Ada Keystore Guide

STEPHANE CARREZ

2019-11-11

Contents

1 Introduction

Ada Keystore is a library and tool to store information in secure wallets and protect the stored information by encrypting the content. It is necessary to know one of the wallet password to access its content. Ada Keystore can be used to safely store passwords, credentials, bank accounts and even documents.

Wallets are protected by a master key using AES-256 and the wallet master key is protected by a user password. The wallet defines up to 7 slots that identify a password key that is able to unlock the master key. To open a wallet, it is necessary to unlock one of these 7 slots by providing the correct password. Wallet key slots are protected by the user's password and the PBKDF2-HMAC-256 algorithm, a random salt, a random counter and they are encrypted using AES-256.

Values stored in the wallet are protected by their own encryption keys using AES-256. A wallet can contain another wallet which is then protected by its own encryption keys and passwords (with 7 independent slots). Because the child wallet has its own master key, it is necessary to known the primary password and the child password to unlock the parent wallet first and then the child wallet.

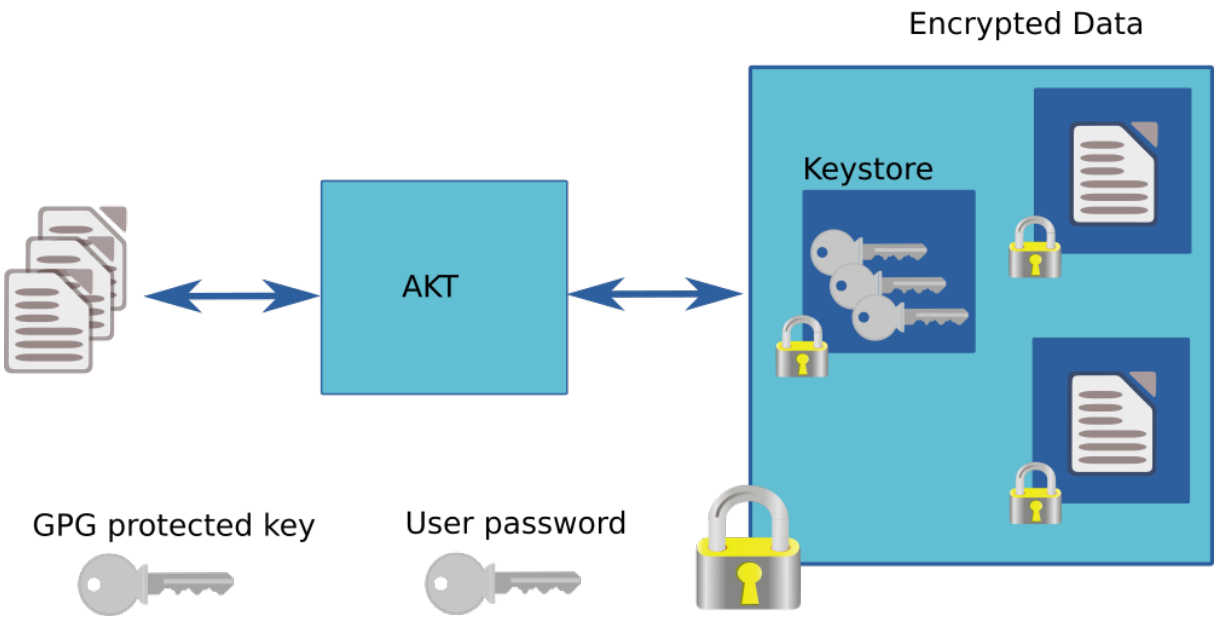

Figure 1: *AKT Overview*

The data is organized in blocks of 4K whose primary content is encrypted either by the wallet master key or by the entry keys. The data block is signed by using HMAC-256. A data block can contain several values but each of them is protected by its own encryption key. Each value is also signed using HMAC-256.

This document describes how to build the tool and library and how you can use the different features to protect your sensitive data.

2 Installation

This chapter explains how to build and install the library.

2.1 Before Building

Before building the library, you will need:

- Ada Utility Library

First get, build and install the Ada Utility Library.

On Debian systems, you may run the following installation command to get a functional GNAT Ada build environment:

```
1 sudo apt-get install -y make git gprbuild gnat-7 libxmlada-sax7-dev
```

2.2 Configuration

The library uses the `configure` script to detect the build environment, check whether Ada Utility Library is available. If some component is missing, the `configure` script will report an error or it will disable the feature. The `configure` script provides several standard options and you may use:

- `--prefix=DIR` to control the installation directory,
- `--enable-gtk` to enable building the Gtk tool,
- `--enable-shared` to enable the build of shared libraries,
- `--disable-static` to disable the build of static libraries,
- `--disable-nls` to disable NLS support,
- `--with-ada-util=PATH` to control the installation path of Ada Utility Library,
- `--with-gtkada=PATH` to control the installation path of Gtk Ada Library,
- `--help` to get a detailed list of supported options.

In most cases you will configure with the following command:

```
1 ./configure
```

On FreeBSD and NetBSD you have to disable the NLS support:

```
1 ./configure --disable-nls
```

2.3 Build

After configuration is successful, you can build the library by running:

```
make
```

After building, it is good practice to run the unit tests before installing the library. The unit tests are built and executed using:

```
make test
```

And unit tests are executed by running the `bin/keystore_harness` test program.

2.4 Installation

The installation is done by running the `install` target:

```
make install
```

If you want to install on a specific place, you can change the `prefix` and indicate the installation direction as follows:

```
make install prefix=/opt
```

2.5 Using

To use the library in an Ada project, add the following line at the beginning of your GNAT project file:

```
with "keystoreada";
```

3 Using Ada Keystore Tool

The `akt` tool is the command line tool that manages the wallet. It provides the following commands:

- `create`: create the keystore
- `edit`: edit the value with an external editor
- `get`: get a value from the keystore
- `help`: print some help
- `list`: list values of the keystore
- `remove`: remove values from the keystore
- `set`: insert or update a value in the keystore

To create the secure file, use the following command and enter your secure password (it is recommended to use a long and complex password):

```
1    akt create -k secure.akt
```

At this step, the secure file is created and it can only be opened by providing the password you entered. To add something, use:

```
1    akt set -k secure.akt bank.password 012345
```

To store a file, use the following command:

```
1    akt set -k secure.akt -f contract.doc
```

and you may also associated the file with another name with the command:

```
1    akt set -k secure.akt my-contract -f contract.doc
```

If you want to retrieve a value, you can use one of:

```
1    akt get -k secure.akt bank.password
2    akt get -k secure.akt -n my-contract > file.doc
```

You can also use the `akt` command together with the `tar` command to create secure backups. You can create the compressed tar file, pipe the result to the `akt` command to store the content in the wallet.

```
1    tar czf - dir-to-backup | akt store -k secure.akt backup.tar.gz
```

To extract the backup you can use the `extract` command and feed the result to the `tar` command as follows:

```
akt extract -k secure.akt backup.tar.gz | tar xzf -
```

4 Programmer's Guide

4.1 Keystore

The `Keystore` package provides operations to store information in secure wallets and protect the stored information by encrypting the content. It is necessary to know one of the wallet password to access its content. Wallets are protected by a master key using AES-256 and the wallet master key is protected by a user password. The wallet defines up to 7 slots that identify a password key that is able to unlock the master key. To open a wallet, it is necessary to unlock one of the 7 slots by providing the correct password. Wallet key slots are protected by the user's password and the PBKDF2-HMAC-256 algorithm, a random salt, a random counter and they are encrypted using AES-256.

4.1.1 Creation

To create a keystore you will first declare a `Wallet_File` instance. You will also need a password that will be used to protect the wallet master key.

```
1 with Keystore.Files;
2 ...
3   WS   : Keystore.Files.Wallet_File;
4   Pass : Keystore.Secret := Keystore.Create ("There was no choice but
       to be pioneers");
```

You can then create the keystore file by using the `Create` operation:

```
1   WS.Create ("secure.akt", Pass);
```

4.1.2 Storing

Values stored in the wallet are protected by their own encryption keys using AES-256. The encryption key is generated when the value is added to the wallet by using the `Add` operation.

```
1   WS.Add ("Grace Hopper", "If it's a good idea, go ahead and do it.");
```

The `Get` function allows to retrieve the value. The value is decrypted only when the `Get` operation is called.

```
1   Citation : constant String := WS.Get ("Grace Hopper");
```

The `Delete` procedure can be used to remove the value. When the value is removed, the encryption key and the data are erased.

```
1    WS.Delete ("Grace Hopper");
```

5 AKT Tool

5.1 NAME

akt - Ada Keystore Tool

5.2 SYNOPSIS

akt [-v] [-vv] [-V] [-f *file*] [-d *dir*] [-p *password*] [–password *password*] [–passfile *file*] [–passenv *name*] [–passfd *fd*] [–passask] [–passcmd *cmd*] *command*

5.3 DESCRIPTION

akt is a tool to store information in secure wallets and protect the stored information by encrypting the content. It is necessary to know one of the wallet password to access its content. *akt* can be used to safely store passwords, credentials, bank accounts and even documents.

Wallets are protected by a master key using AES-256 and the wallet master key is protected by a user password. The wallet defines up to 7 slots that identify a password key that is able to unlock the master key. To open a wallet, it is necessary to unlock one of these 7 slots by providing the correct password. Wallet key slots are protected by the user's password and the PBKDF2-HMAC-256 algorithm, a random salt, a random counter and they are encrypted using AES-256.

Values stored in the wallet are protected by their own encryption keys using AES-256. A wallet can contain another wallet which is then protected by its own encryption keys and passwords (with 7 independent slots). Because the child wallet has its own master key, it is necessary to known the primary password and the child password to unlock the parent wallet first and then the child wallet.

The data is organized in blocks of 4K whose primary content is encrypted either by the wallet master key or by the entry keys. The data block is signed by using HMAC-256. A data block can contain several values but each of them is protected by its own encryption key. Each value is also signed using HMAC-256. Large values can be written to several data blocks and in that case each fragment is encrypted by using its own encryption key.

The tool provides several commands that allow to create a keystore, insert values, retrieve values or delete them. You can use it to store your passwords, your secret keys and even your documents.

Passwords are retrieved using one of the following options:

- by reading a file that contains the password,
- by looking at an environment variable,

- by using a command line argument,
- by getting the password through the *ssh-askpass*(1) external command,
- by running an external command,
- by asking interactively the user for the password,
- by asking through a network socket for the password.

5.4 OPTIONS

The following options are recognized by *akt*:

-V Prints the *akt* version.

-v Enable the verbose mode.

-vv Enable debugging output.

-f file

Specifies the path of the keystore file to open.

-d directory

Specifies the directory path of the keystore data files. When this option is used, the data blocks are written in separate files. The data blocks do not contain the encryption keys and each of them is encrypted with its own secure key.

-p password

The keystore password is passed within the command line. Using this method is convenient but is not safe.

–passenv envname

The keystore password is passed within an environment variable with the given name. Using this method is considered safer but still provides some security leaks.

–passfile path

The keystore password is passed within a file that is read. The file must not be readable or writable by other users or group: its mode must be r??——. The directory that contains the file must also satisfy the not readable by other users or group members, This method is safer.

–passfd fd

The keystore password is passed within a pipe whose file descriptor number is given. The file descriptor is read to obtain the password. This method is safer.

–passask

The keystore password is retrieved by the running the external tool *ssh-askpass*(1) which will ask the password through either KDE, Gnome or another desktop interactive application. The password is retrieved through a pipe that *akt* sets while launching the command.

–passcmd cmd

The keystore password is retrieved by the running the external command defined in *cmd*. The command should print the password on its standard output without end of line. The password is retrieved through a pipe that *akt* sets while launching the command.

5.5 COMMANDS

5.5.1 The create command

```
akt create [--counter-range min:max] [--split count] [--gpg user]
```

Create a new keystore and protect it with the password.

The password to protect the wallet is passed using one of the following options: *–passfile* , *–passenv* , *–password* , *–passsocket* or *–gpg*. When none of these options are passed, the password is asked interactively.

The *–counter-range* option allows to control the range for the random counter used by PBKDF2 to generate the encryption key derived from the specified password. High values provide a strongest derived key at the expense of speed.

The *–split* option indicates to use several separate files for the data blocks and it controls the number of separate files to use. When used, a directory with the name of the keystore file is created and will contain the data files.

The *–gpg* option allows to protect the keystore by using a user's GPG encryption key. The option argument defines the GPG user's name or GPG key. When the keystore password is protected by the user's GPG key, a random password is generated to protect the keystore. The *gpg2*(1) command is used to encrypt that password and save it in the keystore header. The *gpg2*(1) command is then used to decrypt that and be able to unlock the keystore.

5.5.2 The set command

```
akt set name [value | -f file]
```

The *set* command is used to store a content in the wallet. The content is either passed as argument or it can be read from a file by using the *-f* option.

5.5.3 The store command

```
akt store name
```

The *store* command is intended to be used as a target for a pipe command. It reads the standard input and stores the content which is read in the wallet.

5.5.4 The remove command

```
akt remove name ...
```

The *remove* command is used to erase a content from the wallet. The data block that contained the content to protect is erased and replaced by zeros. The secure key that protected the wallet entry is also cleared. It is possible to remove several contents.

5.5.5 The edit command

```
akt edit [-e editor] name
```

The *edit* command can be used to edit the protected wallet entry by calling the user's prefered editor with the content. The content is saved in a temporary directory and in a temporary file. The editor is launched with the path and when editing is finished the temporary file is read. The temporary directory and files are erased when the editor terminates successfully or not. The editor can be specified by using the *-e* option, by setting up the *EDITOR* environment variable or by updating the *editor*(1) alternative with *update-alternative*(1). ### The list command

```
akt list
```

The *list* command describes the entries stored in the wallet.

5.5.6 The get command

```
akt get [-n] name...
```

The *get* command allows to retrieve the value associated with a wallet entry. It retrieves the value for each name passed to the command. By default a newline is emitted after each value. The *-n* option prevents the output of the trailing newline.

5.5.7 The password-add command

```
akt password-add [--new-passfile file] [--new-password password] [--new
    -passenv name]
```

The *password-add* command allows to add a new password in one of the wallet key slot. Up to seven passwords can be defined to protect the wallet. The overall security of the wallet is that of the weakest password. To add a new password, one must know an existing password.

5.5.8 The password-remove command

```
akt password-remove [--force]
```

The *password-remove* command can be used to erase a password from the wallet master key slots. Removing the last password makes the keystore unusable and it is necessary to pass the *–force* option for that.

5.5.9 The password-set command

```
akt password-set [--new-passfile file] [--new-password password] [--new
    -passenv name]
```

The *password-set* command allows to change the current wallet password.

5.6 SECURITY

Wallet master keys are protected by a derived key that is created from the user's password using *PBKDF2* and *HMAC-256* as hashing operation. When the wallet is first created, a random salt and counter are allocated which are then used by the *PBKDF2* generation. The wallet can be protected by up to 7 different passwords. Despite this, the security of the wallet master key still depends on the strength of the user's password. For this matter, it is still critical for the security to use long passphrases.

The passphrase can be passed within an environment variable or within a command line argument. These two methods are considered unsafe because it could be possible for other processes to see these values. It is best to use another method such as using the interactive form, passing the password through a file or passing using a socket based communication.

When the wallet master key is protected using *gpg2*(1) a 256-bytes random binary string is created to protect the wallet master key. This random binary string is then encrypted using the user's

–gpg option is specified only for the creation of the keystore. To unlock the keystore file, the *gpg2*(1) command will be used to decrypt the keystore header content automatically. When the user's GPG private key is not found, it is not possible to unlock the keystore with this method.

Depending on the size, a data stored in the wallet is split in one or several data entry. Each wallet data entry is then protected by their own secret key and IV vector. Wallet data entry are encrypted using AES-256-CBC. The wallet data entry key and IV vectors are protected by the wallet master key.

When the *–split* option is used, the data storage files only contain the data blocks. They do not contain any encryption key. The data storage files use the *.dkt* file extension.

5.7 SEE ALSO

editor(1), *update-alternative(1)*, *ssh-askpass(1)*, *gpg2(1)*

5.8 AUTHOR

Written by Stephane Carrez.

6 Implementation

6.1 File layouts

The data is organized in blocks of 4K. The first block is a header block used to store various information to identify the storage files. Other blocks have a clear 16-byte header and an HMAC-256 signature at the end. Blocks are encrypted either by using the master key, the directory key, the data key or a per-data fragment key.

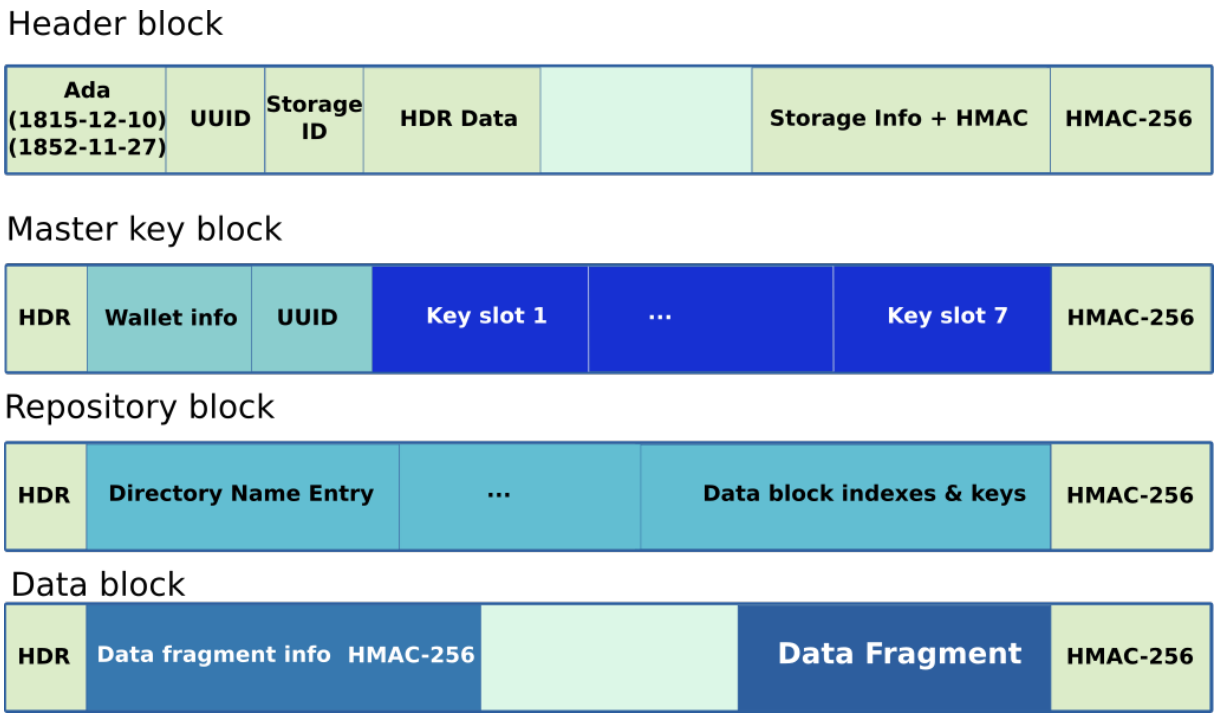

Figure 2: *Keystore blocks overview*

6.1.1 Header block

The first block of the file is the keystore header block which contains clear information signed by an HMAC header. The header block contains the keystore UUID as well as a short description of each storage data file. It also contains some optional header data.

```
+------------------+
| 41 64 61 00      | 4b = Ada
| 00 9A 72 57      | 4b = 10/12/1815
| 01 9D B1 AC      | 4b = 27/11/1852
| 00 01            | 2b = Version 1
```

```
| 00 01             | 2b = File header length in blocks
+-------------------+
| Keystore UUID     | 16b
| Storage ID        | 4b
| Block size        | 4b
| Storage count     | 4b
| Header Data count | 2b
+-------------------+-----
| Header Data size  | 2b
| Header Data type  | 2b = 0 (NONE), 1 (GPG1) 2, (GPG2)
+-------------------+
| Header Data       | Nb
+-------------------+-----
| ...               |
+-------------------+-----
| 0                 |
+-------------------+-----
| ...               |
+-------------------+-----
| Storage ID        | 4b
| Storage type      | 2b
| Storage status    | 2b  00 = open, Ada = sealed
| Storage max bloc  | 4b
| Storage HMAC      | 32b = 44b
+-------------------+----
| Header HMAC-256   | 32b
+-------------------+----
```

6.1.2 Directory Entries

The wallet repository block is encrypted with the wallet directory key.

```
+-------------------+
| 02 02             | 2b
| Encrypt size      | 2b = BT_DATA_LENGTH
| Wallet id         | 4b
| PAD 0             | 4b
| PAD 0             | 4b
+-------------------+
| Next block ID     | 4b  Block number for next repository block with
   same storage
| Data key offset   | 2b  Starts at IO.Block_Index'Last, decreasing
```

```
+------------------+
| Entry ID         | 4b   ^
| Entry type       | 2b   | = T_STRING, T_BINARY
| Name size        | 2b   |
| Name             | Nb   | DATA_NAME_ENTRY_SIZE + Name'Length
| Create date      | 8b   |
| Update date      | 8b   |
| Entry size       | 8b   v
+------------------+
| Entry ID         | 4b   ^
| Entry type       | 2b   | = T_WALLET
| Name size        | 2b   |
| Name             | Nb   | DATA_NAME_ENTRY_SIZE + Name'Length
| Create date      | 8b   |
| Update date      | 8b   |
| Wallet lid       | 4b   |
| Wallet master ID | 4b   v
+------------------+
| ...              |
+------------------+--
| 0 0 0 0          | 16b (End of name entry list)
+------------------+--
| ...              |     (random or zero)
+------------------+--
| 0 0 0 0          | 16b (End of data key list)
+------------------+--
| ...              |
+------------------+
| Storage ID       | 4b   ^ Repeats "Data key count" times
| Data block ID    | 4b   |
| Data size        | 2b   | DATA_KEY_ENTRY_SIZE = 58b
| Content IV       | 16b  |
| Content key      | 32b  v
+------------------+
| Entry ID         | 4b   ^
| Data key count   | 2b   | DATA_KEY_HEADER_SIZE = 10b
| Data offset      | 4b   v
+------------------+
| Block HMAC-256   | 32b
+------------------+--
```

6.1.3 Data Block

Data block start is encrypted with wallet data key, data fragments are encrypted with their own key. Loading and saving data blocks occurs exclusively from the workers package. The data block can be stored in a separate file so that the wallet repository and its keys are separate from the data blocks.

```
+------------------+
| 03 03            | 2b
| Encrypt size     | 2b = DATA_ENTRY_SIZE * Nb data fragment
| Wallet id        | 4b
| PAD 0            | 4b
| PAD 0            | 4b
+------------------+-----
| Entry ID         | 4b  Encrypted with wallet id
| Slot size        | 2b
| 0 0              | 2b
| Data offset      | 8b
| Content HMAC-256 | 32b => 48b = DATA_ENTRY_SIZE
+------------------+
| ...              |
+------------------+-----
| ...              |
+------------------+
| Data content     |     Encrypted with data entry key
+------------------+-----
| Block HMAC-256   | 32b
+------------------+
```