Nano cryptocurrency C library with P2PoW/DPoW support for Embedded

1.0.0

Generated by Doxygen 1.8.13

Contents

1 Overview **1**

2 Data Structure Index **3**

2.1 Data Structures 3

3 File Index **5**

3.1 Files 5

4 Data Structure Documentation **7**

4.1 f_bitcoin_serialize_t Struct Reference 7

4.1.1 Detailed Description 7

4.1.2 Field Documentation 7

4.1.2.1 chain_code 7

4.1.2.2 child_number 8

4.1.2.3 chksum 8

4.1.2.4 finger_print 8

4.1.2.5 master_node 8

4.1.2.6 sk_or_pk_data 8

4.1.2.7 version_bytes 8

4.2 f_block_transfer_t Struct Reference 9

4.2.1 Detailed Description 9

4.2.2 Field Documentation 9

4.2.2.1 account 9

4.2.2.2 balance 9

4.2.2.3 link 10

4.2.2.4 preamble . . . 10
4.2.2.5 prefixes . . . 10
4.2.2.6 previous . . . 10
4.2.2.7 representative . . . 10
4.2.2.8 signature . . . 11
4.2.2.9 work . . . 11
4.3 f_file_info_err_t Struct Reference . . . 11
4.3.1 Detailed Description . . . 11
4.4 f_nano_crypto_wallet_t Struct Reference . . . 11
4.4.1 Detailed Description . . . 12
4.4.2 Field Documentation . . . 12
4.4.2.1 description . . . 12
4.4.2.2 iv . . . 12
4.4.2.3 nano_hdr . . . 12
4.4.2.4 salt . . . 12
4.4.2.5 seed_block . . . 13
4.4.2.6 ver . . . 13
4.5 f_nano_encrypted_wallet_t Struct Reference . . . 13
4.5.1 Detailed Description . . . 13
4.5.2 Field Documentation . . . 13
4.5.2.1 hash_sk_unencrypted . . . 14
4.5.2.2 iv . . . 14
4.5.2.3 reserved . . . 14
4.5.2.4 sk_encrypted . . . 14
4.5.2.5 sub_salt . . . 14
4.6 f_nano_wallet_info_bdy_t Struct Reference . . . 15
4.6.1 Detailed Description . . . 15
4.6.2 Field Documentation . . . 15
4.6.2.1 last_used_wallet_number . . . 15
4.6.2.2 max_fee . . . 15
4.6.2.3 reserved . . . 15
4.6.2.4 wallet_prefix . . . 16
4.6.2.5 wallet_representative . . . 16
4.7 f_nano_wallet_info_t Struct Reference . . . 16
4.7.1 Detailed Description . . . 16
4.7.2 Field Documentation . . . 16
4.7.2.1 body . . . 17
4.7.2.2 desc . . . 17
4.7.2.3 file_info_integrity . . . 17
4.7.2.4 header . . . 17
4.7.2.5 nanoseed_hash . . . 17
4.7.2.6 version . . . 17

5 File Documentation **19**

5.1 f_add_bn_288_le.h File Reference 19

5.1.1 Detailed Description 19

5.1.2 Typedef Documentation 19

5.1.2.1 F_ADD_288 19

5.2 f_add_bn_288_le.h 20

5.3 f_bitcoin.h File Reference 20

5.3.1 Macro Definition Documentation 21

5.3.1.1 F_BITCOIN_BUF_SZ 21

5.3.1.2 F_BITCOIN_P2PKH 21

5.3.1.3 F_BITCOIN_SEED_GENERATOR 21

5.3.1.4 F_BITCOIN_T2PKH 21

5.3.1.5 F_BITCOIN_WIF_MAINNET 22

5.3.1.6 F_BITCOIN_WIF_TESTNET 22

5.3.1.7 F_MAX_BASE58_LENGTH 22

5.3.1.8 F_VERSION_BYTES_IDX_LEN 22

5.3.1.9 MAINNET_PRIVATE 22

5.3.1.10 MAINNET_PUBLIC 22

5.3.1.11 TESTNET_PRIVATE 23

5.3.1.12 TESTNET_PUBLIC 23

5.3.2 Function Documentation 23

5.3.2.1 __attribute__() 23

5.3.2.2 f_bip32_to_public_key_or_private_key() 23

5.3.2.3 f_bitcoin_valid_bip32() 23

5.3.2.4 f_decode_b58_util() 24

5.3.2.5 f_encode_b58() 24

5.3.2.6 f_generate_master_key() 24

5.3.2.7 f_private_key_to_wif() 24

5.3.2.8 f_public_key_to_address() 24

5.3.2.9 f_uncompress_elliptic_curve() 25

5.3.2.10 f_wif_to_private_key() . . . 25
5.3.3 Variable Documentation . . . 25
5.3.3.1 chain_code . . . 25
5.3.3.2 child_number . . . 25
5.3.3.3 chksum . . . 25
5.3.3.4 F_VERSION_BYTES . . . 26
5.3.3.5 finger_print . . . 26
5.3.3.6 master_node . . . 26
5.3.3.7 sk_or_pk_data . . . 26
5.3.3.8 version_bytes . . . 26
5.4 f_bitcoin.h . . . 27
5.5 f_nano_crypto_util.h File Reference . . . 27
5.5.1 Detailed Description . . . 31
5.5.2 Macro Definition Documentation . . . 31
5.5.2.1 DEST_XRB . . . 31
5.5.2.2 F_BRAIN_WALLET_BAD . . . 31
5.5.2.3 F_BRAIN_WALLET_GOOD . . . 32
5.5.2.4 F_BRAIN_WALLET_MAYBE_GOOD . . . 32
5.5.2.5 F_BRAIN_WALLET_NICE . . . 32
5.5.2.6 F_BRAIN_WALLET_PERFECT . . . 32
5.5.2.7 F_BRAIN_WALLET_POOR . . . 33
5.5.2.8 F_BRAIN_WALLET_STILL_WEAK . . . 33
5.5.2.9 F_BRAIN_WALLET_VERY_BAD . . . 33
5.5.2.10 F_BRAIN_WALLET_VERY_GOOD . . . 33
5.5.2.11 F_BRAIN_WALLET_VERY_POOR . . . 34
5.5.2.12 F_BRAIN_WALLET_VERY_WEAK . . . 34
5.5.2.13 F_BRAIN_WALLET_WEAK . . . 34
5.5.2.14 F_DEFAULT_THRESHOLD . . . 34
5.5.2.15 F_IS_SIGNATURE_RAW_HEX_STRING . . . 35
5.5.2.16 F_MESSAGE_IS_HASH_STRING . . . 35

5.5.2.17 F_NANO_POW_MAX_THREAD . . . 35
5.5.2.18 F_SIGNATURE_OUTPUT_NANO_PK . . . 35
5.5.2.19 F_SIGNATURE_OUTPUT_RAW_PK . . . 36
5.5.2.20 F_SIGNATURE_OUTPUT_STRING_PK . . . 36
5.5.2.21 F_SIGNATURE_OUTPUT_XRB_PK . . . 36
5.5.2.22 F_SIGNATURE_RAW . . . 36
5.5.2.23 F_SIGNATURE_STRING . . . 37
5.5.2.24 F_VERIFY_SIG_ASCII_HEX . . . 37
5.5.2.25 F_VERIFY_SIG_NANO_WALLET . . . 37
5.5.2.26 F_VERIFY_SIG_RAW_HEX . . . 37
5.5.2.27 MAX_STR_NANO_CHAR . . . 38
5.5.2.28 NANO_ENCRYPTED_SEED_FILE . . . 38
5.5.2.29 NANO_FILE_WALLETS_INFO . . . 38
5.5.2.30 NANO_PASSWD_MAX_LEN . . . 38
5.5.2.31 NANO_PREFIX . . . 38
5.5.2.32 PUB_KEY_EXTENDED_MAX_LEN . . . 39
5.5.2.33 REP_XRB . . . 39
5.5.2.34 SENDER_XRB . . . 39
5.5.2.35 STR_NANO_SZ . . . 39
5.5.2.36 XRB_PREFIX . . . 39
5.5.3 Typedef Documentation . . . 39
5.5.3.1 F_FILE_INFO_ERR . . . 40
5.5.3.2 F_NANO_CREATE_BLOCK_DYN_ERR . . . 40
5.5.3.3 f_nano_err . . . 40
5.5.3.4 F_TOKEN . . . 40
5.5.3.5 f_uint128_t . . . 40
5.5.3.6 f_write_seed_err . . . 40
5.5.3.7 NANO_PRIVATE_KEY . . . 41
5.5.3.8 NANO_PRIVATE_KEY_EXTENDED . . . 41
5.5.3.9 NANO_PUBLIC_KEY . . . 41

5.5.3.10 NANO_PUBLIC_KEY_EXTENDED . . . 41
5.5.3.11 NANO_SEED . . . 41
5.5.4 Enumeration Type Documentation . . . 41
5.5.4.1 f_file_info_err_t . . . 41
5.5.4.2 f_nano_create_block_dyn_err_t . . . 42
5.5.4.3 f_nano_err_t . . . 43
5.5.4.4 f_write_seed_err_t . . . 44
5.5.5 Function Documentation . . . 45
5.5.5.1 __attribute__() . . . 45
5.5.5.2 f_bip39_to_nano_seed() . . . 45
5.5.5.3 f_cloud_crypto_wallet_nano_create_seed() . . . 45
5.5.5.4 f_extract_seed_from_brainwallet() . . . 46
5.5.5.5 f_generate_nano_seed() . . . 47
5.5.5.6 f_generate_token() . . . 48
5.5.5.7 f_get_dictionary_path() . . . 48
5.5.5.8 f_get_nano_file_info() . . . 49
5.5.5.9 f_is_valid_nano_seed_encrypted() . . . 49
5.5.5.10 f_nano_add_sub() . . . 49
5.5.5.11 f_nano_balance_to_str() . . . 50
5.5.5.12 f_nano_block_to_json() . . . 51
5.5.5.13 f_nano_get_block_hash() . . . 51
5.5.5.14 f_nano_get_p2pow_block_hash() . . . 52
5.5.5.15 f_nano_is_valid_block() . . . 52
5.5.5.16 f_nano_key_to_str() . . . 53
5.5.5.17 f_nano_p2pow_to_JSON() . . . 53
5.5.5.18 f_nano_parse_raw_str_to_raw128_t() . . . 53
5.5.5.19 f_nano_parse_real_str_to_raw128_t() . . . 54
5.5.5.20 f_nano_pow() . . . 54
5.5.5.21 f_nano_raw_to_string() . . . 55
5.5.5.22 f_nano_seed_to_bip39() . . . 56

5.5.5.23 f_nano_sign_block() . . . 56
5.5.5.24 f_nano_transaction_to_JSON() . . . 57
5.5.5.25 f_nano_valid_nano_str_value() . . . 57
5.5.5.26 f_nano_value_compare_value() . . . 58
5.5.5.27 f_nano_verify_nano_funds() . . . 59
5.5.5.28 f_parse_nano_seed_and_bip39_to_JSON() . . . 60
5.5.5.29 f_read_seed() . . . 61
5.5.5.30 f_seed_to_nano_wallet() . . . 61
5.5.5.31 f_set_dictionary_path() . . . 62
5.5.5.32 f_set_nano_file_info() . . . 62
5.5.5.33 f_sign_data() . . . 63
5.5.5.34 f_verify_signed_data() . . . 64
5.5.5.35 f_verify_token() . . . 65
5.5.5.36 f_verify_work() . . . 66
5.5.5.37 f_write_seed() . . . 66
5.5.5.38 from_multiplier() . . . 67
5.5.5.39 is_nano_prefix() . . . 68
5.5.5.40 is_null_hash() . . . 68
5.5.5.41 nano_base_32_2_hex() . . . 68
5.5.5.42 nano_create_block_dynamic() . . . 69
5.5.5.43 pk_to_wallet() . . . 69
5.5.5.44 to_multiplier() . . . 70
5.5.5.45 valid_nano_wallet() . . . 70
5.5.5.46 valid_raw_balance() . . . 71
5.5.6 Variable Documentation . . . 71
5.5.6.1 account . . . 71
5.5.6.2 balance . . . 71
5.5.6.3 body . . . 72
5.5.6.4 desc . . . 72
5.5.6.5 description . . . 72

5.5.6.6 file_info_integrity . . . 72
5.5.6.7 hash_sk_unencrypted . . . 72
5.5.6.8 header . . . 73
5.5.6.9 iv . . . 73
5.5.6.10 last_used_wallet_number . . . 73
5.5.6.11 link . . . 73
5.5.6.12 max_fee . . . 73
5.5.6.13 nano_hdr . . . 74
5.5.6.14 nanoseed_hash . . . 74
5.5.6.15 preamble . . . 74
5.5.6.16 prefixes . . . 74
5.5.6.17 previous . . . 74
5.5.6.18 representative . . . 75
5.5.6.19 reserved . . . 75
5.5.6.20 salt . . . 75
5.5.6.21 seed_block . . . 75
5.5.6.22 signature . . . 75
5.5.6.23 sk_encrypted . . . 76
5.5.6.24 sub_salt . . . 76
5.5.6.25 ver . . . 76
5.5.6.26 version . . . 76
5.5.6.27 wallet_prefix . . . 76
5.5.6.28 wallet_representative . . . 77
5.5.6.29 work . . . 77
5.6 f_nano_crypto_util.h . . . 77
5.7 f_util.h File Reference . . . 83
5.7.1 Detailed Description . . . 84
5.7.2 Macro Definition Documentation . . . 84
5.7.2.1 ENTROPY_BEGIN . . . 84
5.7.2.2 ENTROPY_END . . . 85

5.7.2.3 F_ENTROPY_TYPE_EXCELENT . . . 85
5.7.2.4 F_ENTROPY_TYPE_GOOD . . . 85
5.7.2.5 F_ENTROPY_TYPE_NOT_ENOUGH . . . 85
5.7.2.6 F_ENTROPY_TYPE_NOT_RECOMENDED . . . 86
5.7.2.7 F_ENTROPY_TYPE_PARANOIC . . . 86
5.7.2.8 F_GET_CH_MODE_ANY_KEY . . . 86
5.7.2.9 F_GET_CH_MODE_NO_ECHO . . . 86
5.7.2.10 F_PASS_IS_OUT_OVF . . . 87
5.7.2.11 F_PASS_IS_TOO_LONG . . . 87
5.7.2.12 F_PASS_IS_TOO_SHORT . . . 87
5.7.2.13 F_PASS_MUST_HAVE_AT_LEAST_NONE . . . 87
5.7.2.14 F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE . . . 87
5.7.2.15 F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER . . . 88
5.7.2.16 F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL . . . 88
5.7.2.17 F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE . . . 88
5.7.3 Typedef Documentation . . . 88
5.7.3.1 fn_det . . . 88
5.7.3.2 rnd_fn . . . 88
5.7.4 Function Documentation . . . 88
5.7.4.1 crc32_init() . . . 88
5.7.4.2 f_convert_to_double() . . . 89
5.7.4.3 f_convert_to_long_int() . . . 89
5.7.4.4 f_convert_to_long_int0() . . . 90
5.7.4.5 f_convert_to_long_int0x() . . . 90
5.7.4.6 f_convert_to_long_int_std() . . . 91
5.7.4.7 f_convert_to_unsigned_int() . . . 91
5.7.4.8 f_convert_to_unsigned_int0() . . . 92
5.7.4.9 f_convert_to_unsigned_int0x() . . . 92
5.7.4.10 f_convert_to_unsigned_int_std() . . . 93
5.7.4.11 f_ecdsa_public_key_valid() . . . 94

5.7.4.12 f_ecdsa_secret_key_valid() . . . 94
5.7.4.13 f_gen_ecdsa_key_pair() . . . 94
5.7.4.14 f_get_char_no_block() . . . 94
5.7.4.15 f_get_entropy_name() . . . 95
5.7.4.16 f_hmac_sha512() . . . 95
5.7.4.17 f_is_random_attached() . . . 95
5.7.4.18 f_pass_must_have_at_least() . . . 96
5.7.4.19 f_passwd_comp_safe() . . . 97
5.7.4.20 f_random() . . . 97
5.7.4.21 f_random_attach() . . . 98
5.7.4.22 f_random_detach() . . . 98
5.7.4.23 f_reverse() . . . 98
5.7.4.24 f_ripemd160() . . . 98
5.7.4.25 f_sel_to_entropy_level() . . . 98
5.7.4.26 f_str_to_hex() . . . 99
5.7.4.27 f_uncompress_elliptic_curve() . . . 99
5.7.4.28 f_verify_system_entropy() . . . 100
5.7.4.29 get_console_passwd() . . . 100
5.8 f_util.h . . . 101
5.9 sodium.h File Reference . . . 103
5.9.1 Detailed Description . . . 104
5.10 sodium.h . . . 105

Index **107**

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)
- Fully written in C for maximum performance and portability

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] - Colin LeMahieu - *Nano: A Feeless Distributed Cryptocurrency Network* - (2015)

[2] - Z. S. Spakovszky - *7.3 A Statistical Definition of Entropy* - (2005) - NOTE: Entropy function for cryptography is implemented based on `Definition (7.12)` of this amazing topic

[3] - Kaique Anarkrypto - *Delegated Proof of Work* - (2019)

[4] - `docs.nano.org` - *Node RPCs documentation*

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

f_bitcoin_serialize_t 7
f_block_transfer_t
Nano signed block raw data defined in this `reference` 9
f_file_info_err_t
Error enumerator for info file functions 11
f_nano_crypto_wallet_t
struct of the block of encrypted file to store Nano SEED 11
f_nano_encrypted_wallet_t
struct of the block of encrypted file to store Nano SEED 13
f_nano_wallet_info_bdy_t
struct of the body block of the info file 15
f_nano_wallet_info_t
struct of the body block of the info file 16

Chapter 3

File Index

3.1 Files

Here is a list of all files with brief descriptions:

f_add_bn_288_le.h
Low level implementation of Nano Cryptocurrency C library 19
f_bitcoin.h 20
f_nano_crypto_util.h
This API Integrates Nano Cryptocurrency to low computational devices 27
f_util.h
This ABI is a utility for myNanoEmbedded library and sub routines are implemented here . . . 83
sodium.h
This header file is an implementation of Libsodium library 103

Chapter 4

Data Structure Documentation

4.1 f_bitcoin_serialize_t Struct Reference

`#include <f_bitcoin.h>`

Data Fields

- uint8_t **version_bytes** [4]
- uint8_t **master_node**
- uint8_t **finger_print** [4]
- uint8_t **child_number** [4]
- uint8_t **chain_code** [32]
- uint8_t **sk_or_pk_data** [33]
- uint8_t **chksum** [4]

4.1.1 Detailed Description

Definition at line **27** of file **f_bitcoin.h**.

4.1.2 Field Documentation

4.1.2.1 chain_code

`uint8_t chain_code[32]`

Definition at line **32** of file **f_bitcoin.h**.

4.1.2.2 child_number

```
uint8_t child_number[4]
```

Definition at line **31** of file **f_bitcoin.h**.

4.1.2.3 chksum

```
uint8_t chksum[4]
```

Definition at line **34** of file **f_bitcoin.h**.

4.1.2.4 finger_print

```
uint8_t finger_print[4]
```

Definition at line **30** of file **f_bitcoin.h**.

4.1.2.5 master_node

```
uint8_t master_node
```

Definition at line **29** of file **f_bitcoin.h**.

4.1.2.6 sk_or_pk_data

```
uint8_t sk_or_pk_data[33]
```

Definition at line **33** of file **f_bitcoin.h**.

4.1.2.7 version_bytes

```
uint8_t version_bytes[4]
```

Definition at line **28** of file **f_bitcoin.h**.

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

- **f_bitcoin.h**

4.2 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**

4.2.1 Detailed Description

Nano signed block raw data defined in this `reference`

Definition at line **265** of file **f_nano_crypto_util.h**.

4.2.2 Field Documentation

4.2.2.1 account

`uint8_t account[32]`

Account in raw binary data.

Definition at line **269** of file **f_nano_crypto_util.h**.

4.2.2.2 balance

f_uint128_t balance

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **277** of file **f_nano_crypto_util.h**.

4.2.2.3 link

```
uint8_t link[32]
```

link or destination account

Definition at line **279** of file **f_nano_crypto_util.h**.

4.2.2.4 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **267** of file **f_nano_crypto_util.h**.

4.2.2.5 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **283** of file **f_nano_crypto_util.h**.

4.2.2.6 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **271** of file **f_nano_crypto_util.h**.

4.2.2.7 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **273** of file **f_nano_crypto_util.h**.

4.2.2.8 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **281** of file **f_nano_crypto_util.h**.

4.2.2.9 work

```
uint64_t work
```

Internal use for this API.

Definition at line **285** of file **f_nano_crypto_util.h**.

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

- **f_nano_crypto_util.h**

4.3 f_file_info_err_t Struct Reference

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

4.3.1 Detailed Description

Error enumerator for info file functions.

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

- **f_nano_crypto_util.h**

4.4 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**

4.4.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **396** of file **f_nano_crypto_util.h**.

4.4.2 Field Documentation

4.4.2.1 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **402** of file **f_nano_crypto_util.h**.

4.4.2.2 iv

```
uint8_t iv[16]
```

Initial vector of first encryption layer.

Definition at line **406** of file **f_nano_crypto_util.h**.

4.4.2.3 nano_hdr

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

Header of the file.

Definition at line **398** of file **f_nano_crypto_util.h**.

4.4.2.4 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **404** of file **f_nano_crypto_util.h**.

4.4.2.5 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **408** of file **f_nano_crypto_util.h**.

4.4.2.6 ver

```
uint32_t ver
```

Version of the file.

Definition at line **400** 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_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]

4.5.1 Detailed Description

struct of the block of encrypted file to store Nano SEED

Definition at line **368** of file **f_nano_crypto_util.h**.

4.5.2 Field Documentation

4.5.2.1 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **376** of file **f_nano_crypto_util.h**.

4.5.2.2 iv

```
uint8_t iv[16]
```

Initial sub vector.

Definition at line **372** of file **f_nano_crypto_util.h**.

4.5.2.3 reserved

```
uint8_t reserved[16]
```

Reserved (not used)

Definition at line **374** of file **f_nano_crypto_util.h**.

4.5.2.4 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **378** of file **f_nano_crypto_util.h**.

4.5.2.5 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

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**

4.6 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]

4.6.1 Detailed Description

struct of the body block of the info file

Definition at line **480** of file **f_nano_crypto_util.h**.

4.6.2 Field Documentation

4.6.2.1 last_used_wallet_number

`uint32_t last_used_wallet_number`

Last used wallet number.

Definition at line **484** of file **f_nano_crypto_util.h**.

4.6.2.2 max_fee

`char max_fee[F_RAW_STR_MAX_SZ]`

Custom preferred max fee of Proof of Work.

Definition at line **488** of file **f_nano_crypto_util.h**.

4.6.2.3 reserved

`uint8_t reserved[44]`

Reserved.

Definition at line **490** of file **f_nano_crypto_util.h**.

4.6.2.4 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **482** of file **f_nano_crypto_util.h**.

4.6.2.5 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

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**

4.7 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**

4.7.1 Detailed Description

struct of the body block of the info file

Definition at line **512** of file **f_nano_crypto_util.h**.

4.7.2 Field Documentation

4.7.2.1 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **524** of file **f_nano_crypto_util.h**.

4.7.2.2 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **518** of file **f_nano_crypto_util.h**.

4.7.2.3 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **522** of file **f_nano_crypto_util.h**.

4.7.2.4 header

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

Header magic.

Definition at line **514** of file **f_nano_crypto_util.h**.

4.7.2.5 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **520** of file **f_nano_crypto_util.h**.

4.7.2.6 version

```
uint16_t version
```

Version.

Definition at line **516** 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_bitcoin.h File Reference

```
#include <mbedtls/bignum.h>
```

Data Structures

- struct **f_bitcoin_serialize_t**

Macros

- #define **F_BITCOIN_WIF_MAINNET** (uint8_t)0x80
- #define **F_BITCOIN_WIF_TESTNET** (uint8_t)0xEF
- #define **F_BITCOIN_P2PKH** (uint8_t)0x00
- #define **F_BITCOIN_T2PKH** (uint8_t)0x6F
- #define **F_BITCOIN_BUF_SZ** (size_t)512
- #define **F_MAX_BASE58_LENGTH** (size_t)112
- #define **F_BITCOIN_SEED_GENERATOR** "Bitcoin seed"
- #define **MAINNET_PUBLIC** (size_t)0
- #define **MAINNET_PRIVATE** (size_t)1
- #define **TESTNET_PUBLIC** (size_t)2
- #define **TESTNET_PRIVATE** (size_t)3
- #define **F_VERSION_BYTES_IDX_LEN** (size_t)(sizeof(**F_VERSION_BYTES**)/(4∗sizeof(uint8_t)))

Functions

- struct **f_bitcoin_serialize_t** **__attribute__** ((packed)) BITCOIN_SERIALIZE
- int **f_decode_b58_util** (uint8_t ∗, size_t, size_t ∗, const char ∗)
- int **f_encode_b58** (char ∗, size_t, size_t ∗, uint8_t ∗, size_t)
- int **f_private_key_to_wif** (char ∗, size_t, size_t ∗, uint8_t, uint8_t ∗)
- int **f_wif_to_private_key** (uint8_t ∗, unsigned char ∗, const char ∗)
- int **f_generate_master_key** (BITCOIN_SERIALIZE ∗, size_t, uint32_t)
- int **f_bitcoin_valid_bip32** (BITCOIN_SERIALIZE ∗, int ∗, void ∗, int)
- int **f_uncompress_elliptic_curve** (uint8_t ∗, size_t, size_t ∗, mbedtls_ecp_group_id, uint8_t ∗, size_t)
- int **f_bip32_to_public_key_or_private_key** (uint8_t ∗, uint8_t ∗, uint32_t, const char ∗)
- int **f_public_key_to_address** (char ∗, size_t, size_t ∗, uint8_t ∗, uint8_t)

Variables

- static const uint8_t **F_VERSION_BYTES** [ ][4]
- uint8_t **version_bytes** [4]
- uint8_t **master_node**
- uint8_t **finger_print** [4]
- uint8_t **child_number** [4]
- uint8_t **chain_code** [32]
- uint8_t **sk_or_pk_data** [33]
- uint8_t **chksum** [4]

5.3.1 Macro Definition Documentation

5.3.1.1 F_BITCOIN_BUF_SZ

```
#define F_BITCOIN_BUF_SZ (size_t)512
```

Definition at line **10** of file **f_bitcoin.h**.

5.3.1.2 F_BITCOIN_P2PKH

```
#define F_BITCOIN_P2PKH (uint8_t)0x00
```

Definition at line **8** of file **f_bitcoin.h**.

5.3.1.3 F_BITCOIN_SEED_GENERATOR

```
#define F_BITCOIN_SEED_GENERATOR "Bitcoin seed"
```

Definition at line **12** of file **f_bitcoin.h**.

5.3.1.4 F_BITCOIN_T2PKH

```
#define F_BITCOIN_T2PKH (uint8_t)0x6F
```

Definition at line **9** of file **f_bitcoin.h**.

5.3.1.5 F_BITCOIN_WIF_MAINNET

```
#define F_BITCOIN_WIF_MAINNET (uint8_t)0x80
```

Definition at line **6** of file **f_bitcoin.h**.

5.3.1.6 F_BITCOIN_WIF_TESTNET

```
#define F_BITCOIN_WIF_TESTNET (uint8_t)0xEF
```

Definition at line **7** of file **f_bitcoin.h**.

5.3.1.7 F_MAX_BASE58_LENGTH

```
#define F_MAX_BASE58_LENGTH (size_t)112
```

Definition at line **11** of file **f_bitcoin.h**.

5.3.1.8 F_VERSION_BYTES_IDX_LEN

```
#define F_VERSION_BYTES_IDX_LEN (size_t)(sizeof( F_VERSION_BYTES)/(4*sizeof(uint8_t)))
```

Definition at line **25** of file **f_bitcoin.h**.

5.3.1.9 MAINNET_PRIVATE

```
#define MAINNET_PRIVATE (size_t)1
```

Definition at line **15** of file **f_bitcoin.h**.

5.3.1.10 MAINNET_PUBLIC

```
#define MAINNET_PUBLIC (size_t)0
```

Definition at line **14** of file **f_bitcoin.h**.

5.3.1.11 TESTNET_PRIVATE

```
#define TESTNET_PRIVATE (size_t)3
```

Definition at line **17** of file **f_bitcoin.h**.

5.3.1.12 TESTNET_PUBLIC

```
#define TESTNET_PUBLIC (size_t)2
```

Definition at line **16** of file **f_bitcoin.h**.

5.3.2 Function Documentation

5.3.2.1 __attribute__()

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

5.3.2.2 f_bip32_to_public_key_or_private_key()

```
int f_bip32_to_public_key_or_private_key (
             uint8_t * ,
             uint8_t * ,
             uint32_t ,
             const char *  )
```

5.3.2.3 f_bitcoin_valid_bip32()

```
int f_bitcoin_valid_bip32 (
             BITCOIN_SERIALIZE * ,
             int * ,
             void * ,
             int  )
```

5.3.2.4 f_decode_b58_util()

```
int f_decode_b58_util (
            uint8_t * ,
            size_t ,
            size_t * ,
            const char *  )
```

5.3.2.5 f_encode_b58()

```
int f_encode_b58 (
            char * ,
            size_t ,
            size_t * ,
            uint8_t * ,
            size_t  )
```

5.3.2.6 f_generate_master_key()

```
int f_generate_master_key (
            BITCOIN_SERIALIZE * ,
            size_t ,
            uint32_t  )
```

5.3.2.7 f_private_key_to_wif()

```
int f_private_key_to_wif (
            char * ,
            size_t ,
            size_t * ,
            uint8_t ,
            uint8_t *  )
```

5.3.2.8 f_public_key_to_address()

```
int f_public_key_to_address (
            char * ,
            size_t ,
            size_t * ,
            uint8_t * ,
            uint8_t  )
```

5.3.2.9 f_uncompress_elliptic_curve()

```
int f_uncompress_elliptic_curve (
             uint8_t * ,
             size_t ,
             size_t * ,
             mbedtls_ecp_group_id ,
             uint8_t * ,
             size_t  )
```

5.3.2.10 f_wif_to_private_key()

```
int f_wif_to_private_key (
             uint8_t * ,
             unsigned char * ,
             const char *  )
```

5.3.3 Variable Documentation

5.3.3.1 chain_code

```
uint8_t chain_code[32]
```

Definition at line **24** of file **f_bitcoin.h**.

5.3.3.2 child_number

```
uint8_t child_number[4]
```

Definition at line **23** of file **f_bitcoin.h**.

5.3.3.3 chksum

```
uint8_t chksum[4]
```

Definition at line **26** of file **f_bitcoin.h**.

5.3.3.4 F_VERSION_BYTES

```
const uint8_t F_VERSION_BYTES[][4]  [static]
```

Initial value:

```
= {
   {0x04, 0x88, 0xB2, 0x1E},
   {0x04, 0x88, 0xAD, 0xE4},
   {0x04, 0x35, 0x87, 0xCF},
   {0x04, 0x35, 0x83, 0x94}
}
```

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

5.3.3.5 finger_print

```
uint8_t finger_print[4]
```

Definition at line **22** of file **f_bitcoin.h**.

5.3.3.6 master_node

```
uint8_t master_node
```

Definition at line **21** of file **f_bitcoin.h**.

5.3.3.7 sk_or_pk_data

```
uint8_t sk_or_pk_data[33]
```

Definition at line **25** of file **f_bitcoin.h**.

5.3.3.8 version_bytes

```
uint8_t version_bytes[4]
```

Definition at line **20** of file **f_bitcoin.h**.

5.4 f_bitcoin.h

```
00001 //#include <f_util.h>
00002 #include <mbedtls/bignum.h>
00003 //#include <string.h>
00004 //#include <stdlib.h>
00005
00006 #define F_BITCOIN_WIF_MAINNET (uint8_t)0x80
00007 #define F_BITCOIN_WIF_TESTNET (uint8_t)0xEF
00008 #define F_BITCOIN_P2PKH (uint8_t)0x00 // P2PKH address
00009 #define F_BITCOIN_T2PKH (uint8_t)0x6F // Testnet Address
00010 #define F_BITCOIN_BUF_SZ (size_t)512
00011 #define F_MAX_BASE58_LENGTH (size_t)112//52 // including null char
00012 #define F_BITCOIN_SEED_GENERATOR "Bitcoin seed"
00013
00014 #define MAINNET_PUBLIC (size_t)0
00015 #define MAINNET_PRIVATE (size_t)1
00016 #define TESTNET_PUBLIC (size_t)2
00017 #define TESTNET_PRIVATE (size_t)3
00018
00019 static const uint8_t F_VERSION_BYTES[][4] = {
00020    {0x04, 0x88, 0xB2, 0x1E}, //mainnet public
00021    {0x04, 0x88, 0xAD, 0xE4}, //mainnet private
00022    {0x04, 0x35, 0x87, 0xCF}, //testnet public
00023    {0x04, 0x35, 0x83, 0x94} // testnet private
00024 };
00025 #define F_VERSION_BYTES_IDX_LEN (size_t)(sizeof(F_VERSION_BYTES)/(4*sizeof(uint8_t)))
00026
00027 typedef struct f_bitcoin_serialize_t {
00028    uint8_t version_bytes[4];
00029    uint8_t master_node;
00030    uint8_t finger_print[4];
00031    uint8_t child_number[4];
00032    uint8_t chain_code[32];
00033    uint8_t sk_or_pk_data[33];
00034    uint8_t chksum[4];
00035 } __attribute__((packed)) BITCOIN_SERIALIZE;
00036
00037 int f_decode_b58_util(uint8_t *, size_t, size_t *, const char *);
00038 int f_encode_b58(char *, size_t, size_t *, uint8_t *, size_t);
00039 int f_private_key_to_wif(char *, size_t, size_t *, uint8_t, uint8_t *);
00040 int f_wif_to_private_key(uint8_t *, unsigned char *, const char *);
00041 int f_generate_master_key(BITCOIN_SERIALIZE *, size_t, uint32_t);
00042 int f_bitcoin_valid_bip32(BITCOIN_SERIALIZE *, int *, void *, int);
00043 int f_uncompress_elliptic_curve(uint8_t *, size_t, size_t *, mbedtls_ecp_group_id, uint8_t *, size_t);
00044 int f_bip32_to_public_key_or_private_key(uint8_t *, uint8_t *, uint32_t, const char *);
00045 int f_public_key_to_address(char *, size_t, size_t *, uint8_t *, uint8_t);
00046
```

5.5 f_nano_crypto_util.h File Reference

```
#include <stdint.h>
#include <f_util.h>
#include <f_bitcoin.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 **F_NANO_POW_MAX_THREAD** (size_t)10
- #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
- #define **F_BRAIN_WALLET_VERY_POOR** (uint32_t)0
- #define **F_BRAIN_WALLET_POOR** (uint32_t)1
- #define **F_BRAIN_WALLET_VERY_BAD** (uint32_t)2
- #define **F_BRAIN_WALLET_BAD** (uint32_t)3
- #define **F_BRAIN_WALLET_VERY_WEAK** (uint32_t)4
- #define **F_BRAIN_WALLET_WEAK** (uint32_t)5
- #define **F_BRAIN_WALLET_STILL_WEAK** (uint32_t)6
- #define **F_BRAIN_WALLET_MAYBE_GOOD** (uint32_t)7
- #define **F_BRAIN_WALLET_GOOD** (uint32_t)8
- #define **F_BRAIN_WALLET_VERY_GOOD** (uint32_t)9
- #define **F_BRAIN_WALLET_NICE** (uint32_t)10
- #define **F_BRAIN_WALLET_PERFECT** (uint32_t)11
- #define **F_SIGNATURE_RAW** (uint32_t)1
- #define **F_SIGNATURE_STRING** (uint32_t)2
- #define **F_SIGNATURE_OUTPUT_RAW_PK** (uint32_t)4
- #define **F_SIGNATURE_OUTPUT_STRING_PK** (uint32_t)8
- #define **F_SIGNATURE_OUTPUT_XRB_PK** (uint32_t)16
- #define **F_SIGNATURE_OUTPUT_NANO_PK** (uint32_t)32
- #define **F_IS_SIGNATURE_RAW_HEX_STRING** (uint32_t)64
- #define **F_MESSAGE_IS_HASH_STRING** (uint32_t)128
- #define **F_DEFAULT_THRESHOLD** (uint64_t) 0xfffffffc000000000
- #define **F_VERIFY_SIG_NANO_WALLET** (uint32_t)1
- #define **F_VERIFY_SIG_RAW_HEX** (uint32_t)2
- #define **F_VERIFY_SIG_ASCII_HEX** (uint32_t)4

Typedefs

- typedef uint8_t **F_TOKEN**[16]
- 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**
- typedef enum **f_nano_create_block_dyn_err_t** **F_NANO_CREATE_BLOCK_DYN_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** }
- enum **f_nano_create_block_dyn_err_t** {
  NANO_CREATE_BLK_DYN_OK = 0, **NANO_CREATE_BLK_DYN_BLOCK_NULL** = 8000, **NANO_CR↩EATE_BLK_DYN_ACCOUNT_NULL**, **NANO_CREATE_BLK_DYN_PREV_NULL**,
  NANO_CREATE_BLK_DYN_REP_NULL, **NANO_CREATE_BLK_DYN_BALANCE_NULL**, **NANO_CR↩EATE_BLK_DYN_SEND_RECEIVE_NULL**, **NANO_CREATE_BLK_DYN_LINK_NULL**,
  NANO_CREATE_BLK_DYN_BUF_MALLOC, **NANO_CREATE_BLK_DYN_MALLOC**, **NANO_CREAT↩E_BL_DYN_WRONG_PREVIOUS_SZ**, **NANO_CREATE_BL_DYN_WRONG_PREVIOUS_STR_SZ**,
  NANO_CREATE_BL_DYN_PARSE_STR_HEX_ERR }

Functions

- struct **f_block_transfer_t** **__attribute__** ((packed)) F_BLOCK_TRANSFER
- double **to_multiplier** (uint64_t, uint64_t)
- uint64_t **from_multiplier** (double, uint64_t)
- void **f_set_dictionary_path** (const char *)
- char * **f_get_dictionary_path** (void)
- int **f_generate_token** (**F_TOKEN**, void *, size_t, const char *)
- int **f_verify_token** (**F_TOKEN**, void *, size_t, const char *)
- 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)
- int **f_nano_is_valid_block** (F_BLOCK_TRANSFER *)

- int **f_nano_block_to_json** (char *, size_t *, size_t, F_BLOCK_TRANSFER *)
- int **f_nano_get_block_hash** (uint8_t *, F_BLOCK_TRANSFER *)
- int **f_nano_get_p2pow_block_hash** (uint8_t *, uint8_t *, F_BLOCK_TRANSFER *)
- int **f_nano_p2pow_to_JSON** (char *, size_t *, size_t, F_BLOCK_TRANSFER *)
- 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**)
- **f_write_seed_err f_write_seed** (void *, int, uint8_t *, char *)
- **f_nano_err f_nano_balance_to_str** (char *, size_t, size_t *, **f_uint128_t**)
- int **f_extract_seed_from_brainwallet** (uint8_t *, char **, uint32_t, const char *, const char *)
- int **f_verify_work** (uint64_t *, const unsigned char *, uint64_t *, uint64_t)
- int **f_sign_data** (unsigned char * **signature**, void *out_public_key, uint32_t ouput_type, const unsigned char *message, size_t msg_len, const unsigned char *private_key)
- int **f_verify_signed_data** (const unsigned char *, const unsigned char *, size_t, const void *, uint32_t)
- int **f_is_valid_nano_seed_encrypted** (void *, size_t, int)
- int **nano_create_block_dynamic** (F_BLOCK_TRANSFER **, const void *, size_t, const void *, size_t, const void *, size_t, const void *, const void *, uint32_t, const void *, int)
- int **f_nano_pow** (uint64_t *, unsigned char *, const uint64_t, int)

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**

5.5.1 Detailed Description

This API Integrates Nano Cryptocurrency to low computational devices.

Definition in file **f_nano_crypto_util.h**.

5.5.2 Macro Definition Documentation

5.5.2.1 DEST_XRB

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

Definition at line **434** of file **f_nano_crypto_util.h**.

5.5.2.2 F_BRAIN_WALLET_BAD

```
#define F_BRAIN_WALLET_BAD (uint32_t)3
```

[bad].

Crack within one day

Definition at line **1188** of file **f_nano_crypto_util.h**.

5.5.2.3 F_BRAIN_WALLET_GOOD

```
#define F_BRAIN_WALLET_GOOD (uint32_t)8
```

[good].

Crack within one thousand year

Definition at line **1219** of file **f_nano_crypto_util.h**.

5.5.2.4 F_BRAIN_WALLET_MAYBE_GOOD

```
#define F_BRAIN_WALLET_MAYBE_GOOD (uint32_t)7
```

[maybe good for you].

Crack within one century

Definition at line **1212** of file **f_nano_crypto_util.h**.

5.5.2.5 F_BRAIN_WALLET_NICE

```
#define F_BRAIN_WALLET_NICE (uint32_t)10
```

[very nice].

Crack withing one hundred thousand year

Definition at line **1231** of file **f_nano_crypto_util.h**.

5.5.2.6 F_BRAIN_WALLET_PERFECT

```
#define F_BRAIN_WALLET_PERFECT (uint32_t)11
```

[Perfect!] 3.34x10^53 Years to crack

Definition at line **1237** of file **f_nano_crypto_util.h**.

5.5.2.7 F_BRAIN_WALLET_POOR

```
#define F_BRAIN_WALLET_POOR (uint32_t)1
```

[poor].

Crack within minutes

Definition at line **1176** of file **f_nano_crypto_util.h**.

5.5.2.8 F_BRAIN_WALLET_STILL_WEAK

```
#define F_BRAIN_WALLET_STILL_WEAK (uint32_t)6
```

[still weak].

Crack within one year

Definition at line **1206** of file **f_nano_crypto_util.h**.

5.5.2.9 F_BRAIN_WALLET_VERY_BAD

```
#define F_BRAIN_WALLET_VERY_BAD (uint32_t)2
```

[very bad].

Crack within one hour

Definition at line **1182** of file **f_nano_crypto_util.h**.

5.5.2.10 F_BRAIN_WALLET_VERY_GOOD

```
#define F_BRAIN_WALLET_VERY_GOOD (uint32_t)9
```

[very good].

Crack within ten thousand year

Definition at line **1225** of file **f_nano_crypto_util.h**.

5.5.2.11 F_BRAIN_WALLET_VERY_POOR

```
#define F_BRAIN_WALLET_VERY_POOR (uint32_t)0
```

[very poor].

Crack within seconds or less

Definition at line **1170** of file **f_nano_crypto_util.h**.

5.5.2.12 F_BRAIN_WALLET_VERY_WEAK

```
#define F_BRAIN_WALLET_VERY_WEAK (uint32_t)4
```

[very weak].

Crack within one week

Definition at line **1194** of file **f_nano_crypto_util.h**.

5.5.2.13 F_BRAIN_WALLET_WEAK

```
#define F_BRAIN_WALLET_WEAK (uint32_t)5
```

[weak].

Crack within one month

Definition at line **1200** of file **f_nano_crypto_util.h**.

5.5.2.14 F_DEFAULT_THRESHOLD

```
#define F_DEFAULT_THRESHOLD (uint64_t) 0xfffffffc000000000
```

Default Nano Proof of Work Threshold.

Definition at line **1340** of file **f_nano_crypto_util.h**.

5.5.2.15 F_IS_SIGNATURE_RAW_HEX_STRING

```
#define F_IS_SIGNATURE_RAW_HEX_STRING (uint32_t)64
```

Signature is raw hex string flag.

See also

f_sign_data() (p. **??**)

Definition at line **1327** of file **f_nano_crypto_util.h**.

5.5.2.16 F_MESSAGE_IS_HASH_STRING

```
#define F_MESSAGE_IS_HASH_STRING (uint32_t)128
```

Message is raw hex hash string.

See also

f_sign_data() (p. **??**)

Definition at line **1334** of file **f_nano_crypto_util.h**.

5.5.2.17 F_NANO_POW_MAX_THREAD

```
#define F_NANO_POW_MAX_THREAD (size_t)10
```

(desktop only) Number of threads for Proof of Work routines.

Default 10

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

5.5.2.18 F_SIGNATURE_OUTPUT_NANO_PK

```
#define F_SIGNATURE_OUTPUT_NANO_PK (uint32_t)32
```

Public key is a NANO wallet encoded base32 string.

See also

f_sign_data() (p. **??**)

Definition at line **1320** of file **f_nano_crypto_util.h**.

5.5.2.19 F_SIGNATURE_OUTPUT_RAW_PK

```
#define F_SIGNATURE_OUTPUT_RAW_PK (uint32_t)4
```

Public key is raw data.

See also

f_sign_data() (p. **??**)

Definition at line **1299** of file **f_nano_crypto_util.h**.

5.5.2.20 F_SIGNATURE_OUTPUT_STRING_PK

```
#define F_SIGNATURE_OUTPUT_STRING_PK (uint32_t)8
```

Public key is hex ASCII encoded string.

See also

f_sign_data() (p. **??**)

Definition at line **1306** of file **f_nano_crypto_util.h**.

5.5.2.21 F_SIGNATURE_OUTPUT_XRB_PK

```
#define F_SIGNATURE_OUTPUT_XRB_PK (uint32_t)16
```

Public key is a XRB wallet encoded base32 string.

See also

f_sign_data() (p. **??**)

Definition at line **1313** of file **f_nano_crypto_util.h**.

5.5.2.22 F_SIGNATURE_RAW

```
#define F_SIGNATURE_RAW (uint32_t)1
```

Signature is raw data.

See also

f_sign_data() (p. **??**)

Definition at line **1285** of file **f_nano_crypto_util.h**.

5.5.2.23 F_SIGNATURE_STRING

```
#define F_SIGNATURE_STRING (uint32_t)2
```

Signature is hex ASCII encoded string.

See also

f_sign_data() (p. **??**)

Definition at line **1292** of file **f_nano_crypto_util.h**.

5.5.2.24 F_VERIFY_SIG_ASCII_HEX

```
#define F_VERIFY_SIG_ASCII_HEX (uint32_t)4
```

Public key is a hex ASCII encoded string.

See also

f_verify_signed_data() (p. **??**)

Definition at line **1392** of file **f_nano_crypto_util.h**.

5.5.2.25 F_VERIFY_SIG_NANO_WALLET

```
#define F_VERIFY_SIG_NANO_WALLET (uint32_t)1
```

Public key is a NANO wallet with *XRB* or *NANO* prefixes encoded base32 string.

See also

f_verify_signed_data() (p. **??**)

Definition at line **1378** of file **f_nano_crypto_util.h**.

5.5.2.26 F_VERIFY_SIG_RAW_HEX

```
#define F_VERIFY_SIG_RAW_HEX (uint32_t)2
```

Public key raw 32 bytes data.

See also

f_verify_signed_data() (p. **??**)

Definition at line **1385** of file **f_nano_crypto_util.h**.

5.5.2.27 MAX_STR_NANO_CHAR

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

Defines a max size of Nano char (70 bytes)

Definition at line **149** of file **f_nano_crypto_util.h**.

5.5.2.28 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 **191** of file **f_nano_crypto_util.h**.

5.5.2.29 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 **209** of file **f_nano_crypto_util.h**.

5.5.2.30 NANO_PASSWD_MAX_LEN

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

Password max length.

Definition at line **197** of file **f_nano_crypto_util.h**.

5.5.2.31 NANO_PREFIX

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

Nano prefix.

Definition at line **161** of file **f_nano_crypto_util.h**.

5.5.2.32 PUB_KEY_EXTENDED_MAX_LEN

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

Max size of public key (extended)

Definition at line **155** of file **f_nano_crypto_util.h**.

5.5.2.33 REP_XRB

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

Representative XRB flag.

Destination XRB flag.

Sender XRB flag.

5.5.2.34 SENDER_XRB

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

Definition at line **428** of file **f_nano_crypto_util.h**.

5.5.2.35 STR_NANO_SZ

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

String size of Nano encoded Base32 including NULL char.

Definition at line **203** of file **f_nano_crypto_util.h**.

5.5.2.36 XRB_PREFIX

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

XRB (old Raiblocks) prefix.

Definition at line **167** of file **f_nano_crypto_util.h**.

5.5.3 Typedef Documentation

5.5.3.1 F_FILE_INFO_ERR

F_FILE_INFO_ERR

Typedef Error enumerator for info file functions.

5.5.3.2 F_NANO_CREATE_BLOCK_DYN_ERR

typedef enum **f_nano_create_block_dyn_err_t** **F_NANO_CREATE_BLOCK_DYN_ERR**

5.5.3.3 f_nano_err

f_nano_err

Error function enumerator.

See also

f_nano_err_t (p. **??**)

5.5.3.4 F_TOKEN

typedef uint8_t F_TOKEN[16]

Definition at line **215** of file **f_nano_crypto_util.h**.

5.5.3.5 f_uint128_t

f_uint128_t

128 bit big number of Nano balance

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

5.5.3.6 f_write_seed_err

typedef enum **f_write_seed_err_t** **f_write_seed_err**

5.5.3.7 NANO_PRIVATE_KEY

`NANO_PRIVATE_KEY`

Size of Nano Private Key.

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

5.5.3.8 NANO_PRIVATE_KEY_EXTENDED

`NANO_PRIVATE_KEY_EXTENDED`

Size of Nano Private Key extended.

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

5.5.3.9 NANO_PUBLIC_KEY

`NANO_PUBLIC_KEY`

Size of Nano Public Key.

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

5.5.3.10 NANO_PUBLIC_KEY_EXTENDED

`NANO_PUBLIC_KEY_EXTENDED`

Size of Public Key Extended.

Definition at line **255** of file **f_nano_crypto_util.h**.

5.5.3.11 NANO_SEED

`NANO_SEED`

Size of Nano SEED.

Definition at line **221** of file **f_nano_crypto_util.h**.

5.5.4 Enumeration Type Documentation

5.5.4.1 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 **540** of file **f_nano_crypto_util.h**.

5.5.4.2 f_nano_create_block_dyn_err_t

`enum` **`f_nano_create_block_dyn_err_t`**

Enumerator

NANO_CREATE_BLK_DYN_OK	
NANO_CREATE_BLK_DYN_BLOCK_NULL	
NANO_CREATE_BLK_DYN_ACCOUNT_NULL	
NANO_CREATE_BLK_DYN_PREV_NULL	
NANO_CREATE_BLK_DYN_REP_NULL	
NANO_CREATE_BLK_DYN_BALANCE_NULL	
NANO_CREATE_BLK_DYN_SEND_RECEIVE_NULL	
NANO_CREATE_BLK_DYN_LINK_NULL	
NANO_CREATE_BLK_DYN_BUF_MALLOC	
NANO_CREATE_BLK_DYN_MALLOC	
NANO_CREATE_BL_DYN_WRONG_PREVIOUS_SZ	
NANO_CREATE_BL_DYN_WRONG_PREVIOUS_STR_SZ	
NANO_CREATE_BL_DYN_PARSE_STR_HEX_ERR	

Definition at line **600** of file **f_nano_crypto_util.h**.

5.5.4.3 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.
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 **299** of file **f_nano_crypto_util.h**.

5.5.4.4 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 **436** of file **f_nano_crypto_util.h**.

5.5.5 Function Documentation

5.5.5.1 __attribute__()

```
struct  f_block_transfer_t __attribute__ (
             (packed)  )
```

5.5.5.2 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. **??**)

5.5.5.3 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 strength

Return values

0	On Success, otherwise Error

5.5.5.4 f_extract_seed_from_brainwallet()

```
int f_extract_seed_from_brainwallet (
            uint8_t * seed,
            char ** warning_msg,
            uint32_t allow_mode,
            const char * brainwallet,
            const char * salt )
```

Analyzes a text given a *mode* and if pass then the text in *braiwallet* is translated to a Nano SEED.

Parameters

out	*seed*	Output Nano SEED extracted from *brainwallet*
out	*warning_msg*	Warning message parsed to application. It can be NULL

Parameters

in	*allow_mode*	Allow *mode*. Funtion will return SUCCESS only if permitted mode set by user Allow mode are: • *F_BRAIN_WALLET_VERY_POOR* Crack within seconds or less • *F_BRAIN_WALLET_POOR* Crack within minutes • *F_BRAIN_WALLET_VERY_BAD* Crack within one hour • *F_BRAIN_WALLET_BAD* Crack within one day • *F_BRAIN_WALLET_VERY_WEAK* Crack within one week • *F_BRAIN_WALLET_WEAK* Crack within one month • *F_BRAIN_WALLET_STILL_WEAK* Crack within one year • *F_BRAIN_WALLET_MAYBE_GOOD* Crack within one century • *F_BRAIN_WALLET_GOOD* Crack within one thousand year • *F_BRAIN_WALLET_VERY_GOOD* Crack within ten thousand year • *F_BRAIN_WALLET_NICE* Crack withing one hundred thousand year • *F_BRAIN_WALLET_PERFECT* 3.34x10^53 Years to crack
in	*brainwallet*	Brainwallet text to be parsed. It can be NOT NULL or null string
in	*salt*	Salt of the Braiwallet. It can be NOT NULL or null string

Return values

0	If success, otherwise error.

See also

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

5.5.5.5 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

5.5.5.6 f_generate_token()

```
int f_generate_token (
              F_TOKEN signature,
             void * data,
             size_t data_sz,
             const char * password )
```

Generates a non deterministic token given a message data and a password.

Parameters

out	*signature*	128 bit non deterministic token
in	*data*	Data to be signed in token
in	*data_sz*	Size of data
in	*password*	Password

Return values

0	On Success, otherwise Error

See also

f_verify_token() (p. **??**)

5.5.5.7 f_get_dictionary_path()

```
char * f_get_dictionary_path (
             void  )
```

Get default dictionary path in **myNanoEmbedded** library.

Return values

Path	and name of the dictionary file

See also

f_set_dictionary_path() (p. **??**)

5.5.5.8 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

5.5.5.9 f_is_valid_nano_seed_encrypted()

```
int f_is_valid_nano_seed_encrypted (
            void * stream,
            size_t stream_len,
            int read_from )
```

Verifies if ecrypted Nano SEED is valid.

Parameters

in	*stream*	Encrypted binary data block coming from memory or file
in	*stream_len*	size of *stream* data
in	*read_from*	Source *READ_SEED_FROM_STREAM* if encrypted binary data is in memory or *READ_SEED_FROM_FILE* is in a file.

Return values

0	If invalid, greater than zero if is valid or error if less than zero.

5.5.5.10 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_RES_RAW_128* Output is a raw data 128 bit big number result • *F_NANO_RES_RAW_STRING* Output is a 128 bit Big Integer string • *F_NANO_RES_REAL_STRING* Output is a Real string value • *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

5.5.5.11 f_nano_balance_to_str()

```
f_nano_err f_nano_balance_to_str (
            char * str,
            size_t str_len,
            size_t * out_len,
             f_uint128_t value )
```

Converts a raw Nano balance to string raw balance.

Parameters

out	*str*	Output string pointer
in	*str_len*	Size of string pointer memory
out	*out_len*	Output length of converted value to string. If *out_len* is NULL then *str* returns converted value with NULL terminated string
in	*value*	Raw Nano balance value

Return values

0	If success, otherwise error.

See also

function **f_nano_parse_raw_str_to_raw128_t()** (p. **??**) and return errors **f_nano_err** (p. **??**)

5.5.5.12 f_nano_block_to_json()

```
int f_nano_block_to_json (
            char * dest,
            size_t * olen,
            size_t dest_size,
            F_BLOCK_TRANSFER * user_block )
```

Parse a Nano Block to JSON.

Parameters

out	*dest*	Destination of the converted JSON block
out	*olen*	Output length of the converted JSON block. *olen* can be NULL. If NULL, destination size contains a NULL char
in	*dest_size*	Size of *destmemory buffer*
in	*user_block*	*User Nano block*

Returns

0 if success, non zero if error

5.5.5.13 f_nano_get_block_hash()

```
int f_nano_get_block_hash (
            uint8_t * hash,
            F_BLOCK_TRANSFER * block )
```

Gets a hash from Nano block.

Parameters

out	*hash*	Output hash
in	*block*	Nano Block

Returns

0 if success, non zero if error

5.5.5.14 f_nano_get_p2pow_block_hash()

```
int f_nano_get_p2pow_block_hash (
            uint8_t * user_hash,
            uint8_t * fee_hash,
            F_BLOCK_TRANSFER * block )
```

Get Nano user block hash and Nano fee block hashes from P2PoW block.

Parameters

out	*user_hash*	Hash of the user block
out	*fee_hash*	Hash of the P2PoW block
in	*block*	Input Nano Block

Returns

0 if success, non zero if error

5.5.5.15 f_nano_is_valid_block()

```
int f_nano_is_valid_block (
            F_BLOCK_TRANSFER * block )
```

Checks if Binary Nano Block is valid.

Parameters

in	*block*	Nano Block

Returns

0 if is invalid block or 1 if is valid block

5.5.5.16 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

5.5.5.17 f_nano_p2pow_to_JSON()

```
int f_nano_p2pow_to_JSON (
            char * buffer,
            size_t * olen,
            size_t buffer_sz,
            F_BLOCK_TRANSFER * block )
```

Parse binary P2PoW block to JSON.

Parameters

out	*buffer*	Output JSON string
out	*olen*	Output JSON string size. *olen* can be NULL. If NULL, *buffer* will be terminated with a NULL char
in	*buffer_sz*	Size of memory buffer
in	*block*	P2PoW block

Returns

0 if success, non zero if error

5.5.5.18 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

5.5.5.19 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

5.5.5.20 f_nano_pow()

```
int f_nano_pow (
            uint64_t * PoW_res,
            unsigned char * hash,
            const uint64_t threshold,
            int n_thr )
```

Calculates a Proof of Work given a *hash*, *threshold* and number of threads *n_thr*

Parameters

out	*PoW_res*	Output Proof of Work
in	*hash*	Input *hash*
in	*threshold*	Input *threshold*
in	*n_thr*	Number of threads. Default maximum value: 10. You can modify *F_NANO_POW_MAX_THREAD* in **f_nano_crypto_util.h** (p. **??**)

Mandatory: You need to enable attach a random function to your project using **f_random_attach()** (p. **??**)

Return values

0	If success, otherwise error.

See also

f_verify_work() (p. **??**)

5.5.5.21 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. **??**)

5.5.5.22 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. **??**)

5.5.5.23 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. **??**)

5.5.5.24 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

5.5.5.25 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. **??**)

5.5.5.26 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_GT* if *valA* is greater 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

5.5.5.27 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

5.5.5.28 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. **??**)

5.5.5.29 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. **??**) **f_write_seed()** (p. **??**)

5.5.5.30 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

5.5.5.31 f_set_dictionary_path()

```
void f_set_dictionary_path (
            const char * path )
```

Set default dictionary file and path to **myNanoEmbedded** library.

Parameters

in	*path*	Path to dictionary file

If **f_set_dictionary_path()** (p. **??**) is not used in **myNanoEmbedded** library then default path stored in *BIP39_DICTIONARY* is used

See also

f_get_dictionary_path() (p. **??**)

5.5.5.32 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

5.5.5.33 f_sign_data()

```
int f_sign_data (
            unsigned char * signature,
            void * out_public_key,
            uint32_t ouput_type,
            const unsigned char * message,
            size_t msg_len,
            const unsigned char * private_key )
```

Signs a *message* with a deterministic signature given a *private key*

Parameters

out	*signature*	Output signature
out	*out_public_key*	Output public key. It can be NULL
in	*output_type*	Output type of public key. Public key types are: • *F_SIGNATURE_RAW* Signature is raw 64 bytes long • *F_SIGNATURE_STRING* Singnature is hex ASCII encoded string • *F_SIGNATURE_OUTPUT_RAW_PK* Public key is raw 32 bytes data • *F_SIGNATURE_OUTPUT_STRING_PK* Public key is hes ASCII encoded string • *F_SIGNATURE_OUTPUT_XRB_PK* Public key is a XRB wallet encoded base32 string • *F_SIGNATURE_OUTPUT_NANO_PK* Public key is a NANO wallet encoded base32 string
in	*message*	Message to be signed with Elliptic Curve Ed25519 with blake2b hash
in	*msg_len*	Size of message to be signed
in	*private_key*	Private key to sign message

Return values

0	If success, otherwise error.

See also

f_verify_signed_data() (p. **??**)

5.5.5.34 f_verify_signed_data()

```
int f_verify_signed_data (
            const unsigned char * signature,
            const unsigned char * message,
            size_t message_len,
            const void * public_key,
            uint32_t pk_type )
```

Verifies if a signed message is valid.

Parameters

in	*signature*	Signature of the *message*
in	*message*	Message to be verified
in	*message_len*	Length of the message
in	*public_key*	Public key to verify signed message
in	*pk_type*	Type of the public key. Types are: • *F_VERIFY_SIG_NANO_WALLET* Public key is a NANO wallet with *XRB* or *NANO* prefixes encoded base32 string • *F_VERIFY_SIG_RAW_HEX* Public key is raw 32 bytes data • *F_VERIFY_SIG_ASCII_HEX* Public key is a hex ASCII encoded string

Return value are

- Greater than zero if *signature* is VALID
- 0 (zero) if *signature* is INVALID
- Negative if ERROR occurred

See also

f_sign_data() (p. **??**)

5.5.5.35 f_verify_token()

```
int f_verify_token (
              F_TOKEN signature,
             void * data,
             size_t data_sz,
             const char * password )
```

Verifies if a token is valid given data and password.

Parameters

in	*signature*	128 bit non deterministic token
in	*data*	Data to be signed in token
in	*data_sz*	Size of data
in	*password*	Password

Return values

0	On if invalid; 1 if valid ; less than zero if an error occurs

See also

f_generate_token() (p. **??**)

5.5.5.36 f_verify_work()

```
int f_verify_work (
            uint64_t * result,
            const unsigned char * hash,
            uint64_t * work,
            uint64_t threshold )
```

Verifies if Proof of Work of a given *hash* is valid.

Parameters

out	*result*	Result of work. It can be NULL
in	*hash*	Input *hash* for verification
in	*work*	Work previously calculated to be checked
in	*threshold*	Input *threshold*

Return values

0	If is not valid or less than zero if error or greater than zero if is valid

See also

f_nano_pow() (p. **??**)

5.5.5.37 f_write_seed()

```
f_write_seed_err f_write_seed (
            void * source_data,
```

```
            int source,
            uint8_t * seed,
            char * passwd )
```

Writes a SEED into a ecrypted with password with non deterministic stream in memory or file.

Parameters

out	*source_data*	Memory pointer or file name
in	*source*	Source of output data: • *WRITE_SEED_TO_STREAM* Output data is a pointer to memory to store encrypted Nano SEED data • *WRITE_SEED_TO_FILE* Output is a string filename to store encrypted Nano SEED data
in	*seed*	Nano SEED to be stored in encrypted stream or file
in	*passwd*	(Mandatory) It can not be null string or NULL. See ***f_pass_must_have_at_least()*** *(*p. **??***)* function to check passwords strength

Return values

0	If Success, otherwise error

See also

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

5.5.5.38 from_multiplier()

```
uint64_t from_multiplier (
            double multiplier,
            uint64_t base_difficulty )
```

Calculates a PoW given a multiplier and base difficulty.

Parameters

in	*multiplier*	Multiplier of the work
in	*base_difficulty*	Base difficulty Details here

See also

to_multiplier() (p. **??**)

Return values

Calculated	value

5.5.5.39 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

5.5.5.40 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

5.5.5.41 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. **??**)

5.5.5.42 nano_create_block_dynamic()

```
int nano_create_block_dynamic (
             F_BLOCK_TRANSFER ** ,
             const void * ,
             size_t ,
             const void * ,
             size_t ,
             const void * ,
             size_t ,
             const void * ,
             const void * ,
             uint32_t ,
             const void * ,
             int  )
```

5.5.5.43 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. **??**)

5.5.5.44 to_multiplier()

```
double to_multiplier (
            uint64_t difficulty,
            uint64_t base_difficulty )
```

Calculates a relative difficulty compared PoW with another.

Parameters

in	*dificulty*	Work difficulty
in	*base_difficulty*	Base difficulty Details here

See also

from_multiplier() (p. **??**)

Return values

Calculated	value

5.5.5.45 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

5.5.5.46 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

5.5.6 Variable Documentation

5.5.6.1 account

```
uint8_t account[32]
```

Account in raw binary data.

Definition at line **259** of file **f_nano_crypto_util.h**.

5.5.6.2 balance

```
f_uint128_t balance
```

Big number 128 bit raw balance.

See also

f_uint128_t (p. **??**)

Definition at line **267** of file **f_nano_crypto_util.h**.

5.5.6.3 body

```
F_NANO_WALLET_INFO_BODY body
```

Body of the file info.

Definition at line **267** of file **f_nano_crypto_util.h**.

5.5.6.4 desc

```
char desc[F_NANO_DESC_SZ]
```

Description.

Definition at line **261** of file **f_nano_crypto_util.h**.

5.5.6.5 description

```
uint8_t description[F_DESC_SZ]
```

File description.

Definition at line **261** of file **f_nano_crypto_util.h**.

5.5.6.6 file_info_integrity

```
uint8_t file_info_integrity[32]
```

File info integrity of the body block.

Definition at line **265** of file **f_nano_crypto_util.h**.

5.5.6.7 hash_sk_unencrypted

```
uint8_t hash_sk_unencrypted[32]
```

hash of Nano SEED when unencrypted

Definition at line **263** of file **f_nano_crypto_util.h**.

5.5.6.8 header

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

Header magic.

Definition at line **257** of file **f_nano_crypto_util.h**.

5.5.6.9 iv

```
uint8_t iv
```

Initial sub vector.

Initial vector of first encryption layer.

Definition at line **259** of file **f_nano_crypto_util.h**.

5.5.6.10 last_used_wallet_number

```
uint32_t last_used_wallet_number
```

Last used wallet number.

Definition at line **259** of file **f_nano_crypto_util.h**.

5.5.6.11 link

```
uint8_t link[32]
```

link or destination account

Definition at line **269** of file **f_nano_crypto_util.h**.

5.5.6.12 max_fee

```
char max_fee[F_RAW_STR_MAX_SZ]
```

Custom preferred max fee of Proof of Work.

Definition at line **263** of file **f_nano_crypto_util.h**.

5.5.6.13 nano_hdr

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

Header of the file.

Definition at line **257** of file **f_nano_crypto_util.h**.

5.5.6.14 nanoseed_hash

```
uint8_t nanoseed_hash[32]
```

Nano SEED hash file.

Definition at line **263** of file **f_nano_crypto_util.h**.

5.5.6.15 preamble

```
uint8_t preamble[32]
```

Block preamble.

Definition at line **257** of file **f_nano_crypto_util.h**.

5.5.6.16 prefixes

```
uint8_t prefixes
```

Internal use for this API.

Definition at line **273** of file **f_nano_crypto_util.h**.

5.5.6.17 previous

```
uint8_t previous[32]
```

Previous block.

Definition at line **261** of file **f_nano_crypto_util.h**.

5.5.6.18 representative

```
uint8_t representative[32]
```

Representative for current account.

Definition at line **263** of file **f_nano_crypto_util.h**.

5.5.6.19 reserved

```
uint8_t reserved
```

Reserved (not used)

Reserved.

Definition at line **261** of file **f_nano_crypto_util.h**.

5.5.6.20 salt

```
uint8_t salt[32]
```

Salt of the first encryption layer.

Definition at line **263** of file **f_nano_crypto_util.h**.

5.5.6.21 seed_block

```
F_ENCRYPTED_BLOCK seed_block
```

Second encrypted block for Nano SEED.

Definition at line **267** of file **f_nano_crypto_util.h**.

5.5.6.22 signature

```
uint8_t signature[64]
```

Signature of the block.

Definition at line **271** of file **f_nano_crypto_util.h**.

5.5.6.23 sk_encrypted

```
uint8_t sk_encrypted[32]
```

Secret.

SEED encrypted (second layer)

Definition at line **265** of file **f_nano_crypto_util.h**.

5.5.6.24 sub_salt

```
uint8_t sub_salt[32]
```

Salt of the sub block to be stored.

Definition at line **257** of file **f_nano_crypto_util.h**.

5.5.6.25 ver

```
uint32_t ver
```

Version of the file.

Definition at line **259** of file **f_nano_crypto_util.h**.

5.5.6.26 version

```
uint16_t version
```

Version.

Definition at line **259** of file **f_nano_crypto_util.h**.

5.5.6.27 wallet_prefix

```
uint8_t wallet_prefix
```

Wallet prefix: 0 for NANO; 1 for XRB.

Definition at line **257** of file **f_nano_crypto_util.h**.

5.5.6.28 wallet_representative

```
char wallet_representative[ MAX_STR_NANO_CHAR]
```

Wallet representative.

Definition at line **261** of file **f_nano_crypto_util.h**.

5.5.6.29 work

```
uint64_t work
```

Internal use for this API.

Definition at line **275** of file **f_nano_crypto_util.h**.

5.6 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 #include <f_bitcoin.h>
00011
00012 #ifndef F_DOC_SKIP
00013
00014  #ifdef F_XTENSA
00015
00016   #ifndef F_ESP32
00017    #define F_ESP32
00018   #endif
00019
00020   #include "esp_system.h"
00021
00022  #endif
00023
00024  #include "sodium/crypto_generichash.h"
00025  #include "sodium/crypto_sign.h"
00026  #include "sodium.h"
00027
00028  #ifdef F_ESP32
00029
00030   #include "sodium/private/curve25519_ref10.h"
00031
00032  #else
00033
00034   #include "sodium/private/ed25519_ref10.h"
00035
00036   #define ge_p3 ge25519_p3
00037   #define sc_reduce sc25519_reduce
00038   #define sc_muladd sc25519_muladd
00039   #define ge_scalarmult_base ge25519_scalarmult_base
00040   #define ge_p3_tobytes ge25519_p3_tobytes
00041
00042  #endif
00043
00044 #endif
00045
00128 #ifdef __cplusplus
00129 extern "C" {
00130 #endif
00131
00132
00137 #define F_NANO_POW_MAX_THREAD (size_t)10
```

```
00138
00139 #ifndef F_DOC_SKIP
00140  #ifdef F_ESP32
00141   #undef F_NANO_POW_MAX_THREAD
00142  #endif
00143 #endif
00144
00149 #define MAX_STR_NANO_CHAR (size_t)70 //5+56+8+1
00150
00155 #define PUB_KEY_EXTENDED_MAX_LEN (size_t)40
00156
00161 #define NANO_PREFIX "nano_"
00162
00167 #define XRB_PREFIX "xrb_"
00168
00169 #ifdef F_ESP32
00170
00175 #define BIP39_DICTIONARY "/spiffs/dictionary.dic"
00176 #else
00177
00178  #ifndef F_DOC_SKIP
00179   #define BIP39_DICTIONARY_SAMPLE "../../dictionary.dic"
00180   #define BIP39_DICTIONARY "dictionary.dic"
00181  #endif
00182
00183 #endif
00184
00191 #define NANO_ENCRYPTED_SEED_FILE "/spiffs/secure/nano.nse"
00192
00197 #define NANO_PASSWD_MAX_LEN (size_t)80
00198
00203 #define STR_NANO_SZ (size_t)66// 65+1 Null included
00204
00209 #define NANO_FILE_WALLETS_INFO "/spiffs/secure/walletsinfo.i"
00210
00215 typedef uint8_t F_TOKEN[16];
00216
00221 typedef uint8_t NANO_SEED[crypto_sign_SEEDBYTES];
00222
00227 typedef uint8_t f_uint128_t[16];
00228
00229 #ifndef F_DOC_SKIP
00230  #define EXPORT_KEY_TO_CHAR_SZ (size_t)sizeof(NANO_SEED)+1
00231 #endif
00232
00237 typedef uint8_t NANO_PRIVATE_KEY[sizeof(NANO_SEED)];
00238
00243 typedef uint8_t NANO_PRIVATE_KEY_EXTENDED[crypto_sign_ed25519_SECRETKEYBYTES];
00244
00249 typedef uint8_t NANO_PUBLIC_KEY[crypto_sign_ed25519_PUBLICKEYBYTES];
00250
00255 typedef uint8_t NANO_PUBLIC_KEY_EXTENDED[PUB_KEY_EXTENDED_MAX_LEN];
00256
00265 typedef struct f_block_transfer_t {
00267    uint8_t preamble[32];
00269    uint8_t account[32];
00271    uint8_t previous[32];
00273    uint8_t representative[32];
00277    f_uint128_t balance;
00279    uint8_t link[32];
00281    uint8_t signature[64];
00283    uint8_t prefixes;
00285    uint64_t work;
00286 } __attribute__((packed)) F_BLOCK_TRANSFER;
00287
00288 #ifndef F_DOC_SKIP
00289  #define F_BLOCK_TRANSFER_SIGNABLE_SZ
       (size_t)(sizeof(F_BLOCK_TRANSFER)-64-sizeof(uint64_t)-sizeof(uint8_t))
00290 #endif
00291
00299 typedef enum f_nano_err_t {
00301    NANO_ERR_OK=0,
00303    NANO_ERR_CANT_PARSE_BN_STR=5151,
00305    NANO_ERR_MALLOC,
00307    NANO_ERR_CANT_PARSE_FACTOR,
00309    NANO_ERR_MPI_MULT,
00311    NANO_ERR_CANT_PARSE_TO_BLK_TRANSFER,
00313    NANO_ERR_EMPTY_STR,
00315    NANO_ERR_CANT_PARSE_VALUE,
00317    NANO_ERR_PARSE_MPI_TO_STR,
00319    NANO_ERR_CANT_COMPLETE_NULL_CHAR,
00321    NANO_ERR_CANT_PARSE_TO_MPI,
00323    NANO_ERR_INSUFICIENT_FUNDS,
00325    NANO_ERR_SUB_MPI,
00327    NANO_ERR_ADD_MPI,
00329    NANO_ERR_NO_SENSE_VALUE_TO_SEND_NEGATIVE,
00331    NANO_ERR_NO_SENSE_VALUE_TO_SEND_ZERO,
```

```
00333    NANO_ERR_NO_SENSE_BALANCE_NEGATIVE,
00335    NANO_ERR_VAL_A_INVALID_MODE,
00337    NANO_ERR_CANT_PARSE_TO_TEMP_UINT128_T,
00339    NANO_ERR_VAL_B_INVALID_MODE,
00341    NANO_ERR_CANT_PARSE_RAW_A_TO_MPI,
00343    NANO_ERR_CANT_PARSE_RAW_B_TO_MPI,
00345    NANO_ERR_UNKNOWN_ADD_SUB_MODE,
00347    NANO_ERR_INVALID_RES_OUTPUT
00348 } f_nano_err;
00349
00350 #ifndef F_DOC_SKIP
00351
00352  #define READ_SEED_FROM_STREAM (int)1
00353  #define READ_SEED_FROM_FILE (int)2
00354  #define WRITE_SEED_TO_STREAM (int)4
00355  #define WRITE_SEED_TO_FILE (int)8
00356  #define PARSE_JSON_READ_SEED_GENERIC (int)16
00357  #define F_STREAM_DATA_FILE_VERSION (uint32_t)((1<<16)|0)
00358
00359 #endif
00360
00368 typedef struct f_nano_encrypted_wallet_t {
00370    uint8_t sub_salt[32];
00372    uint8_t iv[16];
00374    uint8_t reserved[16];
00376    uint8_t hash_sk_unencrypted[32];
00378    uint8_t sk_encrypted[32];
00379 } __attribute__ ((packed)) F_ENCRYPTED_BLOCK;
00380
00381 #ifndef F_DOC_SKIP
00382
00383  static const uint8_t NANO_WALLET_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't', 'f',
      'i', 'l', 'e', '_'};
00384  #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 !!!"
00385  #define F_DESC_SZ (size_t) (160-sizeof(uint32_t))
00386
00387 #endif
00388
00396 typedef struct f_nano_crypto_wallet_t {
00398    uint8_t nano_hdr[sizeof(NANO_WALLET_MAGIC)];
00400    uint32_t ver;
00402    uint8_t description[F_DESC_SZ];
00404    uint8_t salt[32];
00406    uint8_t iv[16];
00408    F_ENCRYPTED_BLOCK seed_block;
00409 } __attribute__ ((packed)) F_NANO_CRYPTOWALLET;
00410
00411 #ifndef F_DOC_SKIP
00412
00413 _Static_assert((sizeof(F_NANO_CRYPTOWALLET)&0x1F)==0, "Error 1");
00414 _Static_assert((sizeof(F_ENCRYPTED_BLOCK)&0x1F)==0, "Error 2");
00415
00416 #endif
00417
00422 #define REP_XRB (uint8_t)0x4
00423
00428 #define SENDER_XRB (uint8_t)0x02
00429
00434 #define DEST_XRB (uint8_t)0x01
00435
00436 typedef enum f_write_seed_err_t {
00438    WRITE_ERR_OK=0,
00440    WRITE_ERR_NULL_PASSWORD=7180,
00442    WRITE_ERR_EMPTY_STRING,
00444    WRITE_ERR_MALLOC,
00446    WRITE_ERR_ENCRYPT_PRIV_KEY,
00448    WRITE_ERR_GEN_SUB_PRIV_KEY,
00450    WRITE_ERR_GEN_MAIN_PRIV_KEY,
00452    WRITE_ERR_ENCRYPT_SUB_BLOCK,
00454    WRITE_ERR_UNKNOWN_OPTION,
00456    WRITE_ERR_FILE_ALREDY_EXISTS,
00458    WRITE_ERR_CREATING_FILE,
00460    WRITE_ERR_WRITING_FILE
00461 } f_write_seed_err;
00462
00463 #ifndef F_DOC_SKIP
00464
00465  #define F_RAW_TO_STR_UINT128 (int)1
00466  #define F_RAW_TO_STR_STRING (int)2
00467  #define F_RAW_STR_MAX_SZ (size_t)41 // 39 + '\0' + '.' -> 39 = log10(2^128)
00468  #define F_MAX_STR_RAW_BALANCE_MAX (size_t)40 //39+'\0'
00469  #define F_NANO_EMPTY_BALANCE "0.0"
00470
00471 #endif
00472
00480 typedef struct f_nano_wallet_info_bdy_t {
```

```
00482    uint8_t wallet_prefix; // 0 for NANO; 1 for XRB
00484    uint32_t last_used_wallet_number;
00486    char wallet_representative[MAX_STR_NANO_CHAR];
00488    char max_fee[F_RAW_STR_MAX_SZ];
00490    uint8_t reserved[44];
00491 } __attribute__((packed)) F_NANO_WALLET_INFO_BODY;
00492
00493 #ifndef F_DOC_SKIP
00494
00495  _Static_assert((sizeof(F_NANO_WALLET_INFO_BODY)&0x1F)==0, "Error F_NANO_WALLET_INFO_BODY is not byte
      aligned");
00496
00497  #define F_NANO_WALLET_INFO_DESC "Nano file descriptor used for fast custom access. BUY BITCOIN AND NANO."
00498  #define F_NANO_WALLET_INFO_VERSION (uint16_t)((1<<8)|1)
00499  static const uint8_t F_NANO_WALLET_INFO_MAGIC[] = {'_', 'n', 'a', 'n', 'o', 'w', 'a', 'l', 'l', 'e', 't',
     '_', 'n', 'f', 'o', '_'};
00500
00501  #define F_NANO_DESC_SZ (size_t)78
00502
00503 #endif
00504
00512 typedef struct f_nano_wallet_info_t {
00514    uint8_t header[sizeof(F_NANO_WALLET_INFO_MAGIC)];
00516    uint16_t version;
00518    char desc[F_NANO_DESC_SZ];
00520    uint8_t nanoseed_hash[32];
00522    uint8_t file_info_integrity[32];
00524    F_NANO_WALLET_INFO_BODY body;
00525 } __attribute__((packed)) F_NANO_WALLET_INFO;
00526
00527 #ifndef F_DOC_SKIP
00528
00529  _Static_assert((sizeof(F_NANO_WALLET_INFO)&0x1F)==0, "Error F_NANO_WALLET_INFO is not byte aligned");
00530
00531 #endif
00532
00540 typedef enum f_file_info_err_t {
00542    F_FILE_INFO_ERR_OK=0,
00544    F_FILE_INFO_ERR_CANT_OPEN_INFO_FILE=7001,
00546    F_FILE_INFO_ERR_NANO_SEED_ENCRYPTED_FILE_NOT_FOUND,
00548    F_FILE_INFO_ERR_CANT_DELETE_NANO_INFO_FILE,
00550    F_FILE_INFO_ERR_MALLOC,
00552    F_FILE_INFO_ERR_CANT_READ_NANO_SEED_ENCRYPTED_FILE,
00554    F_FILE_INFO_ERR_CANT_READ_INFO_FILE,
00556    F_FILE_INFO_INVALID_HEADER_FILE,
00558    F_FILE_INFO_ERR_INVALID_SHA256_INFO_FILE,
00560    F_FILE_INFO_ERR_NANO_SEED_HASH_FAIL,
00562    F_FILE_INFO_ERR_NANO_INVALID_REPRESENTATIVE,
00564    F_FILE_INFO_ERR_NANO_INVALID_MAX_FEE_VALUE,
00566    F_FILE_INFO_ERR_OPEN_FOR_WRITE_INFO,
00568    F_FILE_INFO_ERR_EXISTING_FILE,
00570    F_FILE_INFO_ERR_CANT_WRITE_FILE_INFO
00571 } F_FILE_INFO_ERR;
00572
00573 #ifndef F_DOC_SKIP
00574
00575  #define F_NANO_ADD_A_B (uint32_t)(1<<0)
00576  #define F_NANO_SUB_A_B (uint32_t)(1<<1)
00577  #define F_NANO_A_RAW_128 (uint32_t)(1<<2)
00578  #define F_NANO_A_RAW_STRING (uint32_t)(1<<3)
00579  #define F_NANO_A_REAL_STRING (uint32_t)(1<<4)
00580  #define F_NANO_B_RAW_128 (uint32_t)(1<<5)
00581  #define F_NANO_B_RAW_STRING (uint32_t)(1<<6)
00582  #define F_NANO_B_REAL_STRING (uint32_t)(1<<7)
00583  #define F_NANO_RES_RAW_128 (uint32_t)(1<<8)
00584  #define F_NANO_RES_RAW_STRING (uint32_t)(1<<9)
00585  #define F_NANO_RES_REAL_STRING (uint32_t)(1<<10)
00586  #define F_NANO_C_RAW_128 (uint32_t)(F_NANO_B_RAW_128<<16)
00587  #define F_NANO_C_RAW_STRING (uint32_t)(F_NANO_B_RAW_STRING<<16)
00588  #define F_NANO_C_REAL_STRING (uint32_t)(F_NANO_B_REAL_STRING<<16)
00589
00590  #define F_NANO_COMPARE_EQ (uint32_t)(1<<16) //Equal
00591  #define F_NANO_COMPARE_LT (uint32_t)(1<<17) // Lesser than
00592  #define F_NANO_COMPARE_LEQ (F_NANO_COMPARE_LT|F_NANO_COMPARE_EQ) // Less or equal
00593  #define F_NANO_COMPARE_GT (uint32_t)(1<<18) // Greater
00594  #define F_NANO_COMPARE_GEQ (F_NANO_COMPARE_GT|F_NANO_COMPARE_EQ) // Greater or equal
00595  #define DEFAULT_MAX_FEE "0.001"
00596
00597 #endif
00598
00599 #ifndef F_ESP32
00600 typedef enum f_nano_create_block_dyn_err_t {
00601    NANO_CREATE_BLK_DYN_OK = 0,
00602    NANO_CREATE_BLK_DYN_BLOCK_NULL = 8000,
00603    NANO_CREATE_BLK_DYN_ACCOUNT_NULL,
00604    NANO_CREATE_BLK_DYN_PREV_NULL,
00605    NANO_CREATE_BLK_DYN_REP_NULL,
```

```
00606    NANO_CREATE_BLK_DYN_BALANCE_NULL,
00607    NANO_CREATE_BLK_DYN_SEND_RECEIVE_NULL,
00608    NANO_CREATE_BLK_DYN_LINK_NULL,
00609    NANO_CREATE_BLK_DYN_BUF_MALLOC,
00610    NANO_CREATE_BLK_DYN_MALLOC,
00611    NANO_CREATE_BL_DYN_WRONG_PREVIOUS_SZ,
00612    NANO_CREATE_BL_DYN_WRONG_PREVIOUS_STR_SZ,
00613    NANO_CREATE_BL_DYN_PARSE_STR_HEX_ERR
00614
00615 } F_NANO_CREATE_BLOCK_DYN_ERR;
00616
00617 #endif
00618
00630 double to_multiplier(uint64_t, uint64_t);
00631
00643 uint64_t from_multiplier(double, uint64_t);
00644
00654 void f_set_dictionary_path(const char *);
00655
00663 char *f_get_dictionary_path(void);
00664
00677 int f_generate_token(F_TOKEN, void *, size_t, const char *);
00678
00691 int f_verify_token(F_TOKEN, void *, size_t, const char *);
00692
00715 int f_cloud_crypto_wallet_nano_create_seed(size_t, char *, char *);
00716
00729 int f_generate_nano_seed(NANO_SEED, uint32_t);
00730
00745 int pk_to_wallet(char *, char *, NANO_PUBLIC_KEY_EXTENDED);
00746
00764 int f_seed_to_nano_wallet(NANO_PRIVATE_KEY, NANO_PUBLIC_KEY, NANO_SEED, uint32_t);
00765
00775 int f_nano_is_valid_block(F_BLOCK_TRANSFER *);
00776
00789 int f_nano_block_to_json(char *, size_t *, size_t, F_BLOCK_TRANSFER *);
00790
00801 int f_nano_get_block_hash(uint8_t *, F_BLOCK_TRANSFER *);
00802
00814 int f_nano_get_p2pow_block_hash(uint8_t *, uint8_t *, F_BLOCK_TRANSFER *);
00815
00828 int f_nano_p2pow_to_JSON(char *, size_t *, size_t, F_BLOCK_TRANSFER *);
00829
00839 char *f_nano_key_to_str(char *, unsigned char *);
00840
00859 int f_nano_seed_to_bip39(char *, size_t, size_t *, NANO_SEED, char *);
00860
00875 int f_bip39_to_nano_seed(uint8_t *, char *, char *);
00876
00898 int f_parse_nano_seed_and_bip39_to_JSON(char *, size_t, size_t *, void *, int, const char *);
00899
00917 int f_read_seed(uint8_t *, const char *, void *, int, int);
00918
00933 int f_nano_raw_to_string(char *, size_t *, size_t, void *, int);
00934
00943 int f_nano_valid_nano_str_value(const char *);
00944
00952 int valid_nano_wallet(const char *);
00953
00963 int nano_base_32_2_hex(uint8_t *, char *);
00964
00979 int f_nano_transaction_to_JSON(char *, size_t, size_t *, NANO_PRIVATE_KEY_EXTENDED, F_BLOCK_TRANSFER *);
00980
00988 int valid_raw_balance(const char *);
00989
00997 int is_null_hash(uint8_t *);
00998
01010 int is_nano_prefix(const char *, const char *);
01011
01020 F_FILE_INFO_ERR f_get_nano_file_info(F_NANO_WALLET_INFO *);
01021
01031 F_FILE_INFO_ERR f_set_nano_file_info(F_NANO_WALLET_INFO *, int);
01032
01054 f_nano_err f_nano_value_compare_value(void *, void *, uint32_t *);
01055
01076 f_nano_err f_nano_verify_nano_funds(void *, void *, void *, uint32_t);
01077
01087 f_nano_err f_nano_parse_raw_str_to_raw128_t(uint8_t *, const char *);
01088
01098 f_nano_err f_nano_parse_real_str_to_raw128_t(uint8_t *, const char *);
01099
01122 f_nano_err f_nano_add_sub(void *, void *, void *, uint32_t);
01123
01134 int f_nano_sign_block(F_BLOCK_TRANSFER *, F_BLOCK_TRANSFER *, NANO_PRIVATE_KEY_EXTENDED);
01135
01149 f_write_seed_err f_write_seed(void *, int, uint8_t *, char *);
01150
```

```
01163 f_nano_err f_nano_balance_to_str(char *, size_t, size_t *, f_uint128_t);
01164
01165
01170 #define F_BRAIN_WALLET_VERY_POOR (uint32_t)0
01171
01176 #define F_BRAIN_WALLET_POOR (uint32_t)1
01177
01182 #define F_BRAIN_WALLET_VERY_BAD (uint32_t)2
01183
01188 #define F_BRAIN_WALLET_BAD (uint32_t)3
01189
01194 #define F_BRAIN_WALLET_VERY_WEAK (uint32_t)4
01195
01200 #define F_BRAIN_WALLET_WEAK (uint32_t)5
01201
01206 #define F_BRAIN_WALLET_STILL_WEAK (uint32_t)6
01207
01212 #define F_BRAIN_WALLET_MAYBE_GOOD (uint32_t)7
01213
01214
01219 #define F_BRAIN_WALLET_GOOD (uint32_t)8
01220
01225 #define F_BRAIN_WALLET_VERY_GOOD (uint32_t)9
01226
01231 #define F_BRAIN_WALLET_NICE (uint32_t)10
01232
01237 #define F_BRAIN_WALLET_PERFECT (uint32_t)11
01238
01265 int f_extract_seed_from_brainwallet(uint8_t *, char **, uint32_t, const char *, const char *);
01266
01278 int f_verify_work(uint64_t *, const unsigned char *, uint64_t *, uint64_t);
01279
01285 #define F_SIGNATURE_RAW (uint32_t)1
01286
01292 #define F_SIGNATURE_STRING (uint32_t)2
01293
01299 #define F_SIGNATURE_OUTPUT_RAW_PK (uint32_t)4
01300
01306 #define F_SIGNATURE_OUTPUT_STRING_PK (uint32_t)8
01307
01313 #define F_SIGNATURE_OUTPUT_XRB_PK (uint32_t)16
01314
01320 #define F_SIGNATURE_OUTPUT_NANO_PK (uint32_t)32
01321
01327 #define F_IS_SIGNATURE_RAW_HEX_STRING (uint32_t)64
01328
01334 #define F_MESSAGE_IS_HASH_STRING (uint32_t)128
01335
01340 #define F_DEFAULT_THRESHOLD (uint64_t) 0xfffffffc000000000
01341
01365 int f_sign_data(
01366    unsigned char *signature,
01367    void *out_public_key,
01368    uint32_t ouput_type,
01369    const unsigned char *message,
01370    size_t msg_len,
01371    const unsigned char *private_key);
01372
01378 #define F_VERIFY_SIG_NANO_WALLET (uint32_t)1
01379
01385 #define F_VERIFY_SIG_RAW_HEX (uint32_t)2
01386
01392 #define F_VERIFY_SIG_ASCII_HEX (uint32_t)4
01393
01414 int f_verify_signed_data( const unsigned char *, const unsigned char *, size_t, const void *, uint32_t);
01415
01425 int f_is_valid_nano_seed_encrypted(void *, size_t, int);
01426
01427 #ifndef F_ESP32
01428
01429 int nano_create_block_dynamic(
01430    F_BLOCK_TRANSFER **,
01431    const void *,
01432    size_t,
01433    const void *,
01434    size_t,
01435    const void *,
01436    size_t,
01437    const void *,
01438    const void *,
01439    uint32_t,
01440    const void *,
01441    int
01442 );
01443
01456 int f_nano_pow(uint64_t *, unsigned char *, const uint64_t, int);
01457 #endif
```

```
01458
01459 #ifdef __cplusplus
01460 }
01461 #endif
01462
```

5.7 f_util.h File Reference

```
#include <stdint.h>
#include "mbedtls/sha256.h"
#include "mbedtls/aes.h"
#include "mbedtls/ecdsa.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
- #define **F_GET_CH_MODE_NO_ECHO** (int)(1<<16)
- #define **F_GET_CH_MODE_ANY_KEY** (int)(1<<17)

Typedefs

- typedef void(* **rnd_fn**) (void *, size_t)
- typedef int(* **fn_det**) (void *, unsigned char *, 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)
- int **f_get_char_no_block** (int)

- int **f_convert_to_long_int** (unsigned long int *, char *, size_t)
- int **f_convert_to_unsigned_int** (unsigned int *, char *, size_t)
- int **f_convert_to_long_int0x** (unsigned long int *, char *, size_t)
- int **f_convert_to_long_int0** (unsigned long int *, char *, size_t)
- int **f_convert_to_long_int_std** (unsigned long int *, char *, size_t)
- void * **f_is_random_attached** ()
- void **f_random_detach** ()
- int **f_convert_to_unsigned_int0x** (unsigned int *val, char *value, size_t value_sz)
- int **f_convert_to_unsigned_int0** (unsigned int *val, char *value, size_t value_sz)
- int **f_convert_to_unsigned_int_std** (unsigned int *val, char *value, size_t value_sz)
- int **f_convert_to_double** (double *, const char *)
- uint32_t **crc32_init** (unsigned char *, size_t, uint32_t)
- int **f_reverse** (unsigned char *, size_t)
- f_md_hmac_sha512 **f_hmac_sha512** (unsigned char *, const unsigned char *, size_t, const unsigned char *, size_t)
- int **f_ecdsa_secret_key_valid** (mbedtls_ecp_group_id, unsigned char *, size_t)
- int **f_ecdsa_public_key_valid** (mbedtls_ecp_group_id, unsigned char *, size_t)
- f_ecdsa_key_pair_err **f_gen_ecdsa_key_pair** (f_ecdsa_key_pair *, int, **fn_det**, void *)
- int **f_uncompress_elliptic_curve** (uint8_t *, size_t, size_t *, mbedtls_ecp_group_id, uint8_t *, size_t)
- uint8_t * **f_ripemd160** (const uint8_t *, size_t)

5.7.1 Detailed Description

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

Definition in file **f_util.h**.

5.7.2 Macro Definition Documentation

5.7.2.1 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 **153** of file **f_util.h**.

5.7.2.2 ENTROPY_END

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

Ends a entropy function.

See also

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

Definition at line **160** of file **f_util.h**.

5.7.2.3 F_ENTROPY_TYPE_EXCELENT

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

Type of the excelent entropy used for verifier.

Slow

Definition at line **125** of file **f_util.h**.

5.7.2.4 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 **132** of file **f_util.h**.

5.7.2.5 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 **139** of file **f_util.h**.

5.7.2.6 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 **146** of file **f_util.h**.

5.7.2.7 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 **118** of file **f_util.h**.

5.7.2.8 F_GET_CH_MODE_ANY_KEY

```
#define F_GET_CH_MODE_ANY_KEY (int)(1<<17)
```

See also

f_get_char_no_block() (p. **??**)

Definition at line **359** of file **f_util.h**.

5.7.2.9 F_GET_CH_MODE_NO_ECHO

```
#define F_GET_CH_MODE_NO_ECHO (int)(1<<16)
```

See also

f_get_char_no_block() (p. **??**)

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

5.7.2.10 F_PASS_IS_OUT_OVF

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

Password is overflow and cannot be stored.

Definition at line **208** of file **f_util.h**.

5.7.2.11 F_PASS_IS_TOO_LONG

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

Password is too long.

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

5.7.2.12 F_PASS_IS_TOO_SHORT

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

Password is too short.

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

5.7.2.13 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 **166** of file **f_util.h**.

5.7.2.14 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 **190** of file **f_util.h**.

5.7.2.15 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 **172** of file **f_util.h**.

5.7.2.16 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 **178** of file **f_util.h**.

5.7.2.17 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 **184** of file **f_util.h**.

5.7.3 Typedef Documentation

5.7.3.1 fn_det

```
typedef int(* fn_det) (void *, unsigned char *, size_t)
```

Definition at line **523** of file **f_util.h**.

5.7.3.2 rnd_fn

```
rnd_fn
```

Pointer caller for random function.

Definition at line **318** of file **f_util.h**.

5.7.4 Function Documentation

5.7.4.1 crc32_init()

```
uint32_t crc32_init (
            unsigned char * p,
            size_t len,
            uint32_t crcinit )
```

Performs a CRC32 of a given data.

Parameters

in	*p*	Pointer of the data
in	*len*	Size of data in pointer *p*
in	*crcinit*	Init vector of the CRC32

Return values

CRC32	hash

5.7.4.2 f_convert_to_double()

```
int f_convert_to_double (
            double * val,
            const char * value )
```

Convert any valid number im *value* and converts it to double *val*

Parameters

out	*val*	Value converted to double
in	*value*	Value in string to be converted

Return values

0	On Success, Otherwise error

5.7.4.3 f_convert_to_long_int()

```
int f_convert_to_long_int (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a string value to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_unsigned_int() (p. **??**)

5.7.4.4 f_convert_to_long_int0()

```
int f_convert_to_long_int0 (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a octal value in ASCII string to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int0x() (p. **??**)

5.7.4.5 f_convert_to_long_int0x()

```
int f_convert_to_long_int0x (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a hex value in ASCII string to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int0() (p. **??**)

5.7.4.6 f_convert_to_long_int_std()

```
int f_convert_to_long_int_std (
            unsigned long int * val,
            char * value,
            size_t value_sz )
```

Converts a actal/decimal/hexadecimal into ASCII string to unsigned long int.

Parameters

out	*val*	Value stored in a unsigned long int variable
in	*value*	Input value to be parsed to unsigned long int • If a string contains only numbers, it will be parsed to unsigned long int decimal • If a string begins with 0 it will be parsed to octal EX.: 010(octal) = 08(decimal) • If a string contais 0x or 0X it will be parsed to hexadecimal. EX.: 0x10(hexadecimal) = 16 (decimal)
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int() (p. **??**)

5.7.4.7 f_convert_to_unsigned_int()

```
int f_convert_to_unsigned_int (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a string value to unsigned int.

Parameters

out	*val*	Value stored in a unsigned int variable
in	*value*	Input value to be parsed to unsigned int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_long_int() (p. **??**)

5.7.4.8 f_convert_to_unsigned_int0()

```
int f_convert_to_unsigned_int0 (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a octal value in ASCII string to unsigned int.

Parameters

out	*val*	Value stored in a unsigned int variable
in	*value*	Input value to be parsed to unsigned int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_unsigned_int0x() (p. **??**)

5.7.4.9 f_convert_to_unsigned_int0x()

```
int f_convert_to_unsigned_int0x (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a hex value in ASCII string to unsigned int.

Parameters

out	*val*	Value stored in a unsigned int variable
in	*value*	Input value to be parsed to unsigned int
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_unsigned_int0() (p. **??**)

5.7.4.10 f_convert_to_unsigned_int_std()

```
int f_convert_to_unsigned_int_std (
            unsigned int * val,
            char * value,
            size_t value_sz )
```

Converts a actal/decimal/hexadecimal into ASCII string to unsigned int.

Parameters

out	*val*	Value stored in a unsigned int variable
in	*value*	Input value to be parsed to unsigned int • If a string contains only numbers, it will be parsed to unsigned int decimal • If a string begins with 0 it will be parsed to octal EX.: 010(octal) = 08(decimal) • If a string contais 0x or 0X it will be parsed to hexadecimal. EX.: 0x10(hexadecimal) = 16 (decimal)
in	*value_sz*	Max size allowed in *value* string.

Return values

0	On Success, Otherwise error

See also

f_convert_to_unsigned_int() (p. **??**)

5.7.4.11 f_ecdsa_public_key_valid()

```
int f_ecdsa_public_key_valid (
            mbedtls_ecp_group_id ,
            unsigned char * ,
            size_t  )
```

5.7.4.12 f_ecdsa_secret_key_valid()

```
int f_ecdsa_secret_key_valid (
            mbedtls_ecp_group_id ,
            unsigned char * ,
            size_t  )
```

5.7.4.13 f_gen_ecdsa_key_pair()

```
f_ecdsa_key_pair_err f_gen_ecdsa_key_pair (
            f_ecdsa_key_pair * ,
            int ,
             fn_det ,
            void *  )
```

5.7.4.14 f_get_char_no_block()

```
int f_get_char_no_block (
            int mode )
```

Reads a char from console.

Waits a char and returns its value

Parameters

in	*mode*	Mode and/or character to be returned • *F_GET_CH_MODE_NO_ECHO* No echo is on the console string • *F_GET_CH_MODE_ANY_KEY* Returns any key pressed

Example:

```
key=f_get_char_no_block(F_GET_CH_MODE_NO_ECHO|'c'); // Waits 'c' char key and returns value 0x00000063
       without echo 'c' on the screen
```

Return values

key	code: On Success, Negative value on error

5.7.4.15 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

5.7.4.16 f_hmac_sha512()

```
f_md_hmac_sha512 f_hmac_sha512 (
            unsigned char * ,
            const unsigned char * ,
            size_t ,
            const unsigned char * ,
            size_t  )
```

5.7.4.17 f_is_random_attached()

```
void * f_is_random_attached ( )
```

Verifies if system random function is attached in myNanoEmbedded API.

Return values

NULL	if not attached, Otherwise returns the pointer of random number genarator function

See also

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

5.7.4.18 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)

5.7.4.19 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

5.7.4.20 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. **??**)

5.7.4.21 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. **??**)

5.7.4.22 f_random_detach()

```
void f_random_detach ( )
```

Detaches system random numeber genarator from myNanoEmbedded API.

See also

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

5.7.4.23 f_reverse()

```
int f_reverse (
             unsigned char * ,
             size_t  )
```

5.7.4.24 f_ripemd160()

```
uint8_t* f_ripemd160 (
             const uint8_t * ,
             size_t  )
```

5.7.4.25 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*

5.7.4.26 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

5.7.4.27 f_uncompress_elliptic_curve()

```
int f_uncompress_elliptic_curve (
            uint8_t * ,
            size_t ,
            size_t * ,
            mbedtls_ecp_group_id ,
            uint8_t * ,
            size_t  )
```

5.7.4.28 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.

This implementation is based on topic in `Definition 7.12` in MIT opencourseware (7.3 A Statistical Definition of Entropy - 2005)
Many thanks to **Professor Z. S. Spakovszky** for this amazing topic

Return values

0	On Success, otherwise Error

5.7.4.29 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

5.8 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 #include "mbedtls/ecdsa.h"
00017
00018 #ifdef __cplusplus
00019 extern "C" {
00020 #endif
00021
00022 #ifndef F_DOC_SKIP
00023
00024  #define F_LOG_MAX 8*256
00025  #define LICENSE \
00026 "MIT License\n\n\
00027 Copyright (c) 2019 Fábio Pereira da Silva\n\n\
00028 Permission is hereby granted, free of charge, to any person obtaining a copy\n\
00029 of this software and associated documentation files (the \"Software\"), to deal\n\
00030 in the Software without restriction, including without limitation the rights\n\
00031 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell\n\
00032 copies of the Software, and to permit persons to whom the Software is\n\
00033 furnished to do so, subject to the following conditions:\n\n\
00034 The above copyright notice and this permission notice shall be included in all\n\
00035 copies or substantial portions of the Software.\n\n\
00036 THE SOFTWARE IS PROVIDED \"AS IS\", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR\n\
00037 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,\n\
00038 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE\n\
00039 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER\n\
00040 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,\n\
00041 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE\n\
00042 SOFTWARE.\n\n\n"
00043
00044 #endif
00045
00046 #ifdef F_ESP32
00047
00048  #define F_WDT_MAX_ENTROPY_TIME 2*120
00049  #define F_WDT_PANIC true
00050  #define F_WDT_MIN_TIME 20//4
00051
00052 #endif
00053
00071 int f_verify_system_entropy(uint32_t, void *, size_t, int);
00072
00099 int f_pass_must_have_at_least(char *, size_t, size_t, size_t, int);
00100
00101 #ifndef F_DOC_SKIP
00102
00103 int f_verify_system_entropy_begin();
00104 void f_verify_system_entropy_finish();
00105 int f_file_exists(char *);
00106 int f_find_str(size_t *, char *, size_t, char *);
00107 int f_find_replace(char *, size_t *, size_t, char *, size_t, char *, char *);
00108 int f_is_integer(char *, size_t);
00109 int is_filled_with_value(uint8_t *, size_t, uint8_t);
00110
00111 #endif
```

```
00112
00113 //#define F_ENTROPY_TYPE_PARANOIC (uint32_t)1476682819
00118 #define F_ENTROPY_TYPE_PARANOIC (uint32_t)1477682819
00119
00120 //#define F_ENTROPY_TYPE_EXCELENT (uint32_t)1475885281
00125 #define F_ENTROPY_TYPE_EXCELENT (uint32_t)1476885281
00126
00127 //#define F_ENTROPY_TYPE_GOOD (uint32_t)1471531015
00132 #define F_ENTROPY_TYPE_GOOD (uint32_t)1472531015
00133
00134 //#define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1470001808
00139 #define F_ENTROPY_TYPE_NOT_ENOUGH (uint32_t)1471001808
00140
00141 //#define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1469703345
00146 #define F_ENTROPY_TYPE_NOT_RECOMENDED (uint32_t)1470003345
00147
00153 #define ENTROPY_BEGIN f_verify_system_entropy_begin();
00154
00160 #define ENTROPY_END f_verify_system_entropy_finish();
00161
00166 #define F_PASS_MUST_HAVE_AT_LEAST_NONE (int)0
00167
00172 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER (int)1
00173
00178 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL (int)2
00179
00184 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_CASE (int)4
00185
00190 #define F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_CASE (int)8
00191
00196 #define F_PASS_IS_TOO_LONG (int)256
00197
00202 #define F_PASS_IS_TOO_SHORT (int)512
00203
00208 #define F_PASS_IS_OUT_OVF (int)1024//768
00209
00210 #ifndef F_DOC_SKIP
00211
00212  #define F_PBKDF2_ITER_SZ 2*4096
00213
00214 typedef enum f_pbkdf2_err_t {
00215     F_PBKDF2_RESULT_OK=0,
00216     F_PBKDF2_ERR_CTX=95,
00217     F_PBKDF2_ERR_PKCS5,
00218     F_PBKDF2_ERR_INFO_SHA
00219 } f_pbkdf2_err;
00220
00221 typedef enum f_aes_err {
00222     F_AES_RESULT_OK=0,
00223     F_AES_ERR_ENCKEY=30,
00224     F_AES_ERR_DECKEY,
00225     F_AES_ERR_MALLOC,
00226     F_AES_UNKNOW_DIRECTION,
00227     F_ERR_ENC_DECRYPT_FAILED
00228 } f_aes_err;
00229
00230 typedef enum f_md_hmac_sha512_t {
00231    F_HMAC_SHA512_OK = 0,
00232    F_HMAC_SHA512_MALLOC = 304,
00233    F_HMAC_SHA512_ERR_INFO,
00234    F_HMAC_SHA512_ERR_SETUP,
00235    F_HMAC_SHA512_DIGEST_ERROR
00236 } f_md_hmac_sha512;
00238 typedef enum f_ecdsa_key_pair_err_t {
00239    F_ECDSA_KEY_PAIR_OK = 0,
00240    F_ECDSA_KEY_PAIR_NULL = 330,
00241    F_ECDSA_KEY_PAIR_MALLOC
00242 } f_ecdsa_key_pair_err;
00243
00244 typedef struct f_ecdsa_key_pair_t {
00245     size_t public_key_sz;
00246     size_t private_key_sz;
00247     mbedtls_ecdsa_context *ctx;
00248     mbedtls_ecp_group_id gid;
00249     unsigned char public_key[MBEDTLS_ECDSA_MAX_LEN];
00250     unsigned char private_key[MBEDTLS_ECDSA_MAX_LEN];
00251 } f_ecdsa_key_pair;
00252
00253 char *fhex2strv2(char *, const void *, size_t, int);
00254 //uint8_t *f_sha256_digest(uint8_t *, size_t);
00255 int f_sha256_digest(void **, int, uint8_t *, size_t);
00256 f_pbkdf2_err f_pbkdf2_hmac(unsigned char *, size_t, unsigned char *, size_t, uint8_t *);
00257 f_aes_err f_aes256cipher(uint8_t *, uint8_t *, void *, size_t, void *, int);
00258
00259 #endif
00260
00272 int f_passwd_comp_safe(char *, char *, size_t, size_t, size_t);
```

```
00273
00284 char *f_get_entropy_name(uint32_t);
00285
00300 uint32_t f_sel_to_entropy_level(int);
00301
00310 int f_str_to_hex(uint8_t *, char *);
00311
00312 #ifndef F_ESP32
00313
00318 typedef void (*rnd_fn)(void *, size_t);
00319
00327 void f_random_attach(rnd_fn);
00328
00337 void f_random(void *, size_t);
00338
00347 int get_console_passwd(char *, size_t);
00348
00353 #define F_GET_CH_MODE_NO_ECHO (int)(1<<16)
00354
00359 #define F_GET_CH_MODE_ANY_KEY (int)(1<<17)
00360
00376 int f_get_char_no_block(int);
00377
00378 #endif
00379
00390 int f_convert_to_long_int(unsigned long int *, char *, size_t);
00391
00392
00403 int f_convert_to_unsigned_int(unsigned int *, char *, size_t);
00404
00415 int f_convert_to_long_int0x(unsigned long int *, char *, size_t);
00416
00427 int f_convert_to_long_int0(unsigned long int *, char *, size_t);
00428
00442 int f_convert_to_long_int_std(unsigned long int *, char *, size_t);
00443
00451 void *f_is_random_attached();
00452
00459 void f_random_detach();
00460
00471 int f_convert_to_unsigned_int0x(unsigned int *val, char *value, size_t value_sz);
00472
00483 int f_convert_to_unsigned_int0(unsigned int *val, char *value, size_t value_sz);
00484
00498 int f_convert_to_unsigned_int_std(unsigned int *val, char *value, size_t value_sz);
00499
00509 int f_convert_to_double(double *, const char *);
00510
00521 uint32_t crc32_init(unsigned char *, size_t, uint32_t);
00522 //
00523 typedef int (*fn_det)(void *, unsigned char *, size_t);
00524 int f_reverse(unsigned char *, size_t);
00525 f_md_hmac_sha512 f_hmac_sha512(unsigned char *, const unsigned char *, size_t, const unsigned char *,
      size_t);
00526 int f_ecdsa_secret_key_valid(mbedtls_ecp_group_id, unsigned char *, size_t);
00527 int f_ecdsa_public_key_valid(mbedtls_ecp_group_id, unsigned char *, size_t);
00528 f_ecdsa_key_pair_err f_gen_ecdsa_key_pair(f_ecdsa_key_pair *, int, fn_det, void *);
00529 int f_uncompress_elliptic_curve(uint8_t *, size_t, size_t *, mbedtls_ecp_group_id, uint8_t *, size_t);
00530 uint8_t *f_ripemd160(const uint8_t *, size_t);
00531
00532 #ifdef __cplusplus
00533 }
00534 #endif
```

5.9 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"
```

5.9.1 Detailed Description

This header file is an implementation of Libsodium library.

Definition in file **sodium.h**.

5.10 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

__attribute__
    f_bitcoin.h, 23
    f_nano_crypto_util.h, 45

account
    f_block_transfer_t, 9
    f_nano_crypto_util.h, 71

balance
    f_block_transfer_t, 9
    f_nano_crypto_util.h, 71
body
    f_nano_crypto_util.h, 71
    f_nano_wallet_info_t, 16

chain_code
    f_bitcoin.h, 25
    f_bitcoin_serialize_t, 7
child_number
    f_bitcoin.h, 25
    f_bitcoin_serialize_t, 7
chksum
    f_bitcoin.h, 25
    f_bitcoin_serialize_t, 8
crc32_init
    f_util.h, 88

DEST_XRB
    f_nano_crypto_util.h, 31
desc
    f_nano_crypto_util.h, 72
    f_nano_wallet_info_t, 17
description
    f_nano_crypto_util.h, 72
    f_nano_crypto_wallet_t, 12

ENTROPY_BEGIN
    f_util.h, 84
ENTROPY_END
    f_util.h, 84

F_ADD_288
    f_add_bn_288_le.h, 19
F_BITCOIN_BUF_SZ
    f_bitcoin.h, 21
F_BITCOIN_P2PKH
    f_bitcoin.h, 21
F_BITCOIN_SEED_GENERATOR
    f_bitcoin.h, 21
F_BITCOIN_T2PKH
    f_bitcoin.h, 21
F_BITCOIN_WIF_MAINNET
    f_bitcoin.h, 21
F_BITCOIN_WIF_TESTNET
    f_bitcoin.h, 22
F_BRAIN_WALLET_BAD
    f_nano_crypto_util.h, 31
F_BRAIN_WALLET_GOOD
    f_nano_crypto_util.h, 31
F_BRAIN_WALLET_MAYBE_GOOD
    f_nano_crypto_util.h, 32
F_BRAIN_WALLET_NICE
    f_nano_crypto_util.h, 32
F_BRAIN_WALLET_PERFECT
    f_nano_crypto_util.h, 32
F_BRAIN_WALLET_POOR
    f_nano_crypto_util.h, 32
F_BRAIN_WALLET_STILL_WEAK
    f_nano_crypto_util.h, 33
F_BRAIN_WALLET_VERY_BAD
    f_nano_crypto_util.h, 33
F_BRAIN_WALLET_VERY_GOOD
    f_nano_crypto_util.h, 33
F_BRAIN_WALLET_VERY_POOR
    f_nano_crypto_util.h, 33
F_BRAIN_WALLET_VERY_WEAK
    f_nano_crypto_util.h, 34
F_BRAIN_WALLET_WEAK
    f_nano_crypto_util.h, 34
F_DEFAULT_THRESHOLD
    f_nano_crypto_util.h, 34
F_ENTROPY_TYPE_EXCELENT
    f_util.h, 85
F_ENTROPY_TYPE_GOOD
    f_util.h, 85
F_ENTROPY_TYPE_NOT_ENOUGH
    f_util.h, 85
F_ENTROPY_TYPE_NOT_RECOMENDED
    f_util.h, 85
F_ENTROPY_TYPE_PARANOIC
    f_util.h, 86
F_FILE_INFO_ERR
    f_nano_crypto_util.h, 39
F_GET_CH_MODE_ANY_KEY
    f_util.h, 86
F_GET_CH_MODE_NO_ECHO
    f_util.h, 86
F_IS_SIGNATURE_RAW_HEX_STRING
    f_nano_crypto_util.h, 34
F_MAX_BASE58_LENGTH

f_bitcoin.h, 22
F_MESSAGE_IS_HASH_STRING
f_nano_crypto_util.h, 35
F_NANO_CREATE_BLOCK_DYN_ERR
f_nano_crypto_util.h, 40
F_NANO_POW_MAX_THREAD
f_nano_crypto_util.h, 35
F_PASS_IS_OUT_OVF
f_util.h, 86
F_PASS_IS_TOO_LONG
f_util.h, 87
F_PASS_IS_TOO_SHORT
f_util.h, 87
F_PASS_MUST_HAVE_AT_LEAST_NONE
f_util.h, 87
F_PASS_MUST_HAVE_AT_LEAST_ONE_LOWER_↩
CASE
f_util.h, 87
F_PASS_MUST_HAVE_AT_LEAST_ONE_NUMBER
f_util.h, 87
F_PASS_MUST_HAVE_AT_LEAST_ONE_SYMBOL
f_util.h, 88
F_PASS_MUST_HAVE_AT_LEAST_ONE_UPPER_↩
CASE
f_util.h, 88
F_SIGNATURE_OUTPUT_NANO_PK
f_nano_crypto_util.h, 35
F_SIGNATURE_OUTPUT_RAW_PK
f_nano_crypto_util.h, 35
F_SIGNATURE_OUTPUT_STRING_PK
f_nano_crypto_util.h, 36
F_SIGNATURE_OUTPUT_XRB_PK
f_nano_crypto_util.h, 36
F_SIGNATURE_RAW
f_nano_crypto_util.h, 36
F_SIGNATURE_STRING
f_nano_crypto_util.h, 36
F_TOKEN
f_nano_crypto_util.h, 40
F_VERIFY_SIG_ASCII_HEX
f_nano_crypto_util.h, 37
F_VERIFY_SIG_NANO_WALLET
f_nano_crypto_util.h, 37
F_VERIFY_SIG_RAW_HEX
f_nano_crypto_util.h, 37
F_VERSION_BYTES_IDX_LEN
f_bitcoin.h, 22
F_VERSION_BYTES
f_bitcoin.h, 25
f_add_bn_288_le.h, 19, 20
F_ADD_288, 19
f_bip32_to_public_key_or_private_key
f_bitcoin.h, 23
f_bip39_to_nano_seed
f_nano_crypto_util.h, 45
f_bitcoin.h, 20, 27
__attribute__, 23
chain_code, 25
child_number, 25
chksum, 25
F_BITCOIN_BUF_SZ, 21
F_BITCOIN_P2PKH, 21
F_BITCOIN_SEED_GENERATOR, 21
F_BITCOIN_T2PKH, 21
F_BITCOIN_WIF_MAINNET, 21
F_BITCOIN_WIF_TESTNET, 22
F_MAX_BASE58_LENGTH, 22
F_VERSION_BYTES_IDX_LEN, 22
F_VERSION_BYTES, 25
f_bip32_to_public_key_or_private_key, 23
f_bitcoin_valid_bip32, 23
f_decode_b58_util, 23
f_encode_b58, 24
f_generate_master_key, 24
f_private_key_to_wif, 24
f_public_key_to_address, 24
f_uncompress_elliptic_curve, 24
f_wif_to_private_key, 25
finger_print, 26
MAINNET_PRIVATE, 22
MAINNET_PUBLIC, 22
master_node, 26
sk_or_pk_data, 26
TESTNET_PRIVATE, 22
TESTNET_PUBLIC, 23
version_bytes, 26
f_bitcoin_serialize_t, 7
chain_code, 7
child_number, 7
chksum, 8
finger_print, 8
master_node, 8
sk_or_pk_data, 8
version_bytes, 8
f_bitcoin_valid_bip32
f_bitcoin.h, 23
f_block_transfer_t, 9
account, 9
balance, 9
link, 9
preamble, 10
prefixes, 10
previous, 10
representative, 10
signature, 10
work, 11
f_cloud_crypto_wallet_nano_create_seed
f_nano_crypto_util.h, 45
f_convert_to_double
f_util.h, 89
f_convert_to_long_int
f_util.h, 89
f_convert_to_long_int0
f_util.h, 90
f_convert_to_long_int0x
f_util.h, 90

f_convert_to_long_int_std
f_util.h, 91
f_convert_to_unsigned_int
f_util.h, 91
f_convert_to_unsigned_int0
f_util.h, 92
f_convert_to_unsigned_int0x
f_util.h, 92
f_convert_to_unsigned_int_std
f_util.h, 93
f_decode_b58_util
f_bitcoin.h, 23
f_ecdsa_public_key_valid
f_util.h, 93
f_ecdsa_secret_key_valid
f_util.h, 94
f_encode_b58
f_bitcoin.h, 24
f_extract_seed_from_brainwallet
f_nano_crypto_util.h, 46
f_file_info_err_t, 11
f_nano_crypto_util.h, 41
f_gen_ecdsa_key_pair
f_util.h, 94
f_generate_master_key
f_bitcoin.h, 24
f_generate_nano_seed
f_nano_crypto_util.h, 47
f_generate_token
f_nano_crypto_util.h, 48
f_get_char_no_block
f_util.h, 94
f_get_dictionary_path
f_nano_crypto_util.h, 48
f_get_entropy_name
f_util.h, 95
f_get_nano_file_info
f_nano_crypto_util.h, 48
f_hmac_sha512
f_util.h, 95
f_is_random_attached
f_util.h, 95
f_is_valid_nano_seed_encrypted
f_nano_crypto_util.h, 49
f_nano_add_sub
f_nano_crypto_util.h, 49
f_nano_balance_to_str
f_nano_crypto_util.h, 50
f_nano_block_to_json
f_nano_crypto_util.h, 51
f_nano_create_block_dyn_err_t
f_nano_crypto_util.h, 42
f_nano_crypto_util.h, 27, 77
__attribute__, 45
account, 71
balance, 71
body, 71
DEST_XRB, 31
desc, 72
description, 72
F_BRAIN_WALLET_BAD, 31
F_BRAIN_WALLET_GOOD, 31
F_BRAIN_WALLET_MAYBE_GOOD, 32
F_BRAIN_WALLET_NICE, 32
F_BRAIN_WALLET_PERFECT, 32
F_BRAIN_WALLET_POOR, 32
F_BRAIN_WALLET_STILL_WEAK, 33
F_BRAIN_WALLET_VERY_BAD, 33
F_BRAIN_WALLET_VERY_GOOD, 33
F_BRAIN_WALLET_VERY_POOR, 33
F_BRAIN_WALLET_VERY_WEAK, 34
F_BRAIN_WALLET_WEAK, 34
F_DEFAULT_THRESHOLD, 34
F_FILE_INFO_ERR, 39
F_IS_SIGNATURE_RAW_HEX_STRING, 34
F_MESSAGE_IS_HASH_STRING, 35
F_NANO_CREATE_BLOCK_DYN_ERR, 40
F_NANO_POW_MAX_THREAD, 35
F_SIGNATURE_OUTPUT_NANO_PK, 35
F_SIGNATURE_OUTPUT_RAW_PK, 35
F_SIGNATURE_OUTPUT_STRING_PK, 36
F_SIGNATURE_OUTPUT_XRB_PK, 36
F_SIGNATURE_RAW, 36
F_SIGNATURE_STRING, 36
F_TOKEN, 40
F_VERIFY_SIG_ASCII_HEX, 37
F_VERIFY_SIG_NANO_WALLET, 37
F_VERIFY_SIG_RAW_HEX, 37
f_bip39_to_nano_seed, 45
f_cloud_crypto_wallet_nano_create_seed, 45
f_extract_seed_from_brainwallet, 46
f_file_info_err_t, 41
f_generate_nano_seed, 47
f_generate_token, 48
f_get_dictionary_path, 48
f_get_nano_file_info, 48
f_is_valid_nano_seed_encrypted, 49
f_nano_add_sub, 49
f_nano_balance_to_str, 50
f_nano_block_to_json, 51
f_nano_create_block_dyn_err_t, 42
f_nano_err, 40
f_nano_err_t, 42
f_nano_get_block_hash, 51
f_nano_get_p2pow_block_hash, 52
f_nano_is_valid_block, 52
f_nano_key_to_str, 52
f_nano_p2pow_to_JSON, 53
f_nano_parse_raw_str_to_raw128_t, 53
f_nano_parse_real_str_to_raw128_t, 54
f_nano_pow, 54
f_nano_raw_to_string, 55
f_nano_seed_to_bip39, 56
f_nano_sign_block, 56
f_nano_transaction_to_JSON, 57
f_nano_valid_nano_str_value, 57

f_nano_value_compare_value, 58
f_nano_verify_nano_funds, 59
f_parse_nano_seed_and_bip39_to_JSON, 60
f_read_seed, 60
f_seed_to_nano_wallet, 61
f_set_dictionary_path, 62
f_set_nano_file_info, 62
f_sign_data, 63
f_uint128_t, 40
f_verify_signed_data, 64
f_verify_token, 64
f_verify_work, 66
f_write_seed, 66
f_write_seed_err, 40
f_write_seed_err_t, 44
file_info_integrity, 72
from_multiplier, 67
hash_sk_unencrypted, 72
header, 72
is_nano_prefix, 68
is_null_hash, 68
iv, 73
last_used_wallet_number, 73
link, 73
MAX_STR_NANO_CHAR, 37
max_fee, 73
NANO_ENCRYPTED_SEED_FILE, 38
NANO_FILE_WALLETS_INFO, 38
NANO_PASSWD_MAX_LEN, 38
NANO_PREFIX, 38
NANO_PRIVATE_KEY_EXTENDED, 41
NANO_PRIVATE_KEY, 40
NANO_PUBLIC_KEY_EXTENDED, 41
NANO_PUBLIC_KEY, 41
NANO_SEED, 41
nano_base_32_2_hex, 68
nano_create_block_dynamic, 69
nano_hdr, 73
nanoseed_hash, 74
PUB_KEY_EXTENDED_MAX_LEN, 38
pk_to_wallet, 69
preamble, 74
prefixes, 74
previous, 74
REP_XRB, 39
representative, 74
reserved, 75
SENDER_XRB, 39
STR_NANO_SZ, 39
salt, 75
seed_block, 75
signature, 75
sk_encrypted, 75
sub_salt, 76
to_multiplier, 70
valid_nano_wallet, 70
valid_raw_balance, 71
ver, 76
version, 76
wallet_prefix, 76
wallet_representative, 76
work, 77
XRB_PREFIX, 39
f_nano_crypto_wallet_t, 11
description, 12
iv, 12
nano_hdr, 12
salt, 12
seed_block, 12
ver, 13
f_nano_encrypted_wallet_t, 13
hash_sk_unencrypted, 13
iv, 14
reserved, 14
sk_encrypted, 14
sub_salt, 14
f_nano_err
f_nano_crypto_util.h, 40
f_nano_err_t
f_nano_crypto_util.h, 42
f_nano_get_block_hash
f_nano_crypto_util.h, 51
f_nano_get_p2pow_block_hash
f_nano_crypto_util.h, 52
f_nano_is_valid_block
f_nano_crypto_util.h, 52
f_nano_key_to_str
f_nano_crypto_util.h, 52
f_nano_p2pow_to_JSON
f_nano_crypto_util.h, 53
f_nano_parse_raw_str_to_raw128_t
f_nano_crypto_util.h, 53
f_nano_parse_real_str_to_raw128_t
f_nano_crypto_util.h, 54
f_nano_pow
f_nano_crypto_util.h, 54
f_nano_raw_to_string
f_nano_crypto_util.h, 55
f_nano_seed_to_bip39
f_nano_crypto_util.h, 56
f_nano_sign_block
f_nano_crypto_util.h, 56
f_nano_transaction_to_JSON
f_nano_crypto_util.h, 57
f_nano_valid_nano_str_value
f_nano_crypto_util.h, 57
f_nano_value_compare_value
f_nano_crypto_util.h, 58
f_nano_verify_nano_funds
f_nano_crypto_util.h, 59
f_nano_wallet_info_bdy_t, 15
last_used_wallet_number, 15
max_fee, 15
reserved, 15
wallet_prefix, 15
wallet_representative, 16

f_nano_wallet_info_t, 16
body, 16
desc, 17
file_info_integrity, 17
header, 17
nanoseed_hash, 17
version, 17
f_parse_nano_seed_and_bip39_to_JSON
f_nano_crypto_util.h, 60
f_pass_must_have_at_least
f_util.h, 96
f_passwd_comp_safe
f_util.h, 97
f_private_key_to_wif
f_bitcoin.h, 24
f_public_key_to_address
f_bitcoin.h, 24
f_random
f_util.h, 97
f_random_attach
f_util.h, 98
f_random_detach
f_util.h, 98
f_read_seed
f_nano_crypto_util.h, 60
f_reverse
f_util.h, 98
f_ripemd160
f_util.h, 98
f_seed_to_nano_wallet
f_nano_crypto_util.h, 61
f_sel_to_entropy_level
f_util.h, 98
f_set_dictionary_path
f_nano_crypto_util.h, 62
f_set_nano_file_info
f_nano_crypto_util.h, 62
f_sign_data
f_nano_crypto_util.h, 63
f_str_to_hex
f_util.h, 99
f_uint128_t
f_nano_crypto_util.h, 40
f_uncompress_elliptic_curve
f_bitcoin.h, 24
f_util.h, 99
f_util.h, 83, 101
crc32_init, 88
ENTROPY_BEGIN, 84
ENTROPY_END, 84
F_ENTROPY_TYPE_EXCELENT, 85
F_ENTROPY_TYPE_GOOD, 85
F_ENTROPY_TYPE_NOT_ENOUGH, 85
F_ENTROPY_TYPE_NOT_RECOMENDED, 85
F_ENTROPY_TYPE_PARANOIC, 86
F_GET_CH_MODE_ANY_KEY, 86
F_GET_CH_MODE_NO_ECHO, 86
F_PASS_IS_OUT_OVF, 86
F_PASS_IS_TOO_LONG, 87
F_PASS_IS_TOO_SHORT, 87
F_PASS_MUST_HAVE_AT_LEAST_NONE, 87
F_PASS_MUST_HAVE_AT_LEAST_ONE_LO↩
WER_CASE, 87
F_PASS_MUST_HAVE_AT_LEAST_ONE_NU↩
MBER, 87
F_PASS_MUST_HAVE_AT_LEAST_ONE_SYM↩
BOL, 88
F_PASS_MUST_HAVE_AT_LEAST_ONE_UPP↩
ER_CASE, 88
f_convert_to_double, 89
f_convert_to_long_int, 89
f_convert_to_long_int0, 90
f_convert_to_long_int0x, 90
f_convert_to_long_int_std, 91
f_convert_to_unsigned_int, 91
f_convert_to_unsigned_int0, 92
f_convert_to_unsigned_int0x, 92
f_convert_to_unsigned_int_std, 93
f_ecdsa_public_key_valid, 93
f_ecdsa_secret_key_valid, 94
f_gen_ecdsa_key_pair, 94
f_get_char_no_block, 94
f_get_entropy_name, 95
f_hmac_sha512, 95
f_is_random_attached, 95
f_pass_must_have_at_least, 96
f_passwd_comp_safe, 97
f_random, 97
f_random_attach, 98
f_random_detach, 98
f_reverse, 98
f_ripemd160, 98
f_sel_to_entropy_level, 98
f_str_to_hex, 99
f_uncompress_elliptic_curve, 99
f_verify_system_entropy, 99
fn_det, 88
get_console_passwd, 100
rnd_fn, 88
f_verify_signed_data
f_nano_crypto_util.h, 64
f_verify_system_entropy
f_util.h, 99
f_verify_token
f_nano_crypto_util.h, 64
f_verify_work
f_nano_crypto_util.h, 66
f_wif_to_private_key
f_bitcoin.h, 25
f_write_seed
f_nano_crypto_util.h, 66
f_write_seed_err
f_nano_crypto_util.h, 40
f_write_seed_err_t
f_nano_crypto_util.h, 44
file_info_integrity

f_nano_crypto_util.h, 72
f_nano_wallet_info_t, 17
finger_print
f_bitcoin.h, 26
f_bitcoin_serialize_t, 8
fn_det
f_util.h, 88
from_multiplier
f_nano_crypto_util.h, 67

get_console_passwd
f_util.h, 100

hash_sk_unencrypted
f_nano_crypto_util.h, 72
f_nano_encrypted_wallet_t, 13
header
f_nano_crypto_util.h, 72
f_nano_wallet_info_t, 17

is_nano_prefix
f_nano_crypto_util.h, 68
is_null_hash
f_nano_crypto_util.h, 68
iv
f_nano_crypto_util.h, 73
f_nano_crypto_wallet_t, 12
f_nano_encrypted_wallet_t, 14

last_used_wallet_number
f_nano_crypto_util.h, 73
f_nano_wallet_info_bdy_t, 15
link
f_block_transfer_t, 9
f_nano_crypto_util.h, 73

MAINNET_PRIVATE
f_bitcoin.h, 22
MAINNET_PUBLIC
f_bitcoin.h, 22
MAX_STR_NANO_CHAR
f_nano_crypto_util.h, 37
master_node
f_bitcoin.h, 26
f_bitcoin_serialize_t, 8
max_fee
f_nano_crypto_util.h, 73
f_nano_wallet_info_bdy_t, 15

NANO_ENCRYPTED_SEED_FILE
f_nano_crypto_util.h, 38
NANO_FILE_WALLETS_INFO
f_nano_crypto_util.h, 38
NANO_PASSWD_MAX_LEN
f_nano_crypto_util.h, 38
NANO_PREFIX
f_nano_crypto_util.h, 38
NANO_PRIVATE_KEY_EXTENDED
f_nano_crypto_util.h, 41
NANO_PRIVATE_KEY
f_nano_crypto_util.h, 40
NANO_PUBLIC_KEY_EXTENDED
f_nano_crypto_util.h, 41
NANO_PUBLIC_KEY
f_nano_crypto_util.h, 41
NANO_SEED
f_nano_crypto_util.h, 41
nano_base_32_2_hex
f_nano_crypto_util.h, 68
nano_create_block_dynamic
f_nano_crypto_util.h, 69
nano_hdr
f_nano_crypto_util.h, 73
f_nano_crypto_wallet_t, 12
nanoseed_hash
f_nano_crypto_util.h, 74
f_nano_wallet_info_t, 17

PUB_KEY_EXTENDED_MAX_LEN
f_nano_crypto_util.h, 38
pk_to_wallet
f_nano_crypto_util.h, 69
preamble
f_block_transfer_t, 10
f_nano_crypto_util.h, 74
prefixes
f_block_transfer_t, 10
f_nano_crypto_util.h, 74
previous
f_block_transfer_t, 10
f_nano_crypto_util.h, 74

REP_XRB
f_nano_crypto_util.h, 39
representative
f_block_transfer_t, 10
f_nano_crypto_util.h, 74
reserved
f_nano_crypto_util.h, 75
f_nano_encrypted_wallet_t, 14
f_nano_wallet_info_bdy_t, 15
rnd_fn
f_util.h, 88

SENDER_XRB
f_nano_crypto_util.h, 39
STR_NANO_SZ
f_nano_crypto_util.h, 39
salt
f_nano_crypto_util.h, 75
f_nano_crypto_wallet_t, 12
seed_block
f_nano_crypto_util.h, 75
f_nano_crypto_wallet_t, 12
signature
f_block_transfer_t, 10
f_nano_crypto_util.h, 75
sk_encrypted
f_nano_crypto_util.h, 75

f_nano_encrypted_wallet_t, 14
sk_or_pk_data
f_bitcoin.h, 26
f_bitcoin_serialize_t, 8
sodium.h, 103, 105
sub_salt
f_nano_crypto_util.h, 76
f_nano_encrypted_wallet_t, 14

TESTNET_PRIVATE
f_bitcoin.h, 22
TESTNET_PUBLIC
f_bitcoin.h, 23
to_multiplier
f_nano_crypto_util.h, 70

valid_nano_wallet
f_nano_crypto_util.h, 70
valid_raw_balance
f_nano_crypto_util.h, 71
ver
f_nano_crypto_util.h, 76
f_nano_crypto_wallet_t, 13
version
f_nano_crypto_util.h, 76
f_nano_wallet_info_t, 17
version_bytes
f_bitcoin.h, 26
f_bitcoin_serialize_t, 8

wallet_prefix
f_nano_crypto_util.h, 76
f_nano_wallet_info_bdy_t, 15
wallet_representative
f_nano_crypto_util.h, 76
f_nano_wallet_info_bdy_t, 16
work
f_block_transfer_t, 11
f_nano_crypto_util.h, 77

XRB_PREFIX
f_nano_crypto_util.h, 39