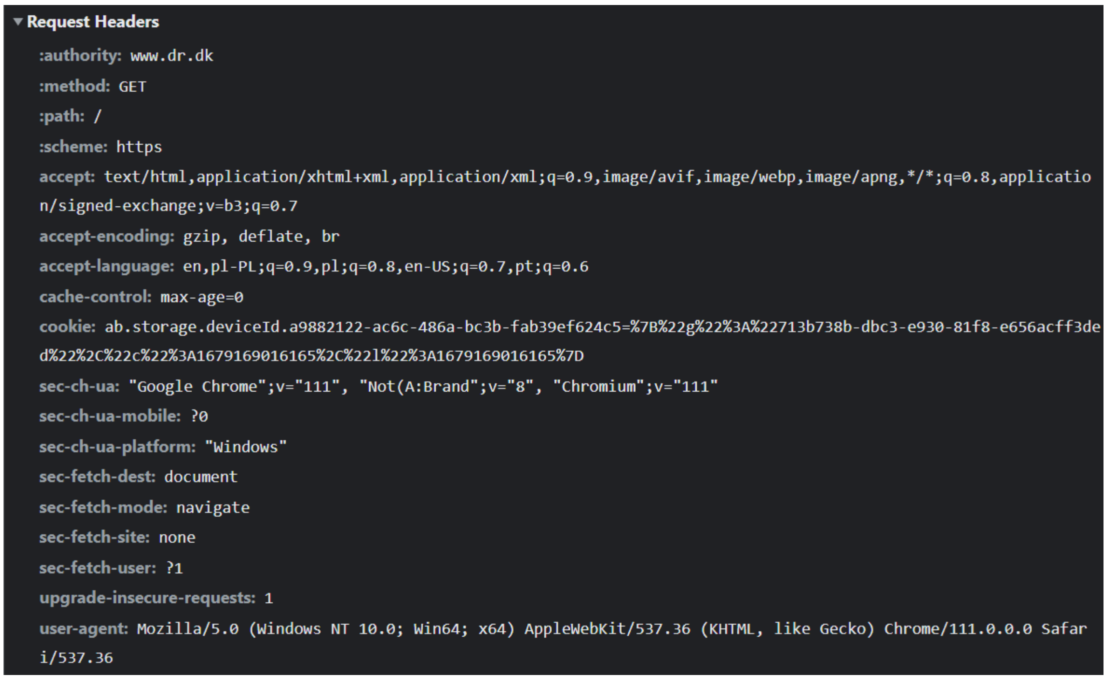
02233 Network Security
Private Communication - solutions

Exercise 1. (**What Does A Webserver Learn About You**)

In this exercise, we will see what information is leaked to a webserver whenever we access a page.

1. Identify what HTTP headers are part of your web browser's usual HTTP request. For that (if you use Chrome or Firefox) open the "Inspect" window, access a website of your choice and look at the HTTP headers your browser generates when accessing the page.
2. If you don't have it already, install an anti-tracking plugin such as Ghostery. Then access e.g. `dr.dk`, `dtu.dk` and `nytimes.com` (or your favorite websites) and check how many trackers they have and what they belong to. Can you identify which companies or websites belong to the respective trackers?

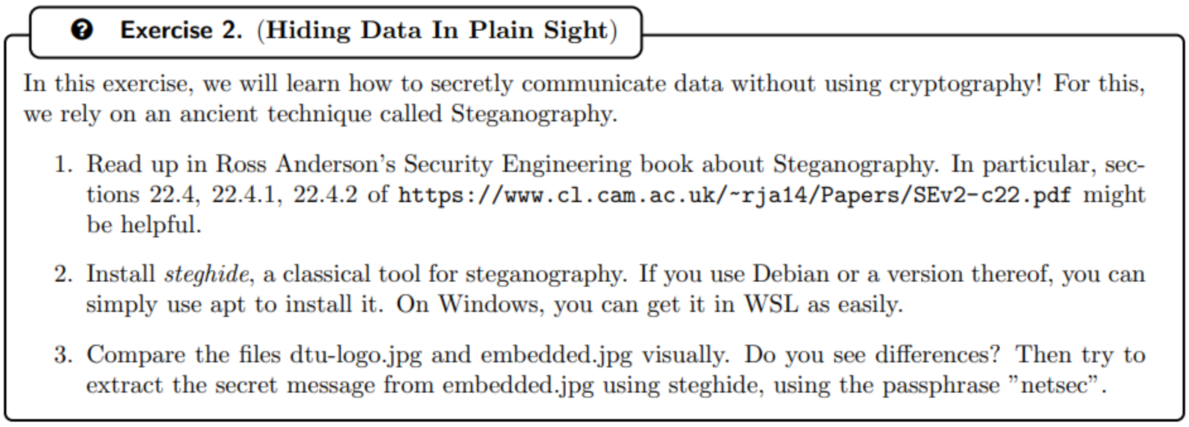
1. The HTTP request headers depend on:
   - the site the request is being sent to - based on the site's configurations,
   - the browser - privacy protecting browsers like Brave would be more likely to limit the request headers to the minimum.



```
▾ Request Headers
   :authority: www.dr.dk
   :method: GET
   :path: /
   :scheme: https
   accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,applicatio
   n/signed-exchange;v=b3;q=0.7
   accept-encoding: gzip, deflate, br
   accept-language: en,pl-PL;q=0.9,pl;q=0.8,en-US;q=0.7,pt;q=0.6
   cache-control: max-age=0
   cookie: ab.storage.deviceId.a9882122-ac6c-486a-bc3b-fab39ef624c5=%7B%22g%22%3A%22713b738b-dbc3-e930-81f8-e656acff3de
   d%22%2C%22c%22%3A1679169016165%2C%22l%22%3A1679169016165%7D
   sec-ch-ua: "Google Chrome";v="111", "Not(A:Brand";v="8", "Chromium";v="111"
   sec-ch-ua-mobile: ?0
   sec-ch-ua-platform: "Windows"
   sec-fetch-dest: document
   sec-fetch-mode: navigate
   sec-fetch-site: none
   sec-fetch-user: ?1
   upgrade-insecure-requests: 1
   user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safar
   i/537.36
```

2. For dr.dk Ghostery has identified 3 trackers:

- Cookiebot (by Cybot - https://www.cookiebot.com/en/)
- Site Analytics (by Segment - https://segment.io/)
- Dr.dk (internal pop-up)



Exercise 2. (**Hiding Data In Plain Sight**)

In this exercise, we will learn how to secretly communicate data without using cryptography! For this, we rely on an ancient technique called Steganography.

1. Read up in Ross Anderson's Security Engineering book about Steganography. In particular, sections 22.4, 22.4.1, 22.4.2 of `https://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c22.pdf` might be helpful.
2. Install *steghide*, a classical tool for steganography. If you use Debian or a version thereof, you can simply use apt to install it. On Windows, you can get it in WSL as easily.
3. Compare the files dtu-logo.jpg and embedded.jpg visually. Do you see differences? Then try to extract the secret message from embedded.jpg using steghide, using the passphrase "netsec".

1. -
2. Install with: *`sudo apt install steghide`*
3. Extract the message from embedded.jpg with:

   ```
   steghide extract -sf embedded.jpg -xf secret.txt
   ```

   extract, --extract
   Extract secret data from a stego file.

   -sf, --stegofile filename
   Specify the name for the stego file that will be created. If this argument is omitted when calling steghide with the embed command, then the modifications to embed the secret data will be made directly to the cover file without saving it under a new name.

   -xf, --extractfile filename
   Create a file with the name filename and write the data that is embedded in the stego file to it. This option overrides the filename that is embedded int the stego file. If this argument is omitted, the embedded data will be saved to the current directory under its original name.

Exercise 3. (Secretly Getting Data Out)

Assume you are a journalist and your friend Alice works for EvilCorp Inc. She desparately wants to give you information about their recent evil deeds, but unfortunately she cannot move data out of EvilCorp physically as USB ports are locked down etc.

1. Given the techniques you have learned so far in the course, in particular those in this lecture, devise a strategy for Alice to send the documents to you digitally.
2. Assume that EvilCorp also disallows the use of e-mail encryption and scans all e-mails or files if steganography was applied to files. Develop a strategy for how Alice could hide information in the HTTP headers when accessing a website, e.g. with the information you can find here `https://www.rfc-editor.org/rfc/rfc9110.html`.

1. Alice could use various techniques to transfer the files outside of the internal network perimeter such as:
   - Cloud storage where access is shared with the journalist,
   - email communication (with PGP encryption),
   - E2E encrypted messaging chat.

   Alice could also zip the files, protect them with a password and send them over via email. Then, she would send the respective password via a different channel.
2. Approach 1: Use the packet fragmentation and send chunks of messages through HTTP headers. The file could be first encoded with a suitable encoding mechanism such as base64. Alice could chose to create custom headers or use steganography techniques to organize headers in a certain manner, agreed upon with the journalist.

   Approach 2: Create a fake cookie and encode the file as cookie data.

   By using the approach 1, the Intrusion Prevention System could potentially recognize nonstandard HTTP headers, raise alarms and block the requests. However, with approach 2, IPS usually would not check the content of the cookie.

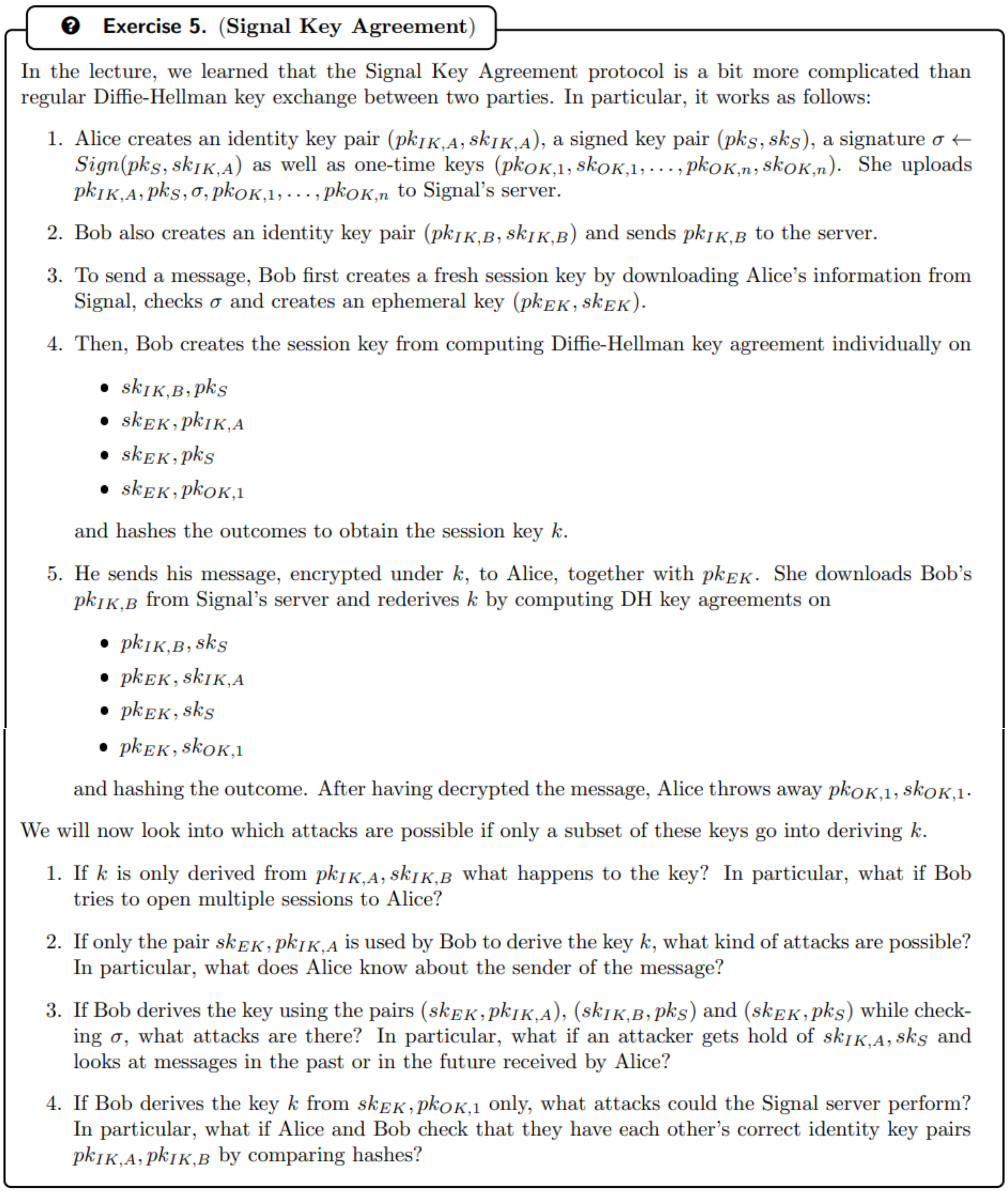
Exercise 4. (Using Tor)

In this exercise, we are trying out the Tor system. For this, we use the so-called Tor Browser which is a bundle of Firefox and Tor that you don't have to configure yourself.

1. Download the Tor Browser and use for fun. Do you notice any difference in response behavior for sites you know? Try to download a file of reasonable size and look at download speeds.
2. You likely googled for the Tor Browser, opened the first result, then downloaded and executed the file. What kind of attacks could a malicious actor which controls the network have done so that you don't download the correct executable?
3. The Tor project allows you to verify that the downloaded file is indeed correct. Follow the instructions on `https://support.torproject.org/tbb/how-to-verify-signature/`. Which assumptions does their verification procedure make and how could it be undermined?

1. Tor Browser is noticably slower due to the additional routing steps and the overhead of encrypting and decrypting traffic. For instance, downloading a Tor Browser installer executable from Google Chrome and Tor Browser took 8s and 3m 7s respectively.
2. Attacker controlling the network could potentially use the following attacks:
   a. Man-in-the-middle (MITM) attack: An attacker could intercept the network traffic between the user and the download server, and replace or modify the downloaded executable file in transit, adding malware or other malicious code to the file.
   b. DNS Spoofing: An attacker could spoof the DNS response for the download server, directing the user to a malicious server that serves a fake executable file containing malware.
   c. Website spoofing: An attacker could create a fake website that looks exactly like the legitimate site and eventually trick the user into downloading and executing a malicious file.
3. Procedure for verification:
   a. Download signature (Save link/file as…).
   b. Import the Tor Browser Developers signing key
   ```
   gpg --auto-key-locate nodefault,wkd --locate-keys
   torbrowser@torproject.org
   ```
   c. Save the key to a file
   ```
   gpg --output ./tor.keyring --export
   0xEF6E286DDA85EA2A4BA7DE684E2C6E8793298290
   ```
   d. Verify the signature
   ```
   gpgv --keyring .\tor.keyring Downloads\torbrowser-install-
   win64-9.0_en-US.exe.asc Downloads\torbrowser-install-win64-
   9.0_en-US.exe
   ```

   Assumptions:
   a. The website that the user is reading the instructions from is legitimate and not spoofed (the mentioned signing key could be replaced, along with the link https://keys.openpgp.org/vks/v1/by-fingerprint/EF6E286DDA85EA2A4BA7DE684E2C6E8793298290).
   The user must make sure that the website is official by verifying the SSL certificate.
   b. The user has downloaded a legitimate GPG tool.



Exercise 5. **(Signal Key Agreement)**

In the lecture, we learned that the Signal Key Agreement protocol is a bit more complicated than regular Diffie-Hellman key exchange between two parties. In particular, it works as follows:

1. Alice creates an identity key pair $(pk_{IK,A}, sk_{IK,A})$, a signed key pair (pk_S, sk_S), a signature $\sigma \leftarrow Sign(pk_S, sk_{IK,A})$ as well as one-time keys $(pk_{OK,1}, sk_{OK,1}, \dots, pk_{OK,n}, sk_{OK,n})$. She uploads $pk_{IK,A}, pk_S, \sigma, pk_{OK,1}, \dots, pk_{OK,n}$ to Signal's server.
2. Bob also creates an identity key pair $(pk_{IK,B}, sk_{IK,B})$ and sends $pk_{IK,B}$ to the server.
3. To send a message, Bob first creates a fresh session key by downloading Alice's information from Signal, checks σ and creates an ephemeral key (pk_{EK}, sk_{EK}).
4. Then, Bob creates the session key from computing Diffie-Hellman key agreement individually on
   - $sk_{IK,B}, pk_S$
   - $sk_{EK}, pk_{IK,A}$
   - sk_{EK}, pk_S
   - $sk_{EK}, pk_{OK,1}$

   and hashes the outcomes to obtain the session key k.
5. He sends his message, encrypted under k, to Alice, together with pk_{EK}. She downloads Bob's $pk_{IK,B}$ from Signal's server and rederives k by computing DH key agreements on
   - $pk_{IK,B}, sk_S$
   - $pk_{EK}, sk_{IK,A}$
   - pk_{EK}, sk_S
   - $pk_{EK}, sk_{OK,1}$

   and hashing the outcome. After having decrypted the message, Alice throws away $pk_{OK,1}, sk_{OK,1}$.

We will now look into which attacks are possible if only a subset of these keys go into deriving k.

1. If k is only derived from $pk_{IK,A}, sk_{IK,B}$ what happens to the key? In particular, what if Bob tries to open multiple sessions to Alice?
2. If only the pair $sk_{EK}, pk_{IK,A}$ is used by Bob to derive the key k, what kind of attacks are possible? In particular, what does Alice know about the sender of the message?
3. If Bob derives the key using the pairs $(sk_{EK}, pk_{IK,A})$, $(sk_{IK,B}, pk_S)$ and (sk_{EK}, pk_S) while checking σ, what attacks are there? In particular, what if an attacker gets hold of $sk_{IK,A}, sk_S$ and looks at messages in the past or in the future received by Alice?
4. If Bob derives the key k from $sk_{EK}, pk_{OK,1}$ only, what attacks could the Signal server perform? In particular, what if Alice and Bob check that they have each other's correct identity key pairs $pk_{IK,A}, pk_{IK,B}$ by comparing hashes?

1. Ephemeral key is generated for each execution of key establishment process – per every session. If an ephemeral key was not used for key derivation process, compromising a key protecting one session, would immediately compromise all the other sessions between Alice and Bob.
2. Alice has no idea who the sender of the message is. Everyone knows her identity public key, and anyone can create an ephemeral key pair. A man-in-the-middle attack would be possible if Bob's private key was not to be used.
3. If Bob didn't use the one-time key generated by Alice, the protocol would lose the Perfect Forward Secrecy property. If the attacker managed to compromise Alice's long-term keys, they would also be able to decrypt all the previous messages starting from when pk_S was published

on Signal's server – however with one-time keys it’s not possible, as once used, they are being discarded.

4. Signal could impersonate Alice to Bob, e.g. by providing their own pk_{OK1}. Then Bob would send a message to Alice and Signal would rederive k using their own sk_{OK1}. Normally, hash comparison between Alice and Bob would solve the problem of identity verification, but since the hash is not computed from their long-term identity secret keys, the verification would not be sufficient - it only makes sense if both parties use the locally generated key (which Signal doesn’t know).