Git-hub link:
https://github.com/EXCEPTIONal-King/Cryptography-project

Commit id: d8c82dfe85bbfab623d234bb76e0b89254422bc6
(final commit of code)
(pdf added after)

Security Goals and Effectiveness

Irretrievability

Irretrievability refers to the assurance that a received message cannot be used to reconstruct the original message.

Implementation:

One-Way Function: The code employs a one-way function to alter the file contents before transmission, explicitly using a hash function.
Hash Function: SHA-256, a widely adopted cryptographic hash function, is utilized to create a hash of the file contents.
Feasibility of Reconstruction: Reconstructing the original file from the hash is considered infeasible due to collision resistance properties. Multiple files could correspond to the same hash, and finding the original file would require breaking the collision resistance property.
Collision Probability: The probability of a false positive (two files having the same hash) is extremely low, with a chance less than 100 in (2^256), ensuring a highly secure one-way function.

Untamperability

Untamperability ensures that an intermediary cannot alter the contents of a message without it being apparent to the receiver.

Implementation:

Digital Signatures: The code employs digital signatures with public key verification to detect tampering with the transmitted data.
Signature Library: A signature library is used to facilitate the creation of signatures and generate corresponding public and secret keys.
Public Key Verification: The recipient uses the sender's public key to verify the signature, ensuring the integrity of the received data.

Secure Key Pair: Public and secret key pairs enhance the security of the signature mechanism, providing a robust means of ensuring message authenticity.

Security Analysis

Digital Signature Verification:

The digital signature verification process within the Peer class relies on the cryptographic functionality provided by the CryptoUtil class, which uses the standard Java security libraries for RSA signatures.

The code includes a robust mechanism for verifying digital signatures using the CryptoUtil class. Verification involves checking the signature against the public key and the hashed data. This contributes significantly to data integrity and authenticity during transmission, enhancing the overall security of the protocol.

Key Size:

The RSA key size is set to 2048 bits, providing a commendable level of security. While larger key sizes offer increased security, the chosen key size balances security and computational efficiency.

Hashing Algorithm:

The SHA-256 hashing algorithm used in the Network class is part of the standard Java security libraries, providing a well-established and secure means of hashing file content.

SHA-256, a widely accepted cryptographic hash function, is employed for hashing file contents. This algorithm ensures a high level of collision resistance, contributing to the overall security of the data exchange process.

Secure Communication:

The secure communication protocols in the code rely on standard Java socket communication libraries. Digital signatures contribute to data integrity and authenticity. The program establishes a secure connection between peers over the network. The integration of cryptographic techniques, including digital signatures, enhances the security of data transmission, ensuring that the exchanged data remains confidential and unaltered.

Specifications of the Algorithm

Steps:

Creates 2 instances (using StartProtocol.java Main() function), one for each communication party. (see Code Specification for StartProtocol arguments)

Each person:

Start 2 threads: a listener and then a sender to communicate with each party. The threads are joined at the end before the comparison

Sender:

Creates a “Network data object”. This contains a hash of a file’s contents, a signature on the hash and the public key corresponding to that signature. The hash is saved locally in an array for later comparison

Listener:

Receives a network data object.

Verifies the signature and stores the result

Stores the data object (containing the hash of a file, signature, and public key) to an

array.

Joined threads:

check signature verifications

If NOT verified: abort comparisons

If verified:

Compare each received hash to all stored local hashes

If any match, “Alice” and “Bob” share a file.

Otherwise, no files match

Files:

Peer: contains threads with function calls + send/receive operations

CryptoUtils: contains signature and hashing functions

Network: defines data object for sending over the network and has a function for creating such an object corresponding to a file.

StartProtocol: takes args, starts the program, creates peer

Each instance(Alice and Bob) has a corresponding folder from which they read the first 5 files to be the files they are attempting to compare. The files are named “codeSegment(1-5).bin” for each instance (we added temporary testing files to git).

Code Specification

Peer Class:

The Peer class utilizes the following libraries:

java.io.IOException for handling input/output exceptions.
java.io.ObjectInputStream and java.io.ObjectOutputStream for object serialization/deserialization.
java.net.ServerSocket and java.net.Socket for socket communication.
java.util.Arrays for array manipulation.

The code organization within the Peer class facilitates the implementation of a peer-to-peer communication protocol. It establishes listening and sending threads, verifies digital signatures, and compares received and locally stored data.

The constructor of the Peer class takes parameters for port, messagesPath, remotePort, and remoteHost. This class is responsible for managing the main operations of the peer, including initiating threads for listening (startListener) and concurrently sending (startSender) data. The code ensures that both listeners have started before attempting to send data.

The verification of digital signatures is a critical aspect of the Peer class. Before processing the received data, the program checks the authenticity and integrity of the data by verifying digital signatures using the CryptoUtil class. Additionally, the code compares received data with locally stored data to identify matches or discrepancies.

CryptoUtil Class:

The CryptoUtil class makes use of the following Java libraries:

java.security.* for cryptographic operations, including key pair generation, signature creation, and verification.
java.util.Base64 for encoding and decoding binary data to and from base64.

This class manages key generation, digital signature operations, and hashing using well-established Java security libraries. The RSA algorithm and SHA-256 hash function contribute to secure cryptographic operations.

The CryptoUtil class manages cryptographic operations, including key generation, signature creation, and verification. It uses the RSA algorithm with a key

size of 2048, balancing security and computational efficiency. The class also incorporates SHA-256 for hashing messages, contributing to data integrity.

This class is instrumental in ensuring the security of the communication protocol by generating and managing key pairs, signing data, and verifying signatures. Additionally, it provides a method to retrieve the public key, a crucial component in signature verification.

Network Class:

The Network class involves the following Java libraries:

java.io.FileInputStream for reading file contents.
java.io.IOException for handling input/output exceptions, particularly during file operations.
java.io.Serializable to make the NetworkData class serializable, allowing instances of this class to be converted to a byte stream for transmission.
java.security.MessageDigest for hashing file content using the SHA-256 algorithm.
java.security.NoSuchAlgorithmException addresses exceptions that may occur when attempting to use a cryptographic algorithm that is not available—specifically used in the context of the SHA-256 algorithm.
java.security.PublicKey Represents the public key component in the NetworkData class. Used during the verification of digital signatures in the Peer class.

The Network class defines the NetworkData class, encapsulating data, signature, and public key. It includes a method to generate NetworkData objects from file content, incorporating hashing and signing processes. The class focuses on the secure preparation and transmission of data between peers.

Using SHA-256 for hashing file contents enhances the collision resistance of the hashing algorithm. The generated NetworkData objects include signatures and public keys, contributing to the overall security of the data exchange process.

StartProtocol Class:

The StartProtocol class is the main entry point for initiating the peer-to-peer communication protocol. It takes command-line arguments for configuration parameters, namely port, messagesPath, remotePort, and remoteHost. The class creates a Peer object with the specified parameters, initiating the protocol.
Including command-line argument validation ensures the program receives the required input for proper execution. This class acts as the orchestrator, coordinating the initialization of the communication protocol. The arguments are used as such:

Argument 0: port that the local server (sender) will be opened upon
Argument 1: path of the file directory that contains “codeSegment(1-5).bin” to be sent
Argument 2: the remote port that the local client listens to in order to obtain the remote files (listening port)
Argument 3: the remote host IP to connect the listener socket to

We used the following command-line arguments for testing:
Peer 1: “6000 src/project/PeerOneMessages/ 7000 localhost”
Peer 2: “7000 src/project/PeerTwoMessages/ 6000 localhost”