2WF90 Algebra for Security

Software Assignment 2:
Polynomial and Finite Field Arithmetic

2022 – 2023

Andreas Hülsing
Responsible Lecturer
a.t.huelsing@tue.nl

Benne de Weger
Lecturer
b.m.m.d.weger@tue.nl

Matthias Meijers
Teaching Assistant
m.c.f.h.p.meijers@tue.nl

October 5, 2023

Table of Contents

1 Introduction

This document describes the second software assignment of the course. In this assignment, you are asked to construct software that can perform certain tasks involving polynomial and finite field arithmetic. Naturally, you are also expected to provide corresponding documentation. The ensuing sections provide all the details necessary to carry out the assignment.

Note: Before you start working on the assignment, make sure to carefully read the "Software Assignments Overview and Guideline" document available on the course site on Canvas. (In addition to, of course, carefully reading this document.)

2 Software

The first deliverable of this assignment is a piece of software that conforms to certain restrictions (see Section 2.5) and is capable of reading and deserializing input exercise data from a file, performing the task(s) indicated by this data, and serializing and writing the answer to a file.

2.1 Expectations

For the software deliverable of this assignment, you are expected to submit a Python 3 source code file called `solve.py`. This file must contain a function called `solve_exercise` that takes two `string` parameters: the first parameter takes the path to a file from which you read the input exercise data, and the second parameter takes the path to a file to which you write the output answer data; for clarity, Listing 1 provides the signature of `solve_exercise`. Given these parameters, `solve_exercise` should figure out which exercise it is asked to solve, solve the exercise, and write the answer in the desired format at the expected location. Here, it is important to note that that you are restricted in the use of the available libraries for Python 3 and the required execution time (for a precise description of these restrictions, see Section 2.5); this is to ensure that your software actually constitutes a self-made implementation of polynomial and finite field arithmetic that is of sufficient quality. Failing to conform to these restrictions may result in a significant loss of points.

```
1 def solve_exercise(exercise_location : str, answer_location : str)
```

Listing 1: Signature of the `solve_exercise` function that must be present in `solve.py`

An example `solve.py` file is available on the course page on Canvas. This file exemplifies, among other things, the input and output handling for this assignment. Feel free to use this file as a template or to draw inspiration from.

Notice that you may actually submit multiple files, as long as the set of files you submit contains a file called `solve.py` with a function called `solve_exercise` that satisfies the signature provided in Listing 1. You can use this to modularize the code, which may facilitate the development process.

2.2 Types of Exercises/Tasks

The types of exercises your software should be able to solve are tasks involving polynomial and finite field arithmetic. More specifically, regarding polynomial arithmetic, your software should be capable of performing the following tasks.

- **Addition**
  Given a prime $p \in [2, 509]$ and polynomials $f, g \in \mathbb{Z}_p[X]$ for which $\deg(f), \deg(g) \in$

$[-1, 256]$[1], compute $f + g$.

- **Subtraction**
  Given a prime $p \in [2, 509]$ and polynomials $f, g \in \mathbb{Z}_p[X]$ for which $\deg(f), \deg(g) \in [-1, 256]$, compute $f - g$.
- **Multiplication**
  Given a prime $p \in [2, 509]$ and polynomials $f, g \in \mathbb{Z}_p[X]$ for which $\deg(f), \deg(g) \in [-1, 128]$, compute $f \cdot g$.
- **Long Division**
  Given a prime $p \in [2, 509]$ and polynomials $f, g \in \mathbb{Z}_p[X]$ for which $\deg(f), \deg(g) \in [-1, 256]$ (but not both the zero polynomial), compute q and r such that $f = q \cdot g + r$ and $\deg(r) < \deg(q)$.
- **Extended Euclidean Algorithm**
  Given a prime $p \in [2, 509]$ and polynomials $f, g \in \mathbb{Z}_p[X]$ for which $\deg(f), \deg(g) \in [-1, 256]$ (not both the zero polynomial), compute a, b, and d such that $a \cdot x + b \cdot y = d$ and $d = \gcd(x, y)$ using the extended Euclidean algorithm. Here, d should be monic.
- **Irreducibility Check**
  Given a prime $p \in [2, 13]$ and a polynomial $f \in \mathbb{Z}_p[X]$ for which $\deg(f) \in [1, 5]$, check whether f is irreducible.
- **Irreducible Element Generation**
  Given a prime $p \in [2, 13]$ and an integer $n \in [1, 5]$, generate $f \in \mathbb{Z}_p[X]$ such that $\deg(f) = n$ and f is irreducible.

Concerning finite field arithmetic, your software should be able to perform the following tasks.

- **Addition**
  Given a prime $p \in [2, 509]$, an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which $\deg(h) \in [2, 256]$, and polynomials $f, g \in \mathbb{Z}_p[X]/(h)$, compute $f + g$.
- **Subtraction**
  Given a prime $p \in [2, 509]$, an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which $\deg(h) \in [2, 256]$, and polynomials $f, g \in \mathbb{Z}_p[X]/(h)$, compute $f - g$.
- **Multiplication**
  Given a prime $p \in [2, 509]$, an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which $\deg(h) \in [2, 128]$, and polynomials $f, g \in \mathbb{Z}_p[X]/(h)$, compute $f \cdot g$.
- **Division**
  Given a prime $p \in [2, 509]$, an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which

[1] $\deg(\cdot)$ is a function that, given a polynomial, returns the degree of this polynomial. For example, $\deg(r)$ denotes the degree of polynomial r. Also, a degree of -1 indicates the zero polynomial.

$\deg(h) \in [2, 256]$, and polynomials $f, g \in \mathbb{Z}_p[X]/(h)$, compute f/g.

- **Inversion**
  Given a prime $p \in [2, 509]$, an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which $\deg(h) \in [2, 256]$, and a polynomial $f \in \mathbb{Z}_p[X]/(h)$, compute f^{-1}.
- **Primitivity Check**
  Given a prime $p \in [2, 13]$, an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which $\deg(h) \in [2, 6]$, and a polynomial $f \in \mathbb{Z}_p[X]/(h)$, check whether f is primitive.
- **Primitive Element Generation**
  Given a prime $p \in [2, 13]$ and an irreducible polynomial $h \in \mathbb{Z}_p[X]$ for which $\deg(h) \in [1, 6]$, generate $f \in \mathbb{Z}_p[X]/(h)$ such that f is primitive.

Your software may assume that the values in the exercises come from the above-specified domains. Moreover, your software may assume that elements are provided in canonical form: for an element from $\mathbb{Z}_p[X]$, this means that all of the element's coefficients lie in the integer range $[0, p-1]$; for an element from $\mathbb{Z}_p[X]/(h)$, this means that the element is the unique representative of its congruence class with a degree less than the degree of h and with all of its coefficients lying in the integer range $[0, p-1]$. However, notice that this does not automatically rule out all possibilities for invalid inputs (i.e., inputs for which certain computations are impossible/undefined). For example, in a finite field division task, the provided divisor may still be the zero polynomial.

Finally, the outputs produced by your software are expected to be provided in canonical form. In case your software outputs elements that are congruent to the correct answer but are not in canonical form, no points will be awarded (for the exercises where this happens).

2.3 Input

The format in which the input exercise data is formatted is JavaScript Object Notation (JSON). More precisely, an exercise is described in a single JSON object that *always* contains the following keys[2].

- `"type"`
  The value of this key indicates the type of the exercise, i.e., whether the exercise is an polynomial arithmetic exercise or a finite field arithmetic exercise.
  This key has one of the following values.
  - `"polynomial_arithmetic"` (type: `string`)
    Indicates that the exercise is an integer arithmetic exercise.
  - `"finite_field_arithmetic"` (type: `string`)
    Indicates that the exercise is a modular arithmetic exercise.

[2] All keys are of type `string`.

- `"task"`
  The value of this key indicates the task to be performed for this exercise. The interpretation of the value of this key depends on the type of the exercise (as indicated by the value of the `"type"` key).
  This key has one of the following values.
  - `"addition"` (type: `string`)
    Indicates that the task to be performed is addition. If the value of the `"type"` key is `"polynomial_arithmetic"`, this refers to addition in $\mathbb{Z}_p[X]$ for some prime p; else, if the value of the `"type"` key is `"finite_field_arithmetic"`, this refers to addition in $\mathbb{Z}_p[X]/(h)$ for some prime p and irreducible polynomial $h \in \mathbb{Z}_p[X]$.
  - `"subtraction"` (type: `string`)
    Indicates that the task to be performed is subtraction. If the value of the `"type"` key is `"polynomial_arithmetic"`, this refers to subtraction in $\mathbb{Z}_p[X]$ for some prime p; else, if the value of the `"type"` key is `"finite_field_arithmetic"`, this refers to subtraction in $\mathbb{Z}_p[X]/(h)$ for some prime p and irreducible polynomial $h \in \mathbb{Z}_p[X]$.
  - `"multiplication"` (type: `string`)
    Indicates that the task to be performed is multiplication. If the value of the `"type"` key is `"polynomial_arithmetic"`, this refers to multiplication in $\mathbb{Z}_p[X]$ for some prime p; else, if the value of the `"type"` key is `"finite_field_arithmetic"`, this refers to multiplication in $\mathbb{Z}_p[X]/(h)$ for some prime p and irreducible polynomial $h \in \mathbb{Z}_p[X]$.
  - `"long_division"` (type: `string`)
    Indicates that the task to be performed is long division (in $\mathbb{Z}_p[X]$ for some prime p). Only possible if the value of the `"type"` key is `"polynomial_arithmetic"`.
  - `"extended_euclidean_algorithm"` (type: `string`)
    Indicates that the task to be performed is the extended Euclidean algorithm (in $\mathbb{Z}_p[X]$ for some prime p). Only possible if the value of the `"type"` key is `"polynomial_arithmetic"`.
  - `"irreducibility_check"` (type: `string`)
    Indicates that the task to be performed is checking for irreducibility (in $\mathbb{Z}_p[X]$ for some prime p). Only possible if the value of the `"type"` key is `"polynomial_arithmetic"`.
  - `"irreducible_element_generation"` (type: `string`)
    Indicates that the task to be performed is the generation of an irreducible element of a certain degree (in $\mathbb{Z}_p[X]$ for some prime p). Only possible if the value of the `"type"` key is `"polynomial_arithmetic"`.

  - `"division"` (type: `string`)
    Indicates that the task to be performed is division (in $\mathbb{Z}_p[X]/(h)$ for some prime p and an irreducible $h \in \mathbb{Z}_p[X]$). Only possible if the value of the `"type"` key is `"finite_field_arithmetic"`.
  - `"inversion"` (type: `string`)
    Indicates that the task to be performed is inversion (in $\mathbb{Z}_p[X]/(h)$ for some prime p and an irreducible $h \in \mathbb{Z}_p[X]$). Only possible if the value of the `"type"` key is `"finite_field_arithmetic"`.
  - `"primitivity_check"` (type: `string`)
    Indicates that the task to be performed is checking for primitivity (in $\mathbb{Z}_p[X]/(h)$ for some prime p and an irreducible $h \in \mathbb{Z}_p[X]$). Only possible if the value of the `"type"` key is `"finite_field_arithmetic"`.
  - `"primitive_element_generation"` (type: `string`)
    Indicates that the task to be performed is the generation of an irreducible element of a certain degree (in $\mathbb{Z}_p[X]/(h)$ for some prime p and an irreducible $h \in \mathbb{Z}_p[X]$). Only possible if the value of the `"type"` key is `"finite_field_arithmetic"`.

- `"integer_modulus"`
  The value of this key denotes the prime modulus that defines the coefficient field over which the polynomials of the exercise are defined. That is, if the value of this key is p, then the polynomials of the exercise have coefficients from $\mathbb{Z}_p$ (i.e., the polynomials in the exercise are elements from $\mathbb{Z}_p[X]$).
  This key has a value of type `number`. In case the value of the `"task"` key does not equal `"irreducibility_check"`, `"irreducible_element_generation"`, `"primitivity_check"`, or `"primitive_element_generation"`, the value of this key lies in the integer range $[2, 509]$; else (if the value of the `"task"` key equals `"irreducibility_check"`, `"irreducible_element_generation"`, `"primitivity_check"`, or `"primitive_element_generation"`), the value of this key lies in the integer range $[2, 13]$.

- `"points"`
  The value of this key indicates the number of points this exercise is worth. This key/value pair is only used for grading; you can safely ignore this.

The remainder of the keys present in a JSON object that describes an exercise is dependent on the values of the `"type"` and `"task"` keys. Indeed, this is because different exercises may be defined by different values. The following is a list of all keys that *may* be present in a JSON object that describes an exercise.

- `"degree"`
  The value of this key denotes the degree of the polynomial to be generated. This key is only (but always) present if the `"task"` key has value `"irreducible_element_generation"`.

This key has a value of type `number` that lies in the integer range $[1, 11]$.

- `"f"`
  The value of this key denotes the value of the left-hand side (or only, in case of a task that concerns only a single operand) operand in the task that is to be performed. This key is only (but always) present if the `"task"` key has value `"addition"`, `"subtraction"`, `"multiplication"`, `"long_division"`, `"extended_euclidean_algorithm"`, `"irreducibility_check"`, `"divison"`, `"inversion"`, or `"primitivity_check"`.
  This key has a value of type `array` with entries of type `number`. The entries in the array denote the coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. Each entry's value lies between 0 (including) and the value of the `"integer_modulus"` key (excluding); the last entry of the array is always non-zero. Lastly, the range in which the length of the array lies depends on the values of the `"type"` and `"task"` keys. Namely, if the value of the `"type"` key is `"polynomial_arithmetic"`, the length of the array is either between 1 (including) and 129 (including) or between 1 (including) and 257 (including). The former is the case when the value of the `"task"` key equals `"multiplication"`; the latter is the case when the value of the `"task"` key does not equal `"multiplication"`. If the value of the `"type"` key is `"finite_field_arithmetic"`, the length of the array is between 1 (including) and the length of the value of the `"polynomial_modulus"` key (excluding).

- `"g"`
  The value of this key denotes the value of the right-hand side operand in the task that is to be performed. This key is only (but always) present if the `"task"` key has value `"addition"`, `"subtraction"`, `"multiplication"`, `"long_division"`, `"extended_euclidean_algorithm"`, or `"divison"`.
  This key has a value of type `array` with entries of type `number`. The entries in the array denote the coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. Each entry's value lies between 0 (including) and the value of the `"integer_modulus"` key (excluding); the last entry of the array is always non-zero. Lastly, the range in which the length of the array lies depends on the values of the `"type"` and `"task"` keys. Namely, if the value of the `"type"` key is `"polynomial_arithmetic"`, the length of the array is either between 1 (including) and 129 (including) or between 1 (including) and 257 (including). The former is the case when the value of the `"task"` key equals `"multiplication"`; the latter is the case when the value of the `"task"` key does not equal `"multiplication"`. If the value of the `"type"` key is `"finite_field_arithmetic"`, the length of the array is between 1 (including) and the length of the value of the

"polynomial_modulus" key (excluding).

- "polynomial_modulus"
  The value of this key denotes the value of the irreducible polynomial modulus that, together with the coefficient field defined by the value of the "integer_modulus" key, defines the finite field from which the polynomials of the exercise are elements. That is, if the value of the "integer_modulus" key is p and the value of this key is h, then the polynomials of the exercise are elements from $\mathbb{Z}_p[X]/(h)$. This key is only (but always) present if the "type" key has value "finite_field_arithmetic".
  This key has a value of type `array` with entries of type `number`. The entries in the array denote the coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. Each entry's value lies between 0 (including) and the value of the "integer_modulus" key (excluding); the last entry of the array is always non-zero. Lastly, the range in which the length of the array lies depends on the value of the "task" key. Namely, if the value of the "task" key equals "multiplication", the length of the array is between 1 (including) and 129 (including). Else, if the value of the "task" key equals "primitivity_check" or "primitive_element_generation", the length of the array is between 2 (including) and 7 (including). Otherwise (if the value of the "task" key does not equal "multiplication", "primitivity_check", or "primitive_element_generation"), the length of the array is between 2 (including) and 257 (including).

Exemplifying the above-specified description of exercises, Listing 2, Listing 3, Listing 4, Listing 5, Listing 6, and Listing 7 respectively provide the descriptions of a polynomial arithmetic addition exercise, a polynomial arithmetic long division exercise, a polynomial arithmetic extended Euclidean algorithm exercise, a polynomial arithmetic irreducible element generation exercise, a finite field arithmetic inversion exercise, and a finite field arithmetic primitive element generation exercise. Of course, these examples are intentionally made simple for instructional purposes. Additional examples of input exercise descriptions (and corresponding output answer descriptions), both simple and realistic, can be found on the course site on Canvas.

2.4 Output

As the input exercise description data, the output answer data is (to be) formatted in JSON. An answer is described in a single JSON object that either contains the three keys "answer-a", "answer-b", and "answer-gcd", the two keys "answer-q" and "answer-r", or the single key "answer". The following elaborates on these keys.

- "answer-a"
  The value of this key denotes the first (Bézout's) coefficient of Bézout's identity (i.e., the a in $a \cdot f + b \cdot g = \gcd(f, g)$) for the operand values (i.e., the polynomials

```
{
    "type": "polynomial_arithmetic",
    "task": "addition",
    "integer_modulus": 3,
    "f": [
        0,
        1,
        1,
        2,
        2
    ],
    "g": [
        1,
        0,
        1,
        0,
        2
    ],
    "points": 0.5
}
```

Listing 2: Example Description of Polynomial Arithmetic Addition Exercise.

```
{
    "type": "polynomial_arithmetic",
    "task": "long_division",
    "integer_modulus": 2,
    "f": [
        0,
        1,
        1,
        1
    ],
    "g": [
        0
    ],
    "points": 0.5
}
```

Listing 3: Example Description of Polynomial Arithmetic Long Division Exercise.

```json
{
    "type": "polynomial_arithmetic",
    "task": "extended_euclidean_algorithm",
    "integer_modulus": 7,
    "f": [
        1,
        3,
        5,
        1,
        6,
        6
    ],
    "g": [
        0,
        5
    ],
    "points": 0.5
}
```

Listing 4: Example Description of Polynomial Arithmetic Extended Euclidean Algorithm Exercise.

```json
{
    "type": "polynomial_arithmetic",
    "task": "irreducible_element_generation",
    "integer_modulus": 3,
    "degree": 3,
    "points": 0.5
}
```

Listing 5: Example Description of Polynomial Arithmetic Irreducible Element Generation Exercise.

```
{
    "type": "finite_field_arithmetic",
    "task": "inversion",
    "integer_modulus": 3,
    "f": [
        0,
        2
    ],
    "polynomial_modulus": [
        1,
        2,
        1,
        0,
        1
    ],
    "points": 0.5
}
```

Listing 6: Example Description of Finite Field Arithmetic Inversion Exercise.

```
{
    "type": "finite_field_arithmetic",
    "task": "primitive_element_generation",
    "integer_modulus": 5,
    "polynomial_modulus": [
        4,
        3,
        4,
        2,
        1
    ],
    "points": 0.5
}
```

Listing 7: Example Description of Finite Field Arithmetic Primitive Element Generation Exercise.

denoted by the values of keys `"f"` and `"g"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"task"` key in the description of the corresponding exercise has value `"extended_euclidean_algorithm"`.
This key has a value of type `array` with entries of type `number`. The entries in the array denote the (polynomial) coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. The last entry of the array is always non-zero, except when the array signifies the zero polynomial (in which case the the array equals `[0]`).

- `"answer-b"`
  The value of this key denotes the second (Bézout's) coefficient of Bézout's identity (i.e., the b in $a \cdot f + b \cdot g = \gcd(f, g)$) for the operand values (i.e., the polynomials denoted by the values of keys `"f"` and `"g"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"task"` key in the description of the corresponding exercise has value `"extended_euclidean_algorithm"`.
  This key has a value of type `array` with entries of type `number`. The entries in the array denote the (polynomial) coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. The last entry of the array is always non-zero, except when the array signifies the zero polynomial (in which case the the array equals `[0]`).

- `"answer-gcd"`
  The value of this key denotes the greatest common divisor of the operand values (i.e., the polynomials denoted by the values of keys `"f"` and `"g"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"task"` key in the description of the corresponding exercise has value `"extended_euclidean_algorithm"`.
  This key has a value of type `array` with entries of type `number`. The entries in the array denote the (polynomial) coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. The last entry of the array is always 1 (since we require the greatest common divisor to be monic).

- `"answer-q"`
  The value of this key denotes the quotient obtained from performing long division with the operand values (i.e., the polynomials denoted by the values of keys `"f"` and `"g"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"task"` key in the description of the correspond-

ing exercise has value `"long_division"`.
This key has a value of type `array` with entries of type `number`. The entries in the array denote the coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. The last entry of the array is always non-zero, except when the array signifies the zero polynomial (in which case the the array equals `[0]`).

- `"answer-r"`
  The value of this key denotes the remainder obtained from performing long division with the operand values (i.e., the polynomials denoted by the values of keys `"f"` and `"g"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"task"` key in the description of the corresponding exercise has value `"long_division"`.
  This key has a value of type `array` with entries of type `number`. The entries in the array denote the coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. The last entry of the array is always non-zero, except when the array signifies the zero polynomial (in which case the the array equals `[0]`).

- `"answer"`
  The value of this key denotes the answer to the corresponding exercise.
  If the value of the `"task"` key in the description of the corresponding exercise equals `"irreducibility_check"` or `"primitivity_check"`, this key has a value of type `boolean` (i.e., either `true` or `false`). Here, `true` indicates that the polynomial denoted by the value of key `"f"` in the description of the corresponding exercise is irreducible or primitive, respectively; `false` indicates that the polynomial denoted by the value of key `"f"` in the description of the corresponding exercise is *not* irreducible or primitive, respectively. In case the value of the `"task"` key in the description of the corresponding exercise does not equal `"irreducibility_check"` or `"primitivity_check"`[3], this key has a value of type `array` with entries of type `number`. The entries in the array denote the coefficients in ascending order based on the degree of the corresponding monomial. That is, the first entry of the array denotes the coefficient corresponding to X^0, the second entry of the array denotes the coefficient corresponding to X^1, et cetera. The last entry of the array is always non-zero, except when the array signifies the zero polynomial (in which case the the array equals `[0]`).

As alluded to before, it may occur that an exercise description describes an exercise that

[3]Of course, the ensuing only applies if the value of the `"task"` key also does not equal `"extended_euclidean_algorithm"` or `"long_division"`; this because exercises concerning these tasks require the other, aforementioned answer keys.

amounts to performing a task that is impossible/undefined (e.g., a finite field division where the divisor is the zero polynomial). In such a case, the keys in the answer description should be the same as in a case where the operation to be performed is well-defined; however, all keys should be assigned the special `null` value.

Illustrating the above-specified description of answers in JSON, Listing 8, Listing 9, Listing 10, Listing 11, Listing 12, and Listing 13 respectively provide the descriptions of the (possible) answers to the exercises described in Listing 2, Listing 3, Listing 4, Listing 5, Listing 6, and Listing 7.

```
{
    "answer": [
        1,
        1,
        2,
        2,
        1
    ]
}
```

Listing 8: Description of Answer to Polynomial Arithmetic Addition Exercise Described in Listing 2.

```
{
    "answer-q": null,
    "answer-r": null
}
```

Listing 9: Description of Answer to Polynomial Arithmetic Long Division Exercise Described in Listing 3.

2.5 Remarks and Restrictions

In order to ensure that your software actually comprises a self-made implementation of polynomial and finite field arithmetic, it should satisfy the following requirements.

- Your software may not use any libraries other than the `json` and `random` libraries. From the latter, your software is only allowed to use the `randint` function. If you want to use any other libraries, you can ask for permission to do so via an

```
{
    "answer-a": [
        1
    ],
    "answer-b": [
        5,
        6,
        4,
        3,
        3
    ],
    "answer-gcd": [
        1
    ]
}
```

Listing 10: Description of Answer to Polynomial Arithmetic Extended Euclidean Algorithm Exercise Described in Listing 4.

```
{
    "answer": [
        2,
        1,
        2,
        2
    ]
}
```

Listing 11: Description of (Possible) Answer to Polynomial Arithmetic Irreducible Element Generation Exercise Described in Listing 5.

```
{
    "answer": [
        2,
        1,
        0,
        1
    ]
}
```

Listing 12: Description of Answer to Finite Field Arithmetic Inversion Exercise Described in Listing 6.

```
{
    "answer": [
        1,
        0,
        2
    ]
}
```

Listing 13: Description of (Possible) Answer to Finite Field Arithmetic Primitive Element Generation Exercise Described in Listing 7.

e-mail to `m.c.f.h.p.meijers@tue.nl`. Naturally, any libraries that make the assignment significantly easier will be rejected.

Then, concerning the quality of your software, the ensuing requirement should be met.

- Your software should, for any exercise, complete within 5 seconds. If your software exceeds this limit for an exercise, you will not be granted any points (for that particular exercise).

For further general software requirements that apply to both software assignments, please make sure to carefully read the "Software Assignments Overview and Guideline" document available on the course site on Canvas.

3 Documentation

The second deliverable of this assignments is documentation for the first deliverable, i.e., your piece of software.

3.1 Expectations

You are expected to hand in a single PDF file containing the documentation for your constructed software. This documentation should, at least, contain the following.

- A title page with an appropriate title (e.g., "2WF90 Software Assignment 2"), the group number, as well as the names and student IDs of the group members.
- Table of contents.
- A clear statement specifying the version of Python 3 in which the submitted software is written.
- In case you have been given the permission to use additional libraries (in addition to the explicitly permitted libraries in the relevant assignment description), a clear statement specifying the additional libraries that are used in the submitted software. Moreover, if necessary, provide clear installation and setup instructions for each of these libraries.
- A concise explanation of the purpose of your software; that is, a short problem description.
- A comprehensive explanation of the approach your software takes to serve its purpose/solve the considered problem. This includes a mathematical description of what your software is capable of doing and how it does so.
- An explanation of the limitations of your software.
- Illustrative examples covering all tasks with different integer moduli, polynomial moduli, and polynomial degrees; these examples should convince the reader that your software works as expected (i.e., provide examples of both regular and special/edge cases).
- A description of each group member's contribution to the submitted work.
- Where relevant, proper references to lecture material and/or literature.
- In case any references are used throughout the document, a proper list of references at the end of the document.

3.2 Remarks and Restrictions

For general documentation requirements that apply to both software assignments, please make sure to carefully read the "Software Assignments Overview and Guideline" document available on the course site on Canvas.