Nano cryptocurrency C library with P2PoW/DPoW support for Embedded

1.0.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

Overview

myNanoEmbedded is a lightweight C library of source files that integrates `Nano Cryptocurrency` to low complexity computational devices to send/receive digital money to anywhere in the world with fast trasnsaction and with a small fee by delegating a Proof of Work with your choice:

- DPoW (Distributed Proof of Work)
- P2PoW (a Descentralized P2P Proof of Work)

API features

- Attaches a random function to TRNG hardware (if available)
- Self entropy verifier to ensure excelent TRNG or PRNG entropy
- Creates a encrypted by password your stream or file to store your Nano SEED
- Bip39 and Brainwallet support
- Convert raw data to Base32
- Parse SEED and Bip39 to JSON
- Sign a block using Blake2b hash with Ed25519 algorithm
- ARM-A, ARM-M, Thumb, Xtensa-LX6 and IA64 compatible
- Linux desktop, Raspberry PI, ESP32 and Olimex A20 tested platforms
- Communication over `Fenix protocol` bridge over TLS
- Libsodium and mbedTLS libraries with smaller resources and best performance
- Optmized for size and speed
- Non static functions (all data is cleared before processed for security)

To add this API in your project you must first:

1. Download the latest version.

   ```
   git clone https://github.com/devfabiosilva/myNanoEmbedded.git --recurse-submodules
   ```

2. Include the main library files in the client application.

   ```
   #include "f_nano_crypto_util.h"
   ```

Initialize API

Function	Description
f_random_attach() (p. **??**)	Initializes the PRNG or TRNG to be used in this API

Transmit/Receive transactions

To transmit/receive your transaction you must use `Fenix` protocol to stabilish a DPoW/P2PoW support

Examples using platforms

The repository has some examples with most common embedded and Linux systems

- `Native Linux`
- `Raspberry Pi`
- `ESP32`
- `Olimex A20`
- `STM`

Credits

Author

Fábio Pereira da Silva

Date

Feb 2020

Version

1.0

Copyright

License MIT `see here`

References:

1- Editing

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 Files

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 f_block_transfer_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **preamble** [32]
- uint8_t **account** [32]
- uint8_t **previous** [32]
- uint8_t **representative** [32]
- **f_uint128_t balance**
- uint8_t **link** [32]
- uint8_t **signature** [64]
- uint8_t **prefixes**
- uint64_t **work**

4.1.1 Detailed Description

Nano signed block raw data defined in this `reference`

Definition at line **240** of file **f_nano_crypto_util.h**.

4.1.2 Field Documentation

4.1.2.1 account

```
uint8_t account[32]
```

Account in raw binary data.

Definition at line **244** of file **f_nano_crypto_util.h**.

4.1.2.2 balance

```
f_uint128_t balance
```

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **252** of file **f_nano_crypto_util.h**.

4.1.2.3 link

```
uint8_t link[32]
```

link or destination account

Definition at line **254** of file **f_nano_crypto_util.h**.

4.1.2.4 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **242** of file **f_nano_crypto_util.h**.

4.1.2.5 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **258** of file **f_nano_crypto_util.h**.

4.1.2.6 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **246** of file **f_nano_crypto_util.h**.

4.1.2.7 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **248** of file **f_nano_crypto_util.h**.

4.1.2.8 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **256** of file **f_nano_crypto_util.h**.

4.1.2.9 work

```
uint64_t work
```

Internal use for this API.

Definition at line **260** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.2 f_file_info_err_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

4.2.1 Detailed Description

Error enumerator for info file functions.

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

- **f_nano_crypto_util.h**

4.3 f_nano_crypto_wallet_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **nano_hdr** [sizeof(NANO_WALLET_MAGIC)]
- uint32_t **ver**
- uint8_t **description** [F_DESC_SZ]
- uint8_t **salt** [32]
- uint8_t **iv** [16]
- F_ENCRYPTED_BLOCK **seed_block**

4.3.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **371** of file **f_nano_crypto_util.h**.

4.3.2 Field Documentation

4.3.2.1 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **377** of file **f_nano_crypto_util.h**.

4.3.2.2 iv

```
uint8_t iv[16]
```

Initial vector of first encryption layer.

Definition at line **381** of file **f_nano_crypto_util.h**.

4.3.2.3 nano_hdr

```
uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)]
```

Header of the file.

Definition at line **373** of file **f_nano_crypto_util.h**.

4.3.2.4 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **379** of file **f_nano_crypto_util.h**.

4.3.2.5 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **383** of file **f_nano_crypto_util.h**.

4.3.2.6 ver

```
uint32_t ver
```

Version of the file.

Definition at line **375** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.4 f_nano_encrypted_wallet_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **sub_salt** [32]
- uint8_t **iv** [16]
- uint8_t **reserved** [16]
- uint8_t **hash_sk_unencrypted** [32]
- uint8_t **sk_encrypted** [32]

4.4.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **343** of file **f_nano_crypto_util.h**.

4.4.2 Field Documentation

4.4.2.1 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **351** of file **f_nano_crypto_util.h**.

4.4.2.2 iv

```
uint8_t iv[16]
```

Initial sub vector.

Definition at line **347** of file **f_nano_crypto_util.h**.

4.4.2.3 reserved

```
uint8_t reserved[16]
```

Reserved (not used)

Definition at line **349** of file **f_nano_crypto_util.h**.

4.4.2.4 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **353** of file **f_nano_crypto_util.h**.

4.4.2.5 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

Definition at line **345** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.5 f_nano_wallet_info_bdy_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **wallet_prefix**
- uint32_t **last_used_wallet_number**
- char **wallet_representative** [**MAX_STR_NANO_CHAR**]
- char **max_fee** [F_RAW_STR_MAX_SZ]
- uint8_t **reserved** [44]

4.5.1 Detailed Description

struct of the body block of the info file

Definition at line **455** of file **f_nano_crypto_util.h**.

4.5.2 Field Documentation

4.5.2.1 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **459** of file **f_nano_crypto_util.h**.

4.5.2.2 max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

Definition at line **463** of file **f_nano_crypto_util.h**.

4.5.2.3 reserved

```
uint8_t reserved[44]
```

Reserved.

Definition at line **465** of file **f_nano_crypto_util.h**.

4.5.2.4 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **457** of file **f_nano_crypto_util.h**.

4.5.2.5 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **461** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.6 f_nano_wallet_info_t Struct Reference

```
#include <f_nano_crypto_util.h>
```

Data Fields

- uint8_t **header** [sizeof(F_NANO_WALLET_INFO_MAGIC)]
- uint16_t **version**
- char **desc** [F_NANO_DESC_SZ]
- uint8_t **nanoseed_hash** [32]
- uint8_t **file_info_integrity** [32]
- F_NANO_WALLET_INFO_BODY **body**

4.6.1 Detailed Description

struct of the body block of the info file

Definition at line **487** of file **f_nano_crypto_util.h**.

4.6.2 Field Documentation

4.6.2.1 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **499** of file **f_nano_crypto_util.h**.

4.6.2.2 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **493** of file **f_nano_crypto_util.h**.

4.6.2.3 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **497** of file **f_nano_crypto_util.h**.

4.6.2.4 header

```
uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)]
```

Header magic.

Definition at line **489** of file **f_nano_crypto_util.h**.

4.6.2.5 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **495** of file **f_nano_crypto_util.h**.

4.6.2.6 version

```
uint16_t version
```

Version.

Definition at line **491** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

Chapter 5

File Documentation

5.1 f_add_bn_288_le.h File Reference

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

Typedefs

- typedef uint8_t **F_ADD_288**[36]

5.1.1 Detailed Description

Low level implementation of Nano Cryptocurrency C library.

Definition in file **f_add_bn_288_le.h**.

5.1.2 Typedef Documentation

5.1.2.1 F_ADD_288

```
F_ADD_288
```

288 bit big number

Definition at line **19** of file **f_add_bn_288_le.h**.

5.2 f_add_bn_288_le.h

```
/*
    AUTHOR: Fábio Pereira da Silva
    YEAR: 2019-20
    LICENSE: MIT
    EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
*/

#include <stdint.h>

typedef uint8_t F_ADD_288[36];


#ifndef F_DOC_SKIP

 void f_add_bn_288_le(F_ADD_288, F_ADD_288, F_ADD_288, int *, int);
 void f_sl_elv_add_le(F_ADD_288, int);

#endif

```

5.3 f_nano_crypto_util.h File Reference

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

Data Structures

- struct **f_block_transfer_t**
- struct **f_nano_encrypted_wallet_t**
- struct **f_nano_crypto_wallet_t**
- struct **f_nano_wallet_info_bdy_t**
- struct **f_nano_wallet_info_t**

Macros

- #define **MAX_STR_NANO_CHAR** (size_t)70
- #define **PUB_KEY_EXTENDED_MAX_LEN** (size_t)40
- #define **NANO_PREFIX** "nano_"
- #define **XRB_PREFIX** "xrb_"
- #define **NANO_ENCRYPTED_SEED_FILE** "/spiffs/secure/nano.nse"
- #define **NANO_PASSWD_MAX_LEN** (size_t)80
- #define **STR_NANO_SZ** (size_t)66
- #define **NANO_FILE_WALLETS_INFO** "/spiffs/secure/walletsinfo.i"
- #define **REP_XRB** (uint8_t)0x4
- #define **SENDER_XRB** (uint8_t)0x02
- #define **DEST_XRB** (uint8_t)0x01

Typedefs

- typedef uint8_t **NANO_SEED**[crypto_sign_SEEDBYTES]
- typedef uint8_t **f_uint128_t**[16]
- typedef uint8_t **NANO_PRIVATE_KEY**[sizeof(**NANO_SEED**)]
- typedef uint8_t **NANO_PRIVATE_KEY_EXTENDED**[crypto_sign_ed25519_SECRETKEYBYTES]
- typedef uint8_t **NANO_PUBLIC_KEY**[crypto_sign_ed25519_PUBLICKEYBYTES]
- typedef uint8_t **NANO_PUBLIC_KEY_EXTENDED**[**PUB_KEY_EXTENDED_MAX_LEN**]
- typedef enum **f_nano_err_t** **f_nano_err**
- typedef enum **f_write_seed_err_t** **f_write_seed_err**
- typedef enum **f_file_info_err_t** **F_FILE_INFO_ERR**

Enumerations

- enum **f_nano_err_t** {
NANO_ERR_OK =0, **NANO_ERR_CANT_PARSE_BN_STR** =5151, **NANO_ERR_MALLOC**, **NANO_E↩
RR_CANT_PARSE_FACTOR**,
NANO_ERR_MPI_MULT, **NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER**, **NANO_ERR_EMPTY_↩
STR**, **NANO_ERR_CANT_PARSE_VALUE**,
NANO_ERR_PARSE_MPI_TO_STR, **NANO_ERR_CANT_COMPLETE_NULL_CHAR**, **NANO_ERR_C↩
ANT_PARSE_TO_MPI**, **NANO_ERR_INSUFICIENT_FUNDS**,
NANO_ERR_SUB_MPI, **NANO_ERR_ADD_MPI**, **NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEG↩
ATIVE**, **NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO**,
NANO_ERR_NO_SENSE_BALANCE_NEGATIVE, **NANO_ERR_VAL_A_INVALID_MODE**, **NANO_ER↩
R_CANT_PARSE_TO_TEMP_UINT128_T**, **NANO_ERR_VAL_B_INVALID_MODE**,
NANO_ERR_CANT_PARSE_RAW_A_TO_MPI, **NANO_ERR_CANT_PARSE_RAW_B_TO_MPI**, **NAN↩
O_ERR_UNKNOWN_ADD_SUB_MODE**, **NANO_ERR_INVALID_RES_OUTPUT** }
- enum **f_write_seed_err_t** {
WRITE_ERR_OK =0, **WRITE_ERR_NULL_PASSWORD** =7180, **WRITE_ERR_EMPTY_STRING**, **WRI↩
TE_ERR_MALLOC**,
WRITE_ERR_ENCRYPT_PRIV_KEY, **WRITE_ERR_GEN_SUB_PRIV_KEY**, **WRITE_ERR_GEN_MAIN↩
_PRIV_KEY**, **WRITE_ERR_ENCRYPT_SUB_BLOCK**,
WRITE_ERR_UNKNOWN_OPTION, **WRITE_ERR_FILE_ALREDY_EXISTS**, **WRITE_ERR_CREATING↩
_FILE**, **WRITE_ERR_WRITING_FILE** }
- enum **f_file_info_err_t** {
F_FILE_INFO_ERR_OK =0, **F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE** =7001, **F_FILE_INFO_ER↩
R_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND**, **F_FILE_INFO_ERR_CANT_DELETE_NANO_IN↩
FO_FILE**,
F_FILE_INFO_ERR_MALLOC, **F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE**,
F_FILE_INFO_ERR_CANT_READ_INFO_FILE, **F_FILE_INFO_INVALID_HEADER_FILE**,
F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE, **F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL**,
F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE, **F_FILE_INFO_ERR_NANO_INVALID_MA↩
X_FEE_VALUE**,
F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO, **F_FILE_INFO_ERR_EXISTING_FILE**, **F_FILE_INFO↩
_ERR_CANT_WRITE_FILE_INFO** }

Functions

- struct **f_block_transfer_t** **__attribute__** ((packed)) F_BLOCK_TRANSFER
- int **f_cloud_crypto_wallet_nano_create_seed** (size_t, char ∗, char ∗)
- int **f_generate_nano_seed** (**NANO_SEED**, uint32_t)
- int **pk_to_wallet** (char ∗, char ∗, **NANO_PUBLIC_KEY_EXTENDED**)
- int **f_seed_to_nano_wallet** (**NANO_PRIVATE_KEY**, **NANO_PUBLIC_KEY**, **NANO_SEED**, uint32_t)
- char ∗ **f_nano_key_to_str** (char ∗, unsigned char ∗)
- int **f_nano_seed_to_bip39** (char ∗, size_t, size_t ∗, **NANO_SEED**, char ∗)
- int **f_bip39_to_nano_seed** (uint8_t ∗, char ∗, char ∗)
- int **f_parse_nano_seed_and_bip39_to_JSON** (char ∗, size_t, size_t ∗, void ∗, int, const char ∗)
- int **f_read_seed** (uint8_t ∗, const char ∗, void ∗, int, int)
- int **f_nano_raw_to_string** (char ∗, size_t ∗, size_t, void ∗, int)
- int **f_nano_valid_nano_str_value** (const char ∗)
- int **valid_nano_wallet** (const char ∗)
- int **nano_base_32_2_hex** (uint8_t ∗, char ∗)
- int **f_nano_transaction_to_JSON** (char ∗, size_t, size_t ∗, **NANO_PRIVATE_KEY_EXTENDED**, F_BL↩
OCK_TRANSFER ∗)
- int **valid_raw_balance** (const char ∗)
- int **is_null_hash** (uint8_t ∗)
- int **is_nano_prefix** (const char ∗, const char ∗)
- **F_FILE_INFO_ERR** **f_get_nano_file_info** (F_NANO_WALLET_INFO ∗)

- **F_FILE_INFO_ERR f_set_nano_file_info** (F_NANO_WALLET_INFO ∗, int)
- **f_nano_err f_nano_value_compare_value** (void ∗, void ∗, uint32_t ∗)
- **f_nano_err f_nano_verify_nano_funds** (void ∗, void ∗, void ∗, uint32_t)
- **f_nano_err f_nano_parse_raw_str_to_raw128_t** (uint8_t ∗, const char ∗)
- **f_nano_err f_nano_parse_real_str_to_raw128_t** (uint8_t ∗, const char ∗)
- **f_nano_err f_nano_add_sub** (void ∗, void ∗, void ∗, uint32_t)
- int **f_nano_sign_block** (F_BLOCK_TRANSFER ∗, F_BLOCK_TRANSFER ∗, **NANO_PRIVATE_KEY_E←↩ XTENDED**)

Variables

- uint8_t **preamble** [32]
- uint8_t **account** [32]
- uint8_t **previous** [32]
- uint8_t **representative** [32]
- **f_uint128_t balance**
- uint8_t **link** [32]
- uint8_t **signature** [64]
- uint8_t **prefixes**
- uint64_t **work**
- uint8_t **sub_salt** [32]
- uint8_t **iv** [16]
- uint8_t **reserved** [16]
- uint8_t **hash_sk_unencrypted** [32]
- uint8_t **sk_encrypted** [32]
- uint8_t **nano_hdr** [sizeof(NANO_WALLET_MAGIC)]
- uint32_t **ver**
- uint8_t **description** [F_DESC_SZ]
- uint8_t **salt** [32]
- F_ENCRYPTED_BLOCK **seed_block**
- uint8_t **wallet_prefix**
- uint32_t **last_used_wallet_number**
- char **wallet_representative** [**MAX_STR_NANO_CHAR**]
- char **max_fee** [F_RAW_STR_MAX_SZ]
- uint8_t **header** [sizeof(F_NANO_WALLET_INFO_MAGIC)]
- uint16_t **version**
- char **desc** [F_NANO_DESC_SZ]
- uint8_t **nanoseed_hash** [32]
- uint8_t **file_info_integrity** [32]
- F_NANO_WALLET_INFO_BODY **body**

5.3.1 Detailed Description

This API Integrates Nano Cryptocurrency to low computational devices.

Definition in file **f_nano_crypto_util.h**.

5.3.2 Macro Definition Documentation

5.3.2.1 DEST_XRB

```
#define DEST_XRB (uint8_t)0x01
```

Definition at line **409** of file **f_nano_crypto_util.h**.

5.3.2.2 MAX_STR_NANO_CHAR

```
#define MAX_STR_NANO_CHAR (size_t)70
```

Defines a max size of Nano char (70 bytes)

Definition at line **129** of file **f_nano_crypto_util.h**.

5.3.2.3 NANO_ENCRYPTED_SEED_FILE

```
#define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
```

Path to non deterministic encrypted file with password.

File containing the SEED of the Nano wallets generated by TRNG (if available in your Hardware) or PRNG. Default name: "nano.nse"

Definition at line **172** of file **f_nano_crypto_util.h**.

5.3.2.4 NANO_FILE_WALLETS_INFO

```
#define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
```

Custom information file path about Nano SEED wallet stored in "walletsinfo.i".

Definition at line **190** of file **f_nano_crypto_util.h**.

5.3.2.5 NANO_PASSWD_MAX_LEN

```
#define NANO_PASSWD_MAX_LEN (size_t)80
```

Password max length.

Definition at line **178** of file **f_nano_crypto_util.h**.

5.3.2.6 NANO_PREFIX

```
#define NANO_PREFIX "nano_"
```

Nano prefix.

Definition at line **141** of file **f_nano_crypto_util.h**.

5.3.2.7 PUB_KEY_EXTENDED_MAX_LEN

```
#define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
```

Max size of public key (extended)

Definition at line **135** of file **f_nano_crypto_util.h**.

5.3.2.8 REP_XRB

```
#define REP_XRB (uint8_t)0x4
```

Representative XRB flag.

Destination XRB flag.

Sender XRB flag.

5.3.2.9 SENDER_XRB

```
#define SENDER_XRB (uint8_t)0x02
```

Definition at line **403** of file **f_nano_crypto_util.h**.

5.3.2.10 STR_NANO_SZ

```
#define STR_NANO_SZ (size_t)66
```

String size of Nano encoded Base32 including NULL char.

Definition at line **184** of file **f_nano_crypto_util.h**.

5.3.2.11 XRB_PREFIX

```
#define XRB_PREFIX "xrb_"
```

XRB (old Raiblocks) prefix.

Definition at line **147** of file **f_nano_crypto_util.h**.

5.3.3 Typedef Documentation

5.3.3.1 F_FILE_INFO_ERR

```
F_FILE_INFO_ERR
```

Typedef Error enumerator for info file functions.

5.3.3.2 f_nano_err

```
f_nano_err
```

Error function enumerator.

See also

f_nano_err_t (p. **??**)

5.3.3.3 f_uint128_t

```
f_uint128_t
```

128 bit big number of Nano balance

Definition at line **202** of file **f_nano_crypto_util.h**.

5.3.3.4 f_write_seed_err

```
typedef enum f_write_seed_err_t f_write_seed_err
```

5.3.3.5 NANO_PRIVATE_KEY

`NANO_PRIVATE_KEY`

Size of Nano Private Key.

Definition at line **212** of file **f_nano_crypto_util.h**.

5.3.3.6 NANO_PRIVATE_KEY_EXTENDED

`NANO_PRIVATE_KEY_EXTENDED`

Size of Nano Private Key extended.

Definition at line **218** of file **f_nano_crypto_util.h**.

5.3.3.7 NANO_PUBLIC_KEY

`NANO_PUBLIC_KEY`

Size of Nano Public Key.

Definition at line **224** of file **f_nano_crypto_util.h**.

5.3.3.8 NANO_PUBLIC_KEY_EXTENDED

`NANO_PUBLIC_KEY_EXTENDED`

Size of Public Key Extended.

Definition at line **230** of file **f_nano_crypto_util.h**.

5.3.3.9 NANO_SEED

`NANO_SEED`

Size of Nano SEED.

Definition at line **196** of file **f_nano_crypto_util.h**.

5.3.4 Enumeration Type Documentation

5.3.4.1 f_file_info_err_t

`enum` **`f_file_info_err_t`**

Enumerator

F_FILE_INFO_ERR_OK	SUCCESS.
F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE	Can't open info file.
F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NO↩ T_FOUND	Encrypted file with Nano SEED not found.
F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE	Can not delete Nano info file.
F_FILE_INFO_ERR_MALLOC	Fatal Error MALLOC.
F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYP↩ TED_FILE	Can not read encrypted Nano SEED in file.
F_FILE_INFO_ERR_CANT_READ_INFO_FILE	Can not read info file.
F_FILE_INFO_INVALID_HEADER_FILE	Invalid info file header.
F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE	Invalid SHA256 info file.
F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL	Nano SEED hash failed.
F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE	Invalid representative.
F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE	Invalid max fee value.
F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO	Can not open info file for write.
F_FILE_INFO_ERR_EXISTING_FILE	Error File Exists.
F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO	Can not write info file.

Definition at line **515** of file **f_nano_crypto_util.h**.

5.3.4.2 f_nano_err_t

`enum` **`f_nano_err_t`**

Enumerator

NANO_ERR_OK	SUCCESS.
NANO_ERR_CANT_PARSE_BN_STR	Can not parse string big number.
NANO_ERR_MALLOC	Fatal ERROR MALLOC.
NANO_ERR_CANT_PARSE_FACTOR	Can not parse big number factor.
NANO_ERR_MPI_MULT	Error multiplication MPI.
NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER	Can not parse to block transfer.
NANO_ERR_EMPTY_STR	Error empty string.
NANO_ERR_CANT_PARSE_VALUE	Can not parse value.
NANO_ERR_PARSE_MPI_TO_STR	Can not parse MPI to string.
NANO_ERR_CANT_COMPLETE_NULL_CHAR	Can not complete NULL char.
NANO_ERR_CANT_PARSE_TO_MPI	Can not parse to MPI.
NANO_ERR_INSUFICIENT_FUNDS	Insuficient funds.
NANO_ERR_SUB_MPI	Error subtract MPI.
NANO_ERR_ADD_MPI	Error add MPI.
NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE	Does not make sense send negativative balance.
NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO	Does not make sense send empty value.
NANO_ERR_NO_SENSE_BALANCE_NEGATIVE	Does not make sense negative balance.
NANO_ERR_VAL_A_INVALID_MODE	Invalid A mode value.
NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T	Can not parse temporary memory to uint_128_t.
NANO_ERR_VAL_B_INVALID_MODE	Invalid A mode value.

Enumerator

NANO_ERR_CANT_PARSE_RAW_A_TO_MPI	Can not parse raw A value to MPI.
NANO_ERR_CANT_PARSE_RAW_B_TO_MPI	Can not parse raw B value to MPI.
NANO_ERR_UNKNOWN_ADD_SUB_MODE	Unknown ADD/SUB mode.
NANO_ERR_INVALID_RES_OUTPUT	Invalid output result.

Definition at line **274** of file **f_nano_crypto_util.h**.

5.3.4.3 f_write_seed_err_t

```
enum f_write_seed_err_t
```

Enumerator

WRITE_ERR_OK	Error SUCCESS.
WRITE_ERR_NULL_PASSWORD	Error NULL password.
WRITE_ERR_EMPTY_STRING	Empty string.
WRITE_ERR_MALLOC	Error MALLOC.
WRITE_ERR_ENCRYPT_PRIV_KEY	Error encrypt private key.
WRITE_ERR_GEN_SUB_PRIV_KEY	Can not generate sub private key.
WRITE_ERR_GEN_MAIN_PRIV_KEY	Can not generate main private key.
WRITE_ERR_ENCRYPT_SUB_BLOCK	Can not encrypt sub block.
WRITE_ERR_UNKNOWN_OPTION	Unknown option.
WRITE_ERR_FILE_ALREDY_EXISTS	File already exists.
WRITE_ERR_CREATING_FILE	Can not create file.
WRITE_ERR_WRITING_FILE	Can not write file.

Definition at line **411** of file **f_nano_crypto_util.h**.

5.3.5 Function Documentation

5.3.5.1 __attribute__()

```
struct f_nano_wallet_info_t __attribute__ (
             (packed) )
```

5.3.5.2 f_bip39_to_nano_seed()

```
int f_bip39_to_nano_seed (
            uint8_t * seed,
            char * str,
            char * dictionary )
```

Parse Nano Bip39 encoded string to raw Nano SEED given a dictionary file.

Parameters

`out`	*seed*	Nano SEED
`in`	*str*	A encoded Bip39 string pointer
`in`	*dictionary*	A string pointer path to file

WARNING Sensive data. Do not share any SEED or Bip39 encoded string !

Return values

0	On Success, otherwise Error

See also

f_nano_seed_to_bip39() (p. **??**)

5.3.5.3 f_cloud_crypto_wallet_nano_create_seed()

```
int f_cloud_crypto_wallet_nano_create_seed (
            size_t entropy,
            char * file_name,
            char * password )
```

Generates a new SEED and saves it to an non deterministic encrypted file.

password is mandatory

Parameters

`in`	*entropy*	Entropy type. Entropy type are: F_ENTROPY_TYPE_PARANOIC F_ENTROPY_TYPE_EXCELENT F_ENTROPY_TYPE_GOOD F_ENTROPY_TYPE_NOT_ENOUGH F_ENTROPY_TYPE_NOT_RECOMENDED
`in`	*file_name*	The file and path to be stored in your file system directory. It can be *NULL*. If you parse a *NULL* value then file will be stored in *NANO_ENCRYPTED_SEED_FILE* variable file system pointer.
`in`	*password*	Password of the encrypted file. It can NOT be *NULL* or EMPTY

WARNING

f_cloud_crypto_wallet_nano_create_seed() *(*p. **??***)* does not verify your password. It is recommended to use a strong password like symbols, capital letters and numbers to keep your SEED safe and avoid brute force attacks.

You can use ***f_pass_must_have_at_least()*** *(*p. **??***)* function to check passwords strenght

Return values

0	On Success, otherwise Error

5.3.5.4 f_generate_nano_seed()

```
int f_generate_nano_seed (
              NANO_SEED seed,
             uint32_t entropy )
```

Generates a new SEED and stores it to *seed* pointer.

Parameters

out	*seed*	SEED generated in system PRNG or TRNG
in	*entropy*	Entropy type. Entropy type are: F_ENTROPY_TYPE_PARANOIC F_ENTROPY_TYPE_EXCELENT F_ENTROPY_TYPE_GOOD F_ENTROPY_TYPE_NOT_ENOUGH F_ENTROPY_TYPE_NOT_RECOMENDED

Return values

0	On Success, otherwise Error

5.3.5.5 f_get_nano_file_info()

```
 F_FILE_INFO_ERR f_get_nano_file_info (
             F_NANO_WALLET_INFO * info )
```

Opens default file *walletsinfo.i* (if exists) containing information *F_NANO_WALLET_INFO* structure and parsing to pointer *info* if success.

Parameters

out	*info*	Pointer to buffer to be parsed struct from *$PATH/walletsinfo.i* file.

Return values

F_FILE_INFO_ERR_OK	If Success, otherwise *F_FILE_INFO_ERR* enum type error

See also

F_FILE_INFO_ERR (p. **??**) enum type error for detailed error and **f_nano_wallet_info_t** (p. **??**) for info type details

5.3.5.6 f_nano_add_sub()

```
f_nano_err f_nano_add_sub (
            void * res,
            void * valA,
            void * valB,
            uint32_t mode )
```

Add/Subtract two Nano balance values and stores value in *res*

Parameters

`out`	*res*	Result value res = valA + valB or res = valA - valB
`in`	*valA*	Input balance A value
`in`	*valB*	Input balance B value
`in`	*mode*	Mode type: • *F_NANO_ADD_A_B* valA + valB • *F_NANO_SUB_A_B* valA - valB • *F_NANO_A_RAW_128* if *balance* is big number raw buffer type • *F_NANO_A_RAW_STRING* if *balance* is big number raw string type • *F_NANO_A_REAL_STRING* if *balance* is real number string type • *F_NANO_B_RAW_128* if *value_to_send* is big number raw buffer type • *F_NANO_B_RAW_STRING* if *value_to_send* is big number raw string type • *F_NANO_B_REAL_STRING* if *value_to_send* is real number string type

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.7 f_nano_key_to_str()

```
char * f_nano_key_to_str (
            char * out,
            unsigned char * key )
```

Parse a raw binary public key to string.

Parameters

out	*out*	Pointer to outuput string
in	*in*	Pointer to raw public key

Returns

A pointer to output string

5.3.5.8 f_nano_parse_raw_str_to_raw128_t()

```
f_nano_err f_nano_parse_raw_str_to_raw128_t (
            uint8_t * res,
            const char * raw_str_value )
```

Parse a raw string balance to raw big number 128 bit.

Parameters

out	*res*	Binary raw balance
in	*raw_str_value*	Raw balance string

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.9 f_nano_parse_real_str_to_raw128_t()

```
f_nano_err f_nano_parse_real_str_to_raw128_t (
            uint8_t * res,
            const char * real_str_value )
```

Parse a real string balance to raw big number 128 bit.

Parameters

out	*res*	Binary raw balance
in	*real_str_value*	Real balance string

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.10 f_nano_raw_to_string()

```
int f_nano_raw_to_string (
            char * str,
            size_t * olen,
            size_t str_sz,
            void * raw,
            int raw_type )
```

Converts Nano raw balance [string | f_uint128_t] to real string value.

Parameters

out	*str*	Output real string value
out	*olen*	Size of output real string value. It can be NULL. If NULL output *str* will have a NULL char at the end.
in	*str_sz*	Size of *str* buffer
in	*raw*	Raw balance.
in	*raw_type*	Raw balance type: • F_RAW_TO_STR_UINT128 for raw **f_uint128_t** balance • F_RAW_TO_STR_STRING for raw **char** balance

Return values

0	On Success, otherwise Error

See also

f_nano_valid_nano_str_value() (p. **??**)

5.3.5.11 f_nano_seed_to_bip39()

```
int f_nano_seed_to_bip39 (
            char * buf,
            size_t buf_sz,
            size_t * out_buf_len,
            NANO_SEED seed,
            char * dictionary_file )
```

Parse Nano SEED to Bip39 encoding given a dictionary file.

Parameters

out	*buf*	Output string containing encoded Bip39 SEED
in	*buf_sz*	Size of memory of buf pointer
out	*out_buf_len*	If *out_buf_len* is NOT NULL then *out_buf_len* returns the size of string encoded Bip39 and *out* with non NULL char. If *out_buf_len* is NULL then *out* has a string encoded Bip39 with a NULL char.
in	*seed*	Nano SEED
in	*dictionary_file*	Path to dictionary file

WARNING Sensive data. Do not share any SEED or Bip39 encoded string !

Return values

0	On Success, otherwise Error

See also

f_bip39_to_nano_seed() (p. **??**)

5.3.5.12 f_nano_sign_block()

```
int f_nano_sign_block (
            F_BLOCK_TRANSFER * user_block,
            F_BLOCK_TRANSFER * fee_block,
            NANO_PRIVATE_KEY_EXTENDED private_key )
```

Signs *user_block* and worker *fee_block* given a private key *private_key*

Parameters

in,out	*user_block*	User block to be signed with a private key *private_key*
in,out	*fee_block*	Fee block to be signed with a private key *private_key*. Can be NULL if worker does not require fee
in	*private_key*	Private key to sign block(s)

Return values

0	If Success, otherwise error

See also

f_nano_transaction_to_JSON() (p. **??**)

5.3.5.13 f_nano_transaction_to_JSON()

```
int f_nano_transaction_to_JSON (
             char * str,
             size_t str_len,
             size_t * str_out,
              NANO_PRIVATE_KEY_EXTENDED private_key,
             F_BLOCK_TRANSFER * block_transfer )
```

Sign a block pointed in *block_transfer* with a given *private_key* and stores signed block to *block_transfer* and parse to JSON Nano RPC.

Parameters

`out`	*str*	A string pointer to store JSON Nano RPC
`in`	*str_len*	Size of buffer in *str* pointer
`out`	*str_out*	Size of JSON string. *str_out* can be NULL
`in`	*private_key*	Private key to sign the block *block_transfer*
`in,out`	*block_transfer*	Nano block containing raw data to be stored in Nano Blockchain

WARNING Sensive data. Do not share any PRIVATE KEY

Return values

0	On Success, otherwise Error

5.3.5.14 f_nano_valid_nano_str_value()

```
int f_nano_valid_nano_str_value (
             const char * str )
```

Check if a real string or raw string are valid Nano balance.

Parameters

`in`	*str*	Value to be checked

Return values

0	If valid, otherwise is invalid

See also

f_nano_raw_to_string() (p. **??**)

5.3.5.15 f_nano_value_compare_value()

```
f_nano_err f_nano_value_compare_value (
            void * valA,
            void * valB,
            uint32_t * mode_compare )
```

Comparare two Nano balance.

Parameters

`in`	*valA*	Nano balance value A
`in`	*valB*	Nano balance value B
`in,out`	*mode_compare*	Input mode and output result Input mode: • *F_NANO_A_RAW_128* if *valA* is big number raw buffer type • *F_NANO_A_RAW_STRING* if *valA* is big number raw string type • *F_NANO_A_REAL_STRING* if *valA* is real number string type • *F_NANO_B_RAW_128* if *valB* is big number raw buffer type • *F_NANO_B_RAW_STRING* if *valB* is big number raw string type • *F_NANO_B_REAL_STRING* if *valB* is real number string type Output type: • *F_NANO_COMPARE_EQ* If *valA* is greater than *valB* • *F_NANO_COMPARE_LT* if *valA* is lesser than *valB* • *F_NANO_COMPARE_LEQ* if *valA* is lesser or equal than *valB* • *F_NANO_COMPARE_GT* if *valA* is greater than *valB* • *F_NANO_COMPARE_GEQ* If *valA* is greater or equal than *valB*

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.16 f_nano_verify_nano_funds()

```
f_nano_err f_nano_verify_nano_funds (
            void * balance,
            void * value_to_send,
            void * fee,
            uint32_t mode )
```

Check if Nano balance has sufficient funds.

Parameters

in	*balance*	Nano balance
in	*value_to_send*	Value to send
in	*fee*	Fee value (it can be NULL)
in	*mode*	Value type mode • *F_NANO_A_RAW_128* if *balance* is big number raw buffer type • *F_NANO_A_RAW_STRING* if *balance* is big number raw string type • *F_NANO_A_REAL_STRING* if *balance* is real number string type • *F_NANO_B_RAW_128* if *value_to_send* is big number raw buffer type • *F_NANO_B_RAW_STRING* if *value_to_send* is big number raw string type • *F_NANO_B_REAL_STRING* if *value_to_send* is real number string type • *F_NANO_C_RAW_128* if *fee* is big number raw buffer type (can be ommited if *fee* is NULL) • *F_NANO_C_RAW_STRING* if *fee* is big number raw string type (can be ommited if *fee* is NULL) • *F_NANO_C_REAL_STRING* if *fee* is real number string type (can be ommited if *fee* is NULL)

Return values

NANO_ERR_OK	If Success, otherwise f_nano_err_t enum type error

See also

f_nano_err_t (p. **??**) for **f_nano_err** (p. **??**) enum error type

5.3.5.17 f_parse_nano_seed_and_bip39_to_JSON()

```
int f_parse_nano_seed_and_bip39_to_JSON (
            char * dest,
            size_t dest_sz,
            size_t * olen,
            void * source_data,
            int source,
            const char * password )
```

Parse Nano SEED and Bip39 to JSON given a encrypted data in memory or encrypted data in file or unencrypted seed in memory.

Parameters

`out`	*dest*	Destination JSON string pointer
`in`	*dest_sz*	Buffer size of *dest* pointer
`out`	*olen*	Size of the output JSON string. If NULL string JSON returns a NULL char at the end of string otherwise it will return the size of the string is stored into *olen* variable without NULL string in *dest*
`in`	*source_data*	Input data source (encrypted file | encrypted data in memory | unencrypted seed in memory)
`in`	*source*	Source data type: • PARSE_JSON_READ_SEED_GENERIC: If seed are in memory pointed in *source_data*. Password is ignored. Can be NULL. • READ_SEED_FROM_STREAM: Read encrypted data from stream pointed in *source_data*. Password is required. • READ_SEED_FROM_FILE: Read encrypted data stored in a file where *source_data* is path to file. Password is required.
`in`	*password*	Required for READ_SEED_FROM_STREAM and READ_SEED_FROM_FILE sources

WARNING Sensive data. Do not share any SEED or Bip39 encoded string !

Return values

0	On Success, otherwise Error

See also

f_read_seed() (p. **??**)

5.3.5.18 f_read_seed()

```
int f_read_seed (
            uint8_t * seed,
            const char * passwd,
            void * source_data,
            int force_read,
            int source )
```

Extracts a Nano SEED from encrypted stream in memory or in a file.

Parameters

out	*seed*	Output Nano SEED
in	*passwd*	Password (always required)
in	*source_data*	Encrypted source data from memory or path pointed in *source_data*
in	*force_read*	If non zero value then forces reading from a corrupted file. This param is ignored when reading *source_data* from memory
in	*source*	Source data type: • READ_SEED_FROM_STREAM: Read encrypted data from stream pointed in *source_data*. Password is required. • READ_SEED_FROM_FILE: Read encrypted data stored in a file where *source_data* is path to file. Password is required.

WARNING Sensive data. Do not share any SEED !

Return values

0	On Success, otherwise Error

See also

f_parse_nano_seed_and_bip39_to_JSON() (p. **??**)

5.3.5.19 f_seed_to_nano_wallet()

```
int f_seed_to_nano_wallet (
              NANO_PRIVATE_KEY private_key,
              NANO_PUBLIC_KEY public_key,
              NANO_SEED seed,
             uint32_t wallet_number )
```

Extracts one key pair from Nano SEED given a wallet number.

Parameters

out	*private_key*	Private key of the *wallet_number* from given *seed*
out	*public_key*	Public key of the *wallet_number* from given *seed*
in,out	*seed*	Nano SEED
in	*wallet_number*	Wallet number of key pair to be extracted from Nano SEED

WARNING 1:

- Seed must be read from memory
- Seed is destroyed when extracting public and private keys

WARNING 2:

- Never expose SEED and private key. This function destroys seed and any data after execution and finally parse public and private keys to output.

Return values

0	On Success, otherwise Error

5.3.5.20 f_set_nano_file_info()

```
F_FILE_INFO_ERR f_set_nano_file_info (
            F_NANO_WALLET_INFO * info,
            int overwrite_existing_file )
```

Saves wallet information stored at buffer struct *info* to file *walletsinfo.i*

Parameters

in	*info*	Pointer to data to be saved at *$PATH/walletsinfo.i* file.
in	*overwrite_existing_file*	If non zero then overwrites file *$PATH/walletsinfo.i*

Return values

F_FILE_INFO_ERR_OK	If Success, otherwise *F_FILE_INFO_ERR* enum type error

See also

F_FILE_INFO_ERR (p. **??**) enum type error for detailed error and **f_nano_wallet_info_t** (p. **??**) for info type details

5.3.5.21 is_nano_prefix()

```
int is_nano_prefix (
            const char * nano_wallet,
            const char * prefix )
```

Checks *prefix* in *nano_wallet*

Parameters

in	*nano_wallet*	Base32 Nano wallet encoded string
in	*prefix*	Prefix type • NANO_PREFIX for nano_
		• XRB_PREFIX for xrb_

Return values

1	If *prefix* in *nano_wallet*, otherwise 0

5.3.5.22 is_null_hash()

```
int is_null_hash (
            uint8_t * hash )
```

Check if 32 bytes hash is filled with zeroes.

Parameters

in	*hash*	32 bytes binary *hash*

Return values

1	If zero filled buffer, otherwise 0

5.3.5.23 nano_base_32_2_hex()

```
int nano_base_32_2_hex (
            uint8_t * res,
            char * str_wallet )
```

Parse Nano Base32 wallet string to public key binary.

Parameters

out	*res*	Output raw binary public key
in	*str_wallet*	Valid Base32 encoded Nano string to be parsed

Return values

0	On Success, otherwise Error

See also

pk_to_wallet() (p. **??**)

5.3.5.24 pk_to_wallet()

```
int pk_to_wallet (
            char * out,
            char * prefix,
             NANO_PUBLIC_KEY_EXTENDED pubkey_extended )
```

Parse a Nano public key to Base32 Nano wallet string.

Parameters

out	*out*	Output string containing the wallet
in	*prefix*	Nano prefix. *NANO_PREFIX* for nano_ *XRB_PREFIX* for xrb_
in,out	*pubkey_extended*	Public key to be parsed to string

WARNING: *pubkey_extended* is destroyed when parsing to Nano base32 encoding

Return values

0	On Success, otherwise Error

See also

nano_base_32_2_hex() (p. **??**)

5.3.5.25 valid_nano_wallet()

```
int valid_nano_wallet (
            const char * wallet )
```

Check if a string containing a Base32 Nano wallet is valid.

Parameters

in	*wallet*	Base32 Nano wallet encoded string

Return values

0	If valid wallet otherwise is invalid

5.3.5.26 valid_raw_balance()

```
int valid_raw_balance (
            const char * balance )
```

Checks if a string buffer pointed in *balance* is a valid raw balance.

Parameters

in	*balance*	Pointer containing a string buffer

Return values

0	On Success, otherwise Error

5.3.6 Variable Documentation

5.3.6.1 account

```
uint8_t account[32]
```

Account in raw binary data.

Definition at line **234** of file **f_nano_crypto_util.h**.

5.3.6.2 balance

```
f_uint128_t balance
```

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **242** of file **f_nano_crypto_util.h**.

5.3.6.3 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **242** of file **f_nano_crypto_util.h**.

5.3.6.4 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **236** of file **f_nano_crypto_util.h**.

5.3.6.5 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **236** of file **f_nano_crypto_util.h**.

5.3.6.6 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **240** of file **f_nano_crypto_util.h**.

5.3.6.7 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **238** of file **f_nano_crypto_util.h**.

5.3.6.8 header

```
uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)]
```

Header magic.

Definition at line **232** of file **f_nano_crypto_util.h**.

5.3.6.9 iv

```
uint8_t iv
```

Initial sub vector.

Initial vector of first encryption layer.

Definition at line **234** of file **f_nano_crypto_util.h**.

5.3.6.10 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **234** of file **f_nano_crypto_util.h**.

5.3.6.11 link

```
uint8_t link[32]
```

link or destination account

Definition at line **244** of file **f_nano_crypto_util.h**.

5.3.6.12 max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

Definition at line **238** of file **f_nano_crypto_util.h**.

5.3.6.13 nano_hdr

```
uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)]
```

Header of the file.

Definition at line **232** of file **f_nano_crypto_util.h**.

5.3.6.14 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **238** of file **f_nano_crypto_util.h**.

5.3.6.15 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **232** of file **f_nano_crypto_util.h**.

5.3.6.16 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **248** of file **f_nano_crypto_util.h**.

5.3.6.17 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **236** of file **f_nano_crypto_util.h**.

5.3.6.18 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **238** of file **f_nano_crypto_util.h**.

5.3.6.19 reserved

```
uint8_t reserved
```

Reserved (not used)

Reserved.

Definition at line **236** of file **f_nano_crypto_util.h**.

5.3.6.20 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **238** of file **f_nano_crypto_util.h**.

5.3.6.21 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **242** of file **f_nano_crypto_util.h**.

5.3.6.22 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **246** of file **f_nano_crypto_util.h**.

5.3.6.23 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **240** of file **f_nano_crypto_util.h**.

5.3.6.24 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

Definition at line **232** of file **f_nano_crypto_util.h**.

5.3.6.25 ver

```
uint32_t ver
```

Version of the file.

Definition at line **234** of file **f_nano_crypto_util.h**.

5.3.6.26 version

```
uint16_t version
```

Version.

Definition at line **234** of file **f_nano_crypto_util.h**.

5.3.6.27 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **232** of file **f_nano_crypto_util.h**.

5.3.6.28 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **236** of file **f_nano_crypto_util.h**.

5.3.6.29 work

```
uint64_t work
```

Internal use for this API.

Definition at line **250** of file **f_nano_crypto_util.h**.

5.4 f_nano_crypto_util.h

```
/*
    AUTHOR: Fábio Pereira da Silva
    YEAR: 2019-20
    LICENSE: MIT
    EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
*/

#include <stdint.h>
#include "f_util.h"

#ifndef F_DOC_SKIP

 #ifdef F_XTENSA

  #ifndef F_ESP32
   #define F_ESP32
  #endif

  #include "esp_system.h"

 #endif

 #include "sodium/crypto_generichash.h"
 #include "sodium/crypto_sign.h"
 #include "sodium.h"

 #ifdef F_ESP32

  #include "sodium/private/curve25519_ref10.h"

 #else

  #include "sodium/private/ed25519_ref10.h"

  #define ge_p3 ge25519_p3
  #define sc_reduce sc25519_reduce
  #define sc_muladd sc25519_muladd
  #define ge_scalarmult_base ge25519_scalarmult_base
  #define ge_p3_tobytes ge25519_p3_tobytes

 #endif

#endif

#ifdef __cplusplus
extern "C" {
#endif

#define MAX_STR_NANO_CHAR (size_t)70 //5+56+8+1

#define PUB_KEY_EXTENDED_MAX_LEN (size_t)40

#define NANO_PREFIX "nano_"

#define XRB_PREFIX "xrb_"

#ifdef F_ESP32

#define BIP39_DICTIONARY "/spiffs/dictionary.dic"
#else

 #ifndef F_DOC_SKIP
  //#define BIP39_DICTIONARY "/spiffs/dictionary.dic"
  #define BIP39_DICTIONARY_SAMPLE "../../dictionary.dic"
  #define BIP39_DICTIONARY "dictionary.dic"
 #endif

#endif
```

```

#define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"

#define NANO_PASSWD_MAX_LEN (size_t)80

#define STR_NANO_SZ (size_t)66// 65+1 Null included

#define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"

typedef uint8_t NANO_SEED[crypto_sign_SEEDBYTES];

typedef uint8_t f_uint128_t[16];

#ifndef F_DOC_SKIP
 #define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof(NANO_SEED)+1
#endif

typedef uint8_t NANO_PRIVATE_KEY[sizeof(NANO_SEED)];

typedef uint8_t NANO_PRIVATE_KEY_EXTENDED[crypto_sign_ed25519_SECRETKEYBYTES];

typedef uint8_t NANO_PUBLIC_KEY[crypto_sign_ed25519_PUBLICKEYBYTES];

typedef uint8_t NANO_PUBLIC_KEY_EXTENDED[PUB_KEY_EXTENDED_MAX_LEN];

typedef struct f_block_transfer_t {
   uint8_t preamble[32];
   uint8_t account[32];
   uint8_t previous[32];
   uint8_t representative[32];
   f_uint128_t balance;
   uint8_t link[32];
   uint8_t signature[64];
   uint8_t prefixes;
   uint64_t work;
} __attribute__((packed)) F_BLOCK_TRANSFER;

#ifndef F_DOC_SKIP
 #define F_BLOCK_TRANSFER_SIGNABLE_SZ
 (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_t)-sizeof(uint8_t))
#endif

typedef enum f_nano_err_t {
   NANO_ERR_OK=0,
   NANO_ERR_CANT_PARSE_BN_STR=5151,
   NANO_ERR_MALLOC,
   NANO_ERR_CANT_PARSE_FACTOR,
   NANO_ERR_MPI_MULT,
   NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER,
   NANO_ERR_EMPTY_STR,
   NANO_ERR_CANT_PARSE_VALUE,
   NANO_ERR_PARSE_MPI_TO_STR,
   NANO_ERR_CANT_COMPLETE_NULL_CHAR,
   NANO_ERR_CANT_PARSE_TO_MPI,
   NANO_ERR_INSUFICIENT_FUNDS,
   NANO_ERR_SUB_MPI,
   NANO_ERR_ADD_MPI,
   NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE,
   NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO,
   NANO_ERR_NO_SENSE_BALANCE_NEGATIVE,
   NANO_ERR_VAL_A_INVALID_MODE,
   NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T,
   NANO_ERR_VAL_B_INVALID_MODE,
   NANO_ERR_CANT_PARSE_RAW_A_TO_MPI,
   NANO_ERR_CANT_PARSE_RAW_B_TO_MPI,
   NANO_ERR_UNKNOWN_ADD_SUB_MODE,
   NANO_ERR_INVALID_RES_OUTPUT
} f_nano_err;

#ifndef F_DOC_SKIP

 #define READ_SEED_FROM_STREAM (int)1
 #define READ_SEED_FROM_FILE (int)2
 #define WRITE_SEED_TO_STREAM (int)4
 #define WRITE_SEED_TO_FILE (int)8
 #define PARSE_JSON_READ_SEED_GENERIC (int)16
 #define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)

#endif

typedef struct f_nano_encrypted_wallet_t {
   uint8_t sub_salt[32];
   uint8_t iv[16];
   uint8_t reserved[16];
   uint8_t hash_sk_unencrypted[32];
   uint8_t sk_encrypted[32];
} __attribute__ ((packed)) F_ENCRYPTED_BLOCK;
```

```

#ifndef F_DOC_SKIP

 static const uint8_t NANO_WALLET_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f',
'i', 'l', 'e', '_'};
 #define F_NANO_FILE_DESC "NANO Seed Encrypted file/stream. Keep it safe and backup it. This file is
 protected by password. BUY BITCOIN and NANO !!!"
 #define F_DESC_SZ (size_t) (160-sizeof(uint32_t))

#endif

typedef struct f_nano_crypto_wallet_t {
   uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)];
   uint32_t ver;
   uint8_t description[F_DESC_SZ];
   uint8_t salt[32];
   uint8_t iv[16];
   F_ENCRYPTED_BLOCK seed_block;
} __attribute__ ((packed)) F_NANO_CRYPTOWALLET;

#ifndef F_DOC_SKIP

_Static_assert((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1");
_Static_assert((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2");

#endif

#define REP_XRB (uint8_t)0x4

#define SENDER_XRB (uint8_t)0x02

#define DEST_XRB (uint8_t)0x01

typedef enum f_write_seed_err_t {
   WRITE_ERR_OK=0,
   WRITE_ERR_NULL_PASSWORD=7180,
   WRITE_ERR_EMPTY_STRING,
   WRITE_ERR_MALLOC,
   WRITE_ERR_ENCRYPT_PRIV_KEY,
   WRITE_ERR_GEN_SUB_PRIV_KEY,
   WRITE_ERR_GEN_MAIN_PRIV_KEY,
   WRITE_ERR_ENCRYPT_SUB_BLOCK,
   WRITE_ERR_UNKNOWN_OPTION,
   WRITE_ERR_FILE_ALREDY_EXISTS,
   WRITE_ERR_CREATING_FILE,
   WRITE_ERR_WRITING_FILE
} f_write_seed_err;

#ifndef F_DOC_SKIP

 #define F_RAW_TO_STR_UINT128 (int)1
 #define F_RAW_TO_STR_STRING (int)2
 #define F_RAW_STR_MAX_SZ (size_t)41 // 39 + '\0' + '.' -> 39 = log10(2^128)
 #define F_MAX_STR_RAW_BALANCE_MAX (size_t)40 //39+'\0'
 #define F_NANO_EMPTY_BALANCE "0.0"

#endif

typedef struct f_nano_wallet_info_bdy_t {
   uint8_t wallet_prefix; // 0 for NANO; 1 for XRB
   uint32_t last_used_wallet_number;
   char wallet_representative[MAX_STR_NANO_CHAR];
   char max_fee[F_RAW_STR_MAX_SZ];
   uint8_t reserved[44];
} __attribute__((packed)) F_NANO_WALLET_INFO_BODY;

#ifndef F_DOC_SKIP

 _Static_assert((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte
 aligned");

 #define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
 #define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
 static const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
'_', 'n', 'f', 'o', '_'};

 #define F_NANO_DESC_SZ (size_t)78

#endif

typedef struct f_nano_wallet_info_t {
   uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)];
   uint16_t version;
   char desc[F_NANO_DESC_SZ];
   uint8_t nanoseed_hash[32];
   uint8_t file_info_integrity[32];
   F_NANO_WALLET_INFO_BODY body;
} __attribute__((packed)) F_NANO_WALLET_INFO;

#ifndef F_DOC_SKIP

 _Static_assert((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned");

#endif

typedef enum f_file_info_err_t {
   F_FILE_INFO_ERR_OK=0,
   F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE=7001,
   F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND,
   F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE,
   F_FILE_INFO_ERR_MALLOC,
   F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE,
   F_FILE_INFO_ERR_CANT_READ_INFO_FILE,
   F_FILE_INFO_INVALID_HEADER_FILE,
   F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE,
   F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL,
   F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE,
   F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE,
   F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO,
   F_FILE_INFO_ERR_EXISTING_FILE,
   F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO
} F_FILE_INFO_ERR;

#ifndef F_DOC_SKIP

 #define F_NANO_ADD_A_B (uint32_t)(1<<0)
 #define F_NANO_SUB_A_B (uint32_t)(1<<1)
 #define F_NANO_A_RAW_128 (uint32_t)(1<<2)
 #define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
 #define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
 #define F_NANO_B_RAW_128 (uint32_t)(1<<5)
 #define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
 #define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
 #define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
 #define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
 #define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
 #define F_NANO_C_RAW_128 (uint32_t)(F_NANO_B_RAW_128<<16)
 #define F_NANO_C_RAW_STRING (uint32_t)(F_NANO_B_RAW_STRING<<16)
 #define F_NANO_C_REAL_STRING (uint32_t)(F_NANO_B_REAL_STRING<<16)

 #define F_NANO_COMPARE_EQ (uint32_t)(1<<16) //Equal
 #define F_NANO_COMPARE_LT (uint32_t)(1<<17) // Lesser than
 #define F_NANO_COMPARE_LEQ (F_NANO_COMPARE_LT|F_NANO_COMPARE_EQ) // Less or equal
 #define F_NANO_COMPARE_GT (uint32_t)(1<<18) // Greater
 #define F_NANO_COMPARE_GEQ (F_NANO_COMPARE_GT|F_NANO_COMPARE_EQ) // Greater or equal
 #define DEFAULT_MAX_FEE "0.001"

#endif

int f_cloud_crypto_wallet_nano_create_seed(size_t, char *, char *);

int f_generate_nano_seed(NANO_SEED, uint32_t);

int pk_to_wallet(char *, char *, NANO_PUBLIC_KEY_EXTENDED);

int f_seed_to_nano_wallet(NANO_PRIVATE_KEY, NANO_PUBLIC_KEY, NANO_SEED, uint32_t);

char *f_nano_key_to_str(char *, unsigned char *);

int f_nano_seed_to_bip39(char *, size_t, size_t *, NANO_SEED, char *);

int f_bip39_to_nano_seed(uint8_t *, char *, char *);

int f_parse_nano_seed_and_bip39_to_JSON(char *, size_t, size_t *, void *, int, const char *);

int f_read_seed(uint8_t *, const char *, void *, int, int);

int f_nano_raw_to_string(char *, size_t *, size_t, void *, int);

int f_nano_valid_nano_str_value(const char *);

int valid_nano_wallet(const char *);

int nano_base_32_2_hex(uint8_t *, char *);

int f_nano_transaction_to_JSON(char *, size_t, size_t *, NANO_PRIVATE_KEY_EXTENDED, F_BLOCK_TRANSFER *);

int valid_raw_balance(const char *);

int is_null_hash(uint8_t *);

int is_nano_prefix(const char *, const char *);

F_FILE_INFO_ERR f_get_nano_file_info(F_NANO_WALLET_INFO *);
F_FILE_INFO_ERR f_set_nano_file_info(F_NANO_WALLET_INFO *, int);

f_nano_err f_nano_value_compare_value(void *, void *, uint32_t *);

f_nano_err f_nano_verify_nano_funds(void *, void *, void *, uint32_t);

f_nano_err f_nano_parse_raw_str_to_raw128_t(uint8_t *, const char *);

f_nano_err f_nano_parse_real_str_to_raw128_t(uint8_t *, const char *);

f_nano_err f_nano_add_sub(void *, void *, void *, uint32_t);

int f_nano_sign_block(F_BLOCK_TRANSFER *, F_BLOCK_TRANSFER *, NANO_PRIVATE_KEY_EXTENDED);

#ifdef __cplusplus
}
#endif
```

5.5 f_util.h File Reference

```
#include <stdint.h>
#include "mbedtls/sha256.h"
#include "mbedtls/aes.h"
```

Macros

- #define **F_ENTROPY_TYPE_PARANOIC** (uint32_t)1477682819
- #define **F_ENTROPY_TYPE_EXCELENT** (uint32_t)1476885281
- #define **F_ENTROPY_TYPE_GOOD** (uint32_t)1472531015
- #define **F_ENTROPY_TYPE_NOT_ENOUGH** (uint32_t)1471001808
- #define **F_ENTROPY_TYPE_NOT_RECOMENDED** (uint32_t)1470003345
- #define **ENTROPY_BEGIN** f_verify_system_entropy_begin();
- #define **ENTROPY_END** f_verify_system_entropy_finish();
- #define **F_PASS_MUST_HAVE_AT_LEAST_NONE** (int)0
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER** (int)1
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL** (int)2
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE** (int)4
- #define **F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE** (int)8
- #define **F_PASS_IS_TOO_LONG** (int)256
- #define **F_PASS_IS_TOO_SHORT** (int)512
- #define **F_PASS_IS_OUT_OVF** (int)1024

Typedefs

- typedef void(∗ **rnd_fn**) (void ∗, size_t)

Functions

- int **f_verify_system_entropy** (uint32_t, void ∗, size_t, int)
- int **f_pass_must_have_at_least** (char ∗, size_t, size_t, size_t, int)
- int **f_passwd_comp_safe** (char ∗, char ∗, size_t, size_t, size_t)
- char ∗ **f_get_entropy_name** (uint32_t)
- uint32_t **f_sel_to_entropy_level** (int)
- int **f_str_to_hex** (uint8_t ∗, char ∗)
- void **f_random_attach** (**rnd_fn**)
- void **f_random** (void ∗, size_t)
- int **get_console_passwd** (char ∗, size_t)

5.5.1 Detailed Description

This ABI is a utility for myNanoEmbedded library and sub routines are implemented here.

Definition in file **f_util.h**.

5.5.2 Macro Definition Documentation

5.5.2.1 ENTROPY_BEGIN

```
#define ENTROPY_BEGIN f_verify_system_entropy_begin();
```

Begins and prepares a entropy function.

See also

f_verify_system_entropy() (p. **??**)

Definition at line **151** of file **f_util.h**.

5.5.2.2 ENTROPY_END

```
#define ENTROPY_END f_verify_system_entropy_finish();
```

Ends a entropy function.

See also

f_verify_system_entropy() (p. **??**)

Definition at line **158** of file **f_util.h**.

5.5.2.3 F_ENTROPY_TYPE_EXCELENT

```
#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
```

Type of the excelent entropy used for verifier.

Slow

Definition at line **123** of file **f_util.h**.

5.5.2.4 F_ENTROPY_TYPE_GOOD

```
#define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
```

Type of the good entropy used for verifier.

Not so slow

Definition at line **130** of file **f_util.h**.

5.5.2.5 F_ENTROPY_TYPE_NOT_ENOUGH

```
#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
```

Type of the moderate entropy used for verifier.

Fast

Definition at line **137** of file **f_util.h**.

5.5.2.6 F_ENTROPY_TYPE_NOT_RECOMENDED

```
#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
```

Type of the not recommended entropy used for verifier.

Very fast

Definition at line **144** of file **f_util.h**.

5.5.2.7 F_ENTROPY_TYPE_PARANOIC

```
#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
```

Type of the very excelent entropy used for verifier.

Very slow

Definition at line **116** of file **f_util.h**.

5.5.2.8 F_PASS_IS_OUT_OVF

```
#define F_PASS_IS_OUT_OVF (int)1024
```

Password is overflow and cannot be stored.

Definition at line **206** of file **f_util.h**.

5.5.2.9 F_PASS_IS_TOO_LONG

```
#define F_PASS_IS_TOO_LONG (int)256
```

Password is too long.

Definition at line **194** of file **f_util.h**.

5.5.2.10 F_PASS_IS_TOO_SHORT

```
#define F_PASS_IS_TOO_SHORT (int)512
```

Password is too short.

Definition at line **200** of file **f_util.h**.

5.5.2.11 F_PASS_MUST_HAVE_AT_LEAST_NONE

```
#define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
```

Password does not need any criteria to pass.

Definition at line **164** of file **f_util.h**.

5.5.2.12 F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8
```

Password must have at least one lower case.

Definition at line **188** of file **f_util.h**.

5.5.2.13 F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
```

Password must have at least one number.

Definition at line **170** of file **f_util.h**.

5.5.2.14 F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
```

Password must have at least one symbol.

Definition at line **176** of file **f_util.h**.

5.5.2.15 F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE

```
#define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
```

Password must have at least one upper case.

Definition at line **182** of file **f_util.h**.

5.5.3 Typedef Documentation

5.5.3.1 rnd_fn

```
rnd_fn
```

Pointer caller for random function.

Definition at line **292** of file **f_util.h**.

5.5.4 Function Documentation

5.5.4.1 f_get_entropy_name()

```
char * f_get_entropy_name (
             uint32_t val )
```

Returns a entropy name given a index/ASCII index or entropy value.

Parameters

in	*val*	Index/ASCII index or entropy value

Return values:

- *NULL* If no entropy index/ASCII/entropy found in *val*
- *F_ENTROPY_TYPE_∗* name if found in index/ASCII or entropy value

5.5.4.2 f_pass_must_have_at_least()

```
int f_pass_must_have_at_least (
             char * password,
             size_t n,
             size_t min,
             size_t max,
             int must_have )
```

Checks if a given password has enought requirements to be parsed to a function.

Parameters

in	*password*	Password string
in	*n*	Max buffer string permitted to store password including NULL char
in	*min*	Minimum size allowed in password string
in	*max*	Maximum size allowed in password
in	*must_have*	Must have a type: • F_PASS_MUST_HAVE_AT_LEAST_NONE Not need any special characters or number • F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER Must have at least one number • F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL Must have at least one symbol • F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE Must have at least one upper case • F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE Must have at least one lower case

Return values:

- *0 (zero):* If password is passed in the test
- *F_PASS_IS_OUT_OVF:* If password lenght exceeds *n* value

- *F_PASS_IS_TOO_SHORT:* If password length is less than *min* value
- *F_PASS_IS_TOO_LONG:* If password length is greater tham *m* value
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE:* If password is required in *must_have* type upper case characters
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE:* If password is required in *must_have* type lower case characters
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL:* If password is required in *must_have* type to have symbol(s)
- *F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER:* if password is required in *must_have* type to have number(s)

5.5.4.3 f_passwd_comp_safe()

```
int f_passwd_comp_safe (
             char * pass1,
             char * pass2,
             size_t n,
             size_t min,
             size_t max )
```

Compares two passwords values with safe buffer.

Parameters

in	*pass1*	First password to compare with *pass2*
in	*pass2*	Second password to compare with *pass1*
in	*n*	Size of Maximum buffer of both *pass1* and *pass2*
in	*min*	Minimun value of both *pass1* and *pass2*
in	*max*	Maximum value of both *pass1* and *pass2*

Return values

0	If *pass1* is equal to *pass2*, otherwise value is less than 0 (zero) if password does not match

5.5.4.4 f_random()

```
void f_random (
             void * random,
             size_t random_sz )
```

Random function to be called to generate a *random* data with *random_sz*

Parameters

out	*random*	Random data to be parsed
in	*random_sz*	Size of random data to be filled

See also

f_random_attach() (p. **??**)

5.5.4.5 f_random_attach()

```
void f_random_attach (
              rnd_fn fn )
```

Attachs a function to be called by ***f_random()*** *(*p. **??***)*

Parameters

in	*fn*	A function to be called

See also

rnd_fn (p. **??**)

5.5.4.6 f_sel_to_entropy_level()

```
uint32_t f_sel_to_entropy_level (
             int sel )
```

Return a given entropy number given a number encoded ASCII or index number.

Parameters

in	*sel*	ASCII or index value

Return values:

- *0 (zero):* If no entropy number found in *sel*
- *F_ENTROPY_TYPE_PARANOIC*
- *F_ENTROPY_TYPE_EXCELENT*
- *F_ENTROPY_TYPE_GOOD*

- *F_ENTROPY_TYPE_NOT_ENOUGH*
- *F_ENTROPY_TYPE_NOT_RECOMENDED*

5.5.4.7 f_str_to_hex()

```
int f_str_to_hex (
             uint8_t * hex_stream,
             char * str )
```

Converts a *str* string buffer to raw *hex_stream* value stream.

Parameters

out	*hex*	Raw hex value
in	*str*	String buffer terminated with NULL char

Return values

0	On Success, otherwise Error

5.5.4.8 f_verify_system_entropy()

```
int f_verify_system_entropy (
             uint32_t type,
             void * rand,
             size_t rand_sz,
             int turn_on_wdt )
```

Take a random number generator function and returns random value only if randomized data have a desired entropy value.

Parameters

in	*type*	Entropy type. Entropy type values are: • F_ENTROPY_TYPE_PARANOIC Highest level entropy recommended for generate a Nano SEED with a paranoic entropy. Very slow • F_ENTROPY_TYPE_EXCELENT Gives a very excellent entropy for generating Nano SEED. Slow • F_ENTROPY_TYPE_GOOD Good entropy type for generating Nano SEED. Normal. • F_ENTROPY_TYPE_NOT_ENOUGH Moderate entropy for generating Nano SEED. Usually fast to create a temporary Nano SEED. Fast • F_ENTROPY_TYPE_NOT_RECOMENDED Fast but not recommended for generating Nano SEED.
out	*rand*	Random data with a satisfied type of entropy
in	*rand_sz*	Size of random data output
in	*turn_on_wdt*	For ESP32, Arduino platform and other microcontrollers only. Turns on/off WATCH DOG (0: OFF, NON ZERO: ON). For Raspberry PI and Linux native is ommited.

Return values

0	On Success, otherwise Error

5.5.4.9 get_console_passwd()

```
int get_console_passwd (
            char * pass,
            size_t pass_sz )
```

Reads a password from console.

Parameters

out	*pass*	Password to be parsed to pointer
in	*pass_sz*	Size of buffer *pass*

Return values

0	On Success, otherwise Error

5.6 f_util.h

```
00001 /*
```

```
    AUTHOR: Fábio Pereira da Silva
    YEAR: 2019-20
    LICENSE: MIT
    EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
*/

#include <stdint.h>
#include "mbedtls/sha256.h"
#include "mbedtls/aes.h"

#ifdef __cplusplus
extern "C" {
#endif

#ifndef F_DOC_SKIP

 #define F_LOG_MAX 8*256
 #define LICENSE \
"MIT License\n\n\
Copyright (c) 2019 Fábio Pereira da Silva\n\n\
Permission is hereby granted, free of charge, to any person obtaining a copy\n\
of this software and associated documentation files (the \"Software\"), to deal\n\
in the Software without restriction, including without limitation the rights\n\
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell\n\
copies of the Software, and to permit persons to whom the Software is\n\
furnished to do so, subject to the following conditions:\n\n\
The above copyright notice and this permission notice shall be included in all\n\
copies or substantial portions of the Software.\n\n\
THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR\n\
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE\n\
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER\n\
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,\n\
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\
SOFTWARE.\n\n\n"

#endif

#ifdef F_ESP32

 #define F_WDT_MAX_ENTROPY_TIME 2*120
 #define F_WDT_PANIC true
 #define F_WDT_MIN_TIME 20//4

#endif

int f_verify_system_entropy(uint32_t, void *, size_t, int);

int f_pass_must_have_at_least(char *, size_t, size_t, size_t, int);

#ifndef F_DOC_SKIP

int f_verify_system_entropy_begin();
void f_verify_system_entropy_finish();
int f_file_exists(char *);
int f_find_str(size_t *, char *, size_t, char *);
int f_find_replace(char *, size_t *, size_t, char *, size_t, char *, char *);
int f_is_integer(char *, size_t);
int is_filled_with_value(uint8_t *, size_t, uint8_t);

#endif

//#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1476682819
#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819

//#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1475885281
#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281

//#define F_ENTROPY_TYPE_GOOD (uint32_t)1471531015
#define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015

//#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1470001808
#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808

//#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1469703345
#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345

#define ENTROPY_BEGIN f_verify_system_entropy_begin();

#define ENTROPY_END f_verify_system_entropy_finish();

#define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0

#define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1

#define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2

#define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4

#define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8

#define F_PASS_IS_TOO_LONG (int)256

#define F_PASS_IS_TOO_SHORT (int)512

#define F_PASS_IS_OUT_OVF (int)1024//768

#ifndef F_DOC_SKIP

 #define F_PBKDF2_ITER_SZ 2*4096

typedef enum f_pbkdf2_err_t {
    F_PBKDF2_RESULT_OK=0,
    F_PBKDF2_ERR_CTX=95,
    F_PBKDF2_ERR_PKCS5,
    F_PBKDF2_ERR_INFO_SHA
} f_pbkdf2_err;

typedef enum f_aes_err {
    F_AES_RESULT_OK=0,
    F_AES_ERR_ENCKEY=30,
    F_AES_ERR_DECKEY,
    F_AES_ERR_MALLOC,
    F_AES_UNKNOW_DIRECTION,
    F_ERR_ENC_DECRYPT_FAILED
} f_aes_err;

char *fhex2strv2(char *, const void *, size_t, int);
uint8_t *f_sha256_digest(uint8_t *, size_t);
f_pbkdf2_err f_pbkdf2_hmac(unsigned char *, size_t, unsigned char *, size_t, uint8_t *);
f_aes_err f_aes256cipher(uint8_t *, uint8_t *, void *, size_t, void *, int);

#endif

int f_passwd_comp_safe(char *, char *, size_t, size_t, size_t);

char *f_get_entropy_name(uint32_t);

uint32_t f_sel_to_entropy_level(int);

int f_str_to_hex(uint8_t *, char *);

#ifndef F_ESP32

typedef void (*rnd_fn)(void *, size_t);

void f_random_attach(rnd_fn);

void f_random(void *, size_t);

int get_console_passwd(char *, size_t);

#endif

#ifdef __cplusplus
}
#endif
```

5.7 sodium.h File Reference

```
#include "sodium/version.h"
#include "sodium/core.h"
#include "sodium/crypto_aead_aes256gcm.h"
#include "sodium/crypto_aead_chacha20poly1305.h"
#include "sodium/crypto_aead_xchacha20poly1305.h"
#include "sodium/crypto_auth.h"
#include "sodium/crypto_auth_hmacsha256.h"
#include "sodium/crypto_auth_hmacsha512.h"
#include "sodium/crypto_auth_hmacsha512256.h"
#include "sodium/crypto_box.h"
#include "sodium/crypto_box_curve25519xsalsa20poly1305.h"
#include "sodium/crypto_core_hsalsa20.h"
```

```
#include "sodium/crypto_core_hchacha20.h"
#include "sodium/crypto_core_salsa20.h"
#include "sodium/crypto_core_salsa2012.h"
#include "sodium/crypto_core_salsa208.h"
#include "sodium/crypto_generichash.h"
#include "sodium/crypto_generichash_blake2b.h"
#include "sodium/crypto_hash.h"
#include "sodium/crypto_hash_sha256.h"
#include "sodium/crypto_hash_sha512.h"
#include "sodium/crypto_kdf.h"
#include "sodium/crypto_kdf_blake2b.h"
#include "sodium/crypto_kx.h"
#include "sodium/crypto_onetimeauth.h"
#include "sodium/crypto_onetimeauth_poly1305.h"
#include "sodium/crypto_pwhash.h"
#include "sodium/crypto_pwhash_argon2i.h"
#include "sodium/crypto_scalarmult.h"
#include "sodium/crypto_scalarmult_curve25519.h"
#include "sodium/crypto_secretbox.h"
#include "sodium/crypto_secretbox_xsalsa20poly1305.h"
#include "sodium/crypto_secretstream_xchacha20poly1305.h"
#include "sodium/crypto_shorthash.h"
#include "sodium/crypto_shorthash_siphash24.h"
#include "sodium/crypto_sign.h"
#include "sodium/crypto_sign_ed25519.h"
#include "sodium/crypto_stream.h"
#include "sodium/crypto_stream_chacha20.h"
#include "sodium/crypto_stream_salsa20.h"
#include "sodium/crypto_stream_xsalsa20.h"
#include "sodium/crypto_verify_16.h"
#include "sodium/crypto_verify_32.h"
#include "sodium/crypto_verify_64.h"
#include "sodium/randombytes.h"
#include "sodium/randombytes_salsa20_random.h"
#include "sodium/randombytes_sysrandom.h"
#include "sodium/runtime.h"
#include "sodium/utils.h"
#include "sodium/crypto_box_curve25519xchacha20poly1305.h"
#include "sodium/crypto_core_ed25519.h"
#include "sodium/crypto_scalarmult_ed25519.h"
#include "sodium/crypto_secretbox_xchacha20poly1305.h"
#include "sodium/crypto_pwhash_scryptsalsa208sha256.h"
#include "sodium/crypto_stream_salsa2012.h"
#include "sodium/crypto_stream_salsa208.h"
#include "sodium/crypto_stream_xchacha20.h"
```

5.7.1 Detailed Description

This header file is an implementation of Libsodium library.

Definition in file **sodium.h**.

5.8 sodium.h

```
00001
```

```
#ifndef sodium_H
#define sodium_H

#include "sodium/version.h"

#include "sodium/core.h"
#include "sodium/crypto_aead_aes256gcm.h"
#include "sodium/crypto_aead_chacha20poly1305.h"
#include "sodium/crypto_aead_xchacha20poly1305.h"
#include "sodium/crypto_auth.h"
#include "sodium/crypto_auth_hmacsha256.h"
#include "sodium/crypto_auth_hmacsha512.h"
#include "sodium/crypto_auth_hmacsha512256.h"
#include "sodium/crypto_box.h"
#include "sodium/crypto_box_curve25519xsalsa20poly1305.h"
#include "sodium/crypto_core_hsalsa20.h"
#include "sodium/crypto_core_hchacha20.h"
#include "sodium/crypto_core_salsa20.h"
#include "sodium/crypto_core_salsa2012.h"
#include "sodium/crypto_core_salsa208.h"
#include "sodium/crypto_generichash.h"
#include "sodium/crypto_generichash_blake2b.h"
#include "sodium/crypto_hash.h"
#include "sodium/crypto_hash_sha256.h"
#include "sodium/crypto_hash_sha512.h"
#include "sodium/crypto_kdf.h"
#include "sodium/crypto_kdf_blake2b.h"
#include "sodium/crypto_kx.h"
#include "sodium/crypto_onetimeauth.h"
#include "sodium/crypto_onetimeauth_poly1305.h"
#include "sodium/crypto_pwhash.h"
#include "sodium/crypto_pwhash_argon2i.h"
#include "sodium/crypto_scalarmult.h"
#include "sodium/crypto_scalarmult_curve25519.h"
#include "sodium/crypto_secretbox.h"
#include "sodium/crypto_secretbox_xsalsa20poly1305.h"
#include "sodium/crypto_secretstream_xchacha20poly1305.h"
#include "sodium/crypto_shorthash.h"
#include "sodium/crypto_shorthash_siphash24.h"
#include "sodium/crypto_sign.h"
#include "sodium/crypto_sign_ed25519.h"
#include "sodium/crypto_stream.h"
#include "sodium/crypto_stream_chacha20.h"
#include "sodium/crypto_stream_salsa20.h"
#include "sodium/crypto_stream_xsalsa20.h"
#include "sodium/crypto_verify_16.h"
#include "sodium/crypto_verify_32.h"
#include "sodium/crypto_verify_64.h"
#include "sodium/randombytes.h"
#ifdef __native_client__
# include "sodium/randombytes_nativeclient.h"
#endif
#include "sodium/randombytes_salsa20_random.h"
#include "sodium/randombytes_sysrandom.h"
#include "sodium/runtime.h"
#include "sodium/utils.h"

#ifndef SODIUM_LIBRARY_MINIMAL
# include "sodium/crypto_box_curve25519xchacha20poly1305.h"
# include "sodium/crypto_core_ed25519.h"
# include "sodium/crypto_scalarmult_ed25519.h"
# include "sodium/crypto_secretbox_xchacha20poly1305.h"
# include "sodium/crypto_pwhash_scryptsalsa208sha256.h"
# include "sodium/crypto_stream_salsa2012.h"
# include "sodium/crypto_stream_salsa208.h"
# include "sodium/crypto_stream_xchacha20.h"
#endif

#endif
```

Index