PV204 project – Key manager for Cryptsetup

Team B – Manoja Kumar Das, Ondrej Mosnáček, Mmabatho Idah Masemene

April 15, 2016

1 Introduction

Cryptsetup[1] is an open-source command-line application for creating and managing encrypted disk partitions on Linux. The main partition format used by Cryptsetup is LUKS (Linux Unified Key Setup)[2], but other formats are also partially supported. For simplicity, we shall focus only on LUKS partitions.

Our project aims to implement a wrapper command-line application that will enable the user to securely manage the encryption keys for the partitions using a smart card. The application shall support creating a new LUKS partition (with the key generated and stored on-card), unlocking an existing partition (such that has been created with the same smart card) and deleting an existing key from the card (optionally also wiping the associated partition's header).

2 Overall design

The application shall consist of three main components:

1. The *JavaCard applet* to be installed on the card.
2. The *host application* written in Java that shall allow communicating with the card using a simple command-line interface.
3. A set of *glue scripts* that shall process the user's commands and invoke the host application and Cryptsetup and pass data between them. These shall be written as shell scripts (or Python scripts if more complex functionality will be needed).

3 Use cases

3.1 Card initialization

Before use, the smart card must be initialized. The initialization should be done in an environment where the host and the communication with the card is secure.

During initialization, an initial *master password* shall be specified. The master password shall then be used to authenticate the user to the card during usage.

During initialization, the card shall also gnerate an RSA key pair, which shall be used to authenticate the card to the host machine during secure channel setup (see section 4) to prevent man-in-the-middle attacks. The private key of this key pair shall never leave the card; the public key can be exported at any time – it should be retrieved in a secure environment after initialization and stored securely (in terms of integrity) on the host.

3.2 Changing the master password

The user can change the card's master password. In order to do that, the old master password needs to be provided. Changing the password shall not erase nor modify the keys or other data stored on the card.

[1]`https://gitlab.com/cryptsetup/cryptsetup`
[2]`https://gitlab.com/cryptsetup/cryptsetup/wikis/LUKS-standard/on-disk-format.pdf`

3.3 Creating an encrypted partition

The application shall allow the user to create a new encrypted partition (i. e. format a block device with a new LUKS header). The user has to supply the card's master password to perform this operation. After verifying the master password, the card generates an encryption key, which is then passed to Cryptsetup to initialize the new partition (it will be used as the partition's *master key*). The key is then stored on the card, associated with the newly created partition based on its UUID (this is generated by Cryptsetup and stored in the partition header).

To allow for recovery of the partition's master key (and the encrypted data) in case the card gets stolen or destroyed, the application shall also set up the partition with a *recovery password.* The recovery password shall be set up via standard mechanism (LUKS key slot). The recovery password shall be automatically generated as a random string of 16 alphanumeric characters and the user shall be advised to store it securely (e. g. on a paper locked in a safe) and only use it in case of emergency.

3.4 Unlocking an encrypted partition

The application shall allow the user to unlock an encryptsed partition, whose key is stored on the card. The user has to supply the card's master password to perform this operation. After verifying the master password, the card provides the encryption key associated with the requested partion (based on the UUID) to the host application. The host application then passes the key to Cryptsetup to unlock the partition.

3.5 Deleting a stored key

The application shall allow the user to delete a key stored on the card. The user has to supply the card's master password to perform this operation. The key to be deleted shall be identified by the corresponding partition's UUID. The user can optionally request also the invocation of Cryptsetup that will erase the partition's header to also prevent future unlocking of the partition using the recovery password.

4 Card-host communication

In order to prevent man-in-the-middle attacks, the card and host must be mutually authenticated.

The authentication of the card to the host is done based on the RSA key pair generated by the card upon initialization (see subsection 3.1). The card's private RSA key is used to sign the card-to-host messages of an intial Diffie-Hellman key exchange. The key exchange is used to estabilish a shared symmetric session key that is used to encrypt (using AES-CBC) all subsequent communication between the card and the host (until the session is terminated).

After this secure channel has been estabilished, the host has to authenticate itself by sending the master password to the card over the secure channel. Only after the password has been successfully verified by the card, is the host authorized to send actual commands to the card (generate/store/retrieve/delete a key, etc.).

The process of estabilising the secure channel is described below. R_{priv} and R_{pub} are the card's private and public RSA authentication keys (respectively).

1. The host generates an EC key pair (H_{priv}, H_{pub}) and sends H_{pub} to the card.
2. The card receives H_{pub}, generates its own EC key pair (C_{priv}, C_{pub}), uses C_{priv} and H_{pub} to estabilish the session key K and sends $sign_{R_{priv}}(C_{pub}, H_{pub})$ to the host (H_{pub} is included in the signed message to prevent a replay attack).
3. The host verifies the received signed message using R_{pub}. If the signature was verified successfully and the recieved H_{pub} matches the one the host had sent, the host uses H_{priv} and C_{pub} to also estabilish the session key K.
4. The host and card now share a common key K, which they use to encrypt further communication.