



Real-Time Authentication Using Digital Signature Schema

Marissa Hollingsworth

BOISECRYPT '09

Overview

- Introduction
- Methodology
- Protocol and Implementation
- Demonstration
- Conclusion
- Future Work



INTRODUCTION

Project Overview

- RSA Secure File Transfer Server
 - Verifies user and sends files across network
- RSA Client
 - Connects to server and requests files
- Authentication
 - Server authenticates client using RSA algorithms
- Data Transfer
 - Server encrypts and client decrypts using RSA algorithms

Motivation

- High demand for secure data transactions over networks
- Online banking, email, instant messaging, online shopping, etc.
- Constant improvements in security is required as internet traffic increases

Application Objectives

- Secure
- Reliable
- User-Friendly
- Robust

Client-Server Architecture and RSA Cryptosystem



METHODOLOGY

Client-Server Programming

- Computer programs that communicate with each other over a network
- Two Programs
 - Server - Program to wait for initiation of communication
 - Client – Program to initiate the communication
- Protocol
 - Controls communication between programs

Client – Server Roles

Server

- Provides some service to client
- Establishes connection
- Usually up all the time
- Stores information about client
- Has access to databases and other data

Client

- Uses service provided by server
- Displays and receives results sent from server
- Responds to prompts from server

Client-Server Communication

- Communication Channel
 - Socket
 - End-point of a two-way communication link between two programs running on a network
- Communication Language
 - Protocol
 - Set of rules used to “talk” between client and server
 - Ex: To speak in class we raise hand. Teacher calls on us and tells us it is ok to talk.



RSA PROTOCOL

So what does the RSA cryptosystem have to do with all this??

Brief overview of RSA

- Cryptosystem for encrypting and decrypting messages (or packets)
- Sender must have *public key* of receiver
- Receiver must have *private key* to unlock the message
- ONLY the receiver should know the private key



RSA Signature Schema

- Similar to RSA encryption
- Use a different key to sign and send messages
- Sender signs message with private key



- Receiver verifies signature with public key of sender



IMPLEMENTATION

(Java)

RSAKey Class

- PublicKey and PrivateKey are inner classes – not globally accessible
- Separate classes so only Public Key can be sent over network
- RSAKey constructor generates a new RSA key according to the RSA key generation algorithm
- Serializable = can be saved and sent as object

```

public class RSAKey implements Serializable {

    private static final long serialVersionUID = 1L;

    private transient int CERTAINTY = 100;

    protected RSAPublicKey publicKey;
    protected RSAPrivateKey privateKey;
    protected transient ASCIIPad p;
    ...
class RSAPublicKey implements Serializable
{
    private static final long serialVersionUID = 1L;

    /** public signature modulus */
    protected BigInteger R = null;

    /** public signature exponent */
    protected BigInteger e = null;
    ...
public class RSAPrivateKey implements Serializable
{
    private static final long serialVersionUID = 1L;
    /** private signature key */
    private BigInteger d;

```

Server

- Multi-threaded so that multiple clients can connect
- Opens a socket on specified port and waits for client to connect
- While running, stores a list of clients and their public keys in a HashMap so they can sign back on

```
import java.net.*;
import java.util.HashMap;
import java.io.*;

public class RSAMultiServer {

    public static void main(String[] args) throws IOException {

        HashMap<Object, RSAKey.RSAPublicKey> clients =
            new HashMap<Object, RSAKey.RSAPublicKey>();

        ServerSocket serverSocket = null;
        boolean listening = true;

        try {
            serverSocket = new ServerSocket(9286);
        } catch (IOException e) {
            System.err.println("Could not listen on port: 9286.");
            System.exit(-1);
        }


        while (listening)
        {
            new RSAMultiServerThread(serverSocket.accept(), clients).start();
        }
        serverSocket.close();
    }
}
```

- Connects to client's Object Output Stream to send data through socket

```
public class RSAMultiServerThread extends Thread {  
  
    private Socket socket = null;  
    private HashMap<Object, RSAKey.RSAPublicKey> clients = null;  
    private Object clientName;  
    private static String dir = null;  
  
    public RSAMultiServerThread(Socket socket, HashMap<Object, RSAKey.RSAPublicKey> clients) {  
        super("RSAMultiServerThread");  
        this.socket = socket;  
        this.clients = clients;  
    }  
  
    public void run() {  
        ObjectInputStream ois = null;  
        ObjectOutputStream oos = null;  
        try {  
            dir = (new File(".")).getCanonicalPath() + File.separator + "serverFiles";  
        } catch (Exception e) {}  
  
        try {  
            ois = new ObjectInputStream (socket.getInputStream());  
            oos = new ObjectOutputStream (socket.getOutputStream());  
        } catch (IOException e) {  
            System.err.println("I/O Streams failed");  
            System.exit(1);  
        }  
    }  
}
```

Client

- Connects to server and waits for data
- Only needs to handle a few commands from server
 - START UNAME LOADKEY
 - KEY FILE SIGN
 - YorN INPUT PRINT
- Stores RSAKeys assigned by server for authentication and decryption of packets

- 
- Can load key from file stored on local machine to connect and decrypt
 - Received data is stored in a file with the client's username

Protocol:

Client-Server Communication

- Most important (and difficult) aspect of socket programming
- Before sending objects across network, server must tell the client what to expect
- Example:
 - To send a file to the client, server must perform the following actions. Client must act based on protocol.

Sending file to client...

```
public boolean processFileChoice(ObjectOutputStream oos, ObjectInputStream ois)
{
    /* Get input from client and save as filename */
    RSAProtocol.sendREQ(RSAProtocol.INPUT, oos);
    String filename = (String) RSAProtocol.receiveObject(ois);

    /*Send file accept request to client */
    RSAProtocol.sendREQ(RSAProtocol.FILE, oos);
    RSAProtocol.sendObject(filename, oos);
    File dir = new File("serverFiles/" + filename);

    /* Encrypt and send file to user */
    try{ FileInputStream fis = new FileInputStream(dir);

        byte[] buff = new byte[1024];
        int num = 0;

        while((num = fis.read(buff)) != -1)
        {
            byte[] b = new byte[num];
            for(int i = 0; i < num; i++)
                b[i] = buff[i];
            byte[] enc = clients.get(clientName).encryptBytes(b);
            oos.write(enc.length);
            oos.write(enc);
            oos.flush();
        }
        fis.close();
    } catch (Exception e) {
```

Client receiving file...

```
case RSAProtocol.INPUT:
    RSAProtocol.sendObject(scan.next(), client.oos);
    break;

case RSAProtocol.FILE:
    String filename = (String) RSAProtocol.receiveObject(client.ois);
    FileOutputStream fos =
        new FileOutputStream(new File(dir + filename));

    byte[] buff;
    int num = client.ois.read();
    System.out.println(num);
    while(num > 0)
    {
        buff = new byte[num];
        client.ois.read(buff);
        num = client.ois.read();
        fos.write(client.key.privateKey.decryptBytes(buff));
        fos.flush();
    }

    fos.close();
    break;
```

Protocol variables

```
public class RSAProtocol {  
  
    public static final int QUIT = -1;  
    public static final int START = 0;  
    public static final int UNAME = 1;  
    public static final int REQ = 2;  
    public static final int NACK = 3;  
    public static final int Yorn = 4;  
  
    public static final int KEY = 10;  
    public static final int INPUT = 11;  
    public static final int LOADKEY = 12;  
  
    public static final int FILE = 20;  
    public static final int SIGN = 21;  
    public static final int FILEMENU = 22;  
  
    public static final int PRINT = 40;  
    public static final int DONE = 50;  
    public static final int SENDING = 50;  
}
```

The fun part! 😊



DEMONSTRATION




CONCLUSION

Successes

- Successfully implemented RSA encryption and signature algorithms
- Able to send encrypted files across a network
- Able to decrypt and the files on the client side
- Cannot decrypt without the private key...
 - i.e. secure
- Learned *a lot* about socket programming


Challenges (and failures 😞)

- Sending objects across the sockets
 - Frustrating at times, but satisfying when worked
- Keeping private keys private
 - Key storage (should we ever store them?)
 - Key generation could be vulnerable to attacks if unlucky
- El-Gamal signature schema
 - Socket programming was a little harder than I had anticipated

- 
- **Protocol implementation**
 - Difficult to synchronize between client and server
 - Approximately 5 different designs while developing
 - **Learning to use BigInteger class for RSA implementation**



FUTURE WORK

- 
- **Multi-Client communication**
 - Ability to sign/encrypt messages and send them over the server to another client
 - **Transfer more than just text files**
 - **Implement more encryption and signature schema**
 - **Graphical User Interface would be nice**



QUESTIONS??