Project 3 :

All Things Cryptography

Fall 2020

The goals of this project :

Students will advance their knowledge of cryptography and hashing by working through example exercises and then trying to exploit vulnerable systems.

Preface :

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 3.6.9. 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 IMPLEMENT PSEUDOCODE WITH PROPER CITATION. What is Pseudocode? --> https://en.wikipedia.org/wiki/Pseudocode

HOWEVER, 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).

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):
    # 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 (10 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):
    # 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:

Knowing that a lot of people like to use these common passwords, make one suggestion for how you could implement improved password security.

Task 3 – Kernelcoin (15 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 their 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):
    # 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 an alternative consensus mechanism and list its strengths and weaknesses compared to proof-of-work.

Task 4 – Attack A Small Key Space (15 points)

The algorithm you search for is dirt simple which makes it hard for attackers to traverse the entire key space with limited resources. Now, 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 4, 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 4, 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 in order to get it and what was the underlying mathematical principle used?

Task 5 – 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_5`. (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_5(self,
           given_public_key_n: int,
           given_public_key_e: int,
           public_key_list: list):
    # TODO: Implement this method for Task 5
    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 6 – 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_6`.

```
def task_6(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):
    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.

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.

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!

The final deliverables:

In total, please submit the following files:

1. `project_3.py`
2. `project_3_report.pdf` : An essay with all of your answers to the reflection questions.

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: https://drive.google.com/drive/folders/1xDYIomn9e9FxbIeFcscISbXHTtHROD1j?usp=sharing.

Direct quotes from source material may comprise a **maximum** of 10% of your essay - you should summarize concepts and put them in your own words (with proper citation of course!).

You will need to submit your essay to both Canvas and Gradescope!

Good luck!