In this document, we give a few hints and intuitions for you to solve this challenge. Note that what is written in this document is a strict subset of the requirements needed to make this attack work. You should work out the details by yourself.

Main idea. Our objective is to recover the plaintext by making use of decryption errors. As seen in the previous challenge, you are able to recover an upper and a lower bound for the padded message. We want to improve these bounds iteratively by reducing the size of the interval at each iteration via a binary search. This means that, if the RSA modulus is k bits in length, an optimal attack will take $\mathcal{O}(k)$ steps to recover a plaintext.

Notation. Let (e, N) be a public RSA key and (d, N) the corresponding private key. The modulus N has a size of k bits. In the challenge $k = 1024$. Let $B = 2^{k-8}$. Let m be the message after the padding scheme has been applied and let $c = m^e \mod N$. Let M be the big-endian representation of m.

The oracle. We now formalise the oracle that you are given. The oracle is given a ciphertext c and decrypts it, obtaining $m = c^d \mod N$ and checks whether the first byte of M is `0x00`. Since the bit-length of N is k, this implies that you can distinguish whether a message m has $m \geq B \mod N$ or $m < B \mod N$. You should be familiar with this oracle, since you should have used it to solve the previous challenge: we are merely formalising it.

Step 1: Initial bounds. From the previous challenge, you should have a bound for m of the form $m_{\min} = 2^{i-1} \leq m < 2^i = m_{\max}$. You can use this as an initial bound.
If needed, restructure your code in order to have an `oracle` function matching the above description that you can use for the next steps.

Step 2: Improving the bounds. Our objective is to find a value α_0 such that $\alpha_0 \cdot m$ is "just above" N. In other words, $\alpha_0 \cdot m$ incurs a *single additional* modular reduction with respect to just having m and α_0 is minimal. First, you should think about how you can use the oracle to detect that a modular reduction has happened. Note that there are multiple possible values of α_0 which would be suitable (α_0 need not be minimal, but somewhat close to being so). A sufficient (but not necessary) condition to find a valid α_0 is that $\alpha_0 \cdot m$ incurs a modular reduction, but $(\alpha_0 - 1)m$ does not.

You can now derive a new interval $[m_{\min}, m_{\max})$, improving the bound for m. (*Hint:* You already know that $\alpha_0 \cdot m > N$. This gives you the new lower bound for m. Since only a *single* modular reduction has happened and α_0 is minimal, can you also find an upper bound that depends on B?)

B-points and Better Bounds. We call an integer of the form $B + rN$ (for some $r \in \mathbb{Z}$) a *B-point*. Assume that you are given bounds for some value $x \in [x_{\min}, x_{\max})$ and the corresponding RSA ciphertext $c = x^e \mod N$. Further assume that there exists a *unique* B-point inside the bounds, i.e.

$$\exists!\, r \in \mathbb{Z}, rN \leq x_{\min} < B + rN \leq x_{\max} \leq (r+1)N \tag{1}$$

Note that this equation is over $\mathbb{Z}$. Think about how you can use the oracle to obtain a better bound for x. To start, imagine that $r = 0$: this implies $x_{\min} < B \leq x_{\max}$, you could then query the oracle

on the value c to immediately improve the bound. Then, convince yourself that this also works when $r \neq 0$.

Step 3: Recovering the Message. Your interval $[m_{\min}, m_{\max})$ from **Step 2** however, is not of the form above: it does not contain a B-point. Let $\alpha \in \mathbb{N}$ and define a bijection $f_\alpha : m \mapsto \alpha \cdot m$ that maps every possible $m \in [m_{\min}, m_{\max})$ to a new $m' = \alpha \cdot m \in [\alpha \cdot m_{\min}, \alpha \cdot m_{\max})$. We want to find α such that this latter interval respects condition (1). Assume that you found such an α, then all the considerations made earlier will apply to the interval $[\alpha \cdot m_{\min}, \alpha \cdot m_{\max})$, allowing you to improve your bound for $\alpha \cdot m$. Think about how this helps you find an improved bound for m. Finally, think about how to repeat this process in a binary search fashion until $m_{\min} = m_{\max} - 1$.

In Listing 1, we provide an algorithm that computes α from $m_{\min}, m_{\max}$, B, and N. The `get_multiplier` method will try to return an α for which condition (1) holds for the interval $[\alpha \cdot m_{\min}, \alpha \cdot m_{\max})$. It may fail to give you a valid α if in **Step 2** you did not get a good enough bound. You may copy and paste this code in your solution.

```
    def ceil(a: int, b: int) -> int:
        # Necessary because of floating point precision loss
        return a // b + (1 if a % b != 0 else 0)

    def get_multiplier(m_max: int, m_min: int, N: int, B: int) -> int:
        tmp = ceil(2 * B, m_max - m_min)
        r = tmp * m_min // N
        alpha = ceil(r * N, m_min)
        return alpha
```

Listing 1: Compute α, given $m_{\max}$ (`m_max`), $m_{\min}$ (`m_min`), the RSA modulus N and the value B. The algorithm first tries to find a value α_{tmp} that satisfies $\alpha_{\text{tmp}} \cdot (m_{\max} - m_{\min}) \approx 2B$ (this helps in eliminating half of the values for each iteration of the binary search). This value is then used to find the value of r. Finally, in the condition $\alpha \cdot m_{\min} > rN$, we take the smallest possible value for α. This ensures that only one B-point will be within our mapped range.

Final notes. Depending on your solution, your script may not terminate correctly with probability 1. Since the amount of queries to the server is relatively small, you may wish to implement your solution as a Las Vegas algorithm.

Overall, the lesson to be learnt from this challenge is: *don't roll your own RSA padding.* In fact, you've seen that even a randomized padding scheme may not be sufficient to provide IND-CCA security, if the developer is careless!

Good luck!