Ira A. Fulton Schools of Engineering
Arizona State University

CSE 535: Mobile Computing

Mobile Security (PQC) Assignment

Purpose

Security is a vital component of any mobile application that handles sensitive data of any kind, such as personal information, health information, and financial data. Typically, security is offered through the use of an existing protocol or library, such as using HTTPS for data in motion or encrypting data at rest with an algorithm such as AES. However, there is value in understanding how these algorithms are implemented, their requirements, and their weaknesses. This assignment will build on the cryptography lecture to introduce a relatively modern cryptographic scheme, Learning with Errors (LWE). By the end of the assignment, you will write Python code that implements the LWE algorithm, including public key generation, encryption, and decryption.

Objectives

Learners will be able to:

- Analyze security requirements of smart mobile applications.
- Analyze the inter-relationship between safety and security of mobile applications.
- Apply popular tools, such as machine learning, security protocols, AI, and software testing, to validate safety and security of smart mobile applications.
- Identify mobile network security threats, issues, and challenges in various domains.
- Discuss advances in mobile security technologies.

Technology Requirements

- Python 3.10.6
- NumPy 2.2.2

Description

Quantum computing poses a significant future thread to the security of data. Quantum algorithms, such as Shor's factoring algorithm mean that security schemes built on the factoring and discrete

logarithm problems will become tractable. This includes many popular cryptographic algorithms, such as AES. Despite the fact that quantum computers are not sufficiently advanced to run such algorithms at scale yet, the threat of harvest now, decrypt later attacks means that malicious parties can hold the encrypted data until that point.

The National Institute of Standards and Technology (NIST) has spearheaded attempts to create new cryptographic algorithms, in a field known as Post-Quantum Cryptography (PQC). These are algorithms which are believed to be intractable, even on a quantum device. Some companies have already implemented PQC into their security systems, such as Zoom who employ Kyber-768 (https://news.zoom.us/post-quantum-e2ee/). Another such algorithm is the Learning with Errors (LWE) algorithm, which is a post-quantum cryptographic scheme that can be implemented on a classical device.

Directions

First, the LWE algorithm will be described. Each subpoint will have an example of that step, resulting in a complete example by the end.

Public Key Generation

1. Generate m vectors a_i randomly from $\mathbb{Z}_p^n$ where p is the modulus. These vectors are typically considered as rows of a matrix A, but the computation can be done in either way.
    a. $m = 3, n = 4, p = 7$
    b. $a_1 = [1, 6, 6, 2]$, $a_2 = [6, 0, 5, 3]$, $a_3 = [2, 5, 4, 1]$
2. Generate an error vector e of length m.
    a. $e = [0, -1, 1]$
3. Generate a random secret key s of length n
    a. $s = [3, 4, 0, 6]$
4. Calculate $b = A \cdot s + e \bmod p$
    a. a_1: $1 * 3 + 6 * 4 + 6 * 0 + 2 * 6 + 0 \bmod 7 = 39 \bmod 7 = 4$
    b. $b = [4, 0, 5]$
5. Public key = (A, b)

Encryption

1. Randomly choose two (2) rows of the public key (A and b). Sum up the corresponding entries in A's rows.
    a. Suppose random choice is [1, 0, 1] (i.e., the first and third rows). This corresponds to $[1, 6, 6, 2], 5$ and $[2, 5, 4, 1], 5$.
    b. $[1, 6, 6, 2] + [2, 5, 4, 1] \; mod \; 7 = [3, 4, 3, 3]$
2. We now need to sum up the corresponding entries in b. This approach will allow encryption of a single bit of information. If the bit to be encrypted is 0, no further addition is needed. If the bit to be encrypted is 1, $\lfloor p/2 \rfloor$ needs to also be added to the b summation.
    a. If message = 1, $4 + 5 + \lfloor 7/2 \rfloor \; mod \; 7 = 9 + 3 \; mod \; 7 = 5$
    b. If message = 0, $4 + 5 \; mod \; 7 = 2$
3. The resulting encryption is the pair of the A summation and b summation. Note that in this example, the random choice is assumed to be consistent, but it doesn't need to be.
    a. $Enc_0 = ([3, 4, 3, 3], 2)$
    b. $Enc_1 = ([3, 4, 3, 3], 5)$

Decryption

1. To decrypt, we take the dot product of the first item in the ciphertext and the secret key s, mod p.
    a. $[3, 4, 3, 3] \cdot [3, 4, 0, 6] \; mod \; 7 = 3 * 3 + 4 * 4 + 3 * 0 + 3 * 6 \; mod \; 7 = 43 \; mod \; 7 = 1$
2. Subtract this value from the second item in the ciphertext, mod p. If it's closer to 0, the message was 0. If it's closer to $\lfloor p/2 \rfloor$, the message was 1. Note that we won't get exact message values from the computation and need to do the comparison due to the use of randomness in this algorithm.
    a. Enc_0: $(2 - 1) \; mod \; 7 = 1$ (since 1 is closer to 0 than 3, the message is 0)
    b. Enc_1: $(5 - 1) \; mod \; 7 = 4$ (since 4 is closer to 3 than 0, the message is 1)

Implementation Steps

1. All methods should be defined within a class named A2 (and therefore take as input self as an additional first parameter. If an instruction requests two parameters, you would have self and then those two. Please name your file a2.py.
2. Define a method generate_public_key_vector that takes as input four values, the matrix A, the secret key s, the error vector e, and the modulus p. This method should compute the vector b and return it as a NumPy array.
    a. Note that you will not need to randomly generate values in this assignment, to facilitate testing, whereas in a real-world version you would randomly generate the values.
3. Define a method encrypt that takes as input four values, the public key, a random binary vector, a message, and the modulus p. This method should encrypt the input message and return the encryption tuple.
    a. Hint: you can return a tuple in Python as a set of comma-separated values. For example, if I wanted to return the tuple containing values 1 and 2, I could write: return 1, 2
    b. The random binary vector is used to index the public key.
4. Define a method decrypt that takes as input three values, the private key, the ciphertext, and the modulus p. This method should decrypt the ciphertext and return the message.

Submission Directions for Deliverables

You must submit your Mobile Security (PQC) Assignment deliverable through Gradescope. Carefully review submission directions outlined in this overview document in order to correctly earn credit for your work. Learners may not email or use other means to submit any assignment or project for review, including feedback, and grading.

The Mobile Security (PQC) Assignment includes one (1) deliverable:

- **a2.py:** Source code name must match exactly. Ensure you match the class and method naming conventions described in the directions.

Submitting to Gradescope

After completing work in GitHub, you must submit your work into Gradescope to receive credit for the course:

1. Go to the Canvas Assignment, "**Submission: Mobile Security (PQC) Assignment**".

2. Click the "**Load Submission…in new window**" button.
3. Once in Gradescope, select the assignment titled "**Mobile Security (PQC) Assignment**" and a pop-up window will appear.
4. In the pop-up:
    a. Submit your Python file "**a2.py**".
    b. Click "**Upload**" to submit your work for grading.
5. If needed: to resubmit the assignment in Gradescope:
    a. Return to the Canvas submission and open Gradescope.
    b. You will be navigated to the "Autograder Results" page (if it is not your first submission).
    c. Click the "**Resubmit**" button on the bottom right corner of the page and repeat the process from Step 3.

Your submission will be reviewed by the course team and then, after the due date has passed, your score will be populated from Gradescope into your Canvas grade.

Evaluation

Your submission will be auto-graded. Please review the Evaluation Criteria for how your source code will be assessed. Submissions will be evaluated based on the criteria and will receive a total score. Submissions missing any part of the project will be graded based on what was submitted against the evaluation criteria. Missing parts submitted after the submission deadline will **not** be graded.

Review the course syllabus for details regarding late penalties.

Evaluation Criteria

Your submission will be evaluated and scored based on these criteria:

- Your code will be tested against a series of eleven (11) increasingly difficult test cases
- Each test case will verify your public key vector, encryption method, and decryption method.
- To ensure reasonable partial credit, the first test gives 40 points for the public key vector, and 20 points each for encryption and decryption.
- The latter ten (10) test cases offer 1, 0.5, and 0.5 points respectively.