Homework 3 - RSA Cryptography and Elliptic Curves

Cryptography and Security 2021

- You are free to use any programming language you want, although SAGE is recommended.
- Put all your answers **and only your answers** in the provided SCIPER-answers.txt file. This means you need to provide us with all `Q` values specified in the questions below. You can download your **personal** files from the following link:
`http://lasec.epfl.ch:80/courses/cs21/hw3/index.php`
- **Please do not put any comment or strange character or any new line** in the .txt file.
- We also ask you to submit your **source code**. This file can of course be of any readable format and we encourage you to comment your code. Notebook files are allowed, but we prefer if you export your code as a text file with a sage/python script.
- The plaintexts of most of the exercises contain some random words. Don't be offended by them and Google them at your own risk. Note that they might be really strange.
- If you worked with some other people, please list all the names in your answer file.
- We might announce some typos/corrections in this homework on Moodle in the "news" forum. Everybody is subscribed to it and does receive an email as well. If you decided to ignore Moodle emails we recommend that you check the forum regularly.
- The homework is due on Moodle on **Friday the 10th of December** at 23h59.

Exercise 1 Elliptic fun

A cryptographically minded student has noticed that thousands of papers in the cryptographic literature make use of bilinear maps/pairings to construct interesting cryptographic schemes. This student devised the following challenge to test his friends' knowledge of bilinear pairings.

A bilinear map $\hat{e} : G \times G \to G'$ for the purpose of this exercise satisfies the following properties. Note we use multiplicative notation for G.

- $e(g, g) \neq 1_{G'}$ for some $g \in G$.
- It is bilinear, i.e. $e(g^a, g^b) = e(g, g^b)^a = e(g^a, g)^b = e(g, g)^{ab}$.
- $|G| = |G'|$.

You are given the parameters of a pairing-friendly elliptic curve $E : y^2 = x^3 + x$ defined over field $\mathbb{F}_p$ which admits an efficient Tate pairing $e : G \times G \to G'$. In particular, you are given:

- The (prime) characteristic of the field, p, stored in `Q1_p` as an integer.
- The (prime) order of G, r, stored in `Q1_r` as an integer.
- The embedding degree of E, $h = 2$, stored in `Q1_h` as an integer.

The Tate pairing is such that $e(g, g) = 1$, where $g \in G \subset E(\mathbb{F}_{p^2})$, which contradicts the properties we require of a bilinear pairing $\hat{e}$. However, we can define a pairing $\hat{e}$ based on e with an appropriate *distortion map* to get around this.

Let $g, h \in E$ defined over $\mathbb{F}_p$. Then, we set $\hat{e}(g, h) = e(g, \phi(h))$, where we define $\phi(h) = (-x, i \cdot y)$ where $h = (x, y)$ is a point and $i^2 = -1$; this works due to our choice of p which is 3 modulo 4 and our curve which has embedding degree 2.

For your implementation, we suggest that you construct this modified Tate pairing $\hat{e}$ using the `tate_pairing` function provided by SageMath, i.e. by computing `P.tate_pairing(phi_implementation(Q), ...)`. Note that in addition to the above parameters, you are given:

- A generator of G, g, stored as a point $(x, y) =$ (`Q1_g1`, `Q1_g2`) as a pair of integers in $\mathbb{F}_p$.

Let g generate G as above. You are given pairs of the form $(g^{x_i}, g^{z_i})_{i \in I}$ for index set I, stored as two pairs of points (`(Q1_g1xi, Q1_g2xi), (Q1_g1zi, Q1_g2zi)`), where each element is in integer format. We say that (g^{x_i}, g^{z_i}) is a valid Strong Diffie-Hellman pair (or valid SDH pair) if and only if $g^{x_i^2} = g^{z_i}$.

- ▷ Your first task is to determine the maximal subset J where $\emptyset \subset J \subseteq I$ such that each value (g^{x_j}, g^{z_j}) is a valid SDH pair. Record your answer in `Q1_J` as an array of integers, i.e. in the form `Q1_J = [500, 501, 4957, 26666]`.

We emphasise that, although your points are given as elements in E defined over $\mathbb{F}_p$, that the function `tate_pairing` should be called with input defined over $\mathbb{F}_{p^2}$. Since the distortion map involves the value i such that $i^2 = -1$, we suggest you instantiate the extension field with the irreducible polynomial $x^2 + 1$, e.g. `K.<a> = GF(p**2, modulus = x**2 + 1)` where `x` was previously defined via `R.<x> = GF(p)[]`.

Our cryptographic student noticed that there are many different curves that are specified in practice for use. The student devised this challenge to determine what kind of information might suffice to represent an elliptic curve.

Let $E : y^2 = x^3 + Ax + B$ be an elliptic curve defined over a field $\mathbb{F}_q$, where q is given in `Q1_q` as an integer. You are given that the points $S = \{(x, y), (x + 1, y), (x + 2, y)\}$ lie on E for some x and y, and that (x_0, y_0) lies on E, which is stored as `Q1_x0, Q1_y0` in integer format.

▷ Your task is to recover the j-invariant inv of E (as defined in lecture slide 369). Record your answer as two integers a, b such that $inv = \frac{a}{b}$ and $\gcd(a, b) = 1$. Record your answer in `Q1_a` and `Q1_b` as integers for a and b as above respectively. If inv is negative, you must set a, b such that $a < 0$ and $b > 0$.

Exercise 2 It's Paillier Time

Being proud of his French heritage, our crypto-apprentice decided that Paillier cryptosystem is the best cryptosystem ever! First we describe how the Paillier cryptosystem works.

It consists of 3 efficient algorithms Gen,Enc, Dec.

Key generation We start by describing the key generation algorithm. Gen picks 2 distinct primes (p, q) of equal length such that $p \nmid q-1$ and $q \nmid p-1$, and computes $n = pq$, $\lambda = LCM(p-1, q-1)$ and $g \in \mathbb{Z}^*_{n^2}$, where $n|ord(g)$ [1]. μ is defined to be $(L(g^\lambda \mod n^2))^{-1} \mod n$, where $L(x) = \frac{x-1}{n}$. We define $pk = (n, g)$ and $sk = (\lambda, \mu)$.

Encryption Let $pk = (n, g)$. To encrypt a message $0 \leq m < n$, first we pick a non-zero random element $r \in \mathbb{Z}^*_n$ and compute ciphertext $c = g^m r^n \mod n^2$.

Decryption Let c be the ciphertext, $pk = (n, g)$ be the public key and $sk = (\lambda, \mu)$ be the secret key. The decrypted message $m = L(c^\lambda \mod n^2).\mu \mod n$.

You are given access to the oracle O which upon receiving a ciphertext c returns $\mathsf{LSB}(\mathsf{Dec}_{sk}(\mathrm{c}))$, the least significant bit of the decryption of the ciphertext. Given Q3_pk and access to oracle O, you are asked to decrypt a challenge ciphertext Q3_ck and enter it in your answers file in integer format.

We have implemented the aforementioned oracle and we host it on `lasecpi1.epfl.ch:5557`. In order to query you can do:

echo $SCIPER $QUERY| ncat lasecpi0.epfl.ch:5555

Where $SCIPER is your sciper and $QUERY is the ciphertext you want to query in integer format. The server will respond with a bit 0 or 1.

Hint: A nice property of the Paillier cryptosystem is that $Dec(Enc(m_1) \times Enc(m_2)) = m_1 + m_2$.

Exercise 3 Is that a Delphi reference ?

Throughout this exercise, we denote by $\mathsf{INT}(M)$ *the integer value of the hexadecimal representation* $\mathsf{HEX}(M)$ *of an UTF-8 message* M *(e.g.* $\mathsf{HEX}(\texttt{"A"}) = \texttt{0x41}$ *and* $\mathsf{INT}(\texttt{"A"}) = 65$*). The goal of this exercise is to assess the security of the textbook RSA cryptosystem when the underlying prime numbers are insecurely generated or when the adversary is given access to some decryption oracles whose usage is described at the end of the exercise. The exercise contains three independent challenges ordered by difficulty.*

An RSA public key is denoted by $\mathsf{pk} = (e, N)$*, where* e *is the public exponent and* $N = pq$ *is an RSA modulus where* p *and* q *are distinct* λ*-bit primes. The corresponding secret key is denoted by* $\mathsf{sk} = (d, N)$ *where* $d = e^{-1} \bmod \phi(N)$*. The RSA textbook encryption and decryption algorithms are denoted by* $\mathsf{Enc}(\cdot, \mathsf{pk})\colon \mathbf{Z}_N \longrightarrow \mathbf{Z}_N$ *and* $\mathsf{Dec}(\cdot, \mathsf{sk})\colon \mathbf{Z}_N \longrightarrow \mathbf{Z}_N$ *respectively. In particular, a ciphertext can be uniquely decrypted only if the original plaintext is an element of* $\mathbf{Z}_N$*.*

Parameters and answers values are expected to either be Python `int` *or* `str` *instances. In particular, base64 encodings must be reported as* `answer="aGVsbG8="` *or* `answer='aGVsbG8='` *and <u>not</u>* `answer=b'aGVsbG8='` *(also note the presence of the quotes).*

In a world of swords and magic, a famous archaeologist explores some ancient ruins, but loses his way. While searching for the exit, he stumbles upon a door which foreign characters

[1] $ord(g)$ is the multiplicative order of g in $\mathbb{Z}^*_{p^2}$

are engraved on, indicating the direction to follow. However, this direction will only be revealed if he manages to solve the following challenge.

Algorithm 1: `get_prime`(1^λ)

Input: A security parameter λ, an odd integer α, a sparse integer μ and an integer ℓ.
Output: A random λ-bit prime p.

1 $s \xleftarrow{\$} \mathbf{Z}_{2^\ell}$ ▷ `A random` ℓ`-bit integer.`
2 $p \leftarrow \perp$
3 **while** p *is not prime* **do**
4     $s \leftarrow \mathsf{bitwise\_or}(s, \mu)$ ▷ `In Python:  s |= mu`
5     $p \leftarrow 0$
6   **for** $i = 0, \ldots, \lfloor \lambda/\ell \rfloor - 1$ **do**
7         $p \leftarrow (p \mathbin{<\!<} \ell) + s$
8         $s \leftarrow (\alpha s) \bmod 2^\ell$
9 **return** p

▷ Consider an odd integer α given as `Q3a_a`, a sparse integer μ given as `Q3a_m` and a small integer ℓ given as `Q3a_l`. Let $p < q$ be 1024-bit prime numbers generated by Algorithm 1 with parameters (α, μ, ℓ). Given the 2048-bit integer $N = pq$ as `Q3a_N`, recover p and q such that $p < q$ and report them under `Q3a_p` and `Q3a_q` respectively in the answers file as Python `int` objects.

Hint: Look at the *binary* representation of μ. What can you say about the first and last bits of μ and how does it affect s at line 4.

Hint: Each for-loop iteration updates s as $s \leftarrow \alpha s \bmod 2^\ell$. Let $s^{(i)}$ denote the value of s at the beginning of iteration $i = 0, \ldots, \lfloor \lambda/\ell \rfloor - 1 = t$. Express the integer p as a polynomial $\psi(\alpha, s^{(0)})$ and express $s^{(i)}$ in terms of α and $s^{(0)}$. If $p = \psi(\alpha, s^{(0)})$ is prime, then we say that $s^{(0)} = s_p^{(0)}$ is an initial seed for p.

Hint: By writing $N = pq = \psi(\alpha, s_p^{(0)}) \cdot \psi(\alpha, s_q^{(0)})$ with $s_p^{(0)}$ and $s_q^{(0)}$ the respective initial seeds, express $N \bmod 2^{2\ell}$ in terms of α, ℓ, $s_p^{(t)}$ and $s_q^{(t)}$ and recover $s_p^{(t)} s_q^{(t)}$. Reverse the prime generation to recover p and q.

After solving the first challenge, our archaeologist keeps going on until being stopped by a magical force field. The latter can only be deactivated by entering a specific password in a terminal whose textbook RSA ciphertext is displayed on. In order to solve this challenge, the oracle $\mathcal{O}_1(\cdot)$ described by Algorithm 2 can be accessed at `lasecpi1.epfl.ch:8888`.

Algorithm 2: $\mathcal{O}_1(c)$

Data: An RSA key pair $(\mathsf{pk}, \mathsf{sk})$ for a 1024-bit modulus N.
Input: An RSA ciphertext $c \in \mathbf{Z}_N$.
Output: A bit $b \in \{0, 1\}$.

$m \leftarrow \mathsf{Dec}(c, \mathsf{sk})$
$b \leftarrow m \bmod 2$
return b

▷ Given an RSA public key (e, N) as `Q3b_e` and `Q3b_N` and a ciphertext $C \in \mathbf{Z}_N$ as `Q3b_C`, recover the UTF-8 plaintext M such that $C = \mathsf{INT}(M)^e \bmod N$. Report the base64 encoding of M under Q3b_M in the answers file as a Python `str` object.

Hint: The oracle reveals the least significant bit of a plaintext. Does the oracle suffer from some homomorphic property? Stated otherwise, are there functions $f_i(\mathsf{pk})$ such that $\mathcal{O}_1(C \cdot f_i(\mathsf{pk}))$ leaks the i-th bit of the plaintext?

The path being cleared, our protagonist continues his exploration and eventually finds the sealed exit. As expected, this final riddle also asks the challenger to decrypt some ciphertext and provides the oracle $\mathcal{O}_2(\cdot)$ described by Algorithm 3 at `lasecpi1.epfl.ch:8889` and parametrized by an integer $\ell \geq 1$.

Algorithm 3: $\mathcal{O}_2(c)$

Data: An RSA key pair $(\mathsf{pk}, \mathsf{sk})$ for a 1024-bit modulus N and a parameter $\ell \geq 1$.
Input: An RSA ciphertext $c \in \mathbf{Z}_N$.
Output: A bit $b \in \{0, 1\}$.

$\mathsf{pk} \to (e, N)$
$t \xleftarrow{\$} [7, N/\ell] \cap \mathbf{Z}$ ▷ `t is a random integer`
$m \leftarrow \mathsf{Dec}(c, \mathsf{sk})$
$b \leftarrow (m < t) \mathbin{?} 1 : 0$
return b

▷ Given an RSA public key (e, N) as Q3c_e and Q3c_N, a parameter ℓ as Q3c_l and a ciphertext $C \in \mathbf{Z}_N$ as Q3c_C, recover the UTF-8 plaintext M such that $C = \mathsf{INT}(M)^e \bmod N$. Report the base64 encoding of M under Q3c_M in the answers file as a Python `str` object.

Hint: For $f(x) = C \cdot \mathsf{Enc}(x, \mathsf{pk})$, express $g(x) = \mathsf{Dec}(f(x), \mathsf{sk})$ as a function of $\mathsf{INT}(M)$ and x. Deduce that $\mathcal{O}_2(f(x))$ leaks whether $g(x)$ is small or not.

Hint: The extended Euclidean algorithm over rings $S \subseteq \mathbf{Z}$ for which a division with remainder (henceforth: `div_rem`) makes sense is given as follows[2]:

```
def xgcd(x, y, zero=0):
    a, b, s, t, u, v = x, y, 1, 0, 0, 1
    while b != zero:
        q, r = div_rem(a, b)
        a, b = b, r
        s, t = s - q * u, t - q * v
        s, t, u, v = u, v, s, t
    return a, s, t
```

Given a ring $S \subseteq \mathbf{Z}$, let $(q, r) = \texttt{div\_rem}(x, y)$ be such that $x = qy + r$ (in S). Those values[3] can be found by applying a binary search on the interval $[1, 2^b] \cap \mathbf{Z}$ where b is the smallest integer for which $2^b y \succcurlyeq x$ for some "comparison operator" $\succcurlyeq$ for S. The binary search then outputs some q such that $|x - qy|$ is minimal and we write $r = x - qy$.

Hint: Let $S = \{(x, f(x)) : x \in \mathbf{Z}_N\}$ be embedded into $\mathbf{Z}$ by projecting onto the second

[2] If `div_rem` is the `divmod` Python builtin, then `xgcd` is the usual EEA.
[3] If $S = \mathbf{Z}$, then we would have the usual division with remainder.

component[4]. For all $s_1 = (x_1, f(x_1))$ and $s_2 = (x_2, f(x_2))$ in S, we define

$$s_1 +_S s_2 = (x_1 + x_2, f(x_1 + x_2)) \quad \text{and} \quad s_1 \cdot_S s_2 = (x_1 x_2, f(x_1 x_2))$$

and let $\mathbf{0} = f(0)$ and $\mathbf{1} = f(1)$. Then, $(S, +_S, \cdot_S, \mathbf{0}, \mathbf{1})$ inherits an Euclidean ring structure compatible with the absolute value. We eventually define an *heuristic*[5] "comparison operator" $\succcurlyeq$ by $s_1 \succcurlyeq s_2$ if $\mathcal{O}_2(f(x_1 - x_2)) = 1$. Then, one may implement the `div_rem` and `xgcd` algorithms for S.

Hint: If $(d, f(d)) \in S$ satisfies $f(d) = 1$, then M can be easily recovered. In practice, find $a, b \in S$ such that $a, b \succcurlyeq \mathbf{0}$ and $\gcd(a, b) = (d, f(d))$ satisfies $f(d) = 1$. The number of oracle calls should be around $\mathcal{O}(2^{13})$. If this is not the case, start the attack with an other pair (a, b) since most of the oracles are done when computing `div_rem` in `xgcd`.

Oracle communications

The oracles $\mathcal{O}_1$ and $\mathcal{O}_2$ accessible[6] at `lasecpi1.epfl.ch:8888` and `lasecpi1.epfl.ch:8889` respectively expect their input $c \in \mathbf{Z}_N$ to be formatted as[7]

$$\texttt{payload} = \texttt{str(sciper)} + \texttt{" "} + \texttt{hex(c)[2:]} + \texttt{"\textbackslash n"}$$

Both the `sciper` and c values must be Python `int` objects. For instance, if `sciper = 123456` and c = INT("A?") = `0x413f`, the payload will be `payload = "123456 413f\n"`. The following example illustrates the usage of the Python builtin `socket` module.

```
#!usr/bin/env python3
# -*- coding: utf-8 -*-

import socket

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as csock:
    csock.connect(('lasecpi1.epfl.ch', 8888)) # connect to the oracle (O_1)
    for c in range(1, 100):
        data = str(123456) + " " + hex(c)[2:] + "\n"
        csock.send(payload.encode()) # call the oracle O_1(c)
        b = int(csock.recv(1).decode()) # read the result
        ... # do some other work
    # socket resources are released when exiting this context
```

[4]Note that the first component is entirely determined by the second one, but is kept for sake of clarity

[5]The comparison is heuristic because the oracle is probabilistic.

[6]Students must use EPFL VPN or be on site to be able to access the domain.

[7]The final "\n" character is of utmost importance as it tells the oracle the end of the payload.