Department of Computing and Mathematics

ASSESSMENT COVER SHEET 2024/25

Module Code and Title:	(6G6Z0024) Applied Cryptography
Assessment Set By:	Dr Killian O'Brien
Assessment ID:	1CWK100
Assessment Weighting:	100%
Assessment Title	Portfolio
Type:	Individual submission
Hand-In Deadline:	Friday 17th January 2025. See submission link on Moodle.
Hand-In Format and Mechanism:	Jupyer notebook `.ipynb` file submitted to the 1CWK100 submission point on the module's Moodle area.

Learning outcomes being assessed:

LO1	Explain the meaning of mathematical concepts involved in cryptography, calculate correctly with them and solve problems about them requiring rigorous mathematical reasoning.
LO2	Describe correctly the various cipher systems from the module and solve problems based on them in a suitable programming environment.

Note: it is your responsibility to make sure that your work is complete and available for marking by the deadline. Make sure that you have followed the submission instructions carefully, and your work is submitted in the correct format, using the correct hand-in mechanism (e.g., Moodle upload). If submitting via Moodle, you are advised to check your work after upload, to make sure it has uploaded properly. If submitting via OneDrive, ensure that your tutors have access to the work. Do not alter your work after the deadline. You should make at least one full backup copy of your work.

Penalties for late submission

The timeliness of submissions is strictly monitored and enforced.

All coursework has a late submission window of 7 calendar days, but any work submitted within the late window will be capped at 40%, unless you have an agreed extension. Work submitted after the 7-day late window will be capped at zero unless you have an agreed extension. See 'Assessment Mitigation' below for further information on extensions.

Please note that individual tutors are unable to grant extensions to assessments.

Assessment Mitigation

If there is a valid reason why you are unable to submit your assessment by the deadline you may apply for assessment mitigation. There are two types of mitigation you can apply for via the module area on Moodle (in the 'Assessments' block on the right-hand side of the page):

- **Non-evidenced extension**: does not require you to submit evidence. It allows you to add a short extension to a deadline. This is not available for event-based assessments such as in-class tests, presentations, interviews, etc. You can apply for this extension during the assessment weeks, and the request must be made before the submission deadline. For this assessment, non-evidenced extension is **2 days**.
- **Evidenced extensions:** requires you to provide independent evidence of a situation which has impacted you. Allows you to apply for a longer extension and is available for event-based assessment such as in-class test, presentations, interviews, etc. For event-based assessments, the normal outcome is that the assessment will be deferred to the summer reassessment period.

Further information about Assessment Mitigation is available on the dedicated Assessments page: https://www.mmu.ac.uk/student-life/course/assessments#ai-69991-0

Plagiarism

Plagiarism is the unacknowledged representation of another person's work, or use of their ideas, as one's own. Manchester Metropolitan University takes care to detect plagiarism, employs plagiarism detection software, and imposes severe penalties, as outlined in the Student Code of Conduct and Academic Misconduct Policy. Poor referencing or submitting the wrong assignment may still be treated as plagiarism. If in doubt, seek advice from your tutor.

As part of a plagiarism check, you may be asked to attend a meeting with the Module Leader, or another member of the module delivery team, where you will be asked to explain your work (e.g. explain the code in a programming assignment). If you are called to one of these meetings, it is very important that you attend.

Use of generative AI

The use of generative AI is permitted in this assessment, so long as it is used in accordance with the instructions provided in the 'Are you allowed to use AI in assessments?' section of the AI Literacy Rise Study Pack. All submitted work must be your own original content.

This is the default Man Met University position, but please make sure you follow these specific instructions:

- You should familiarise yourself with the contents of the AI Literacy Rise Study Pack.

- In particular take note of the following point from the Can generative AI be used for assessments? page there:
  - *The short answer is to use it as a tool to help you learn, not as a tool to do your assessments. We want you to feel confident using generative AI tools to help you learn. But when it comes to demonstrating that learning in your assessments, we want it to come directly from you. You are the author or creator of the assessment, so it should be your voice, your decisions, and ultimately, your work.*

For any other uses of generative AI, you should also follow the instructions in the 'Are you allowed to use AI in assessments?' section of the AI Literacy Rise Study Pack or speak to your tutor. All submitted work must be your own original content.

If you are unable to upload your work to Moodle

If you have problems submitting your work through Moodle, you can send your work to the Assessment Management Team using the Contingency Submission Form. Assessment Management will then forward your work to the appropriate person for marking. If you use this submission method, your work must be sent **before the published deadline**, or it will be logged as a late submission. Alternatively, you can save your work into a single zip folder then upload the zip folder to your university OneDrive and submit a Word document to Moodle which includes a link to the folder. **It is your responsibility to make sure you share the OneDrive folder with the Module Leader, or it will not be possible to mark your work.**

Assessment Regulations

For further information see Assessment Regulations for Undergraduate Programmes of Study on the Assessments and Results information pages.

Feedback and support

Formative Feedback and support:	*Available from the module teaching team in the weekly computer labs.* Contact us by email: **k.m.obrien@mmu.ac.uk** or on teams: **Microsoft Teams chat** **safiullah.khan@mmu.ac.uk** or on teams: **Microsoft Teams chat**
Summative Feedback:	*Marking and feedback comments on your work provided through Moodle.*

Marking scheme

The four courseworks you have been asked to submit will be weighted equally towards your overall mark for this module.

- Number Theory questions (25%)
- DES questions (25%)

- AES questions (25%)
- Public key cryptosystems questions (25%)

For each individual question there are up to 10 marks awarded to the programming/technical elements and 10 marks awarded for the explanatory elements. The correctness of the technical programming and mathematical elements will be weighted to contribute 80% of the marks, while the accompanying explanatory elements will weighted to contribute 20% of the marks.

The tables below shows some marking descriptors for the code/calculation and explanation elements.

Code/calculations

Mark range	Descriptor
8 - 10	Very clear explanation that concisely and precisely explains the code/calculations and summarises how they implement the particular system.
6 - 7	Clear explanations of the code/calculations with some minor inaccuracies or omissions.
4 - 5	Some good explanations of the code/calculations but with significant innaccuracies or omissions.
0 - 3	Poor explanations of the code/calculations without a demonstration of understanding.

Explanations

Mark range	Descriptor
8 - 10	Accurate code/calculations that provide the correct output/answers.
6 - 7	Mostly accurate code/calculations that provide the correct output/asnwers with some minor inaccuracies or omissions.
4 - 5	Some good code/calculations that provide the some of the correct output/asnwers but with significant inaccuracies or omissions.
0 - 3	Code/calculations with incorrect output/answers.

Preparing your submission

You will be submitting a single Jupyter notebook `.ipynb` file to the 1CWK100 submission point on the module's Moodle page. If you have prepared separate notebook files for different questions up to now then you should assemble these into a single notebook file for submission. Cells can be copy-and-pasted across from one notebook file to another. When pasting into the destination notebook make sure that you have selected the cell before which you want your pasted content to appear.

Please answer the questions in order down your notebook file and include clear headings to indicate the different questions.

Number Theory questions

Introduction and question parameters

The following questions require inputs derived from your student ID number. Let n denote your 8-digit Man Met student ID number. Now let a and b denote the 4-digit integers formed by, respectively, the beginning and ending, four digits halves of n.

For example, if $n = 21395495$ then it would split as $2139 \cdot 5495$ and we would have $a = 2139$ and $b = 5495$. But when working on these questions make sure that you are using **your correct** values of n, a and b, derived from your student id number n.

- NT.01

Demonstrate your understanding of the Euclidean Algorithm by performing the sequence of integer divisions with remainder that is necessary to calculate $\gcd(a, b)$ using the Euclidean algorithm.

These calculations should be done using a version of the function `mygcd`, written in lab session 01, that prints out the sequence of integer divisions. Include the code and associated ouput in your submission and provide a text cell that gives a brief summary, in your own words, of how the algorithm is working and why the number produced is the required $\gcd$ value.

- NT.02

Use a code cell and an appropriate command from the `Sympy` library to determine the prime factorizations of a and b. Use a text cell to briefly explain why the value of $\gcd(a, b)$ you calculated in **NT.01** can be *seen* in the prime factorizations of a and b.

- NT.03

Demonstrate your understading of the Extended Euclidean Algorithm by determining the values of integer coefficients x and y that satisfy

$$\gcd(a, b) = xa + yb.$$

This should be done using a version of the `mygcdex` code, written in lab session 01, that implements the extended Euclidean algorithm, and an accompanying text block that demonstrates your understanding by briefly explaining in your own words, the main points of how the code works and why the output it produces are the required coefficients x and y.

- NT.04

Use a text block to explain how to alter the definition of the Python function `mygcdex` so that it produces a different pair of coefficients (x, y) from the pair produced in question NT.03, bu that still satisfies the condition

$$\gcd(a, b) = xa + yb.$$

Then use a code block to write a Python routine that can generate any number of different coefficient pairs (x, y). Demonstrate it by producing 10 distinct such pairs that satisfy

$$\gcd(a, b) = xa + yb.$$

- NT.05

Let p be the prime number

$$p = 113.$$

Use the result of Fermat's Little Theorem to calculate the value of

$$(3^n \bmod p\ ).$$

Use text cell(s) to explain your working and you may use code cell(s) to carry out any necessary calculations. You can make use of the `Mod` command from the `Sympy` library, but you should do so using inputs influenced by the application of Fermat's Little Theorem, (i.e. smaller inputs than the raw `3**n` value).

You can of course make use of any call to `sympy.Mod` to confirm your work.

- NT.06

Use the method of *repeated squaring* to calculate the value of

$$(5^n \bmod p\ ),$$

without any reference to Fermat's Little Theorem or Euler's Theorem.

In your answer you should use code cell(s) to determine the binary representation of n and to determine the necessary sequence of powers for the repeated squaring technique.

Again, confirm your work with the use of any call to `sympy.Mod`, and use a text cell to briefly explain your answer.

Digital Encryption Standard (DES) questions

Introduction

These questions are inspired by question 4.11 from Stallings. The question parts guide you through the application of a single round of the DES encryption algorithm.

The algorithm is fully specified in appendix C of Stallings. There you will find flowcharts and explanations of how the DES algorithm works.

The numbered steps below will take you through the initial preparation of your message m and key k, the calculation of the first sub-key K_1, and then the calculation of L_1 and R_1, the outputs of round 1 of DES. You are asked to maintain m, k and the other intermediate outputs as *binary strings*, i.e. Python strings consisting made up of the characters `1` and `0`.

At the bottom of this notebook you will find Python definitions for lists defining the various transformation functions that form part of DES as described in appendix C. You are encouraged to make use of these, and suitable Python commands to answer the questions.

Guidance

In answering these questions, the message m and key k are represented as 64 bit binary strings, i.e. strings of zeros and ones that are 64 characters long.

The message m you will use is the 64 bit binary string representing the ASCII encoding of your 8-digit university ID number.

The key k you will use is the 64 bit binary string defined as

```
k =
'0011000100110010001100110011010000110101001101100011011100111000'
```

This string is the binary representation of the ASCII encoding of the string `'12345678'` which is calculated as shown in the Python block below. You should use similar commands to generate your initial message string m that you will encrypt.

To help you check your workings and detect any errors the final 64-bit binary string output, which is the concatenation of L_1 and R_1, for your university ID number, can be found in the table at the following link: encrypted ID numbers.

In [5]:
```
k = '12345678'

# the command below redefines k as a list of the binary strings of the ASCII enc
# characters of the initial string k.
# bin(ord(x)) gives the python binary string representing the ASCII encoding of
# the character x. Slicing this from index 2 onwards removes the `0b` binary ind
# from the front of the string. zfill(8) fills in any leading zeros that were st
# from bin(ord(x)) to give a standard 8 bit binary string.

k = [bin(ord(x))[2:].zfill(8) for x in k]

# the join method joins the eight 8-bit binary strings from the list k with the
# concatenates the eight 8-bit binary strings into a single 64-bit binary string
# each of the strings with the empty string ''.
```

```
k = ''.join(x for x in k)
print(k)
```

```
00110001001100100011001100110100001101010011011000110111001111000
```

Warning on array/string indexing

In the description of DES in Appendix C of the Stallings textbook, when referencing the position of a bit in a message block he uses index references beginning at 1, which is appropriate when speaking in plain English. This convention is also used in the various variables defined below that define the various transformations from Appendix C. So the left-most bit in a message block is at index 1, the next bit is at index 2 and so on. However, when referencing bit positions in binary strings in your Python commands you will need to remember that Python references positions in arrays, strings etc using indexing beginning at 0. So the leftmost bit in a binary string is at index 0, the second bit is at index 1, and so on. You will need to be careful when switching between plain language descriptions and Python code to avoid any *off-by-one* errors.

Questions

Prepare a Jupyter notebook and for each question below define the asked for variable. **Make sure you use the exact variable names given in the questions**. For each question you should also prepare a text-cell giving a brief explanation of how you answered the question. The labelling of intermediate variables follows the notation used in figure C.2 of Stallings, pg. 769.

- DES.01 Determine the 64 bit binary string given by the ASCII encoding of your 8-digit Man Met university ID number. Assign this string to the variable `m` in your notebook. This is your message to be encrypted.
- DES.02 Determine the result of applying the Initial Permutation to your message m. Store the resulting 64 bit binary string in the variable `IPm` in your notebook.
- DES.03 Determine the left and right halves that are the inputs to the first round. Store these 32 bit binary strings in the variables `L0` and `R0` respectively.
- DES.04 Determine the Expansion/permutation of R_0 and store the resulting 48 bit binary string in the variable `ER0`.
- DES.05 Determine the first subkey K_1 and store the 48 bit binary string in the variable `K1`.
- DES.06 Determine the XOR operation on the inputs ER_0 and K_1. Store the resulting 48 bit binary string the variable `A`.
- DES.07 Determine the result of applying the S-box substitution/choice operation to your bit string A. Store the resulting 32 bit binary string in the variable `SA`.
- DES.08 Determine the result of applying the permutation function to SA and store the resulting 32 bit binary string the variable `PSA`.
- DES.09 Determine the two 32 bit binary output strings L_1 and R_1 and store these in the variables `L1` and `R1` respectively.

Table data, as in Appendix C of the Stallings textbook

In [6]:
```
# Initial Permutation
IP=[
58, 50, 42, 34, 26, 18, 10, 2,
60, 52, 44, 36, 28, 20, 12, 4,
62, 54, 46, 38, 30, 22, 14, 6,
64, 56, 48, 40, 32, 24, 16, 8,
57, 49, 41, 33, 25, 17, 9, 1,
59, 51, 43, 35, 27, 19, 11, 3,
61, 53, 45, 37, 29, 21, 13, 5,
63, 55, 47, 39, 31, 23, 15, 7]
```

In [7]:
```
# Expansion permutation
E = [32,1,2,3,4,5,4,5,6,7,8,9,8,9,10,11,12,13,12,13,14,15,16,17,
16,17,18,19,20,21,20,21,22,23,24,25,24,25,26,27,28,29,28,29,30,31,32,1]
```

In [8]:
```
#S-boxes, as a list of lists
SBOX = [
# Box-1
[
[14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7],
[0,15,7,4,14,2,13,1,10,6,12,11,9,5,3,8],
[4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0],
[15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13]
],
# Box-2

[
[15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10],
[3,13,4,7,15,2,8,14,12,0,1,10,6,9,11,5],
[0,14,7,11,10,4,13,1,5,8,12,6,9,3,2,15],
[13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9]
],

# Box-3

[
[10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8],
[13,7,0,9,3,4,6,10,2,8,5,14,12,11,15,1],
[13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7],
[1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12]

],

# Box-4
[
[7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15],
[13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9],
[10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4],
[3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14]
],

# Box-5
[
[2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9],
[14,11,2,12,4,7,13,1,5,0,15,10,3,9,8,6],
[4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14],
[11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3]
],
```

```
# Box-6

[
[12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11],
[10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8],
[9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6],
[4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13]

],
# Box-7
[
[4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1],
[13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6],
[1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2],
[6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12]
],
# Box-8

[
[13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7],
[1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2],
[7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8],
[2,1,14,7,4,10,8,13,15,12,9,0,3,5,6,11]
]

]
```

In [9]:
```
# Permutation P
P = [16,7,20,21,29,12,28,17,1,15,23,26,5,18,31,10,
2,8,24,14,32,27,3,9,19,13,30,6,22,11,4,25]
```

In [10]:
```
# Permuted choice 1, used in key generation
PC1 = [57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,
52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,30,22,14,6,61,53,45,37,29,21,13,5,28
```

In [11]:
```
# Permuted choice 2, used in key generation
PC2 = [14,17,11,24,1,5,3,28,15,6,21,10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,
31,37,47,55,30,40,51,45,33,48,44,49,39,56,34,53,46,42,50,36,29,32]
```

In [12]:
```
# Python function for carrying out the xor of two binary strings

def XOR(a,b):
    # a and b should be binary strings of equal length
    c = ''.join(str((int(x) + int(y))%2) for x,y in zip(a,b))
    return c

#example usage
XOR('1100','0101')
```

Out[12]:
```
'1001'
```

Advanced Encryption Standard (AES) questions

Introduction

The finite field $\mathrm{GF}\left(2^8\right)$

The Advanceed Encryption Standard (AES) makes use of the field $\mathrm{GF}\left(2^8\right)$, consisting of polynomials of degree less than or equal to 7, with binary coefficients and polynomial operations carried out modulo the irreducible polynomial

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

The integers y in the range $0 \leq y \leq 255 = 2^8 - 1$ can be represented by the elements of $\mathrm{GF}\left(2^8\right)$ by considering the the 8 binary digits of the binary representation of y to be the coefficients of the polynomial. For example, the integer $y = 201$ has the representation `11001001` as a binary string. Remember to fill with leading zeroes if necessary so the resulting string has eight characters. This string can be obtained from the output of the Python command `bin(201)`. So the polynomial corresponding to $y = 201$ is given by

$$y = 201 \leftrightarrow x^7 + x^6 + x^3 + 1,$$

i.e. the digits in the binary expansion of 201, reading left to right, give the coefficients of x^j, for $j = 7, 6, \ldots 2, 1, 0$.

Question parameters

The following questions require inputs derived from your student ID number. Let n denote your 8-digit Man Met student ID number. Now let a and b denote the 4-digit integers formed by, respectively, the left-hand and right-hand halves of n.

For example, if $n = 21395495$ then we would have $a = 2139$ and $b = 5495$. But when working on these questions make sure that you are using your correct values of n, a and b .

Questions

Let $A = (a \mod 256)$ and $B = (b \mod 256)$. Then let $P_A(x)$ and $P_B(x)$ be the polynomials corresponding to A and B respectively, as described in the introduction.

- AES.01 Carry out the sequence of polynomial divisions with remainder of the Euclidean algorithm for polynomials, to confirm that $\gcd(P_A(x), m(x))$ is indeed 1.
- AES.02 Determine the product polynomial $P_A(x) \cdot P_B(x)$, where the product is carried out with the multiplication operation of $\mathrm{GF}(2^8)$.
- AES.03 Determine the polynomial $Q(x) = \left(P_A(x)\right)^{-1}$, i.e. $Q(x)$ is the multiplicative inverse of $P_A(x)$ in $\mathrm{GF}\left(2^8\right)$. The polynomial $Q(x)$ can be found from the extended Euclidean algorithm.

It is not expected that you carry out these calulculations using Python code. Instead, write up your solutions of these problems using Markdown cells in the Jupyter notebook. There is no need to typeset lots of integer arithmetic.

Public-key cryptosystems (PKC) questions

Introduction

Question parameters

The following questions require inputs derived from your student ID number. Let n denote your 8-digit Man Met student ID number. Now let a and b denote the 4-digit integers formed by, respectively, the left-hand and right-hand halves of n.

For example, if $n = 21395495$ then we would have $a = 2139$ and $b = 5495$. But when working on these questions make sure that you are using your correct values of n, a and b .

Sympy functions

Useful functions from the Sympy library for working on these problems are given below. For guidance and examples on using these functions, either quickly view the functions docstring within the Jupyter by executing `sympy.nextprime?` for example, or consult the Sympy documentation or other resources from sympy.org.

- `sympy.nextprime()`
- `sp.is_primitive_root()`

Questions

RSA Questions 1 — 4 are about the RSA cryptosystem.

- PKC.01 Let p be the smallest prime satisfying $p \geq n$ and let q be the smallest prime satisfying $q \geq (n + 10^6)$. Let m denote the product $m = p \cdot q$. Determine the values `p,q,m` with the help of Sympy's `nextprime` function.
- PKC.02 Your public key is the pair $(101, m)$. Determine your corresponding private key pair (d, m). Store the relevant value in the variable `d` .
- PKC.03 Use your public key to encrypt the message represented by your number b. Store the encrypted value in the variable `M` .
- PKC.04 Finally, decrypt the encrypted message M with the use of your private key and confirm that the plaintext message b is recovered.

For each of the parts PKC.01 — 04 include a text cell with a brief explanation of how your code implements the RSA specification.

Diffie-Hellman key exchange Questions 5 — 7 concern the Diffie-Hellman key exchange protocol.

- PKC.05 Let p be defined as in Q1, i.e. the smallest prime satisfying $p \geq n$. Determine the smallest positive integer α such that α is a primitive root modulo p. Store this in

a variable `alpha`. Your code should not use the Sympy function `sympy.primitive_root()` to find the smallest primitive root. Rather you should loop through the candidates and use the function `sympy.is_primitive_root()` to test each one to find the smallest root. Yu can then use `sympy.primitive_root()` to confirm the correctness of your answer.

- PKC.06 Alice and Bob agree to a Diffie-Hellman key exchange using the prime p and primitive root α. Alice's private key is $X_A = a$ and Bob's private key is $X_B = b$. Determine the public keys Y_A and Y_B exchanged by Alice and Bob. Store these in variables `YA` and `YB`.
- PKC.07 Determine the shared secret key K that Alice and Bob can compute. Store this in a variable `K`.

For each of the parts PKC.05 — 07 include a text cell with a brief explanation of how your code implements the Diffie-Hellman key exchange protocol specification.

Miller-Rabin primality test This question is based on the Miller-Rabin probabilistic primality test. You will need to read about this test in section 2.6 of chapter 2 of the Stallings textbook.

- PKC.08 Let p be defined as in Q1, i.e. the smallest prime satisfying $p \geq n$. Confirm that p passes the Miller-Rabin test by carrying out the necessary computations to confirm the expected value(s) modulo p of the exponential(s) with base a. Include a text cell with an explanation of how your code implements the Miller-Rabin technique outlined in the module.

In [ ]: