Homework 2 – PRG and Signature

Cryptography and Security 2025

- ◇ 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.sage` 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.
- ◇ **Please do not put any comment or strange character or any new line** in the `[id]-answers.sage` file and do **NOT** rename the provided files.
- ◇ Do **NOT** modify the `id` headers in the `[id]-answers.sage` file.
- ◇ **Submissions that do not respect the expected format may lose points.**
- ◇ If you do not answer a question, set the answer parameter in `[id]-answers.sage` as `None`.
- ◇ 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.
- ◇ You are allowed to work in a group of two. Please list the name of the person you worked with in the `readme` of your source code. Note that each of you must **individually** submit a valid answer file on Moodle.
- ◇ 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 17th** at 23h59.

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

Exercise 1 Trouble with Trusted Setup.

This exercise will cover two attempts to construct a pseudorandom number generator from a deterministic function, using trusted setup. At each round, the PRG will take as input a secret seed s_{i-1}, and output a tuple (s_i, t_i), with s_i secret, and t_i public. The value s_i can then be used to compute (s_{i+1}, t_{i+1}), and so on. After ℓ rounds, the output string is $(t_1, ..., t_\ell)$. The goal of this exercise is to first implement this PRG, and then to break it by predicting output values of the PRG, given only previously published values, public parameters, and a trapdoor.

Question 1.1

The first attempt is to build the PRG from the discrete logarithm problem. The public parameters are obtained from a trusted third party, and consist of a prime p, $g \in \mathbb{Z}_p^*$, which is of order q, where q is a large prime, and $h \neq g$, which is a distinct generator of the group generated by g. We consider the following algorithm

$f_1(pp = (p, g, h), s_{i-1})$

1 : $r_i \leftarrow g^{s_{i-1}} \mod p$
2 : $s_i \leftarrow g^{r_i} \mod p$
3 : $t_i \leftarrow h^{r_i} \mod p$
4 : **return** (s_i, t_i)

▷ Implement $\mathsf{PRG}_1(p, g, h, s_0, \ell)$ which takes public parameters, an initial seed s_0, and a length parameter ℓ, and returns $(t_1, ..., t_\ell)$ formatted as a list. Given the initial seed $\mathsf{Q1a\_s_0}$ and public parameters $\mathsf{Q1a\_p}, \mathsf{Q1a\_g}, \mathsf{Q1a\_h}$ in your parameter file, compute $\mathsf{PRG}_1(\mathsf{Q1a\_p}, \mathsf{Q1a\_g}, \mathsf{Q1a\_h}, \mathsf{Q1a\_s_0}, 5)$. This should return a list of values

$$[\mathsf{Q1a\_t_1}, \mathsf{Q1a\_t_2}, \mathsf{Q1a\_t_3}, \mathsf{Q1a\_t_4}, \mathsf{Q1a\_t_5}]$$

Question 1.2

Break PRG_1 using a trapdoor: now, you are not given the initial seed. Instead, you are given the public parameters $\mathsf{Q1b\_pp} = (\mathsf{Q1b\_p}, \mathsf{Q1b\_g}, \mathsf{Q1b\_h})$ and the first output of PRG_1: $\mathsf{Q1b\_t_1} = \mathsf{PRG}_1(\mathsf{Q1b\_pp}, \mathsf{Q1b\_s_0}, 1)$, where $\mathsf{Q1b\_s_0}$ is hidden. You are also given $\mathsf{Q1b\_e}$ such that $\mathsf{Q1b\_g} = \mathsf{Q1b\_h}^{\mathsf{Q1b\_e}}$.

▷ Compute $\mathsf{PRG}_1(\mathsf{Q1b\_p}, \mathsf{Q1b\_g}, \mathsf{Q1b\_h}, \mathsf{Q1b\_s_0}, 5)$. i.e., use the trapdoor to predict the next 4 outputs of PRG_1, without knowing the initial seed. This should return a list of values

$$[\mathsf{Q1b\_t_2}, \mathsf{Q1b\_t_3}, \mathsf{Q1b\_t_4}, \mathsf{Q1b\_t_5}]$$

Question 1.3

An attempt to circumvent this attack with malicious setup is to build the PRG from elliptic curves. Now our public parameters consist of a prime $p \equiv 3 \mod 4$, A, B which describe a non-singular elliptic curve $E : y^2 = x^3 + Ax + B$ over $\mathbb{F}_p$, and two points $P := (P_x, P_y), Q := (Q_x, Q_y)$ over $E(\mathbb{F}_p)$. Let $x : E(\mathbb{F}_p) \to \mathbb{F}_p$ be the function that takes a point on $\mathrm{E}(\mathbb{F}_p)$ and returns it's x-coordinate. We define f as follows

$f_2(pp = (p, A, B, P_x, P_y, Q_x, Q_y), s_{i-1})$

1 : $r_i \leftarrow x(s_{i-1}P)$
2 : $s_i \leftarrow x(r_iP)$
3 : $t_i \leftarrow x(r_iQ)$
4 : **return** (s_i, t_i)

▷ Implement $\mathsf{PRG}_2(pp, s_0, \ell)$ which takes public parameters pp, an initial seed s_0, and a length parameter ℓ, and returns $(t_1, ..., t_\ell)$ formatted as a list. Given the initial seed $\mathsf{Q1c\_s_0}$ and public parameters $(\mathsf{Q1c\_p}, \mathsf{Q1c\_A}, \mathsf{Q1c\_B}, \mathsf{Q1c\_Px}, \mathsf{Q1c\_Py}, \mathsf{Q1c\_Qx}, \mathsf{Q1c\_Qy})$ in your parameter file, compute $\mathsf{PRG}_2((\mathsf{Q1c\_p}, \mathsf{Q1c\_A}, \mathsf{Q1c\_B}, \mathsf{Q1c\_Px}, \mathsf{Q1c\_Py}, \mathsf{Q1c\_Qx}, \mathsf{Q1c\_Qy}), \mathsf{Q1c\_s_0}, 5)$. This should return a list of values This should return a list of values

$$[\mathsf{Q1c\_t_1}, \mathsf{Q1c\_t_2}, \mathsf{Q1c\_t_3}, \mathsf{Q1c\_t_4}, \mathsf{Q1c\_t_5}]$$

Question 1.4

Break PRG_2 using a trapdoor. Now, you are not given the initial seed. Instead, you are given the public parameters $\mathsf{Q1d\_pp} = (\mathsf{Q1d\_p}, \mathsf{Q1d\_A}, \mathsf{Q1d\_B}, \mathsf{Q1d\_Px}, \mathsf{Q1d\_Py}, \mathsf{Q1d\_Qx}, \mathsf{Q1d\_Qy})$ and the first output of PRG_2: $\mathsf{Q1d\_t_1} = \mathsf{PRG}_2(\mathsf{Q1d\_pp}, \mathsf{Q1d\_s_0}, 1)$, where $\mathsf{Q1d\_s_0}$ is hidden. You are also given $\mathsf{Q1d\_c}$ such that $\mathsf{Q1d\_P} = \mathsf{Q1d\_cQ1d\_Q}$.

▷ Compute $\mathsf{PRG}_2(\mathsf{Q1d\_pp}, \mathsf{Q1d\_s_0}, 5)$. i.e., use the trapdoor to predict the next 4 outputs of PRG_2, without knowing the initial seed. This should return a list of values

$$[\mathsf{Q1d\_t_2}, \mathsf{Q1d\_t_3}, \mathsf{Q1d\_t_4}, \mathsf{Q1d\_t_5}]$$

hint: how many points in $E(\mathbb{F}_p)$ have t_1 as their x-coordinate?

Question 1.5

Suggest a countermeasure to each of the above attacks. Upload your response as a seperate pdf or txt file.

Exercise 2 Signature Built on Unusual Assumption

The advent of large-scale quantum computers, which can efficiently solve the Discrete Logarithm problem using Shor's algorithm, poses a threat to classical cryptographic schemes such as Schnorr signatures and DSA. To counter this threat, cryptographers have proposed new schemes based on alternative mathematical tools like lattices, isogenies, and hash functions. These schemes are referred to as *post-quantum* because, so far, no efficient quantum algorithms are known for solving their underlying hard problems.

In this exercise, we study a signature scheme built on an unconventional mathematical problem that, unlike the Discrete Logarithm Problem, have not been found solvable efficiently by quantum algorithms.

Definition 1. Let p be a prime number and let $x \in \mathbb{Z}_p^*$ satisfy $\gcd(x, p-1) = 1$. Given p and $y = x^x \bmod p$, find x.

The signature scheme is described as follows, where λ is the security parameter and H is a hash function that maps a message m to an integer.

$\mathsf{KeyGen}(1^\lambda)$	$\mathsf{Sign}(\mathsf{sk}, \mathsf{m})$	$\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma)$
1 : Sample a prime p of λ bits	1 : parse $\mathsf{sk} \to (x, p)$	1 : parse $\mathsf{pk} \to (y, p)$
2 : **repeat**	2 : $h \leftarrow H(\mathsf{m})$	2 : parse $\sigma \to (r, s)$
3 : $\quad x \leftarrow\$ \mathbb{Z}_p^*$	3 : **repeat**	3 : $h \leftarrow H(\mathsf{m})$
4 : $\quad y \leftarrow x^x \bmod p$	4 : $\quad k \leftarrow\$ \mathbb{Z}_p^*$	4 : $a \leftarrow y^{r \times h} \times r^{s+r} \bmod p$
5 : **until** $\gcd(x, p-1) = 1 \wedge y \neq 1$	5 : **until** $\gcd(k, p-1) = 1$	5 : $b \leftarrow s^r \bmod p$
6 : $\mathsf{sk} \leftarrow (x, p)$	6 : $r \leftarrow x^{(-x \times h + k) \times x^{-k}} \bmod p$	6 : **if** $a = b$
7 : $\mathsf{pk} \leftarrow (y, p)$	7 : $n \leftarrow p \times (p-1)$	7 : $\quad$ **return** 1
8 : **return** $(\mathsf{sk}, \mathsf{pk})$	8 : $s \leftarrow r \times x^k \bmod n$	8 : **else**
	9 : **return** (r, s)	9 : $\quad$ **return** 0

Note that for a prime p and an integer x, $x^{x^{-1}} \bmod p$ is defined as $x^{x^{-1} \bmod (p-1)} \bmod p$.

In this exercise, we use the following hash function.

```
import hashlib
def H(m: str):
    return int.from_bytes(hashlib.sha256(m.encode()).digest(), 'big')
```

Question 2.1

▷ Implement the Sign algorithm and generate the signature $(\mathsf{Q2a\_r}, \mathsf{Q2a\_s})$ given the secret key $\mathsf{sk} = (\mathsf{Q2a\_x}, \mathsf{Q2a\_p})$ and message $\mathsf{Q2a\_m}$.

Hint: We also provide $\mathsf{Q2a\_y}$ *and encourage you to also implement the* Verify *algorithm to test the correctness of your answer.*

Question 2.2

▷ Given the public key $\mathsf{pk} = (\mathsf{Q2b\_y}, \mathsf{Q2b\_p})$ and message $\mathsf{Q2b\_m}$, forge a signature $(\mathsf{Q2b\_r}, \mathsf{Q2b\_s})$ of $\mathsf{Q2b\_m}$.

Hint: Let $n = p(p-1)$*. Find* $z \in \mathbb{Z}_n^*$ *such that* $z^z \bmod p = y$*.*