

Assignment 2

Introduction

The purpose of this assignment is to gain experience in implementing a cryptographic protocol that involves key exchange, digital signatures and encryption.

General description

In this assignment, you are asked to implement a protocol similar to the SSL or the SSH protocols. The protocol involves certificates, key exchange, digital signatures, encryption and decryption. We will use both public key and symmetric encryptions. This assignment can be done either individually, or in teams of 2. If done in a team of 2, each student will need to implement the code on their own computer and submit their own copy of the report. Submit a report with the parts described below. If the part ask for files, include the file names in the appropriate part of the report and include all the files in a .zip folder.

Part 1: Creating RSA cryptographic keys

The program `CreatePemKeys.java` creates an RSA key pair, and saves the private and public parts in 2 different files, named `privateKey.pem` and `publicKey.pem`. The files are in PEM format, which is readable in a text editor. It uses the static methods in `PemUtils.java`, which transforms between Java key objects and PEM format. The methods use Base64 encoding. Base64 encoding is included in Java 8. It was possible to use Base64 encoding in prior versions of Java, but I think some file downloads were necessary and maybe the method names were different. Modify the program to ask which file names the user wants to use. Modify the comments appropriately.

Use your program to create four pairs of keys, two for the client and two for the server. The client and the server will use different key pairs for signature and for encryption. Choose file names that refer to you (like `JohnDoeClientSignPublic` or `JohnDoeServerEncryptPrivate`). The RSA algorithm

can be used both for digital signature and for encryption, so one can use the same keys for both purposes. In practice, even if we use RSA, we use different key pairs for encryption and for signature. You can test your keys for encryption and decryption with programs `Encrypt.java` and `Decrypt.java`. You can test your keys for digital signatures with programs `Sign.java` and `Verify.java`. Include in your report your modified `CreatePemKeys.java` and the keys you generated. Include in your report the encryption of a string containing your name and current date. Include in your report the digital signature of a string containing your name and current date. Also include in your report the answer to the following question: If we encode a byte array of length n with Base64 encoding, what is the length of the string derived from the encoding? (Include any reference you have consulted, if any.)

Part 2: Generating key certificates

I have set up a web server that acts as a certificate authority. The server will take a name, an encryption public key, a signature public key, and will execute a PHP program to produce a certificate. The server can be accessed at <http://cs5339.cs.utep.edu/longpre/authority/CertificateRequest.html>. The authority's public key is provided in file `CApublicKey.pem`. In practice, the X.509 standard dictates an elaborate structure for digital certificates, and the certificate includes a digital signature. The signature is Base64 encoded to make it readable. For this assignment, to keep it simple, the certificate only contains the name, the date, the encryption public key, the signature public key, and a digital signature of those items with the authority's private key, in a format similar to the PEM format. (Last year, I spent quite a bit of time making the PHP versions of RSA encryption/signature compatible with the Java versions.) Include in your report the certificates you generated.

Part 3: Verify your certificates

The program `verifyCert.java` verifies the format and the signature of a certificate. It assumes the certificate is in file `certificate.txt` and that the public key is in `CApublicKey.pem`. Make sure that it works with the certificates you obtained in Part 2.

Part 4: Communication

The programs `MultiEchoServer.java` and `EchoClient.java` are programs that use sockets to communicate with each other. First run the Server. It waits for connections on port 8008 from clients. Then run the Client. It may be more realistic to use another instance of the IDE instead of running the two programs from the same instance. Better yet, use two different IDEs or run one from the command line. There are many IDEs like Dr.Java, Eclipse, Netbeans. The client asks for an IP address or localhost, which resolves to your computer's IP address. Use localhost when the server and the client run on the same computer. Also try for something even more realistic: run one program on a virtual machine and another program on your computer. You will need the IP address of the server, which will now be different from localhost. On the Windows command line, use the `ipconfig` command to get the IP address of your computer. Look for IPv4 or inet address. On Linux, use the `ifconfig` command. You may have to play with the network settings of your virtual machine. First try both NAT and bridged.

After the client connects, the server echoes all the strings back to the client. In your report, give the result of running these programs. When experimenting with the virtual machine, try both at UTEP and out of UTEP. Although you don't need to succeed communicating between the virtual machine and the computer, you need to try and include the result of your attempts in your report. Last year, I succeeded in the communication whether the server is on the computer or on the virtual machine when at home, but only in one direction when at UTEP. This year I succeeded in both directions at UTEP, and only tested the server on the virtual machine at home.

Part 5: Encrypting communications through the socket

The programs `CryptoMultiEchoServer.java` and `CryptoEchoClient.java` are the programs from Part 4 modified to perform AES encryption. The programs derive the AES key from an array of random bytes. The number of random bytes must match the size of the AES key, which must be 16, 24 or 32. We are using 16 in this program. AES keys could also be derived from provided passwords instead of random bytes. To ensure a shared key, both the client and the server use the same random bytes source by reading the bytes from a file named `randomBytes`. Use the program `CreateRandomSecret.java` to create the `randomBytes` file. In the final pro-

tol, the client and the server will generate a shared random secret. The client and server programs also communicate by sending serializable objects through the socket instead of strings. We could use Base64 encoding to convert to strings, but we show here how to deal with serializable java objects. In the provided programs, the communication from the client to the server is encrypted, but the communication from the server to the client is not encrypted. Modify the programs to encrypt communications in both direction. Include your modified programs in your report.

Part 6: The secure communication protocol

Now you have to implement our protocol which follows similar steps as the SSL protocol. The protocol works as follows.

- The client opens a socket connection with the server and sends the string “hello” to the server using `println()`.
- The server sends its certificate to the client using `println()`.
- The client receives the certificate. The certificate is verified for format, the signature in the certificate is verified, and the two server public keys are extracted from the certificate.
- The client sends its certificate to the server using `println()`.
- The server receives the certificate. The certificate is verified for format, the signature in the certificate is verified, and the two client public keys are extracted from the certificate.
- The server generates 8 random bytes to be part of a shared secret. The random bytes are encrypted with the client public key and the encrypted bytes are sent to the client as a byte array object.
- The server hashes the random bytes using SHA-256 and signs the hash using its own private key. The signature is sent to the client as a byte array object.
- The client receives the encrypted random bytes and the signature. The client decrypts the server random bytes and verifies the signature.

- The client generates 8 random bytes to be part of a shared secret. The random bytes are encrypted with the server public key and the encrypted bytes are sent to the server as a byte array object.
- The client hashes the random bytes using SHA-256 and signs the hash using its own private key. The signature is sent to the server as a byte array object.
- The server receives the encrypted random bytes and the signature. The server decrypts the client random bytes and verifies the signature.
- Both the server and the client generates a 16 bytes shared secret by creating a 16 bytes array that contains the server random bytes as the first 8 bytes and the client random bytes as the next 8 bytes. They then create a shared AES key from the random bytes.
- The server initializes a cipher for AES encryption, retrieve the initialization vector of the cipher and sends it to the client as a byte array object.
- The client receives the server's initialization vector and initializes the AES cipher for decryption with that initialization vector.
- The client initializes a cipher for AES encryption, retrieve the initialization vector of the cipher and sends it to the server as a byte array object.
- The server receives the client's initialization vector and initializes the AES cipher for decryption with that initialization vector.
- The client and the server enter an echo communication loop. the following steps are repeated:
 - The client asks for a string from the user. The client uses the AES key to encrypt the string and sends the encrypted string to the server as a byte array object.
 - The server receives and decrypts the string from the client, constructs an echo string, uses the AES key to encrypt it and sends the encrypted string to the client as a byte array object.

- The client decrypts the string from the server and display the decrypted string to the user.
- The loop continues until the user enters the string “BYE”.

We are planning to set up a server that implements this protocol. Your client should be able to communicate successfully with the server. The IP address of the server will be communicated later. When the server receives all zeroes for the client random bytes, it enters testing mode and the shared secret is assumed to be all zeroes. You don't have to implement the testing mode. We provide a skeleton client program that communicates with the server in testing mode.

Include your client and server programs and any other files or programs necessary to run your programs in your report.

Part 7: Conclusion

Include the answer to the following with your report.

1. How easy was this assignment? Explain any challenge you encountered and how you overcame the challenge.
2. What difficulties did you encounter during this assignment, and how did you overcome the difficulties?
3. If you worked in a group of 2, explain your group dynamic, including whether you worked physically together, whether you worked individually and communicated by e-mail, whether you separated the work, and generally, the contribution of each member of the group.
4. Explain what you learned in this assignment.
5. Do you have any suggestion on how to improve this assignment if we use it the next time we teach this course.

Turn in:

Write a report with a section for each part of the assignment. In each section of the report, give a brief description of what you've done, references if any, instructions on how to run the program if appropriate, and the file names for

the files associated with the section. Submit a .zip or other standard archive including the report and the requested programs and files. If working in a team of 2, each must turn in a submission. Of course, the programs will probably be the same and we may only grade those in one of the submissions. But the keys and a few other subsections will have differences.

Due date:

April 2, 11pm. Submit by e-mail to both *longpre@utep.edu* and the TA at *afgarciacontreras@miners.utep.edu*. The penalty for a late homework is 1% per hour up to 10% per day, for up to one week late, counting Saturday and Sunday as well. If your submission is missing part 6, you may submit part 6 at a later point but no later than April 30th, for half of the credit on that part.