# Assignment 6

## Part 3: Implement Security in Client & Server

You will work individually on this assignment.

1. Modify the Python client and the Java server source code files to provide security mechanisms described below. You may use the instructor’s solution for both the client and server as your starting point.
2. Implement one of the security mechanisms described in 2.a or 2.b.
	1. Implement the SSL/TLS protocols between the Python-based client and Java-based server. See the **SSL/TLS Protocols** section for more details. When you choose this option, this is the first thing your client and server should do (i.e., establish an SSL/TLS connection). Once the secure channel is established, the client and server should follow the **Authentication Protocol**. A student may earn up to 15 bonus points if they document their attempts at getting this to work.
	2. Implement cryptography as described in the **Synchronous Cryptography** section. When you choose this option, all messages sent between your client and server shall be encrypted only after authentication has completed.
3. Implement authentication as described in the **Authentication Protocol** section.
4. As a bonus, implement logging as described in the **Compromise Recording Log** section. Implementing logging on the client is worth up to 10 bonus points. Implementing logging on the server is worth up to 10 bonus points.
5. Submit a zip file that contains all of your source code files for your updated client and updated server.

## Additional Information

* The public key and private key values to use in the **Authentication Protocol** are described in the **Static Values** section. These were generated using the RSA asymmetric algorithm.
* The 23-digit vending machine id value to use in the **Authentication Protocol** are described in the **Static Values** section.
* The server will continue to use the VendMach\_MachineData and VendMach\_MachineDataError classes (given to you in the JAR file for assignment 5) to obtain INV data. The INV data produced by these classes is in plaintext.

## Summary of Security Mechanisms

The table below identifies the security mechanisms used for steps 2 through 4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Security Mechanism** | **Step 2a** | **Step 2b** | **Step 3** | **Step 4** |
| Establish SSL/TLS connection | Yes |  |  |  |
| Public key cryptography (RSA) |  |  | Yes |  |
| Synchronous cryptography (AES) |  | Yes |  |  |
| Cryptographic hash function (SHA-256) |  |  | Yes |  |
| Write log file |  |  |  | Yes |

# SSL/TLS Protocols (details for step 2.a)

This section provides details on creating a SSL/TLS connection between the client and server.

1. The student will need to explore the ***Resources for SSL/TLS and Crypto*** document included in this Canvas assignment page.
2. It looks like the SSL/TLS protocol requires that the following be done to establish a secure channel.
	1. Generate a digital certificate that includes the private key value specified in the **Static Values** section.
	2. Store this digital certificate in a keystore.
	3. Use the keystore on the server-side of the SSL/TLS channel.
	4. There is likely more that needs to be done, so explore.

# Synchronous Cryptography (details for step 2.b)

This section provides details on having the client and server both use synchronous cryptography.

1. The student will need to explore the ***Resources for SSL/TLS and Crypto*** document included in this Canvas assignment page.
2. Use a synchronous cryptographic algorithm that is supported by both Java and Python.
	1. Use a version of the AES crypto algorithm if it exists in both languages. (Note: the instructor knows that Java supports the AES/ECB/PKCS5Padding symmetric key algorithm.)

# Authentication Protocol (details for step 3)

This section provides details on the authentication protocol that is used to authenticate the client to the server and the server to the client. The authentication protocol that you shall implement is based on whether you are implementing step 2.a (SSL/TLS) or step 2.b (synchronous cryptography).

Refer to the **Authentication Message Fields** section for details on the fields found in each authentication message.

## If Implementing step 2.a (SSL/TLS)

Since you are using SSL/TLS, your authentication messages will automatically be encrypted and decrypted by the Transport layer (SSL/TLS) protocol. Given this, follow the steps below to have the client and server authenticate to each other.

1. The Client sends the first authentication message to the server. This message is:

**CHELO digest-machId digest-random-number cipher-machId-private null3**

1. The server will validate this CHELO message as follows:
	1. The server will ensure that the first six characters are CHELO followed by a space character. When the first six characters do not match what is expected, a validation error has occurred.
	2. The server shall decrypt ciphertext-machID-private using the public key for Vending Machine, Inc., then use the resulting plaintext as input into the SHA-256 cryptographic hash function to produce a message digest. It will then compare the computed digest with digest-machId. When the two digests do not match, a validation error has occurred.
2. If the server finds any validation error, it terminates the TCP connection.
3. If the server finds no validation errors, the server sends the second authentication message to the client. This message is:

**SHELO digest-random-number2 null3**

1. The client will validate this SHELO message as follows:
	1. The client will ensure that the first six characters are SHELO followed by a space character. When the first six characters do not match what is expected, a validation error has occurred.
	2. It shall compute the second random number (using its secure random number object), compute a digest for this random number using the SHA-256 cryptographic hash function, and then compare this computed digest to digest-random-number2. When the two digests do not match, a validation error has occurred.
2. If the client finds any validation error, it terminates the TCP connection.
3. Assuming the client validation of the SHELO message finds no errors, the client sends the third authentication message to the server. This message is:

**CHLLO null3**

1. The server will validate this CHLLO message as follows:
	1. The client will ensure that the first six characters are SHLLO followed by a space character. When the first six characters do not match what is expected, a validation error has occurred.
2. If the server finds any validation error, it terminates the TCP connection.
3. Assuming the server validation of the CHLLO message finds no errors, authentication is successful. Vending Machine protocol messages may now be sent between client and server.

## If implementing step 2.b (synchronous cryptography)

Since you will be using synchronous cryptography, your authentication messages will not be encrypted until after authentication has successfully completed. Given this, follow the steps below to have the client and server authenticate to each other.

1. The Client sends the first authentication message to the server. This message is:

**CHELO digest-machId digest-random-number cipher-machId-private null3**

1. The server will validate this CHELO message as follows:
	1. The server will ensure that the first six characters are CHELO followed by a space character. When the first six characters do not match what is expected, a validation error has occurred.
	2. The server shall decrypt ciphertext-machID-private using the public key for Vending Machine, Inc., then use the resulting plaintext as input into the SHA-256 cryptographic hash function to produce a message digest. It will then compare the computed digest with digest-machId. When the two digests do not match, a validation error has occurred.
2. If the server finds any validation error, it terminates the TCP connection.
3. Assuming the server validation of the CHELO message finds no errors, the server sends the second authentication message to the client. This message is:

**SHELO digest-random-number2 digest-secretKey cipher-secretKey-private null3**

1. The client will validate this SHELO message as follows:
	1. The client will ensure that the first six characters are SHELO followed by a space character. When the first six characters do not match what is expected, a validation error has occurred.
	2. It shall compute the second random number (using its secure random number object), compute a digest for this random number using the SHA-256 cryptographic hash function, and then compare this computed digest to digest-random-number2. When the two digests do not match, a validation error has occurred.
	3. It shall decrypt cipher-secretKey-private using the Vending Machine, Inc. public key. It will then compute the digest for this key using the SHA-256 cryptographic hash function. It will compare this computed digest to digest-secretKey. When the two digests do not match, a validation error has occurred.
2. If the client finds any validation error, it terminates the TCP connection.
3. Assuming the client validation of the SHELO message finds no errors, the client sends the third authentication message to the server. This message is:

**CHLLO null3**

1. The server will validate this CHLLO message as follows:
	1. The client will ensure that the first six characters are SHLLO followed by a space character. When the first six characters do not match what is expected, a validation error has occurred.
2. If the server finds any validation error, it terminates the TCP connection.
3. Assuming the server validation of the CHLLO message finds no errors, authentication is successful. Vending Machine protocol messages may now be sent between client and server.

## Authentication Message Fields

The fields of the authentication messages are listed alphabetically in the table below.

| **Authentication Message Field** | **Description** |
| --- | --- |
| CHELOSHELOCHLLO | The name of the message.This must be five characters long followed by a space character.***IMPORTANT***: None of the other fields in an authentication message have a delimiter character that ends the field. That is, the fields below appear as consecutive bytes in the message. |
| cipher-machId-private | The cipher text produced by encrypting the vending machine’s 23-digit identifier using the private key for Vending Machine, Inc. This value is represented as a byte array (in Java) or a bytes object (in Python).The length of this ciphertext is unknown |
| cipher-secretKey-private | The cipher text produced by encrypting the secret key generated by the vending machine acting as the server (see digest-key-secret for another use of this same secret key). This ciphertext is encrypted using the private key for Vending Machine, Inc.The length of this ciphertext is unknown. |
| digest-machId | The message digest value for the 23-digit id of the vending machine acting as the server. This value is represented as a byte array (in Java) or a bytes object (in Python).This must use the SHA-256 cryptographic hash function to produce the digest value.This digest value is exactly 32 bytes in length. |
| digest-random-number | The message digest value for the ***first*** random number generated by a cryptographically secure random number generator using the vending machine’s 23-digit identifier modulo 263-1 as the seed. This value is represented as a byte array (in Java) or a bytes object (in Python).This must use the SHA-256 cryptographic hash function to produce the digest value.This digest value is exactly 32 bytes in length. |
| digest-random-number2 | The message digest value for the ***second*** random number generated by a cryptographically secure random number generator using the vending machine’s 23-digit identifier modulo 263-1 as the seed. This value is represented as a byte array (in Java) or a bytes object (in Python).This must use the SHA-256 cryptographic hash function to produce the digest value.This digest value is exactly 32 bytes in length. |
| digest-secretKey | The message digest value for the secret key generated by the vending machine acting as the server. The message digest value is represented as a byte array (in Java) or a bytes object (in Python). See the **Symmetric Cryptography** section for how the secret key value is generated.This must use the SHA-256 cryptographic hash function to produce the digest value.This digest value is exactly 32 bytes in length. |
| null3 | A message terminator value that consists of exactly three null bytes i.e., three 0x00 values stored in a byte array (in Java) or in a bytes object (in Python). |

# Compromise Recording Log (details for step 4)

This section provides details on how the client and server will log messages that is sends and receives.

1. The log file created by the client or server will have a similar format, as described below.
	1. Each client request will be written to a single text line in the log file. This format is as follows:

**client yyyy-mm-dd hh:mm:ss:MM msg**

where hh is based on a 24-hour day, mm is the minutes within the hour, ss is the seconds within the minute, MM is the milliseconds within the second, and msg is the plaintext of the client request message.

* 1. Each server response will be written to a single text line in the log file. This format is as follows:

**server yyyy-mm-dd hh:mm:ss:MM msg**

where hh is based on a 24-hour day, mm is the minutes within the hour, ss is the seconds within the minute, MM is the milliseconds within the second, and msg is the plaintext of the server response message.

* 1. When msg contains binary data (e.g., see the **Authentication Protocol** section), each byte of this binary data should be written to the log file as two hexadecimal digits.

# Static Values

While in a *real-world* application, we would ***never*** hard-code the values below into our code, this is exactly what we will do in our Vending Machine project.

## Private Key (RSA)

The private key value contains 635 bytes. The integer values of these are as follows.

[48, -126, 2, 119, 2, 1, 0, 48, 13, 6, 9, 42, -122, 72, -122, -9, 13, 1, 1, 1, 5, 0, 4, -126, 2, 97, 48, -126, 2, 93, 2, 1, 0, 2, -127, -127, 0, -49, -26, -98, -41, 48, -120, 103, 25, -68, 67, 10, -44, 26, -94, -62, -114, -87, 15, -118, 26, 49, 117, 116, -110, 71, -80, -88, 81, -90, 13, 111, -98, -86, -38, -127, -35, -96, 27, 82, 37, 2, 11, -123, -101, 91, -67, -72, 7, 25, -7, 63, 77, 16, 3, -92, 51, -101, 6, 126, 74, 113, 103, -68, 60, -118, -58, -90, -78, -9, -63, 58, -31, -50, -89, -37, 117, -33, 108, 27, 53, -71, -49, 60, 88, 91, 33, -10, -63, 24, -110, 79, -24, -33, -4, -17, -63, 16, 93, 81, 46, 94, 84, -2, -31, 56, -68, 56, 89, -114, 2, 83, -14, 51, -122, 46, -31, -21, -18, 46, 100, -122, 84, -22, 76, 18, -79, 10, 49, 2, 3, 1, 0, 1, 2, -127, -127, 0, -124, 92, 19, 44, 117, 93, -94, 38, 36, 81, -102, -121, 71, 29, -16, 55, -37, 50, 101, 126, 31, -106, -29, -52, 109, -113, -18, -39, 108, -102, -119, 65, -65, 12, -74, 57, 95, 65, -123, -98, -62, 26, -117, -55, 73, 51, 20, 2, -111, 106, -71, -2, 89, -1, -73, 126, -109, 104, 34, -25, 74, 35, -3, 22, 120, 91, 5, -87, 2, -108, -23, 18, -74, 8, -8, -128, 22, 22, 9, -123, -88, 12, 9, 66, 11, 26, -103, -24, -97, -45, -58, -75, 11, 100, -33, 100, 95, -69, 102, 98, 74, -112, -31, -71, 79, 33, -19, 89, -68, 0, -63, -84, 54, 27, 121, 48, 47, -96, -28, 87, -90, 64, -118, -86, -45, 33, 91, 113, 2, 65, 0, -17, -63, 106, -28, -25, -124, -6, -92, -59, 18, -11, 80, -27, 18, -110, -94, -3, -84, 107, -115, -35, -66, 54, -47, 100, -13, 40, -75, -22, 78, -84, 0, 101, -75, -27, 75, -36, -98, -99, -47, -65, 110, -124, 69, 20, -76, 31, -41, 16, 73, -38, 83, -62, 67, -43, 53, -43, 114, -48, 92, 20, -85, -39, -121, 2, 65, 0, -35, -4, -82, 56, 122, 31, -112, 44, -65, 50, 50, -6, -52, 107, 15, 29, -107, -48, -80, 116, -62, -10, -127, -114, 3, 78, -49, -108, 9, 46, 22, -119, -59, 97, -63, -15, -86, -1, 93, -81, 73, -97, 102, -52, 111, 33, 53, 20, -18, 7, -79, 8, 84, 55, 12, -107, 48, 27, -28, -125, 78, -23, 12, -121, 2, 64, 45, 76, 31, -60, 11, -79, -26, 33, 7, 82, -29, -94, 91, 5, 61, 75, 42, 30, 79, 91, -56, 79, 77, -48, 22, 0, -110, -22, 123, -48, 77, 82, -19, -28, -123, -29, -27, 84, -38, -128, 108, 104, -106, -8, -18, 70, 113, -43, -110, 48, 65, -35, -99, 80, 111, -111, -14, 55, -11, 1, -48, -83, -1, 53, 2, 65, 0, -61, 48, -35, -104, 119, 121, -83, 114, 75, 36, 62, -61, 92, 95, 126, -55, 10, -28, -105, 99, -13, -46, -127, 20, 120, 64, 34, 50, 14, -24, 97, -108, -2, -29, -8, 127, 33, 86, -83, -17, -27, -110, -83, -9, 112, 120, 8, -114, -64, -58, 46, -102, 30, 66, -47, 99, -107, 11, -47, -96, -52, 16, 11, -99, 2, 64, 102, 99, -80, 102, 82, 81, 0, 84, -90, 105, 97, 113, -101, 117, 98, 18, -46, 43, 99, -56, 113, 100, -47, 74, -63, -40, 126, 56, -111, 115, -21, -41, 4, -110, 42, 95, -94, 61, -117, -125, -11, 10, 34, 115, 67, -117, 75, -101, 3, -26, 55, 13, 26, -94, 89, 112, 100, 61, 106, 110, 85, 92, -43, -78]

## Public Key (RSA)

The public key value contains 162 bytes. The integer values of these are as follows.

 [48, -127, -97, 48, 13, 6, 9, 42, -122, 72, -122, -9, 13, 1, 1, 1, 5, 0, 3, -127, -115, 0, 48, -127, -119, 2, -127, -127, 0, -49, -26, -98, -41, 48, -120, 103, 25, -68, 67, 10, -44, 26, -94, -62, -114, -87, 15, -118, 26, 49, 117, 116, -110, 71, -80, -88, 81, -90, 13, 111, -98, -86, -38, -127, -35, -96, 27, 82, 37, 2, 11, -123, -101, 91, -67, -72, 7, 25, -7, 63, 77, 16, 3, -92, 51, -101, 6, 126, 74, 113, 103, -68, 60, -118, -58, -90, -78, -9, -63, 58, -31, -50, -89, -37, 117, -33, 108, 27, 53, -71, -49, 60, 88, 91, 33, -10, -63, 24, -110, 79, -24, -33, -4, -17, -63, 16, 93, 81, 46, 94, 84, -2, -31, 56, -68, 56, 89, -114, 2, 83, -14, 51, -122, 46, -31, -21, -18, 46, 100, -122, 84, -22, 76, 18, -79, 10, 49, 2, 3, 1, 0, 1]

## Vending Machine 23-Digit Identifier

The vending machine identifier to use on both the client and server is:

71309548720180936275295

Please note that 263-1 is equal to 9223372036854775807.

Also note that 263-1 is the largest positive value that can be stored in a variable of type long (in Java).