BBM465 – INFORMATION SECURITY LAB.

ASSIGNMENT 2

10.11.2024

Group ID: 15

ECE NUR TAŞ – 2210356039

BÜŞRA BURHAN – 2220356051

```
// java ichecker createCert -k files\privatekey.enc -c files\publickey.cer
// java ichecker createReg -r files\registerfile.txt -p files\monitored -l files\logfile.txt -h SHA-256 -k files\privatekey.enc
// java ichecker check -r files\registerfile.txt -p files\monitored -l files\logfile.txt -h SHA-256 -c files\publickey.cer
```

publickey.cer, privatekey.enc, registerfile.txt, logfile.txt and an example folder monitored are placed in the files folder in the src.

CreateSertificate.java

General Overview

The CreateCertificate class is a Java tool designed to:

1. Generate an RSA key pair and a certificate in a Java KeyStore.
2. Export the certificate to a file.
3. Encrypt the private key using AES and save it to a file.

Explanation of Variables and Constants

- RSA_KEY_SIZE: Sets the size of the RSA key to 2048 bits.
- privateKeyPath and publicKeyCertPath: Store file paths for the private key and certificate.
- keystore, alias, storepass, keypass: Define the KeyStore's settings for storing the keys and certificate.
- The password should be entered by user is "**password1**".

Methods and Their Functions

1. createCertificate(String[] args): This is the main method that controls the process of creating keys and certificates. Its function can be described in steps:
- Reads command-line arguments for privateKeyPath and publicKeyCertPath.
- Prompts the user to enter a password, which will be used to encrypt the private key with AES.
- Generates an RSA key pair and a self-signed certificate.
- Exports the certificate to a file.
- Calls encryptPrivateKey to encrypt the private key using AES and save it to a file.

2. encryptPrivateKey(String keystore, String alias, String privateKeyPath, String password, String storepass, String keypass): This method encrypts the private key with AES and saves it to a file. Here's how it works:
- Loads the KeyStore and retrieves the private key.
- Hashes the user-provided password using MD5 to create the AES key.
- Encrypts the private key using AES and saves it to a file.

Step-by-Step Code Flow

1. Getting User Input:
- The program first reads command-line arguments provided by the user to retrieve the file paths where the private key (privateKeyPath) and certificate (publicKeyCertPath) will be saved.
- The program prompts the user to enter a password. This password is used later to encrypt the private key securely. The password itself is not stored but is instead converted into a key for encryption.

2. Key Pair Generation:
- Using Java's keytool command, the program generates an RSA key pair (a private key and a public key) with a key size of 2048 bits.
- It creates a self-signed certificate associated with this key pair, which acts as a digital identity proof for the generated public key.
- The generated key pair and certificate are stored in a Java KeyStore file called ichecker.jks.
- The alias "ichecker" is used to label the key pair, and a password (storepass) is set for accessing the KeyStore. Another password (keypass) is set specifically for accessing the private key within the KeyStore.
- The keytool command (keytool -genkeypair -alias ichecker -keyalg RSA -keysize 2048 -validity 365 -keystore ichecker.jks -storepass password1 -keypass password1 -dname) is executed using Java's ProcessBuilder to manage the key and certificate creation process.

3. Certificate Export:

- The certificate, which holds the public key, is exported from the KeyStore using the keytool command. This certificate is saved to the specified publicKeyCertPath, making it available for public use or sharing.
  - This export allows other parties to verify the identity of the owner and encrypt messages for the owner using this public key, while the private key remains secured within the KeyStore.
- The keytool command (keytool -exportcert -alias ichecker -file publickey.cer -keystore ichecker.jks -storepass password1) is executed using Java's ProcessBuilder to manage the key and certificate creation process.

4. Private Key Encryption:
- To secure the private key, the encryptPrivateKey method is used, which performs the following actions:
  - The method loads the private key from the KeyStore using its alias and password.
  - The user-provided password (entered in step 1) is hashed using the MD5 algorithm to generate a 128-bit AES encryption key.
  - This AES key is then used to encrypt the private key with AES encryption. AES is chosen for its speed and security in encrypting sensitive data.
  - Finally, the encrypted private key is saved to the specified privateKeyPath file. This makes the private key accessible only to those with the correct password, adding an extra layer of security.

This approach securely generates, stores, and exports the RSA key pair and certificate while protecting the private key through encryption.

CreateRegisterFile.java

General Overview

The CreateRegisterFile class is a Java tool designed to:

1. Generate a registry file containing hashed values of files within a specified directory.
2. Use a private RSA key to sign the registry file.
3. Log the process in a log file for auditing and traceability.

Explanation of Variables and Constants

AES_ALGORITHM and RSA_ALGORITHM: Constants defining the algorithms (AES for decryption and RSA for signature generation).

registryFilePath, directoryPath, logFilePath, privateKeyPath, hashAlgorithm: Stores file paths and settings for the registry file, target directory, log file, private key, and hashing algorithm.

Methods and Their Functions

CreateRegister(String[] args): The main method that drives the registry file creation and signing process, described as follows:

-Parses command-line arguments for paths to the registry file, target directory, log file, hash algorithm, and private key

-Prompts the user to enter a password, which will be used to decrypt the private key for signing the registry file.

-Traverses the specified directory and generates a hash for each file using the selected algorithm (MD5 or SHA-256).

-Writes the hashed values to the registry file.

-Signs the registry file with the decrypted RSA private key.

-Logs the process details to a log file.

hashFilesInDirectory(String directoryPath, String hashAlgorithm): Hashes each file in the specified directory.

-Recursively traverses the directory.

-For each file, calculates the hash using the chosen algorithm (MD5 or SHA-256).

-Stores the hash in the registry file, creating a verifiable list of file states at a given time.

signRegistryFile(String privateKeyPath, String registryFilePath, String password): Signs the registry file to ensure integrity.

Loads the private key from the specified path, decrypting it with the provided password.

Generates a signature using RSA, authenticating the registry file.

Appends the signature to the registry file, ensuring that modifications to the file can be detected.

Step-by-Step Code Flow

1.Parsing Command-Line Arguments:

Reads command-line arguments to retrieve paths for the registry file, target directory, log file, hash algorithm, and private key.

Validates the hash algorithm to ensure it is either "MD5" or "SHA-256".

Prints usage instructions if any required argument is missing.

2.Hashing Files in Directory:

Traverses the specified directory and generates a hash for each file using the chosen algorithm (MD5 or SHA-256).

Writes each hashed value to the registry file, creating a snapshot of the file states in the directory at the time of execution.

3.Signing the Registry File:

Prompts the user for a password to decrypt the private key.

Decrypts the private key using AES, with the password hashed to create the AES key.

Signs the registry file using the private key to provide integrity verification.

The signature is appended to the registry file, ensuring that any modifications to the file can be detected.

4.Logging:

Writes a log entry for each hashed file, detailing the file path and generated hash.

Records process steps in the log file, creating an audit trail for the registry file creation and signing process.

IntegrityChecker.java

General Overview

We cannot figure out why verification always fails. We checked the additional newline characters etc. But we ensure that the rest of the code would work if the verification completed as expected.

The IntegrityChecker class is a Java tool designed to:

1. Verify the digital signature of a registry file using a public key certificate.
2. Check the integrity of a specified directory against the contents of the registry file.
3. Log changes (such as created, altered, or deleted files) into a log file with timestamps.

Explanation of Variables and Constants

-registryFilePath, directoryPath, logFilePath, hashAlgorithm, certPath: Variables that store the paths and algorithm information provided via command-line arguments.

-hashAlgorithm: Specifies the hashing algorithm used, which can be either "MD5" or "SHA-256".

-certPath: The path to the public key certificate file used to verify the digital signature.

Methods and Their Functions

1. **checkIntegrity(String[] args)**:

   -This is the main method that controls the process of verifying the signature and checking the integrity of the directory. The steps include:

   -Reads command-line arguments to retrieve file paths and settings.

-Verifies the digital signature of the registry file using the verifySignature method.

-If the signature is valid, it checks the directory for changes using the checkDirectoryIntegrity method.

2. **verifySignature(String registryFilePath, String certPath)**:

   -Verifies the digital signature at the end of the registry file using the public key stored in the specified certificate.

   -The function performs the following steps:

   -Loads the content of the registry file and extracts the signature from the last line.

   -Loads the public key from the certificate file.

   -Verifies the extracted signature against the contents of the registry file.

3. **checkDirectoryIntegrity(String registryFilePath, String directoryPath, String logFilePath, String hashAlgorithm)**:

   -Compares the current state of the specified directory with the entries in the registry file.

   -Logs changes (such as created, altered, or deleted files) to the specified log file.

4. **loadRegistry(String registryFilePath)**:

   -Loads the contents of the registry file into a Map where the key is the file path and the value is its hash.

5. **log(String logFilePath, String message)**:

   -Logs a message to the specified log file with a timestamp in the format dd-MM-yyyy HH:mm:ss.

6. **hashFile(Path filePath, String algorithm)**:

   -Computes the hash of a file using the specified hashing algorithm ("MD5" or "SHA-256").

Step-by-Step Code Flow

1. **Getting User Input**:

   -The program first reads command-line arguments provided by the user to retrieve the paths for the registry file, directory, log file, hash algorithm, and certificate file.

   -Validates the input to ensure all required arguments are provided.

2. **Digital Signature Verification**:

-The verifySignature method reads the registry file and extracts the signature.

-It loads the public key from the certificate file and verifies the signature against the content of the registry file (excluding the signature line).

-If the verification fails, it logs the error and exits.

3. **Directory Integrity Check**:

   -If the registry file verification is successful, the program proceeds to check the integrity of the directory.

   -It reads the contents of the registry file and compares the stored hashes with the current state of the files in the specified directory.

   -If a file is found to be altered, deleted, or newly created, it logs the change.

4. **Logging Changes**:

   -The program logs any detected changes to the log file with a timestamp.

   -If no changes are detected, it logs a message indicating that the directory is unchanged.