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:

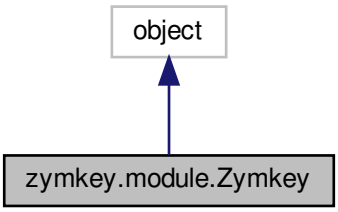
Chapter 5

Class Documentation

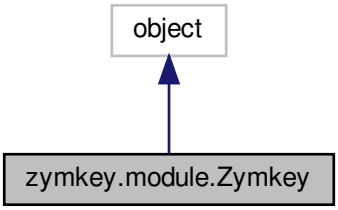
5.1 zymkey.module.Zymkey Class Reference

Return class for Zymkey.get_accelerometer_data.

Inheritance diagram for zymkey.module.Zymkey:



Collaboration diagram for zymkey.module.Zymkey:



Classes

- class ZymkeyAccelAxisData

Public Member Functions

- def __init__ (self)

  The class initialization opens and stores an instance of a Zymkey context.
- def **__del__** (self)
- def led_on (self)

  Turn the LED on.
- def led_off (self)

  Turn the LED off.
- def led_flash (self, on_ms, off_ms=0, num_flashes=0)

  Flash the LED.
- def get_random (self, num_bytes)

  Get some random bytes.
- def create_random_file (self, file_path, num_bytes)

  Deposit random data in a file.
- def lock (self, src, dst=None, encryption_key=ZYMKEY_ENCRYPTION_KEY)

  Lock up source (plaintext) data.
- def unlock (self, src, dst=None, encryption_key=ZYMKEY_ENCRYPTION_KEY, raise_exception=True)

  Unlock source (ciphertext) data.
- def sign (self, src, slot=0)

  Generate a signature using the Zymkey's ECDSA private key.
- def sign_digest (self, sha256, slot=0)

  Generate a signature using the Zymkey's ECDSA private key.
- def verify (self, src, sig, raise_exception=True, pubkey_slot=None, foreign=False)

  Verify the given buffer against the given signature.
- def verify_digest (self, sha256, sig, raise_exception=True, pubkey_slot=None, foreign=False)

  Verify a signature using the Zymkey's ECDSA public key.
- def ecdh (self, local_slot, peer_pubkey, kdf_func_type="none", salt=[ ], info=[ ], num_iterations=1, peer_↩ pubkey_slot_is_foreign=True, derived_key_size=32)

  Derive a key or a pre-master secret from an ECDH operation.
- def create_ecdsa_public_key_file (self, filename, slot=0)

  Create a file with the PEM-formatted ECDSA public key.
- def create_public_key_file (self, filename, slot=0, foreign=False)

  Create a file with the PEM-formatted public key.
- def get_ecdsa_public_key (self, slot=0)

  Retrieves the ECDSA public key as a binary bytearray.
- def get_public_key (self, slot=0, foreign=False)

  Retrieves a public key as a binary bytearray.
- def store_foreign_public_key (self, key_type, pubkey)

  Stores a foreign public key on the Zymkey foreign keyring.
- def disable_public_key_export (self, slot=0, foreign=False)

  Disables exporting of a public key at a given slot.
- def gen_key_pair (self, key_type)

  Generates a new key pair.
- def gen_ephemeral_key_pair (self, key_type)

  Generates a new ephemeral key pair.

- def remove_key (self, slot, foreign=False)

  Remove a key at the designated slot.
- def invalidate_ephemeral_key (self, slot, foreign=False)

  Invalidate the ephemeral key.
- def gen_wallet_master_seed (self, key_type, master_gen_key, wallet_name, bip39=False)

  Generates a new master seed for creating a new BIP32 wallet.
- def gen_wallet_child_key (self, parent_key_slot, index, hardened)

  Generates a child key based on a parent key that is in a wallet.
- def restore_wallet_master_seed_from_bip39_mnemonic (self, key_type, master_gen_key, wallet_name, bip39_mnemonic)

  Restore a wallet's master seed based on a BIP39 mnemonic string.
- def get_wallet_node_addr (self, slot)

  Get a wallet node address from a key slot.
- def get_wallet_key_slot (self, node_addr, wallet_name=None, master_seed_slot=None)

  Look up a wallet key slot number from a node address.
- def get_slot_alloc_list (self, foreign=False)

  Get a list of the allocated slots in the key store.
- def set_i2c_address (self, address)

  Sets the i2c address of the Zymkey (i2c versions only)
- def set_tap_sensitivity (self, axis='all', pct=50.0)

  Sets the sensitivity of tap operations.
- def get_time (self, precise=False)

  Get current GMT time.
- def wait_for_tap (self, timeout_ms=-1)

  Wait for tap event.
- def get_accelerometer_data (self)

  Get current accelerometer data and tap info.
- def wait_for_perimeter_event (self, timeout_ms=-1)

  Wait for a perimeter breach event to be detected.
- def set_perimeter_event_actions (self, channel, action_notify=True, action_self_destruct=False)

  Set perimeter breach action.
- def get_perimeter_detect_info (self)

  Get current perimeter detect info.
- def clear_perimeter_detect_info (self)

  Clear perimeter detect info.

Static Public Attributes

- int **EPHEMERAL_KEY_SLOT** = -1
- **restype**
- **argtypes**
- **rettype**

5.1.1 Detailed Description

Return class for Zymkey.get_accelerometer_data.

This class is the return type for Zymkey.get_accelerometer_data. It contains the instantaneous reading of an axis along with the direction of force that caused the latest tap event. The Zymkey class definition

This class provides access to the Zymkey within Python

5.1.2 Member Function Documentation

5.1.2.1 clear_perimeter_detect_info()

```
def zymkey.module.Zymkey.clear_perimeter_detect_info (
              self )
```

Clear perimeter detect info.

This function clears all perimeter detect info and rearms all perimeter detect channels

5.1.2.2 create_ecdsa_public_key_file()

```
def zymkey.module.Zymkey.create_ecdsa_public_key_file (
              self,
              filename,
              slot = 0 )
```

Create a file with the PEM-formatted ECDSA 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.
slot	This parameter specifies the key slot for the public key.

5.1.2.3 create_public_key_file()

```
def zymkey.module.Zymkey.create_public_key_file (
              self,
              filename,
              slot = 0,
              foreign = False )
```

Create a file with the PEM-formatted 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.
slot	This parameter specifies the key slot for the public key.

5.1.2.4 create_random_file()

```
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

5.1.2.5 disable_public_key_export()

```
def zymkey.module.Zymkey.disable_public_key_export (
              self,
              slot = 0,
              foreign = False )
```

Disables exporting of a public key at a given slot.

This method permanently disables exporting a public key from a given slot.

Parameters

slot	This parameter specifies the key slot for the public key.
foreign	If true, the slot refers to the foreign public keyring.

5.1.2.6 ecdh()

```
def zymkey.module.Zymkey.ecdh (
              self,
              local_slot,
              peer_pubkey,
              kdf_func_type = "none",
              salt = [],
              info = [],
              num_iterations = 1,
              peer_pubkey_slot_is_foreign = True,
              derived_key_size = 32 )
```

Derive a key or a pre-master secret from an ECDH operation.

Parameters

local_slot	This parameter specifies the local key slot to use.
peer_pubkey	
kdf_func_type	
salt	
info	

Returns

a byte array of the signature

5.1.2.7 gen_ephemeral_key_pair()

```
def zymkey.module.Zymkey.gen_ephemeral_key_pair (
              self,
              key_type )
```

Generates a new ephemeral key pair.

This method generates a new ephemeral key pair of the specified type, overwriting the previous ephemeral key pair.

Parameters

key_type	This parameter indicates the EC curve type that should be associated with the new key pair.

Returns

the slot allocated to the key or less than one for failure.

5.1.2.8 gen_key_pair()

```
def zymkey.module.Zymkey.gen_key_pair (
              self,
              key_type )
```

Generates a new key pair.

This method generates a new key pair of the specified type.

Parameters

key_type	This parameter indicates the EC curve type that should be associated with the new key pair.

Returns

the slot allocated to the key or less than one for failure.

5.1.2.9 gen_wallet_child_key()

```
def zymkey.module.Zymkey.gen_wallet_child_key (
              self,
              parent_key_slot,
              index,
              hardened )
```

Generates a child key based on a parent key that is in a wallet.

This method generates a child key based on a parent key that is in a wallet.

Parameters

parent_key_slot	This parameter specifies the parent key slot. This key must already be part of a wallet.
index	This parameter represents the index for the child key derivation which becomes part of the node address.
hardened	If true, the key is a hardened key.

Returns

the allocated slot on success

5.1.2.10 gen_wallet_master_seed()

```
def zymkey.module.Zymkey.gen_wallet_master_seed (
              self,
              key_type,
              master_gen_key,
              wallet_name,
              bip39 = False )
```

Generates a new master seed for creating a new BIP32 wallet.

This method generates a new master seed for creating a new BIP32 wallet.

Parameters

key_type	This parameter indicates the EC curve type that should be associated with the new key pair.
master_gen_key	The master generator key used in the derivation of the child key.
wallet_name	The name of the wallet that this master seed is attached to
bip39	If true, generate the seed per BIP39 and return the mnemonic string.

Returns

a tuple with the slot and the BIP39 mnemonic if specified

5.1.2.11 get_accelerometer_data()

```
def zymkey.module.Zymkey.get_accelerometer_data (
              self )
```

Get current accelerometer data and tap info.

This function gets the most recent accelerometer data in units of g forces plus the tap direction per axis.

Parameters

x	(output) An array of accelerometer readings in units of g-force. array index 0 = x axis 1 = y axis 2 = z axis tap_dir (output) The directional information for the last tap event. A value of -1 indicates that the tap event was detected in a negative direction for the axis, +1 for a positive direction and 0 for stationary.

5.1.2.12 get_ecdsa_public_key()

```
def zymkey.module.Zymkey.get_ecdsa_public_key (
              self,
              slot = 0 )
```

Retrieves the ECDSA public key as a binary bytearray.

This method is used to retrieve the public key in binary form.

Parameters

slot	This parameter specifies the key slot for the public key.

5.1.2.13 get_perimeter_detect_info()

```
def zymkey.module.Zymkey.get_perimeter_detect_info (
              self )
```

Get current perimeter detect info.

This function gets the timestamp of the first perimeter detect event for the given channel

Returns

The array of timestamps for each channel for the first detected event in epoch seconds

5.1.2.14 get_public_key()

```
def zymkey.module.Zymkey.get_public_key (
              self,
              slot = 0,
              foreign = False )
```

Retrieves a public key as a binary bytearray.

This method is used to retrieve the public key in binary form.

Parameters

slot	This parameter specifies the key slot for the public key.

5.1.2.15 get_random()

```
def zymkey.module.Zymkey.get_random (
              self,
              num_bytes )
```

Get some random bytes.

Parameters

num_bytes	The number of random bytes to get

5.1.2.16 get_slot_alloc_list()

```
def zymkey.module.Zymkey.get_slot_alloc_list (
              self,
              foreign = False )
```

Get a list of the allocated slots in the key store.

This method gets a list of the allocated slots in the key store.

Parameters

foreign	If True, the allocation list of the foreign key store is returned

Returns

the allocation list and the maximum number of keys

5.1.2.17 get_time()

```
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)

5.1.2.18 get_wallet_key_slot()

```
def zymkey.module.Zymkey.get_wallet_key_slot (
              self,
              node_addr,
              wallet_name = None,
              master_seed_slot = None )
```

Look up a wallet key slot number from a node address.

This method gets a wallet key slot number from its node address and wallet name or master seed key slot. Either the wallet name or the master seed slot must be present.

Parameters

node_addr	The desired node address to look up
wallet_name	The name of the wallet that the node address belongs to. Either this parameter or master_seed_slot must be specified or this function will fail.
master_seed_slot	The master seed slot that the node address belongs to. Either this parameter or wallet_name must be specified or this function will fail.

Returns

the key slot.

5.1.2.19 get_wallet_node_addr()

```
def zymkey.module.Zymkey.get_wallet_node_addr (
              self,
              slot )
```

Get a wallet node address from a key slot.

This method gets a wallet entry's node address from its key slot assignment. The wallet name and master seed slot are also returned.

Parameters

slot	The key slot assignment.

Returns

the node address, wallet name and master seed key slot.

5.1.2.20 invalidate_ephemeral_key()

```
def zymkey.module.Zymkey.invalidate_ephemeral_key (
              self,
              slot,
              foreign = False )
```

Invalidate the ephemeral key.

This method invalidates the ephemeral key, effectively removing it from service until a new key is generated.

5.1.2.21 led_flash()

```
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.

5.1.2.22 lock()

```
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.

5.1.2.23 remove_key()

```
def zymkey.module.Zymkey.remove_key (
              self,
              slot,
              foreign = False )
```

Remove a key at the designated slot.

This method removes a key at the designated slot in either the standard key store or the foreign public keyring.

Parameters

slot	This parameter specifies the key slot for the key.
foreign	If true, a public key in the foreign keyring will be deleted.

5.1.2.24 restore_wallet_master_seed_from_bip39_mnemonic()

```
def zymkey.module.Zymkey.restore_wallet_master_seed_from_bip39_mnemonic (
              self,
              key_type,
              master_gen_key,
              wallet_name,
              bip39_mnemonic )
```

Restore a wallet's master seed based on a BIP39 mnemonic string.

This method restores a wallet's master seed based on a BIP39 mnemonic string and a master generator key. This method can be used in the process of wallet duplication.

Parameters

key_type	This parameter indicates the EC curve type that should be associated with the new key pair.
master_gen_key	The master generator key used in the derivation of the child key.
bip39_mnemonic	The BIP39 mnemonic string.

Returns

the allocated slot on success

5.1.2.25 set_i2c_address()

```
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.

5.1.2.26 set_perimeter_event_actions()

```
def zymkey.module.Zymkey.set_perimeter_event_actions (
              self,
              channel,
              action_notify = True,
              action_self_destruct = False )
```

Set perimeter breach action.

This function specifies the action to take when a perimeter breach event occurs. The possible actions are any combination of:

1. Notify host
2. Zymkey self-destruct

Parameters

channel	(input) The channel that the action flags will be applied to action_flags (input) The actions to apply to the perimeter event channel: (a) Notify (ZK_PERIMETER_EVENT_ACTION_NOTIFY) (b) Self-destruct (ZK_PERIMETER_EVENT_ACTION_SELF_DESTRUCT)

5.1.2.27 set_tap_sensitivity()

```
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.

5.1.2.28 sign()

```
def zymkey.module.Zymkey.sign (
              self,
              src,
              slot = 0 )
```

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.
slot	This parameter specifies the key slot used for signing.

Returns

a byte array of the signature

5.1.2.29 sign_digest()

```
def zymkey.module.Zymkey.sign_digest (
              self,
              sha256,
              slot = 0 )
```

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

Parameters

sha256	A hashlib.sha256 instance.
slot	This parameter specifies the key slot used for signing.

5.1.2.30 store_foreign_public_key()

```
def zymkey.module.Zymkey.store_foreign_public_key (
              self,
              key_type,
              pubkey )
```

Stores a foreign public key on the Zymkey foreign keyring.

This method stores a foreign public key onto the Zymkey foreign public keyring.

Parameters

key_type	This parameter indicates the EC curve type that should be associated with the public key
pubkey	The public key binary data

Returns

the slot allocated to the key or less than one for failure.

5.1.2.31 unlock()

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

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.
raise_exception	Specifies if an exception should be raised if the locked object signature fails.

5.1.2.32 verify()

```
def zymkey.module.Zymkey.verify (
                self,
                src,
                sig,
                raise_exception = True,
                pubkey_slot = None,
                foreign = False )
```

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
pubkey_slot	The key slot to use to verify the signature against. Defaults to the first key slot.

Returns

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

5.1.2.33 verify_digest()

```
def zymkey.module.Zymkey.verify_digest (
                self,
                sha256,
                sig,
                raise_exception = True,
                pubkey_slot = None,
                foreign = False )
```

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
pubkey_slot	The key slot to use to verify the signature against. Defaults to the first key slot.

Returns

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

5.1.2.34 wait_for_perimeter_event()

```
def zymkey.module.Zymkey.wait_for_perimeter_event (
              self,
              timeout_ms = -1 )
```

Wait for a perimeter breach event to be detected.

This function is called in order to wait for a perimeter breach event to occur. This function blocks the calling thread unless called with a timeout of zero.

Parameters

timeout_ms	(input) The maximum amount of time in milliseconds to wait for a tap event to arrive.

5.1.2.35 wait_for_tap()

```
def zymkey.module.Zymkey.wait_for_tap (
              self,
              timeout_ms = -1 )
```

Wait for tap event.

Wait for a tap event to be detected

This function is called in order to wait for a tap event to occur. This function blocks the calling thread unless called with a timeout of zero.

Parameters

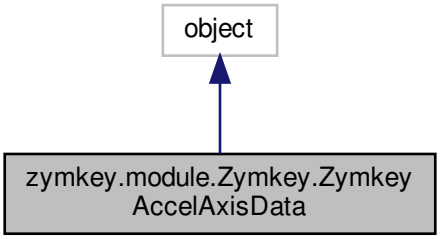
timeout_ms	(input) The maximum amount of time in milliseconds to wait for a tap event to arrive.

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

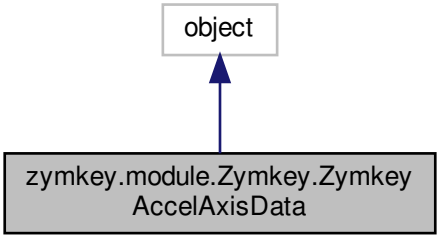
- zymkey/module.py

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**
- dictionary **zymkey.module.keyTypes**
- dictionary **zymkey.module.kdfFuncTypes**
- **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 )
```

6.1.2.2 kdfFuncTypes

```
dictionary zymkey.module.kdfFuncTypes
```

Initial value:

```
1 =  {
2     "none"           : 0,
3     "rfc5869-sha256" : 1,
4     "rfc5869-sha512" : 2,
5     "pbkdf2-sha256"  : 3,
6     "pbkdf2-sha512"  : 4
7 }
```

6.1.2.3 keyTypes

```
dictionary zymkey.module.keyTypes
```

Initial value:

```
1 =  {
2     "secp256r1" : 0,
3     "nistp256"  : 0,
4     "secp256k1" : 1
5 }
```

Index