Homework 6

This problem set is new this year, and it's somewhat different from the problem sets you've had so far in this class. Please let your TAs know about any bugs, typos, or autograder issues, and also let us know if you have any feedback (positive or negative) about this assignment!

We've provided starter code for you to use; after reading this PDF, fill in the missing functions in `ps6_submission.py`, and turn in your code to the Gradescope assignment "Problem Set 6".

Firstly, you'll review how RSA works.

Then, you'll review why textbook RSA is insecure.

Finally, you'll implement an attack which targets a vulnerable implementation of RSA signature verification (this vulnerability will be explained later in this handout).

Problem 1 [10 points] Problem 1: Security of RSA.

Recall that, in our key generation phase of RSA, we generate N, e and d, where (N, e) becomes our public key and (N, d) becomes our private key.

1. **[5 points]** Suppose everyone uses the same e during key generation, but randomly generates a different N every time. Is this secure? Explain your answer in no more than two sentences.

2. **[5 points]** Suppose, instead, that everyone uses the same N during key generation, but randomly generates a different e every time. Is this secure? Explain your answer in no more than two sentences.

Problem 2 [30 points] Problem 2: Attacking Textbook RSA.

Let $\mathcal{K}_{\text{rsa}}$ be a RSA generator with security parameter $k \geq 32$.

Consider the digital signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ whose component algorithms are below, and whose message space is Z_N^*:

Alg $\mathcal{K}$
$(N, p, q, e, d) \xleftarrow{\$} \mathcal{K}_{\text{rsa}}$; Return $((N, e), (N, d))$

Alg $\mathcal{S}((N, d), M)$
$s \leftarrow M^d \bmod N$; Return s

Alg $\mathcal{V}((N, e), M, s)$

If $(s^e \bmod N = M)$ then return 1 else return 0

Present in pseudocode a $\mathcal{O}(1)$-time adversary A making **no** queries to its **Sign** oracle and achieving $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS}}(A) = 1$.

Problem 3 [50 points] Problem 3: Bleichenbacher's Attack on low public exponents.

In this problem, you are given a public key (named in `ps6_submission.py` as `publickey`) for RSA signing (this time, with padding).

You are told that function used for verifying signatures is vulnerable to Daniel Bleichenbacher's low public exponent attack. The function will validate the signature, without checking the length of the FF bytes, and without checking that the hash is in the least significant bits of the message. (you can see this flawed verification algorithm inside `ps6_submission.py`, in the function named `sus_verify`).

Your goal is to implement this attack in the function `bleichenbacher`, in the starter code. Your function should take in a message m to forge, and it should return some s such that (m, s) verifies successfully under the public key `publickey`.
