such that y = r * s^e: 264
y^2 mod n = 30

Tested Value = 30
YES ! ALICE IS AUTHENTICATED TO LOGIN


...Program finished with exit code 0
Press ENTER to exit console.
```

```
HI ! I AM ALICE (Claimant)

Enter values of p and q:

3
17
n = 3*17 = 51 (public)

s = 8 (your[Alice] secret)

v = 13 give it to Bob (public: s*s)

r = 33 (A's private)

x = 18

Enter e challenge from Bob

1
Calculated y = 264

...Program finished with exit code 0
Press ENTER to exit console.
```

Advantages and Drawbacks:

Advantages of Zero-Knowledge Protocols:

- It is secured because it doesn't require someone to reveal a secret.
- It is simple.

Disadvantages of Zero-Knowledge Protocols:

- It is still not perfect because the Intruder can still intercept the message (i.e. messages to the Verifier might be modified or destroyed).
- Limited use because translation might be necessary if secret is not a number.
- It is lengthy, it takes a lot of time to compute.

References:

[1]https://link.springer.com/article/10.1007/BF02351717

[2]Kim, Hyungjoo et al. "Zero-Knowledge Authentication for Secure Multi-cloud Computing Environments." *CSA/CUTE* (2015).

[3] Kim, Minjin et al. "Design of Authentication Protocol Based on Distance-Bounding and Zero-Knowledge for Anonymity in VANET." *CSA/CUTE* (2015).

[4] http://ethesis.nitrkl.ac.in/5755/1/110CS0371-2.pdf