Programming Project: Build a Local Server to Generate Schnorr Proofs

Objective:

In this project, you will build a simple Python server that generates and returns Schnorr proofs in JSON format. The server can be queried as many times as desired. This project will introduce you to cryptographic concepts like Schnorr proofs, as well as practical skills such as setting up a Flask web server and Dockerizing applications.

This programming excercie values **10 points** in total, the remaining are theoretical exercises in `DM1-theorique.pdf`.

Deadline: 23h59, Friday, October 18, 2024

Schnorr Proof Recap:

The Schnorr proof is a cryptographic protocol used for proving knowledge of a secret exponent `x` without revealing the value of `x` itself. The proof has three key components:

- `r` (commitment): Generated using a random nonce `k`.
- `e` (challenge): Computed by hashing the commitment.
- `s` (response): A function of the nonce `k`, challenge `e`, and secret exponent `x`.

The goal is to build a server that will allow users to query the server for Schnorr proofs. It is recommended to revise the slides from the first class.

Project Requirements:

1. **Language**: Python
2. **Library**: Flask for the web server, `pycryptodome` for cryptographic operations.
3. **Environment**: The project will run locally but will be Dockerized for easy setup.
4. **Basic Goal**: A user can make a request to your server, and the server will return a

Schnorr proof in JSON format.

Project Steps:

1. Install Prerequisites:

- Install Python (version 3.8 or higher recommended)
- Install Flask and pycryptodome:

```
pip install Flask pycryptodome
```

2. Build the Server:

Task 1 (3 points). Create a file called `schnorr_server.py` with the following content, following the TODOs:

```python
from flask import Flask, jsonify
from hashlib import sha256
from random import randint
from Crypto.Util.number import getPrime

# Flask app initialization
app = Flask(__name__)

# Set the parameters for Schnorr proof
p = getPrime(512)  # Large prime modulus
g = 2  # Generator
x = randint(1, p - 1)  # Private key (secret exponent)
y = pow(g, x, p)  # Public key g^x mod p

@app.route('/schnorr-proof', methods=['GET'])
def schnorr_proof():
    """
    Generate Schnorr proof for the secret exponent `x`.
    Return the proof as a JSON object.
    TODO: complete the code in ?? below
    """
    ??

    # Return the proof (r, e, s) to the user
    proof = {
        'r': r,
        'e': e,
        's': s,
        'y': y,  # Public key
        'g': g,  # Generator
        'p': p,  # Prime modulus
    }

    return jsonify(proof)


@app.route('/')
def home():
    return "Schnorr Proof Server - Query /schnorr-proof to get a Schn


if __name__ == '__main__':
    # Start the Flask server
    app.run(host='0.0.0.0', port=5000)
```

Important: This is the non-interactive version of Schnorr proof, this `schnorr_server.py` plays the role of prover, and use Fiat-Shamir transformation to

derive the challenge (instead of receiving it from the verifier).

3. Test locally

Task 2 (1 points). After commpleting the server code, it must be testable locally, using

```
    python schnorr_server.py
    curl http://localhost:5000/schnorr-proof
```

The response should be a JSON containing the Schnorr proof with `r, e, s`, along with the public key `y`, generator `g`, and prime modulus `p`.

4. Test as a docker

Task 3 (1 points). As another option for testing, make your server Dockerized.

Create a Dockerfile to run the server in a container, following the TODOs:

```
# Use an official Python runtime as a parent image
FROM python:3.9-slim

# TODO: Complete the ??
# Setup the working directory in the container
??

# Install the required dependencies
??

# Make port 5000 available, then run the Flask app
??
```

The Docker image should be built and its container can be run successfully using:
`bash docker build -t schnorr-server . docker run -p 5000:5000 schnorr-server`

5. Add a verifier endpoint

Task 4 (3 points). The next goal is to extend the server to include verification.

- Add a verification function to check if the proof is valid.

- Create a new endpoint `/verify-proof` where the user can send the proof they want to verify.
- Check the proof in the new endpoint by using the verification function.

Modify your `schnorr_server.py` to include the verification endpoint, following the TODOs:

```
from flask import Flask, jsonify, request
from hashlib import sha256
from random import randint
from Crypto.Util.number import getPrime

# Flask app initialization
app = Flask(__name__)

# Set the parameters for Schnorr proof
p = getPrime(512)  # Large prime modulus
g = 2  # Generator
x = randint(1, p - 1)  # Private key (secret exponent)
y = pow(g, x, p)  # Public key g^x mod p


@app.route('/schnorr-proof', methods=['GET'])
def schnorr_proof():
    """
    Generate Schnorr proof
    for the secret exponent `x`.
    Return the proof as a JSON object.

    This is your code from the above tasks
    """
    pass


def verify_proof(y, r, e, s, g, p):
    """
    Verify the Schnorr proof.
    :param y: Public key
    :param r: Commitment (r)
    :param e: Challenge (e)
    :param s: Response (s)
    :param g: Generator
    :param p: Prime modulus
    :return: Boolean indicating whether the proof
    is valid or not
```

```
        TODO: complete the ??
        """
        lhs = ??
        rhs = ??
        return lhs == rhs


    @app.route('/verify-proof', methods=['POST'])
    def verify_schnorr_proof():
        """
        Verify the Schnorr proof provided by the user.
        Expects JSON with fields: 'r', 'e', 's', 'y', 'g', 'p'
        """
        data = request.get_json()

        # TODO: complete the ??
        # Extract the proof data from the request, Call the verification
        # return
        ??

        result = {
            'valid': ??,
            'message': 'Proof is valid!' if ?? else 'Proof is invalid!'
        }

        return jsonify(result)


    @app.route('/')
    def home():
        return "Schnorr Proof Server - Query /schnorr-proof to get a Sch


    if __name__ == '__main__':
        # Start the Flask server
        app.run(host='0.0.0.0', port=5000)
```

Important. Your code should be testable locally using (commands are broken end-of-line to put on PDF)

```
curl http://localhost:5000/schnorr-proof
curl -X POST http://localhost:5000/verify-proof
  -H "Content-Type: application/json" \
-d '{"r": <r_value>, "e": <e_value>, "s": <s_value>,
      "y": <y_value>, "g": <g_value>, "p": <p_value>}'
```

where `<r_value>, <e_value>, <s_value>, <y_value>, <g_value>`, and `<p_value>` can be replaced with the actual proof data you received from the /schnorr-proof endpoint. The server must return whether the proof is valid.

6. Attacking a faulty implementation

Task 5 (2 points). A `schnorr_server.py` implementation is given below:

```python
from flask import Flask, jsonify, request
from hashlib import sha256
from random import randint
from Crypto.Util.number import getPrime

app = Flask(__name__)

# Set the parameters for Schnorr proof
p = getPrime(512)  # Large prime modulus
g = 2  # Generator
x = randint(1, p - 1)  # Private key (secret exponent)
y = pow(g, x, p)  # Public key g^x mod p

#  nonce `k`
k = randint(1, p - 1)

# the below code is faulty
@app.route('/vulnerable-proof', methods=['GET'])
def vulnerable_schnorr_proof():
    """
    Vulnerable server
    """
    r = pow(g, k, p)  # Commitment

    # Calculate challenge e as hash of r
    e = int(sha256(str(r).encode()).hexdigest(), 16)

    # Response s = k + e*x mod (p-1)
    s = (k + e * x) % (p - 1)

    proof = {
        'r': r,
        'e': e,
        's': s,
        'y': y,  # Public key
        'g': g,  # Generator
        'p': p,  # Prime modulus
    }
```

```
    return jsonify(proof)


@app.route('/')
def home():
    return "Schnorr Proof Server - Query /vulnerable-proof for the vulne

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Given the code of the above server, your goal is to interact with it and recover the secret key.

Submission instruction

- Submit the Python server code (`schnorr_server.py`).
- Optionally, submit the Dockerfile and instructions on how to build and run the Docker image.
- Include documentation on how to test the server and query for Schnorr proofs.
- If the exploit for **Task 5** is made automatic - querying to a container then output the secret - there will be bonus points.