# F20CN/F21CN Computer Network Security
# Coursework 2 (CW2)

Due Friday, 22nd November (week 11) at 23:39 local time

## 1 Overview

The learning objective of this coursework is for you to become familiar with the concepts in asymmetric cryptography, and firewall rules maintenance.

This is a **group coursework** to be completed by groups of two students (see Canvas for instructions on forming groups). You are expected to work only **within your group** to complete the coursework tasks. Your coursework submissions will be automatically checked for plagiarism and any other type of academic misconduct. Here are some further points to take into consideration:

- Coursework reports must be written in your own words and any code in your coursework must be your own code. If some text or code in the coursework has been taken from other sources, these sources must be properly referenced.
- Failure to reference work that has been obtained from other sources or to copy the words and/or code of another student is plagiarism and if detected, this will be reported to the School's Discipline Committee. If a student is found guilty of plagiarism, the penalty could involve voiding the course or worse.
- Students must never give hard or soft entire or partial copies of their coursework reports or code to another student outside of their group. Students must always refuse any request from another student for a copy of their report and/or code.
- Sharing a coursework report and/or code with another student outside of your group is collusion, and if detected, this will be reported to the School's Discipline Committee. If found guilty of collusion, the penalty could involve voiding the course or worse.
- And remember: the consequences of taking unacceptable shortcuts in coursework are much worse than getting a bad mark (or even no marks) on a piece of coursework. There has been a recent case where a student was awarded an Ordinary BSc degree (rather than an Honours degree) because of the sanction imposed by the University's Discipline Committee. The offence was plagiarism of coursework.
- The course does not tolerate any form of use of generative AI (e.g., ChatGPT, Copilot). Using any of those tools will result in being reported for academic misconduct and penalties could involve voiding the course or worse.

- Further information on academic misconduct can be found in: https://heriotwatt.sharepoint. com/sites/academicintegrity/SitePages/Academic-Misconduct.aspx

Your submission will include a **group submission** including **group report** and **two code files**, as well as a short **individual report** (one for F20CN students, two for F21CN students). At the end of each task, information that you need to provide for a task is indicated, and the final section of this coursework descriptor describes the submission. For further information on the coursework (including the marking rubric) and groups, please visit our Canvas page for the course.

# 2 Coursework Tasks

## Task 1: Alternative Method of Public-Key Encryption

Consider the following public-key encryption method that was proposed by a recent PhD student. To create a public key, Bob does the following:

1. He selects a random set of positive integers $e = (e_1, e_2, ..., e_n)$ with the property that $e_i > \sum_{j=1}^{i-1} e_j$ for $1 \le i \le n$. Thus, each new element of the set is greater than the sum of the previous elements in the set.

2. He selects a prime number q such that $q > 2e_n$, and random $w$ such that gcd($w$,$q$) = 1, and computes $h = (h_1, h_2, ..., h_n)$ where $h_i = w.e_i \bmod q$, for $1 \le i \le n$.

In this case, $h$ is the public key, and ($e$,$q$,$w$) are the private key. Thus, it should be difficult for an attacker to recover $e$ from $h$[1]. The inventor of this method referred to $e$ as the *easy key* and h as the *hard key*.

To **encrypt** an n-bit message $m = (m_1, m_2, ..., m_n)$ for Bob, **Alice** would compute $c = h_1m_1 + h_2m_2 + \cdots + h_nm_n$. To **decrypt** a ciphertext c, notice that with knowledge of $w^{-1} \bmod q$, which Bob can compute from his private key, **Bob** can compute $c' = cw^{-1} \bmod q$ so that in fact, $c' = e_1m_1 + e_2m_2 + \cdots + e_nm_n$. Notice that the property in the first step of the public key generation above (that each $e_i$ is greater than the sum of all previous values) allows Bob to recover each bit of the plaintext one at a time. To do this from $c'$, start from $e_n$ and check if the existing sum (starting with $c'$) is larger than **or equal to** $e_n$. If it is, then $m_n = 1$ and you compute $c' = c' - e_n$, but if it's not then $m_n = 0$). Then do the same for $e_{n-1}$, and so on, all the way to $e_1$.

**Your task** is to write a Python program that allows a user to compute a public/private key pair and perform the encryption and decryption of given plaintexts or ciphertexts. These operations should follow the steps above, including generating random integers. You can use standard math libraries for computations such as gcd

[1] Note that this has since been shown to be computationally feasible to attack, and hence this method is not secure.

and random number generation (reference this appropriately with a comment in your code). All other code must be written independently by your group.

**In your group report, include**

Pseudo-code A short description of how you approached writing your program, including pseudo-code (https://en.wikipedia.org/wiki/Pseudocode) that describes your solution. This part should not exceed one page in length.

Python Code Include your Python code in an Appendix in your report, and also attach as a separate file with your report submission (your submitted file should be named with the task number, followed by your Canvas group number, e.g., task1-g90.py (replacing g90 ("group 90") with your own Canvas group number). Your code should be clear, using appropriate variable names, and suitably commented, and flexible, using variables and inputted values where appropriate as opposed to hard-coded values.

Testing You should show the results of testing your code on two example key pairs, with one plaintext of 500 characters or more (you can use the same plaintext with both key pairs). To do this, find your own plaintext example of English text and start it with the phrase "To X:", where X replaced with your name. In your report include a short description of your testing results that show the key values as well as some example plaintext and ciphertext characters (no more than one page), and in an appendix include your plaintext, ciphertext, and a screenshot of your key generation, encryption and decryption.

## Task 2: Firewall Rules Application

A key ingredient in a firewall is how it manages access control through rules that describe who is allowed through the firewall, where "who" is typically based on network identifiers such as IP addresses and application ports. Firewall rules are ordered so that a firewall that receives a network packet will make a decision about *allowing* or *denying* a packet based on the first rule that it encounters that is a match for the packet.

**Your task** is to write a Python program that manages firewall rules for *denying* access to IPv4 packets through the firewall based on the packet's direction of travel and its IPv4 address. It should provide support for the following parameters that are to be *entered at the command line*:

- add [rule] [-in|-out] addr

  This adds a new rule to the list of firewall rules. rule is a positive integer starting from 1, where lower numbers are higher in the list of rules (and thus given higher priority). If a rule of the specified number already exists, the new rule is inserted *above* the existing rule with a higher priority (and all following rules get their numbers incremented by 1). If no rule number is provided, then it is assumed to be rule 1. -in and -out specify whether the rule applies either to incoming or outgoing traffic direction (only one should be used at a time); if no direction is provided, then the new rule applies to both directions. addr is the IPv4 address in

dotted decimal notation (e.g., 10.0.0.1) which might also be provided as an address range (e.g., 10.0.0.1-10.0.0.128). For the command to be valid it must contain an address or address range. For your program you can just consider addresses in the range 10.0.0.0-10.0.0.255, e.g., 10.1.2.3 and 10.0.1.1-10.0.10.10.

- remove rule [-in|-out]
  This removes an existing rule from the list of firewall rules, if such a rule exists, else an error should be returned. A rule number must be specified. If an existing rule was specified to apply to both incoming and outgoing traffic, this command may be used to remove one of the directions from the existing rule. If no direction is specified, then the whole rule is removed.
- list [rule] [-in|-out] [addr]
  This lists the existing firewall rules. If no rule is specified, then all rules matching the subsequent parameters are displayed. -in|-out can be used to list only incoming or outgoing packets; if not specified then both will be listed. addr will list the rules matching the provided address or address range; if not specified, then all rules matching the other parameters will be displayed, regardless of the IP addresses used in them.

**In your group report, include**

Pseudo-code A short description of how you approached writing your program, including pseudo-code (https://en.wikipedia.org/wiki/Pseudocode) that describes your solution. This part should not exceed one page in length.

Python Code Include your Python code in an Appendix your report, and also attach as a separate file with your report submission (your submitted file should be named with the task number, followed by your Canvas group number, e.g., task2-g90.py (replacing g90 ("group 90") with your own Canvas group number). Your code should be clear, using appropriate variable names, and suitably commented, and flexible, using variables and inputted values where appropriate as opposed to hard-coded values. Your command line parameters should be validated, with appropriate errors returned.

Testing You should show the results of testing your code with appropriate example parameters that demonstrated the variety of options for entering commands. Include a short description of your results, including screen shots showing the behaviour of your program for different inputs (no more than 2 pages).

**Each F21CN student must also submit an individual report as follows:**

Rule conflicts Suppose that in addition to *denying* packet access through the firewall, your application also included the ability to add rules to explicitly *allow* certain packets through the firewall. For example, suppose that there is a rule to deny incoming packets from the address range

10.0.0.10-10.0.0.12 and a rule to allow incoming packets with address 10.0.0.11. First, describe how the firewall would behave, depending on the rule number associated with each of the two rules. Then, describe what feedback/warning your application could provide to a user when adding one of the rules (where the other already exists) regarding the potential conflict between the two rules. Include some pseudo-code to show what updates you might make in this case.

# 3 Your Submission

**Once you complete a "Declaration of Authorship"**, there are multiple submission links for your coursework: two for F20CN students and three for F21CN students, as described below.

## A Group Submission

There is one Canvas submission link for each group of students to submit the three files described below. Your **group submission** on Canvas should consist of three files **submitted as separate files, NOT a zip file**:

1. Your group report, consisting of
   (a) A title page with the course code and name, the coursework number, the number of your group (from Canvas), and the names of the group members.
   (b) Task 1 pseudo-code (at most 1 page) and testing (at most 1 page).
   (c) Task 2 pseudo-code (at most 1 page) and testing (at most 2 pages).
   (d) Your appendices, which contain information for each task as described above. These must be clearly marked indicating to which part of which task they correspond.
2. Task 1 code as a separate file, named with the task number, followed by your Canvas group number, e.g., task1-g90.py (replacing g90 ("group 90") with your own Canvas group number)
3. Task 2 code as a separate file, named with the task number, followed by your Canvas group number, e.g., task2-g90.py (replacing g90 ("group 90") with your own Canvas group number)

## B Individual Submission (all students): Summary of Contributions

There's one Canvas submission link for each student to submit the report described below.

Each student must also submit an **individual report**, individually written and submitted by each student. This individual report (at most a half page) should describe

the work performed by each group member towards completing the group report and two code files, and **include an estimate of the percentage contribution by each group member**. This report will not be assessed on its own, but may be used to allocate a group submission mark for group members. By default, group members will be assigned the same mark for the group submission. In cases in which there was a significant difference in contributions to the group submission by each member, different marks may be allocated to each member at the determination of the lecturer.

Your report should have a title page (not counted in the page count) with the course number and name, the coursework number (CW2), and your name and matriculation number (as well as your group number).

## C Individual Submission (F21CN students only): Rule Conflicts Discussion

There's one Canvas submission link for each F21CN student to submit the report described below.

Each F21CN student must submit a report (at most 1 page) on the Task 2 rule conflicts discussion. This report must be individually prepared and submitted by each F21CN student.

Your individual report should have a title page (not counted in the page count) with the course number and name, the coursework number (CW2), your name and matriculation number (as well as your group number).