CO876 Week 19 Class Worksheet

Budi Arief (b.arief@kent.ac.uk) – Based on materials by Shujun Li
Class Supervisors: Yichao Wang (Y.Wang-301@kent.ac.uk) and Joseph Kearney (J.Kearney@kent.ac.uk)

Friday 10 December 2021

In this week's class, you will work on some exercises beyond user authentication to gain a better understanding of authentication in the wider sense. The first three are easier and mandatory, while the rest are more advanced and optional. The latter can be used for Part 2 of your CW2 assignment if you so wish.

Like in the last week's class, for this week's class exercises, **you are encouraged to use existing third-party libraries and tools as much as you can**, as long as you will still write some source code yourself to get the exercises done. You should use Google and/or other search engines to get help and look for hints. Work independently first before asking for help.

For all exercises, feel free to show the work you have completed to class supervisors; but if you are confident with your results and do not feel the need to get them checked, that is also fine.

Note: This class is the third class that you will need to include in Part 1 of your CW2 practical report (i.e. the log book). *Please keep any evidence of the work you achieved (e.g. as screenshots or answers to specific exercises), and include them in the practical report that you will submit as part of CW2.*

Exercise 1 – Message authentication with HMAC (hash-based message authentication code)

Define a shortened HMAC algorithm to produce 16-bit HMACs. (Hint: Look for a library supporting HMAC algorithms and then truncate the returned MAC.)

Imagine that Alice is sending £10 to her friend Bob via her bank. Assume that Alice shares a secret key with the banking server, and agrees with the banking server to use the above 16-bit HMAC algorithm to authenticate any transaction requests.

Now assume that there is an attacker Eve who controls the communications channel between Alice and the banking server, and can see the messages transmitted in clear and make changes to the communications. Eve did not know Alice's key shared with the banking server. Write a script to answer the following questions:

- If Eve just changes the message to something like "Alice, Eve, £1000" without changing the HMAC, will the server reject the manipulated message?
- If Eve keeps changing the transfer amount and sending a manipulated message to the server, how many times does he need to try so that he can finally produce a message accepted by the server? How does this relate to the size of the HMAC values (16 bits)? Assume the banking server does not block repeated unsuccessful attempts. (In real world, the server will normally tolerate up to a fixed number of consecutive failures.)

Assuming the key shared between Alice and the banking server was derived from a <u>weak</u> password chosen by Alice following a <u>public</u> (<u>known</u>) key derivation algorithm. Can Eve find a way to crack the key (password)? No need to write a separate script to crack an example weak password (which you should have done in previous weeks). Just include a discussion in your summary of this exercise in Part 1 of your CW2 report.

Exercise 2 – Needham-Schroeder protocol

Write a script to demonstrate how Needham-Schroeder protocol shown on Slide 30 of Lecture 19 (covered on Monday 6 December 2021, i.e. the first lecture in Week 19) works.

- Assumptions:
  - A and B both trust a server S (an identity provider).
  - A and S share a secret key K_{AS}.   **Symmetric-key version**
  - B and S share a secret key K_{BS}.
- The goal
  - For A and B to establish a session key K_{AB}.
- The protocol
  1. A → S: A, B, N_A **N_A is a nonce.**
  2. S → A: $\{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$ **$\{x\}_K$ = using K as the key to encrypt x**
  3. A → B: $\{K_{AB}, A\}_{K_{BS}}$
  4. B → A: $\{N_B\}_{K_{AB}}$ **N_B is also a nonce.**
  5. A → B: $\{N_B - 1\}_{K_{AB}}$

30

No need to use three separate processes to simulate the protocol. It is sufficient to use a single script that simulate how the three parties (A, B and S) interact with each other to finally allow A and B to agree on a key (which is generated by S for A and B).

Print all useful information for all steps to show the internal working mechanisms of the protocol. Include authentication of each message when it is relevant and stop the protocol if a message does not pass authentication.

The output of the script should look like what is shown in the screenshot of an example PHP script (which was run on a Windows platform).

```
C:\Windows\System32\cmd.exe                                                      —  □  ×

C:\!hooklee\Work\Teaching\Uni-Kent\CO876 Computer Security\2020-21\classes\Week10>php Exercise2.php

Pre-shared key between Alice and Server: 0x5af501bdd1e71c9c8650508fb01a9b0e1db766f318329e46b4abe1366895c89a
Pre-shared key between Bob and Server: 0x16ebe3ae1cae4a1d48607b68896f7a7f1d5cbe20bb431ddcc5bde778787b5506

1 (Alice): N_A = 4969678081353687032
1 (Alice => Server): (A, B, N_A) = (Alice, Bob, 4969678081353687032)

2 (Server): K_AB = 0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2
2 (Server): E_{K_BS}(K_AB, A) = E_{0x16ebe3ae1cae4a1d48607b68896f7a7f1d5cbe20bb431ddcc5bde778787b5506}(0x09f173d29498c59
c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2, Alice) = 0xb2e84e3337a965b53135f0c7c78c520a68f6e1d4ecff97c187b8669b4b
c6629a5913bd49a1e7805e8dc01eba4f4d1e7060896ff8d9bbf4b857f9f375d3f8392cedddcfad68175e54a4046be3a86a0210
2 (Server => Alice): E_{K_AS}(N_A, B, K_AB, E_{K_BS}(K_AB,A)) = E_{0x5af501bdd1e71c9c8650508fb01a9b0e1db766f318329e46b4a
be1366895c89a}(4969678081353687032, Bob, 0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2, 0xb2e84e333
7a965b53135f0c7c78c520a68f6e1d4ecff97c187b8669b4bc6629a5913bd49a1e7805e8dc01eba4f4d1e7060896ff8d9bbf4b857f9f375d3f8392ce
dddcfad68175e54a4046be3a86a0210) = 0xff23bd841c076d34d39c9c263befe5d45ded06fa4ae69c622a1e434c0fe2787a90bcd3a88558aff2e54
4914fb6d5763a0754d52429307851bb655328d9ab4c179dd5cdffbf9e8c49047a499f9299878af5fe16f6d6198671f931da4fd913f78e5e09adf77e0
625c89e3d7df7494d29512be0f03e6a8104ccba5205db326fec747668872749acfdeaae516b67fa3448475119b9c9579693e4ad1073628d14b351a8e
9b1b4ac14612a6f4f86c6711b649c56e430ec5220fc96d32265d16f45a5fdc69bd28fe9c3205b8563c59cf7568f934127399e4a86f7a836eb68da9d6
aa4d420cb800a7fc5c5ee348e7920d9c19d7166660a1cc8871af7b470e3bed3742c8f
2 (Alice): (N_A, B, K_AB, E_{K_BS}(K_AB,A)) = (4969678081353687032, Bob, 0x09f173d29498c59c4613bc51190ccd23b158966468ff6
bd7fdcc66d96811e0f2, 0xb2e84e3337a965b53135f0c7c78c520a68f6e1d4ecff97c187b8669b4bc6629a5913bd49a1e7805e8dc01eba4f4d1e706
0896ff8d9bbf4b857f9f375d3f8392cedddcfad68175e54a4046be3a86a0210)
=> Message 2 authentication was successful!

3 (Alice => Bob): E_{K_BS}(K_AB, A) = E_{0x16ebe3ae1cae4a1d48607b68896f7a7f1d5cbe20bb431ddcc5bde778787b5506}(0x09f173d29
498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2, Alice) = 0xb2e84e3337a965b53135f0c7c78c520a68f6e1d4ecff97c187b8
669b4bc6629a5913bd49a1e7805e8dc01eba4f4d1e7060896ff8d9bbf4b857f9f375d3f8392cedddcfad68175e54a4046be3a86a0210
3 (Bob): (K_AB, A) = (0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2, Alice)
=> Message 3 authentication was successful

4 (Bob): N_B = 6405442935968965580
4 (Bob => Alice): E_{K_AB}(N_B) = E_{0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2}(640544293596896
5580) = 0x8411aefb994ca7dc45e8b7fb8d12bd1b
4 (Alice): N_B = 6405442935968965580
=> Message 4 authentication was successful!

5 (Alice => Bob): E_{K_AB}(N_B-1) = E_{0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2}(6405442935968
965579) = 0x62500174060800a4d88c023b7988f4b4
5 (Bob): N_B-1 = 6405442935968965579
=> Message 5 authentication was successful!

The key agreed between Alice and Bob: 0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2

C:\!hooklee\Work\Teaching\Uni-Kent\CO876 Computer Security\2020-21\classes\Week10>
```

Exercise 3 – Attacking Needham-Schroeder protocol

Modify the script you implemented in Exercise 2 above to demonstrate how the replay attack described on Slide 31 of Lecture 19 (in Week 19) works for an attacker.

Needham-Schroeder protocol: an attack

University of Kent

- Assumptions
  - E (an attacker) somehow got K_{AB}.
    - How?
  - E observed the session between A and B when K_{AB} was established.
- The goal
  - E wants to reuse K_{AB} to impersonate A to communicate with B in future.
- The (replay) attack
  3. E → B: $\{K_{AB}, A\}_{K_{BS}}$
  4. B → E: $\{N_B\}_{K_{AB}}$
  5. E → B: $\{N_B - 1\}_{K_{AB}}$

B **will believe he is communicating with A after this step.**

31

Note that you need to record K_AB and Message 3 from a previous session of the protocol. You will also need the same K_BS assuming B has not changed her shared secret with S.


```
C:\Windows\System32\cmd.exe                                          —    □    ×

C:\!hooklee\Work\Teaching\Uni-Kent\CO876 Computer Security\2020-21\classes\Week10>php Exercise3.php

Unknown to Eve:
Pre-shared key between Alice and Server: [does not matter / unused in the attack]
Pre-shared key between Bob and Server: 0x16ebe3ae1cae4a1d48607b68896f7a7f1d5cbe20bb431ddcc5bde778787b5506

Known to Eve (collected from a previous session between Alice and Bob):
Pre-recorded K_AB: 0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2
Pre-recorded Message 3 (Alice => Bob): 0xb2e84e3337a965b53135f0c7c78c520a68f6e1d4ecff97c187b8669b4bc6629a5913bd49a1e7805
e8dc01eba4f4d1e7060896ff8d9bbf4b857f9f375d3f8392cedddcfad68175e54a4046be3a86a0210

3 (Eve => Bob): E_{K_BS}(K_AB, A) = E_{0x16ebe3ae1cae4a1d48607b68896f7a7f1d5cbe20bb431ddcc5bde778787b5506}(0x09f173d2949
8c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2, Alice) = 0xb2e84e3337a965b53135f0c7c78c520a68f6e1d4ecff97c187b866
9b4bc6629a5913bd49a1e7805e8dc01eba4f4d1e7060896ff8d9bbf4b857f9f375d3f8392cedddcfad68175e54a4046be3a86a0210
3 (Bob): (K_AB, A) = (0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2, Alice)
=> Eve successfully passed Message 3 authentication!

4 (Bob): N_B = 2968589878680208424
4 (Bob => Eve): E_{K_AB}(N_B) = E_{0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2}(29685898786802084
24) = 0xba5511f725b56ff0271972c510c8c5f3
4 (Eve): N_B = 2968589878680208424
=> Eve successfully decrypted Message 4 to get N_B!

5 (Eve => Bob): E_{K_AB}(N_B-1) = E_{0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2}(296858987868020
8423) = 0xa65bc20e64851874baeb25aa33754037
5 (Bob): N_B-1 = 2968589878680208423
=> Eve successfully passed Message 5 authentication!

Eve successfully launched a reply attack to reuse a previouslly recorded session key agreed between Eve and Bob:
0x09f173d29498c59c4613bc51190ccd23b158966468ff6bd7fdcc66d96811e0f2

C:\!hooklee\Work\Teaching\Uni-Kent\CO876 Computer Security\2020-21\classes\Week10>
```

Exercise 4 (optional) – Fixing Needham-Schroeder protocol

Revise the script you wrote for Exercise 2 to implement a more secure version of Needham-Schroeder protocol shown on Slide 32 of Lecture 19. Show that the replay attack demonstrated in Exercise 3 will not work anymore.

- Introducing a **timestamp** so old session keys cannot be simply reused.
  - This is a common security fix for many security protocols!
- Adding authentication of A to the protocol.
  1. A → B: A
  2. B → A: $\{A, N_B'\}_{K_{BS}}$ — (A, N_B') **binds the nonce** N_B' **with the sender** A.
  3. A → S: $A, B, N_A, \{A, N_B'\}_{K_{BS}}$
  4. S → A: $\{N_A, B, K_{AB}, \{K_{AB}, A, N_B'\}_{K_{BS}}\}_{K_{AS}}$
  5. A → B: $\{K_{AB}, A, N_B'\}_{K_{BS}}$
  6. B → A: $\{N_B'\}_{K_{AB}}$
  7. A → B: $\{N_B' - 1\}_{K_{AB}}$

32

Exercise 5 (optional) – Breaking a selected CAPTCHA scheme

Select an existing CAPTCHA scheme used by one or more real world websites, and write a script to break it with a sufficiently high accuracy.

References:

[1] Shujun Li, http://www.hooklee.com/CAPTCHA.asp, web page

[2] Shujun Li, http://www.hooklee.com/eBanking_CAPTCHAs.asp, web page

[3] Antreas Dionysiou and Elias Athanasopoulos, "SoK: Machine vs. machine – A systematic classification of automated machine learning-based CAPTCHA solvers," *Computers & Security*, vol. 97, Article No. 101947, Elsevier, 2020, https://doi.org/10.1016/j.cose.2020.101947