# Authentication Protocol Implementation

## Technology used

Java

## Notation used

{X}k : Encryption of X using key ‘k’.

{X}k-1 : Decryption of X using key ‘k’.

P → Q : M represents P sends message M to Q.

P : … represents some computation step in P.

## Symmetric Basic Protocol

### The protocol

P : Create message m = “I am P”

: Compute m′ = { m, Q}k

P → Q : m, m′

Q : Verify { m, Q}k = m′

: If equal then accept; otherwise authentication failed.

### Implementation

#### Project Structure

symmetric\_basic\_protocol (Project name)

* + cryptotool (Package)
    - CaeserCipher (Class)
  + host (Package)
    - Client (Class)
    - Server (Class)
    - Intruder (Class)

#### Description

##### CaeserCipher

This class contains the methods for encryption and decryption of a string using Caesar’s cipher technique.

###### Encryption algorithm

Step 1: Read string **str** and key **k**.

Step 2: For each character ch in str do

Step 3: If ch is an alphabet then

Step 4: ch = (ch + k) mod 26 // Rotate right the character k positions.

Step 5: Else if ch is a digit then

Step 6: ch = (ch + k) mod 10

[End if]

Step 7: Append ch to new\_str

Step 8: Repeat until end of str reached

Step 9: Return new\_str

###### Decryption algorithm

Step 1: Read string **str** and key **k**.

Step 2: For each character ch in str do

Step 3: If ch is an alphabet then

Step 4: If ch is in lower case then

Step 5: end\_letter = ‘z’

Step 6: Else

Step 7: end\_letter = ‘Z’

[End if]

Step 8: ch = endLetter - ((endLetter - ch + k) mod 26) // Rotate left the character k positions.

Step 9: Else if ch is a digit then

Step 10: ch = ('9' - (('9' - ch + k) mod 10))

[End if]

Step 11: Append ch to new\_str

Step 12: Repeat until end of str reached

Step 13: Return new\_str

###### Program

package cryptotool;

public class CaeserCipher {

    public static String encrypt(String str, int key)

    {

        StringBuffer sb = new StringBuffer();

        int len = str.length();

        char startLetter;

        for(int i=0; i<len; i++)

        {

            char ch = str.charAt(i);

            if(Character.isAlphabetic(ch))

            {

                startLetter = (Character.isLowerCase(ch) ? 'a' : 'A');

                ch = (char)(((ch + key - startLetter) % 26) + startLetter);

            }

            else if(Character.isDigit(ch))

                ch = (char)(((ch + key - 48) % 10) + 48);

            sb.append(ch);

        }

        return sb.toString();

    }

    public static String decrypt(String str, int key)

    {

        StringBuffer sb = new StringBuffer();

        int len = str.length();

        char endLetter;

        for(int i=0; i<len; i++)

        {

            char ch = str.charAt(i);

            if(Character.isAlphabetic(ch))

            {

                endLetter = (Character.isLowerCase(ch) ? 'z' : 'Z');

                ch = (char)(endLetter - ((endLetter - ch + key) % 26));

            }

            else if(Character.isDigit(ch))

                ch = (char)('9' - (('9' - ch + key) % 10));

            sb.append(ch);

        }

        return sb.toString();

    }

}

##### Server

This class is responsible for the server side process.

###### Working functionality

* We have to run the class from a new terminal.
* When executing, the server process will create a ServerSocket object which will wait for a client to connect.
* By default it will listen to the port 9001 at localhost.
* When a client request for a connection, ServerSocket will return a Socket object which will work as communicate end point for the server to communicate with the newly connected client.
* An input and output stream will be instantiated over the socket to receive and send objects (mainly String) respectively.
* Server will also ask to set a “key” which will act as the private key for the authentication.
* After a client connects to the server it will wait for a message packet.
* Getting the packet it will extract the original message(m) and its encrypted version(m′) from the packet and will verify if they are encrypted with the authenticate key or not i.e. if {m, server\_port}k = m′ ?
* If the verification succeeds then Server will send a response to the client as “Authentication successful” otherwise “Authentication failed”.

###### Program

package host;

import java.net.ServerSocket;

import java.net.Socket;

import cryptotool.CaeserCipher;

import java.io.\*;

public class Server {

    private ServerSocket srvSoc;

    private Socket soc;

    private ObjectInputStream socIn;

    private DataOutputStream socOut;

    private int key, port;

    public Server(int port, int key)

    {

        try

        {

            this.port = port;

            this.key = key;

            this.srvSoc = new ServerSocket(port);

            System.out.println("Server started at port: " +

            port + "\nWaiting for the client...");

            this.soc = srvSoc.accept();

            System.out.println("Connection established");

            socIn = new ObjectInputStream(soc.getInputStream());

            socOut = new DataOutputStream(soc.getOutputStream());

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

    public static void main(String[] args) {

        try

        {

            String msg;

            String[] transMsg = new String[2];

            BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));

            System.out.print("Enter the key: ");

            Server srv = new Server(9001, Integer.parseInt(userIn.readLine()));

            transMsg = (String[]) srv.socIn.readObject();

            System.out.println("Incomming packet from client contains:\n" +

            transMsg[0] + "\n" + transMsg[1]);

            msg = transMsg[0] + String.valueOf(srv.port);

            msg = CaeserCipher.encrypt(msg, srv.key);

            if(msg.equals(transMsg[1]))

            {

                srv.socOut.writeUTF("Authentication successful");

                System.out.println("Authentication successful");

            }

            else

            {

                srv.socOut.writeUTF("Authentication unsuccessful");

                System.out.println("Authentication unsuccessful");

            }

            srv.socOut.close();

            srv.socIn.close();

            srv.soc.close();

            userIn.close();

            System.out.println("Communication terminated");

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

}

##### Client

This class is responsible for client process.

###### Working functionality

* We have to run the class from a new terminal in order to create a new client process.
* After creation the client will first ask the port number of the server.
  + As because we are running both the client and the server on the same computer thus only the port address will be different for the processes, IP address will be same as “localhost”.
* On getting the correct port number Client will send request to the Server and instantiate a Socket object.
* This Socket object will act as a communication end point for the Client.
* Input and output stream will be created over the socket to receive and send messages.
* After this the user will be asked the private key.
* Then the user will be asked to enter a message (m).
* Then it will concatenate the port number of the server to the message m and encrypt the string with the key to create m′.
* After that it will sent {m, m′} to the server.
* Finally if the verification at server side succeeds then our client will get the response “Authentication successful” otherwise “Authentication failed”.

###### Program

package host;

import java.net.Socket;

import java.io.\*;

import cryptotool.CaeserCipher;

public class Client {

    private Socket soc;

    private DataInputStream socIn;

    private ObjectOutputStream socOut;

    private int port, key;

    public Client(int port, int key)

    {

        try

        {

            this.port = port;

            this.key = key;

            System.out.println("Client started");

            this.soc = new Socket("localhost", port);

            socIn = new DataInputStream(soc.getInputStream());

            socOut = new ObjectOutputStream(soc.getOutputStream());

            System.out.println("Connection established with port: " + port);

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

    public static void main(String[] args) {

        try

        {

            BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));

            int srvPort, srvKey;

            System.out.print("Enter the port address of the server: ");

            srvPort = Integer.parseInt(userIn.readLine());

            System.out.print("Enter the key(should be known for authentic client): ");

            srvKey = Integer.parseInt(userIn.readLine());

            Client clt = new Client(srvPort, srvKey);

            String[] transMsg = new String[2];

            String response;

            System.out.print("Enter the message: ");

            transMsg[0] = userIn.readLine();

            transMsg[1] = transMsg[0] + clt.port;

            transMsg[1] = CaeserCipher.encrypt(transMsg[1], clt.key);

            clt.socOut.writeObject(transMsg);

            response = clt.socIn.readUTF();

            System.out.println(response);

            clt.socOut.close();

            clt.socIn.close();

            clt.soc.close();

            userIn.close();

            System.out.println("Communication terminated");

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

}

##### Intruder

This class creates a client process which will act as an intruder and try to trick the authentication process.

###### Working functionality

* This class will execute similarly like the client but it will ask two messages from the user.
* If the user enters m (a message) and its corresponding m′ (encrypted message) encrypted with correct key then authentication will succeed otherwise fail.
* This shows the vulnerability of this authentication protocol that even a client with proper authentication packet i.e. {m, m′} can masquerade himself as authenticated user without knowing the private key.

###### Program

package host;

import java.net.Socket;

import java.io.\*;

public class Intruder {

    public static void main(String[] args) {

        int port;

        String packet[] = new String[2];

        Socket soc;

        ObjectOutputStream out;

        DataInputStream in;

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        try

        {

            System.out.print("Enter the port of server: ");

            port = Integer.parseInt(br.readLine());

            System.out.print("Enter a message: ");

            packet[0] = br.readLine();

            System.out.print("Enter the encrypted version of the message: ");

            packet[1] = br.readLine();

            soc = new Socket("localhost", port);

            System.out.println("Successfully connected");

            out = new ObjectOutputStream(soc.getOutputStream());

            in = new DataInputStream(soc.getInputStream());

            out.writeObject(packet);

            System.out.println(in.readUTF());

            soc.close();

            in.close();

            out.close();

            System.out.println("Connection terminated");

        }

        catch(Exception e)

        {

            e.getStackTrace();

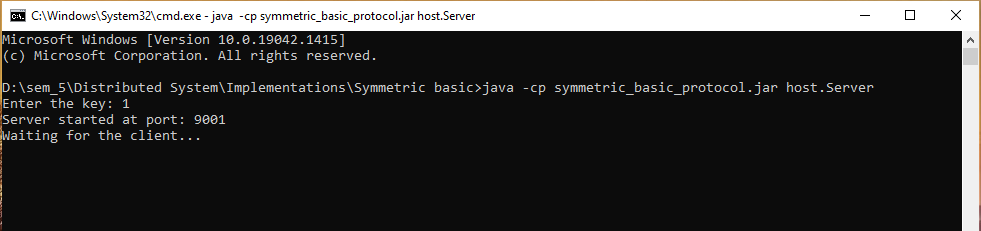
        }

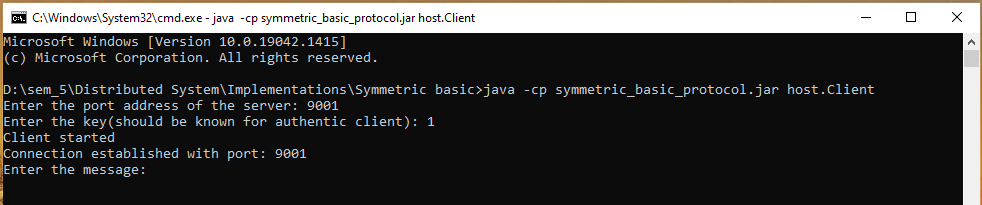
    }

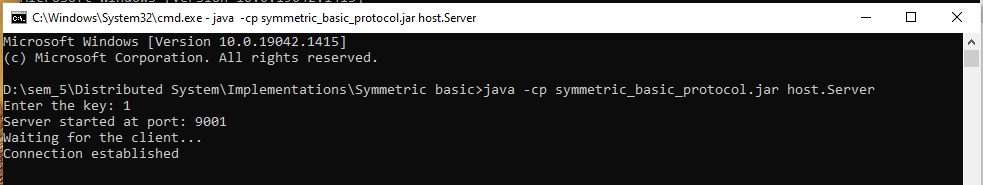
}

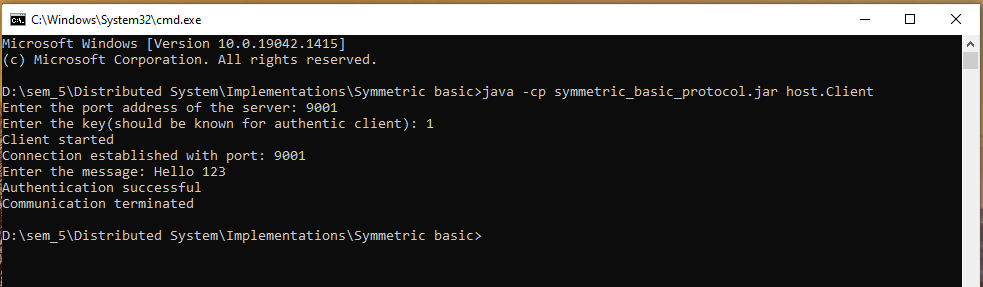
### Output

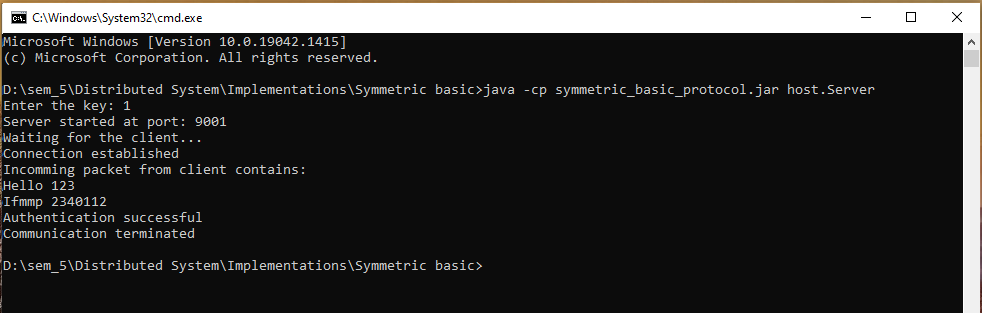
#### Normal output



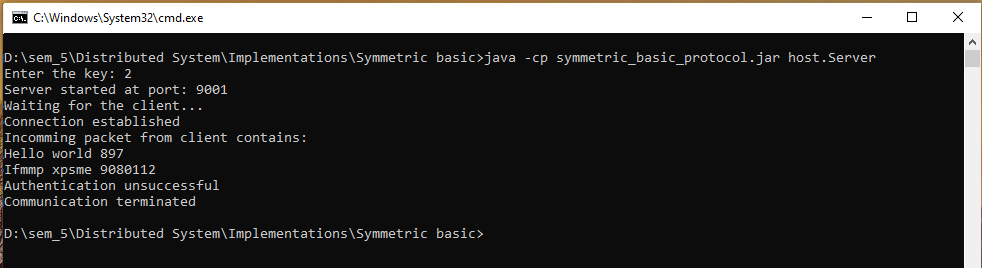


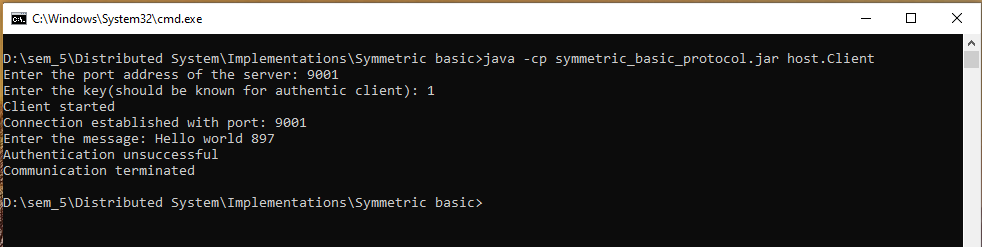




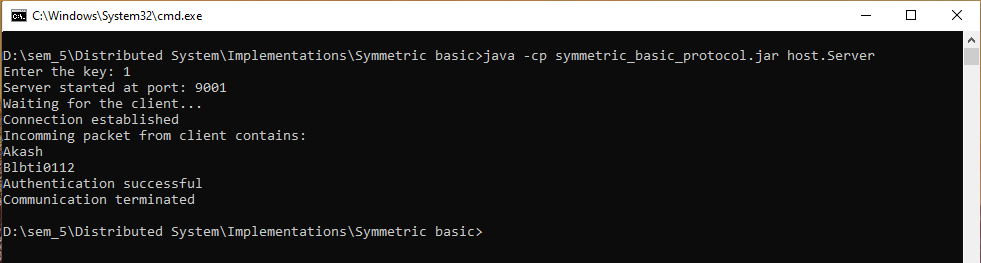


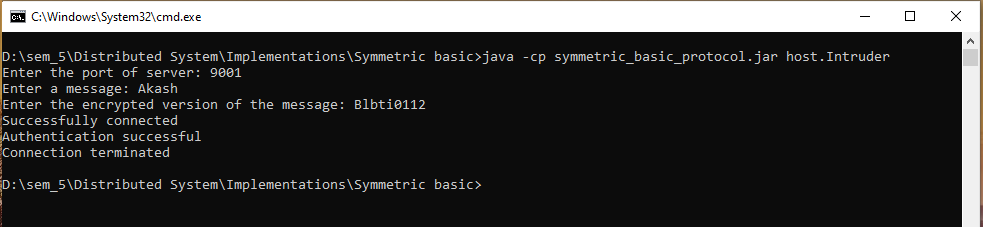
#### Unsuccessful output





#### Intruder output





### Considerations

* The program is designed to run both the client and the server on same computer, thus we kept the IP address to localhost and assigned different port address to client and server.
* If we want we can run client and server on two different computer, in that case we have to consider “IP address : port number” as the identity of each process (Client and Server).
* We have used java’s socket programming construct for the networking purpose.
* Java sockets use TCP(Transmission Control Protocol) as networking protocol.
* The private key is asked to the user because here we have assumed that it is already known to the authenticated user. That is why it is not shared before the authentication process.

## Modified protocol with nonce

### The protocol

P → Q: “I am P”

Q: Generate nonce n

Q → P: n

P: compute m′ = { P, Q, n}k

P → Q: m′

Q: verify {P, Q, n}k = m′

: If equal then accept otherwise authentication fails.

### Implementation

#### Project structure

symmetric\_nonce\_protocol (Project name)

* + cryptotool (Package)
    - CaeserCipher (Class)
  + host (Package)
    - Client (Class)
    - Server (Class)

#### Description

##### CaeserCipher:

Same as previous.

##### Server

###### Working functionality

* The differences between this Server and the previous Server (symmetric basic Server) is that
  + After getting request from the client it generates a random number ‘n’ also called nonce.
  + Then server sends the nonce.
  + In this protocol authentication packet contains the identity of the client, identity of the server and the nonce encrypted with the private key.

###### Program

package host;

import java.io.\*;

import java.net.ServerSocket;

import java.net.Socket;

import cryptotool.CaesarCipher;

public class Server {

    private ServerSocket srvSoc;

    private Socket soc;

    private DataInputStream socIn;

    private DataOutputStream socOut;

    private int key, port;

    public Server(int port, int key)

    {

        try

        {

            this.port = port;

            this.key = key;

            this.srvSoc = new ServerSocket(port);

            System.out.println("Server started at port: " +

            port + "\nWaiting for the client...");

            this.soc = srvSoc.accept();

            System.out.println("Connection established");

            socIn = new DataInputStream(soc.getInputStream());

            socOut = new DataOutputStream(soc.getOutputStream());

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

    public static void main(String[] args) {

        try

        {

            String request, verCode;

            BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));

            System.out.print("Enter the key: ");

            Server srv = new Server(9001, Integer.parseInt(userIn.readLine()));

            request = srv.socIn.readUTF();

            System.out.println("Got the request from: " + request);

            int nonce = (int)(Math.random() \* 10000);

            System.out.println("Generated nonce = " + nonce);

            srv.socOut.writeUTF(String.valueOf(nonce));

            System.out.println("Nonce sent");

            verCode = srv.socIn.readUTF();

            System.out.println("Packet received from client: " + verCode);

            String str = request + String.valueOf(srv.port) + String.valueOf(nonce);

            System.out.println("Constructed packet: " + str);

            str = CaesarCipher.encrypt(str, srv.key);

            System.out.println("Encrypted packet: " + str);

            if(str.equals(verCode))

            {

                srv.socOut.writeUTF("Authentication successful");

                System.out.println("Authentication successful");

            }

            else

            {

                srv.socOut.writeUTF("Authentication unsuccessful");

                System.out.println("Authentication unsuccessful");

            }

            srv.socOut.close();

            srv.socIn.close();

            srv.soc.close();

            userIn.close();

            System.out.println("Communication terminated");

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

}

##### Client

###### Working functionality

It also works as the previous one but here the authentication packet contains the identity of the client, identity of the server and the nonce encrypted with the private key.

###### Program

package host;

import java.io.\*;

import java.net.InetAddress;

import java.net.Socket;

import cryptotool.CaesarCipher;

public class Client {

    private Socket soc;

    private DataInputStream socIn;

    private DataOutputStream socOut;

    private int serverPort, localPort, key;

    public Client(int srvPort, int localPort, int key)

    {

        try

        {

            this.serverPort = srvPort;

            this.localPort = localPort;

            this.key = key;

            System.out.println("Client started");

            this.soc = new Socket(InetAddress.getLocalHost(), this.serverPort, InetAddress.getLocalHost(), this.localPort);

            socIn = new DataInputStream(soc.getInputStream());

            socOut = new DataOutputStream(soc.getOutputStream());

            System.out.println("Local port address: " + this.localPort);

            System.out.println("Connection established with port: " + this.serverPort);

        }

        catch(Exception e)

        {

            e.printStackTrace();

        }

    }

    public static void main(String[] args) {

        try

        {

            BufferedReader userIn = new BufferedReader(new InputStreamReader(System.in));

            int srvPort, srvKey;

            System.out.print("Enter the port address of the server: ");

            srvPort = Integer.parseInt(userIn.readLine());

            System.out.print("Enter the key(should be known for authentic client): ");

            srvKey = Integer.parseInt(userIn.readLine());

            Client clt = new Client(srvPort, 9010, srvKey);

            clt.socOut.writeUTF(String.valueOf(clt.localPort));

            System.out.println("Request sent: " + clt.localPort);

            String nonce = clt.socIn.readUTF();

            System.out.println("Got nonce: " + nonce);

            String verCode = String.valueOf(clt.localPort) + String.valueOf(clt.serverPort) + nonce;

            System.out.println("Generated packet: " + verCode);

            verCode = CaesarCipher.encrypt(verCode, clt.key);

            System.out.println("Encrypted packet: " + verCode);

            clt.socOut.writeUTF(verCode);

            System.out.println("Packet sent");

            String response = clt.socIn.readUTF();

            System.out.println(response);

            clt.socOut.close();

            clt.socIn.close();

            clt.soc.close();

            userIn.close();

            System.out.println("Communication terminated");

        }

        catch(Exception e)

        {

            e.printStackTrace();

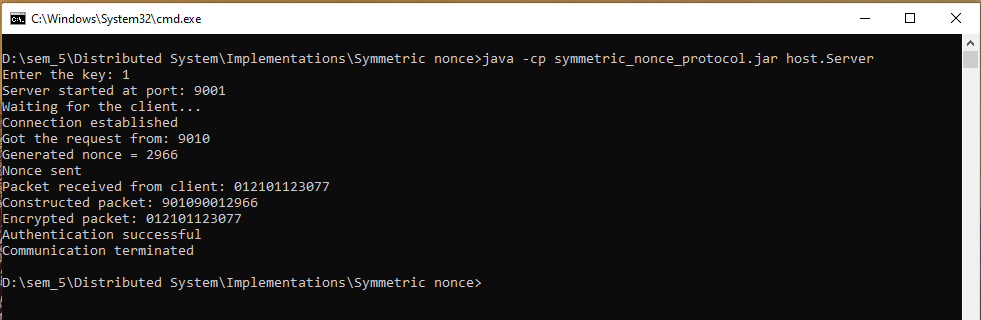
        }

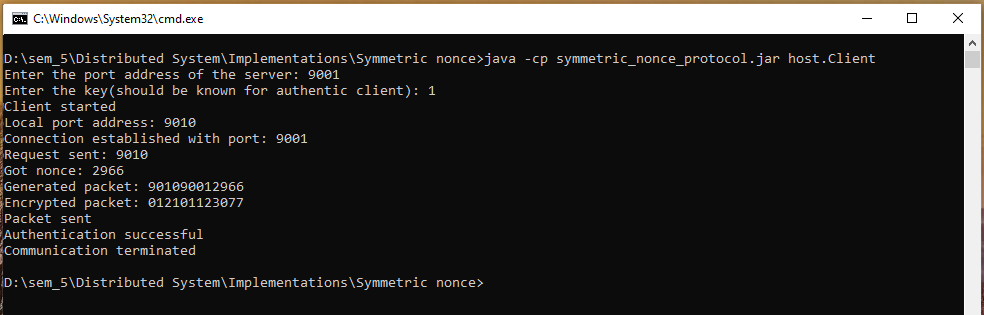
    }

}

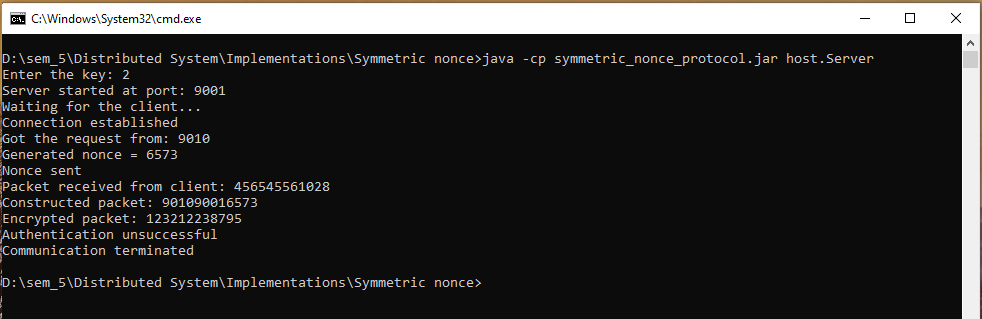
### Output

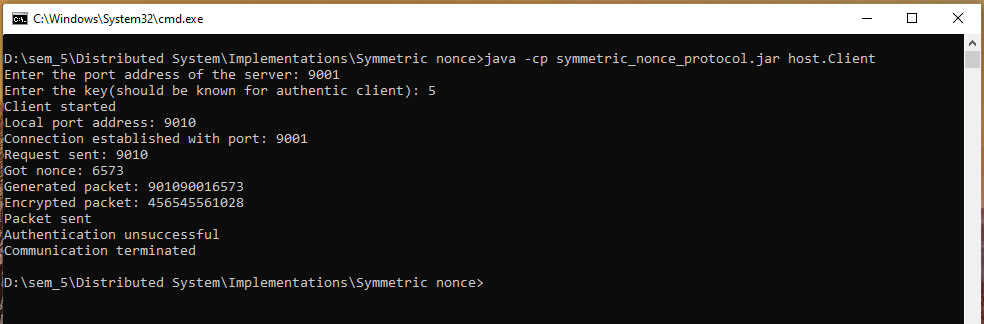
#### Normal output





#### Unsuccessful output





### Considerations

* Identity of a host is its port number as all the host will run on the same computer.
* Nonce is just a pseudo-random number generated by the server.
* Server gets the identity of the client when a client send a request at the beginning.