Instructions

- You are allowed to work in groups of size at most 2.
- The assignments must be typed in Latex, and the resulting pdf must be submitted on Gradescope.
- **Plagiarism policy**: You should not discuss your solutions with other group members. Sharing your solutions with other group members is strictly not allowed, and if there are significant similarities in two or more submissions, all relevant group members will be penalized.

  You can refer to resources online, but you should write your solutions in your own words (and also cite the resources used).

Notations

For any positive integer t, $\mathbb{Z}_t$ denotes the set $\{0, 1, \ldots, t-1\}$ and $\mathbb{Z}_t^*$ denotes the set of all integers in $\mathbb{Z}_t$ that are co-prime to t.

Questions

1. (15 marks) **A CRHF construction**

   Let $N = p \cdot q$ be the RSA modulus and e is a random prime in $\mathbb{Z}_{\phi(N)}$ that is co-prime to $\phi(N)$. Consider the following hash function family. The key is N, e, and a random integer $z \leftarrow \mathbb{Z}_N^*$. The hash function $H_{N,e,z} : \mathbb{Z}_N^* \times \mathbb{Z}_e \to \mathbb{Z}_N^*$, where $H_{N,e,z}(x, y) = x^e \cdot z^y \bmod N$. Show that this is a secure collision-resistant hash function, assuming RSA is secure.[1]

   Note: Wherever you compute an inverse, specify why the inverse can be computed efficiently.

2. (10 marks) **Use of pre-challenge decryption queries**

   Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a CCA-secure public key encryption scheme. Construct an encryption scheme $\mathcal{E}'$ that is secure **if no pre-challenge decryption queries** are made, but is not CCA-secure. More formally, consider the security game (CCA-no-pre) defined in Figure 1.

CCA-no-pre

- (Setup phase) Challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ and sends pk to $\mathcal{A}$.
- (Challenge phase) Adversary sends two challenge messages m_0, m_1. Challenger chooses bit $b \leftarrow \{0, 1\}$, and sends $\mathsf{ct}^* \leftarrow \mathsf{Enc}(m_b, \mathsf{pk})$.
- (Post-challenge decryption queries) Adversary sends decryption queries. For every decryption query $\mathsf{ct}_i \neq \mathsf{ct}^*$, challenger computes $y_i = \mathsf{Dec}(\mathsf{ct}_i, \mathsf{sk})$ and sends y_i to $\mathcal{A}$.
- (Guess) After polynomially many decryption queries, adversary sends its guess b' and wins if $b = b'$.

Figure 1: CCA security with no pre-challenge queries

Definition 1. An encryption scheme is CCA-no-pre secure if no p.p.t. adversary can win the CCA-no-pre security game with non-negligible advantage.

Use $\mathcal{E}$ to construct an encryption scheme $\mathcal{E}'$ that is CCA-no-pre secure, but not CCA secure. Clearly describe the CCA attack, and give a formal proof of security for showing that $\mathcal{E}'$ is CCA-no-pre secure.

[1] For this problem, you can use the RSA variant where e is a random prime in $Z_{\phi(N)}$ that is co-prime to $\phi(N)$.

The above scheme will be a very contrived scheme. However, there are certain encryption schemes where even one decryption query reveals the secret key. Such schemes are semantically secure, but they don't even satisfy CCA-1 security. You can read about Rabin's encryption scheme, an early encryption scheme that was semantically secure, but even one decryption query reveals the entire secret key.

3. (5 marks) **An assignment/quiz question on signatures**

   Imagine you are the instructor of COL759, and your students want to test their understanding of the unforgeability definition(s).

   Design a signature scheme that is not secure. You can use any building blocks. Try to keep the scheme relatively simple (in description). However, the attack should not be a 'trivial attack' (be creative!). [2]

4. (20 marks) **Coding Question: Attack on RSA PKCS Padding: Bleichenbacher's attack**

   This is a Chosen Ciphertext Attack against Protocols based on the RSA Encyption Standard PKCS#1 v1.5

   When encrypting something with RSA, using PKCS#1 v1.5, the data that is to be encrypted is first padded, then the padded value is converted into an integer, which is then followed by RSA modular exponentiation (with the public exponent e). On decryption, the modular exponentiation of the ciphertext (with the private exponent d) is applied and the padding is removed to recover the message.

   The core of this attack relies on a padding oracle: given a sequence of bytes of the same length as the ciphertext, tells whether the decrypted message would have a proper padding or not.

   Padding Scheme: Let N be the public modulus for the RSA and e be the public exponent. Let M be the sequence of bytes which is to be encrypted. The PKCS#1 v1.5 padding scheme adds some bytes to the left, so that the total length of the padded message is equal to that of N. Let k be the byte length of N.

   A properly padded message M has the following format:

   $$\texttt{0x00 0x02 [PS] 0x00 } [M]$$

   1. The sequence of bytes begins with a zero byte, which is followed by a byte of value 2
   2. `PS`: The padding string, which is a sequence of random bytes, which cannot have a zero byte
   3. Then a byte of value 0, followed by the message M itself
   4. Restrictions:
      (a) The padding string `PS`, has $k - 3 - |M|$ bytes
      (b) $|M|$ cannot exceed $k - 11$ bytes, which means that the byte length of `PS` is at least 8
      (c) The encryption block $00 \,||\, 02 \,||\, \texttt{PS} \,||\, 00 \,||\, M$ is formed, converted to an integer x and encrypted with RSA, giving the ciphertext $c = x^e \pmod N$

   Attack: The attacker has access to the encrypted message or the ciphertext c. To decrypt the message, the attacker makes use of the padding oracle as follows:

   - The adversary knows that $c = m^e \bmod n$, where e is the public exponent, and m is the padded message.
   - The attacker will initiate many requests to the oracle.

[2] This is meant to be a 'fun' question, and I will assign full marks for any reasonable solution here.

- For each request, the attacker generates a value s and sends, a value $c' = c \cdot s^e \bmod n$.
- Most of the times, the padding oracle will return a 'bad padding' error. However, with a low but non-negligible probability, it will not return a padding error. Using this information from the oracle and the value of s, attacker can narrow down the search for m in a specific range.
- Refer to the paper by Bleichenbacher to learn about the attack. There are also several expository articles on Bleichenbacher's attack. If you refer to any of them, please cite them appropriately in your submission.

Files given: Since the messages and the cipher-texts are arbitrary sequences of bytes, we represent each of them by a list of integers within the range $[0,\ 255]$. You are provided with the following files on Moodle (`A5_code.zip`):

- `rsa.py`:
  - It contains a function called `check_padding`(c) : Takes the ciphertext and decrypts it to get the padded message. It checks whether it has a valid padding i.e, it should have the first byte as 1, the second byte as 2 which is followed by a sequence of $(k - 3 - |M|)$ non-zero bytes and finally 0. If any of these is violated, outputs `False`, and outputs `True` otherwise.
  - It has a function called `encryption(msg)`, which takes the unpadded messaged (list of integers) as input, internally pads it and returns the ciphertext. You can use it to check the correctness of your code.
- `attack.py`:
  - You are required to implement your attack in this file. The function to be implemented is `attack(cipher_text, N, e)`
  - It is supposed to take a ciphertext, the public modulus and the public exponent as input, and the return the original message (as a list of integers)
  - You are allowed to make calls to `check_padding()` from `rsa.py`

Instructions:

- You would need to install the `pycryptodome` python package to run the given files. Installation instructions can be found on this link
- You are only required to submit `attack.py`, with your implementation of `attack()`. You don't need to submit `rsa.py`
- Submit the `attack.py` file on Gradescope in the `Assignment 5 Code` assignment
- During grading, we would change the keyPair so you cannot simply decrypt the ciphertext