LZSCC.252 — Secure Cyber Systems

2024–2025 ASSESSMENTS

UG Part II

Lancaster University
Leipzig

Coursework

Released:	17 February 2025
Assessment Weight:	30%
Submission Deadline:	21 March 2025, 15:00 (Leipzig time)
Submission Box:	Moodle

Academic Honesty and Integrity

Students at Lancaster University are part of an academic community that values trust, fairness and respect and actively encourages students to act with honesty and integrity. It is a University policy that students take responsibility for their work and comply with the university's standards and requirements- found in the Manual of Academic Regulations and Practice. By submitting their answers students will be confirming that the work submitted is completely their own. Academic misconduct regulations are in place for all forms of assessment and students may familiarise themselves with this via the university website: https://www.lancaster.ac.uk/academic-standards-and-quality/regulations-policies-and-committees/manual-of-academic-regulations-and-procedures/

Plagiarism

Plagiarism involves the unacknowledged use of someone else's work and passing it off as if it were one's own. This covers every form of submitted work, from written essays, video vignettes, and coding exercises. However, deliberately plagiarism with the intent to deceive and gain academic benefit is unacceptable. This is a conscious, pre-meditated form of cheating and is regarded as a serious breach of the core values of the University. More information may be found via the plagiarism framework website. All coursework is to be submitted electronically and will be run through our plagiarism detection mechanisms. Please ensure you are familiar with the University's Plagiarism rules and if you are in any doubt please contact your module tutor.
See also: https://portal.lancaster.ac.uk/ask/plagiarism/

Use of Generative AI

For this assignment, you are not allowed to use generative AI, i.e., it falls into category RED of the university's guidance on the use of generative AI.
See also: https://portal.lancaster.ac.uk/ask/university-position-ai/

General Guidance

This is an individual assignment that will count for 30% of your overall marks for this module. The marks will be based on (a) the correctness of the solution, and (b) the corresponding explanation (e.g., comments in code) on how the solution was achieved. You can be awarded at most 30 points in this coursework. Solve all three questions.

Submission

You must submit a ZIP file containing code as source files and any explanations, reasoning, or comments either included in your code as comments, or as separate plain text files (.txt) or PDF documents. Any code or raw data submitted in PDF or Word documents will NOT be accepted.

Questions

1. (a) $\boxed{\text{2 points}}$ Explain in 2–3 sentences why the hash digest of a message alone cannot prove the authenticity of a message.

   (b) Consider the following hash function h: For a bytesequence m (e.g., an instance of Python's data type bytes), count the number of bits which are 0. Return this count (i.e., the digest value).

      i. $\boxed{\text{3 points}}$ Provide a Python implementation of this hash function. Use the skeleton code provided below.

```python
def hash_by_counting_zero_bits(message):
  # TODO implement the hash function
  return digest_value
```

      ii. $\boxed{\text{1 point}}$ Find a collision, i.e., two messages for which h outputs the same digest value.

   (c) The following hash digest (in hexadecimal notation) has been produced using SHA-256 for a message consisting of five lower-case letters of the Latin alphabet (i.e., a, ..., z).
   `aa3a05d9d57c08d694cf230a36386221f56890092e54cf8d6b04fab1b2cc0702`

      i. $\boxed{\text{3 points}}$ Implement a Python function to determine the plaintext message. Explain your strategy to determine the message. Use the skeleton code provided below.

```python
def find_message_for_sha256_digest_value():
  # TODO implement your search function
  return plaintext_message
```

      ii. $\boxed{\text{1 point}}$ State the message which is hashed to the value provided above.

2. Consider the following cryptosystem $S = (P, C, K, E, D)$ with

   - the message and crypto-text space $P = C = \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}\}$ (i.e., uppercase letters),
   - the key space $K = \{(k_0, \ldots, k_{25}) \in C^{26} | \forall i \neq j : k_i \neq k_j\}$ (i.e., each key is a vector of 26 pairwise disjoint characters),
   - the encryption function $E_k(m)$ which looks up the ciphertext c in the vector $k = (k_0, \ldots k_{25})$ as follows: if m is `A`, return k_0, if m is `B`, return k_1, and so on,
   - the decryption function $D_k(c)$, such that $\forall k \in K, m \in P : m = D_k(E_k(m))$.

   (a) $\boxed{\text{4 points}}$ Provide an algorithmic description of the decryption function D, e.g., in pseudocode or as a Python implementation. Explain in 2–3 sentences how your algorithm works.

   (b) $\boxed{\text{2 points}}$ Which effect can you observe in cryptosystems (such as the one above) when you encrypt the same character multiple times?

   (c) $\boxed{\text{4 points}}$ An English text has been encrypted using the ECB mode (i.e., each letter has been encrypted independent of each other). The ciphertext is provided in Figure 1. Your task is to find a correct key and decrypt the text. Describe your approach and each step to analyse the ciphertext.

   Hint: It is quite infeasible to try all possible keys. However, you can take advantage of typical letter frequencies in the English language. Count the occurrence of each letter in the ciphertext and — based on how often the letter appears in the text — state a hypothesis (or several!) to which plaintext letter it corresponds. Refer to Figure 2 for letter frequencies of the English language.

Provide the plaintext and a correct key. Explain how you obtained both and provide evidence of how you solved this challenge. Such evidence includes, e.g., the hypothesis for each step of finding the key. If you write code which helped you to find the solution, include this code as well.

```
ZXIOI OYDQH WDZJC FDGYZ OWLAC FQYDX WPAQO ACHDQ VIWKO PQZZQ DOYLF
ZQPQM DYAOX YVQEQ HZCWK HDQZZ CGQPP KHZWF YZQYO ZWYPP ZXYZX YOWUU
KDDQF ILZXI OAWOZ MWFBW DOYEQ LUWDL QDWBZ XQGWD PFZXQ PWLMQ DWLQO
ZYCOX QDQZX QAWDQ FWQOZ XQOHI DIZWB ZXQAW WDOIL EILZW WLQOO WKPIZ
OVYOZ LQOOY LFYPO WIZOM DIAUX YDAGX QLCWK YDQWL UQWKZ KHWLI ZOJWO
WACWK XYVQP QBZYP PZDYU QOWBA WFQDL QLMPY LFJQX ILFCW KJKZW LZXQW
ZXQDX YLFCW KYDQU WLOUI WKOQV QDCGX QDQWB ZXQXW AQOYL FZXQG WDEWB
ZXQHD QXIOZ WDIUH QWHPQ
```

Figure 1: An encrypted text (spaces have been added for readability). This ciphertext is also provided in a separate file.

Letter	Frequency	Letter	Frequency	Letter	Frequency	Letter	Frequency
A	0.08167	H	0.06094	O	0.07507	V	0.00978
B	0.01492	I	0.06966	P	0.01929	W	0.02360
C	0.02782	J	0.00153	Q	0.00095	X	0.00150
D	0.04253	K	0.00772	R	0.05987	Y	0.01974
E	0.12702	L	0.04025	S	0.06327	Z	0.00074
F	0.02228	M	0.02406	T	0.09056		
G	0.02015	N	0.06749	U	0.02758		

Figure 2: Relative Frequencies of Letters in General English Plain text (source: *Cryptographical Mathematics, by Robert Edward Lewand*)

3. In this task, you will write a Python program that implements the following MAC scheme:

   We consider messages of a length which is a multiple of 16 bytes. We use keys of 16 bytes length. The messages are handled as a sequence of blocks of 16 bytes length each.

   The MAC tag value is calculated as depicted in Figure 3. That is, each block is XORed with the result of the previous block, then it is encrypted using the AES cryptosystem. As there is no "previous block" for the first block, we consider that the bits of the "previous block" are all 1, i.e., `b'\xFF'*16`. The calculation result for the encryption of the last (XORed) block is then returned as the MAC tag.

   (a) 4 points Implement a Python function which calculates a MAC tag value for a message.

   (b) 2 points Produce a MAC tag value for your student id number using some random key. Provide the precise input (i.e., how you encoded your student id number), the key, and the output of your function as copyable text such that it can be verified without manually typing any of the values (a screenshot would **not** satisfy this criterion).

   (c) 4 points Implement a Python function which verifies a message using a given MAC tag value and key.

   Use the skeleton code provided below.

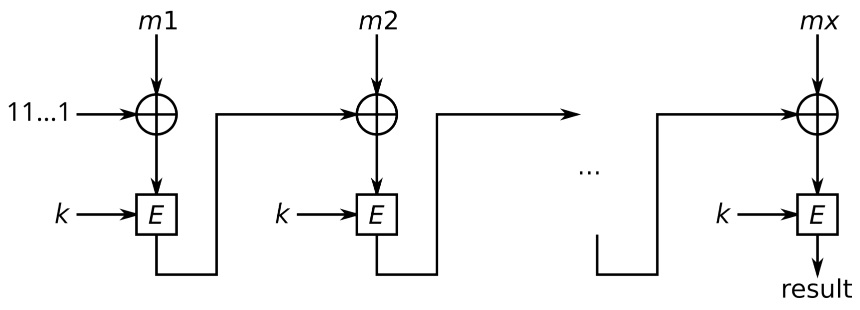

Figure 3: MAC construction

```python
def mac(message, key):
  # TODO implement the MAC function
  return tag
def verify(message, key, tag):
  # TODO implement the verification function
  return True_or_False
```