Project 3:
All Things Cryptography

Project Files and PDF/docx Templates:

You can download a zip file with the starter files here.

Goals of the Project:

- Understand and advance their knowledge of cryptography and hashing
- Understanding how these are implemented by working through examples
- Understanding how and why the exploit could be completed on vulnerable systems

Information:

Before starting, make **SURE** you are using Python VERSION 3.7.x OR LOWER. Version 3.8 includes some functionality that may not be compatible with the autograder environment, which runs Python Version 3.6.7. To check your version of Python, open a command prompt and run the command:

```
python --version
```

For the established algorithms that you may need to use, you are allowed to reference and implement pseudocode with **PROPER CITATION.** What is Pseudocode? https://en.wikipedia.org/wiki/Pseudocode

UNDER NO CIRCUMSTANCES should you copy/paste code into the project. Doing so is an honor code violation (not to mention a real world security concern) and will result in a zero (Refer to syllabus for more information).

The final deliverables:

You will submit `project_3.pdf` in 2 places.

1. **Project 3 - Essay** on Gradescope**. NOTE: You must use page mapping or lose points**
2. **Project 3: Cryptography** on Canvas**.**

Submission name format is `project_3.pdf`

You will submit `project_3.py` in 2 places.

1. **Project 3 - Autograder** on Gradescope**.**
2. **Project 3 .py file: Auto-Graded Portion** on Canvas**.**

Submission name format is `project_3.py`

Notes:

- **If you do not submit to both (Canvas and Gradescope), you will receive a 0. This is non-negotiable and will be enforced heavily.**
- **When submitting your essay in Gradescope, you will be asked to tag questions to page numbers in your report. You must do this accurately, or else a 5 point deduction will be applied.**
- Your written report **must** be submitted in the Joyner Document Format (JDF). A template has been provided for you in Microsoft Word format, but you may find further useful resources here.
- You **MUST** provide citations in JDF format which uses APA style; weblinks alone do **NOT** count. Refer to sections “3.1 In-line citations”, “3.2 Reference lists”, and “4 References”.
- We know Canvas adds -x (x being a number) to the end of your resubmissions. **That is OK.**

Plagiarism will not be tolerated! For information: GT Academic Honor Code and the Syllabus.

Intro :

RSA is one of the most widely-used public key cryptosystems in the world. It's composed of three algorithms: key generation (Gen), encryption (Enc), and decryption (Dec). In RSA, the public key is a pair of integers (e, N), and the private key is an integer d.

The key pair is generated by the following steps:

1. Choose two distinct big prime numbers with the same bit size, say p and q.

2. Let $N = p * q$, and $\varphi(N) = (p - 1) * (q - 1)$.

3. Pick up an integer e, such that $1 < e < \varphi(N)$ and $gcd(e, \varphi(N)) = 1$.

4. Get the modular inverse of e: $d \equiv e^{-1} \, mod \, \varphi(N)$ $(i.e., \, d * e \equiv 1 \, mod \, \varphi(N))$.

5. Return (N, e) as public key, and d as private key.

Enc - To encrypt integer m with public key (N, e), the cipher integer $c \equiv m^{e} \, mod \, N$.
Dec - To decrypt cipher integer c with private key d, the plain integer $m \equiv c^{d} \, mod \, N$.

Task 1 – Warm-up, Get Familiar with RSA - (5 points)

The goal of this task is to get you familiar with RSA. You are given an RSA key pair $(N, e)$and d, and a unique encrypted message c. You are required to get the decrypted message$m$.

TODO: In the provided `project_3.py` file, implement the stub method `task_1`. **Hint:** Don't overthink it, this can be done with a single Python command...

```
def task_1(self, n_str: str, d_str: str, c_str: str) -> str:
    # TODO: Implement this method for Task 1
    n = int(n_str, 16)
    d = int(d_str, 16)
    c = int(c_str, 16)
    m = 0

    return hex(m).rstrip('L')
```

Task 2 – Warm-up, Get Familiar with Hashes (7 points)

By now we've learned that hashes are one-way functions. Because of this unique feature, passwords are often stored as hashes in order to protect them from prying eyes. Even if a hacker infiltrated our state-of-the-art Georgia Tech security systems, he or she would not be able to derive the plaintext passwords from the hashes. But what if we made the critical mistake of using a common password? **How safe would we be?**

Let's find out…

You are given a list of some of the most commonly-used passwords on the Internet. You are also given the **SHA256** hash of a password randomly selected from this list. Your job is to discover the plaintext password behind the hash.

The complete list of common passwords is pre-loaded for you in `project_3.py`.

TODO: In the provided `project_3.py` file, implement the stub method `task_2`.

```
def task_2(self, password_hash: str) -> str:
    # TODO: Implement this method for Task 2

    password = common_password_list[0]
    # This is how you get the SHA-256 hash:
    hashed_password = hashlib.sha256(password.encode()).hexdigest()

    return password
```

Reflection

In a maximum of 200 words, address the following prompt:

- Someone suggests that you add a salt value to passwords before hashing them as a means to keep out malicious actors. Would this effectively protect a weak password? If yes, explain why, if no, how could this scheme be defeated? Make one additional suggestion for how you could implement improved password security.

Task 3 – Kernelcoin Part 1 (9 points)

Background: A blockchain is a distributed, immutable ledger that derives its security, in part, from a chain of cryptographic hash values. For more detail, please read Section II of Hassan et al., Blockchain and the Future of the Internet: A Comprehensive Review, arXiv:1904.00733v1 (23 Feb. 2019), available online at: https://arxiv.org/pdf/1904.00733.pdf.

Today is your lucky day! You've discovered a brand new cryptocurrency called Kernelcoin (symbol: RTI). There are rumors that Costco will soon announce Kernelcoin as the preferred payment method in its warehouse stores. This news is sure to send the price of Kernelcoin to the moon, and Kernelcoin holders to the nearest Lamborghini dealership.

You plan to start mining Kernelcoin so that you can earn even more. In order to do so, you need to create a valid block to append to the previous block. A valid block contains the lowest nonce value that, when concatenated with the transaction string, and the hash of the previous block (in that order, i.e. nonce + transaction string + previous block hash), will produce a SHA256 hash with two leading zeros (the proof-of-work for this particular blockchain). Transaction strings have the syntax "UserID1:UserID2:X", indicating that UserID1has transferred X Kernelcoin to UserID2. You are given all of these values, and your goal is to find the lowest possible nonce value for the resulting block.

TODO: In the provided `project_3.py` file, implement the method `task_3`.

```
def task_3(self, user_id_1: str, user_id_2: str, amount: int, prev_block_hash:
str) -> int:
    # TODO: Implement this method for Task 3
    nonce = 0

    return nonce
```

Reflection

In a maximum of 200 words, address the following prompt:

The kernelcoin blockchain uses a proof-of-work scheme as a consensus mechanism (i.e., finding a hash with a certain number of leading zeros).

- Name and briefly explain an alternative consensus mechanism.
- List its strengths and weaknesses compared to proof-of-work.

Task 4 – Kernelcoin Part 2 (9 points)

Sure enough, once /r/WallStreetBets found out about Kernelcoin the price rose to nosebleed levels. The Kernelcoin that you mined is now worth a fortune! Feeling generous, you decide to donate a small portion of your gains to Georgia Tech so that the school can give its TAs a much-deserved raise. As you prepare to send the transaction, you start to wonder how Kernelcoin verifies that transactions are valid...

After doing some research you find that a Kernelcoin transaction is hashed and encrypted with your private key to create a digital signature. This signature is broadcast to the network along with the original transaction string. If the signature checks out, then the transaction is a candidate for inclusion in the next block.

TODO: In the provided `project_3.py` file, finish the code for signing a Kernelcoin transaction in the method `task_4`. (You may find the code that you wrote in Task 1 helpful for this.)

```
def task_4(self, from_user_id: str, to_user_id: str, amount: int, d: int, e: int,
n: int) -> int:
    # TODO: Implement this method for Task 4

    return signature
```

Reflection

In a maximum of 200 words, address the following prompt:

Imagine that you are coding a function that accepts a Kernelcoin transaction string and a digital signature. The public key of the signer is also passed to the function. The purpose of the function is to verify the validity of the transaction (i.e. it returns a boolean value).

- Explain the high-level steps necessary to implement this function. No code is required. You should use your own words.

Task 5 – Attack A Small Key Space (15 points)

Modern day RSA keys are sufficiently large that it is impossible for attackers to traverse the entire key space with limited resources. But in this task, you're given a unique RSA public key with a relatively small key size (**64 bits**).

Your goal is to get the private key.

TODO: In the provided `project_3.py` file, implement the method `get_factors`. n is the given public key, and your goal is to get its factors.

```
def get_factors(self, n: int):
    # TODO: Implement this method for Task 5, Step 1
    p = 0
    q = 0

    return p, q
```

TODO: In the provided `project_3.py` file, implement the method `get_private_key_from_p_q_e` to get the private key.

```
def get_private_key_from_p_q_e(self, p: int, q: int, e: int):
    # TODO: Implement this method for Task 5, Step 2
    d = 0

    return d
```

Reflection

In a maximum of 500 words, address the following prompts:

Explain in your own words how you were able to get the private key.

- What were the steps you followed?
- What was the underlying mathematical principle?

Task 6 – Where's Waldo (25 Points)

Read the paper "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices", which can be found at: https://factorable.net/weakkeys12.extended.pdf. **You will not be able to understand the purpose of this task nor write about it properly in your essay unless you read the entire paper**. Do not skip it, do not skim it, read the whole of it.

You are given a unique RSA public key, but the RNG (random number generator) used in the key generation suffers from a vulnerability described in the paper above. In addition, you are given a list of public keys that were generated by the same RNG on the same system. Your goal is to get the unique private key from your given public key using only the provided information.

TODO: In the provided `project_3.py` file, implement the method `task_6`. (More information about Waldo, and why everyone keeps looking for him can be found here: https://en.wikipedia.org/wiki/Where%27s_Wally%3F. Knowledge of "Where's Waldo?" isn't strictly necessary to solve this task, but it might give you a nudge in the right direction...)

```
def task_6(self,
           given_public_key_n: int,
           given_public_key_e: int,
           public_key_list: list) -> int:
    # TODO: Implement this method for Task 6
    d = 0

    return d
```

Reflection

In a maximum of 500 words, address the following prompts:

- Why is the public key used in this task vulnerable? Explain this in your own words. Please talk about the potential problems with the key generation and the associated mathematical principles in your answer.

- What steps did you take to derive the private key result in this task? Please discuss the underlying mathematical principles at a high level and explain how you arrived at your answer.

Task 7 – Broadcast RSA Attack (30 Points)

A message was encrypted with three different 1,024-bit RSA public keys, resulting in three different encrypted messages. All of them have the public exponent $e = 3$.

You are given the three pairs of public keys and associated encrypted messages. Your job is to recover the original message.

TODO: In the provided `project_3.py` file, implement the method `task_7`.

```
def task_7(self,
           n_1_str: str, c_1_str: str,
           n_2_str: str, c_2_str: str,
           n_3_str: str, c_3_str: str) -> str:
    n_1 = int(n_1_str, 16)
    c_1 = int(c_1_str, 16)
    n_2 = int(n_2_str, 16)
    c_2 = int(c_2_str, 16)
    n_3 = int(n_3_str, 16)
    c_3 = int(c_3_str, 16)

    msg = ''
    m = 0

    # Solve for m, which is an integer value,
    # the line below will convert it to a string
    msg = bytes.fromhex(hex(m).rstrip('L')[2:]).decode('UTF-8')

    return msg
```

Reflection

In a maximum of 500 words, address the following prompts:

- How does the broadcast RSA attack work?
- What causes the vulnerability?
- Explain this in your own words and explain at a high level the mathematical principles behind it.
- Explain how you recovered the message, ensuring that you give thorough detail on *all* of your steps.

Important Notes :

The skeleton code in the `project_3.py` file has all of the packages that you will need imported for you. You are **NOT** allowed to import anything else.

Your entire submission must run in 10 minutes or less. The autograder will not give you any feedback if it times out. We encourage you to test locally to avoid unnecessarily using submissions.

Your code will run in an autograded environment in Gradescope and give you immediate feedback and a grade. **However, you are limited to 10 autograder submissions in Gradescope. After 10 submissions, penalties will be assessed as follows:**

10 < # submissions <=20	**-10**
20 < # submissions <=30	**-20**
30 < # submissions <=40	**-30**
40 < # submissions <=50	**-40**
50 < # submissions <=60	**-50**
# submissions > 60	**-55**

You will be able to keep the score of your highest run but you must activate it manually.

Gradescope can get very busy and even potentially unavailable near submission deadlines. **Please do not wait until the last minute to make your submissions to the autograder.** Try to get them in as early as possible. **Late deductions due to Gradescope being busy for last minute submissions will not be removed.**

You are also given a unit test file (`test_project_3.py`) to help you test your program. We encourage you to read up on Python unit tests, but in general, the syntax should resemble either:

```
python -m unittest test_project_3
```

or:

```
python test_project_3.py
```

However, keep in mind that passing the unit test does NOT guarantee that your code will pass the autograder!

Good luck!