Hardware Security Module (HSM) Integration Datasheet

Enterprise security has always been a priority in the development of WatchDox technologies and solutions. Thus, integrating with products such as hardware security modules (HSM) is one of the many ways in which WatchDox flexibly enables enterprises to build truly comprehensive security architectures, tailored to your organization's custom needs and IT environment.

This document provides an overview of how HSMs work, how it is used in WatchDox, and how to set up and configure an HSM within the WatchDox system.

Overview

Documents shared through WatchDox are stored in the server and encrypted symmetrically with AES-256. The HSM creates a different symmetric encryption key per document, by encrypting a hash value of the document's UUID (128-bit value generated by the WatchDox server). Every time this key is needed, the encryption operation is re-enacted. The WatchDox server never stores this key in non-volatile memory. In addition, for every new user in the system, the server generates a dedicated pair of asymmetric private and public keys that are stored in the database. The private key is encrypted using a symmetric key kept in the HSM. The system stores only root keys, but not the "per document" or "per user" private keys and is not used for SSL offload. WatchDox has integrated with the following HSM system:

1) HSM- SafeNet Luna SA K5 v4.4 (other models might be supported but verification is required)
2) The SafeNet Luna supports High Availability
3) Managed from command line shell

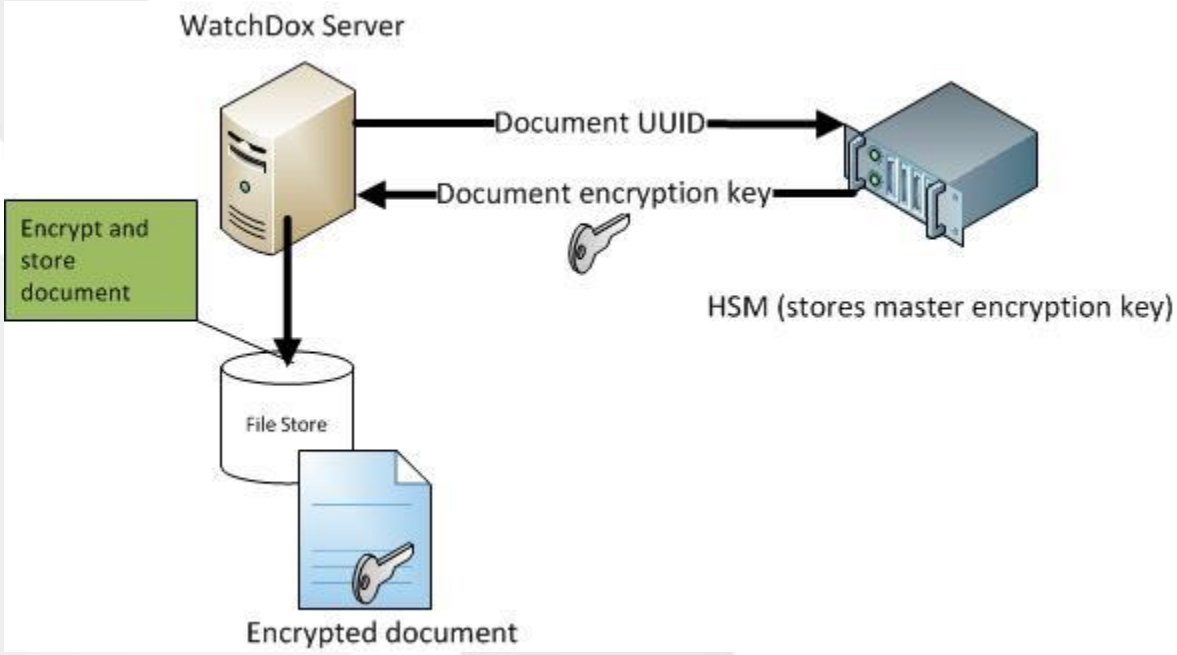

Figure 1: HSM & WatchDox – High Level

The WatchDox Upload, Conversion and Encryption Process

The following steps summarize document uploading and conversion, focusing on the security and encryption aspects of the process.

1. WatchDox workspace administrator uploads the incoming file over SSL to a Linux/Tomcat Machine. B. File is stream-encrypted and stored on NAS using a symmetric key. This key is generated as follows:
   a. An internal UUID is generated for each document.
   b. A series of dedicated keys are maintained on the HSM.
   c. The HSM encrypts the UUID based on an HSM key (Key1) using an AES algorithm.
   d. This generates a symmetric key (Key2) unique to that document which is used to encrypt and decrypt the document. This symmetric key is generated on demand by request to the HSM, and is not stored on the appliance.
2. If necessary, Tomcat inserts a conversion request message to Windows into the message queue.
3. Windows completes the following tasks:
   a. Monitors the message queue and pulls out any conversion requests sent from Tomcat.
   b. Extracts the file properties from the conversion request message.
   c. Sends a restful API call over SSL to Tomcat to retrieve this file.
4. Tomcat decrypts the requested file and sends it over SSL to the Windows machine
5. Windows converts the file to PDF or SWF format.
6. Windows seals the file as follows:
   a. The HSM encrypts the UUID using an AES algorithm. The encryption is based on a dedicated key (Key3) taken from the set maintained on the HSM. Key selection is based on the file type.
   b. This generates a new symmetric key unique to that file, which is used to encrypt and decrypt the converted document.
7. Windows sends the output files back to Tomcat over SSL using a restful API call.
8. The output files that were not sealed on the conversion machine (e.g. SWF) are encrypted on the Tomcat using keys generated using the UUID encrypted by separate HSM keys in a similar manner.

Keychain Management in iOS

The iOS keychain incorporates a series of keys which are generated through the Main Key, encrypted by the Head Key using AES-256 CBC. The Head Key is generated at time of passcode unlock by re-applying SHA-256 100 times with a salt on the passcode that the user enters. The Head Key is not stored permanently on the device. The effect of the Head Key being compromised is equivalent to revealing the passcode.

Note that the passcode is verified against a different stored value (through a passcode verification hash) which is derived from the passcode by re-applying SHA-256 100 times with a different salt.

The following information is encrypted through an AES key in CBC mode and is based on the Main Key:

- Device ID
- User's Private Key

- Most recent successful refresh time, which is necessary to enforce time limits for offline document access
- Offline document viewing notifications, stored for later sending
- Date when the app was last activated. Each time the app is activated, the current date is compared with the stored date. If the date was set back (for example, tampered in an attempt to view documents beyond their time limit), all cached documents and licenses are deleted. Note that this is *not* a full wipe. This simply prevents the user from viewing downloaded, offline documents beyond the permissible time limit.

File and Key Exchange Process

Encrypted files and key are sent to the device via SSL. The Private Key is sent only once.

Once on the device, the app's portion of the key chain is encrypted when the device is locked. Encryption is currently based on the device's passcode, if defined. Future versions will implement encryption based on a WatchDox passcode. The key chain is only available on the device on which it was stored; it does not migrate with backups to any other devices.

The user can configure the device to require a passcode. In this case, if at least ten minutes have passed since the last user activity, the user will have to enter a passcode to continue. Note that the device app continues to run in the background.

When the WatchDox app is first launched, if a passcode has been defined for the device then the user must enter that passcode before the app can actually do anything. Without the passcode the app is not able to decrypt the Main Key, which was used to encrypt the DeviceID, the Private Key, the database entries, and so on. Once the passcode is entered and Main Key is extracted and stored in RAM, it is not removed during normal device usage.

Wiping Files from Mobile Devices

Files are wiped from a device through WatchDox under the following circumstances:

- If the user attempts to enter the PIN incorrectly ten times, the Main Key is wiped and the app terminates. The next time the application is launched the encrypted files, keychain and other relevant data are wiped from the device.
- If a remote wipe command is sent while the application is running, the same wipe behavior (as described in the first bullet) is applied.
- If a jailbreak is detected upon application launch, a complete wipe is immediately implemented. (Wiping the Main Key, the encrypted files, keychain, and other data.) In addition, the app terminates after the launch, so that it cannot be run on jail-broken devices.
- When the app is activated, if it detects that someone has tampered with the device time setting, (for example, rolling the date back since the last time the app was activated), the app clears only the encrypted files. This prevents users from viewing documents that were downloaded for offline viewing if the time limit has expired.