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`

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 **235** 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 **239** 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 **247** of file **f_nano_crypto_util.h**.

4.1.2.3 link

```
uint8_t link[32]
```

link or destination account

Definition at line **249** of file **f_nano_crypto_util.h**.

4.1.2.4 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **237** of file **f_nano_crypto_util.h**.

4.1.2.5 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **253** of file **f_nano_crypto_util.h**.

4.1.2.6 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **241** of file **f_nano_crypto_util.h**.

4.1.2.7 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **243** of file **f_nano_crypto_util.h**.

4.1.2.8 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **251** of file **f_nano_crypto_util.h**.

4.1.2.9 work

```
uint64_t work
```

Internal use for this API.

Definition at line **255** 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 **366** 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 **372** 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 **376** 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 **368** 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 **374** 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 **378** of file **f_nano_crypto_util.h**.

4.3.2.6 ver

```
uint32_t ver
```

Version of the file.

Definition at line **370** 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 **338** 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 **346** of file **f_nano_crypto_util.h**.

4.4.2.2 iv

```
uint8_t iv[16]
```

Initial sub vector.

Definition at line **342** of file **f_nano_crypto_util.h**.

4.4.2.3 reserved

```
uint8_t reserved[16]
```

Reserved (not used)

Definition at line **344** 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 **348** 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 **340** 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 **450** 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 **454** 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 **458** of file **f_nano_crypto_util.h**.

4.5.2.3 reserved

```
uint8_t reserved[44]
```

Reserved.

Definition at line **460** 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 **452** 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 **456** 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 **482** 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 **494** of file **f_nano_crypto_util.h**.

4.6.2.2 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **488** 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 **492** 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 **484** 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 **490** of file **f_nano_crypto_util.h**.

4.6.2.6 version

```
uint16_t version
```

Version.

Definition at line **486** 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

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00008 #include <stdint.h>
00009
00019 typedef uint8_t F_ADD_288[36];
00020
00021
00022 #ifndef F_DOC_SKIP
00023
00033  void f_add_bn_288_le(F_ADD_288, F_ADD_288, F_ADD_288, int *, int);
00034  void f_sl_elv_add_le(F_ADD_288, int);
00035
00036 #endif
00037
```

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_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** }
- 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** }
- 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** }

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_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)

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 **404** 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 **125** 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 **167** 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 **185** 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 **173** of file **f_nano_crypto_util.h**.

5.3.2.6 NANO_PREFIX

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

Nano prefix.

Definition at line **137** 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 **131** 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 **398** 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 **179** of file **f_nano_crypto_util.h**.

5.3.2.11 XRB_PREFIX

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

XRB (old Raiblocks) prefix.

Definition at line **143** 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 **197** 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 **207** 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 **213** 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 **219** 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 **225** of file **f_nano_crypto_util.h**.

5.3.3.9 NANO_SEED

`NANO_SEED`

Size of Nano SEED.

Definition at line **191** 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_NOT_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_ENCRYPTED_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 **510** 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 **269** 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 **406** 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_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

5.3.5.13 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. **??**)

5.3.5.14 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

Parameters

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

5.3.5.15 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)

Parameters

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

5.3.5.16 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*

Parameters

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. **??**)

5.3.5.17 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. **??**)

5.3.5.18 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

5.3.5.19 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

5.3.5.20 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

5.3.5.21 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

5.3.5.22 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. **??**)

5.3.5.23 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. **??**)

5.3.5.24 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

5.3.5.25 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 **229** 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 **237** 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 **237** of file **f_nano_crypto_util.h**.

5.3.6.4 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **231** of file **f_nano_crypto_util.h**.

5.3.6.5 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **231** 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 **235** 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 **233** 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 **227** 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 **229** 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 **229** of file **f_nano_crypto_util.h**.

5.3.6.11 link

```
uint8_t link[32]
```

link or destination account

Definition at line **239** 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 **233** 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 **227** 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 **233** of file **f_nano_crypto_util.h**.

5.3.6.15 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **227** of file **f_nano_crypto_util.h**.

5.3.6.16 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **243** of file **f_nano_crypto_util.h**.

5.3.6.17 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **231** of file **f_nano_crypto_util.h**.

5.3.6.18 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **233** of file **f_nano_crypto_util.h**.

5.3.6.19 reserved

```
uint8_t reserved
```

Reserved (not used)

Reserved.

Definition at line **231** 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 **233** 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 **237** of file **f_nano_crypto_util.h**.

5.3.6.22 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **241** 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 **235** 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 **227** of file **f_nano_crypto_util.h**.

5.3.6.25 ver

```
uint32_t ver
```

Version of the file.

Definition at line **229** of file **f_nano_crypto_util.h**.

5.3.6.26 version

```
uint16_t version
```

Version.

Definition at line **229** 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 **227** 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 **231** of file **f_nano_crypto_util.h**.

5.3.6.29 work

```
uint64_t work
```

Internal use for this API.

Definition at line **245** of file **f_nano_crypto_util.h**.

5.4 f_nano_crypto_util.h

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00008 #include <stdint.h>
00009 #include "f_util.h"
00010
00011 #ifndef F_DOC_SKIP
00012
00013  #ifdef F_XTENSA
00014
00015   #ifndef F_ESP32
00016    #define F_ESP32
00017   #endif
00018
00019   #include "esp_system.h"
00020
00021  #endif
00022
00023  #include "sodium/crypto_generichash.h"
00024  #include "sodium/crypto_sign.h"
00025  #include "sodium.h"
00026
00027  #ifdef F_ESP32
00028
00029   #include "sodium/private/curve25519_ref10.h"
00030
00031  #else
00032
00033   #include "sodium/private/ed25519_ref10.h"
00034
00035   #define ge_p3 ge25519_p3
00036   #define sc_reduce sc25519_reduce
00037   #define sc_muladd sc25519_muladd
00038   #define ge_scalarmult_base ge25519_scalarmult_base
00039   #define ge_p3_tobytes ge25519_p3_tobytes
00040
00041  #endif
00042
00043 #endif
00044
00117 #ifdef __cplusplus
00118 extern "C" {
00119 #endif
00120
00125 #define MAX_STR_NANO_CHAR (size_t)70 //5+56+8+1
00126
00131 #define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
00132
00137 #define NANO_PREFIX "nano_"
00138
00143 #define XRB_PREFIX "xrb_"
00144
00145 #ifdef F_ESP32
00146
00151 #define BIP39_DICTIONARY "/spiffs/dictionary.dic"
00152 #else
00153
00154  #ifndef F_DOC_SKIP
00155   #define BIP39_DICTIONARY "/spiffs/dictionary.dic"
00156   #define BIP39_DICTIONARY_SAMPLE "../../dictionary.dic"
00157  #endif
00158
00159 #endif
00160
```

```
00167 #define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
00168
00173 #define NANO_PASSWD_MAX_LEN (size_t)80
00174
00179 #define STR_NANO_SZ (size_t)66// 65+1 Null included
00180
00185 #define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
00186
00191 typedef uint8_t NANO_SEED[crypto_sign_SEEDBYTES];
00192
00197 typedef uint8_t f_uint128_t[16];
00198
00199 #ifndef F_DOC_SKIP
00200  #define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof(NANO_SEED)+1
00201 #endif
00202
00207 typedef uint8_t NANO_PRIVATE_KEY[sizeof(NANO_SEED)];
00208
00213 typedef uint8_t NANO_PRIVATE_KEY_EXTENDED[crypto_sign_ed25519_SECRETKEYBYTES];
00214
00219 typedef uint8_t NANO_PUBLIC_KEY[crypto_sign_ed25519_PUBLICKEYBYTES];
00220
00225 typedef uint8_t NANO_PUBLIC_KEY_EXTENDED[PUB_KEY_EXTENDED_MAX_LEN];
00226
00235 typedef struct f_block_transfer_t {
00237    uint8_t preamble[32];
00239    uint8_t account[32];
00241    uint8_t previous[32];
00243    uint8_t representative[32];
00247    f_uint128_t balance;
00249    uint8_t link[32];
00251    uint8_t signature[64];
00253    uint8_t prefixes;
00255    uint64_t work;
00256 } __attribute__((packed)) F_BLOCK_TRANSFER;
00257
00258 #ifndef F_DOC_SKIP
00259  #define F_BLOCK_TRANSFER_SIGNABLE_SZ
       (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_t)-sizeof(uint8_t))
00260 #endif
00261
00269 typedef enum f_nano_err_t {
00271    NANO_ERR_OK=0,
00273    NANO_ERR_CANT_PARSE_BN_STR=5151,
00275    NANO_ERR_MALLOC,
00277    NANO_ERR_CANT_PARSE_FACTOR,
00279    NANO_ERR_MPI_MULT,
00281    NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER,
00283    NANO_ERR_EMPTY_STR,
00285    NANO_ERR_CANT_PARSE_VALUE,
00287    NANO_ERR_PARSE_MPI_TO_STR,
00289    NANO_ERR_CANT_COMPLETE_NULL_CHAR,
00291    NANO_ERR_CANT_PARSE_TO_MPI,
00293    NANO_ERR_INSUFICIENT_FUNDS,
00295    NANO_ERR_SUB_MPI,
00297    NANO_ERR_ADD_MPI,
00299    NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE,
00301    NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO,
00303    NANO_ERR_NO_SENSE_BALANCE_NEGATIVE,
00305    NANO_ERR_VAL_A_INVALID_MODE,
00307    NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T,
00309    NANO_ERR_VAL_B_INVALID_MODE,
00311    NANO_ERR_CANT_PARSE_RAW_A_TO_MPI,
00313    NANO_ERR_CANT_PARSE_RAW_B_TO_MPI,
00315    NANO_ERR_UNKNOWN_ADD_SUB_MODE,
00317    NANO_ERR_INVALID_RES_OUTPUT
00318 } f_nano_err;
00319
00320 #ifndef F_DOC_SKIP
00321
00322  #define READ_SEED_FROM_STREAM (int)1
00323  #define READ_SEED_FROM_FILE (int)2
00324  #define WRITE_SEED_TO_STREAM (int)4
00325  #define WRITE_SEED_TO_FILE (int)8
00326  #define PARSE_JSON_READ_SEED_GENERIC (int)16
00327  #define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)
00328
00329 #endif
00330
00338 typedef struct f_nano_encrypted_wallet_t {
00340    uint8_t sub_salt[32];
00342    uint8_t iv[16];
00344    uint8_t reserved[16];
00346    uint8_t hash_sk_unencrypted[32];
00348    uint8_t sk_encrypted[32];
00349 } __attribute__ ((packed)) F_ENCRYPTED_BLOCK;
00350
```

```
00351 #ifndef F_DOC_SKIP
00352
00353  static const uint8_t NANO_WALLET_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f',
      'i', 'l', 'e', '_'};
00354  #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 !!!"
00355  #define F_DESC_SZ (size_t) (160-sizeof(uint32_t))
00356
00357 #endif
00358
00366 typedef struct f_nano_crypto_wallet_t {
00368    uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)];
00370    uint32_t ver;
00372    uint8_t description[F_DESC_SZ];
00374    uint8_t salt[32];
00376    uint8_t iv[16];
00378    F_ENCRYPTED_BLOCK seed_block;
00379 } __attribute__ ((packed)) F_NANO_CRYPTOWALLET;
00380
00381 #ifndef F_DOC_SKIP
00382
00383 _Static_assert((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1");
00384 _Static_assert((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2");
00385
00386 #endif
00387
00392 #define REP_XRB (uint8_t)0x4
00393
00398 #define SENDER_XRB (uint8_t)0x02
00399
00404 #define DEST_XRB (uint8_t)0x01
00405
00406 typedef enum f_write_seed_err_t {
00408    WRITE_ERR_OK=0,
00410    WRITE_ERR_NULL_PASSWORD=7180,
00412    WRITE_ERR_EMPTY_STRING,
00414    WRITE_ERR_MALLOC,
00416    WRITE_ERR_ENCRYPT_PRIV_KEY,
00418    WRITE_ERR_GEN_SUB_PRIV_KEY,
00420    WRITE_ERR_GEN_MAIN_PRIV_KEY,
00422    WRITE_ERR_ENCRYPT_SUB_BLOCK,
00424    WRITE_ERR_UNKNOWN_OPTION,
00426    WRITE_ERR_FILE_ALREDY_EXISTS,
00428    WRITE_ERR_CREATING_FILE,
00430    WRITE_ERR_WRITING_FILE
00431 } f_write_seed_err;
00432
00433 #ifndef F_DOC_SKIP
00434
00435  #define F_RAW_TO_STR_UINT128 (int)1
00436  #define F_RAW_TO_STR_STRING (int)2
00437  #define F_RAW_STR_MAX_SZ (size_t)41 // 39 + '\0' + '.' -> 39 = log10(2^128)
00438  #define F_MAX_STR_RAW_BALANCE_MAX (size_t)40 //39+'\0'
00439  #define F_NANO_EMPTY_BALANCE "0.0"
00440
00441 #endif
00442
00450 typedef struct f_nano_wallet_info_bdy_t {
00452    uint8_t wallet_prefix; // 0 for NANO; 1 for XRB
00454    uint32_t last_used_wallet_number;
00456    char wallet_representative[MAX_STR_NANO_CHAR];
00458    char max_fee[F_RAW_STR_MAX_SZ];
00460    uint8_t reserved[44];
00461 } __attribute__((packed)) F_NANO_WALLET_INFO_BODY;
00462
00463 #ifndef F_DOC_SKIP
00464
00465  _Static_assert((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte
       aligned");
00466
00467  #define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
00468  #define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
00469  static const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
      '_', 'n', 'f', 'o', '_'};
00470
00471  #define F_NANO_DESC_SZ (size_t)78
00472
00473 #endif
00474
00482 typedef struct f_nano_wallet_info_t {
00484    uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)];
00486    uint16_t version;
00488    char desc[F_NANO_DESC_SZ];
00490    uint8_t nanoseed_hash[32];
00492    uint8_t file_info_integrity[32];
00494    F_NANO_WALLET_INFO_BODY body;
00495 } __attribute__((packed)) F_NANO_WALLET_INFO;
```

```
00496
00497 #ifndef F_DOC_SKIP
00498
00499  _Static_assert((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned");
00500
00501 #endif
00502
00510 typedef enum f_file_info_err_t {
00512    F_FILE_INFO_ERR_OK=0,
00514    F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE=7001,
00516    F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND,
00518    F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE,
00520    F_FILE_INFO_ERR_MALLOC,
00522    F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE,
00524    F_FILE_INFO_ERR_CANT_READ_INFO_FILE,
00526    F_FILE_INFO_INVALID_HEADER_FILE,
00528    F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE,
00530    F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL,
00532    F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE,
00534    F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE,
00536    F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO,
00538    F_FILE_INFO_ERR_EXISTING_FILE,
00540    F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO
00541 } F_FILE_INFO_ERR;
00542
00543 #ifndef F_DOC_SKIP
00544
00545  #define F_NANO_ADD_A_B (uint32_t)(1<<0)
00546  #define F_NANO_SUB_A_B (uint32_t)(1<<1)
00547  #define F_NANO_A_RAW_128 (uint32_t)(1<<2)
00548  #define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
00549  #define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
00550  #define F_NANO_B_RAW_128 (uint32_t)(1<<5)
00551  #define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
00552  #define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
00553  #define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
00554  #define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
00555  #define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
00556  #define F_NANO_C_RAW_128 (uint32_t)(F_NANO_B_RAW_128<<16)
00557  #define F_NANO_C_RAW_STRING (uint32_t)(F_NANO_B_RAW_STRING<<16)
00558  #define F_NANO_C_REAL_STRING (uint32_t)(F_NANO_B_REAL_STRING<<16)
00559
00560  #define F_NANO_COMPARE_EQ (uint32_t)(1<<16) //Equal
00561  #define F_NANO_COMPARE_LT (uint32_t)(1<<17) // Lesser than
00562  #define F_NANO_COMPARE_LEQ (F_NANO_COMPARE_LT|F_NANO_COMPARE_EQ) // Less or equal
00563  #define F_NANO_COMPARE_GT (uint32_t)(1<<18) // Greater
00564  #define F_NANO_COMPARE_GEQ (F_NANO_COMPARE_GT|F_NANO_COMPARE_EQ) // Greater or equal
00565  #define DEFAULT_MAX_FEE "0.001"
00566
00567 #endif
00568
00591 int f_cloud_crypto_wallet_nano_create_seed(size_t, char *, char *);
00592
00605 int f_generate_nano_seed(NANO_SEED, uint32_t);
00606
00621 int pk_to_wallet(char *, char *, NANO_PUBLIC_KEY_EXTENDED);
00622
00640 int f_seed_to_nano_wallet(NANO_PRIVATE_KEY, NANO_PUBLIC_KEY, NANO_SEED, uint32_t);
00641
00651 char *f_nano_key_to_str(char *, unsigned char *);
00652
00671 int f_nano_seed_to_bip39(char *, size_t, size_t *, NANO_SEED, char *);
00672
00687 int f_bip39_to_nano_seed(uint8_t *, char *, char *);
00688
00710 int f_parse_nano_seed_and_bip39_to_JSON(char *, size_t, size_t *, void *, int, const char *);
00711
00729 int f_read_seed(uint8_t *, const char *, void *, int, int);
00730
00745 int f_nano_raw_to_string(char *, size_t *, size_t, void *, int);
00746
00755 int f_nano_valid_nano_str_value(const char *);
00756
00764 int valid_nano_wallet(const char *);
00765
00775 int nano_base_32_2_hex(uint8_t *, char *);
00776
00791 int f_nano_transaction_to_JSON(char *, size_t, size_t *, NANO_PRIVATE_KEY_EXTENDED, F_BLOCK_TRANSFER *);
00792
00800 int valid_raw_balance(const char *);
00801
00809 int is_null_hash(uint8_t *);
00810
00822 int is_nano_prefix(const char *, const char *);
00823
00832 F_FILE_INFO_ERR f_get_nano_file_info(F_NANO_WALLET_INFO *);
00833
```

```
00843 F_FILE_INFO_ERR f_set_nano_file_info(F_NANO_WALLET_INFO *, int);
00844
00868 f_nano_err f_nano_value_compare_value(void *, void *, uint32_t *);
00869
00890 f_nano_err f_nano_verify_nano_funds(void *, void *, void *, uint32_t);
00891
00901 f_nano_err f_nano_parse_raw_str_to_raw128_t(uint8_t *, const char *);
00902
00912 f_nano_err f_nano_parse_real_str_to_raw128_t(uint8_t *, const char *);
00913
00933 f_nano_err f_nano_add_sub(void *, void *, void *, uint32_t);
00934
00935 #ifdef __cplusplus
00936 }
00937 #endif
00938
```

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_IS_TOO_LONG** (int)256
- #define **F_PASS_IS_TOO_SHORT** (int)512
- #define **F_PASS_IS_OUT_OVF** (int)768

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)
- void **f_random_attach** (**rnd_fn**)
- void **f_random** (void ∗, 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 **131** 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 **138** 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 **103** 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 **110** 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 **117** 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 **124** 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 **96** of file **f_util.h**.

5.5.2.8 F_PASS_IS_OUT_OVF

```
#define F_PASS_IS_OUT_OVF (int)768
```

Password is overflow and cannot be stored.

Definition at line **180** 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 **168** 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 **174** 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 **144** of file **f_util.h**.

5.5.2.12 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 **150** of file **f_util.h**.

5.5.2.13 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 **156** of file **f_util.h**.

5.5.2.14 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 **162** 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 **215** of file **f_util.h**.

5.5.4 Function Documentation

Parameters

5.5.4.1 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

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_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)

5.5.4.2 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. **??**)

5.5.4.3 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. **??**)

5.5.4.4 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

5.6 f_util.h

```
00001 /*
00002     AUTHOR: Fábio Pereira da Silva
00003     YEAR: 2019-20
00004     LICENSE: MIT
00005     EMAIL: fabioegel@gmail.com or fabioegel@protonmail.com
00006 */
00007
00013 #include <stdint.h>
00014 #include "mbedtls/sha256.h"
00015 #include "mbedtls/aes.h"
00016
00017 #ifdef __cplusplus
00018 extern "C" {
00019 #endif
00020
00021 #ifndef F_DOC_SKIP
00022
00023  #define F_LOG_MAX 8*256
00024
00025 #endif
00026
00027 #ifdef F_ESP32
00028
00029  #define F_WDT_MAX_ENTROPY_TIME 2*120
00030  #define F_WDT_PANIC true
00031  #define F_WDT_MIN_TIME 20//4
00032
00033 #endif
00034
00051 int f_verify_system_entropy(uint32_t, void *, size_t, int);
00052
00077 int f_pass_must_have_at_least(char *, size_t, size_t, size_t, int);
00078
00079 #ifndef F_DOC_SKIP
00080
00081 int f_verify_system_entropy_begin();
00082 void f_verify_system_entropy_finish();
```

```
00083 int f_file_exists(char *);
00084 int f_find_str(size_t *, char *, size_t, char *);
00085 int f_find_replace(char *, size_t *, size_t, char *, size_t, char *, char *);
00086 int f_is_integer(char *, size_t);
00087 int is_filled_with_value(uint8_t *, size_t, uint8_t);
00088
00089 #endif
00090
00091 //#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1476682819
00096 #define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
00097
00098 //#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1475885281
00103 #define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
00104
00105 //#define F_ENTROPY_TYPE_GOOD (uint32_t)1471531015
00110 #define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
00111
00112 //#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1470001808
00117 #define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
00118
00119 //#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1469703345
00124 #define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
00125
00131 #define ENTROPY_BEGIN f_verify_system_entropy_begin();
00132
00138 #define ENTROPY_END f_verify_system_entropy_finish();
00139
00144 #define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
00145
00150 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
00151
00156 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
00157
00162 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
00163
00168 #define F_PASS_IS_TOO_LONG (int)256
00169
00174 #define F_PASS_IS_TOO_SHORT (int)512
00175
00180 #define F_PASS_IS_OUT_OVF (int)768
00181
00182 #ifndef F_DOC_SKIP
00183
00184  #define F_PBKDF2_ITER_SZ 2*4096
00185
00186 typedef enum f_pbkdf2_err_t {
00187     F_PBKDF2_RESULT_OK=0,
00188     F_PBKDF2_ERR_CTX=95,
00189     F_PBKDF2_ERR_PKCS5,
00190     F_PBKDF2_ERR_INFO_SHA
00191 } f_pbkdf2_err;
00192
00193 typedef enum f_aes_err {
00194     F_AES_RESULT_OK=0,
00195     F_AES_ERR_ENCKEY=30,
00196     F_AES_ERR_DECKEY,
00197     F_AES_ERR_MALLOC,
00198     F_AES_UNKNOW_DIRECTION,
00199     F_ERR_ENC_DECRYPT_FAILED
00200 } f_aes_err;
00201
00202 char *fhex2strv2(char *, const void *, size_t, int);
00203 uint8_t *f_sha256_digest(uint8_t *, size_t);
00204 f_pbkdf2_err f_pbkdf2_hmac(unsigned char *, size_t, unsigned char *, size_t, uint8_t *);
00205 f_aes_err f_aes256cipher(uint8_t *, uint8_t *, void *, size_t, void *, int);
00206
00207 #endif
00208
00209 #ifndef F_ESP32
00210
00215 typedef void (*rnd_fn)(void *, size_t);
00216
00224 void f_random_attach(rnd_fn);
00225
00234 void f_random(void *, size_t);
00235
00236 #endif
00237
00238 #ifdef __cplusplus
00239 }
00240 #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
00005 #ifndef sodium_H
00006 #define sodium_H
00007
00008 #include "sodium/version.h"
00009
00010 #include "sodium/core.h"
00011 #include "sodium/crypto_aead_aes256gcm.h"
00012 #include "sodium/crypto_aead_chacha20poly1305.h"
00013 #include "sodium/crypto_aead_xchacha20poly1305.h"
00014 #include "sodium/crypto_auth.h"
00015 #include "sodium/crypto_auth_hmacsha256.h"
00016 #include "sodium/crypto_auth_hmacsha512.h"
00017 #include "sodium/crypto_auth_hmacsha512256.h"
00018 #include "sodium/crypto_box.h"
00019 #include "sodium/crypto_box_curve25519xsalsa20poly1305.h"
00020 #include "sodium/crypto_core_hsalsa20.h"
00021 #include "sodium/crypto_core_hchacha20.h"
00022 #include "sodium/crypto_core_salsa20.h"
00023 #include "sodium/crypto_core_salsa2012.h"
00024 #include "sodium/crypto_core_salsa208.h"
00025 #include "sodium/crypto_generichash.h"
00026 #include "sodium/crypto_generichash_blake2b.h"
00027 #include "sodium/crypto_hash.h"
00028 #include "sodium/crypto_hash_sha256.h"
00029 #include "sodium/crypto_hash_sha512.h"
00030 #include "sodium/crypto_kdf.h"
00031 #include "sodium/crypto_kdf_blake2b.h"
00032 #include "sodium/crypto_kx.h"
00033 #include "sodium/crypto_onetimeauth.h"
00034 #include "sodium/crypto_onetimeauth_poly1305.h"
00035 #include "sodium/crypto_pwhash.h"
00036 #include "sodium/crypto_pwhash_argon2i.h"
00037 #include "sodium/crypto_scalarmult.h"
00038 #include "sodium/crypto_scalarmult_curve25519.h"
00039 #include "sodium/crypto_secretbox.h"
00040 #include "sodium/crypto_secretbox_xsalsa20poly1305.h"
00041 #include "sodium/crypto_secretstream_xchacha20poly1305.h"
00042 #include "sodium/crypto_shorthash.h"
00043 #include "sodium/crypto_shorthash_siphash24.h"
00044 #include "sodium/crypto_sign.h"
00045 #include "sodium/crypto_sign_ed25519.h"
00046 #include "sodium/crypto_stream.h"
00047 #include "sodium/crypto_stream_chacha20.h"
00048 #include "sodium/crypto_stream_salsa20.h"
00049 #include "sodium/crypto_stream_xsalsa20.h"
00050 #include "sodium/crypto_verify_16.h"
00051 #include "sodium/crypto_verify_32.h"
00052 #include "sodium/crypto_verify_64.h"
00053 #include "sodium/randombytes.h"
00054 #ifdef __native_client__
00055 # include "sodium/randombytes_nativeclient.h"
00056 #endif
00057 #include "sodium/randombytes_salsa20_random.h"
00058 #include "sodium/randombytes_sysrandom.h"
00059 #include "sodium/runtime.h"
00060 #include "sodium/utils.h"
00061
00062 #ifndef SODIUM_LIBRARY_MINIMAL
00063 # include "sodium/crypto_box_curve25519xchacha20poly1305.h"
00064 # include "sodium/crypto_core_ed25519.h"
00065 # include "sodium/crypto_scalarmult_ed25519.h"
00066 # include "sodium/crypto_secretbox_xchacha20poly1305.h"
00067 # include "sodium/crypto_pwhash_scryptsalsa208sha256.h"
```

```
00068 # include "sodium/crypto_stream_salsa2012.h"
00069 # include "sodium/crypto_stream_salsa208.h"
00070 # include "sodium/crypto_stream_xchacha20.h"
00071 #endif
00072
00073 #endif
```

Index