AN ATTACK ON RSA-PKCS #1 V1.5 (18 POINTS)

THE RSA-PKCS #1 SCHEME

The PKE (public key encryption) scheme consists of three algorithms **Gen**, **Enc** and **Dec**.

Key Generation

The key generation is the same as in the textbook version of RSA encryption.
The algorithm **Gen** on input 1^λ (the security parameter) samples two primes p, q of bit length λ uniformly at random.
Then it sets $n := p \cdot q$.
Then it samples $e \leftarrow \mathbb{Z}_n^* \setminus \{1\}$.

The public key is $\mathsf{pk} := (n, e)$ and the secret key is $\mathsf{sk} := (\phi(n), e)$ where ϕ is Euler's totient function of n and $e \cdot d = 1 \mod \phi(n)$.

$d = e^{-1} \mod \phi(n)$

Encryption

Let k be the bytelength of n and let $d \leq k - 11$.
Given a public key n and a message $m \in \{0,1\}^{8d}$ the algorithm **Enc** constructs the bitstring $m_{\mathsf{pad}} := 00||02||pad||00||m$ where 00 and 02 are bytes and $pad \in \{0,1\}^{8p}$ consists of $p := k - 3 - d$ non-zero bytes that are sampled uniformly at random.
Then **Enc** outputs the ciphertext $c := m_{\mathsf{pad}}^e \mod n$ where m_{pad} is interpreted as an

$c^d = m^{ed} = m$

integer in $\mathbb{Z}_n$.

Decryption

The algorithm Dec on input a ciphertext $c \in \mathbb{Z}_n$ computes $m'_{\mathsf{pad}} := c^d \mod n$.
Then it checks if m'_{pad} is of the form $00||02||pad||00||m'$ where pad consists of $p \in [8, k-3]$ non-zero bytes.
If so it outputs, m'.
Otherwise, it outputs $\perp$.

SECURITY NOTION FOR RSA-PKCS #1

We say a PKE is oneway secure under conformity leakage (OWCL secure), iff for any PPT adversary $\mathcal{A}$ the probability of winning the following OWCL game is negligble in the security parameter λ:
The challenger $\mathcal{C}$ generates a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and samples a message $m \leftarrow \{0,1\}^{8d}$ where d is as described above, and encrypts $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$.
Let $m' \leftarrow \mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda, pk, c)$ be the adversary's solution.
Here the adversary has access to the PCKS conformity oracle $\mathcal{O}$ which on input c returns true returns true iff c is PKCS conforming.
The adversary wins if $m = m'$.

Solve owcl using RSAPCK.

YOUR TASK

You will implement an attack that uses the fact that incorrect ciphertexts are decrypted to $\perp$, i.e., if the decrypted (padded) message m'_{pad} is not of the form $00||02||pad||00||m'$ subject to the conditions mentioned above — we call such ciphertexts *PKCS conforming*.\
Specifically, you will implement a successful OWCL adversary, namely the class `RSAPKCS_OWCL_Adversary`.
In particular, its `run` method takes as input an `RSAPKCS_OWCL_Challenger challenger` with which you can interact, e.g. get the scheme's public key `challenger.getPk()` or get the challenger's challenge `challenger.getChallenge()`.
Once you find the solution to the challenger's challenge you can simply return it in the `run` method.\
To help you win the OWCL game, the challenger provides an oracle in the form of the `challenger.isPKCSConforming(ciphertext)` method which returns true iff `ciphertext` is PKCS conforming.

NOTE ON THE NUMBER OF QUERIES

The challenger implicitly counts how many queries you make to the PKCS conformity oracle which can be read by the method `challenger.getQueryCounter()` (this may be useful for debugging purposes).
If the maximal number of queries `challenger.maxQueries` is exceeded, the method `challenger.isPKCSConforming(ciphertext)` will throw an exception
You don't necessarily have to catch exceptions as your test enviroment does this automatically, however a uncaught exception will count as a failed challenge.
Your adversary does not need to solve every single challenge while staying within the query limit.
In particular, it is natural to exhaust the query limit on some challenges (say < 20%); if your attack does so, you may still get full points for the exercise.
Nevertheless, we *may* award more point to attacks that are more efficient in general.

NOTE ON DEBUGGING

To be sure that your attack works without relying on the feedback from the testrunner, you can also check (by bruteforce over the padding string) whether a candidate solution corresponds to a given challenge ciphertext.
This code should **not** be contained in the submitted code but only locally for debugging/checking purposes.

TESTING YOUR IMPLEMENTATION

The recommended technical setup uses VSCode and Docker.

- Install both on your system according to the instructions on the respective website.
- Download the container.zip file and unzip it.
- Open VSCode, press F1 and select `Extensions: Install Extensions...` and install the extension `Dev Containers` by Microsoft.
- Press F1 and select `Remote Containers: Open Folder in Container...` and select the unzipped folder `{{task}}`.
- Click "Yes, I trust the authors" (optionally also check "Trust the authors of all files on the parent folder 'build'").
- You can run/debug you solution by clicking `Run` / `Debug` directly above the `main` method of the respective `*TestRunner.java` file.
  (It might take a while until the java extension initialization is completed.)

Alternatively, you can also install a JDK/JRE 17 and run the code locally or on an online service like codio.

Scores and Points

If your implementation is correct, it should succeed in each test case and get full points.

Important Note:
The tests which we run in Code Expert and on the TestRunner we gave you are only **preliminary**. After the submission deadline, we will run more exhaustive tests on your solution and review it manually.

Therefore, a solution which is only partially correct may receive full points in the preliminary tests but will get only partial points, eventually.
Therefore, make sure that your reductions are correct in the formally theoretic sense of cryptographic reductions!

Time and Memory Restrictions

The resources the TestRunner can use to test your solution are limited.
We expect your solution to use less than 1 minute of CPU time and a restriced space of memory when run several times.

Solutions which run into TimeOut- or OutOfMemoryExceptions will be rejected by us and receive 0 points.

Cheater Warning

The purpose of this task is to find plaintexts corresponding to a given RSA-PKCS ciphertext problem.

Any solution which tries to "trick" the testing environment is considered to be a cheating attempt and will receive zero points.