Zymkey App Utils: Python

Generated by Doxygen 1.8.13

Contents

Chapter 1

Intro

The Zymkey App Utils library provides an API which allows user space applications to incorporate Zymkey's cryptographic features, including:

- Generation of random numbers
- Locking and unlocking of data objects
- ECDSA signature generation and verification

In addition, the Zymkey App Utils library provides interfaces for administrative functions, such as:

- Control of the LED
- Setting the i2c address (i2c units only)
- Setting the tap detection sensitivity

A Note About Files

Some of the interfaces can take a filename as an argument. The following rules must be observed when using these interfaces:

- Absolute path names must be provided.
- For destination filenames, the permissions of the path (or existing file) must be set:
  - Write permissions for all.
  - Write permissions for common group: in this case, user `zymbit` must be added to the group that has permissions for the destination directory path and/or existing file.
  - Destination path must be fully owned by user and/or group `zymbit`.
- Similar rules exist for source filenames:
  - Read permissions for all.
  - Read permissions for common group: in this case, user `zymbit` must be added to the group that has permissions for the source directory path and/or existing file.
  - Source path must be fully owned by user and/or group `zymbit`.

Crypto Features

Random Number Generation

This feature is useful when the default host random number generator is suspected of having cryptographic weakness. It can also be used to supplement existing random number generation sources. Zymkey bases its random number generation on an internal TRNG (True Random Number Generator) and performs well under Fourmilab's ent.

Data Locker

Zymkey includes a feature, called Data Locking. This feature is essentially an AES encryption of the data block followed by an ECDSA signature trailer.

Data Locker Keys

In addition to a unique ECDSA private/public key pair, each Zymkey has two unique AES keys that are programmed at the factory. These keys are referred to as "one-way" and "shared":

- "one-way": the one-way key is completely self contained on the Zymkey and is never exported or changeable. Consequently, data that is locked using a Zymkey cannot be unlocked on another system (host/SD card/↩ Zymkey: See Binding).
- "shared": the shared key is used whenever the data is intended to be published to the Zymbit cloud. Using the shared key allows the Zymbit cloud to unlock the data.

ECDSA Operations

Each Zymkey comes out of the factory with a unique ECDSA private/public key pair. The private key is randomly programmed within hardware at the time of manufactor and never exported. In fact, Zymbit doesn't even know what the value of the private key is.

There are three ECDSA operations available:

- Generate signature: the Zymkey is cabable of generating an ECDSA signature.
- Verification signature: the Zymkey is capable of verifying an ECDSA signature.
- Export the ECDSA public key and saving it to a file in PEM format. This operation is useful for generating a Certificate Signing Request (CSR).

Other Features

LED

The Zymkey has an LED which can be turned on, off or flashed at an interval.

i2c Address

For Zymkeys with an i2c interface, the base address can be changed to work around addressing conflicts. The default address is 0x30, but can be changed in the ranges 0x30 - 0x37 and 0x60 - 0x67.

Tap Sensitivity

The Zymkey has an accelerometer which can perform tap detection. The sensitivity of the tap detection is configurable.

Currently tap can only be detected via the Zymbit cloud.

Programming Language Support

Currently, C, C++ and Python are supported.

Binding

Before a Zymkey can be effectively used on a host computer, it must be "bound" to it. Binding is a process where a "fingerprint" is made which is composed of the host computer and its SD card serial numbers as well as the Zymkey serial number. If the host computer or SD card is changed from the time of binding, the Zymkey will refuse to accept commands.

To learn more about binding your zymkey, go to the Zymbit Community "Getting Started"page for your Zymkey model (e.g. `Getting Started with ZYMKEY`)

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 zymkey.module.Zymkey Class Reference

Return class for Zymkey.get_accelerometer_data.

Inheritance diagram for zymkey.module.Zymkey:

5.2 zymkey.module.Zymkey.ZymkeyAccelAxisData Class Reference

Inheritance diagram for zymkey.module.Zymkey.ZymkeyAccelAxisData:

Collaboration diagram for zymkey.module.Zymkey.ZymkeyAccelAxisData:

Public Member Functions

- def **__init__** (self, g_force, tap_dir)

Public Attributes

- **g_force**
- **tap_dir**

The documentation for this class was generated from the following file:

- zymkey/module.py

Chapter 6

File Documentation

6.1 zymkey/module.py File Reference

Python interface class to Zymkey Application Utilities Library.

Classes

- class zymkey.module.Zymkey

  Return class for Zymkey.get_accelerometer_data.
- class zymkey.module.Zymkey.ZymkeyAccelAxisData

Variables

- string **zymkey.module.CLOUD_ENCRYPTION_KEY** = 'cloud'
- string **zymkey.module.ZYMKEY_ENCRYPTION_KEY** = 'zymkey'
- tuple **zymkey.module.ENCRYPTION_KEYS**
- **zymkey.module.zkalib** = None
- list **zymkey.module.prefixes** = [ ]

6.1.1 Detailed Description

Python interface class to Zymkey Application Utilities Library.

Author

Scott Miller

Version

1.0

Date

November 17, 2016

Copyright

Zymbit, Inc.

This file contains a Python class which interfaces to the the Zymkey Application Utilities library. This class facilitates writing user space applications which use Zymkey to perform cryptographic operations, such as:

1. Signing of payloads using ECDSA
2. Verification of payloads that were signed using Zymkey
3. Exporting the public key that matches Zymkey's private key
4. "Locking" and "unlocking" data objects
5. Generating random data Additionally, there are methods for changing the i2c address (i2c units only), setting tap sensitivity and controlling the LED.

6.1.2 Variable Documentation

6.1.2.1 ENCRYPTION_KEYS

`tuple zymkey.module.ENCRYPTION_KEYS`

Initial value:

```
1 =  (
2    CLOUD_ENCRYPTION_KEY,
3    ZYMKEY_ENCRYPTION_KEY
4 )
```

Index