iButton Pam Module

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 hmac_sha256_ctx_t Struct Reference

`#include <hmac-sha256.h>`

Collaboration diagram for hmac_sha256_ctx_t:

Public Attributes

- sha256_ctx_t a
- sha256_ctx_t b

3.1.1 Member Data Documentation

3.1.1.1 a

`sha256_ctx_t hmac_sha256_ctx_t::a`

3.1.1.2 b

`sha256_ctx_t hmac_sha256_ctx_t::b`

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

- src/headers/hmac-sha256.h

3.2 ibutton_keys_ Struct Reference

database,which contains encrypted keys

`#include <_serial.h>`

Public Attributes

- const char * ibutton_crc_id
- char hash_str [65]

3.2.1 Detailed Description

database,which contains encrypted keys

3.2.2 Member Data Documentation

3.2.2.1 hash_str

```
char ibutton_keys_::hash_str[65]
```

3.2.2.2 ibutton_crc_id

```
const char* ibutton_keys_::ibutton_crc_id
```

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

- src/headers/_serial.h

3.3 sha256_ctx_t Struct Reference

`#include <sha256.h>`

Public Attributes

- uint32_t h [8]
- uint64_t length

3.3.1 Member Data Documentation

3.3.1.1 h

```
uint32_t sha256_ctx_t::h[8]
```

3.3.1.2 length

```
uint64_t sha256_ctx_t::length
```

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

- src/headers/sha256.h

Chapter 4

File Documentation

4.1 src/headers/_serial.h File Reference

```
#include <libserialport.h>
#include <sysexits.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

Include dependency graph for _serial.h:

4.2 src/headers/hmac-sha256.h File Reference

```
#include "sha256.h"
```

Include dependency graph for hmac-sha256.h: This graph shows which files directly or indirectly include this file:

Classes

- struct hmac_sha256_ctx_t

Macros

- #define HMAC_SHA256_BITS SHA256_HASH_BITS
- #define HMAC_SHA256_BYTES SHA256_HASH_BYTES
- #define HMAC_SHA256_BLOCK_BITS SHA256_BLOCK_BITS
- #define HMAC_SHA256_BLOCK_BYTES SHA256_BLOCK_BYTES

Functions

- void hmac_sha256_init (hmac_sha256_ctx_t *s, const void *key, uint16_t keylength_b)
- void hmac_sha256_nextBlock (hmac_sha256_ctx_t *s, const void *block)
- void hmac_sha256_lastBlock (hmac_sha256_ctx_t *s, const void *block, uint16_t length_b)
- void hmac_sha256_final (void *dest, hmac_sha256_ctx_t *s)
- void hmac_sha256 (void *dest, const void *key, uint16_t keylength_b, const void *msg, uint32_t msglength↩ _b)

4.2.1 Macro Definition Documentation

4.2.1.1 HMAC_SHA256_BITS

```
#define HMAC_SHA256_BITS SHA256_HASH_BITS
```

4.2.1.2 HMAC_SHA256_BLOCK_BITS

```
#define HMAC_SHA256_BLOCK_BITS SHA256_BLOCK_BITS
```

4.2.1.3 HMAC_SHA256_BLOCK_BYTES

```
#define HMAC_SHA256_BLOCK_BYTES SHA256_BLOCK_BYTES
```

4.2.1.4 HMAC_SHA256_BYTES

```
#define HMAC_SHA256_BYTES SHA256_HASH_BYTES
```

4.2.2 Function Documentation

4.2.2.1 hmac_sha256()

```
void hmac_sha256 (
            void * dest,
            const void * key,
            uint16_t keylength_b,
            const void * msg,
            uint32_t msglength_b )
```

4.2.2.2 hmac_sha256_final()

```
void hmac_sha256_final (
            void * dest,
            hmac_sha256_ctx_t * s )
```

4.2.2.3 hmac_sha256_init()

```
void hmac_sha256_init (
            hmac_sha256_ctx_t * s,
            const void * key,
            uint16_t keylength_b )
```

4.2.2.4 hmac_sha256_lastBlock()

```
void hmac_sha256_lastBlock (
            hmac_sha256_ctx_t * s,
            const void * block,
            uint16_t length_b )
```

4.2.2.5 hmac_sha256_nextBlock()

```
void hmac_sha256_nextBlock (
            hmac_sha256_ctx_t * s,
            const void * block )
```

4.3 src/headers/notify.h File Reference

```
#include <sys/inotify.h>
#include <unistd.h>
#include <stdbool.h>
```

Include dependency graph for notify.h: This graph shows which files directly or indirectly include this file:

Functions

- static bool search_com_device (const char ∗desired_port, int ttl)

  This is a part of project witch is resposible for detecting a new serial devices via inotify dir detection.
- static uint8_t hex (char ch)

  The functions to_byte_array & to_byte_array were taken from: https://stackoverflow.com/a/50085715.
- static int to_byte_array (const char ∗in, size_t in_size, uint8_t ∗out)

  to_byte_array - prepare hexadecimal string to byte array for hash calculating [in] in - input hex string [in] in_size - count of symbols [in] out - output hex 8 byte array

4.3.1 Function Documentation

4.3.1.1 hex()

```
static uint8_t hex (
            char ch )  [inline], [static]
```

The functions to_byte_array & to_byte_array were taken from:
https://stackoverflow.com/a/50085715.

4.3.1.2 search_com_device()

```
static inline bool search_com_device (
            const char * desired_port,
            int ttl )  [inline], [static]
```

This is a part of project witch is resposible for detecting a new serial devices via inotify dir detection.

The inotify helps us to check state of any path of the system

Parameters

desired_port	the path to the tty device
ttl	(time to live) the time for authorization by ibutton

Returns

true - Device has been found, false - Device hasn`t been found or ttl overflow

4.3.1.3 to_byte_array()

```
static int to_byte_array (
            const char * in,
            size_t in_size,
            uint8_t * out )  [inline], [static]
```

to_byte_array - prepare hexadecimal string to byte array for hash calculating [in] in - input hex string
[in] in_size - count of symbols [in] out - output hex 8 byte array

Returns

count - number of bytes

4.4 src/headers/sha256.h File Reference

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

Include dependency graph for sha256.h: This graph shows which files directly or indirectly include this file:

Classes

- struct sha256_ctx_t

Macros

- #define __LITTLE_ENDIAN__
- #define SHA256_HASH_BITS 256
- #define SHA256_HASH_BYTES (SHA256_HASH_BITS/8)
- #define SHA256_BLOCK_BITS 512
- #define SHA256_BLOCK_BYTES (SHA256_BLOCK_BITS/8)

Typedefs

- typedef uint8_t sha256_hash_t[SHA256_HASH_BYTES]

  SHA-256 hash value type.

Functions

- void sha256_init (sha256_ctx_t *state)

  initialise a SHA-256 context
- void sha256_nextBlock (sha256_ctx_t *state, const void *block)

  update the context with a given block
- void sha256_lastBlock (sha256_ctx_t *state, const void *block, uint16_t length_b)

  finalize the context with the given block
- void sha256_ctx2hash (sha256_hash_t *dest, const sha256_ctx_t *state)

  convert the hash state into the hash value This function reads the context and writes the hash value to the destination
- void sha256 (sha256_hash_t *dest, const void *msg, uint32_t length_b)

  simple SHA-256 hashing function for direct hashing

4.4.1 Detailed Description

Author

Daniel Otte

Date

2006-05-16 GPLv3 or later

4.4.2 Macro Definition Documentation

4.4.2.1 __LITTLE_ENDIAN__

```
#define __LITTLE_ENDIAN__
```

4.4.2.2 SHA256_BLOCK_BITS

```
#define SHA256_BLOCK_BITS 512
```

defines the size of a SHA-256 input block in bits

4.4.2.3 SHA256_BLOCK_BYTES

```
#define SHA256_BLOCK_BYTES (SHA256_BLOCK_BITS/8)
```

defines the size of a SHA-256 input block in bytes

4.4.2.4 SHA256_HASH_BITS

```
#define SHA256_HASH_BITS 256
```

defines the size of a SHA-256 hash value in bits

4.4.2.5 SHA256_HASH_BYTES

```
#define SHA256_HASH_BYTES (SHA256_HASH_BITS/8)
```

defines the size of a SHA-256 hash value in bytes

4.4.3 Typedef Documentation

4.4.3.1 sha256_hash_t

```
sha256_hash_t
```

SHA-256 hash value type.

A variable of this type may hold the hash value produced by the sha256_ctx2hash(sha256_hash_t ∗dest, const sha256_ctx_t ∗state) function.

4.4.4 Function Documentation

4.4.4.1 sha256()

```
void sha256 (
            sha256_hash_t * dest,
            const void * msg,
            uint32_t length_b )
```

simple SHA-256 hashing function for direct hashing

This function automaticaly hashes a given message of arbitary length with the SHA-256 hashing algorithm.

Parameters

dest	pointer to the location where the hash value is going to be written to
msg	pointer to the message thats going to be hashed
length_b	length of the message in bits

4.4.4.2 sha256_ctx2hash()

```
void sha256_ctx2hash (
            sha256_hash_t * dest,
            const sha256_ctx_t * state )
```

convert the hash state into the hash value This function reads the context and writes the hash value to the destination

Parameters

dest	pointer to the location where the hash value should be written
state	pointer to the SHA-256 hash context

4.4.4.3 sha256_init()

```
void sha256_init (
            sha256_ctx_t * state )
```

initialise a SHA-256 context

This function sets a sha256_ctx_t to the initial values for hashing.

Parameters

state	pointer to the SHA-256 hashing context

initialise a SHA-256 context

Parameters

state	pointer to a sha256 context

Returns

none

4.4.4.4 sha256_lastBlock()

```
void sha256_lastBlock (
            sha256_ctx_t * state,
            const void * block,
            uint16_t length )
```

finalize the context with the given block

This function finalizes the SHA-256 hash context by processing the given block of variable length.

Parameters

state	pointer to the SHA-256 hash context
block	pointer to the block of fixed length (512 bit = 64 byte)
length_b	the length of the block in bits

finalize the context with the given block

Parameters

state	Pointer to the context in which this block should be processed.
block	Pointer to the message wich should be hashed.
length	is the length of only THIS block in BITS not in bytes! bits are big endian, meaning high bits come first. if you have a message with bits at the end, the byte must be padded with zeros

4.4.4.5 sha256_nextBlock()

```
void sha256_nextBlock (
            sha256_ctx_t * state,
            const void * block )
```

update the context with a given block

This function updates the SHA-256 hash context by processing the given block of fixed length.

Parameters

state	pointer to the SHA-256 hash context
block	pointer to the block of fixed length (512 bit = 64 byte)

block must be, 512, Bit = 64, Byte, long !!!

4.5 src/hmac-sha256.c File Reference

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

```
#include "headers/sha256.h"
#include "headers/hmac-sha256.h"
```
Include dependency graph for hmac-sha256.c:

Macros

- #define IPAD 0x36
- #define OPAD 0x5C

Functions

- void hmac_sha256_init (hmac_sha256_ctx_t *s, const void *key, uint16_t keylength_b)
- void hmac_sha256_nextBlock (hmac_sha256_ctx_t *s, const void *block)
- void hmac_sha256_lastBlock (hmac_sha256_ctx_t *s, const void *block, uint16_t length_b)
- void hmac_sha256_final (void *dest, hmac_sha256_ctx_t *s)
- void hmac_sha256 (void *dest, const void *key, uint16_t keylength_b, const void *msg, uint32_t msglength_b)

4.5.1 Macro Definition Documentation

4.5.1.1 IPAD

```
#define IPAD 0x36
```

implementation of HMAC as described in RFC2104 Author: Daniel Otte email: daniel.otte@rub.de License: GPLv3 or later

4.5.1.2 OPAD

```
#define OPAD 0x5C
```

4.5.2 Function Documentation

4.5.2.1 hmac_sha256()

```
void hmac_sha256 (
            void * dest,
            const void * key,
            uint16_t keylength_b,
            const void * msg,
            uint32_t msglength_b )
```

4.5.2.2 hmac_sha256_final()

```
void hmac_sha256_final (
             void * dest,
             hmac_sha256_ctx_t * s )
```

4.5.2.3 hmac_sha256_init()

```
void hmac_sha256_init (
             hmac_sha256_ctx_t * s,
             const void * key,
             uint16_t keylength_b )
```

4.5.2.4 hmac_sha256_lastBlock()

```
void hmac_sha256_lastBlock (
             hmac_sha256_ctx_t * s,
             const void * block,
             uint16_t length_b )
```

4.5.2.5 hmac_sha256_nextBlock()

```
void hmac_sha256_nextBlock (
             hmac_sha256_ctx_t * s,
             const void * block )
```

4.6 src/pam_module.c File Reference

```
#include <libserialport.h>
#include <libconfig.h>
#include <security/pam_appl.h>
#include <security/pam_modules.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include "headers/_serial.h"
#include "headers/notify.h"
#include "headers/sha256.h"
#include "headers/hmac-sha256.h"
```

Include dependency graph for pam_module.c:

Macros

- #define PAM_CONFIG "/etc/ibutton_pam.config"

Functions

- PAM_EXTERN int pam_sm_setcred (pam_handle_t *pamh, int flags, int argc, const char **argv)
- PAM_EXTERN int pam_sm_acct_mgmt (pam_handle_t *pamh, int flags, int argc, const char **argv)
- PAM_EXTERN int pam_sm_authenticate (pam_handle_t *pamh, int flags, int argc, const char **argv)

  This is the base function for authorization.

4.6.1 Macro Definition Documentation

4.6.1.1 PAM_CONFIG

```
#define PAM_CONFIG "/etc/ibutton_pam.config"
```

4.6.2 Function Documentation

4.6.2.1 pam_sm_acct_mgmt()

```
PAM_EXTERN int pam_sm_acct_mgmt (
             pam_handle_t * pamh,
             int flags,
             int argc,
             const char ** argv )
```

4.6.2.2 pam_sm_authenticate()

```
PAM_EXTERN int pam_sm_authenticate (
             pam_handle_t * pamh,
             int flags,
             int argc,
             const char ** argv )
```

This is the base function for authorization.

4.6.2.3 pam_sm_setcred()

```
PAM_EXTERN int pam_sm_setcred (
             pam_handle_t * pamh,
             int flags,
             int argc,
             const char ** argv )
```

4.7 src/sha256.c File Reference

SHA-256 implementation.

```
#include <stdint.h>
#include <string.h>
#include "headers/sha256.h"
```

Include dependency graph for sha256.c:

Macros

- #define LITTLE_ENDIAN
- #define CH(x, y, z) (((x)&(y)) ^ ((∼(x))&(z)))
- #define MAJ(x, y, z) (((x)&(y)) ^ ((x)&(z)) ^ ((y)&(z)))
- #define SIGMA0(x) (rotr32((x),2) ^ rotr32((x),13) ^ rotr32((x),22))
- #define SIGMA1(x) (rotr32((x),6) ^ rotr32((x),11) ^ rotr32((x),25))
- #define SIGMA_a(x) (rotr32((x),7) ^ rotr32((x),18) ^ ((x)>>3))
- #define SIGMA_b(x) (rotr32((x),17) ^ rotr32((x),19) ^ ((x)>>10))

Functions

- void sha256_init (sha256_ctx_t ∗state)

  `sh256_init` initialises a sha256 context for hashing. `sh256_init` c initialises the given sha256 context for hashing
- uint32_t rotr32 (uint32_t x, uint8_t n)
- uint32_t change_endian32 (uint32_t x)
- void sha256_nextBlock (sha256_ctx_t ∗state, const void ∗block)

  update the context with a given block
- void sha256_lastBlock (sha256_ctx_t ∗state, const void ∗block, uint16_t length)

  function to process the last block being hashed
- void sha256 (sha256_hash_t ∗dest, const void ∗msg, uint32_t length)

  simple SHA-256 hashing function for direct hashing
- void sha256_ctx2hash (sha256_hash_t ∗dest, const sha256_ctx_t ∗state)

  convert the hash state into the hash value This function reads the context and writes the hash value to the destination

Variables

- uint32_t sha256_init_vector [ ]
- uint32_t k [ ]

4.7.1 Detailed Description

SHA-256 implementation.

Author

Daniel Otte

Date

16.05.2006

License:

GPL

4.7.2 Macro Definition Documentation

4.7.2.1 CH

```
#define CH(
              x,
              y,
              z ) (((x)&(y)) ^ ((~(x))&(z)))
```

4.7.2.2 LITTLE_ENDIAN

```
#define LITTLE_ENDIAN
```

4.7.2.3 MAJ

```
#define MAJ(
              x,
              y,
              z ) (((x)&(y)) ^ ((x)&(z)) ^ ((y)&(z)))
```

4.7.2.4 SIGMA0

```
#define SIGMA0(
              x ) (rotr32((x),2) ^ rotr32((x),13) ^ rotr32((x),22))
```

4.7.2.5 SIGMA1

```
#define SIGMA1(
             x ) (rotr32((x),6) ^ rotr32((x),11) ^ rotr32((x),25))
```

4.7.2.6 SIGMA_a

```
#define SIGMA_a(
             x ) (rotr32((x),7) ^ rotr32((x),18) ^ ((x)>>3))
```

4.7.2.7 SIGMA_b

```
#define SIGMA_b(
             x ) (rotr32((x),17) ^ rotr32((x),19) ^ ((x)>>10))
```

4.7.3 Function Documentation

4.7.3.1 change_endian32()

```
uint32_t change_endian32 (
             uint32_t x )
```

4.7.3.2 rotr32()

```
uint32_t rotr32 (
             uint32_t x,
             uint8_t n )
```

rotate x right by n positions

4.7.3.3 sha256()

```
void sha256 (
             sha256_hash_t * dest,
             const void * msg,
             uint32_t length )
```

simple SHA-256 hashing function for direct hashing

This function automaticaly hashes a given message of arbitary length with the SHA-256 hashing algorithm.

Parameters

dest	pointer to the location where the hash value is going to be written to
msg	pointer to the message thats going to be hashed
length_b	length of the message in bits

4.7.3.4 sha256_ctx2hash()

```
void sha256_ctx2hash (
            sha256_hash_t * dest,
            const sha256_ctx_t * state )
```

convert the hash state into the hash value This function reads the context and writes the hash value to the destination

Parameters

dest	pointer to the location where the hash value should be written
state	pointer to the SHA-256 hash context

4.7.3.5 sha256_init()

```
void sha256_init (
            sha256_ctx_t * state )
```

`sh256_init` initialises a sha256 context for hashing. `sh256_init` c initialises the given sha256 context for hashing

initialise a SHA-256 context

Parameters

state	pointer to a sha256 context

Returns

none

4.7.3.6 sha256_lastBlock()

```
void sha256_lastBlock (
            sha256_ctx_t * state,
```

```
            const void * block,
            uint16_t length )
```

function to process the last block being hashed

finalize the context with the given block

Parameters

state	Pointer to the context in which this block should be processed.
block	Pointer to the message wich should be hashed.
length	is the length of only THIS block in BITS not in bytes! bits are big endian, meaning high bits come first. if you have a message with bits at the end, the byte must be padded with zeros

4.7.3.7 sha256_nextBlock()

```
void sha256_nextBlock (
            sha256_ctx_t * state,
            const void * block )
```

update the context with a given block

block must be, 512, Bit = 64, Byte, long !!!

4.7.4 Variable Documentation

4.7.4.1 k

```
uint32_t k[]
```

Initial value:

```
={
    0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,
    0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,
    0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,
    0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,
    0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,
    0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,
    0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,
    0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2
}
```

4.7.4.2 sha256_init_vector

```
uint32_t sha256_init_vector[]
```

Initial value:

```
={
    0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A,
    0x510E527F, 0x9B05688C, 0x1F83D9AB, 0x5BE0CD19 }
```

Index