EE 4723 PA 1

Spring 2018

Due: 23 March 2018

Implementing A Simple KDC

At a very high level, we are going to develop a KDC that implements the *Expanded* Needham-Schroeder protocol. You will need to create three programs: A client (Alice), a server (Bob), and a KDC. Bob will be a single-purpose server that has a plain text file. After authenticating herself to Bob, Alice will send a GET request to receive the encrypted contents of the file. Once received, Alice will decrypt the contents and print them to the screen. All communication will be over UDP, just for simplicity.

Programming Language Choice

You will be using standard sockets to do this assignment and it is expected you will choose C/C++, Java or Python. C# is an acceptable alternative if you really, really want to use it.

Cryptographic Libraries

This isn't an assignment where you should be implementing crypto algorithms from scratch. Java has a built-in encryption extension, the Java Cryptography Extension. (See `http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html`). Python implementations should use PyCryptodome (*not* PyCrypto, which is no longer under active development). Information on installing and using PyCryptodome is here: `https://pypi.python.org/pypi/pycryptodome`. If you're using C, the simplest option *appears* to be libgcrypt (see: `https://gnupg.org/software/libgcrypt/`). C# has built-in crypto libraries.

The Client (Alice)

Here is a specific list of things that Alice needs to do.

- Upon startup, Alice can assume that the KDC and Bob are already running.
- Following Expanded N-S, she should send a UDP packet with a plaintext ASCII payload, `Let's talk`.
- Then she would wait for an encrypted nonce from Bob (again, as an ASCII string).
- When the nonce is received, Alice creates a payload of 4 concatenated strings: her nonce, the text `Alice`, the text `Bob`, and Bob's encrypted nonce. She opens a connection to the KDC and sends this payload. Then she waits for a response from the KDC.
- When the ticket and session key comes back from the KDC, Alice can close the connection to the KDC. Alice should then decrypt the packet to find the session key. Then she invents a nonce, encrypts it and builds a packet with the ticket and encrypted nonce. Next, she opens a connection to Bob and sends this packet to him.
- She waits for a response and decrypts it with the session key. She verifies that Bob appropriately modified her nonce. If so, she decrements the nonce that Bob sends, encrypts it and sends it back.
- Alice waits for the user to type anything to kick off the request. She sends a packet with the encrypted ASCII string `GET`.
- After receiving the response from Bob, she decrypts it, prints it out and exits.

The Server (Bob)

Here is a specific list of things that Bob needs to do.

- Bob can be started before or after the KDC, but *must* be started before Alice.
- He should listen on a UDP port for Alice's plaintext message. He invents a nonce, wraps it up in his key and sends it as an ASCII payload back to Alice. Then he just waits.
- When Alice sends another message back to him, he should expect a ticket and encrypted nonce. He should decrypt the ticket, verify the nonce he sent to the KDC came back properly, use the session key to decrypt Alice's nonce and then modify it appropriately. He invents his own nonce, appends it to Alice's modified nonce and sends the whole payload back to Alice. He waits for her response.
- When Bob receives the `GET` request, he reads the file `ForAlice.txt` to find a short plaintext message. He should encrypt it with the session key and send it to Alice.

- Bob should then go back to listening for a new request from Alice. If Alice were to somehow send the same ticket back, he should *not* accept it.
- Bob should be able to handle another incoming authentication request if it should happen between when Alice is authenticated and when she sends the `GET` request.

Alice and Bob's Commonalities

Here are things that Alice and Bob have in common.

- Alice and Bob should both be very verbose. Whenever a message is sent or received, they should print out information to the screen (e.g., "Received encrypted message from Alice. Decrypted nonce is: 0x12345678.").
- In terms of "inventing nonces", you are free to decide how you want to do this. Part of your grade will be determined by how you choose your nonces. Let all of them be 32-bit values.
- The communication between Alice and Bob should be over UDP with the port number being the last 4 digits of your M-number (provided this doesn't conflict with other established services). If you do have a problem, just use port 4723.

The KDC

Here are the specific things the KDC needs to do.

- The KDC should be listening on UDP port 8888 (that's twice as many 8's as Kerberos uses!) for incoming requests from Alice. When the request comes in, it should:
  - Break up the payload appropriately,
  - Look up Alice's key
  - Look up Bob's key
  - Create the ticket for Bob (again, by concatenating ASCII strings)
  - Build the full payload for Alice
  - Encrypt it
  - Send it
- Then the KDC should just go back to waiting for another request.
- The KDC can be started up before or after Bob, but *must* be started before Alice.

- It should do basic checking on the contents of Alice's request. If she doesn't request Bob or doesn't properly self-identify, the KDC can simply ignore the request.
- The KDC can use static keys for Alice and Bob and even hardcode them in. There is no need to provide a registration service to add new users.

Universal Commonalities

We need to select a common encryption algorithm, so let's use AES-256 for everyone. We'll also use the same keys:

- Alice: `23FCE5AE61E7BFCB29AC85725E7EC77DB9DBA460EACA7458070B719CE0B1DC31`
- Bob: `A41503A5D9E66B34FAC9F2FC9FD14CA24D728B17DE0FCC2C3676DED6A191A1F1`

The KDC doesn't have a key in Expanded N-S. You can choose whatever key you would like as the session key for Alice and Bob.

Grading

This project is worth 300 points. The test is worth 225 points. Detail in a README file how you chose to generate your nonces and explicitly mention what line(s) of code do this. Your generation of nonces is worth 25 points. Subjectively, the quality and usefulness of the messages each program prints out will be worth 25 points. (If you have questions about how I feel about your messages, send me a quick email or stop by my office.) The fact that Bob and the KDC should keep running after the each interaction is worth 25 points. The programs should run on a university Linux environment, a Mac or a Windows 10 environment. Submit *everything* possible. If I need to install something myself (such as PyCryptodome), detail this in your README file.