



Zymkey App Utils

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Intro</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>5</b>
2.1	Data Structures . . . . .	5
<b>3</b>	<b>File Index</b>	<b>7</b>
3.1	File List . . . . .	7
<b>4</b>	<b>Data Structure Documentation</b>	<b>9</b>
4.1	zkAccelAxisDataType Struct Reference . . . . .	9
4.1.1	Detailed Description . . . . .	9
4.1.2	Field Documentation . . . . .	9
4.1.2.1	g . . . . .	9
4.1.2.2	tapDirection . . . . .	9
<b>5</b>	<b>File Documentation</b>	<b>11</b>
5.1	zk_app_utils.h File Reference . . . . .	11
5.1.1	Detailed Description . . . . .	16
5.1.2	Function Documentation . . . . .	16
5.1.2.1	zkClearPerimeterDetectEvents() . . . . .	16
5.1.2.2	zkClose() . . . . .	16
5.1.2.3	zkCreateRandDataFile() . . . . .	17
5.1.2.4	zkDisablePubKeyExport() . . . . .	17
5.1.2.5	zkDoECDHAndKDF() . . . . .	18
5.1.2.6	zkDoECDHAndKDFWithIntPeerPubkey() . . . . .	18

5.1.2.7	zkDoRawECDH()	19
5.1.2.8	zkDoRawECDHWithIntPeerPubkey()	20
5.1.2.9	zkExportPubKey()	20
5.1.2.10	zkExportPubKey2File()	21
5.1.2.11	zkGenECDSASigFromDigest()	21
5.1.2.12	zkGenEphemeralKeyPair()	22
5.1.2.13	zkGenKeyPair()	22
5.1.2.14	zkGenWalletChildKey()	23
5.1.2.15	zkGenWalletMasterSeed()	23
5.1.2.16	zkGetAccelerometerData()	24
5.1.2.17	zkGetAllocSlotsList()	25
5.1.2.18	zkGetBatteryVoltage()	25
5.1.2.19	zkGetCPUTemp()	26
5.1.2.20	zkGetCurrentBindingInfo()	26
5.1.2.21	zkGetECDSAPubKey()	26
5.1.2.22	zkGetFirmwareVersionString()	27
5.1.2.23	zkGetModelNumberString()	27
5.1.2.24	zkGetPerimeterDetectInfo()	28
5.1.2.25	zkGetRandBytes()	28
5.1.2.26	zkGetRTCDrift()	29
5.1.2.27	zkGetSerialNumberString()	29
5.1.2.28	zkGetTime()	29
5.1.2.29	zkGetWalletKeySlotFromNodeAddr()	30
5.1.2.30	zkGetWalletNodeAddrFromKeySlot()	30
5.1.2.31	zkInvalidateEphemeralKey()	31
5.1.2.32	zkLEDFlash()	31
5.1.2.33	zkLEDOff()	32
5.1.2.34	zkLEDOn()	32
5.1.2.35	zkLockBinding()	33
5.1.2.36	zkLockDataB2B()	33

5.1.2.37	zkLockDataB2F()	34
5.1.2.38	zkLockDataF2B()	34
5.1.2.39	zkLockDataF2F()	35
5.1.2.40	zkOpen()	36
5.1.2.41	zkRemoveKey()	36
5.1.2.42	zkRestoreWalletMasterSeedFromMnemonic()	36
5.1.2.43	zkSaveECDSAPubKey2File()	37
5.1.2.44	zkSetBatteryVoltageAction()	38
5.1.2.45	zkSetBatteryVoltageThreshold()	38
5.1.2.46	zkSetCPUHighTempThreshold()	39
5.1.2.47	zkSetCPULowTempThreshold()	39
5.1.2.48	zkSetCPUTempAction()	39
5.1.2.49	zkSetDigitalPerimeterDetectDelays()	40
5.1.2.50	zkSetDigitalPerimeterDetectLPMaxBits()	40
5.1.2.51	zkSetDigitalPerimeterDetectLPPeriod()	41
5.1.2.52	zkSetI2CAddr()	41
5.1.2.53	zkSetPerimeterEventAction()	42
5.1.2.54	zkSetTapSensitivity()	42
5.1.2.55	zkStoreForeignPubKey()	43
5.1.2.56	zkUnlockDataB2B()	43
5.1.2.57	zkUnlockDataB2F()	44
5.1.2.58	zkUnlockDataF2B()	45
5.1.2.59	zkUnlockDataF2F()	45
5.1.2.60	zkVerifyECDSASigFromDigest()	46
5.1.2.61	zkVerifyECDSASigFromDigestWithForeignKeySlot()	46
5.1.2.62	zkWaitForPerimeterEvent()	47
5.1.2.63	zkWaitForTap()	47



# 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 Fournilab'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.

### ECDSA Operations

Each Zymkey comes out of the factory with three (Zymkey 4i/HSM4) or 12 (HSM6) unique ECDSA private/public key pairs. These private keys are randomly programmed within hardware at the time of manufacture and never exported. In fact, Zymbit doesn't even know what the value of the private keys are.

There are three ECDSA operations available:

- Generate signature: the Zymkey is capable 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.

## 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](#))

## HSM4

### Soft Bind Lock

Unlike Zymkey, HSM4 does not have a physical lock tab that is cut to lock the host binding. Instead, it has a software API for locking the binding to the host. See the soft bind lock API description for more information.

## HSM6

### Soft Bind Lock

See "HSM4: Soft Bind Lock"

### ECDH

ECDH with some Key Derivation Functions (KDFs) is available. The available KDFs include:

1. none: the raw pre-master secret is returned
2. rfc5869: this KDF can be invoked with SHA256 or SHA512 as the hashing function
3. pbkdf2: like rfc5869, SHA256 or SHA512 can be specified

### Koblitz Curve Support

In addition to NIST P-256 which is available on Zymkey and HSM4, the Koblitz 256 bit curve is available as well

### More Key Slots

HSM6 now has 12 factory pre-configured NIST P-256 key slots and 512 key pairs for generated key pair and digital wallet allocation for NIST P-256 or Koblitz

1. All of these key slots can be used for ECDSA as well as ECDH operations. It is also possible to remove the keys in the 512 key pair store and also prohibit public key export.

### Foreign Public Keyring

128 public key slots are available to form a public key store for storing foreign party public keys. Like the 512 key pair store, public keys can be removed and prohibited from export.

### Digital Wallet

HSM6 provides APIs which allow the creation of 1 or more digital wallets per BIP 32/39/44. The number of master seeds and child nodes is only limited by the number of key pairs provided for in the 512 byte key pair store. The BIP 39 feature provides the mnemonic sentence using the 2048 word English dictionary.

### Battery and Temperature Monitoring

The HSM6 has APIs for reading the battery voltage as well as monitoring policies for taking action if certain thresholds are exceeded.

#### Battery Monitoring

The Battery Monitoring feature provides for 1 of 2 actions if the battery voltage below specified threshold (settable from 2.5V or lower via API).

1. self-destruct: if the battery voltage goes lower than the threshold, destroy all key material. Good for use cases where a low battery voltage is interpreted as an attempt to attack the low power tamper detect operation.
2. sleep-until-battery-recovery: do not permit full functionality until the battery has gone above the threshold. Good for use cases where battery replacement is desirable.

#### Temperature Monitoring

The Temperature Monitoring feature provides protection from attacks that are based on manipulation of temperature (e.g. memory freeze attacks). If invoked, the system may be configured to destroy keys if temperature thresholds are exceeded.

## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">zkAccelAxisDataType</a>	ZkGetAccelerometer data output . . . . .	9
-------------------------------------	--	---



# Chapter 3

## File Index

### 3.1 File List

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

<a href="#">zk_app_utils.h</a>	C interface to Zymkey Application Utilities Library . . . . .	<a href="#">11</a>
--------------------------------	---	--------------------



## Chapter 4

# Data Structure Documentation

## 4.1 zkAccelAxisDataType Struct Reference

zkGetAccelerometer data output.

```
#include <zk_app_utils.h>
```

### Data Fields

- double g
- int tapDirection

#### 4.1.1 Detailed Description

zkGetAccelerometer data output.

#### 4.1.2 Field Documentation

##### 4.1.2.1 g

```
double zkAccelAxisDataType::g
```

the axis reading in units of g-force

##### 4.1.2.2 tapDirection

```
int zkAccelAxisDataType::tapDirection
```

the direction of the force along the axis which caused a tap event: -1 = negative +1 = positive 0 = did not cause a tap event

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

- [zk\\_app\\_utils.h](#)



# Chapter 5

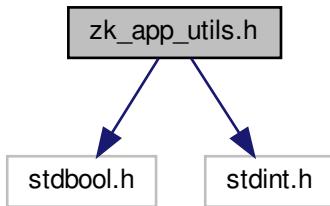
## File Documentation

### 5.1 zk\_app\_utils.h File Reference

C interface to Zymkey Application Utilities Library.

```
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for zk\_app\_utils.h:



#### Data Structures

- struct `zkAccelAxisDataType`  
*zkGetAccelerometer data output.*

#### Macros

- #define `ZK_PERIMETER_EVENT_ACTION_NOTIFY` (1 << 0)  
*Perimeter breach action flag definitions.*
- #define `ZK_PERIMETER_EVENT_ACTION_SELF_DESTRUCT` (1 << 1)

## Typedefs

- `typedef void * zkCTX`
- `typedef enum ZK_EC_KEY_TYPE ZK_EC_KEY_TYPE`  
*Supported key types.*
- `typedef enum ZK_ECDH_KDF_TYPE ZK_ECDH_KDF_TYPE`  
*Supported ECDH key derivation function types.*
- `typedef enum ZK_ACCEL_AXIS_TYPE ZK_ACCEL_AXIS_TYPE`  
*Accelerometer axis enum, used to set tap sensitivity.*
- `typedef enum ZK_THRESHOLD_ACTION_TYPE ZK_THRESHOLD_ACTION_TYPE`  
*Possible actions for threshold monitor functions.*
- `typedef struct zkAccelAxisDataType zkAccelAxisDataType`  
*zkGetAccelerometer data output.*

## Enumerations

- `enum ZK_EC_KEY_TYPE { ZK_NISTP256, ZK_SECP256R1 = ZK_NISTP256, ZK_SECP256K1 }`  
*Supported key types.*
- `enum ZK_ECDH_KDF_TYPE { ZK_KDF_RFC5869_SHA256, ZK_KDF_RFC5869_SHA512, ZK_KDF_PBKDF2_SHA256, ZK_KDF_PBKDF2_SHA512 }`  
*Supported ECDH key derivation function types.*
- `enum ZK_ACCEL_AXIS_TYPE { ZK_ACCEL_AXIS_X, ZK_ACCEL_AXIS_Y, ZK_ACCEL_AXIS_Z, ZK_ACCEL_AXIS_ALL }`  
*Accelerometer axis enum, used to set tap sensitivity.*
- `enum ZK_THRESHOLD_ACTION_TYPE { ZK_ACTION_NONE, ZK_ACTION_SELF_DESTRUCT, ZK_ACTION_SLEEP }`  
*Possible actions for threshold monitor functions.*

## Functions

### Zymkey Context

- `int zkOpen (zkCTX *ctx)`  
*Open a Zymkey context.*
- `int zkClose (zkCTX ctx)`  
*Close a Zymkey context.*

### Random Number Generation

- `int zkCreateRandDataFile (zkCTX ctx, const char *dst_filename, int rdata_sz)`  
*Fill a file with random numbers.*
- `int zkGetRandBytes (zkCTX ctx, uint8_t **rdata, int rdata_sz)`  
*Get an array of random bytes.*

### Lock Data

- `int zkLockDataF2F (zkCTX ctx, const char *src_pt_filename, const char *dst_ct_filename, bool use_shared_key)`  
*Lock up source (plaintext) data from a file and store the results (ciphertext) in a destination file.*
- `int zkLockDataB2F (zkCTX ctx, const uint8_t *src_pt, int src_pt_sz, const char *dst_ct_filename, bool use_shared_key)`  
*Lock up source (plaintext) data from a byte array and store the results (ciphertext) in a destination file.*

- int `zkLockDataF2B` (zkCTX ctx, const char \*src\_pt\_filename, uint8\_t \*\*dst\_ct, int \*dst\_ct\_sz, bool use\_shared\_key)
 

*Lock up source (plaintext) data from a file and store the results (ciphertext) in a destination byte array.*
- int `zkLockDataB2B` (zkCTX ctx, const uint8\_t \*src\_pt, int src\_pt\_sz, uint8\_t \*\*dst\_ct, int \*dst\_ct\_sz, bool use\_shared\_key)
 

*Lock up source (plaintext) data from a byte array and store the results (ciphertext) in a destination byte array.*

## Unlock Data

- int `zkUnlockDataF2F` (zkCTX ctx, const char \*src\_ct\_filename, const char \*dst\_pt\_filename, bool use\_shared\_key)
 

*Unlock source (ciphertext) data from a file and store the results (plaintext) in a destination file.*
- int `zkUnlockDataB2F` (zkCTX ctx, const uint8\_t \*src\_ct, int src\_ct\_sz, const char \*dst\_pt\_filename, bool use\_shared\_key)
 

*Unlock source (ciphertext) data from a byte array and store the results (plaintext) in a destination file.*
- int `zkUnlockDataF2B` (zkCTX ctx, const char \*src\_ct\_filename, uint8\_t \*\*dst\_pt, int \*dst\_pt\_sz, bool use\_shared\_key)
 

*Unlock source (ciphertext) data from a file and store the results (plaintext) in a destination byte array.*
- int `zkUnlockDataB2B` (zkCTX ctx, const uint8\_t \*src\_ct, int src\_ct\_sz, uint8\_t \*\*dst\_pt, int \*dst\_pt\_sz, bool use\_shared\_key)
 

*Unlock source (ciphertext) data from a byte array and store the results (plaintext) in a destination byte array.*

## ECDSA

- int `zkGenECDSASigFromDigest` (zkCTX ctx, const uint8\_t \*digest, int slot, uint8\_t \*\*sig, int \*sig\_sz)
 

*Generate a signature using the Zymkey's ECDSA private key.*
- int `zkVerifyECDSASigFromDigest` (zkCTX ctx, const uint8\_t \*digest, int pubkey\_slot, const uint8\_t \*sig, int sig\_sz)
 

*Verify a signature using one of the Zymkey's public keys.*
- int `zkVerifyECDSASigFromDigestWithForeignKeySlot` (zkCTX ctx, const uint8\_t \*digest, int pubkey\_slot, const uint8\_t \*sig, int sig\_sz)
 

*Verify a signature using one of the Zymkey's foreign public keys.*

## ECDH and KDF

- int `zkDoRawECDH` (zkCTX ctx, int slot, const uint8\_t \*peer\_pubkey, int peer\_pubkey\_sz, uint8\_t \*\*pre\_master\_secret)
 

*Perform a raw ECDH operation. (model >= HSM6)*
- int `zkDoRawECDHWithIntPeerPubkey` (zkCTX ctx, int slot, int peer\_pubkey\_slot, bool peer\_pubkey\_slot\_is\_foreign, uint8\_t \*\*pre\_master\_secret)
 

*Perform a raw ECDH operation. (model >= HSM6)*
- int `zkDoECDHAndKDF` (zkCTX ctx, `ZK_ECDH_KDF_TYPE` kdf\_type, int slot, const uint8\_t \*peer\_pubkey, int peer\_pubkey\_sz, const uint8\_t \*salt, int salt\_sz, const uint8\_t \*info, int info\_sz, int num\_iterations, int derived\_key\_sz, uint8\_t \*\*derived\_key)
 

*Perform an ECDH operation plus Key Derivation Function. (model >= HSM6)*
- int `zkDoECDHAndKDFWithIntPeerPubkey` (zkCTX ctx, `ZK_ECDH_KDF_TYPE` kdf\_type, int slot, int peer\_pubkey\_slot, bool peer\_pubkey\_slot\_is\_foreign, const uint8\_t \*salt, int salt\_sz, const uint8\_t \*info, int info\_sz, int num\_iterations, int derived\_key\_sz, uint8\_t \*\*derived\_key)
 

*Perform an ECDH operation plus Key Derivation Function. (model >= HSM6)*

## Key Management

- int `zkSaveECDSAPubKey2File` (zkCTX ctx, const char \*filename, int slot)
 

*[DEPRECATED] Use `zkExportPubKey2File`. Store the public key to a host file in PEM format.*
- int `zkExportPubKey2File` (zkCTX ctx, const char \*filename, int pubkey\_slot, bool slot\_is\_foreign)
 

*Store the public key to a host file in PEM format.*

- int `zkGetECDSPubKey` (zkCTX ctx, uint8\_t \*\*pk, int \*pk\_sz, int slot)
 

*[DEPRECATED] Use zkExportPubKey. Gets the public key and stores in a byte array created by this function.*
- int `zkExportPubKey` (zkCTX ctx, uint8\_t \*\*pk, int \*pk\_sz, int pubkey\_slot, bool slot\_is\_foreign)
 

*Gets the public key and stores in a byte array created by this function.*
- int `zkGetAllocSlotsList` (zkCTX ctx, bool is\_foreign, int \*max\_num\_keys, int \*\*alloc\_key\_list, int \*alloc\_key\_list\_sz)
 

*Get the list of allocated keys (model >= HSM6).*
- int `zkStoreForeignPubKey` (zkCTX ctx, ZK\_EC\_KEY\_TYPE pk\_type, uint8\_t \*pk, int pk\_sz)
 

*Store a new foreign public key in Zymkey. (model >= HSM6)*
- int `zkDisablePubKeyExport` (zkCTX ctx, int pubkey\_slot, bool slot\_is\_foreign)
 

*Prevent a public key from being exported to the host. (model >= HSM6)*
- int `zkGenKeyPair` (zkCTX ctx, ZK\_EC\_KEY\_TYPE type)
 

*Generate a new persistent key pair. (model >= HSM6)*
- int `zkGenEphemeralKeyPair` (zkCTX ctx, ZK\_EC\_KEY\_TYPE type)
 

*Generate an ephemeral key pair. (model >= HSM6)*
- int `zkRemoveKey` (zkCTX ctx, int slot, bool slot\_is\_foreign)
 

*Remove a key pair or a foreign public key. (model >= HSM6)*
- int `zkInvalidateEphemeralKey` (zkCTX ctx)
 

*Invalidate the ephemeral key. (model >= HSM6)*

### Digital Wallet (BIP32/39/44)

- int `zkGenWalletMasterSeed` (zkCTX ctx, ZK\_EC\_KEY\_TYPE type, const char \*wallet\_name, const uint8\_t \*master\_generator\_key, int master\_generator\_key\_size, char \*\*bip39\_mnemonic)
 

*Generate master seed to start a new blockchain wallet. (model >= HSM6)*
- int `zkGenWalletChildKey` (zkCTX ctx, int parent\_key\_slot, uint32\_t index, bool is\_hardened)
 

*Generate child key from a parent key in a blockchain wallet. (model >= HSM6)*
- int `zkRestoreWalletMasterSeedFromMnemonic` (zkCTX ctx, ZK\_EC\_KEY\_TYPE type, const char \*wallet\_name, const uint8\_t \*master\_generator\_key, int master\_generator\_key\_size, char \*bip39\_mnemonic)
 

*Restore a master seed from a BIP39 mnemonic and a master generator key. (model >= HSM6)*
- int `zkGetWalletNodeAddrFromKeySlot` (zkCTX ctx, int slot, char \*\*node\_addr, char \*\*wallet\_name, int \*master\_seed\_slot)
 

*Derive the node address from a key slot number. (model >= HSM6)*
- int `zkGetWalletKeySlotFromNodeAddr` (zkCTX ctx, const char \*node\_addr, const char \*wallet\_name, int master\_seed\_slot, int \*slot)
 

*Derive the slot address from a node address. (model >= HSM6)*

### Perimeter Detect

- int `zkSetPerimeterEventAction` (zkCTX ctx, int channel, uint32\_t action\_flags)
 

*Set perimeter breach action.*
- int `zkSetDigitalPerimeterDetectLPPeriod` (zkCTX ctx, int lp\_period)
 

*Set the low power period (model >= HSM6).*
- int `zkSetDigitalPerimeterDetectLPMaxBits` (zkCTX ctx, int max\_num\_bits)
 

*Set the low power max number of bits (model >= HSM6).*
- int `zkSetDigitalPerimeterDetectDelays` (zkCTX ctx, int min\_delay\_ns, int max\_delay\_ns)
 

*Set the delays (model >= HSM6).*
- int `zkWaitForPerimeterEvent` (zkCTX ctx, uint32\_t timeout\_ms)
 

*Wait for a perimeter breach event to be detected.*
- int `zkGetPerimeterDetectInfo` (zkCTX ctx, uint32\_t \*\*timestamps\_sec, int \*num\_timestamps)
 

*Get current perimeter detect info.*
- int `zkClearPerimeterDetectEvents` (zkCTX ctx)
 

*Clear perimeter detect events.*

### LED Control

- int `zkLEDOff` (zkCTX ctx)  
*Turns the LED off.*
- int `zkLEDOn` (zkCTX ctx)  
*Turns the LED on.*
- int `zkLEDFlash` (zkCTX ctx, uint32\_t on\_ms, uint32\_t off\_ms, uint32\_t num\_flashes)  
*Flashes the LED.*

## Administrative Ops

- int `zkSetI2CAddr` (zkCTX ctx, int addr)  
*Sets the i2c address of the Zymkey (i2c versions only)*

## Time

- int `zkGetTime` (zkCTX ctx, uint32\_t \*epoch\_time\_sec, bool precise\_time)  
*Get current GMT time.*

## Accelerometer

- int `zkSetTapSensitivity` (zkCTX ctx, int axis, float pct)  
*Sets the sensitivity of tap operations.*
- int `zkWaitForTap` (zkCTX ctx, uint32\_t timeout\_ms)  
*Wait for a tap event to be detected.*
- int `zkGetAccelerometerData` (zkCTX ctx, zkAccelAxisDataType \*x, zkAccelAxisDataType \*y, zkAccelAxisDataType \*z)  
*Get current accelerometer data and tap info.*

## Binding Management

- int `zkLockBinding` (zkCTX ctx)  
*Set soft binding lock.*
- int `zkGetCurrentBindingInfo` (zkCTX ctx, bool \*binding\_is\_locked, bool \*is\_bound)  
*Get current binding info.*

## Module Info

- int `zkGetModelNumberString` (zkCTX ctx, char \*\*model\_str)  
*Get Zymkey model number.*
- int `zkGetFirmwareVersionString` (zkCTX ctx, char \*\*fw\_ver\_str)  
*Get Zymkey firmware version.*
- int `zkGetSerialNumberString` (zkCTX ctx, char \*\*serial\_num\_str)  
*Get Zymkey serial number.*
- int `zkGetCPUTemp` (zkCTX ctx, float \*cpu\_temp)  
*Get current HSM CPU temperature. (model >= HSM6)*
- int `zkGetRTCDrift` (zkCTX ctx, float \*rtc\_drift)  
*Get current RTC drift. (model >= HSM6)*
- int `zkGetBatteryVoltage` (zkCTX ctx, float \*batt\_voltage)  
*Get the battery voltage (model >= HSM6).*

## Battery Voltage Monitor

- int `zkSetBatteryVoltageAction` (zkCTX ctx, int action)  
*Set battery voltage threshold action. (model >= HSM6)*
- int `zkSetBatteryVoltageThreshold` (zkCTX ctx, float threshold)  
*Sets the battery voltage threshold. (model >= HSM6)*

## CPU Temperature Monitor

- int `zkSetCPUTempAction` (zkCTX ctx, int action)  
*Set HSM CPU temperature threshold action. (model >= HSM6)*
- int `zkSetCPULowTempThreshold` (zkCTX ctx, float threshold)  
*Sets the HSM CPU low temperature threshold. (model >= HSM6)*
- int `zkSetCPUHighTempThreshold` (zkCTX ctx, float threshold)  
*Sets the HSM CPU high temperature threshold. (model >= HSM6)*

### 5.1.1 Detailed Description

C interface to Zymkey Application Utilities Library.

#### Author

Scott Miller

#### Version

1.0

#### Date

November 17, 2016

#### Copyright

Zymbit, Inc.

This file contains the C API to the the Zymkey Application Utilities library. This API 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 functions for changing the i2c address (i2c units only), setting tap sensitivity and controlling the LED.

### 5.1.2 Function Documentation

#### 5.1.2.1 zkClearPerimeterDetectEvents()

```
int zkClearPerimeterDetectEvents (
    zkCTX ctx )
```

Clear perimeter detect events.

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

#### Returns

0 for success, less than 0 for failure.

#### 5.1.2.2 zkClose()

```
int zkClose (
    zkCTX ctx )
```

Close a Zymkey context.

**Parameters**

<i>ctx</i>	(input) The Zymkey context to close
------------	-------------------------------------

**Returns**

0 for success, less than 0 for failure.

**5.1.2.3 zkCreateRandDataFile()**

```
int zkCreateRandDataFile (
    zkCTX ctx,
    const char * dst_filename,
    int rdata_sz )
```

Fill a file with random numbers.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>dst_filename</i>	(input) Absolute path name for the destination file.
<i>rdata_sz</i>	(input) The number of random bytes to generate.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.4 zkDisablePubKeyExport()**

```
int zkDisablePubKeyExport (
    zkCTX ctx,
    int pubkey_slot,
    bool slot_is_foreign )
```

Prevent a public key from being exported to the host. (model >= HSM6)

This function prevents the public key at the specified slot from being exported to the host using the API zkExport←PubKey.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>pubkey_slot</i>	(input) The key slot to disable pubkey export on.
<i>slot_is_foreign</i>	(input) The slot parameter refers to a slot in the foreign keyring.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.5 zkDoECDHAndKDF()**

```
int zkDoECDHAndKDF (
    zkCTX ctx,
    ZK_ECDH_KDF_TYPE kdf_type,
    int slot,
    const uint8_t * peer_pubkey,
    int peer_pubkey_sz,
    const uint8_t * salt,
    int salt_sz,
    const uint8_t * info,
    int info_sz,
    int num_iterations,
    int derived_key_sz,
    uint8_t ** derived_key )
```

Perform an ECDH operation plus Key Derivation Function. (model >= HSM6)

Perform an ECDH operation with Key Derivation Function (KDF). The derived key is returned in the response. The peer public key is presented in the call.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>slot</i>	(input) The key slot to use for the local key. If this parameter is -1, the ephemeral key is used.
<i>peer_pubkey</i>	(input) The peer public key.
<i>peer_pubkey_sz</i>	(input) Size of the peer public key.
<i>salt</i>	(input) The salt to use for the selected KDF.
<i>salt_sz</i>	(input) The salt size. Must be less than or equal to 128 bytes.
<i>info</i>	(input) The info field to use for RFC 5869. Ignored for PBKDF2.
<i>info_sz</i>	(input) The size of the info parameter. Must be less than or equal to 128 bytes.
<i>num_iterations</i>	(input) Number of iterations to carry out (PBKDF only)
<i>derived_key_sz</i>	(input) The desired number of bytes to return for the KDF. For RFC 5869, this value must be less than 8160 bytes (SHA256) or 16320 (SHA512).
<i>derived_key</i>	(output) returned pointer to the derived key.

**Returns**

0 for success, less than 0 for general failure.

**5.1.2.6 zkDoECDHAndKDFWithIntPeerPubkey()**

```
int zkDoECDHAndKDFWithIntPeerPubkey (
    zkCTX ctx,
```

```
ZK_ECDH_KDF_TYPE kdf_type,
int slot,
int peer_pubkey_slot,
bool peer_pubkey_slot_is_foreign,
const uint8_t * salt,
int salt_sz,
const uint8_t * info,
int info_sz,
int num_iterations,
int derived_key_sz,
uint8_t ** derived_key )
```

Perform an ECDH operation plus Key Derivation Function. (model >= HSM6)

Perform an ECDH operation with Key Derivation Function (KDF). The derived key is returned in the response. The peer public key is referenced from the zymkey internal key store.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>slot</i>	(input) The key slot to use for the local key. If this parameter is -1, the ephemeral key is used.
<i>peer_pubkey_slot</i>	(input) The peer public key slot where the peer public key is to be found.
<i>peer_pubkey_slot_is_foreign</i>	(input) If true, the peer public key slot is found in the foreign public keyring.
<i>salt</i>	(input) The salt to use for the selected KDF.
<i>salt_sz</i>	(input) The salt size. Must be less than or equal to 128 bytes.
<i>info</i>	(input) The info field to use for RFC 5869. Ignored for PBKDF2.
<i>info_sz</i>	(input) The size of the info parameter. Must be less than or equal to 128 bytes.
<i>num_iterations</i>	(input) Number of iterations to carry out (PBKDF only)
<i>derived_key_sz</i>	(input) The desired number of bytes to return for the KDF. For RFC 5869, this value must be less than 8160 bytes (SHA256) or 16320 (SHA512).
<i>derived_key</i>	(output) returned pointer to the derived key.

#### Returns

0 for success, less than 0 for general failure.

#### 5.1.2.7 zkDoRawECDH()

```
int zkDoRawECDH (
    zkCTX ctx,
    int slot,
    const uint8_t * peer_pubkey,
    int peer_pubkey_sz,
    uint8_t ** pre_master_secret )
```

Perform a raw ECDH operation. (model >= HSM6)

Perform an ECDH operation with no Key Derivation Function (KDF). The raw pre-master secret is returned in the response. The peer public key is presented in the call.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>slot</i>	(input) The key slot to use for the local key. If this parameter is -1, the ephemeral key is used.
<i>peer_pubkey</i>	(input) The peer public key.
<i>peer_pubkey_sz</i>	(input) Size of the peer public key.
<i>pre_master_secret</i>	(output) returned pointer to the pre-master secret

**Returns**

0 for success, less than 0 for general failure.

**5.1.2.8 zkDoRawECDHWithIntPeerPubkey()**

```
int zkDoRawECDHWithIntPeerPubkey (
    zkCTX ctx,
    int slot,
    int peer_pubkey_slot,
    bool peer_pubkey_slot_is_foreign,
    uint8_t ** pre_master_secret )
```

Perform a raw ECDH operation. (model >= HSM6)

Perform an ECDH operation with no Key Derivation Function (KDF). The raw pre-master secret is returned in the response. The peer public key is referenced from the zymkey internal key store.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>slot</i>	(input) The key slot to use for the local key. If this parameter is -1, the ephemeral key is used.
<i>peer_pubkey_slot</i>	(input) The peer public key slot where the peer public key is to be found.
<i>peer_pubkey_slot_is_foreign</i>	(input) If true, the peer public key slot is found in the foreign public keyring.
<i>pre_master_secret</i>	(output) returned pointer to the pre-master secret

**Returns**

0 for success, less than 0 for general failure.

**5.1.2.9 zkExportPubKey()**

```
int zkExportPubKey (
    zkCTX ctx,
    uint8_t ** pk,
```

```
int * pk_sz,
int pubkey_slot,
bool slot_is_foreign )
```

Gets the public key and stores in a byte array created by this function.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>pk</i>	(output) Pointer to a pointer created by this function which contains the public key.
<i>pk_sz</i>	(output) Pointer to an integer which contains the size of the public key.
<i>pubkey_slot</i>	(input) The key slot to retrieve. Zymkey and HSM4 have slots 0, 1, and 2.
<i>slot_is_foreign</i>	(input) If true, designates the pubkey slot to come from the foreign keystore (model >= HSM6).

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.10 zkExportPubKey2File()

```
int zkExportPubKey2File (
    zkCTX ctx,
    const char * filename,
    int pubkey_slot,
    bool slot_is_foreign )
```

Store the public key to a host file in PEM format.

This function is useful for generating Certificate Signing Requests (CSR).

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>filename</i>	(input) Filename where PEM formatted public key is to be stored.
<i>pubkey_slot</i>	(input) The key slot to retrieve. Zymkey and HSM4 have slots 0, 1, and 2.
<i>slot_is_foreign</i>	(input) If true, designates the pubkey slot to come from the foreign keystore. (model >= HSM6)

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.11 zkGenECDSASigFromDigest()

```
int zkGenECDSASigFromDigest (
    zkCTX ctx,
```

```
const uint8_t * digest,
int slot,
uint8_t ** sig,
int * sig_sz )
```

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

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>digest</i>	(input) This parameter contains the digest of the data that will be used to generate the signature.
<i>slot</i>	(input) The key slot to generate a signature from. This parameter is only valid for Zymkey models 4i and beyond.
<i>sig</i>	(output) A pointer to a pointer to an array of unsigned bytes which contains the generated signature. This pointer is created by this function and must be freed by the application when no longer needed.
<i>sig_sz</i>	(output) A pointer to an integer which contains the size of the signature.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.12 zkGenEphemeralKeyPair()

```
int zkGenEphemeralKeyPair (
    zkCTX ctx,
    ZK_EC_KEY_TYPE type )
```

Generate an ephemeral key pair. (model >= HSM6)

This function generates an ephemeral key pair of the specified type. Ephemeral key pairs are useful when performing ECDH for time-of-flight encryption. Only one ephemeral key slot is available and is not persistent between reboots.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>type</i>	(input) The type of key to generate (ZK_EC_KEY_TYPE).

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.13 zkGenKeyPair()

```
int zkGenKeyPair (
    zkCTX ctx,
    ZK_EC_KEY_TYPE type )
```

Generate a new persistent key pair. (model >= HSM6)

This function generates a new key pair of the specified type and store it persistently. This key pair cannot be used as part of the zymkey's digital wallet operations.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>type</i>	(input) The type of key to generate (ZK_EC_KEY_TYPE).

#### Returns

allocated slot number if successful, less than 0 for failure.

#### 5.1.2.14 zkGenWalletChildKey()

```
int zkGenWalletChildKey (
    zkCTX ctx,
    int parent_key_slot,
    uint32_t index,
    bool is_hardened )
```

Generate child key from a parent key in a blockchain wallet . (model >= HSM6)

This function generates a new child key descendent from a specified parent key in a wallet.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>parent_key_slot</i>	(input) The parent key slot to base the child key derivation on.
<i>index</i>	(input) The index of the child seed. This determines the node address as well as the outcome of the key generation.
<i>is_hardened</i>	(input) If true, a hardened key is generated.

#### Returns

allocated slot number if successful, less than 0 for failure.

#### 5.1.2.15 zkGenWalletMasterSeed()

```
int zkGenWalletMasterSeed (
    zkCTX ctx,
    ZK_EC_KEY_TYPE type,
    const char * wallet_name,
    const uint8_t * master_generator_key,
```

```
int master_generator_key_size,
char ** bip39_mnemonic )
```

Generate master seed to start a new blockchain wallet. (model >= HSM6)

This function generates a new blockchain master seed for creating a new wallet.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>type</i>	(input) The type of key to generate (ZK_EC_KEY_TYPE).
<i>wallet_name</i>	(input) An ASCII string which contains the name of the wallet.
<i>master_generator_key</i>	(input) The master generator key used to help generate the master seed.
<i>master_generator_key_size</i>	(input) The size of the master generator key. If 0, no master generator key is used in the formulation of the master seed.
<i>bip39_mnemonic</i>	(output) A pointer to the bip39 mnemonic sentence. If NULL, the master seed is generated per BIP32. Otherwise, the master seed is generated per BIP39 and the mnemonic sentence is returned in this parameter. The string is null terminated and encoded in UTF-8 NFKD from the BIP39 English dictionary.

#### Returns

allocated slot number if successful, less than 0 for failure.

#### 5.1.2.16 zkGetAccelerometerData()

```
int zkGetAccelerometerData (
    zkCTX ctx,
    zkAccelAxisDataType * x,
    zkAccelAxisDataType * y,
    zkAccelAxisDataType * z )
```

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

<i>x</i>	(output) x axis accelerometer information y (output) y axis accelerometer information z (output) z axis accelerometer information
----------	---

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.17 zkGetAllocSlotsList()

```
int zkGetAllocSlotsList (
    zkCTX ctx,
    bool is_foreign,
    int * max_num_keys,
    int ** alloc_key_list,
    int * alloc_key_list_sz )
```

Get the list of allocated keys (model >= HSM6).

This function returns a list of all allocated key slots.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>is_foreign</i>	(input) if true, retrieve allocation list of the foreign keys
<i>max_num_keys</i>	(input) retrieves the key pool size
<i>alloc_key_list</i>	(output) a pointer to an array of integers provided by this function to the caller
<i>alloc_key_list_sz</i>	(output) a pointer to an integer which contains the size of the returned key list

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.18 zkGetBatteryVoltage()

```
int zkGetBatteryVoltage (
    zkCTX ctx,
    float * batt_voltage )
```

Get the battery voltage (model >= HSM6).

This function gets the current battery voltage

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>battV</i>	(output) The current battery voltage value

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.19 zkGetCPUtemp()

```
int zkGetCPUtemp (
    zkCTX ctx,
    float * cpu_temp )
```

Get current HSM CPU temperature. (model >= HSM6)

This function gets the current HSM CPU temp.

#### Parameters

<i>cpu_temp</i>	(output) The temperature in celsius of the CPU.
-----------------	---

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.20 zkGetCurrentBindingInfo()

```
int zkGetCurrentBindingInfo (
    zkCTX ctx,
    bool * binding_is_locked,
    bool * is_bound )
```

Get current binding info.

This function gets the current binding lock state as well as the current binding state. This API is only valid for devices in the HSM family.

#### Parameters

<i>binding_is_locked</i>	(output) Binary value which expresses the current binding lock state.
	is_bound (output) Binary value which expresses the current bind state.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.21 zkGetECDSAPubKey()

```
int zkGetECDSAPubKey (
    zkCTX ctx,
    uint8_t ** pk,
    int * pk_sz,
    int slot )
```

[DEPRECATED] Use zkExportPubKey. Gets the public key and stores in a byte array created by this function.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>pk</i>	(output) Pointer to a pointer created by this function which contains the public key.
<i>pk_sz</i>	(output) Pointer to an integer which contains the size of the public key.
<i>slot</i>	(input) The key slot to retrieve. Only valid for model 4i and above.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.22 zkGetFirmwareVersionString()**

```
int zkGetFirmwareVersionString (
    zkCTX ctx,
    char ** fw_ver_str )
```

Get Zymkey firmware version.

This function retrieves the firmware version number of the zymkey referred to in a specified context

**Parameters**

<i>ctx</i>	(input) Zymkey context which was created with zkOpen
<i>version_str</i>	(output) A double pointer to the firmware version string. This function allocates this string. It is the caller's responsibility to free it.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.23 zkGetModelNumberString()**

```
int zkGetModelNumberString (
    zkCTX ctx,
    char ** model_str )
```

Get Zymkey model number.

This function retrieves the model number of the zymkey referred to in a specified context

**Parameters**

<i>ctx</i>	(input) Zymkey context which was created with zkOpen
<i>model_str</i>	(output) A double pointer to the model string. This function allocates this string. It is the caller's responsibility to free it.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.24 zkGetPerimeterDetectInfo()**

```
int zkGetPerimeterDetectInfo (
    zkCTX ctx,
    uint32_t ** timestamps_sec,
    int * num_timestamps )
```

Get current perimeter detect info.

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

**Parameters**

<i>timestamps_sec</i>	(output) The timestamps for when any breach occurred. The index in this array corresponds to the channel number used by zkSetPerimeterEventAction. A 0 value means no breach has occurred on this channel. This array is allocated by this routine and so it must be freed by the caller.
<i>num_timestamps</i>	(output) The number of timestamps in the returned array

**Returns**

0 for success, less than 0 for failure.

**5.1.2.25 zkGetRandBytes()**

```
int zkGetRandBytes (
    zkCTX ctx,
    uint8_t ** rdata,
    int rdata_sz )
```

Get an array of random bytes.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>rdata</i>	(input) Pointer to a pointer of bytes.
<i>rdata_sz</i>	(input) The number of random bytes to generate.

**Returns**

0 for success, less than 0 for failure.

### 5.1.2.26 zkGetRTCDrift()

```
int zkGetRTCDrift (
    zkCTX ctx,
    float * rtc_drift )
```

Get current RTC drift. (model >= HSM6)

This function is called to get the current RTC drift.

#### Parameters

<i>rtc_drift</i>	(output) The RTC drift.
------------------	-------------------------

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.27 zkGetSerialNumberString()

```
int zkGetSerialNumberString (
    zkCTX ctx,
    char ** serial_num_str )
```

Get Zymkey serial number.

This function retrieves the serial number of the zymkey referred to in a specified context

#### Parameters

<i>ctx</i>	(input) Zymkey context which was created with zkOpen
<i>serial_num_str</i>	(output) A double pointer to the serial number string. This function allocates this string. It is the caller's responsibility to free it.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.28 zkGetTime()

```
int zkGetTime (
    zkCTX ctx,
    uint32_t * epoch_time_sec,
    bool precise_time )
```

Get current GMT time.

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

**Parameters**

<i>epoch_time_sec</i>	(output) The time in seconds from the epoch (Jan. 1, 1970).
<i>precise_time</i>	(input) 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**

0 for success, less than 0 for failure.

**5.1.2.29 zkGetWalletKeySlotFromNodeAddr()**

```
int zkGetWalletKeySlotFromNodeAddr (
    zkCTX ctx,
    const char * node_addr,
    const char * wallet_name,
    int master_seed_slot,
    int * slot )
```

Derive the slot address from a node address. (model >= HSM6)

This function returns the slot number associated with a given node address.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>node_addr</i>	(input) A pointer which contains the node address in ASCII.
<i>wallet_name</i>	(input) A pointer which contains the wallet name in ASCII, used to identify the wallet identity. If desired, this parameter can be NULL and the <i>master_seed_slot</i> parameter can be specified instead.
<i>master_seed_slot</i>	(input) The master seed slot. Can be used to specify the wallet identity instead of the wallet name.
<i>slot</i>	(output) A pointer to an integer which contains the associated key slot.

**Returns**

0 if successful, less than 0 for failure.

**5.1.2.30 zkGetWalletNodeAddrFromKeySlot()**

```
int zkGetWalletNodeAddrFromKeySlot (
    zkCTX ctx,
    int slot,
    char ** node_addr,
```

```
char ** wallet_name,
int * master_seed_slot )
```

Derive the node address from a key slot number. (model >= HSM6)

This function derives a node address from an input key slot number.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>slot</i>	(input) A key slot number that is part of a digital wallet.
<i>node_addr</i>	(output) A pointer to a pointer which will contain the node address in ASCII.
<i>wallet_name</i>	(output) A pointer to a pointer which will contain the wallet name in ASCII. If NULL, this parameter will not be retrieved.
<i>master_seed_slot</i>	(output) A pointer to an integer which will contain the master seed key slot. If NULL, this parameter will not be retrieved.

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.31 zkInvalidateEphemeralKey()

```
int zkInvalidateEphemeralKey (
    zkCTX ctx )
```

Invalidate the ephemeral key. (model >= HSM6)

This function invalidates the ephemeral key.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
------------	-------------------------

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.32 zkLEDFlash()

```
int zkLEDFlash (
    zkCTX ctx,
    uint32_t on_ms,
    uint32_t off_ms,
    uint32_t num_flashes )
```

Flashes the LED.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>on_ms</i>	(input) The amount of time, in milliseconds, that the LED will stay on during a flash cycle.
<i>off_ms</i>	(input) The amount of time, in milliseconds, that the LED will stay off during a flash cycle.
<i>num_flashes</i>	(input) The number of on/off flash cycles to complete. If this parameter is 0, then the LED will flash indefinitely.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.33 zkLEDOff()**

```
int zkLEDOff (
    zkCTX ctx )
```

Turns the LED off.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
------------	-------------------------

**Returns**

0 for success, less than 0 for failure.

**5.1.2.34 zkLEDOn()**

```
int zkLEDOn (
    zkCTX ctx )
```

Turns the LED on.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
------------	-------------------------

**Returns**

0 for success, less than 0 for failure.

### 5.1.2.35 zkLockBinding()

```
int zkLockBinding (
    zkCTX ctx )
```

Set soft binding lock.

This function locks the binding for a specific HSM. This API is only valid for HSM series products.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.36 zkLockDataB2B()

```
int zkLockDataB2B (
    zkCTX ctx,
    const uint8_t * src_pt,
    int src_pt_sz,
    uint8_t ** dst_ct,
    int * dst_ct_sz,
    bool use_shared_key )
```

Lock up source (plaintext) data from a byte array and store the results (ciphertext) in a destination byte array.

This function encrypts and signs a block of plaintext data and stores the result in a binary byte array.

#### Note

(See zkLockDataF2F for notes about keys)

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>src_pt</i>	(input) Binary plaintext source byte array.
<i>src_pt_sz</i>	(input) Size of plaintext source data.
<i>dst_ct</i>	(output) A pointer to a pointer to an array of unsigned bytes created by this function. This pointer must be freed by the application when no longer needed.
<i>dst_ct_sz</i>	(output) A pointer to an integer which contains the size of the destination array.
<i>use_shared_key</i>	(input) Specifies if shared key is to be used. See zkLockDataF2F.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.37 zkLockDataB2F()

```
int zkLockDataB2F (
    zkCTX ctx,
    const uint8_t * src_pt,
    int src_pt_sz,
    const char * dst_ct_filename,
    bool use_shared_key )
```

Lock up source (plaintext) data from a byte array and store the results (ciphertext) in a destination file.

This function encrypts and signs a block of binary plaintext data and stores the result in a destination file.

#### Note

(See zkLockDataF2F for notes about keys)

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>src_pt</i>	(input) Binary plaintext source byte array.
<i>src_pt_sz</i>	(input) Size of plaintext source data.
<i>dst_ct_filename</i>	(input) The absolute path to the file where the destination (ciphertext) data should be deposited.
<i>use_shared_key</i>	(input) Specifies if shared key is to be used. See zkLockDataF2F.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.38 zkLockDataF2B()

```
int zkLockDataF2B (
    zkCTX ctx,
    const char * src_pt_filename,
    uint8_t ** dst_ct,
    int * dst_ct_sz,
    bool use_shared_key )
```

Lock up source (plaintext) data from a file and store the results (ciphertext) in a destination byte array.

This function encrypts and signs a block of plaintext data from a file and stores the result in a binary byte array.

#### Note

(See zkLockDataF2F for notes about keys)

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>src_pt_filename</i>	(input) The absolute path to the file where the source (plaintext) data is located.
<i>dst_ct</i>	(output) A pointer to a pointer to an array of unsigned bytes created by this function. This pointer must be freed by the application when no longer needed.
<i>dst_ct_sz</i>	(output) A pointer to an integer which contains the size of the destination array.
<i>use_shared_key</i>	(input) Specifies if shared key is to be used. See zkLockDataF2F.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.39 zkLockDataF2F()**

```
int zkLockDataF2F (
    zkCTX ctx,
    const char * src_pt_filename,
    const char * dst_ct_filename,
    bool use_shared_key )
```

Lock up source (plaintext) data from a file and store the results (ciphertext) in a destination file.

This function encrypts and signs a block of plaintext data from a file and stores the result in a destination file.

**Note**

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**

<i>ctx</i>	(input) Zymkey context.
<i>src_pt_filename</i>	(input) The absolute path to the file where the source (plaintext) data is located.
<i>dst_ct_filename</i>	(input) The absolute path to the file where the destination (ciphertext) data should be deposited.
<i>use_shared_key</i>	(input) This parameter specifies which key will be used to used to lock the data up. A value of 'false' specifies that the Zymkey will use the one-way key whereas 'true' specifies that the shared key will be used. Specify 'true' for publishing data to another that has the shared key (e.g. Zymbit cloud) and 'False' when the data is meant to reside exclusively withing the host computer.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.40 zkOpen()**

```
int zkOpen (
    zkCTX * ctx )
```

Open a Zymkey context.

**Parameters**

<i>ctx</i>	(output) returns a pointer to a Zymkey context.
------------	---

**Returns**

0 for success, less than 0 for failure.

**5.1.2.41 zkRemoveKey()**

```
int zkRemoveKey (
    zkCTX ctx,
    int slot,
    bool slot_is_foreign )
```

Remove a key pair or a foreign public key. (model >= HSM6)

This function deletes a key pair or a foreign public key from persistent storage.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>slot</i>	(input) The slot
<i>slot_is_foreign</i>	(input) The slot parameter refers to a slot in the foreign keyring.

**Returns**

0 if successful, less than 0 for failure.

**5.1.2.42 zkRestoreWalletMasterSeedFromMnemonic()**

```
int zkRestoreWalletMasterSeedFromMnemonic (
    zkCTX ctx,
```

```
ZK_EC_KEY_TYPE type,
const char * wallet_name,
const uint8_t * master_generator_key,
int master_generator_key_size,
char * bip39_mnemonic )
```

Restore a master seed from a BIP39 mnemonic and a master generator key. (model >= HSM6)

This function restores a wallet master seed from a supplied BIP39 mnemonic string and a master generator key.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>type</i>	(input) The type of key to generate (ZK_KEY_TYPE).
<i>wallet_name</i>	(input) An ASCII string which contains the name of the wallet.
<i>master_generator_key</i>	(input) The master generator key used to help generate the master seed.
<i>master_generator_key_size</i>	(input) The size of the master generator key. If 0, no master generator key is used in the formulation of the master seed.
<i>bip39_mnemonic</i>	(input) The bip39_mnemonic string, null terminated and UTF-8 NFKD encoded from the BIP39 English dictionary.

#### Returns

allocated slot number if successful, less than 0 for failure.

#### 5.1.2.43 zkSaveECDSAPubKey2File()

```
int zkSaveECDSAPubKey2File (
    zkCTX ctx,
    const char * filename,
    int slot )
```

[DEPRECATED] Use zkExportPubKey2File. Store the public key to a host file in PEM format.

This function is useful for generating Certificate Signing Requests (CSR).

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>filename</i>	(input) Filename where PEM formatted public key is to be stored.
<i>slot</i>	(input) The key slot to retrieve. Only valid for model 4i and above.

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.44 zkSetBatteryVoltageAction()

```
int zkSetBatteryVoltageAction (
    zkCTX ctx,
    int action )
```

Set battery voltage threshold action. (model >= HSM6)

This function specifies the action to take when the battery voltage falls below the threshold set by zkSetBatteryVoltageThreshold. If this function is never called, do nothing is default. There are three actions:

- Do nothing
- Go to sleep until battery is replaced
- Self-destruct

#### Parameters

<i>action</i>	(input) The action to apply, specify one of the ZK_THRESHOLD_ACTION_TYPE: – Do nothing (ZK_ACTION_NONE) – Sleep (ZK_ACTION_SLEEP) – Self-destruct (ZK_ACTION_SELF_DESTRUCT)
---------------	--

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.45 zkSetBatteryVoltageThreshold()

```
int zkSetBatteryVoltageThreshold (
    zkCTX ctx,
    float threshold )
```

Sets the battery voltage threshold. (model >= HSM6)

This function sets the threshold at which if the battery voltage falls below, the action set by zkSetBatteryVoltageAction will be carried out. The recommended threshold is 2.3V. If this function isn't called 2.5V is assumed by default. 2.5V is also the maximum threshold you can set.

#### Parameters

<i>threshold</i>	(input) The threshold in Volts.
------------------	---------------------------------

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.46 zkSetCPUHighTempThreshold()

```
int zkSetCPUHighTempThreshold (
    zkCTX ctx,
    float threshold )
```

Sets the HSM CPU high temperature threshold. (model >= HSM6)

This function sets the threshold at which if the on-board HSM CPU's tempreature rises above, the action set by zkSetCPUTempAction will be carried out. WARNING: You can lock yourself out in dev mode if you set a threshold below the CPU's ambient temperature. The recommended setting is no less than 40C. If no threshold is set, 65 degrees celsius is set as default.

#### Parameters

<i>threshold</i>	(input) The threshold in celsius.
------------------	-----------------------------------

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.47 zkSetCPULowTempThreshold()

```
int zkSetCPULowTempThreshold (
    zkCTX ctx,
    float threshold )
```

Sets the HSM CPU low temperature threshold. (model >= HSM6)

This function sets the threshold at which if the on-board HSM CPU's tempreature falls below, the action set by zkSetCPUTempAction will be carried out. WARNING: You can lock yourself out in dev mode if you set a threshold above the CPU's ambient temperature. The recommended setting is no more than 20C. If no threshold is set, -10 degrees celsius is set as default.

#### Parameters

<i>threshold</i>	(input) The threshold in celsius.
------------------	-----------------------------------

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.48 zkSetCPUTempAction()

```
int zkSetCPUTempAction (
    zkCTX ctx,
    int action )
```

Set HSM CPU temperature threshold action. (model >= HSM6)

This function specifies the action to take when the HSM CPU temperature falls below the threshold set by zkSetCPULowTempThreshold, or rises above the threshold set by zkSetCPUHighTempThreshold. There are two actions to apply:

- Do nothing
- Self-destruct

#### Parameters

<i>action</i>	(input) The action to apply, used it's named constant from ZK_THRESHOLD_ACTION_TYPE: – Do nothing (ZK_ACTION_NONE) – Self-destruct (ZK_ACTION_SELF_DESTRUCT)
---------------	--

#### Returns

0 for success, less than 0 for failure.

### 5.1.2.49 zkSetDigitalPerimeterDetectDelays()

```
int zkSetDigitalPerimeterDetectDelays (
    zkCTX ctx,
    int min_delay_ns,
    int max_delay_ns )
```

Set the delays (model >= HSM6).

This function sets delays on the digital perimeter detect

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>min_delay_ns</i>	(input) minimum delay in nanoseconds
<i>max_delay_ns</i>	(input) maximum delay in nanoseconds

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.50 zkSetDigitalPerimeterDetectLPMaxBits()

```
int zkSetDigitalPerimeterDetectLPMaxBits (
    zkCTX ctx,
    int max_num_bits )
```

Set the low power max number of bits (model >= HSM6).

This function sets low power max number of bits on the digital perimeter detect

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>max_num_bits</i>	(input) max number of bits

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.51 zkSetDigitalPerimeterDetectLPPPeriod()

```
int zkSetDigitalPerimeterDetectLPPPeriod (
    zkCTX ctx,
    int lp_period )
```

Set the low power period (model >= HSM6).

This function sets low power period on the digital perimeter detect

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>lp_period</i>	(input) low power period in microseconds

#### Returns

0 if successful, less than 0 for failure.

### 5.1.2.52 zkSetI2CAddr()

```
int zkSetI2CAddr (
    zkCTX ctx,
    int addr )
```

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**

<code>addr</code>	(input) The i2c address that the Zymkey will set itself to.
-------------------	---

**Returns**

0 for success, less than 0 for failure.

**5.1.2.53 zkSetPerimeterEventAction()**

```
int zkSetPerimeterEventAction (
    zkCTX ctx,
    int channel,
    uint32_t action_flags )
```

Set perimeter breach action.

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

- Notify host
- Zymkey self-destruct

**Parameters**

<code>channel</code>	(input) The channel (0 or 1) that the action flags will be applied to.
<code>action_flags</code>	(input) The actions to apply to the perimeter event channel: <ul style="list-style-type: none"> <li>– Notify (ZK_PERIMETER_EVENT_ACTION_NOTIFY)</li> <li>– Self-destruct (ZK_PERIMETER_EVENT_ACTION_SELF_DESTRUCT)</li> </ul>

**Returns**

0 for success, less than 0 for failure.

**5.1.2.54 zkSetTapSensitivity()**

```
int zkSetTapSensitivity (
    zkCTX ctx,
    int axis,
    float pct )
```

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**

<i>axis</i>	(input) The axis to configure. This parameter should contain one of the values in the enum typedef ACCEL_AXIS_TYPE.
<i>pct</i>	(input) The sensitivity expressed as percentage. <ul style="list-style-type: none"> <li>1. 0% = Shut down: Tap detection should not occur along the axis.</li> <li>2. 100% = Maximum sensitivity.</li> </ul>

**Returns**

0 for success, less than 0 for failure.

**5.1.2.55 zkStoreForeignPubKey()**

```
int zkStoreForeignPubKey (
    zkCTX ctx,
    ZK_EC_KEY_TYPE pk_type,
    uint8_t * pk,
    int pk_sz )
```

Store a new foreign public key in Zymkey. (model >= HSM6)

This function stores a new foreign public key in the Zymkey public key ring. This public key can be used for signature verification in use cases where it is desirable to hide the public key.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>pk_type</i>	(input) The type of the public key.
<i>pk</i>	(input) Pointer to the public key to store.
<i>pk_sz</i>	(input) The public key size.

**Returns**

allocated slot number in foreign key store, less than 0 for failure.

**5.1.2.56 zkUnlockDataB2B()**

```
int zkUnlockDataB2B (
    zkCTX ctx,
    const uint8_t * src_ct,
    int src_ct_sz,
    uint8_t ** dst_pt,
```

```
int * dst_pt_sz,
bool use_shared_key )
```

Unlock source (ciphertext) data from a byte array and store the results (plaintext) in a destination byte array.

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

#### Note

(See zkLockDataF2F for notes about keys)

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>src_ct</i>	(input) Binary ciphertext source byte array.
<i>src_ct_sz</i>	(input) Size of ciphertext source data.
<i>dst_pt</i>	(output) A pointer to a pointer to an array of unsigned bytes created by this function. This pointer must be freed by the application when no longer needed.
<i>dst_pt_sz</i>	(output) A pointer to an integer which contains the size of the destination array.
<i>use_shared_key</i>	(input) Specifies if shared key is to be used. See zkLockDataF2F.

#### Returns

0 for success, less than 0 for failure.

#### 5.1.2.57 zkUnlockDataB2F()

```
int zkUnlockDataB2F (
    zkCTX ctx,
    const uint8_t * src_ct,
    int src_ct_sz,
    const char * dst_pt_filename,
    bool use_shared_key )
```

Unlock source (ciphertext) data from a byte array and store the results (plaintext) in a destination file.

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

#### Note

(See zkLockDataF2F for notes about keys)

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>src_ct</i>	(input) Binary ciphertext source byte array.
<i>src_ct_sz</i>	(input) Size of ciphertext source data.
<i>dst_pt_filename</i>	(input) The absolute path to the file where the destination (plaintext) data should be deposited.
<i>use_shared_key</i>	(input) Specifies if shared key is to be used. See zkLockDataF2F.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.58 zkUnlockDataF2B()**

```
int zkUnlockDataF2B (
    zkCTX ctx,
    const char * src_ct_filename,
    uint8_t ** dst_pt,
    int * dst_pt_sz,
    bool use_shared_key )
```

Unlock source (ciphertext) data from a file and store the results (plaintext) in a destination byte array.

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

**Note**

(See zkLockDataF2F for notes about keys)

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>src_ct_filename</i>	(input) The absolute path to the file where the source (ciphertext) data is located.
<i>dst_pt</i>	(output) A pointer to a pointer to an array of unsigned bytes created by this function. This pointer must be freed by the application when no longer needed.
<i>dst_pt_sz</i>	(output) A pointer to an integer which contains the size of the destination array.
<i>use_shared_key</i>	(input) Specifies if shared key is to be used. See zkLockDataF2F.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.59 zkUnlockDataF2F()**

```
int zkUnlockDataF2F (
    zkCTX ctx,
    const char * src_ct_filename,
    const char * dst_pt_filename,
    bool use_shared_key )
```

Unlock source (ciphertext) data from a file and store the results (plaintext) in a destination file.

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

**Note**

(See zkLockDataF2F for notes about keys)

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>src_ct_filename</i>	(input) The absolute path to the file where the source (ciphertext) data is located.
<i>dst_pt_filename</i>	(input) The absolute path to the file where the destination (plaintext) data should be deposited.
<i>use_shared_key</i>	(input) This parameter specifies which key will be used to lock the data up. A value of 'false' specifies that the Zymkey will use the one-way key whereas 'true' specifies that the shared key will be used. Specify 'true' for publishing data to another that has the shared key (e.g. Zymbit cloud) and 'False' when the data is meant to reside exclusively within the host computer.

**Returns**

0 for success, less than 0 for failure.

**5.1.2.60 zkVerifyECDSASigFromDigest()**

```
int zkVerifyECDSASigFromDigest (
    zkCTX ctx,
    const uint8_t * digest,
    int pubkey_slot,
    const uint8_t * sig,
    int sig_sz )
```

Verify a signature using one of the Zymkey's public keys.

Verify a signature using an internal public key from the Zymkey private/public key store.

**Parameters**

<i>ctx</i>	(input) Zymkey context.
<i>digest</i>	(input) This parameter contains the digest of the data that will be used to generate the signature.
<i>pubkey_slot</i>	(input) The key slot to generate a signature from. This parameter is only valid for Zymkey models 4i and beyond.
<i>sig</i>	(input) Array of bytes which contains the signature.
<i>sig_sz</i>	(input) Size of signature.

**Returns**

0 for signature verification failed, 1 for signature verification passed, less than 0 for general failure.

**5.1.2.61 zkVerifyECDSASigFromDigestWithForeignKeySlot()**

```
int zkVerifyECDSASigFromDigestWithForeignKeySlot (
    zkCTX ctx,
```

```
const uint8_t * digest,
int pubkey_slot,
const uint8_t * sig,
int sig_sz )
```

Verify a signature using one of the Zymkey's foreign public keys.

Verify a signature using a public key from the Zymkey foreign key store.

#### Parameters

<i>ctx</i>	(input) Zymkey context.
<i>digest</i>	(input) This parameter contains the digest of the data that will be used to generate the signature.
<i>pubkey_slot</i>	(input) The key slot to generate a signature from. This parameter is only valid for Zymkey models 4i and beyond.
<i>sig</i>	(input) Array of bytes which contains the signature.
<i>sig_sz</i>	(input) Size of signature.

#### Returns

0 for signature verification failed, 1 for signature verification passed, less than 0 for general failure.

### 5.1.2.62 zkWaitForPerimeterEvent()

```
int zkWaitForPerimeterEvent (
    zkCTX ctx,
    uint32_t timeout_ms )
```

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. Note that, in order to receive perimeter events, the zymkey must have been configured to notify the host on either or both of the perimeter detect channels via a call to "zkSetPerimeterEventAction".

#### Parameters

<i>timeout_ms</i>	(input) The maximum amount of time in milliseconds to wait for a perimeter event to arrive.
-------------------	---

#### Returns

0 for success, less than 0 for failure, -ETIMEDOUT when no perimeter events detected within the specified timeout

### 5.1.2.63 zkWaitForTap()

```
int zkWaitForTap (
    zkCTX ctx,
    uint32_t timeout_ms )
```

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

<i>timeout_ms</i>	(input) The maximum amount of time in milliseconds to wait for a tap event to arrive.
-------------------	---

#### Returns

0 for success, less than 0 for failure, -ETIMEDOUT when no tap events detected within the specified timeout

# Index

g  
  zkAccelAxisDataType, 9

tapDirection  
  zkAccelAxisDataType, 9

zk\_app\_utils.h, 11  
  zkClearPerimeterDetectEvents, 16  
  zkClose, 16  
  zkCreateRandDataFile, 17  
  zkDisablePubKeyExport, 17  
  zkDoECDHAndKDFWithIntPeerPubkey, 18  
  zkDoECDHAndKDF, 18  
  zkDoRawECDHWithIntPeerPubkey, 20  
  zkDoRawECDH, 19  
  zkExportPubKey, 20  
  zkExportPubKey2File, 21  
  zkGenECDSASigFromDigest, 21  
  zkGenEphemeralKeyPair, 22  
  zkGenKeyPair, 22  
  zkGenWalletChildKey, 23  
  zkGenWalletMasterSeed, 23  
  zkGetAccelerometerData, 24  
  zkGetAllocSlotsList, 24  
  zkGetBatteryVoltage, 25  
  zkGetCPUtemp, 25  
  zkGetCurrentBindingInfo, 26  
  zkGetECDSAPubKey, 26  
  zkGetFirmwareVersionString, 27  
  zkGetModelNumberString, 27  
  zkGetPerimeterDetectInfo, 28  
  zkGetRTCDrift, 28  
  zkGetRandBytes, 28  
  zkGetSerialNumberString, 29  
  zkGetTime, 29  
  zkGetWalletKeySlotFromNodeAddr, 30  
  zkGetWalletNodeAddrFromKeySlot, 30  
  zkInvalidateEphemeralKey, 31  
  zkLEDFlash, 31  
  zkLEDOFF, 32  
  zkLEDON, 32  
  zkLockBinding, 32  
  zkLockDataB2B, 33  
  zkLockDataB2F, 33  
  zkLockDataF2B, 34  
  zkLockDataF2F, 35  
  zkOpen, 36  
  zkRemoveKey, 36  
  zkRestoreWalletMasterSeedFromMnemonic, 36  
  zkSaveECDSAPubKey2File, 37

  zkSetBatteryVoltageAction, 37  
  zkSetBatteryVoltageThreshold, 38  
  zkSetCPUHighTempThreshold, 38  
  zkSetCPULowTempThreshold, 39  
  zkSetCPUTempAction, 39  
  zkSetDigitalPerimeterDetectDelays, 40  
  zkSetDigitalPerimeterDetectLPMaxBits, 40  
  zkSetDigitalPerimeterDetectLPPPeriod, 41  
  zkSetI2CAddr, 41  
  zkSetPerimeterEventAction, 42  
  zkSetTapSensitivity, 42  
  zkStoreForeignPubKey, 43  
  zkUnlockDataB2B, 43  
  zkUnlockDataB2F, 44  
  zkUnlockDataF2B, 45  
  zkUnlockDataF2F, 45  
  zkVerifyECDSASigFromDigest, 46  
  zkVerifyECDSASigFromDigestWithForeignKey←  
    Slot, 46  
  zkWaitForPerimeterEvent, 47  
  zkWaitForTap, 47

zkAccelAxisDataType, 9  
  g, 9  
    tapDirection, 9

zkClearPerimeterDetectEvents  
  zk\_app\_utils.h, 16

zkClose  
  zk\_app\_utils.h, 16

zkCreateRandDataFile  
  zk\_app\_utils.h, 17

zkDisablePubKeyExport  
  zk\_app\_utils.h, 17

zkDoECDHAndKDFWithIntPeerPubkey  
  zk\_app\_utils.h, 18

zkDoECDHAndKDF  
  zk\_app\_utils.h, 18

zkDoRawECDHWithIntPeerPubkey  
  zk\_app\_utils.h, 20

zkDoRawECDH  
  zk\_app\_utils.h, 19

zkExportPubKey  
  zk\_app\_utils.h, 20

zkExportPubKey2File  
  zk\_app\_utils.h, 21

zkGenECDSASigFromDigest  
  zk\_app\_utils.h, 21

zkGenEphemeralKeyPair  
  zk\_app\_utils.h, 22

zkGenKeyPair

zk\_app\_utils.h, 22  
zkGenWalletChildKey  
    zk\_app\_utils.h, 23  
zkGenWalletMasterSeed  
    zk\_app\_utils.h, 23  
zkGetAccelerometerData  
    zk\_app\_utils.h, 24  
zkGetAllocSlotsList  
    zk\_app\_utils.h, 24  
zkGetBatteryVoltage  
    zk\_app\_utils.h, 25  
zkGetCPUTemp  
    zk\_app\_utils.h, 25  
zkGetCurrentBindingInfo  
    zk\_app\_utils.h, 26  
zkGetECDSSAPubKey  
    zk\_app\_utils.h, 26  
zkGetFirmwareVersionString  
    zk\_app\_utils.h, 27  
zkGetModelNumberString  
    zk\_app\_utils.h, 27  
zkGetPerimeterDetectInfo  
    zk\_app\_utils.h, 28  
zkGetRTCDrift  
    zk\_app\_utils.h, 28  
zkGetRandBytes  
    zk\_app\_utils.h, 28  
zkGetSerialNumberString  
    zk\_app\_utils.h, 29  
zkGetTime  
    zk\_app\_utils.h, 29  
zkGetWalletKeySlotFromNodeAddr  
    zk\_app\_utils.h, 30  
zkGetWalletNodeAddrFromKeySlot  
    zk\_app\_utils.h, 30  
zkInvalidateEphemeralKey  
    zk\_app\_utils.h, 31  
zkLEDFlash  
    zk\_app\_utils.h, 31  
zkLEDOff  
    zk\_app\_utils.h, 32  
zkLEDOn  
    zk\_app\_utils.h, 32  
zkLockBinding  
    zk\_app\_utils.h, 32  
zkLockDataB2B  
    zk\_app\_utils.h, 33  
zkLockDataB2F  
    zk\_app\_utils.h, 33  
zkLockDataF2B  
    zk\_app\_utils.h, 34  
zkLockDataF2F  
    zk\_app\_utils.h, 35  
zkOpen  
    zk\_app\_utils.h, 36  
zkRemoveKey  
    zk\_app\_utils.h, 36  
zkRestoreWalletMasterSeedFromMnemonic  
    zk\_app\_utils.h, 36  
zkSaveECDSSAPubKey2File  
    zk\_app\_utils.h, 37  
zkSetBatteryVoltageAction  
    zk\_app\_utils.h, 37  
zkSetBatteryVoltageThreshold  
    zk\_app\_utils.h, 38  
zkSetCPUHighTempThreshold  
    zk\_app\_utils.h, 38  
zkSetCPULowTempThreshold  
    zk\_app\_utils.h, 39  
zkSetCPUTempAction  
    zk\_app\_utils.h, 39  
zkSetDigitalPerimeterDetectDelays  
    zk\_app\_utils.h, 40  
zkSetDigitalPerimeterDetectLPMaxBits  
    zk\_app\_utils.h, 40  
zkSetDigitalPerimeterDetectLPPPeriod  
    zk\_app\_utils.h, 41  
zkSetI2CAddr  
    zk\_app\_utils.h, 41  
zkSetPerimeterEventAction  
    zk\_app\_utils.h, 42  
zkSetTapSensitivity  
    zk\_app\_utils.h, 42  
zkStoreForeignPubKey  
    zk\_app\_utils.h, 43  
zkUnlockDataB2B  
    zk\_app\_utils.h, 43  
zkUnlockDataB2F  
    zk\_app\_utils.h, 44  
zkUnlockDataF2B  
    zk\_app\_utils.h, 45  
zkUnlockDataF2F  
    zk\_app\_utils.h, 45  
zkVerifyECDSASigFromDigest  
    zk\_app\_utils.h, 46  
zkVerifyECDSASigFromDigestWithForeignKeySlot  
    zk\_app\_utils.h, 46  
zkWaitForPerimeterEvent  
    zk\_app\_utils.h, 47  
zkWaitForTap  
    zk\_app\_utils.h, 47