Number theory in cryptography

– Exercise set 2 –

The exercises T2.1, P2.1 have to be handed in on Tuesday, 5th March 2024, 8:30 at latest. As usual, theoretical exercises have to be uploaded on Moodle, as a PDF file (e.g., a scan of a handwritten version or a PDF obtained from a LaTeX file). Programming exercises (as P2.1 here) have to be completed in the file `HW_2_2024_SAGE.ipynb` to be found in the `CoCalc` project (see P1.1 from last week). We will then automatically collect this file online; you do *not* need to send/share it.

Theoretical questions

T 2.1 Prove the correctness of the `RECURSIVE-DFT` algorithm supposing $n = 2^k$.

T 2.2 Recall the extended Euclidean algorithm from any source that you like. What is the run-time of the algorithm assuming that the two inputs a and b are both ℓ-bit integers? Show all the steps in your run-time analysis.

Programming exercises

P 2.1

a) Write the function `RecursiveDFT` taking as input a vector $a \in \mathbb{Z}^n$ (e.g., the coefficients of a polynomial $p(x)$ of degree n) as well as a parameter ω (e.g., a complex root of unity) and that outputs the vector $\widehat{a} = \mathrm{DFT}_\omega(a)$. Here we can assume that $n = 2^k$ is a power of 2.

b) Write the function `InverseRecursiveDFT`.

c) Write a function `DFT_product` that computes the product of two integers, using fast Fourier transform. Hint: we do not require you to necessarily implement Schönhage-Strassen's algorithm, but you can use the idea that 2 is a primitive $2n$-th root of unity in $\mathbb{Z}/(2^n+1)\mathbb{Z}$. Otherwise working with complex roots of unity is fine.

P 2.2

a) Write a program `timeExtGCD(`ℓ`, N)` that takes as input two integers ℓ and N and does the following: you sample at random N pairs of ℓ-bit integers (a, b) and measure the time it takes to compute the SAGE function `xgcd` on each of those. Your program would then output the average of these times. One way to time a block of code is to use the SAGE function `cputime()` as follows:

```
t0 = cputime()
<YOUR CODE>
t = cputime(t0)
```

The variable t now contains the CPU time (in seconds) needed for the execution of the code. (Hint: you might want to look up in the SAGE reference manual how to generate random integers in the interval $[1, x]$).

b) For different values of ℓ, calculate $t(\ell) = \mathtt{timeExtGCD}(\ell, 100)$. Plot the points $(\ell, t(\ell))$ and compare against the theoretical estimate from T2.2.

P 2.3

a) Write a *recursive* function `Fibo_recursive(n)` (i.e., your function calls itself) that outputs the n-th Fibonacci number (1, 1, 2, 3, 5, 8, 13, ...).

b) Write an *iterative* function `Fibo_iterative(n)` (i.e., using a `for` loop) that outputs the n-th Fibonacci number.

c) Run your two functions on $n = 32$, time and compare the results.