Project II
Public Key Cryptography

Yfantidis Dimitrios (AEM: 3938) 2

June 2023

Introduction

This is the report of the second part of the work in the course "Cryptography Foundations".

The report is written in LATEX and compiled by the **MiKTeX** compiler (ver: One MiKTeX Utility 1.7 - MiKTeX 23.5).

The implementations of the exercises were done in **Python (v 3.10.6)**. For each exercise that requires a code implementation, there is a corresponding folder:

proj_2 crypto 3938/code/ex*

Where there is a single python script, **ex*.py,** containing the implementation and possibly some ".txt" files or other resources for the program.

Almost all the mathematical notations that appear in the text are commonly accepted. Exceptions are the following, which are specified:

x mod y : The remainder of the integer division of x by y, while **x** (**mod y**) is assigned to an equivalence class.

N : The set of natural numbers including 0, i.e. [0, 1, 2, ...].

N* : The set of natural numbers without 0, i.e. [1, 2, 3, ...].

P : The set of prime numbers.

Exercise 1 (6.1)

The algorithm (7.2.2) of subsection (7.2.3) was implemented in the function fast_pow_mod m(b: int, e: int, m: int). The Diffie-Hellman protocol requires the computation of the common key, the integer g^{ab} mod m. Thus, b := g, e := a - b, m := m the assignments to the parameters of the above-mentioned function.

For the given values, (g, p, a, b) = (3, 101, 77, 91), we get as a result:

$$k = 3^{7007} \bmod 101 = 66$$

(Implementation: code/ex1/ex1/ex1**.py**)

Exercise 2 (7.3)

The implementation is the same as in the previous exercise. We get as result:

$$5^{77} \bmod 19 = 9$$

(Implementation: code/ex2/ex2/ex2**.py**)

Exercise 3 (8.19)

A n.d.o. is wanted:

$$p_n < 2^{2^n}, \ \forall n \in \mathbb{N}^* \tag{1}$$

Observation (8.2.4) informs us that $N = p_1 - p_2 - \ldots - p_n + 1$ has a prime divisor in the interval (p_n, N) or is itself prime. Let **j** = 1, then we have that:

$$p_{n+1} \le \left(\prod_{i=1}^{n} p_i\right) + 1, \quad \forall n \in \mathbb{N}^* \tag{2}$$

Applying mathematical induction:

For $n = 1$: (1) ⇒ $p_1 < 2^{2^1}$ ⇔ $2 < 4$, holds so (2) is true.

Suppose that (1) holds for all $2 \le k < n$, so:

$$p_k < 2^{2^k} \Rightarrow \prod_{i=1}^{k} p_i < \prod_{i=1}^{k} 2^{2^i} \tag{3}$$

We examine whether (1) is true for $n = k + 1$:

$$(2) \Rightarrow p_{k+1} \le \left(\prod_{i=1}^{k} p_i\right) + 1 \overset{(3)}{\Longrightarrow} p_{k+1} < \left(\prod_{i=1}^{k} 2^{2^i}\right) + 1 \tag{4}$$

From ID: $\prod_{i=1}^{k} 2^{2^i} = 2^2 - 2^4 - 2^8 - \ldots - 2^{2^k} = 2^{2+4+8+\ldots+2^k} = 2^{\sum_{i=1}^{k} 2^i}$ (5)

$$a^m - b^m = (a - b)(a^{m-1} + a^{m-2} b + a b^{m-3 2} + \ldots + ab^{m-2} + b^{m-1})$$

for $a = 2$, $b = 1$, $m = k + 1$:

$$2^{k+1} - 1^{k+1} = (2 - 1)(2^k + 2^{k-1} - 1 + 2^{k-2} - 1^2 + \ldots + 2 - 1^{k-1} + 1^k) \Leftrightarrow$$
$$\Leftrightarrow 2^{k+1} - 1 = 2^k + 2^{k-1} + 2^{k-2} + \ldots + 2 + 1 \Leftrightarrow$$
$$\Leftrightarrow 2^{k+1} - 2 = 2^k + 2^{k-1} + 2^{k-2} + \ldots + 2$$

Ara

$$\sum_{i=1}^{k} 2^i = 2^{k+1} - 2 \tag{6}$$

From relations (5) and (6) :

$$\prod_{i=1}^{k} 2^{2^i} = 2^{2^{k+1}-2} \tag{7}$$

From relations (4) and (7) :

$$p_{k+1} < 2^{2^{k+1}-2} + 1 \tag{8}$$

We solve the equation:

$$2^{2u-2} + 1 < 2^u \Leftrightarrow \frac{2^{2u}}{4} + 1 < 2^{2u}$$
$$\Leftrightarrow 2^{2u} + 4 < 4 - 2^{2u}$$
$$\Leftrightarrow 3 - 2^{2u} > 4$$

The above equation holds for any positive integer u (true for $u = 1$ and $f(u) = 2^u$ ascending). So, if we set u to 2^{k+1} then, we have:

$$2^{2^{k+1}-2} + 1 < 2^{2^{k+1}}, \forall k \in \mathbb{N}^* \tag{9}$$

From relations (8) and (9), it follows that (1) is true for $n = k + 1$:

$$p_{k+1} < 2^{2^{k+1}}$$

Finally, we proved relation (1):

$$p_n < 2^{2^n}, \forall n \in \mathbb{N}^*$$

Exercise 4 (8.32)

Question (i):

$$\gcd(a, b) = 1 \Leftrightarrow \exists\, n_1, m_1 \in \mathbb{Z} : an_1 + bm_1 = 1 \qquad (1)$$

$$d := \gcd(c, b) \Leftrightarrow \exists\, n_2, m_2 \in \mathbb{Z} : cn_2 + bm_2 = d \qquad (2)$$

$$d' := \gcd(ac, b) \Leftrightarrow \exists\, n_3, m_3 \in \mathbb{Z} : acn_3 + bm_3 = d' \qquad (3)$$

It will turn out that $d|d'$ and $d'|d$, so they are equal.

$$(1) \Leftrightarrow an_1 d + bm_1 d = d \overset{(2)}{\Longleftrightarrow} an_1 (cn_2 + bm_2) + bm_1 (cn_2 + bm_2) = d \Leftrightarrow$$

$$\Leftrightarrow acn\, n_{12} + abn\, m_{12} + bcn\, m_{21} + b\, m\, m^2_{12} = d \Leftrightarrow$$

$$\Leftrightarrow n\, n_{12} - ac + (an\, m_{12} + cn\, m_{21} + bm\, m_{12})b = d \qquad (4)$$

So d is a linear combination of $a - c$ and b. We have that d' is the MCD of $a - c$ and b, άρα:

$$\begin{matrix} d'|ac \\ d'|b \end{matrix} \Rightarrow d'|(u - ac + v - b),\ \forall u, v \in \mathbb{Z}. \qquad (5)$$

Therefore, $(4) \wedge (5) \Rightarrow d'|d$. Also d is the MCD of c and b:

$$\begin{matrix} d|c \\ d|b \end{matrix} \Rightarrow d|(u - c + v - b),\ \forall u, v \in \mathbb{Z} \xRightarrow[u=an_3,\ v=m_3]{(3)} d|d'$$

Finally:

$$\begin{matrix} d|d' \\ d'|d \\ d, d' \geq 0 \text{ as MCA} \end{matrix} \Rightarrow d' = d \Leftrightarrow \gcd(ac, b) = \gcd(c, b)$$

Question (ii):

΄Εστω $d := \gcd(a + b, a - b)$, τότε:

$$d \mid [(a + b)n + (a - b)m],\ \forall n, m \in \mathbb{Z} \Rightarrow \begin{matrix} d|2a & \text{for } (n, m) = (1, 1) \\ d|2b & \text{for } (n, m) = (1, -1) \end{matrix}$$

So:

$$d \mid \gcd(2a, 2b) \Leftrightarrow d \mid [2 - \gcd(a, b)] \Leftrightarrow d \mid 2 \Leftrightarrow (d = \pm 1 \vee d = \pm 2)$$

It holds that $d \geq 0$ as a MCD, so $d \in \{1. 2\}$. In particular, if a and b are odd integers then $a + b$ and $a - b$ are even, so $d = 2$.

$$\begin{matrix} a := 2k_1 + 1 \\ b := 2k_2 + 1 \end{matrix} \Rightarrow \begin{matrix} a + b = 2k_1 + 2k_2 + 2 = 2(k_1 + k_2 + 1) \\ a - b = 2k_1 - 2k_2 = 2(k_1 - k_2) \end{matrix} \xRightarrow[c_2 = k_1 - k_2]{c_1 = k_1 + k_2 + 1} \begin{matrix} a + b = 2c_1 \\ a - b = 2c_2 \end{matrix}$$

$$d = \gcd(a + b, a - b) = \gcd(2c_1, 2c_2) = 2 - \gcd(c_1, c_2) \neq 1$$

$$\xRightarrow{d \in \{1, 2\}} \mathrm{d} = 2$$

Question (iii):

The following suggestions apply:

$$\mathbf{a \equiv b \pmod{m} \wedge b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}} \qquad (\Pi.1)$$

$$\mathbf{a \equiv b \pmod{m} \Rightarrow a^n \equiv b^n \pmod{m}} \qquad (\Pi.2)$$

$$\mathbf{a \equiv b \pmod{m} \Rightarrow na \equiv nb \pmod{m}} \qquad (\Pi.3)$$

Initially:

$$\gcd(a, b) = 1 \Rightarrow \exists\, x, y \in Z : 1 = ax + by \qquad (6)$$

Let $d := \gcd(2^a - 1, 2^b - 1)$, then:

$$\begin{matrix} \exists\, g_1 \in Z : 2^a - 1 = g_1 d \\ \exists\, g_2 \in Z : 2^b - 1 = g_2 d \end{matrix} \Leftrightarrow \begin{matrix} 2^a = g_1 d + 1 \\ 2^b = g_2 d + 1 \end{matrix}$$

So:

$$\begin{matrix} 2^a \equiv 1 \pmod{d} \\ 2^b \equiv 1 \pmod{d} \end{matrix} \xRightarrow{(\Pi.2)} \begin{matrix} (2^a)^x \equiv 1^x \pmod{d} \\ (2^b)^y \equiv 1^y \pmod{d} \end{matrix}$$

Or else:

$$(2\ )^{ax} \equiv 1 \pmod{d} \qquad (7)$$

$$(2\ )^{by} \equiv 1 \pmod{d} \qquad (8)$$

From (7) and (P.3):

$$(2\ )^{ax}\,(2\ )^{by} \equiv (2\ )^{by} \pmod{d} \qquad (9)$$

From (8), (9) and (P.1):

$$\mathbf{(2\ )^{ax}\,(2\ )^{by} \equiv 1 \pmod{d}} \qquad (10)$$

Finally, from (6) and (10):

$$2 = 2^{ax+by} = (2\ )^{ax}\,(2\ )^{by} \equiv 1 \pmod{d} \Rightarrow$$
$$\Rightarrow 2 \equiv 1 \pmod{d} \Rightarrow$$
$$\Rightarrow d | 2 - 1 \Rightarrow d | 1 \Rightarrow$$
$$\Rightarrow d = \pm 1 \xRightarrow{d>0} d = 1$$

It is shown that for $\gcd(a, b) = 1$:

$$\mathbf{\gcd(2^a - 1, 2^b - 1) = 1}$$

Question (iv):

The divisors of $p \in P$ are 1 and p, while the divisors of $q \in P$ are 1 and q. Therefore $\mathbf{gcd(p, q) = 1}$ (since $p \neq q$) and hence $\mathbf{gcd(2^p - 1, 2^q - 1) = 1}$, as proved in the previous question.

Exercise 6 (8.45)

The Python script **"ex6.py"** checks if the equation

$$\frac{\sigma(n)}{n} < \frac{e^{\gamma}}{2} \cdot \ln(\ln(n)) + \frac{0.74}{\ln(\ln(n))} \tag{6.1}$$

is true for every n odd (positive) integer with $n < 2^{20}$. The exception is $n = 1$ as the right-hand side of the equation goes out of its scope of definition. Therefore the truth value of the above relation is checked for all

$$n \in \{3, 5, 7, 9, ..., 2^{20} - 3, 2^{20} - 1\}$$

They all rely on the function find counter argument in interval(...), whose mathematical modelling would be

$$f : \mathbb{Z}^2 \to \{False, True\} \times \mathbb{Z}, \text{ with}$$

$$f(a, b) = \begin{cases} (False, -1) & \text{if (6.1) holds for any } a \leq n < b,\ 2 \nmid n \\ (True, n_0) & \text{if } \exists n_0 : \frac{\sigma(n_0)}{n_0} \geq \frac{e^{\gamma}}{2} \ln(\ln(n_0)) + \frac{0.74}{\ln(\ln(n_0))},\ a \leq n_0 < b,\ 2 \nmid n_0 \end{cases}$$

Techniques for optimizing the above function are the a priori computation of the constant $e^{\gamma}/2$ and the storage of the expression $\ln(\ln(n))$ in a variable to avoid 2 additional calls to the function math.log().

Otherwise, these small optimizations are not particularly efficient for $a = 3$ and $b = 2^{20}$. Therefore, hardware-level parallelism was employed using Python's built-in concurrent.futures library.

Instead of calculating $f(3, 2^{20})$ 9 values are chosen $v_1 < v_2 < v < ... < v_9$ so that a partitioning of the interval $[3, 2^{20})$ is performed, i.e. (assuming $v_0 = 3$ and $v_{10} = 2^{20}$):

$$\bigcup_{i=0}^{9} [v_i, v_{i+1}) = [v_0, v_{10}) = [3, 2^{20})$$

$$[v_i, v_{i+1}) \cap [v_j, v_{j+1}) = \emptyset, \forall i \neq j$$

Moreover, assuming that f_1 is the logical variable returned by f and f_2, respectively, the integer, then:

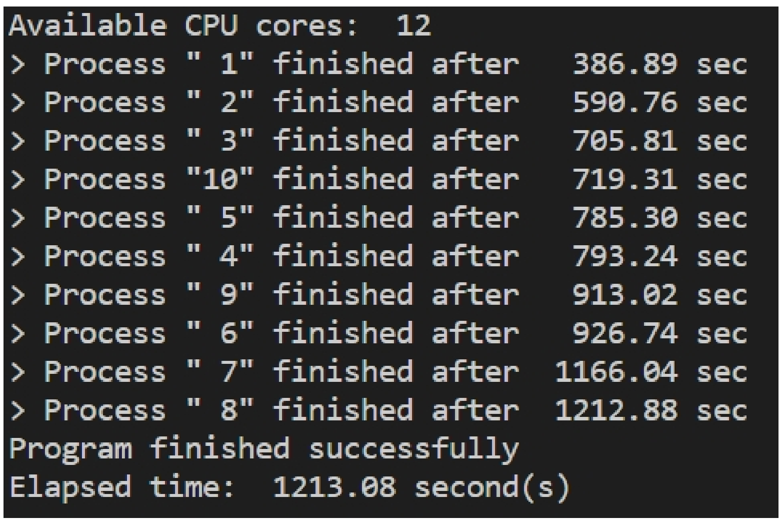
$$\sum_{i=0}^{9} f_1(v_i, v_{i+1}) = f_1(3, 2^{20})$$

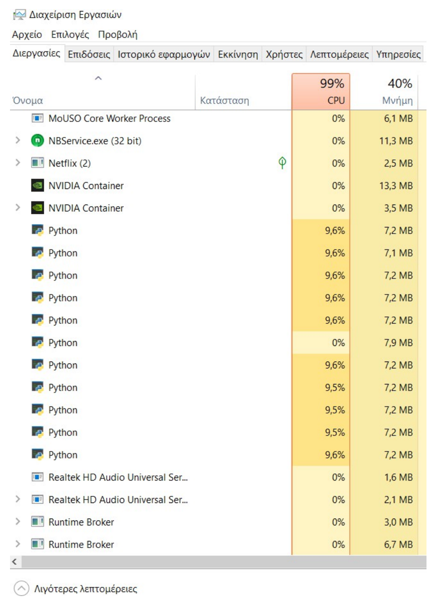
(sum: meaning logical decoupling | f_2 is printed if $f_1 = True$)

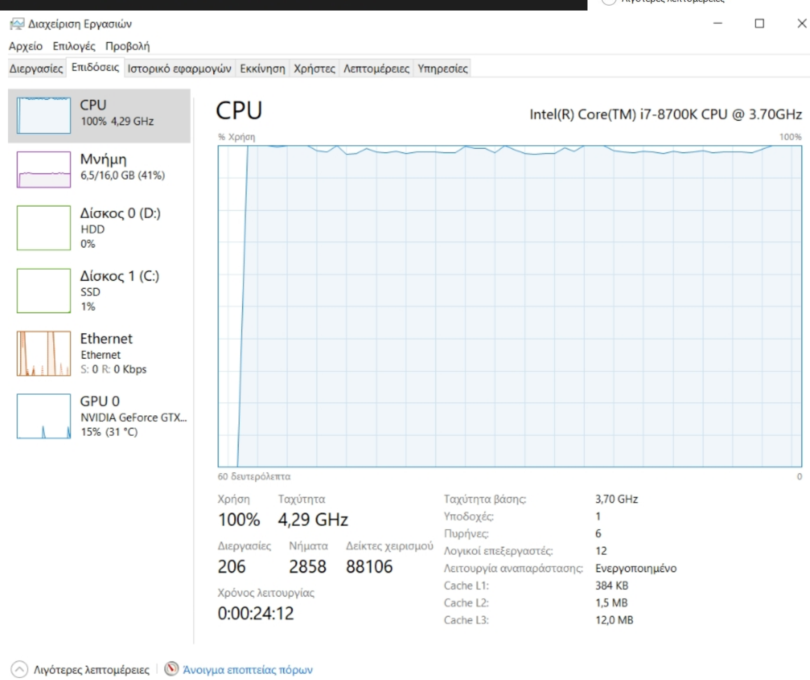
So, we can find out if (6.1) is true for each $v_0 \leq n < v_{10}$ by checking each of the 10 consecutive pairs in parallel by creating a ProcessPoolExecutor with 10 parallel processes, one for each $f(v_i, v_{i+1})$. If even one of these intervals found counterexample then $\sum_{i=0}^{9} f_1(v_i, v_{i+1}) = True$ and the program will terminate with the message "Mathematical formula is invalid" otherwise with "Program finished successfully".

Commentary on Results:

It took the program 20 minutes to decide that there is no odd integer $3 \leq n < 2^{20}$ that violates equation (6.1), which is as long as the slowest procedure. The serial version of the program would take about 2 hours and 16 minutes (approx.
6.76 times longer), as the sum of the execution times of the 10 procedures. Ideally, if the optimal v_i had been chosen as the ends of the intervals, then all procedures would have taken the same time, so the program would have been 10 times faster. The values of v_i were chosen instinctively and no deeper process was performed on the above issue.
Thus, it proved the point of the exercise.







Exercise 7 (9.18)

Let the function $P(n) = (\vec{p}, \vec{\alpha})$, where $\vec{p} \in P^k$ and $\vec{\alpha} \in N^{*k}$, $k \in N^*$ express the factorization of n to prime, i.e:

$$n = \prod_{i=1}^{k} p_i^{\alpha_i}$$

$$p_i \neq p_j \ \forall i \neq j$$

$$p_i \in P, \ \alpha_i \in N^*$$

This is exactly what the function prime_factorization(n:int) (test division algorithm) implements. It quickly yields an answer for the integers 6553130926752006031481761 and 9999109081 as all their prime factors are small.

According to the Korselt criterion, for each of them, the following must hold:

$a_i = 1$ for each $1 \leq i \leq k$

$p_i - 1 \mid n - 1$ for each $1 \leq i \leq k$

The given integers meet the above criteria.

(Implementation: code/ex7/ex7.**py**)

Exercise 8 (9.28)

The integers $835335 - 2^{39014} \pm 1$ pass Fermat's test.

$\mathbf{n_1} = \mathbf{835335 - 2^{39014} + 1}$: Pseudo-first with respect to the base $a_1 \in Z$, $a_1 \approx 2^{39033.011512}$

$\mathbf{n_2} = \mathbf{835335 - 2^{39014} - 1}$: False prime with respect to the base $a_2 \in Z$, $a_2 \approx 2^{39033.632355}$

The a_1 and a_2 are printed in detail in the n1.txt and n2.txt files respectively.

n1 is probable prime with respect 2^39033.011512 (elapsed: 84.548 sec) n2 is probable prime with respect 2^39033.632355 (elapsed: 166.835 sec)

(Implementation: code/ex8/ex8/ex8**.py**)

Exercise 9 (10.1)

The test division algorithm produces the following results:

$$\mathbf{2^{62} - 1} = 3 - 715827883 - 2147483647$$

$$\mathbf{2^{102} - 1} = 3^2 - 7 - 103 - 307 - 2143 - 2857 - 6529 - 11119 - 43691 - 131071$$

(Implementation: code/ex9/ex9/ex9**.py**)

Exercise 10 (10.8)

The program selects 1000 random integers of 100 bits, i.e:

$$n_i \xleftarrow{\$} [2^{99}, 2^{100}) \cap \mathbb{Z},\ i \in \{1, 2, ..., 1000\}$$

The function lehman(int, float) has two input parameters: the number to be generated, **n,** and a time limit in seconds (equal to 10 in this exercise). As a result, it returns a factor **f** of n. The algorithm is successful if

1. terminate before the time limit and in addition
2. $f \neq 1 \lor f \neq n \lor f \in \mathbb{Z}$ (no value None) In total, a

factor was found for 16 integers. Indicatively:

n_{21} = 674,902,139,001,917,536,149,940,578,006 with 1,739,779,667,412,498 | n_{21}

n_{31} = 876,328,564,129,129,523,250,183,808,043,827 with 29,891,192,611,171 | n_{31}

n_{551} = 949,629,530,912,133,951,962,690,604,832 with 49,596,489,117,889,648 | n_{551}

The results are detailed in the **"results.txt"** file.

Produced a factor for 1.600% of integers (16 / 1000)
Elapsed time: 1641.572 sec

(Implementation: code/ex10/ex10/ex10**.py**)

Exercise 11 (10.21)

The implementation done for Pollard-r in this exercise aims to find a factor of the integer N (unique parameter). Initializations apply:

$F(x) = (x^2 + 1) \bmod N$

$X_0 \xleftarrow{\$} \{2, 3, ..., N - 1\}$

$X := X_0$

$Y := X_0$

The algorithm terminates when $1 < d < N$ and d is returned as the result, where
$d = \gcd(|X - Y|, N)$.

For $\mathbf{N = 2^{257} - 1}$ and initializing the seed of the Python generator with $s = 42$, the following results are obtained:

Found non-trivial factor: 535006138814359 Elapsed
time: 70.564 sec
Execution steps: 17571888

(Implementation: code/ex11/ex11/ex11**.py**)

Exercise 12 (11.3)

The following were used to solve the exercise:

1. Test division function: as in the previous exercises.
2. **Euler** function: for $n = p_1^{a_1} \cdot p_2^{a_2} \cdot \ldots \cdot p_k^{a_k}$, returns:

$$\phi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right)$$

   or equivalent, to avoid the use of floating-point numbers:

$$\phi(n) = \Phi_{k+1}(n)$$

$$\Phi_i(n) = \begin{cases} \Phi_{i-1}(n) - \lfloor \frac{\Phi_{i-1}(n)}{p_{i-1}} \rfloor & \text{for } i \geq 2 \\ n & \text{for } i = 1 \end{cases}$$

3. Private Key Computation Algorithm: For input $pk = (e, N)$, computes $sk = (d, N)$ with $e \cdot d \equiv 1 \pmod{\phi(N)}$.
4. **RSA** algorithm: for input the ciphertext and sk, returns the decrypted message.

First, we have $pk = (19, 11413)$ and thus we calculate $sk = (1179, 11413)$:

$$C = (3203, 909, 3143, 5255, 5343, 3203, 909, 9958, 5278, 5343, 9958, 5278, 4674, 909, 9958, 792, 909, 4132, 3143, 9958, 3203, 5343, 792, 3143, 4443)$$

then the call to RSA(sk, C) returns:

$$M = (119, 101, 108, 99, 111, 119, 101, 32, 116, 111, 32, 116, 104, 101, 32, 114, 101, 97, 108, 32, 119, 111, 114, 108, 100)$$

Whose decoding in ASCII characters is:

"welcowe to the real world"

(Implementation: code/ex12/ex12/ex12**.py**)

Exercise 13 (12.2)

For input pk = (e, N) = (50736902528669041, 194749497518847283) the steps of Wiener's attack are given:

1. We store in a list, A, the continuous fraction of $\frac{e}{N}$ with an accuracy of 40 coefficients:

$$\frac{e}{N} = [0; 3, 1, 5, 5, 3, 2, 1, 1, 2, 1, 2, 1, 1, 1, 2, 1, 4, 1, 26, 4,$$
$$2, 3, 1, 18, 10, 6, 3, 180, 2, 2, 1, 1, 4, 2, 5, 1, 2, 3, 83, 9]$$

2. For each i $\in [1, 40]$ we store in a list, F, the real numbers x_i , where $x_i = [A_0 ; A_1 , ..., A_i]$
3. For each i $\in [1, 40]$ we use the class fractions.Fraction to show us the turn the integers N_i and D_i , where $\frac{N_i}{D_i}$ the derivative fraction of x_i . In the replay for i = 12, $x_{12} \approx 0.260523921268139$, so $(N_{12} , D_{12}) = (5440, 20881)$.
4. It holds that $\mathbf{2^{e \cdot D_{12}} \equiv 2 (mod\ N)}$, so $D_{12} = 20881$ is returned as a possible private key.

Entering the ciphertext into a Base64 decoder results in a Python list:

$$C = [47406263192693509, 51065178201172223, ..., 134434295894803806, 57208077766585306]$$

Entering sk = (D_{12} , N) and C in RSA results in a list of integers whose encoding in ASCII characters is:

"Just because you are a character doesn't mean that you have character"

(Implementation: code/ex13/ex13/ex13**.py**)

Exercise 14 (13.2)

For N = 899, e = 839, m = 3, s = 301 ⇒ $\mathbf{a = s^e \bmod N = 675 \neq m}$, so the digital signature s is wrong.

(Implementation: code/ex14/ex14/ex14**.py**)