

Tutorial 4

Orfeas Stefanos Thyfronitis Litos
Computer Security
School of Informatics
University of Edinburgh

February 27, 2019

This is the first coursework for the Introduction to Computer Security course. You are asked to mount a (Wo)man-in-the-Middle (MitM) attack against the provided toy implementation of an encrypted chat between terminals.

You are free to discuss these questions and their solutions with fellow students also taking the course, and also to discuss in the course forum. Bear in mind that if other people simply give you the solution directly, you may not learn as much as you would by solving the coursework for yourself; also, it may be harder for you to assess your progress with the course material.

1 High-level overview

When Alice and Bob hear about encryption, they immediately set out to implement an encrypted chat client so that they are sure no one eavesdrops their intimate discussions. They decide to use AES to encrypt their messages, since everyone says it's the best. They also hear of the Diffie-Hellman key exchange (DHKE) and figure it would be cool to use a new secret key for AES every time they connect.

1.1 AES

Just like every symmetric encryption scheme, AES consists of two algorithms:

- The encryption algorithm takes a key K_1 and a message M_1 as input and returns a ciphertext C_1 as output: $C_1 = \text{Enc}(K_1, M_1)$
- The decryption algorithm takes a key K_2 and a ciphertext C_2 as input and returns a message M_2 as output: $M_2 = \text{Dec}(K_2, C_2)$

If a message M is encrypted with key K and the resulting ciphertext C is decrypted with the same key K , the result of the decryption will be the original message M : $\forall K \forall M, M = \text{Dec}(K, \text{Enc}(K, M))$

A simple library for encrypting and decrypting using `pyaes` is provided in `symmetric.py`.

1.2 Diffie-Hellman Key Exchange

This is a protocol between two parties (say Alice and Bob) that want to obtain a common key that is unknown to anybody else. Their communication takes place over an insecure channel that anyone can eavesdrop.

A physical-world equivalent is the following: A group of people sit around a table and two of them want to speak in private. They can have a brief exchange (of very long numbers) *which*

everybody hears. After that they will possess a common secret *that no one else knows*. They can use this secret as the key for encrypting, sending and decrypting private messages in plain sight.

We assume that both parties have agreed beforehand on a finite cyclic group G and a generator g of G . For production software, these parameters are standardised by cryptographers and hardcoded in the implementation by the developers.

These are the steps of the protocol:

- Alice chooses a random number s_a and calculates $a = g^{s_a}$.
- Alice sends a to Bob.
- Bob chooses a random number s_b and calculates $b = g^{s_b}$.
- Bob sends b to Alice.
 - Now all eavesdroppers know a and b , but not s_a and s_b .
- Alice derives the common secret b^{s_a} .
- Bob derives the common secret a^{s_b} .

Given that $(g^x)^y = (g^y)^x$, both Alice and Bob have derived the same common secret. Assuming that an eavesdropper cannot find s_a from a or s_b from b , we conclude that no one else can derive the common secret.

A simple library for doing the necessary steps of DHKE is provided in `diffie_hellman.py`. You can see how to use it in the `do_Diffie_Hellman()` function in `util.py`.

1.3 Putting it all together

The entire process of chatting is then as follows:

1. Alice and Bob establish a communication socket
2. They do DHKE over this socket
3. Bob encrypts his message under the derived key (with AES)
4. Bob sends the resulting ciphertext through the socket
5. Alice decrypts the ciphertext using the derived key
6. Alice reads the message

Steps 3–6 can be repeated as many times as desired, possibly with changed roles. (In our implementation, the process is repeated only twice, so Bob sends first, then Alice, then both parties terminate.)

1.4 MitM attack

The described approach sounds very reasonable. Unfortunately Alice and Bob overlooked a fatal flaw: When communicating over the internet (or even locally), one cannot know with certainty that they are speaking to the intended party, at least not without using some form of cryptographic *authentication*.

Going back to our round-table example, consider the case where every member of the group wears a different mask, uses a voice jammer and sits at random seats. Alice would be unable to

recognize Bob. In an even worse scenario, if Bob happens to be missing from the table, someone with a good disguise could impersonate him and fool Alice into performing DHKE with him. This is why Alice and Bob should have agreed to only speak to each other after authenticating themselves.

Given that no authentication takes place, Eve the attacker is now able to do the following: After Bob opens his end of the socket and before Alice connects, Eve connects and performs a DHKE with Bob. Eve then opens a new socket and waits for Alice to connect. When Alice and Eve connect, they perform another DHKE. Now Eve can decrypt messages from one party, read them and reencrypt them for the other party. If she so wishes she can even send arbitrary messages, completely unrelated to the original ones. In short, she has complete control of the channel while Alice and Bob think they communicate with each other.

2 Implementation details

2.1 How to use the provided code

Open two terminals and navigate to the directory with the scripts. First run `python bob.py` in one and then `python alice.py` in the other (the order is important). You should see secure channel establishment, a couple of messages being exchanged and finally channel closing.

2.2 Code overview

Open both aforementioned scripts with your favourite editor. Each of the two scripts calls `setup()` with its name and the name of the pre-agreed buffer file over which communication happens. This name is set in `const.py`. Then Bob waits for a message, while Alice sends it. Bob then prints the message and the roles are reversed. Finally both parties close their sockets.

Familiarise yourself with the scripts and understand which lines correspond to each of the steps above. You can optionally dive in the code of the various supporting scripts as well.

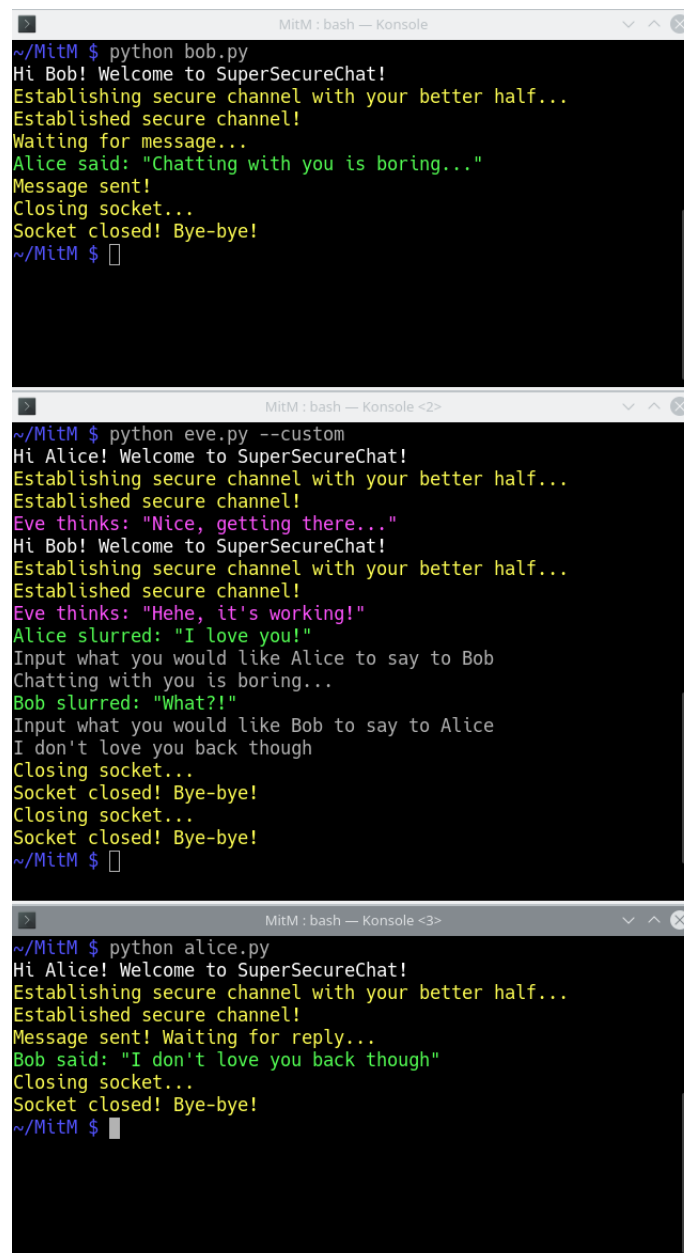
2.3 Exercise

You will have to implement and submit `eve.py`. The attack should execute correctly when first `bob.py` is started in one terminal, then `eve.py` in a second and last `alice.py` in a third. `eve.py` should be followed by exactly one of the following three flags: `--relay`, `--break-heart` or `--custom`.

- If the flag is `--relay`, Eve should just relay the two messages from Bob to Alice and from Alice to Bob. In this case, the outputs of both `alice.py` and `bob.py` in the terminals should be identical to the case when the MitM attack isn't executed.
- With the `--break-heart` flag, Eve should change the messages so that Bob receives the message "I hate you!" and Alice receives "You broke my heart...". Remember, Eve still has to encrypt both messages correctly.
- As for the `--custom` flag, after receiving Alice's message, Eve must prompt the user to input a message to the terminal and then must send this message to Bob instead. The same should happen for Bob's message; Eve must prompt the user for a second message and this time send it to Alice.

Hint: Your solution will have to use the buffer file somehow. The function `os.rename()` will prove helpful.

Note: It may happen that a script dies without closing its socket gracefully. In that case, you should manually remove the remaining buffer file (by default called `buffer`) before restarting the scripts.



```
MitM : bash — Konsole
~/MitM $ python bob.py
Hi Bob! Welcome to SuperSecureChat!
Establishing secure channel with your better half...
Established secure channel!
Waiting for message...
Alice said: "Chatting with you is boring..."
Message sent!
Closing socket...
Socket closed! Bye-bye!
~/MitM $

MitM : bash — Konsole <2>
~/MitM $ python eve.py --custom
Hi Alice! Welcome to SuperSecureChat!
Establishing secure channel with your better half...
Established secure channel!
Eve thinks: "Nice, getting there..."
Hi Bob! Welcome to SuperSecureChat!
Establishing secure channel with your better half...
Established secure channel!
Eve thinks: "Hehe, it's working!"
Alice slurred: "I love you!"
Input what you would like Alice to say to Bob
Chatting with you is boring...
Bob slurred: "What?!"
Input what you would like Bob to say to Alice
I don't love you back though
Closing socket...
Socket closed! Bye-bye!
Closing socket...
Socket closed! Bye-bye!
~/MitM $

MitM : bash — Konsole <3>
~/MitM $ python alice.py
Hi Alice! Welcome to SuperSecureChat!
Establishing secure channel with your better half...
Established secure channel!
Message sent! Waiting for reply...
Bob said: "I don't love you back though"
Closing socket...
Socket closed! Bye-bye!
~/MitM $
```

Figure 1: `eve.py --custom` execution example