Sicurezza dell'informazione M

Lorenzo Angelini (N. matricola 0000749514)

Descrizione del progetto

Il progetto ha come scopo quello di creare un canale di comunicazione sicuro tra due applicativi Java (*Client* e *Server*) collegati ad Internet. Gli applicativi utilizzano come canale di comunicazione non sicuro una *Socket TCP* e implementano il protocollo di *Diffie Hellman Anonimo* per accordarsi su una chiave di cifratura simmetrica che useranno poi per cifrare/decifrare i dati trasmessi con algoritmo *AES* in modalità *CBC*.

In questo modo i due host sono in grado di comunicare in maniera riservata, senza che un intruso possa capire quali informazioni si stanno scambiando sul canale. Infatti, pur potendo intercettare l'intera comunicazione tra i due hosts (che comprende le chiavi pubbliche di Diffie Hellman di entrambi ed il contenuto della comunicazione in forma cifrata), l'intruso non sarà in grado di decifrare nessuna informazione.

Come esempio di utilizzo il *Client*, una volta stabilito un canale riservato con il *Server*, gli chiede di essere informato ogni 2 secondi sul suo carico di sistema fino ad un massimo di N volte.

Implementazione del protocollo di Diffie-Hellman

Di seguito sono descritti i passi grazie ai quali *Client* e *Server* riescono ad ottenere una chiave comune K:

1. Il *Client* genera i parametri p e g

   $$p : \textit{numero primo}$$

   $$g : \textit{generatore del gruppo moltiplicativo degli interi modulo } p$$

2. Il *Client* genera un numero casuale a e calcola il valore di $A = g^a \bmod p$ (dove *mod* indica l'operazione modulo, ovvero il resto della divisione intera).
   In questo modo ha creato una coppia di chiavi, una pubblica PK ed una privata SK, definite come:

   $$SK = a$$

   $$PK = (g, p, A)$$

3. Il *Client* invia la sua chiave pubblica DH al *Server*

4. Il *Server*, grazie ai parametri contenuti nella chiave pubblica DH del *Client* può generare a sua volta una coppia di chiavi definite come:

$$SK = b = numero\ casuale\ generato\ dal\ Server$$

$$PK = B = g^b \bmod p$$

5. Il *Server* invia la sua chiave pubblica DH PK al Client

6. Il *Client* calcola la chiave a 128 bit K

$$K = serverPK^a \bmod p = B^a \bmod p = g^{ab} \bmod p$$

7. Il *Server* calcola la chiave a 128 bit K

$$K = clientPK^b \bmod p = A^b \bmod p = g^{ab} \bmod p$$

8. Le chiavi calcolate K sono identiche

A questo punto i due interlocutori sono entrambi in possesso della chiave segreta K e possono cominciare ad usarla per cifrare le comunicazioni successive.

Un attaccante può benissimo ascoltare tutto lo scambio, ma per calcolare i valori a e b avrebbe bisogno di risolvere l'operazione del logaritmo discreto, che è computazionalmente onerosa e richiede parecchio tempo, in quanto sub-esponenziale (sicuramente molto più del tempo di conversazione tra i 2 interlocutori).

Algoritmo di cifratura AES-128 CBC

Per la cifratura/decifratura dei dati trasmessi viene utilizzato l'algoritmo *AES* (Advanced Encryption Standard) impiegando la chiave da 128 bit concordata precedentemente.

AES è un algoritmo di cifratura a blocchi utilizzato come standard dal governo degli Stati Uniti d'America. AES è stato scelto perchè è tra gli algoritmi di cifratura più sicuri, si pensi che *NSA* ritiene che una chiave a 128 bit impiegata con algoritmo *AES* sia adeguata per proteggere documenti classificati come *SECRET*, per documenti classificati come *TOP SECRET* si rende invece necessaria una chiave da 192 o 256 bit.

AES viene utilizzata in modalità *CBC* (Cipher Block Chaining) per risolvere il problema del determinismo di cui è affetta invece la modalità *ECB*: per uno stesso blocco di testo in chiaro viene sempre generato lo stesso testo cifrato. Questa peculiarità potrebbe fornire ad un eventuale attaccante una vulnerabilità da studiare per cercare di scoprire la parola chiave utilizzata.

CBC invece, prima di cifrare, somma modulo 2 il blocco di testo in chiaro con il precedente blocco cifrato. Questo comporta che ogni blocco di testo cifrato generato dipenda in maniera diretta dai blocchi di testo cifrato precedente. Il primo blocco di testo in chiaro verrà sommato modulo 2 con un vettore di inizializzazione che viene generato dal *Client* e inviato in chiaro al *Server*.

Codice sorgente

Test

```
import diffiehellman.client.ClientThread;
import diffiehellman.server.ServerThread;


public class Test {

  public static void main(String[] args) {

    (new Thread(new ServerThread())).start();
    (new Thread(new ClientThread())).start();

  }
}
```

ClientThread

```
package diffiehellman.client;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.security.AlgorithmParameterGenerator;
import java.security.AlgorithmParameters;
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.Cipher;
import javax.crypto.KeyAgreement;
import javax.crypto.SecretKey;
import javax.crypto.spec.DHParameterSpec;

public class ClientThread implements Runnable {

  private int serverPort = 8412;
  private String serverAddr = "localhost";

  public ClientThread() {
```

```
}

public void run() {

  // Inizializzazione della Socket
  Socket server = null;
  try {
    server = new Socket(serverAddr, serverPort);
  } catch (IOException e) {
    System.out.println("Unable to reach server");
    e.printStackTrace();
    return;
  }

  // Crea stream di in/out
  ObjectOutputStream outSocket = null;
  ObjectInputStream inSocket = null;
  try {
    outSocket = new ObjectOutputStream(server.getOutputStream());
    inSocket = new ObjectInputStream(server.getInputStream());
  } catch (IOException e) {
    System.out.println("Unable to get socket streams");
    e.printStackTrace();
    return;
  }

  // Adesso la connessione écorrettamente stabilita
  try {
    SecretKey key = AESDHKeyAgreement(outSocket, inSocket);

    String times = String.valueOf(4);
    sendAESCryptedString(times, key, outSocket, inSocket);

    String cleartext;
    do {
      cleartext = receiveAESCryptedString(key, outSocket, inSocket);
      System.out.println("Client: decrypted text: "+cleartext);
    } while(!cleartext.equals("stop"));

  } catch (Exception e) {
    System.out.println("Client: ERROR");
    e.printStackTrace();
  }
}

public SecretKey AESDHKeyAgreement(ObjectOutputStream outSocket,
    ObjectInputStream inSocket) throws Exception {
```

```
/* Il Client genera i parametri g e p, operazione costosa
 * p: numero primo
 * g: generatore del gruppo moltiplicativo degli interi modulo p
 */
System.out.println("Creating Diffie-Hellman parameters (takes VERY
    long) ...");
AlgorithmParameterGenerator paramGen =
    AlgorithmParameterGenerator.getInstance("DH");
paramGen.init(1024);
AlgorithmParameters params = paramGen.generateParameters();
DHParameterSpec dhSkipParamSpec = (DHParameterSpec)
    params.getParameterSpec(DHParameterSpec.class);

/* Il Client crea la sua coppia di chiavi DH utilizzando i parametri
    generati sopra
 * SK = a: numero casuale
 * PK = (g,p,A)
 * A: (g^a) mod p
 */
System.out.println("Client: Generate DH keypair ...");
KeyPairGenerator clientKpairGen = KeyPairGenerator.getInstance("DH");
clientKpairGen.initialize(dhSkipParamSpec);
KeyPair clientKpair = clientKpairGen.generateKeyPair();

// Il Client codifica la sua chiave pubblica e la invia al Server
byte[] clientPubKeyEnc = clientKpair.getPublic().getEncoded();
outSocket.writeObject(clientPubKeyEnc);

/*
 * Il Client riceve la chiave pubblica DH del Server in forma
    codificata.
 * Istanzia un oggetto PublicKey dai dati che ha ricevuto
 */
byte[] serverPubKeyEnc = (byte[]) inSocket.readObject();
KeyFactory clientKeyFac = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new
    X509EncodedKeySpec(serverPubKeyEnc);
PublicKey serverPubKey = clientKeyFac.generatePublic(x509KeySpec);

/* Il Client crea ed inizializza un oggetto KeyAgreement di DH che,
 * grazie alla SK del Client e alla PK del Server,
 * puó calcolare la chiave concordata K
 * K = (serverPK^a) mod p = (B^a) mod p = (g^ab) mod p
 */
System.out.println("Client: Initialization ...");
KeyAgreement clientKeyAgree = KeyAgreement.getInstance("DH");
clientKeyAgree.init(clientKpair.getPrivate());
```

```
    System.out.println("Client: calculating agreed KEY ...");
    clientKeyAgree.doPhase(serverPubKey, true);

    /*
     * A questo punto sia il Client che il Server hanno completato il
        protocollo di DH.
     * Hanno ottenuto entrambi la chiave concordata K
     */
    return clientKeyAgree.generateSecret("AES");
  }

  private void sendAESCryptedString(String cleartext, SecretKey key,
      ObjectOutputStream outSocket, ObjectInputStream inSocket) throws
      Exception {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] encodedParams = cipher.getParameters().getEncoded();
    outSocket.writeObject(encodedParams);
    byte[] ciphertext = cipher.doFinal(cleartext.getBytes());
    outSocket.writeObject(ciphertext);
  }

  private String receiveAESCryptedString(SecretKey key, ObjectOutputStream
      outSocket, ObjectInputStream inSocket) throws Exception {
    byte[] encodedParams = (byte[]) inSocket.readObject();
    AlgorithmParameters params = AlgorithmParameters.getInstance("AES");
    params.init(encodedParams);
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, key, params);
    byte[] ciphertext = (byte[]) inSocket.readObject();
    byte[] recovered = cipher.doFinal(ciphertext);
    return new String(recovered);
  }
}
```

ServerThread

```
package diffiehellman.server;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.management.ManagementFactory;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.AlgorithmParameters;
```

```
import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PublicKey;
import java.security.spec.X509EncodedKeySpec;

import javax.crypto.Cipher;
import javax.crypto.KeyAgreement;
import javax.crypto.SecretKey;
import javax.crypto.interfaces.DHPublicKey;
import javax.crypto.spec.DHParameterSpec;

public class ServerThread implements Runnable {

  private int serverPort = 8412;

  public ServerThread() {
  }

  public void run() {
    // Socket in ascolto su localhost
    ServerSocket server = null;
    try {
      server = new ServerSocket(serverPort);
      System.out.println("Server listening on "+serverPort);
    } catch (IOException e) {
      System.out.println("Error while starting server");
      e.printStackTrace();
      return;
    }

    // Loop infinito di accettazione richieste di connessione
    while(true)
    {
      Socket socket = null;
      try {
        socket = server.accept();
        // Nuova connessione
        System.out.println("New Connection on port : " +
            socket.getLocalPort());
      } catch (IOException e) {
        System.out.println("Error while enstablishing connection");
        e.printStackTrace();
        continue;
      }

      // Crea stream di in/out
```

```
    ObjectOutputStream outSocket;
    ObjectInputStream inSocket;
    try {
      outSocket = new ObjectOutputStream(socket.getOutputStream());
      inSocket = new ObjectInputStream(socket.getInputStream());

    } catch (IOException e) {
      System.out.println("Unable to get socket streams");
      e.printStackTrace();
      continue;
    }

    // Adesso la connessione écorrettamente stabilita
    try {
      SecretKey key = AESDHKeyAgreement(outSocket, inSocket);

      String cleartext = receiveAESCryptedString(key, outSocket,
          inSocket);
      System.out.println("Server: decrypted text: "+cleartext);

      int times = Integer.parseInt(cleartext);
      for(int i = 0; i<times; i++) {
        String SysLoadAvg =
            String.valueOf(ManagementFactory.getOperatingSystemMXBean()
        .getSystemLoadAverage());
        sendAESCryptedString(SysLoadAvg, key, outSocket, inSocket);
        Thread.sleep(2000);
      }
      sendAESCryptedString("stop", key, outSocket, inSocket);

    } catch (Exception e) {
      System.out.println("Server: ERROR");
      e.printStackTrace();
      continue;
    }
  }
}

public SecretKey AESDHKeyAgreement(ObjectOutputStream outSocket,
    ObjectInputStream inSocket) throws Exception {

  /*
   * Il Server riceve la chiave pubblica DH del Client in forma
      codificata.
   * Istanzia un oggetto PublicKey dai dati che ha ricevuto
   */
  byte[] clientPubKeyEnc = (byte[]) inSocket.readObject();
```

```
  KeyFactory serverKeyFac = KeyFactory.getInstance("DH");
  X509EncodedKeySpec x509KeySpec = new
     X509EncodedKeySpec(clientPubKeyEnc);
  PublicKey clientPubKey = serverKeyFac.generatePublic(x509KeySpec);

  /*
   * Il Server preleva i parametri di DH associati alla chiave pubblica
      del Client.
   * PK = (g,p,A)
   * p: numero primo
   * g: generatore del gruppo moltiplicativo degli interi modulo p
   * A: (g^a) mod p
   * Gli serviranno per generare la sua coppia di chiavi DH
   */
  DHParameterSpec dhParamSpec = ((DHPublicKey) clientPubKey).getParams();

  /* Il Server genera la sua coppia di chiavi DH
   * SK = b: numero casuale
   * PK = B: (g^b) mod p
   */
  System.out.println("Server: Generate DH keypair ...");
  KeyPairGenerator serverKpairGen = KeyPairGenerator.getInstance("DH");
  serverKpairGen.initialize(dhParamSpec);
  KeyPair serverKpair = serverKpairGen.generateKeyPair();

  // Il Server codifica la sua chiave pubblica e la invia al Client
  byte[] serverPubKeyEnc = serverKpair.getPublic().getEncoded();
  outSocket.writeObject(serverPubKeyEnc);

  /* Il Server crea ed inizializza un oggetto KeyAgreement di DH che,
   * grazie alla SK del Server e alla PK del Client,
   * puó calcolare la chiave concordata K
   * K = (clientPK^b) mod p = (A^b) mod p =(g^ab) mod p
   */
  System.out.println("Server: Initialization ...");
  KeyAgreement serverKeyAgree = KeyAgreement.getInstance("DH");
  serverKeyAgree.init(serverKpair.getPrivate());
  System.out.println("Server: calculating agreed KEY ...");
  serverKeyAgree.doPhase(clientPubKey, true);

  /*
   * A questo punto sia il Client che il Server hanno completato il
      protocollo di DH.
   * Hanno ottenuto entrambi la chiave concordata K
   */
  return serverKeyAgree.generateSecret("AES");
}
```

```
  private void sendAESCryptedString(String cleartext, SecretKey key,
      ObjectOutputStream outSocket, ObjectInputStream inSocket) throws
      Exception {
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, key);

    byte[] encodedParams = cipher.getParameters().getEncoded();
    outSocket.writeObject(encodedParams);

    byte[] ciphertext = cipher.doFinal(cleartext.getBytes());
    outSocket.writeObject(ciphertext);
  }

  private String receiveAESCryptedString(SecretKey key, ObjectOutputStream
      outSocket, ObjectInputStream inSocket) throws Exception {
    byte[] encodedParams = (byte[]) inSocket.readObject();
    AlgorithmParameters params = AlgorithmParameters.getInstance("AES");
    params.init(encodedParams);

    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, key, params);
    byte[] ciphertext = (byte[]) inSocket.readObject();
    byte[] recovered = cipher.doFinal(ciphertext);
    return new String(recovered);
  }
}
```