# INF143A 23V Applied Cryptography
# Third mandatory assignment

Nikolay Kaleyski

## 1 General information

- **Deadline:** The assignment is due on **May 10, 2023**, at 23:59.
- **Submission:** A copy of the solution should be uploaded to mitt, under the "Assignments" tab. Please put all of your files in a single ZIP archive.
- **Score:** The mandatory assignment accounts for 15% of the final grade.
- **Collaboration:** You can freely discuss the assignment with each other, but the solutions must be prepared individually; plagiarism will result in all involved parties failing the assignment.
- **Passing grade:** You must obtain at least 20% to successfully pass the assignment.

## 2 Problems

**Important:** For all problems, please ensure that the output provided by your program is in the same format as that of the sample output files. This is because your solution will in part be graded by automatically comparing its output against sample output produced by our implementation. Having extra spaces, commas, newlines, etc. will result in a mismatch. Thus, to get maximum points and avoid any possibility of a misunderstanding, please ensure that your output **exactly matches the format of the sample output files**.

**Problem 1.** ***45 % NB:*** *The task asks you to construct a MAC using the HMAC construction. Using an HMAC implementation from a library is not allowed since the purpose of the problem is to test your knowledge of the construction.*

*In this problem, you will implement a* ***message authentication code (MAC) from a hash function*** *using the HMAC construction (as described in the slides from the lecture, or in Applied Cryptography by Paar and Pelzl). This will be based on the SHA-256 hash function.*

*Create a Python script called* `hmac.py`*, which can be run like*

```
python hmac.py file key output_file
```

*and should write the generated MAC to the output file.*

*When implementing your solution, you can use the auxiliary functions for reading and writing files in binary from* `cipher.py`*, and you can use the implementation of SHA-256 in Python's* `hashlib` *library. You can assume that the key is always exactly equal to the block size, so it is not necessary to expand it.*

***NB:*** *The implementation of SHA-256 provided in* `hashlib` *follows the* `FIPS PUB 180-4` *standard and has a fixed IV, so no IV file is provied on the input.*

*A sample set of files is provided by* `hash_in`*,* `hash_key`*, and* `hash_out`*.*

**Problem 2.** ***35 %*** *In previous problems, you have implemented block ciphers and modes of operation that work on a single block of data, or on files whose size is an exact multiple of the block size. In particular, Problem 1 of the second assignment asked you to implement the CBC mode of operation, assuming that the input file can be evenly divided into blocks.*

*In this problem, you will* ***add padding*** *to your CBC implementation so that it can handle files of any size. You can use your own implementation of the CBC mode of operation, or you can start from the sample implementation provided in* `cbc.py`*.*

*You will use the following padding scheme which is a variant of PKCS#5 and PKCS#7 padding. The general principle is that if p bytes need to be added in order to get a complete block, then p bytes with value p are added.*

- *Since the entire block is of size 16 bits, it can be seen as consisting of 2 bytes.*
- *Data from a file is read in sequences of bytes, so we should at most most need to pad by a single byte.*
- *If a single byte needs to be added to get a complete block, then we append a byte with value* 1*.*
- *If no padding is necessary, we append two bytes of value* 16*.*

*Your program should be able to correctly encrypt and decrypt any file on your computer, regardless of whether its size is a multiple of the block size or not. The program should be run by*

```
python penc plaintext key iv output_file
```

*and*

```
python pdec ciphertext key iv output_file.
```

*A sample set of input/output files is given as* `pad_in`*,* `pad_key`*,* `pad_iv` *and* `pad_out`*.*

**Problem 3.** ***20 %*** *If you visit www.uib.no, you will observe that the connection is using the HTTPS protocol. This means that UiB identifies itself using a digital signature which can be verified using UiB's public key; and the fact that what is shown as UiB's public key really is UiB's public key is guaranteed by the signature of a certification authority, whose public key is guaranteed by an even higher level authority, etc.*

*For this problem, you will use the tools available in your browser to investigate the chain of trust guaranteeing the security of the HTTPS connection to UiB. More precisely, find all entities involved in the chain of trust (starting from UiB), and for each of them, list:*

- *what digital signature algorithm was used to authenticate them;*
- *what their public key is (this should include all necessary information to verify the signature).*

*Submit your solution as a* ***series of screenshots*** *showing how and where you found the necessary information. Please* ***include your name*** *somewhere on the computer screen when taking the screenshot (for example, open a text file in Notepad and write your name there).*

***NB:*** *Different browsers and different version of browsers may provide information about certificates in different formats and with a varying level of detail. If you are not able to obtain the necessary information using your current browser, you can try using a different version or different software altogether.*