01UDNOV - Cybersecurity for Embedded Systems

Alekos Interrante Bonadia (S319849)
Lorenzo Sebastiano Mathis (S314875)
Pietro Mazza (S314897)

Politecnico di Torino

Table of content

Intro to Cryptomator

Overview: Cryptomator is an **open-source encryption software** used to secure files before uploading them to the cloud. It creates **encrypted virtual disks** (vaults) using AES-256 encryption.

Key Features:

- Cross-platform compatibility (Windows, macOS, Linux, Android, iOS).
- Password-based encryption for secure data access.
- Integration with cloud services (Dropbox, Google Drive, OneDrive).
- Metadata encryption for added privacy.

Intro to SECube

Overview: SECube is a hardware security module (**HSM**) that combines an ARM Cortex M4 microcontroller, FPGA, and smart card to provide **robust cryptographic capabilities**.

Libraries:

- **L0 Library:** Basic communication functions.
- **L1 Library:** Authentication, key management, cryptographic operations.
- **L2 Libraries:** SEFile (encrypted file systems), SELink (secure communication), SEKey (distributed key management).

Communication and JNI

What is JNI (Java Native Interface): JNI is the bridge that **connects the Java bytecode** running in the JVM **with native code** (typically written in C or C++), enabling the use of native libraries and their performances and capabilities while maintaining the portability.

JNI vs Intermediary Programs:

- JNI enables tighter **integration** between Java and native libraries **without any intermediary process**.
- An intermediary process allows for greater decoupling between the components but introduces several security issues like mutual authentication of the two parties and Man-in-the-Middle (MITM) attack protection in IPC (Inter-Process Communication).

Elements:

- `System.loadLibrary(String libname)`: Loads a shared library into memory and makes its functions available to Java code.
- `jni.h`: Enable the C++ module to use the JNI types like `JNIEXPORT`, `JNICALL` or `JNIEnv.`

Example:

- In Java: `private native String myNativeMethod(String param);`
- In C++: `JNIEXPORT jstring JNICALL package_myNativeMethod(JNIEnv *, jobject, jstring);`

SECube API

SECubePublic API is our library which provides all the primitives allowing Cryptomator to interact with SECube

The public interface exposes:

- `vector<pair<string, string>> EnumerateSECubeDevices()`: to list all the SECube connected devices.

- `void GenerateSecurePassword(const string &vaultID, const string &serial, const string &pin, const uint8_t * password, int size)`: to **generate a new passphrase and store it** inside the secure database.

- `void RetrieveSecurePassword(const string &vaultID, const string &serial, const string &pin, const uint8_t * password, int size)`: to **retrieve the passphrase** from the secure database.

The module also provide various private functions used to perform all the required sub-tasks.

Note: the SECube API (`SECubeAPI.h`) is decoupled from the JNI logic (`SECubeConnector.cpp`) and it is a L3 library, it can be used outside the Cryptomator project.

Passphrase generation

Problem: how we can rely on the onboard TRNG to generate the passphrase if it can only be used to generate non-exportable keys?

Idea: we can use a strong encryption algorithm like AES with the IND-CPA property to generate a ciphertext that can be used as passphrase.

Implementation:

1. Generate a **random plaintext** (32 bytes) **and IV** (16 bytes) for AES **without the TRNG** (i.e. on Linux we used /dev/urandom).
2. Generate a **temporary 256 bit key** (new for each passphrase) **using the onboard TRNG** (using L1 library).
3. Encrypt the plaintext with **AES-256-CTR** and these parameters.
4. Use the **ciphertext as passphrase**, Cryptomator will receive 32 bytes ciphertext as 64 character hexadecimal string.

Device configuration

SECubePublic API is ou The `./src/main/cpp/` folder contains also an example and the `SECubeConfiguration.h` library which provides a set of function to setup from scratch the SECube device:

1. `ERROR_CODE SECubeFactoryInitRoutine()`: to **initialize the device** and to set all the parameters (serial number, admin pin and user pin).

2. `ERROR_CODE SECubeDBKeyInitRoutine()`: to **(re)generate the key** (`se3Key`) used to encrypt the secure database.

3. `ERROR_CODE SECubeDBInitRoutine()`: to **(re)create the SQLite3 database**.

In addition, it provides **some utility functions** that can be useful to list the devices and select one of them or to dump the database content for debugging purposes.

Live demo…

Known Issues & Future Work

Known Issues

- **MacOS Compatibility**
  - Current status: Unavailable for testing

Future Work

- **Improving TRNG Usage**
  - Current Method: Relies on TRNG and AES algorithm
  - Future Improvement: Direct API to use the TRNG directly in SECube firmware and L0 capabilities
- **Vault ID Generation**
  - Current Method: **9 bytes** random sequence using Java's Pseudo RNG
  - Future Improvement: Use UUID technologies (e.g., UUIDv1) for enhanced uniqueness