Zymkey App Utils: Python

Generated by Doxygen 1.8.8

Fri May 12 2017 08:08:42

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.

Binding is always performed from the Zymbit Zymkey Cloud. Read about how to bind your Zymkey to its host at https://community.zymbit.com/t/zymkey-setup/79#Bind_ZymKey_to_Host

Chapter 2

Todo List

Member zymkey.module.Zymkey.sign

Allow for overloading of source parameter in similar fashion to lock/unlockData.

Member zymkey.module.Zymkey.sign_digest

Allow for overloading of source parameter in similar fashion to lock/unlockData.

Member zymkey.module.Zymkey.verify

Allow for overloading of source parameter in similar fashion to lock/unlockData.

Member zymkey.module.Zymkey.verify_digest

Allow for overloading of source parameter in similar fashion to lock/unlockData.

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

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

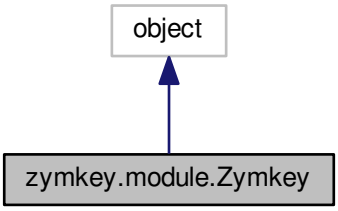
Chapter 6

Class Documentation

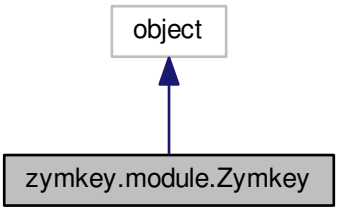
6.1 zymkey.module.Zymkey Class Reference

The Zymkey class definition.

Inheritance diagram for zymkey.module.Zymkey:



Collaboration diagram for zymkey.module.Zymkey:



Public Member Functions

- def __init__

The class initialization opens and stores an instance of a Zymkey context.

- def led_on

  Turn the LED on.
- def led_off

  Turn the LED off.
- def led_flash

  Flash the LED.
- def get_random

  Get some random bytes.
- def create_random_file

  Deposit random data in a file.
- def lock

  Lock up source (plaintext) data.
- def unlock

  Unlock source (ciphertext) data.
- def sign

  Generate a signature using the Zymkey's ECDSA private key.
- def sign_digest

  Generate a signature using the Zymkey's ECDSA private key.
- def verify

  Verify the given buffer against the given signature.
- def verify_digest

  Verify a signature using the Zymkey's ECDSA public key.
- def create_ecdsa_public_key_file

  Create a file with the PEM-formatted ECEDSA public key.
- def get_ecdsa_public_key

  Retrieves the ECEDSA public key as a binary bytearray.
- def set_i2c_address

  Sets the i2c address of the Zymkey (i2c versions only)
- def set_tap_sensitivity

  Sets the sensitivity of tap operations.
- def get_time

  Get current GMT time.
- def set_GMT_time

  Set current GMT time.

6.1.1 Detailed Description

The Zymkey class definition.

This class provides access to the Zymkey within Python

6.1.2 Member Function Documentation

6.1.2.1 def zymkey.module.Zymkey.create_ecdsa_public_key_file (*self, filename*)

Create a file with the PEM-formatted ECEDSA public key.

This method is useful for generating a Certificate Signing Request.

Parameters

filename	The absolute file path where the public key will be stored in PEM format.

6.1.2.2 def zymkey.module.Zymkey.create_random_file (*self, file_path, num_bytes*)

Deposit random data in a file.

Parameters

file_path	The absolute path name for the destination file
num_bytes	The number of random bytes to get

6.1.2.3 def zymkey.module.Zymkey.get_ecdsa_public_key (*self*)

Retrieves the ECEDSA public key as a binary bytearray.

6.1.2.4 def zymkey.module.Zymkey.get_random (*self, num_bytes*)

Get some random bytes.

Parameters

num_bytes	The number of random bytes to get

6.1.2.5 def zymkey.module.Zymkey.get_time (*self, precise* = `False`)

Get current GMT time.

This function is called to get the time directly from a Zymkey's Real Time Clock (RTC)

Parameters

precise	If true, this API returns the time after the next second falls. This means that the caller could be blocked up to one second. If false, the API returns immediately with the current time reading.

Returns

The time in seconds from the epoch (Jan. 1, 1970)

6.1.2.6 def zymkey.module.Zymkey.led_flash (*self, on_ms, off_ms* = `0`, *num_flashes* = `0`)

Flash the LED.

Parameters

on_ms	The amount of time in milliseconds that the LED will be on for
off_ms	The amount of time in milliseconds that the LED will be off for. If this parameter is set to 0 (default), the off time is the same as the on time.
num_flashes	The number of on/off cycles to execute. If this parameter is set to 0 (default), the LED flashes indefinitely.

6.1.2.7 def zymkey.module.Zymkey.lock (*self, src, dst* = `None`, *encryption_key* = `ZYMKEY_ENCRYPTION_KEY`)

Lock up source (plaintext) data.

This method encrypts and signs a block of data.

```
The zymkey has two keys that can be used for locking/unlocking
operations, designated as 'shared' and 'one-way'.
  1. The one-way key is meant to lock up data only on the
     local host computer. Data encrypted using this key cannot
     be exported and deciphered anywhere else.
  2. The shared key is meant for publishing data to other
     sources that have the capability to generate the shared
     key, such as the Zymbit cloud server.
```

Parameters

src	The source (plaintext) data. If typed as a basestring, it is assumed to be an absolute file name path where the source file is located, otherwise it is assumed to contain binary data.
dst	The destination (ciphertext) data. If specified as a basestring, it is assumed to be an absolute file name path where the destination data is meant to be deposited. Otherwise, the locked data result is returned from the method call as a bytearray. The default is 'None', which means that the data will be returned to the caller as a bytearray.
encryption_key	Specifies which key will be used to lock the data up. A value of 'zymkey' (default) specifies that the Zymkey will use the one-way key. A value of 'cloud' specifies that the shared key is used. Specify 'cloud' for publishing data to some other source that is able to derive the shared key (e.g. Zymbit cloud) and 'zymkey' when the data is meant to reside exclusively within the host computer.

6.1.2.8 def zymkey.module.Zymkey.set_GMT_time (*self*)

Set current GMT time.

This function is called to set the RTC time to the system GMT time, similar to hwclock -w

6.1.2.9 def zymkey.module.Zymkey.set_i2c_address (*self, address*)

Sets the i2c address of the Zymkey (i2c versions only)

This method should be called if the i2c address of the Zymkey is shared with another i2c device on the same i2c bus. The default i2c address for Zymkey units is 0x30. Currently, the address may be set in the ranges of 0x30 - 0x37 and 0x60 - 0x67.

After successful completion of this command, the Zymkey will reset itself.

Parameters

address	The i2c address that the Zymkey will set itself to.

6.1.2.10 def zymkey.module.Zymkey.set_tap_sensitivity (*self, axis* = `'all'`, *pct* = `50.0`)

Sets the sensitivity of tap operations.

This method permits setting the sensitivity of the tap detection feature. Each axis may be individually configured or all at once.

Parameters

axis	The axis to configure. Valid values include: 1. 'all': Configure all axes with the specified sensitivity value. 2. 'x' or 'X': Configure only the x-axis 3. 'y' or 'Y': Configure only the y-axis 4. 'z' or 'Z': Configure only the z-axis
pct	The sensitivity expressed as percentage. 1. 0% = Shut down: Tap detection should not occur along the axis. 2. 100% = Maximum sensitivity.

6.1.2.11 def zymkey.module.Zymkey.sign (*self, src*)

Generate a signature using the Zymkey's ECDSA private key.

Parameters

src	This parameter contains the digest of the data that will be used to generate the signature.

Returns

a byte array of the signature

Todo Allow for overloading of source parameter in similar fashion to lock/unlockData.

6.1.2.12 def zymkey.module.Zymkey.sign_digest (*self, sha256*)

Generate a signature using the Zymkey's ECDSA private key.

Parameters

sha256	A hashlib.sha256 instance.

Todo Allow for overloading of source parameter in similar fashion to lock/unlockData.

6.1.2.13 def zymkey.module.Zymkey.unlock (*self, src, dst* = `None`, *encryption_key* = `ZYMKEY_ENCRYPTION_KEY`)

Unlock source (ciphertext) data.

This method verifies a locked object signature and decrypts the associated ciphertext data.

The zymkey has two keys that can be used for locking/unlocking operations, designated as shared and one-way.

1. The one-way key is meant to lock up data only on the local host computer. Data encrypted using this key cannot be exported and deciphered anywhere else.
2. The shared key is meant for publishing data to other sources that have the capability to generate the shared key, such as the Zymbit cloud server.

Parameters

src	The source (ciphertext) data. If typed as a basestring, it is assumed to be an absolute file name path where the source file is located, otherwise it is assumed to contain binary data.
dst	The destination (plaintext) data. If specified as a basestring, it is assumed to be an absolute file name path where the destination data is meant to be deposited. Otherwise, the locked data result is returned from the method call as a bytearray. The default is 'None', which means that the data will be returned to the caller as a bytearray.
encryption_key	Specifies which key will be used to unlock the source data. A value of 'zymkey' (default) specifies that the Zymkey will use the one-way key. A value of 'cloud' specifies that the shared key is used. Specify 'cloud' for publishing data to another source that has the shared key (e.g. Zymbit cloud) and 'zymkey' when the data is meant to reside exclusively withing the host computer.

6.1.2.14 def zymkey.module.Zymkey.verify (*self, src, sig, raise_exception* = `True`)

Verify the given buffer against the given signature.

The public key is not specified in the parameter list to ensure that the public key that matches the Zymkey's ECDSA private key is used.

Parameters

src	The buffer to verify
sig	This parameter contains the signature to verify.
raise_exception	By default, when verification fails a VerificationError will be raised, unless this is set to False

Returns

True for a good verification or False for a bad verification when raise_exception is False

Todo Allow for overloading of source parameter in similar fashion to lock/unlockData.

6.1.2.15 def zymkey.module.Zymkey.verify_digest (*self, sha256, sig, raise_exception* = `True`)

Verify a signature using the Zymkey's ECDSA public key.

The public key is not specified in the parameter list to ensure that the public key that matches the Zymkey's ECDSA private key is used.

Parameters

sha256	A hashlib.sha256 instance that will be used to generate the signature.
sig	This parameter contains the signature to verify.
raise_exception	By default, when verification fails a VerificationError will be raised, unless this is set to False

Returns

True for a good verification or False for a bad verification when raise_exception is False

Todo Allow for overloading of source parameter in similar fashion to lock/unlockData.

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

- zymkey/module.py

Chapter 7

File Documentation

7.1 zymkey/module.py File Reference

Python interface class to Zymkey Application Utilities Library.

Classes

- class zymkey.module.Zymkey

  The Zymkey class definition.

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** = [ ]
- string **zymkey.module._zymkey_library_path** = '{}{}'

7.1.1 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.