PROJECT DESCRIPTION:

In this project, we implemented a communication channel and protocol that enables two participants, Alice and Bob, to check if they have any files in common without revealing the contents of the files to one another.

GITHUB LINK (with last commit id):

https://github.com/Aryan-Jain-1710/CS355-Project.git

Version Number (Last Commit ID): 0abc6ff3170e1f428d6c1258e5d31e0884525f34

PROTOCOL SPECIFICATION:

The protocol's objective is to identify common files between Alice and Bob without revealing the actual file contents of one to the other.

- Alice and Bob are subcontractors (security auditors) of the same company, so they only have access to the code segments they receive from the company.
- Each of them is given **5** code-segments in the form of files, each of size ∼500MB.
- Both Alice and Bob operate with incredible hostility, so they will exploit any local data they receive through the client-socket channel. They are also unwilling to share the actual contents of their files.
- Adversaries attempting to attack the communication channel are anticipated.

IMPLEMENTATION:

1. Project Structure:

- **socket_server.py**: The server side of the application, representing Bob.
- **socket_client.py**: The client side of the application, representing Alice.
- **main.py**: Contains helper functions for hashing contents of input files with SHA-256, and `sim_check2()` for finding overlaps between Alice and Bob's files.
- **rsa_gen.py**: Contains helper function for RSA key generation.

2. Procedure:

1. **Key Generation:** Each participant generates their own private and public RSA keys using the `rsa.generate()` function from the `PyCryptodome` library. This is done by generating a number N, the product of 2 random equal-length prime numbers p and q, and computing e, d such that $e \cdot d = 1 \bmod \phi(N)$. The private key is d, and the public key is (N, e).
2. **Key Exchange and Verification:** Participants exchange their public RSA keys through the established socket connection. They also send the received public keys back to each other for verification. The connection is terminated if public key verification fails, i.e., if the public key sent back doesn't match the public key in their possession that they sent out.
3. **File Exchange:** For each file, participants send two messages: the hashed value of the file contents and the RSA signature computed for the hashed file. The contents of the file are hashed using the SHA-256 algorithm, yielding a 256-bit digest of the file's contents without revealing any information about the file itself. The signature is then computed following the hash-and-sign paradigm, where each participant signs the hashed output of their file with their private key d and sends it to the other participant through the socket connection.
4. **Signature Verification:** Upon receiving messages, participants verify the RSA signatures on the hashed value of the file received with the public key in their possession from Step 2 to authenticate the source of the message. If a signature is not verified correctly, all computations are ceased and the connection is closed. The computations for the $\text{Sign}_{sk}(m)$ and $\text{Vrfy}_{pk}(m, \sigma)$ functions are as follows:

$$\text{Public Key} : (N, e) \qquad \text{Private Key} : d$$

$$\text{Sign}_{sk}(m) = H(m)^d \bmod N$$

$$\text{Vrfy}_{pk}(m, \sigma) : \text{ output 1 iff } \sigma^e = H(m) \bmod N$$

5. **Similarity Check:** After successfully exchanging files, a similarity check is performed using the `sim_check2()` function, and each participant is shown how many and the contents of which of their files are in common with the other participant.

3. Dependencies

- **socket**: For client-server communication.
- **PyCryptodome**: For RSA key generation.
- **hashlib**: For SHA-256 hashing.

4. Setup

1. Clone the repository:

```
git clone https://github.com/Aryan-Jain-1710/CS355-Project.git
cd CS355-Project
```

2. Install the required dependencies:

```
pip install pycryptodome
```

5. How to Run:

1. Run the server:

```
python socket_server.py
```

2. Run the client:

```
python socket_client.py
```

3. Follow the prompts to enter the names of 5 files.
4. The files will be securely exchanged between the participants, and the results of the file similarity check will be displayed for each participant.

6. References:

- https://cryptobook.nakov.com/digital-signatures/rsa-sign-verify-examples - for RSA Digital Signature Implementation
- https://realpython.com/python-sockets/ - for socket programming in python

SECURITY ANALYSIS (Security Goals and How They Are Achieved)

1. Secure Public Key Exchange

- **Objective:** Ensure the secure exchange of public keys, preventing man-in-the-middle attacks.
- **Achieved By:** Alice and Bob exchange public keys securely through the client-server socket connection. They also send the public keys they receive back for verification from the sender. The connection is terminated if the public key received back doesn't match the public key in their possession that they sent.

2. Confidentiality: Compare files without revealing file contents

- **Objective:** Enable Alice and Bob to identify common files without disclosing the actual contents of their files to one another.
- **Achieved By:** The contents of files are hashed using SHA256, an irreversible, one-way hashing algorithm, before being sent through the client-server channel. Moreover, Alice and Bob only have access to the files they are given as subcontractors of the company, so they don't have access to any other files, which prevents them from brute-forcing the hashing algorithm on the company's codebase.

3. Authentication: Ensure hashed file content is exchanged between Alice and Bob only

- **Objective:** Authenticate files to prevent unauthorized access and ensure data integrity.
- **Achieved By:** Files are authenticated using RSA signatures. A participant signs the hashed contents of their file with their private key and the other participant attempts to verify this signature with the public key that they receive. Any anomaly in communication or failure in signature verification results in the termination of the connection.

4. Limited Knowledge: Learn only about similarity results in the context of files Alice/Bob themselves possess

- **Objective:** Restrict the information revealed through similarity results specific to the files possessed by Alice and Bob.
- **Achieved By:** Each participant has a list of the hashes of the contents of the files they are given by the company. They also receive the hashed values of the contents of the other participant's files. Then, the `sim_check2()` function finds matches in these lists of hashed values and is used to display results tailored to each participant by only displaying the names of files owned by them that they have in common with the other participant. This prevents the disclosure of any information about the other participant's files (including not even revealing the other party's file name).

CONCLUSION:

These security measures collectively provide a robust framework for secure file exchange and communication between Alice and Bob.