# ECMM462: Fundamentals of Security

**Continuous Assessment (v22.1)**

Diego Marmsoler

due date: Wednesday 23rd November 2022

The CA is worth 40% of your final mark and intended to last for 40 hours. You can get up to 100 marks in total split into four exercises:

| Topic | Marks | Target date |
|---|---|---|
| Symmetric Encryption | 25 | October 19 |
| Asymmetric Encryption | 25 | November 02 |
| Protocol Verification | 25 | November 16 |
| Access Control | 25 | November 23 |

**Format of Submission** Please prepare the following for your submission:

- `ECMM462.pdf` which contains the following sections:
  - "Encryption" contains the solution to task 1 of exercise 2
  - "Verification" contains the solution to task 2 of exercise 2
  - "Protocol" contains the solution to task 2 of exercise 3
- `feistel.py` contains the solution to exercise 1
- `protocol.AnB` contains the solution to task 1 of exercise 3
- `fixed.AnB` contains the solution to task 3 of exercise 3
- `blpcheck.py` contains the solution to exercise 4

Then, submit your solutions via the electronic submission system BART (`https://bart.exeter.ac.uk/`) by Wednesday 23rd November 2022 noon GMT.

**Python Libraries** For the exercises requiring you to implement something in Python do not use any advanced Python libraries.

**Referencing, and academic conduct** You are required to cite the work of others used in your solution and include a list of references, and must avoid plagiarism, collusion and any academic misconduct behaviours: `https://vle.exeter.ac.uk/course/view.php?id=1957`

**Questions** If you have any questions regarding the assessment brief please post them to the corresponding topic in the discussion forum: `https://vle.exeter.ac.uk/mod/forum/view.php?id=2446771`. If you don't want to disclose your identity then the forum has the option to post anonymously.

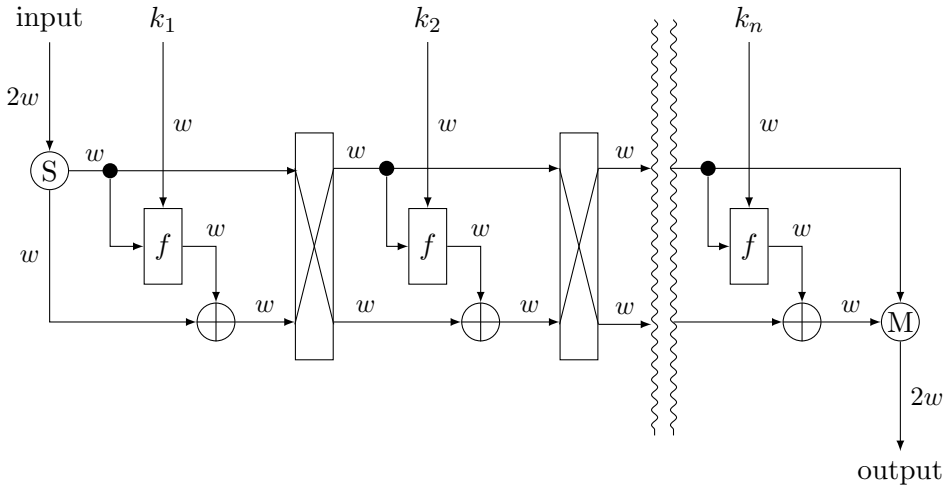



Figure 1: Feistel structure.

# 1 Symmetric Encryption (25 marks)

In the lecture we discussed the Feistel structure as a foundation of many modern symmetric block ciphers. Its general structure is repeated in Figure 1. It requires an $2w$ bit input and splits it into a left (bottom) and right (top) part of $w$ bit each. It then proceeds in $n$ rounds to produce a $2w$ bit output. Each round consists of the following steps:

- The top $w$ bits are combined with the round key $k_i$ by a round function $f$.
- The bottom $w$ bits are combined with the output of $f$ by bit-wise exclusive or.
- Then the top and bottom bits are swapped and passed on to the next round.

For this task you should implement a version of the Feistel structure in Python3. To this end you should complete the template file `feistel.py` provided on the ELE page.

Your implementation should work on 8 bit inputs and use bitwise AND as its round function. Your program should be called `feistel.py` and take the following input parameters:

- The first parameter is an optional option `-d` (for decryption).
- The second parameter is an 8 bit input string.
- The third parameter is the number of rounds.
- The next parameters are the different round keys.

The program should return only the 8 bit result.

For example,

```
>python3 feistel.py 10101010 5 0101 1111 1010 0101 0101
>XXXXXXXX
```

should encrypt 10101010 in 5 rounds with round keys 0101, 1111, 1010, 0101, and 0101 respectively[1].

Marking: You will get a maximum of 15 marks for correctly implementing encryption and a maximum of additional 10 marks for correctly implementing decryption. Partial marks will be awarded if your implementation is partially correct, i.e., if it works correctly for some but not all inputs.

[1]Note that the XXXXXXXX should be the encrypted string in terms of 0 and 1s.

## 2 Asymmetric Encryption (25 marks)

Consider the following scheme by which $B$ encrypts a message for $A$.

1. A chooses two large primes $P$ and $Q$, such that
   - $P$ is relatively prime to $Q-1$,
   - and $Q$ is relatively prime to $P-1$.
2. A publishes $N = PQ$ as its public key.
3. A calculates $P'$ and $Q'$, such that
   - $PP' \equiv 1 \pmod{Q-1}$,
   - and $QQ' \equiv 1 \pmod{P-1}$.
4. B encrypts message $M$ as $C = M^N \bmod N$.
5. A finds $M$ by solving
   - $M \equiv C^{P'} \pmod{Q}$,
   - and $M \equiv C^{Q'} \pmod{P}$.

### 2.1 Task 1 (10 marks)

For this part you should first encrypt the number 5 555 555 and then decrypt the number 5 555 555 using the above scheme.

- The prime numbers you choose should be between 5 000 and 10 000.
- List every intermediate steps for encryption and decryption.

You may use the following tools:

- Chinese Remainder Calculator: `https://www.dcode.fr/chinese-remainder`
- Modular exponentiation: `https://www.dcode.fr/modular-exponentiation`
- Coprime checker: `https://www.dcode.fr/coprimes`

### 2.2 Task 2 (15 marks)

For this part you should verify that the scheme is correct. To this end you should prove that encrypting a message with a public key and decrypting it again with the corresponding private key yields the original message.

For your proof you may use

- Fermat's little theorem
- Chinese remainder theorem

described in the following.

#### 2.2.1 Fermat's little theorem

This theorem states that if $p$ is a prime number, then

$$a^p \equiv a \pmod{p} \tag{1}$$

In addition, if $a$ is not divisible by $p$ Fermat's little theorem guarantees that

$$a^{p-1} \equiv 1 \bmod p) \tag{2}$$

### 2.2.2 Chinese reminder theorem

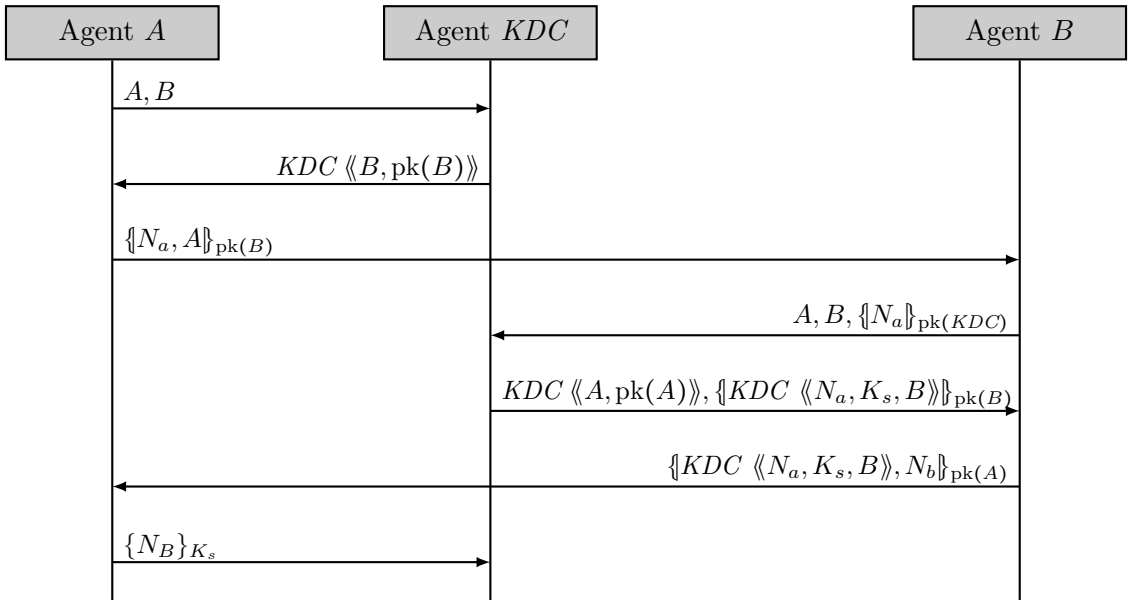
The Chinese reminder theorem states that for a sequence of congruences

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

such that $n_i$ are pairwise co-prime, the system has a unique solution in $\pmod{N}$ where $N = n_1 * \cdots * n_k$.

## 3 Protocol Verification (25 marks)

Consider the following protocol where $X \langle\!\langle M \rangle\!\rangle$ denotes a message $M$ digitally signed by agent $X$:




Note that in this scenario, $A$ does not know the public key of $B$ and $B$ does not know the public key of $A$. The aim of the protocol is to allow Agent $A$ and Agent $B$ to exchange a shared secret key $K_s$.

**Task 1 (10 marks)** Formalize the protocol and the corresponding security property in OFMC.

**Task 2 (5 marks)** The protocol does not meet its goal. Explain why.

**Task 3 (10 marks)** Fix the protocol.

Note that OFMC might take some time to find a possible attack so you should wait until it was at least able to verify 2 sessions before you stop it.

# 4 Access Control (25 marks)

For this exercise you should implement a tool in Python3 to validate properties of a given Bell-LaPadula configuration. To this end you should complete the template file `blpcheck.py` provided on the ELE page.

The system should work for a fixed set of subjects, objects, security levels, and categories:

- Subjects: Alice (A), Bob (B), Charlie (C)
- Objects: Document 1 (1), Document 2 (2), Document 3 (3)
- Security levels: low (l), high (h)
- Categories: A, B, and C

Subject-specific configurations are given in text files `alice.txt`, `bob.txt`, and `charlie.txt`. Each file contains the following information for the corresponding subject:

- The first row contains the access rights for the documents.
- The second row contains the maximum security level and corresponding categories.
- The third row contains the current security level and corresponding categories.

For example, a configuration file in which Alice has

- write rights for Document 1, no rights for Document 2, and append rights for Document 3
- a maximum security level of high for categories A and B
- a current security level of low for category A

looks as follows:

Listing 1: alice.txt

```
w, , a
h:A,B
l:A
```

The currently executed rights are provided as command line parameters. The tool should output whether the model satisfies the Simple security condition, the star property, and/or the discretionary security property.

For example, in the following we check whether a state in which Alice writes to doc1 and Bob reads doc2 is secure:

```
>python3 blpcheck.py Alice:doc1:w Bob:doc2:r
>SSC: yes
>Star: yes
>DS: no
```