UNIVERSITY OF MALTA

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

CPS3232 Applied Cryptography: 2020/21 Project

Software Security Module

Instructions

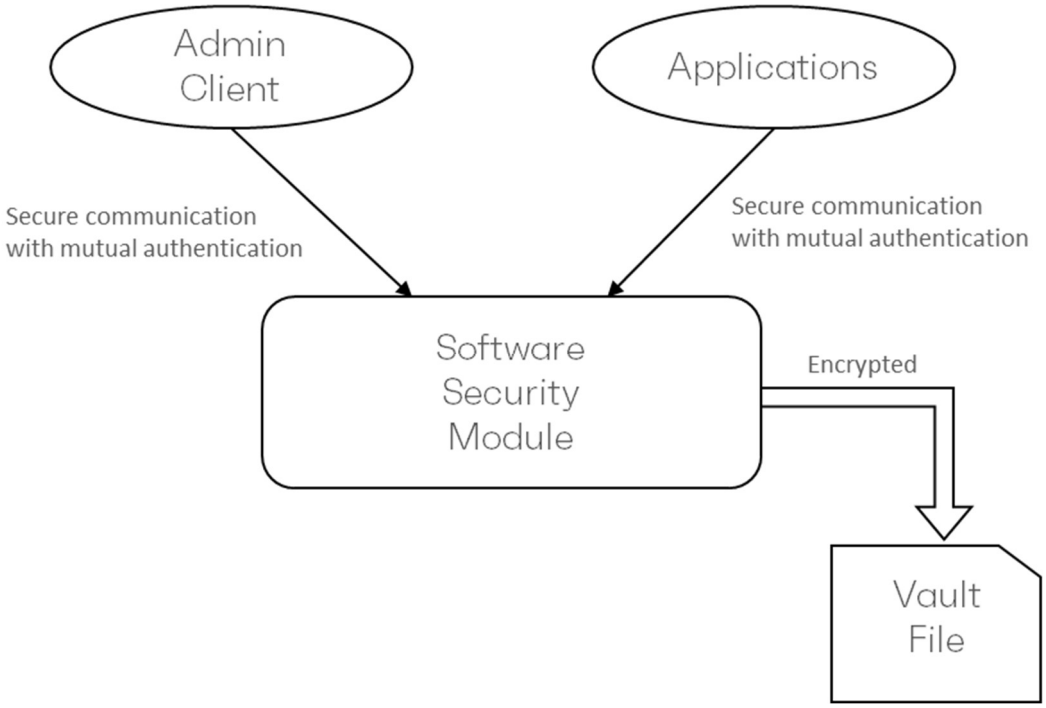
1. This is an individual assignment.
2. You may be required to demonstrate and present your work to an exam board.
3. A soft-copy of the report along with all digital artefacts must be uploaded to the VLE upload area by the deadline below. All files must be archived into a single .zip file. It is the student's responsibility to ensure that the uploaded zip file and all contents are valid. You must include all source, make files, any scripts and instructions required to compile the code.
4. Reports (and code) that are difficult to follow due to low quality in the writing-style/organisation/presentation will be penalised.
5. Assignment queries are to be made strictly during the beginning of lectures or posted to the assignment's forum on VLE, as of when individual tasks are announced in class till one week before the deadline (excluding recess).
6. This assignment comprises 50% of the final CPS3232 assessment mark.
7. You are to allocate 40 to 50 hours for this assignment.
8. You are allowed to make use of any programming language of your choice, unless otherwise indicated within specific tasks.
9. All programming tasks must be presented in the form of an overall design followed by code-snippet explanations.
10. While it is strongly recommended that you start working on the task as soon as they are announced in class, the firm submission deadline is **Thursday 28th January 2021 at noon.**

Overview

A **hardware security module** (**HSM**) is a secure , tamper-resistant physical computing device that safeguards and manages cryptographic keys, and performs cryptographic functions with these keys such as encryption and decryption for privacy, and **message authentication code** (**MAC**) systems for strong message authentication.

This assignment is based on the implementation of a software version of an HSM, namely a Software Security Module (SSM). A language of your choice (preferably Java or Python) can be used together with the necessary libraries (Java Cryptography Architecture, PyCryptodome, etc).

The overall architecture of the service is outlined in the following diagram:



- The ***Software Security Module*** is the core of the system which manages the keys and performs cryptographic functions. Its stores the ***Application keys*** in a ***Vault File*** , encrypted by a ***Key Encryption Key (KEK)*** . This KEK is also stored in the Vault, encrypted by a ***Master Key (MK)*** .

- The Application Keys are administered through an ***Admin Client Application*** , which connects to the SMM over a secure connection, with mutual authentication (e.g. TLS connection)

- ***Applications*** can connect securely to the SSM to use cryptographic services acting upon the configured application keys. These application keys are never exposed out of the SSM.

Concepts

The ***Master Key (MK)*** is a 2-part key with each part of minimum 16 characters long. Two different persons should be the owners of the 2 parts of the MK and should adhere to the key management policy. The MK is used for:

1. Authenticating the users administrating the SSM (to manage keys and applications).
2. A key derived from the MK, namely MK_1, is used to encrypt a Key Encryption Key (KEK) which is stored in the SSM vault - a file holding KEK and application keys.
3. Another key derived from the MK, namely MK_2, is used for a Strong integrity check of the stored $Encrypt_{MK2}\{KEK\}$ – this is checked upon reading the KEK from vault.

The MK is NEVER stored and is kept in memory for the minimum time possible. Thus the MK is required for every management operation.

The ***Key Encryption Key (KEK)*** is used to securely store Application Keys (AKs) which can be configured in the module. Two keys derived from the KEK are used to:

1. Encrypt the Application Keys data structure
2. Store a Strong integrity check of the encrypted data – this is checked upon reading the Application Keys data structure from the vault.

It is generated automatically by the SSM on vault creation and on the trigger of KEK rotation. It is stored in the vault encrypted with the MK. The KEK and the encrypted version $Encrypt_{MK1}\{KEK\}$ are both stored in memory, so that the vault can be persisted whenever application keys are automatically rotated.

Application keys are the cryptographic algorithm keys used by applications. Each one stores the following information:

- a unique name of the Application Key
- the algorithm e.g. AES, DES, RSA etc
- the algorithm parameters e.g. for symmetric (block) ciphers operation/padding, e.g. CBC/PKCS5Padding
- the algorithm initialization vector (IV) if required (e.g. for chaining block ciphers)
- Key Lifetime in days (crypto period)
- Auto-rotate Boolean (if true, while encrypting or signing, if key is expired, it is marked as DEACTIVATED and a new version is created)
- A list of the key values (versions) with the following data:
    - Key Version (auto increment integer)
    - Key value (or pair of values if you decide to cater for asymmetric keys as well)
    - Key creation time
    - Key expiry time
    - Key state ACTIVATED / DEACTIVATED / DESTROYED / COMPROMISED

The **Vault** is a file which holds the KEK encrypted and MACed by keys derived from the multi-part MK, MK_1 and MK_2 respectively. It also holds the application keys encrypted and MACed by keys derived from the KEK, KEK_1 and KEK_2 respectively.

Loading and persisting the vault (with new configuration) requires the MK for authentication purposes. The only exception is when application keys expire and newly ones are automatically created, in which case the vault will be persisted without any human intervention.

The **Admin Client** is an application which is used to connect to the SSM to manage applications (key requesters) and keys. It is an application which communicates with SSM over a secure channel (e.g. TLS). The SSM server certificate is required for server authentication, so it needs to be added to the admin client trusted-store.

Applications or **Application Clients** are SSM Service Users, which connect securely to the SSM and use cryptographic services with configured keys, for example encrypting or decrypting some text with a pre-configured key.

Tasks

1. Implement the SSM, which manages and stores application keys in the vault as explained above.
   The SSM accepts Administration Connection with the ability to execute the following commands:
   a. CHANGE_MASTER_KEY: Change the master keys parts and save a new copy of the Vault with a freshly coded KEK
   b. ROTATE_KEY_ENCRYPTION_KEY : Rotate the key with which application keys are encrypted & MACed in vault, and save a new copy of the Vault with a freshly coded Application Keys
   c. GET_APPLICATION_KEYS_INFO : Display summary info on all application keys currently in the vault
   d. GET_APPLICATION_KEY_INFO : Display info on a particular key with all its version history (except the key value!)
   e. CREATE_ APPLICATION_KEY : Create a new named application key. The key value can be randomly generated. It is stored in the vault.
   f. UPDATE_APPLICATION_KEY : Update parameters of an application key, updating the vault
   g. UPDATE_APPLICATION_KEY_STATE : Update state of an application key. If state is set to anything other than ACTIVATED, a new version is created. The vault is updated accordingly

   The SSM accepts Application Connections with the ability to execute the following commands:
   a. encrypt(keyName, plaintext) returning ciphertext. An error/exception is returned if Auto-Rotate is disabled and key is not in an ACTIVATED state.
   b. decrypt(keyName, keyVersion, cipherText) returning plaintext. An error/exception is returned if key state is DESTROYED.

   And if implementing Digital Signatures:
   c. sign(keyName, message) signing a message (string) using the private key of the application key with name 'keyName'. An error/exception is returned if Auto-Rotate is disabled and key is not in an ACTIVATED state.
   d. verify(keyName, keyVersion, message, digest) to verify the integrity of a message, using the public key of the application key with name 'keyName'. An error/exception is returned if key state is DESTROYED.

   You might need to use Base64 encoding to convert between binary and string representation of data.

The application keys are stored in the Vault. The structure of the elements stored is the following:

$Encr_{MK1}$ {KEK}
MAC_{MK2} {KEK}
$Encr_{KEK1}$ {List<Application-Keys>}
MAC_{KEK2} {List<Application-Keys>}

In your report describe the overall architecture/design of your solution and explain the core functionality (methods) of the SSM.

Implement the Admin Client service, which connects securely to the SSM Admin Port, and has the ability to execute the administration commands offered. Document with screenshots the use of these commands that the SSM offers.

Test the system with a sample Client Application, which connects to the SSM-Applications-Port securely and uses the encrypt, decrypt (and possibly sign and verify) services. In your documentation include screenshots of encrypting and decrypting back some text (and signing and verifying data authenticity if implemented).

For the Admin and Application clients, the SSM services can be exposed as 2 sets of APIs (e.g. REST), the admin APIs and Application Service APIs, and use a tool like Postman to connect securely (also using TLS certificate) and execute the commands.

[70 marks]

2. Using the developed software, test and document the performance of different encryption algorithms, namely DES, 3DES, and AES with 3 different key sizes. Define the 5 different keys in SSM and use a very large string (possibly repeating operation multiple times) or a text file for your plaintext. Take timing averages of 3 encryption and 3 decryption runs for each algorithm (using the same plaintext).

   [10 marks]

3. Discuss ways of enhancing the security of the SSM, such as supporting different ciphers, auditing operations with integrity checks, etc. (Compare with what HSMs offer).

   [10 marks]

4. As part of the documentation, outline a Key Management Policy for the Master Key.

   [10 marks]