# Homework 2 – RSA and Elliptic Curves

*Cryptography and Security 2022*

◇ 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 `[id]-answers.txt` file where `[id]` is an integer derived from the student ID.[1] This means you need to provide us with all `Q`-values specified in the questions below. Personal files are to be found at

http://lasec.epfl.ch:80/courses/cs22/hw2/index.php.

◇ **Please do not put any comment or strange character or any new line** in the submission file and do **NOT** rename the provided files.

◇ Do **NOT** modify the `SCIPER`, `id` and `seed` headers in the `[id]-answers.txt` file.

◇ **Submissions that do not respect the expected format may lose points.**

◇ 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 normal textual files containing Python/SAGE code. If an answer is incorrect, we may grant partial marks depending on the implementation.

◇ Be careful to always cite external code that was used in your implementation if the latter is not part of the public domain and include the corresponding license if needed. Submissions that do not meet this guideline may be flagged as plagiarism or cheating.

◇ Some plaintexts may contain random words. Do not be offended by them and search them online at your own risk. Note that they might be really strange.

◇ Please list the names of **every** person you worked with in the designated area of the answers file. Write down the list as a **comma-separated list** on a **single line**.

◇ Corrections and revisions may be announced on Moodle in the "News" forum. By default, 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 **December 1st, 2022** at 23h59.

[1] Depending on the nature of the exercise, an example of parameters and answers will be provided on Moodle.

## Exercise 1 Why bother hiding?

**Notation 1.** By convention, $\mathbb{N}$ denotes the set of positive integers $\{1, \ldots, \}$. For $a \leq b$, we denote by $[\![a, b]\!]$ the integral range $\{a, a+1, \ldots, b\}$.

**Notation 2.** A polynomial $h = h_0 + h_1 x + \cdots + h_\ell x^\ell \in \mathbf{Z}[x]$ is identified to its vector of components $\boldsymbol{h} \overset{\text{def.}}{=} (h_0, \ldots, h_\ell) \in \mathbf{Z}^{\ell+1}$. We abusively write $\|h\|$ instead of $\|\boldsymbol{h}\| \overset{\text{def.}}{=} \sqrt{\sum_{i=0}^{\ell} |h_i|^2}$ and $\|h(\gamma x)\|$ instead of $\|(h_0, \gamma h_1, \ldots, \gamma^\ell h_\ell)\|$ for $\gamma \in \mathbf{Z}$.

A textbook RSA keypair $(\mathsf{pk}, \mathsf{sk})$ for a security parameter $1^\lambda$ written in unary notation consists of a *public key* $\mathsf{pk} = (e, N)$ and a *private key* $\mathsf{sk} = (d, N)$ such that $N = pq$ is the product of two distinct $\lambda$-bit prime numbers, $d = e^{-1} \bmod \phi$ and $\phi = \varphi(N)$ where $\varphi \colon \mathbb{N} \longrightarrow \mathbb{N}$ denotes the Euler totient function. In this exercise, we are interested in recovering the private key given partial information. We first look at the following problem:

> *Given an integer $N$ of unknown factorization, a monic polynomial $f \in \mathbf{Z}[x]$ of degree $\delta$ and a bound $\gamma$, find $x_0 \in \mathbf{Z}$ such that $|x_0| < \gamma$ and $f(x_0) \equiv 0 \pmod N$.* $(\mathbf{P}_0)$

One approach to solve the above problem is to find a polynomial $h \in \mathbf{Z}[x]$ such that every integer $x_0 \in \mathbf{Z}$ satisfying $|x_0| < \gamma$ and $f(x_0) \equiv 0 \pmod N$ also satisfies $h(x_0) = 0$ over the integers (i.e., deprived of a modulo $N$). Stated otherwise, we reduce the problem of finding a *modular* root to the problem of finding an *integral* root for which we have standard methods. In order to construct such $h$, let $m \geq 1$ and $t \geq 1$ be positive integers and define the polynomials $u_{ij}(x) \overset{\text{def.}}{=} x^i N^{m-j} f^j(x) \in \mathbf{Z}[x]$ for all $(i, j) \in [\![0, \delta-1]\!] \times [\![0, m-1]\!]$ and the polynomials $v_k(x) \overset{\text{def.}}{=} x^k f^m(x) \in \mathbf{Z}[x]$ for all $k \in [\![0, t-1]\!]$.

Clearly, if $f(x_0) \equiv 0 \pmod N$, then $u_{ij}(x_0) \equiv 0 \pmod{N^m}$ and $v_k(x_0) \equiv 0 \pmod{N^m}$. More generally, if $h$ is an integral linear combination of $u_{ij}$ and $v_k$, then $h(x_0) \equiv 0 \pmod{N^m}$ as well. In particular, if $|h(x_0)| < N^m$, then $h(x_0) = 0$ over the integers since the only multiple of $N^m$ strictly smaller than $N^m$ in absolute value is 0. This observation is generalized by the following result:

**Theorem 1.** *Let $h \in \mathbf{Z}[x]$ be a polynomial with $n$ monomials and let $\gamma$, $N$ and $m$ be positive integers such that $\|h(\gamma x)\| < N^m/\sqrt{n}$. If $x_0 \in \mathbf{Z}$ satisfies $|x_0| < \gamma$ and $h(x_0) \equiv 0 \pmod{N^m}$, then the identity $h(x_0) = 0$ holds over the integers.*

In order to apply the theorem above, the polynomial $h$ *must* be of "small norm" (illustrated by the condition $\|h(\gamma x)\| < N^m/\sqrt{n}$). Since $h$ is an integral linear combination of $u_{ij}$ and $v_k$, it follows that $\hat{h} \overset{\text{def.}}{=} h(\gamma x)$ is an integral linear combination of $\hat{u} \overset{\text{def.}}{=} u_{ij}(\gamma x)$ and $\hat{v}_k \overset{\text{def.}}{=} v_k(\gamma x)$.

**Definition 1** (lattice)**.** Let $\mathscr{B} = \{\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k\} \subset \mathbb{R}^n$ be a set of $k$ linearly independent vectors in $\mathbb{R}^n$ and let $\mathbf{G}(\mathscr{B}) \overset{\text{def.}}{=} \begin{bmatrix} \boldsymbol{b}_1 & \cdots & \boldsymbol{b}_k \end{bmatrix} \in \mathbb{M}_{n \times k}(\mathbb{R})$ be the corresponding matrix. The *lattice generated by* $\mathscr{B}$, denoted by $\Lambda = \Lambda(\mathscr{B})$, is defined to be the set of all *integral* linear combinations of $\mathscr{B}$, namely $\Lambda(\mathscr{B}) \overset{\text{def.}}{=} \left\{\sum_{i=1}^{k} z_i \boldsymbol{b}_i \colon z_i \in \mathbf{Z}\right\} = \left\{\mathbf{G}(\mathscr{B}) \cdot \boldsymbol{z} \colon \boldsymbol{z} \in \mathbf{Z}^k\right\}$. We say that $n$ (resp. $k$) is the *dimension* (resp. *rank*) of $\Lambda$ and that $\Lambda$ is *full-rank* if $n = k$. The set $\mathscr{B}$ is called a *basis* for $\Lambda$ and the *determinant* of $\Lambda$, denoted by $\det \Lambda$, is defined to be $\det \Lambda \overset{\text{def.}}{=} |\det \mathbf{G}(\mathscr{B})|$.

Under the identification of polynomials to their vector of coordinates, $\hat{\boldsymbol{h}}$ is an integral linear combination of $\hat{\boldsymbol{u}}_{ij}$ and $\hat{\boldsymbol{v}}_k$, namely $\hat{\boldsymbol{h}} \in \Lambda(\mathscr{B})$ where $\mathscr{B} = \{\hat{\boldsymbol{u}}_{0,0}, \ldots, \hat{\boldsymbol{u}}_{\delta-1,m-1}, \hat{\boldsymbol{v}}_0, \ldots, \hat{\boldsymbol{v}}_{t-1}\}$. Since Theorem 1 requires $\hat{h}$ to satisfy $\|\hat{h}\| < N^m/\sqrt{n}$, we need a way to find "short" vectors in the lattice $\Lambda(\mathscr{B})$. This can be achieved via the Lenstra-Lenstra-Lovaśz (LLL) algorithm, which takes as input an $n$-dimensional full-rank *integral* lattice $\Lambda = \Lambda(\mathscr{B}) \subseteq \mathbf{Z}^n$ and outputs a reduced ordered basis $\mathscr{B}^* = \{\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_n^*\}$ for $\Lambda$ such that $\|\boldsymbol{b}_1^*\| \leq 2^{\frac{n-1}{4}}(\det \Lambda)^{\frac{1}{n}}$.[2] In our case, $n = m\delta + t$ and $\mathscr{B} = \{\hat{\boldsymbol{u}}_{0,0}, \ldots, \hat{\boldsymbol{u}}_{\delta-1,m-1}, \hat{\boldsymbol{v}}_0, \ldots, \hat{\boldsymbol{v}}_{t-1}\} \subseteq \mathbf{Z}^n$. By construction, $\mathbf{G}(\mathscr{B})$ is an upper-triangular matrix in the canonical basis. Therefore, $\det \Lambda(\mathscr{B}) = N^{\frac{m\delta(m+1)}{2}} \gamma^{\frac{n(n-1)}{2}}$. It remains to choose the integers $m$ and $t$ such that $2^{\frac{n-1}{4}}(\det \Lambda)^{\frac{1}{n}} < N^m/\sqrt{n}$ and the first vector of an LLL-reduced basis would be a suitable candidate for Theorem 1. Note that the whole approach that we presented can be extended to *any* of the divisors $\sigma$ of $N$. In particular, this can be used to factor $N$ by considering the following special case of ($\mathbf{P}_0$):

> *Given an integer $N$ of unknown factorization, a monic polynomial $f \in \mathbf{Z}[x]$ of degree $\delta$ and a real $\beta \in (0, 1]$ such that $N$ admits a divisor $\sigma \geq N^\beta$, find $x_0 \in \mathbf{Z}$ such that $|x_0| < N^{\beta^2/\delta}$ and $f(x_0) \equiv 0 \pmod{\sigma}$.* ($\mathbf{P}_1$)

Assume that there exists a real $0 < \varepsilon \leq \beta/7$ such that $\gamma = \lceil \frac{1}{2} N^{\beta^2/\delta - \varepsilon} \rceil$ and set $m = \lceil \beta^2/(\delta\varepsilon) \rceil$ and $t = \lfloor \delta m(1-\beta)/\beta \rfloor$. Recall that $u_{ij} = x^i N^{m-j} f^j$ and $v_k = x^k f^m$ and that $\hat{u}_{ij} \overset{\text{def.}}{=} u_{ij}(\gamma x)$ and $\hat{v}_k \overset{\text{def.}}{=} v_k(\gamma x)$. Let $n = m\delta + t$ and consider the $n$-dimensional full-rank lattice $\Lambda(\mathscr{B})$ generated by $\mathscr{B} = \{\hat{\boldsymbol{u}}_{0,0}, \ldots, \hat{\boldsymbol{u}}_{\delta-1,m-1}, \hat{\boldsymbol{v}}_0, \ldots, \hat{\boldsymbol{v}}_{t-1}\}$. By construction, the first basis vector $\hat{\boldsymbol{h}}$ of an LLL-reduced basis satisfies $\|\hat{\boldsymbol{h}}\| < N^{\beta m}/\sqrt{n} \leq \sigma^m/\sqrt{n}$. By Theorem 1, this implies that a root $x_0$ of the polynomial $h(x)$ derived from the polynomial $h(\gamma x)$ corresponding to $\hat{\boldsymbol{h}}$ is a root of $f$ modulo $\sigma$. Since $\sigma$ divides $N$, it follows that $\gcd(f(x_0), N)$ is a divisor of $N$.

*Example* 1. Let $p > q$ be the 40-bit primes defined by

$$p = \texttt{0xdaab98ea97}, \quad q = \texttt{0x988fc3cd75}$$

and let $N = pq = \texttt{0x8250afe5e986eac32203}$ be the corresponding 80-bit RSA modulus. Assume that the $\ell = 12$ least significant bits of $p$ are unknown, namely $p = 2^\ell p' + p''$ for some unknown $0 \leq p'' < 2^{12}$. Let $a \overset{\text{def.}}{=} 2^\ell p' = \texttt{0xdaab98e000}$ and $f(x) = a + x \in \mathbf{Z}[x]$ be a polynomial of degree $\delta = 1$. Set $\beta = \frac{1}{2}$ and $\varepsilon = \beta/7$ and define $\gamma = \lceil \frac{1}{2} N^{\beta^2/\delta - \varepsilon} \rceil = 8852 > 2^{12} > p''$. Now, choose $m = \lceil \beta^2/(\delta\varepsilon) \rceil = 4$ and $t = \lfloor \delta m(1-\beta)/\beta \rfloor = 4$ and look for a short vector in the lattice generated by the matrix corresponding to $\mathscr{B} = \{\hat{\boldsymbol{u}}_{00}, \hat{\boldsymbol{u}}_{01}, \hat{\boldsymbol{u}}_{02}, \hat{\boldsymbol{u}}_{03}, \hat{\boldsymbol{v}}_0, \hat{\boldsymbol{v}}_1, \hat{\boldsymbol{v}}_2, \hat{\boldsymbol{v}}_3\}$:

$$\mathbf{G}(\mathscr{B}) = \begin{array}{cc} \begin{array}{cccccccc} \hat{\boldsymbol{u}}_{00} & \hat{\boldsymbol{u}}_{01} & \hat{\boldsymbol{u}}_{02} & \hat{\boldsymbol{u}}_{03} & \hat{\boldsymbol{v}}_0 & \hat{\boldsymbol{v}}_1 & \hat{\boldsymbol{v}}_2 & \hat{\boldsymbol{v}}_3 \end{array} & \\ \begin{bmatrix} N^4 & aN^3 & a^2N^2 & a^3N & a^4 & & & \\ & \gamma N^3 & 2\gamma a N^2 & 3\gamma a^2 N & 4\gamma a^3 & \gamma a^4 & & \\ & & \gamma^2 N^2 & 3\gamma^2 a N & 6\gamma^2 a^2 & 4\gamma^2 a^3 & \gamma^2 a^4 & \\ & & & \gamma^3 N & 4\gamma^3 a^4 & 6\gamma^3 a^2 & 4\gamma^3 a^3 & \gamma^3 a^4 \\ & & & & \gamma^4 & 4\gamma^4 a & 6\gamma^4 a^2 & 4\gamma^4 a^3 \\ & & & & & \gamma^5 & 4\gamma^5 a & 6\gamma^5 a^2 \\ & & & & & & \gamma^6 & 4\gamma^6 a \\ & & & & & & & \gamma^7 \end{bmatrix} & \begin{array}{c} 1 \\ x \\ x^2 \\ x^3 \\ x^4 \\ x^5 \\ x^6 \\ x^7 \end{array} \end{array}.$$

[2]In SAGE, `B.LLL()` computes the LLL-reduced matrix corresponding to the basis associated with $B^\top$.

The first vector obtained by computing an LLL-reduced basis of the lattice $\Lambda(\mathscr{B})$ is

$$\hat{\boldsymbol{h}} = \begin{bmatrix} \hat{h}_0 \\ \hat{h}_1 \\ \hat{h}_2 \\ \hat{h}_3 \\ \hat{h}_4 \\ \hat{h}_5 \\ \hat{h}_6 \\ \hat{h}_7 \end{bmatrix} = \begin{bmatrix} \texttt{0x1099fd3ae2d66c3f108a6a399d9199cc4c9} \\ \texttt{0x9c34a810996e8dc54182bb12fb9ce0a6c10} \\ -\texttt{0x262a7de97f5ca539f77699e3141df22b7270} \\ -\texttt{0x267da0f49ba88bfa4bbb0f4165b9ae8cd9c0} \\ \texttt{0x3db25765c91eb5d70d119e88495008dff600} \\ \texttt{0x2c5a78bc3b62a6bb9403f33c8442326f0400} \\ -\texttt{0x63952b22c3a1a3cfcec22e2298a4db48000} \\ \texttt{0x2dc72e94b4a97d56b3964708a981c0c8000} \end{bmatrix},$$

By construction, this corresponds to a polynomial $h(\gamma x)$. In particular,

$$\boldsymbol{h} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \end{bmatrix} = \begin{bmatrix} \hat{h}_0/\gamma^0 \\ \hat{h}_1/\gamma^1 \\ \hat{h}_2/\gamma^2 \\ \hat{h}_3/\gamma^3 \\ \hat{h}_4/\gamma^4 \\ \hat{h}_5/\gamma^5 \\ \hat{h}_6/\gamma^6 \\ \hat{h}_7/\gamma^7 \end{bmatrix} = \begin{bmatrix} \texttt{0x1099fd3ae2d66c3f108a6a399d9199cc4c9} \\ \texttt{0x4847900d3bc66a8d0448fc3784bcbbb4} \\ -\texttt{0x82bf5ed6b1d25be7fc66480afcd7f} \\ -\texttt{0x3d03ab12805b00b63d74a4e993} \\ \texttt{0x2d40fd8484ac5c6907a7356} \\ \texttt{0xf0db71cdc783ea8a4bd} \\ -\texttt{0xfa3a3ecaf11f508} \\ \texttt{0x3539f55fb97a} \end{bmatrix}$$

corresponds to $h(x)$. Since $r = 2711 < \gamma$ is a root of $h$, it follows that $p = \gcd(f(r), N)$. Note that the approach above can be extended to the case when the least significant bits are known.

The following questions are direct applications of the lattice-based method that was described above. Each question essentially asks you to find the polynomial $f(x)$ to attack. Note that each RSA moduli are sufficiently large to withstand factorization via SAGE.

## Question 1.1

- ▷ Given a 1024-bit RSA modulus $N = pq$ as Q1a_N with $p > q$, a public exponent $e \geq 3$ associated with an unknown private exponent $d = e^{-1} \bmod \varphi(N)$ as Q1a_e and the $\ell$ *most* significant bits of $d_p = d \bmod \varphi(p)$ as Q1a_top with $\ell =$ Q1a_ell, recover $d_p$ and report it as an integer under Q1a_val.

  **Hint**: Write $d_p$ as $d_p = \mathsf{top} + r$ where top are the $\ell$ most significant bits of $d_p$.

  **Hint**: Apply the Chinese Remainder Theorem to the identity $ed = 1 \bmod \varphi(N)$ to find a polynomial $f(x) = a + x$ such that $a$ is a known constant and find a small root of $f$.

## Question 1.2

- ▷ Given a 1024-bit RSA modulus $N = pq$ as Q1b_N with $p > q$, a public exponent $e \geq 3$ associated with an unknown private exponent $d = e^{-1} \bmod \varphi(N)$ as Q1b_e and the $\ell$ *least* significant bits of $d$ as Q1b_low with $\ell =$ Q1b_ell, recover $d$ and report it as an integer under Q1b_val.

  **Hint**: Use the $\ell$ least significant bits of $d$ to recover the $\ell$ *least* significant bits of $p$.

**Hint**: Study the solutions of the equation $kx^2 - k(p+q)x + kN \equiv 0 \pmod{2^\ell}$ for $k \in \mathbf{Z}$.

**Hint**: Use Hensel's lifting lemma to solve quadratic equations modulo a prime power.

**Lemma** (Hensel)**.** *Let $p$ be a prime, $k \geq 1$ be an integer and $h \in \mathbf{Z}[x]$ be a polynomial of first order derivative $\frac{dh}{dx} \in \mathbf{Z}[x]$. Let $r \in \mathbf{Z}$ be a root of $h$ modulo $p^k$ and $\alpha \overset{\text{def.}}{=} \frac{dh}{dx}(r)$.*

*(a) If $p$ does not divide $\alpha$, then, for every integer $\bar{\alpha} \in \mathbf{Z}$ such that $\alpha\bar{\alpha} \equiv 1 \pmod{p}$, the integer $\bar{r} = r - (\bar{\alpha} \bmod p) \cdot h(r)$ satisfies $h(\bar{r}) \equiv 0 \pmod{p^{k+1}}$ and $\bar{r} \equiv r \pmod{p^k}$.*

*(b) If $p$ divides $\alpha$, then exactly one of the following assertions holds:*

*i. $h(r) \equiv 0 \pmod{p^k}$ and $h(r + tp^k) \equiv 0 \pmod{p^{k+1}}$ for all $t \in [\![0, p-1]\!]$.*

*ii. $h(r) \not\equiv 0 \pmod{p^k}$ and there is no lifting of $r$ to a root of $h$ modulo $p^{k+1}$.*

## Question 1.3

Let $\mathcal{R}(x,y) \overset{\text{def.}}{=} \mathbb{1}_{\gcd(x,\varphi(y))=1}$ be the function which outputs 1 if $\gcd(x, \varphi(y)) = 1$ and 0 otherwise.

▷ Given a list of 1024-bit RSA moduli $(N_0, \ldots, N_k)$ as Q1c_N such that $N_i = p_i q_i$ together with a list $(e_0, \ldots, e_k)$ of prime numbers such that $\gcd(e_i, N_i) = 1$ and $e_i \geq \sqrt{2} \cdot N_i^{3/10}$ as Q1c_E, compute the integer $b \overset{\text{def.}}{=} \sum_{i=0}^{k} \mathcal{R}(e_i, N_i) \cdot 2^i$ and report it under Q1c_flag.

**Hint**: Use the fact that $m \mapsto m^{e_i} \bmod N_i$ is a bijection if and only if $\mathcal{R}(e_i, N_i) = 1$ and study the consequences of $\varphi(N_i) \equiv 0 \pmod{e_i}$.

**Hint**: Use the fact that $\gcd(e_i, N_i) = 1$ to construct an algorithm $\mathcal{A}(e_i, N_i) \to (p_i, q_i)/\perp$ that factors $N_i$ if and only if $\mathcal{R}(e_i, N_i) = 0$.

## Exercise 2 Forgot my keys

We decided to use the RSA cryptosystem to encrypt our messages. Moreover, we want to encrypt our message bit by bit instead of a whole. However, it would be trivial to break since the plaintext space is too small. Hence, we decided to come up with the following encryption scheme.

```
KeyGen()
1 : $(p, q) \leftarrow\!\!\$\ \mathsf{RSAPrimeGen}(1024)$
2 : $g \leftarrow \mathsf{GetPrime}(20)$
3 : $e \leftarrow 65537$
4 : $N \leftarrow p \cdot q$
5 : return $(e, N, g)$

Enc$(e, N, g, m)$
1 : parse $m \rightarrow (m_1, \ldots, m_\ell)$
2 : $C \leftarrow$ []
3 : for $i = 1 \ldots n$
4 :     $k \leftarrow\!\!\$\ \mathsf{PrimeGen}(64)$
5 :     $t \leftarrow (g^{k+m_i} \cdot k^{2022}) \bmod N$
6 :     $C \leftarrow C \| \mathtt{hex}(t^e \bmod N)$
7 : return $C$
```

- The $\mathsf{RSAPrimeGen}(\lambda)$ procedure outputs a pair of random $\lambda$-bit primes.
- The $\mathsf{PrimeGen}(\lambda)$ procedure outputs a randomly sampled $\lambda$-bit prime.
- The $\mathsf{GetPrime}(\lambda)$ procedure outputs a $\lambda$-bit prime from a predefined list of primes.
- `hex`$(c)$ is the hexadecimal representation of $c$ without a `0x` prefix, e.g., `hex`$(255) =$ `ff`.
- The message $m$ is parsed as the concatenation of its ASCII encodings, stripped from leading zeros. For instance, if `m="A B"`, then $(m_1, \ldots, m_\ell) =$ (1000001 00100000 01000010) since `"A" = 01000001, " "= 00100000, "B"= 01000010`.
- `[]` denotes the empty list and $\|$ denotes the list concatenation.

### Question 2.1

Using the encryption scheme described above, we encrypted a message $m$ using the provided public key (Q2a_e, Q2a_N, Q2a_g) and obtained the ciphertext Q2a_C. However, we realized that our key generation does not output the private key and thus are unable to decrypt that ciphertext. We only know that the prime $g$ has a certain structure. Can you find $m$ and report as a plaintext string it under Q2a_m?

## Exercise 3 RFCs are meant to be followed.

Let $E$ be an elliptic curve defined over a finite field $\mathbb{F}_p$ of characteristic $p > 3$ and consider the corresponding Weierstrass equation $E\colon y^2 = x^3 + ax + b$. Recall that the set of $\mathbb{F}_p$-rational points $E(\mathbb{F}_p)$ on $E$, namely points whose coordinates are elements of $\mathbb{F}_p$, together with the point at infinity $(\infty)$, form an abelian group. Let $G = (x_G, y_G) \in E(\mathbb{F}_p)$ be a rational point of order $n$ and cofactor $h \overset{\text{def.}}{=} \#E(\mathbb{F}_p)/n$. The goal of this exercise is to study key recovery attacks on the following *simplified* Elliptic Curve Diffie-Hellman (ECDH*) protocol:

ECDH* protocol for public parameters $\mathsf{pp} = (a, b, p, x_G, y_G, n, h)$

| **Alice** | | **Bob** |
|---|---|---|
| $\mathsf{sk}_A \xleftarrow{\$} \mathbf{Z}_n^\times$ | | $\mathsf{sk}_B \xleftarrow{\$} \mathbf{Z}_n^\times$ |
| $\mathsf{pk}_A \leftarrow \mathsf{sk}_A \cdot (x_G, y_G)$ | send $\mathsf{pk}_A$ to Bob $\longrightarrow$ | $\mathsf{pk}_B \leftarrow \mathsf{sk}_B \cdot (x_G, y_G)$ |
| $Z \leftarrow \mathsf{xcoord}(\mathsf{sk}_A \cdot \mathsf{pk}_B)$ | $\longleftarrow$ send $\mathsf{pk}_B$ to Alice | $Z \leftarrow \mathsf{xcoord}(\mathsf{sk}_B \cdot \mathsf{pk}_A)$ |

### Question 3.1 Do you want a smoothie?

▷ Given a tuple of ECDH* public parameters $\mathsf{pp} = (a, b, p, x_G, y_G, n, h)$ as Q3a_pp and Bob's public key $\mathsf{pk}_B$ as Q3a_pk, recover Bob's secret key $\mathsf{sk}_B$ and report it as an integer under Q3a_sk.

### Question 3.2 CRT to the rescue

Let $\mathbf{\Omega} \overset{\text{def.}}{=} (\mathbb{F}_p \times \mathbb{F}_p) \cup \{(\infty)\}$ be the ambient space of $\mathbb{F}_p$-rational points (including the point at infinity) of any elliptic curve defined over $\mathbb{F}_p$. Let $\oplus_{\mathsf{pp}} \colon \mathbf{\Omega} \times \mathbf{\Omega} \longrightarrow \mathbf{\Omega}$ be the addition law parametrized by a tuple of ECDH* public parameters $\mathsf{pp}$ and described by Algorithm 1 where arithmetic operations are modulo $p$.

**Algorithm 1:** add(pp, $P$, $Q$)

```
Input: A tuple pp = $(a, b, p, x_G, y_G, n, h)$ of ECDH* public parameters
       and the operands $P = (x_P, y_P) \in \mathbf{\Omega}$ and $Q = (x_Q, y_Q) \in \mathbf{\Omega}$.
Output: A point $R = (x_R, y_R) = P \oplus_{\mathsf{pp}} Q \in \mathbf{\Omega}$.
if $P = (\infty)$ then
    return $Q$
if $Q = (\infty)$ then
    return $P$
if $x_P = x_Q$ and $y_P = -y_Q$ then
    return $(\infty)$
if $x_P \neq x_Q$ then
    $\lambda \leftarrow \frac{y_Q - y_P}{x_Q - x_P}$
else
    $\lambda \leftarrow \frac{3x_P^2 + a}{2y_P}$
$x_R \leftarrow \lambda^2 - x_P - x_Q$
$y_R \leftarrow (x_P - x_R)\lambda - y_P$
return $(x_R, y_R)$
```

We now consider the following ECDH* oracle parametrized by a tuple $\mathsf{pp}$ of ECDH* public parameters and a secret key $\mathsf{sk} \in \mathbf{Z}_n^\times$. Given a point $P \in \mathbf{\Omega}$, the oracle computes $\mathsf{sk} \cdot P$ using the textbook double-and-add algorithm (with respect to the addition law $\oplus_{\mathsf{pp}}$).

```
Algorithm 2: $\mathcal{O}_{\mathsf{pp},\mathsf{sk}}(x_P, y_P)$
  Input: A point $P = (x_P, y_P) \in \mathbf{\Omega}$.
  Output: A point $Q = (x_Q, y_Q) \in \mathbf{\Omega}$.
  Data: A tuple $\mathsf{pp} = (a, b, p, x_G, y_G, n, h)$ of ECDH* public parameters
        and a secret key $\mathsf{sk} \in \mathbf{Z}_n^\times$.
1 $Q \leftarrow \mathsf{sk} \cdot P$     ▷ textbook double-and-add algorithm with addition given by $\mathsf{add}(\mathsf{pp}, -, -)$
2 return $Q$
```

▷ Given a tuple of ECDH* public parameters $\mathsf{pp} = (a, b, p, x_G, y_G, n, h)$ as Q3b_pp, use the remote oracle $\mathcal{O}_{\mathsf{pp},\mathsf{sk}}$ parametrized by $\mathsf{pp}$ and a secret key $\mathsf{sk}$ to recover $\mathsf{sk}$ and report it as a *positive* integer under Q3b_sk.

**Hint:** Find an elliptic curve $E' : y^2 = x^3 + ax + b'$ defined over $\mathbb{F}_p$ for some $b' \neq b$ and a rational point $P' \in E'(\mathbb{F}_p)$ of order 2 such that the double-and-add algorithm on $E$ coincides with the double-and-add algorithm on $E'$ for that point (carefully look at the addition formulae in Algorithm 1). Deduce the parity of $\mathsf{sk}$ using $P'$.

**Hint:** Generalize the above approach to completely recover $\mathsf{sk}$ using the Chinese Remainder Theorem.

*Remark* 1. Depending on the parameters and the oracle queries, this question may take a long time to be solved, especially if a naive bruteforce approach is considered. On average, this question should be solvable in less that 5 minutes. We highly recommend to implement a local oracle with smaller parameters before querying the real oracle.

## Oracle communications

The remote oracle endpoint for $\mathcal{O}_{\mathsf{pp},\mathsf{sk}}$ can be accessed[3] at `lasecpi1.epfl.ch:8888`. The oracle can be queried with $P = (x_P, y_P) \in \mathbf{\Omega}$ by sending the following payload:

```
args = [str(seed), format(x_P, "x"), format(y_P, "x")]
payload = " ".join(args) + "\n"
```

By convention, if $P$ is the point at infinity, namely $P = (\infty)$, then the payload is:

```
args = [str(seed), "inf", "inf"]
payload = " ".join(args) + "\n"
```

The `seed` is an `int` specified at the top of the parameters file and the final `"\n"` character is of utmost importance as it tells the oracle the end of the payload. For instance, if `seed = 123456` and `x_P = 222` and `y_P = 48`, then `payload` = `"123456 de 30\n"`. The following example illustrates the usage of the Python `socket` module for remote communications.

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

```
import socket

seed = 123456
P = (222, 48)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as csock:
    csock.connect(("lasecpi1.epfl.ch", 8888))
    args = [str(seed), format(P[0], "x"), format(P[1], "x")]
    payload = " ".join(args) + "\n"
    csock.send(payload.encode()) # call the oracle
    chunk1: bytes = csock.recv(SIZE) # read a chunk of SIZE bytes
    ... # maybe need to read more chunks
    res = b"".join([chunk1, chunk2, ...])
    # the oracle replies by sending x_Q and y_Q separated by a
    # space if Q != (oo), otherwise it replies with "inf inf"
    x_Q, y_Q = res.decode().split(" ")
    if x_Q == "inf" and y_Q == "inf":
        # do something knowing Q = (oo)
        ...
    else:
        x_Q, y_Q = int(x_Q), int(y_Q)
        # do something with the integers
        ...
    # socket resources are released when exiting the context
```

If the oracle call fails (e.g., because the payload is wrongly formatted), a descriptive help is sent as a response instead (the `SIZE` should be sufficiently large in order to read the whole message), unless the error was critical. For critical errors, please contact benedikt.tran@epfl.ch.