libwally-core Documentation

Release 0.8.5

Jon Griffiths

Apr 12, 2022

Contents:

CHAPTER 1

Core Functions

int **wally_init**(uint32_t *flags*)
Initialize wally.

This function must be called once before threads are created by the application.

Parameters

- **flags** – Flags controlling what to initialize. Currently must be zero.

Returns See *Error Codes*

int **wally_cleanup**(uint32_t *flags*)
Free any internally allocated memory.

Parameters

- **flags** – Flags controlling what to clean up. Currently must be zero.

Returns See *Error Codes*

struct secp256k1_context_struct ***wally_get_secp_context**(void)
Fetch the wally internal secp256k1 context object.

By default, a single global context is created on demand. This behaviour can be overriden by providing a custom context fetching function when calling *wally_set_operations*.

struct secp256k1_context_struct ***wally_get_new_secp_context**(void)
Create a new wally-suitable secp256k1 context object.

The created context is initialised to be usable by all wally functions.

void **wally_secp_context_free**(struct secp256k1_context_struct **ctx*)
Free a secp256k1 context object created by *wally_get_new_secp_context*.

This function must only be called on context objects returned from *wally_get_new_secp_context*, it should not be called on the default context returned from *wally_get_secp_context*.

int **wally_bzero**(void **bytes*, size_t *bytes_len*)
Securely wipe memory.

Parameters

- **bytes** – Memory to wipe
- **bytes_len** – Size of `bytes` in bytes.

Returns See *Error Codes*

int **wally_free_string**(char **str*)
Securely wipe and then free a string allocated by the library.

Parameters

- **str** – String to free (must be NUL terminated UTF-8).

Returns See *Error Codes*

int **wally_secp_randomize**(const unsigned char **bytes*, size_t *bytes_len*)
Provide entropy to randomize the libraries internal libsecp256k1 context.

Random data is used in libsecp256k1 to blind the data being processed, making side channel attacks more difficult. By default, Wally uses a single internal context for secp functions that is not initially randomized.

The caller should call this function before using any functions that rely on libsecp256k1 (i.e. Anything using public/private keys). If the caller has overriden the library's default libsecp context fetching using *wally_set_operations*, then it may be necessary to call this function before calling wally functions in each thread created by the caller.

If wally is used in its default configuration, this function should either be called before threads are created or access to wally functions wrapped in an application level mutex.

Parameters

- **bytes** – Entropy to use.
- **bytes_len** – Size of `bytes` in bytes. Must be `WALLY_SECP_RANDOMIZE_LEN`.

Returns See *Error Codes*

int **wally_hex_verify**(const char **hex*)
Verify that a hexadecimal string is valid.

Parameters

- **hex** – String to verify.

Returns See *Error Codes*

int **wally_hex_n_verify**(const char **hex*, size_t *hex_len*)
Verify that a known-length hexadecimal string is valid.

See *wally_hex_verify*. :return: See *Error Codes*

int **wally_hex_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, char ***output*)
Convert bytes to a (lower-case) hexadecimal string.

Parameters

- **bytes** – Bytes to convert.
- **bytes_len** – Size of `bytes` in bytes.
- **output** – Destination for the resulting hexadecimal string. The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

int **wally_hex_to_bytes**(const char **hex*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Convert a hexadecimal string to bytes.

Parameters

- **hex** – String to convert.
- **bytes_out** – Where to store the resulting bytes.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_hex_n_to_bytes**(const char **hex*, size_t *hex_len*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Convert a known-length hexadecimal string to bytes.

See *wally_hex_to_bytes*. :return: See *Variable Length Output Buffers*

int **wally_base58_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, char ***output*)
Create a base 58 encoded string representing binary data.

Parameters

- **bytes** – Binary data to convert.
- **bytes_len** – The length of `bytes` in bytes.
- **flags** – Pass `BASE58_FLAG_CHECKSUM` if `bytes` should have a checksum calculated and appended before converting to base 58.
- **output** – Destination for the base 58 encoded string representing `bytes`. The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

int **wally_base58_to_bytes**(const char **str_in*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Decode a base 58 encoded string back into into binary data.

Parameters

- **str_in** – Base 58 encoded string to decode.
- **flags** – Pass `BASE58_FLAG_CHECKSUM` if `bytes_out` should have a checksum validated and removed before returning. In this case, `len` must contain an extra `BASE58_CHECKSUM_LEN` bytes to calculate the checksum into. The returned length will not include the checksum.
- **bytes_out** – Destination for converted binary data.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the length of the decoded bytes.

Returns See *Variable Length Output Buffers*

int **wally_base58_n_to_bytes**(const char **str_in*, size_t *str_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Decode a known-length base 58 encoded string back into into binary data.

See *wally_base58_to_bytes*. :return: See *Variable Length Output Buffers*

int **wally_base58_get_length**(const char **str_in*, size_t **written*)
Return the length of a base 58 encoded string once decoded into bytes.

Returns the exact number of bytes that would be required to store `str_in` as decoded binary, including any embedded checksum. If the string contains invalid characters then WALLY_EINVAL is returned. Note that no checksum validation takes place.

In the worst case (an all zero buffer, represented by a string of '1' characters), this function will return strlen(`str_in`). You can therefore safely use the length of `str_in` as a buffer size to avoid calling this function in most cases.

Parameters

- **str_in** – Base 58 encoded string to find the length of.
- **written** – Destination for the length of the decoded bytes.

Returns See *Error Codes*

int **wally_base58_n_get_length**(const char **str_in*, size_t *str_len*, size_t **written*)
Return the length of a known-length base 58 encoded string once decoded into bytes.

See *wally_base58_get_length*. :return: See *Error Codes*

int **wally_base64_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, char ***output*)
Create a base64 encoded string representing binary data.

Parameters

- **bytes** – Binary data to convert.
- **bytes_len** – The length of `bytes` in bytes.
- **flags** – Must be 0.
- **output** – Destination for the base64 encoded string representing `bytes`. The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

int **wally_base64_to_bytes**(const char **str_in*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Decode a base64 encoded string back into into binary data.

Parameters

- **str_in** – Base64 encoded string to decode.
- **flags** – Must be 0.
- **bytes_out** – Destination for converted binary data.
- **len** – The length of `bytes_out` in bytes. See `wally_base64_get_maximum_length`.
- **written** – Destination for the length of the decoded bytes.

Returns See *Variable Length Output Buffers*

int **wally_base64_get_maximum_length**(const char **str_in*, uint32_t *flags*, size_t **written*)
Return the maximum length of a base64 encoded string once decoded into bytes.

Since base64 strings may contain line breaks and padding, it is not possible to compute their decoded length without fully decoding them. This function cheaply calculates the maximum possible decoded length, which can be used to allocate a buffer for `wally_base64_to_bytes`. In most cases the decoded data will be shorter than the value returned.

Parameters

- **str_in** – Base64 encoded string to find the length of.
- **flags** – Must be 0.
- **written** – Destination for the maximum length of the decoded bytes.

Returns See *Error Codes*

int **wally_get_operations**(struct wally_operations **output*)
Fetch the current overridable operations used by wally.

Parameters

- **output** – Destination for the overridable operations.

Returns See *Error Codes*

int **wally_set_operations**(const struct wally_operations **ops*)
Set the current overridable operations used by wally.

Parameters

- **ops** – The overridable operations to set.

Note: Any NULL members in the passed structure are ignored.

Returns See *Error Codes*

int **wally_is_elements_build**(size_t **written*)
Determine if the library was built with elements support.

Parameters

- **written** – 1 if the library supports elements, otherwise 0.

Returns See *Error Codes*

CHAPTER 2

Crypto Functions

int **wally_scrypt**(const unsigned char **pass*, size_t *pass_len*, const unsigned char **salt*, size_t *salt_len*, uint32_t *cost*, uint32_t *block_size*, uint32_t *parallelism*, unsigned char **bytes_out*, size_t *len*)

Derive a pseudorandom key from inputs using an expensive application of HMAC SHA-256.

Parameters

- **pass** – Password to derive from.
- **pass_len** – Length of `pass` in bytes.
- **salt** – Salt to derive from.
- **salt_len** – Length of `salt` in bytes.
- **cost** – The cost of the function. The larger this number, the longer the key will take to derive.
- **block_size** – The size of memory blocks required.
- **parallelism** – Parallelism factor.
- **bytes_out** – Destination for the derived pseudorandom key.
- **len** – The length of `bytes_out` in bytes.

Returns See *Error Codes*

int **wally_aes**(const unsigned char **key*, size_t *key_len*, const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Encrypt/decrypt data using AES (ECB mode, no padding).

Parameters

- **key** – Key material for initialisation.
- **key_len** – Length of `key` in bytes. Must be an **AES_KEY_LEN_** constant.
- **bytes** – Bytes to encrypt/decrypt.
- **bytes_len** – Length of `bytes` in bytes. Must be a multiple of `AES_BLOCK_LEN`.

- **flags** – AES_FLAG_ constants indicating the desired behavior.
- **bytes_out** – Destination for the encrypted/decrypted data.
- **len** – The length of `bytes_out` in bytes. Must be a multiple of `AES_BLOCK_LEN`.

Returns See *Error Codes*

int **wally_aes_cbc**(const unsigned char **key*, size_t *key_len*, const unsigned char **iv*, size_t *iv_len*, const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Encrypt/decrypt data using AES (CBC mode, PKCS#7 padding).

Parameters

- **key** – Key material for initialisation.
- **key_len** – Length of `key` in bytes. Must be an AES_KEY_LEN_ constant.
- **iv** – Initialisation vector.
- **iv_len** – Length of `iv` in bytes. Must be `AES_BLOCK_LEN`.
- **bytes** – Bytes to encrypt/decrypt.
- **bytes_len** – Length of `bytes` in bytes. Must be a multiple of `AES_BLOCK_LEN`.
- **flags** – AES_FLAG_ constants indicating the desired behavior.
- **bytes_out** – Destination for the encrypted/decrypted data.
- **len** – The length of `bytes_out` in bytes. Must be a multiple of `AES_BLOCK_LEN`.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_sha256**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

SHA-256(m)

Parameters

- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting hash.
- **len** – The length of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_sha256_midstate**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

SHA-256(m) midstate

Parameters

- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting hash.
- **len** – The length of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_sha256d**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

SHA-256(SHA-256(m)) (double SHA-256).

Parameters

- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting hash.
- **len** – The length of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_sha512**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)
SHA-512(m).

Parameters

- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting hash.
- **len** – The length of `bytes_out` in bytes. Must be `SHA512_LEN`.

Returns See *Error Codes*

int **wally_ripemd160**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)
RIPEMD-160(m).

Parameters

- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting hash.
- **len** – The length of `bytes_out` in bytes. Must be `RIPEMD160_LEN`.

Returns See *Error Codes*

int **wally_hash160**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)
RIPEMD-160(SHA-256(m)).

Parameters

- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting hash.
- **len** – The length of `bytes_out` in bytes. Must be `HASH160_LEN`.

Returns See *Error Codes*

int **wally_hmac_sha256**(const unsigned char **key*, size_t *key_len*, const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)
Compute an HMAC using SHA-256.

Parameters

- **key** – The key for the hash.
- **key_len** – The length of `key` in bytes.
- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.

- **bytes_out** – Destination for the resulting HMAC.
- **len** – The length of `bytes_out` in bytes. Must be `HMAC_SHA256_LEN`.

Returns See *Error Codes*

int **wally_hmac_sha512**(const unsigned char **key*, size_t *key_len*, const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

Compute an HMAC using SHA-512.

Parameters

- **key** – The key for the hash.
- **key_len** – The length of `key` in bytes.
- **bytes** – The message to hash.
- **bytes_len** – The length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting HMAC.
- **len** – The length of `bytes_out` in bytes. Must be `HMAC_SHA512_LEN`.

Returns See *Error Codes*

int **wally_pbkdf2_hmac_sha256**(const unsigned char **pass*, size_t *pass_len*, const unsigned char **salt*, size_t *salt_len*, uint32_t *flags*, uint32_t *cost*, unsigned char **bytes_out*, size_t *len*)

Derive a pseudorandom key from inputs using HMAC SHA-256.

Parameters

- **pass** – Password to derive from.
- **pass_len** – Length of `pass` in bytes.
- **salt** – Salt to derive from.
- **salt_len** – Length of `salt` in bytes.
- **flags** – Reserved, must be 0.
- **cost** – The cost of the function. The larger this number, the longer the key will take to derive.
- **bytes_out** – Destination for the derived pseudorandom key.
- **len** – The length of `bytes_out` in bytes. This must be a multiple of `PBKDF2_HMAC_SHA256_LEN`.

Returns See *Error Codes*

int **wally_pbkdf2_hmac_sha512**(const unsigned char **pass*, size_t *pass_len*, const unsigned char **salt*, size_t *salt_len*, uint32_t *flags*, uint32_t *cost*, unsigned char **bytes_out*, size_t *len*)

Derive a pseudorandom key from inputs using HMAC SHA-512.

Parameters

- **pass** – Password to derive from.
- **pass_len** – Length of `pass` in bytes.
- **salt** – Salt to derive from.
- **salt_len** – Length of `salt` in bytes.
- **flags** – Reserved, must be 0.

- **cost** – The cost of the function. The larger this number, the longer the key will take to derive.
- **bytes_out** – Destination for the derived pseudorandom key.
- **len** – The length of `bytes_out` in bytes. This must be a multiple of `PBKDF2_HMAC_SHA512_LEN`.

Returns See *Error Codes*

int **wally_ec_private_key_verify**(const unsigned char **priv_key*, size_t *priv_key_len*)
Verify that a private key is valid.

Parameters

- **priv_key** – The private key to validate.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.

Returns See *Error Codes*

int **wally_ec_public_key_verify**(const unsigned char **pub_key*, size_t *pub_key_len*)
Verify that a public key is valid.

Parameters

- **pub_key** – The public key to validate.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN` or `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`.

Returns See *Error Codes*

int **wally_ec_public_key_from_private_key**(const unsigned char **priv_key*, size_t *priv_key_len*, unsigned char **bytes_out*, size_t *len*)
Create a public key from a private key.

Parameters

- **priv_key** – The private key to create a public key from.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes_out** – Destination for the resulting public key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PUBLIC_KEY_LEN`.

Returns See *Error Codes*

int **wally_ec_public_key_decompress**(const unsigned char **pub_key*, size_t *pub_key_len*, unsigned char **bytes_out*, size_t *len*)
Create an uncompressed public key from a compressed public key.

Parameters

- **pub_key** – The public key to decompress.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **bytes_out** – Destination for the resulting public key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`.

Returns See *Error Codes*

int **wally_ec_public_key_negate**(const unsigned char **pub_key*, size_t *pub_key_len*, unsigned char **bytes_out*, size_t *len*)

Negates a public key.

Parameters

- **pub_key** – The public key to negate.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **bytes_out** – Destination for the resulting public key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PUBLIC_KEY_LEN`.

Returns See *Error Codes*

int **wally_ec_sig_from_bytes**(const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Sign a message hash with a private key, producing a compact signature.

Parameters

- **priv_key** – The private key to sign with.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes** – The message hash to sign.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **flags** – **EC_FLAG_** flag values indicating desired behavior.
- **bytes_out** – Destination for the resulting compact signature.
- **len** – The length of `bytes_out` in bytes. Must be `EC_SIGNATURE_LEN` if EC_FLAG_RECOVERABLE is not set, otherwise must be `EC_SIGNATURE_RECOVERABLE_LEN`.

Returns See *Error Codes*

int **wally_ec_sig_normalize**(const unsigned char **sig*, size_t *sig_len*, unsigned char **bytes_out*, size_t *len*)

Convert a signature to low-s form.

Parameters

- **sig** – The compact signature to convert.
- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_LEN`.
- **bytes_out** – Destination for the resulting low-s signature.
- **len** – The length of `bytes_out` in bytes. Must be `EC_SIGNATURE_LEN`.

Returns See *Error Codes*

int **wally_ec_sig_to_der**(const unsigned char **sig*, size_t *sig_len*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Convert a compact signature to DER encoding.

Parameters

- **sig** – The compact signature to convert.
- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_LEN`.
- **bytes_out** – Destination for the resulting DER encoded signature.

- **len** – The length of `bytes_out` in bytes. Must be `EC_SIGNATURE_DER_MAX_LEN`.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_ec_sig_from_der**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

Convert a DER encoded signature to a compact signature.

Parameters

- **bytes** – The DER encoded signature to convert.
- **bytes_len** – The length of `sig` in bytes.
- **bytes_out** – Destination for the resulting compact signature.
- **len** – The length of `bytes_out` in bytes. Must be `EC_SIGNATURE_LEN`.

Returns See *Error Codes*

int **wally_ec_sig_verify**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, const unsigned char **sig*, size_t *sig_len*)

Verify a signed message hash.

Parameters

- **pub_key** – The public key to verify with.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **bytes** – The message hash to verify.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **flags** – **EC_FLAG_** flag values indicating desired behavior.
- **sig** – The compact signature of the message in `bytes`.
- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_LEN`.

Returns See *Error Codes*

int **wally_ec_sig_to_public_key**(const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **sig*, size_t *sig_len*, unsigned char **bytes_out*, size_t *len*)

Recover compressed public key from a recoverable signature.

Parameters

- **bytes** – The message hash signed.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **sig** – The recoverable compact signature of the message in `bytes`.
- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_RECOVERABLE_LEN`.
- **bytes_out** – Destination for recovered public key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PUBLIC_KEY_LEN`.

Note: The successful recovery of the public key guarantees the correctness of the signature.

Returns See *Error Codes*

int **wally_format_bitcoin_message**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Format a message for use as a bitcoin signed message.

Parameters

- **bytes** – The message string to sign.
- **bytes_len** – The length of `bytes` in bytes. Must be less than or equal to BITCOIN_MESSAGE_MAX_LEN.
- **flags** – **BITCOIN_MESSAGE_FLAG_** flags indicating the desired output. if BITCOIN_MESSAGE_FLAG_HASH is passed, the double SHA256 hash of the message is placed in `bytes_out` instead of the formatted message. In this case `len` must be at least `SHA256_LEN`.
- **bytes_out** – Destination for the formatted message or message hash.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_ecdh**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **priv_key*, size_t *priv_key_len*, unsigned char **bytes_out*, size_t *len*)

Compute an EC Diffie-Hellman secret in constant time.

Parameters

- **pub_key** – The public key.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **priv_key** – The private key.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes_out** – Destination for the shared secret.
- **len** – The length of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_s2c_sig_from_bytes**(const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **s2c_data*, size_t *s2c_data_len*, uint32_t *flags*, unsigned char **s2c_opening_out*, size_t *s2c_opening_out_len*, unsigned char **bytes_out*, size_t *len*)

Sign a message hash with a private key, producing a compact signature which commits to additional data using sign-to-contract (s2c).

Parameters

- **priv_key** – The private key to sign with.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes** – The message hash to sign.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **s2c_data** – The data to commit to.
- **s2c_data_len** – The length of `s2c_data` in bytes. Must be `WALLY_S2C_DATA_LEN`.

- **flags** – Must be EC_FLAG_ECDSA.
- **s2c_opening_out** – Destination for the resulting opening information.
- **s2c_opening_out_len** – The length of s2c_opening_out in bytes. Must be WALLY_S2C_OPENING_LEN.
- **bytes_out** – Destination for the resulting compact signature.
- **len** – The length of bytes_out in bytes. Must be EC_SIGNATURE_LEN.

Returns See *Error Codes*

int **wally_s2c_commitment_verify**(const unsigned char *sig, size_t *sig_len*, const unsigned char *s2c_data, size_t *s2c_data_len*, const unsigned char *s2c_opening, size_t *s2c_opening_len*, uint32_t *flags*)

Verify a sign-to-contract (s2c) commitment.

Parameters

- **sig** – The compact signature.
- **sig_len** – The length of sig in bytes. Must be EC_SIGNATURE_LEN.
- **s2c_data** – The data that was committed to.
- **s2c_data_len** – The length of s2c_data in bytes. Must be WALLY_S2C_DATA_LEN.
- **s2c_opening** – The opening information produced during signing.
- **s2c_opening_len** – The length of s2c_opening in bytes. Must be WALLY_S2C_OPENING_LEN.
- **flags** – Must be EC_FLAG_ECDSA.

Returns See *Error Codes*

CHAPTER 3

Address Functions

int **wally_addr_segwit_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, const char **addr_family*, uint32_t *flags*, char ***output*)

Create a segwit native address from a v0 or later witness program.

Parameters

- **bytes** – Witness program bytes, including the version and data push opcode.
- **bytes_len** – Length of `bytes` in bytes. Must be `HASH160_LEN` or `SHA256_LEN` for v0 witness programs.
- **addr_family** – Address family to generate, e.g. "bc" or "tb".
- **flags** – For future use. Must be 0.
- **output** – Destination for the resulting segwit native address string.

Returns See *Error Codes*

int **wally_addr_segwit_to_bytes**(const char **addr*, const char **addr_family*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Get a scriptPubKey containing the witness program from a segwit native address.

Parameters

- **addr** – Address to fetch the witness program from.
- **addr_family** – Address family to generate, e.g. "bc" or "tb".
- **flags** – For future use. Must be 0.
- **bytes_out** – Destination for the resulting scriptPubKey (including the version and data push opcode)
- **len** – Length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_addr_segwit_n_to_bytes**(const char **addr*, size_t *addr_len*, const char **addr_family*, size_t *addr_family_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Get a scriptPubKey containing the witness program from a known-length segwit native address.

See *wally_addr_segwit_to_bytes*. :return: See *Variable Length Output Buffers*

int **wally_addr_segwit_get_version**(const char **addr*, const char **addr_family*, uint32_t *flags*, size_t **written*)

Get the segwit version of a segwit native address.

Parameters

- **addr** – Address to fetch the witness program from.
- **addr_family** – Address family to generate, e.g. "bc" or "tb".
- **flags** – For future use. Must be 0.
- **written** – Destination for the segwit version from 0 to 16 inclusive.

Returns See *Error Codes*

int **wally_addr_segwit_n_get_version**(const char **addr*, size_t *addr_len*, const char **addr_family*, size_t *addr_family_len*, uint32_t *flags*, size_t **written*)

Get the segwit version of a known-length segwit native address.

See *wally_addr_segwit_get_version*. :return: See *Error Codes*

int **wally_address_to_scriptpubkey**(const char **addr*, uint32_t *network*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Infer a scriptPubKey from an address.

Parameters

- **addr** – Base58 encoded address to infer the scriptPubKey from. For confidential Liquid addresses first call *wally_confidential_addr_to_addr()*
- **network** – One of `WALLY_NETWORK_BITCOIN_MAINNET`, `WALLY_NETWORK_BITCOIN_TESTNET`, `WALLY_NETWORK_LIQUID`, `WALLY_NETWORK_LIQUID_REGTEST`.
- **bytes_out** – Destination for the resulting scriptPubKey
- **len** – Length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_scriptpubkey_to_address**(const unsigned char **scriptpubkey*, size_t *scriptpubkey_len*, uint32_t *network*, char ***output*)

Infer address from a scriptPubKey. For SegWit addresses, use *wally_addr_segwit_from_bytes* instead. To find out if an address is SegWit, use *wally_scriptpubkey_get_type*.

Parameters

- **scriptpubkey** – scriptPubKey bytes.
- **scriptpubkey_len** – Length of `scriptpubkey` in bytes.
- **network** – One of `WALLY_NETWORK_BITCOIN_MAINNET`, `WALLY_NETWORK_BITCOIN_TESTNET`, `WALLY_NETWORK_LIQUID`, `WALLY_NETWORK_LIQUID_REGTEST`.
- **output** – Destination for the resulting Base58 encoded address string.

Returns See *Error Codes*

int **wally_wif_from_bytes**(const unsigned char **priv_key*, size_t *priv_key_len*, uint32_t *prefix*, uint32_t *flags*, char ***output*)

Convert a private key to Wallet Import Format.

Parameters

- **priv_key** – Private key bytes.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **prefix** – Expected prefix byte, e.g. `WALLY_ADDRESS_VERSION_WIF_MAINNET` or `WALLY_ADDRESS_VERSION_WIF_TESTNET`.
- **flags** – Pass `WALLY_WIF_FLAG_COMPRESSED` if the corresponding pubkey is compressed, otherwise `WALLY_WIF_FLAG_UNCOMPRESSED`.
- **output** – Destination for the resulting Wallet Import Format string.

Returns See *Error Codes*

int **wally_wif_to_bytes**(const char **wif*, uint32_t *prefix*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Convert a Wallet Import Format string to a private key.

Parameters

- **wif** – Private key in Wallet Import Format.
- **prefix** – Prefix byte to use, e.g. `WALLY_ADDRESS_VERSION_WIF_MAINNET` or `WALLY_ADDRESS_VERSION_WIF_TESTNET`.
- **flags** – Pass `WALLY_WIF_FLAG_COMPRESSED` if the corresponding pubkey is compressed, otherwise `WALLY_WIF_FLAG_UNCOMPRESSED`.
- **bytes_out** – Destination for the private key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PRIVATE_KEY_LEN`.

Returns See *Error Codes*

int **wally_wif_is_uncompressed**(const char **wif*, size_t **written*)

Determine if a private key in Wallet Import Format corresponds to an uncompressed public key.

Parameters

- **wif** – Private key in Wallet Import Format to check.
- **written** – 1 if the corresponding public key is uncompressed, 0 if compressed.

Returns See *Error Codes*

int **wally_wif_to_public_key**(const char **wif*, uint32_t *prefix*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a public key corresponding to a private key in Wallet Import Format.

Parameters

- **wif** – Private key in Wallet Import Format.
- **prefix** – Prefix byte to use, e.g. 0x80, 0xef.
- **bytes_out** – Destination for the resulting public key.
- **len** – The length of `bytes_out`.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_bip32_key_to_address**(const struct ext_key **hdkey*, uint32_t *flags*, uint32_t *version*, char ***output*)

Create a legacy or wrapped SegWit address corresponding to a BIP32 key.

Parameters

- **hdkey** – The extended key to use.
- **flags** – WALLY_ADDRESS_TYPE_P2PKH for a legacy address, WALLY_ADDRESS_TYPE_P2SH_P2WPKH for P2SH-wrapped SegWit.
- **version** – Version byte to generate address, e.g. with Bitcoin: WALLY_ADDRESS_VERSION_P2PKH_MAINNET, WALLY_ADDRESS_VERSION_P2PKH_TESTNET, WALLY_ADDRESS_VERSION_P2SH_MAINNET and WALLY_ADDRESS_VERSION_P2SH_TESTNET.
- **output** – Destination for the resulting address string.

Returns See *Error Codes*

int **wally_bip32_key_to_addr_segwit**(const struct ext_key **hdkey*, const char **addr_family*, uint32_t *flags*, char ***output*)

Create a native SegWit address corresponding to a BIP32 key.

Parameters

- **hdkey** – The extended key to use.
- **addr_family** – Address family to generate, e.g. "bc" or "tb".
- **flags** – For future use. Must be 0.
- **output** – Destination for the resulting segwit native address string.

Returns See *Error Codes*

int **wally_wif_to_address**(const char **wif*, uint32_t *prefix*, uint32_t *version*, char ***output*)

Create a P2PKH address corresponding to a private key in Wallet Import Format.

Parameters

- **wif** – Private key in Wallet Import Format.
- **prefix** – Prefix byte to use, e.g. 0x80, 0xef.
- **version** – Version byte to generate address, e.g. WALLY_ADDRESS_VERSION_P2PKH_MAINNET, WALLY_ADDRESS_VERSION_P2PKH_TESTNET.
- **output** – Destination for the resulting address string.

Returns See *Error Codes*

int **wally_confidential_addr_to_addr**(const char **address*, uint32_t *prefix*, char ***output*)

Extract the address from a confidential address.

Parameters

- **address** – The base58 encoded confidential address to extract the address from.
- **prefix** – The confidential address prefix byte, e.g. WALLY_CA_PREFIX_LIQUID.
- **output** – Destination for the resulting address string.

Returns See *Error Codes*

int **wally_confidential_addr_to_ec_public_key**(const char **address*, uint32_t *prefix*, unsigned char **bytes_out*, size_t *len*)

Extract the blinding public key from a confidential address.

Parameters

- **address** – The base58 encoded confidential address to extract the public key from.
- **prefix** – The confidential address prefix byte, e.g. `WALLY_CA_PREFIX_LIQUID`.
- **bytes_out** – Destination for the public key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PUBLIC_KEY_LEN`.

Returns See *Error Codes*

int **wally_confidential_addr_from_addr**(const char **address*, uint32_t *prefix*, const unsigned char **pub_key*, size_t *pub_key_len*, char ***output*)

Create a confidential address from an address and blinding public key.

Parameters

- **address** – The base58 encoded address to make confidential.
- **prefix** – The confidential address prefix byte, e.g. `WALLY_CA_PREFIX_LIQUID`.
- **pub_key** – The blinding public key to associate with `address`.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **output** – Destination for the resulting address string.

Returns See *Error Codes*

int **wally_confidential_addr_to_addr_segwit**(const char **address*, const char **confidential_addr_family*, const char **addr_family*, char ***output*)

Extract the segwit native address from a confidential address.

Parameters

- **address** – The blech32 encoded confidential address to extract the address from.
- **confidential_addr_family** – The confidential address family of `address`.
- **addr_family** – The address family to generate.
- **output** – Destination for the resulting address string. The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

int **wally_confidential_addr_segwit_to_ec_public_key**(const char **address*, const char **confidential_addr_family*, unsigned char **bytes_out*, size_t *len*)

Extract the blinding public key from a segwit confidential address.

Parameters

- **address** – The blech32 encoded confidential address to extract the public key from.
- **confidential_addr_family** – The confidential address family of `address`.
- **bytes_out** – Destination for the public key.
- **len** – The length of `bytes_out` in bytes. Must be `EC_PUBLIC_KEY_LEN`.

Returns See *Error Codes*

int **wally_confidential_addr_from_addr_segwit**(const char **address*, const char **addr_family*, const char **confidential_addr_family*, const unsigned char **pub_key*, size_t *pub_key_len*, char ***output*)

Create a confidential address from a segwit native address and blinding public key.

Parameters

- **address** – The bech32 encoded address to make confidential.
- **addr_family** – The address family to generate.
- **confidential_addr_family** – The confidential address family to generate.
- **pub_key** – The blinding public key to associate with `address`.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **output** – Destination for the resulting address string. The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

CHAPTER 4

Bip32 Functions

int **bip32_key_free**(const struct ext_key **hdkey*)

Free a key allocated by *bip32_key_from_seed_alloc*, *bip32_key_from_seed_custom* or *bip32_key_unserialize_alloc*.

Parameters

- **hdkey** – Key to free.

Returns See *Error Codes*

int **bip32_key_init**(uint32_t *version*, uint32_t *depth*, uint32_t *child_num*, const unsigned char **chain_code*, size_t *chain_code_len*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **hash160*, size_t *hash160_len*, const unsigned char **parent160*, size_t *parent160_len*, struct ext_key **output*)

Initialize a key. :return: See *Error Codes*

int **bip32_key_init_alloc**(uint32_t *version*, uint32_t *depth*, uint32_t *child_num*, const unsigned char **chain_code*, size_t *chain_code_len*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **hash160*, size_t *hash160_len*, const unsigned char **parent160*, size_t *parent160_len*, struct ext_key ***output*)

As per *bip32_key_init*, but allocates the key. :return: See *Error Codes*

int **bip32_key_from_seed_custom**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *version*, const unsigned char **hmac_key*, size_t *hmac_key_len*, uint32_t *flags*, struct ext_key **output*)

Create a new master extended key from entropy.

This creates a new master key, i.e. the root of a new HD tree. The entropy passed in may produce an invalid key. If this happens, WALLY_ERROR will be returned and the caller should retry with new entropy.

Parameters

- **bytes** – Entropy to use.
- **bytes_len** – Size of `bytes` in bytes. Must be one of `BIP32_ENTROPY_LEN_128`, `BIP32_ENTROPY_LEN_256` or `BIP32_ENTROPY_LEN_512`.

- **version** – Either BIP32_VER_MAIN_PRIVATE or BIP32_VER_TEST_PRIVATE, indicating mainnet or testnet/regtest respectively.
- **hmac_key** – Custom data to HMAC-SHA512 with bytes before creating the key. Pass NULL to use the default BIP32 key of "Bitcoin seed".
- **hmac_key_len** – Size of hmac_key in bytes, or 0 if hmac_key is NULL.
- **flags** – Either BIP32_FLAG_SKIP_HASH to skip hash160 calculation, or 0.
- **output** – Destination for the resulting master extended key.

Returns See *Error Codes*

int **bip32_key_from_seed**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *version*, uint32_t *flags*, struct ext_key **output*)

As per *bip32_key_from_seed_custom* With the default BIP32 seed. :return: See *Error Codes*

int **bip32_key_from_seed_custom_alloc**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *version*, const unsigned char **hmac_key*, size_t *hmac_key_len*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_from_seed_custom*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_from_seed_alloc**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *version*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_from_seed*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_serialize**(const struct ext_key **hdkey*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Serialize an extended key to memory using BIP32 format.

Parameters

- **hdkey** – The extended key to serialize.
- **flags** – BIP32_FLAG_KEY_ Flags indicating which key to serialize. You can not serialize a private extended key from a public extended key.
- **bytes_out** – Destination for the serialized key.
- **len** – Size of bytes_out in bytes. Must be BIP32_SERIALIZED_LEN.

Returns See *Error Codes*

int **bip32_key_unserialize**(const unsigned char **bytes*, size_t *bytes_len*, struct ext_key **output*)

Un-serialize an extended key from memory.

Parameters

- **bytes** – Storage holding the serialized key.
- **bytes_len** – Size of bytes in bytes. Must be BIP32_SERIALIZED_LEN.
- **output** – Destination for the resulting extended key.

Returns See *Error Codes*

int **bip32_key_unserialize_alloc**(const unsigned char **bytes*, size_t *bytes_len*, struct ext_key ***output*)

As per *bip32_key_unserialize*, but allocates the key.

Note: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_from_parent**(const struct ext_key **hdkey*, uint32_t *child_num*, uint32_t *flags*, struct ext_key **output*)

Create a new child extended key from a parent extended key.

Parameters

- **hdkey** – The parent extended key.
- **child_num** – The child number to create. Numbers greater than or equal to BIP32_INITIAL_HARDENED_CHILD represent hardened keys that cannot be created from public parent extended keys.
- **flags** – BIP32_FLAG_KEY_ Flags indicating the type of derivation wanted. You can not derive a private child extended key from a public parent extended key.
- **output** – Destination for the resulting child extended key.

Returns See *Error Codes*

int **bip32_key_from_parent_alloc**(const struct ext_key **hdkey*, uint32_t *child_num*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_from_parent*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_from_parent_path**(const struct ext_key **hdkey*, const uint32_t **child_path*, size_t *child_path_len*, uint32_t *flags*, struct ext_key **output*)

Create a new child extended key from a parent extended key and a path.

Parameters

- **hdkey** – The parent extended key.
- **child_path** – The path of child numbers to create.
- **child_path_len** – The number of child numbers in child_path.
- **flags** – BIP32_FLAG_ Flags indicating the type of derivation wanted.
- **output** – Destination for the resulting child extended key.

Returns See *Error Codes*

int **bip32_key_from_parent_path_alloc**(const struct ext_key **hdkey*, const uint32_t **child_path*, size_t *child_path_len*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_from_parent_path*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_from_parent_path_str**(const struct ext_key **hdkey*, const char **path_str*, uint32_t *child_num*, uint32_t *flags*, struct ext_key **output*)

Create a new child extended key from a parent extended key and a path string.

Parameters

- **hdkey** – The parent extended key.
- **path_str** – The BIP32 path string of child numbers to create.
- **child_num** – The child number to use if `path_str` contains a `*` wildcard.
- **flags** – `BIP32_FLAG_` Flags indicating the type of derivation wanted.
- **output** – Destination for the resulting child extended key.

int **bip32_key_from_parent_path_str_n**(const struct ext_key **hdkey*, const char **path_str*, size_t *path_str_len*, uint32_t *child_num*, uint32_t *flags*, struct ext_key **output*)

Create a new child extended key from a parent extended key and a known-length path string.

See *bip32_key_from_parent_path_str*.

int **bip32_key_from_parent_path_str_alloc**(const struct ext_key **hdkey*, const char **path_str*, uint32_t *child_num*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_from_parent_path_str*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

int **bip32_key_from_parent_path_str_n_alloc**(const struct ext_key **hdkey*, const char **path_str*, size_t *path_str_len*, uint32_t *child_num*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_from_parent_path_str_n*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

int **bip32_key_with_tweak_from_parent_path**(const struct ext_key **hdkey*, const uint32_t **child_path*, size_t *child_path_len*, uint32_t *flags*, struct ext_key **output*)

Derive the pub tweak from a parent extended key and a path.

Parameters

- **hdkey** – The parent extended key.
- **child_path** – The path of child numbers to create.
- **child_path_len** – The number of child numbers in `child_path`.
- **bytes_out** – Destination for the resulting pub tweak.
- **len** – Length of `bytes_out` in bytes. Must be `EC_PRIVATE_KEY_LEN`.

Returns See *Error Codes*

int **bip32_key_with_tweak_from_parent_path_alloc**(const struct ext_key **hdkey*, const uint32_t **child_path*, size_t *child_path_len*, uint32_t *flags*, struct ext_key ***output*)

As per *bip32_key_with_tweak_from_parent_path*, but allocates the key. .. note:: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_to_base58**(const struct ext_key **hdkey*, uint32_t *flags*, char ***output*)

Convert an extended key to base58.

Parameters

- **hdkey** – The extended key.
- **flags** – `BIP32_FLAG_KEY_` Flags indicating which key to serialize. You can not serialize a private extended key from a public extended key.

- **output** – Destination for the resulting key in base58. The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

int **bip32_key_from_base58**(const char **base58*, struct ext_key **output*)
Convert a base58 encoded extended key to an extended key.

Parameters

- **base58** – The extended key in base58.
- **output** – Destination for the resulting extended key.

Returns See *Error Codes*

int **bip32_key_from_base58_n**(const char **base58*, size_t *base58_len*, struct ext_key **output*)
Convert a known-length base58 encoded extended key to an extended key.

See *bip32_key_from_base58*. :return: See *Error Codes*

int **bip32_key_from_base58_alloc**(const char **base58*, struct ext_key ***output*)
As per *bip32_key_from_base58*, but allocates the key.

Note: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_from_base58_n_alloc**(const char **base58*, size_t *base58_len*, struct ext_key ***output*)
As per *bip32_key_from_base58_n*, but allocates the key.

Note: The returned key should be freed with *bip32_key_free*.

Returns See *Error Codes*

int **bip32_key_strip_private_key**(struct ext_key **hdkey*)
Converts a private extended key to a public extended key. Afterwards, only public child extended keys can be derived, and only the public serialization can be created. If the provided key is already public, nothing will be done.

Parameters

- **hdkey** – The extended key to covert.

Returns See *Error Codes*

int **bip32_key_get_fingerprint**(struct ext_key **hdkey*, unsigned char **bytes_out*, size_t *len*)
Get the BIP32 fingerprint for an extended key. Performs hash160 calculation if previously skipped with `BIP32_FLAG_SKIP_HASH`.

Parameters

- **hdkey** – The extended key.
- **bytes_out** – Destination for the fingerprint.
- **len** – Size of `bytes_out` in bytes. Must be `BIP32_KEY_FINGERPRINT_LEN`.

Returns See *Error Codes*

CHAPTER 5

Bip38 Functions

int **bip38_raw_from_private_key**(const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **pass*, size_t *pass_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Encode a private key in raw BIP 38 address format.

Parameters

- **bytes** – Private key to use.
- **bytes_len** – Size of `bytes` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **pass** – Password for the encoded private key.
- **pass_len** – Length of `pass` in bytes.
- **flags** – **BIP38_KEY_** flags indicating desired behavior.
- **bytes_out** – Destination for the resulting raw BIP38 address.
- **len** – Size of `bytes_out` in bytes. Must be `BIP38_SERIALIZED_LEN`.

Returns See *Error Codes*

int **bip38_from_private_key**(const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **pass*, size_t *pass_len*, uint32_t *flags*, char ***output*)

Encode a private key in BIP 38 address format.

Parameters

- **bytes** – Private key to use.
- **bytes_len** – Size of `bytes` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **pass** – Password for the encoded private key.
- **pass_len** – Length of `pass` in bytes.
- **flags** – **BIP38_KEY_** flags indicating desired behavior.
- **output** – Destination for the resulting BIP38 address.

Returns See *Error Codes*

int **bip38_raw_to_private_key**(const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **pass*, size_t *pass_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Decode a raw BIP 38 address to a private key.

Parameters

- **bytes** – Raw BIP 38 address to decode.
- **bytes_len** – Size of `bytes` in bytes. Must be `BIP38_SERIALIZED_LEN`.
- **pass** – Password for the encoded private key.
- **pass_len** – Length of `pass` in bytes.
- **flags** – **BIP38_KEY_** flags indicating desired behavior.
- **bytes_out** – Destination for the resulting private key.
- **len** – Size of `bytes_out` in bytes. Must be `EC_PRIVATE_KEY_LEN`.

Returns See *Error Codes*

int **bip38_to_private_key**(const char **bip38*, const unsigned char **pass*, size_t *pass_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Decode a BIP 38 address to a private key.

Parameters

- **bip38** – BIP 38 address to decode.
- **pass** – Password for the encoded private key.
- **pass_len** – Length of `pass` in bytes.
- **flags** – **BIP38_KEY_** flags indicating desired behavior.
- **bytes_out** – Destination for the resulting private key.
- **len** – Size of `bytes_out` in bytes. Must be `EC_PRIVATE_KEY_LEN`.

Returns See *Error Codes*

int **bip38_raw_get_flags**(const unsigned char **bytes*, size_t *bytes_len*, size_t **written*)

Get compression and/or EC mult flags.

Parameters

- **bytes** – Raw BIP 38 address to get the flags from.
- **bytes_len** – Size of `bytes` in bytes. Must be `BIP38_SERIALIZED_LEN`.
- **written** – **BIP38_KEY_** flags indicating behavior.

Returns See *Error Codes*

int **bip38_get_flags**(const char **bip38*, size_t **written*)

Get compression and/or EC mult flags.

Parameters

- **bip38** – BIP 38 address to get the flags from.
- **written** – **BIP38_KEY_** flags indicating behavior.

Returns See *Error Codes*

CHAPTER 6

Bip39 Functions

int **bip39_get_languages**(char **_output_)
Get the list of default supported languages.

..note:: The string returned should be freed using `wally_free_string`. :return: See *Error Codes*

int **bip39_get_wordlist**(const char *_lang_, struct words **_output_)
Get the default word list for a language.

Parameters

- **lang** – Language to use. Pass NULL to use the default English list.
- **output** – Destination for the resulting word list.

Note: The returned structure should not be freed or modified.

Returns See *Error Codes*

int **bip39_get_word**(const struct words *_w_, size_t _index_, char **_output_)
Get the 'index'th word from a word list.

Parameters

- **w** – Word list to use. Pass NULL to use the default English list.
- **index** – The 0-based index of the word in `w`.
- **output** – Destination for the resulting word.

The string returned should be freed using `wally_free_string`. :return: See *Error Codes*

int **bip39_mnemonic_from_bytes**(const struct words *_w_, const unsigned char *_bytes_, size_t _bytes_len_, char **_output_)
Generate a mnemonic sentence from the entropy in `bytes`.

Parameters

- **w** – Word list to use. Pass NULL to use the default English list.
- **bytes** – Entropy to convert.
- **bytes_len** – The length of `bytes` in bytes.
- **output** – Destination for the resulting mnemonic sentence.

Note: The string returned should be freed using *wally_free_string*.

Returns See *Error Codes*

int **bip39_mnemonic_to_bytes**(const struct words *w, const char *mnemonic, unsigned char *bytes_out, size_t len, size_t *written)

Convert a mnemonic sentence into entropy at `bytes_out`.

Parameters

- **w** – Word list to use. Pass NULL to use the default English list.
- **mnemonic** – Mnemonic to convert.
- **bytes_out** – Where to store the resulting entropy.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **bip39_mnemonic_validate**(const struct words *w, const char *mnemonic)

Validate the checksum embedded in a mnemonic sentence.

Parameters

- **w** – Word list to use. Pass NULL to use the default English list.
- **mnemonic** – Mnemonic to validate.

Returns See *Error Codes*

int **bip39_mnemonic_to_seed**(const char *mnemonic, const char *passphrase, unsigned char *bytes_out, size_t len, size_t *written)

Convert a mnemonic into a binary seed.

Parameters

- **mnemonic** – Mnemonic to convert.
- **passphrase** – Mnemonic passphrase or NULL if no passphrase is needed.
- **bytes_out** – The destination for the binary seed.
- **len** – The length of `bytes_out` in bytes. Currently This must be `BIP39_SEED_LEN_512`.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

CHAPTER 7

Psbt Functions

int **wally_map_init_alloc**(size_t *allocation_len*, struct wally_map **output*)
Allocate and initialize a new map.

Parameters

- **allocation_len** – The number of items to allocate.
- **output** – Destination for the new map.

Returns See *Error Codes*

int **wally_map_free**(struct wally_map *map_in*)
Free a map allocated by *wally_map_init_alloc*.

Parameters

- **map_in** – The map to free.

Returns See *Error Codes*

int **wally_map_find**(const struct wally_map *map_in*, const unsigned char *key*, size_t *key_len*, size_t *written*)
Find an item in a map.

Parameters

- **map_in** – The map to find `key` in.
- **key** – The key to find.
- **key_len** – Length of `key` in bytes.
- **written** – On success, set to zero if the item is not found, otherwise the index of the item plus one.

Returns See *Error Codes*

int **wally_map_add**(struct wally_map *map_in*, const unsigned char *key*, size_t *key_len*, const unsigned char *value*, size_t *value_len*)
Add an item to a map.

Parameters

- **map_in** – The map to add to.
- **key** – The key to add.
- **key_len** – Length of `key` in bytes.
- **value** – The value to add.
- **value_len** – Length of `value` in bytes.

Returns See *Error Codes*

int **wally_map_add_keypath_item**(struct wally_map **map_in*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **fingerprint*, size_t *fingerprint_len*, const uint32_t **child_path*, size_t *child_path_len*)

Convert and add a pubkey/keypath to a map.

Parameters

- **map_in** – The map to add to.
- **pub_key** – The pubkey to add.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **fingerprint** – The master key fingerprint for the pubkey.
- **fingerprint_len** – Length of `fingerprint` in bytes. Must be `BIP32_KEY_FINGERPRINT_LEN`.
- **child_path** – The BIP32 derivation path for the pubkey.
- **child_path_len** – The number of items in `child_path`.

Returns See *Error Codes*

int **wally_map_sort**(struct wally_map **map_in*, uint32_t *flags*)

Sort the items in a map.

Parameters

- **map_in** – The map to sort.
- **flags** – Flags controlling sorting. Must be 0.

Returns See *Error Codes*

int **wally_psbt_input_is_finalized**(const struct wally_psbt_input **input*, size_t **written*)

Determine if a PSBT input is finalized.

Parameters

- **input** – The input to check.
- **written** – On success, set to one if the input is finalized, otherwise zero.

Returns See *Error Codes*

int **wally_psbt_input_set_utxo**(struct wally_psbt_input **input*, const struct wally_tx **utxo*)

Set the utxo in an input.

Parameters

- **input** – The input to update.
- **utxo** – The (non witness) utxo for this input if it exists.

Returns See *Error Codes*

int **wally_psbt_input_set_witness_utxo**(struct wally_psbt_input **input*, const struct wally_tx_output **witness_utxo*)

Set the witness_utxo in an input.

Parameters

- **input** – The input to update.
- **witness_utxo** – The witness utxo for this input if it exists.

Returns See *Error Codes*

int **wally_psbt_input_set_redeem_script**(struct wally_psbt_input **input*, const unsigned char **script*, size_t *script_len*)

Set the redeem_script in an input.

Parameters

- **input** – The input to update.
- **script** – The redeem script for this input.
- **script_len** – Length of `script` in bytes.

Returns See *Error Codes*

int **wally_psbt_input_set_witness_script**(struct wally_psbt_input **input*, const unsigned char **script*, size_t *script_len*)

Set the witness_script in an input.

Parameters

- **input** – The input to update.
- **script** – The witness script for this input.
- **script_len** – Length of `script` in bytes.

Returns See *Error Codes*

int **wally_psbt_input_set_final_scriptsig**(struct wally_psbt_input **input*, const unsigned char **final_scriptsig*, size_t *final_scriptsig_len*)

Set the final_scriptsig in an input.

Parameters

- **input** – The input to update.
- **final_scriptsig** – The scriptSig for this input.
- **final_scriptsig_len** – Length of `final_scriptsig` in bytes.

Returns See *Error Codes*

int **wally_psbt_input_set_final_witness**(struct wally_psbt_input **input*, const struct wally_tx_witness_stack **final_witness*)

Set the final_witness in an input.

Parameters

- **input** – The input to update.
- **final_witness** – The witness stack for the input, or NULL if no witness is present.

Returns See *Error Codes*

int **wally_psbt_input_set_keypaths**(struct wally_psbt_input **input*, const struct wally_map **map_in*)

Set the keypaths in an input.

Parameters

- **input** – The input to update.
- **map_in** – The HD keypaths for this input.

Returns See *Error Codes*

int **wally_psbt_input_find_keypath**(struct wally_psbt_input **input*, const unsigned char **pub_key*, size_t *pub_key_len*, size_t **written*)

Find a keypath matching a pubkey in an input.

Parameters

- **input** – The input to search in.
- **pub_key** – The pubkey to find.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **written** – On success, set to zero if the item is not found, otherwise the index of the item plus one.

Returns See *Error Codes*

int **wally_psbt_input_add_keypath_item**(struct wally_psbt_input **input*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **fingerprint*, size_t *fingerprint_len*, const uint32_t **child_path*, size_t *child_path_len*)

Convert and add a pubkey/keypath to an input.

Parameters

- **input** – The input to add to.
- **pub_key** – The pubkey to add.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **fingerprint** – The master key fingerprint for the pubkey.
- **fingerprint_len** – Length of `fingerprint` in bytes. Must be `BIP32_KEY_FINGERPRINT_LEN`.
- **child_path** – The BIP32 derivation path for the pubkey.
- **child_path_len** – The number of items in `child_path`.

Returns See *Error Codes*

int **wally_psbt_input_set_signatures**(struct wally_psbt_input **input*, const struct wally_map **map_in*)

Set the partial signatures in an input.

Parameters

- **input** – The input to update.
- **map_in** – The partial signatures for this input.

Returns See *Error Codes*

int **wally_psbt_input_find_signature**(struct wally_psbt_input **input*, const unsigned char **pub_key*, size_t *pub_key_len*, size_t **written*)

Find a partial signature matching a pubkey in an input.

Parameters

- **input** – The input to search in.
- **pub_key** – The pubkey to find.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **written** – On success, set to zero if the item is not found, otherwise the index of the item plus one.

Returns See *Error Codes*

int **wally_psbt_input_add_signature**(struct wally_psbt_input **input*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **sig*, size_t *sig_len*)

Add a pubkey/partial signature item to an input.

Parameters

- **input** – The input to add the partial signature to.
- **pub_key** – The pubkey to find.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **sig** – The DER-encoded signature plus sighash byte to add.
- **sig_len** – The length of `sig` in bytes.

Returns See *Error Codes*

int **wally_psbt_input_set_unknowns**(struct wally_psbt_input **input*, const struct wally_map **map_in*)

Set the unknown values in an input.

Parameters

- **input** – The input to update.
- **map_in** – The unknown key value pairs for this input.

Returns See *Error Codes*

int **wally_psbt_input_find_unknown**(struct wally_psbt_input **input*, const unsigned char **key*, size_t *key_len*, size_t **written*)

Find an unknown item matching a key in an input.

Parameters

- **input** – The input to search in.
- **key** – The key to find.
- **key_len** – Length of `key` in bytes.
- **written** – On success, set to zero if the item is not found, otherwise the index of the item plus one.

Returns See *Error Codes*

int **wally_psbt_input_set_sighash**(struct wally_psbt_input **input*, uint32_t *sighash*)

Set the sighash type in an input.

Parameters

- **input** – The input to update.
- **sighash** – The sighash type for this input.

Returns See *Error Codes*

int **wally_psbt_output_set_redeem_script**(struct wally_psbt_output **output*, const unsigned char **script*, size_t *script_len*)

Set the redeem_script in an output.

Parameters

- **output** – The input to update.
- **script** – The redeem script for this output.
- **script_len** – Length of `script` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_witness_script**(struct wally_psbt_output **output*, const unsigned char **script*, size_t *script_len*)

Set the witness_script in an output.

Parameters

- **output** – The output to update.
- **script** – The witness script for this output.
- **script_len** – Length of `script` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_keypaths**(struct wally_psbt_output **output*, const struct wally_map **map_in*)

Set the keypaths in an output.

Parameters

- **output** – The output to update.
- **map_in** – The HD keypaths for this output.

Returns See *Error Codes*

int **wally_psbt_output_find_keypath**(struct wally_psbt_output **output*, const unsigned char **pub_key*, size_t *pub_key_len*, size_t **written*)

Find a keypath matching a pubkey in an output.

Parameters

- **output** – The output to search in.
- **pub_key** – The pubkey to find.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **written** – On success, set to zero if the item is not found, otherwise the index of the item plus one.

Returns See *Error Codes*

int **wally_psbt_output_add_keypath_item**(struct wally_psbt_output **output*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **fingerprint*, size_t *fingerprint_len*, const uint32_t **child_path*, size_t *child_path_len*)

Convert and add a pubkey/keypath to an output.

Parameters

- **output** – The output to add to.
- **pub_key** – The pubkey to add.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_UNCOMPRESSED_LEN` or `EC_PUBLIC_KEY_LEN`.
- **fingerprint** – The master key fingerprint for the pubkey.
- **fingerprint_len** – Length of `fingerprint` in bytes. Must be `BIP32_KEY_FINGERPRINT_LEN`.
- **child_path** – The BIP32 derivation path for the pubkey.
- **child_path_len** – The number of items in `child_path`.

Returns See *Error Codes*

int **wally_psbt_output_set_unknowns**(struct wally_psbt_output **output*, const struct wally_map **map_in*)

Set the unknown map in an output.

Parameters

- **output** – The output to update.
- **map_in** – The unknown key value pairs for this output.

Returns See *Error Codes*

int **wally_psbt_output_find_unknown**(struct wally_psbt_output **output*, const unsigned char **key*, size_t *key_len*, size_t **written*)

Find an unknown item matching a key in an output.

Parameters

- **output** – The output to search in.
- **key** – The key to find.
- **key_len** – Length of `key` in bytes.
- **written** – On success, set to zero if the item is not found, otherwise the index of the item plus one.

Returns See *Error Codes*

int **wally_psbt_init_alloc**(uint32_t *version*, size_t *inputs_allocation_len*, size_t *outputs_allocation_len*, size_t *global_unknowns_allocation_len*, struct wally_psbt ***output*)

Allocate and initialize a new PSBT.

Parameters

- **version** – The version of the PSBT. Must be 0.
- **inputs_allocation_len** – The number of inputs to pre-allocate space for.
- **outputs_allocation_len** – The number of outputs to pre-allocate space for.

- **global_unknowns_allocation_len** – The number of global unknowns to allocate space for.
- **output** – Destination for the resulting PSBT output.

Returns See *Error Codes*

int **wally_psbt_free**(struct wally_psbt **psbt*)
Free a PSBT allocated by `wally_psbt_init_alloc`.

Parameters

- **psbt** – The PSBT to free.

Returns See *Error Codes*

int **wally_psbt_is_finalized**(const struct wally_psbt **psbt*, size_t **written*)
Determine if all PSBT inputs are finalized.

Parameters

- **psbt** – The PSBT to check.
- **written** – On success, set to one if the PSBT is finalized, otherwise zero.

Returns See *Error Codes*

int **wally_psbt_set_global_tx**(struct wally_psbt **psbt*, const struct wally_tx **tx*)
Set the global transaction for a PSBT.

Parameters

- **psbt** – The PSBT to set the transaction for.
- **tx** – The transaction to set.

The global transaction can only be set on a newly created PSBT. After this call completes the PSBT will have empty inputs and outputs for each input and output in the transaction `tx` given. :return: See *Error Codes*

int **wally_psbt_add_input_at**(struct wally_psbt **psbt*, uint32_t *index*, uint32_t *flags*, const struct wally_tx_input **input*)
Add a transaction input to PBST at a given position.

Parameters

- **psbt** – The PSBT to add the input to.
- **index** – The zero-based index of the position to add the input at.
- **flags** – Flags controlling input insertion. Must be 0 or `WALLY_PSBT_FLAG_NON_FINAL`.
- **input** – The transaction input to add.

Returns See *Error Codes*

int **wally_psbt_remove_input**(struct wally_psbt **psbt*, uint32_t *index*)
Remove a transaction input from a PBST.

Parameters

- **psbt** – The PSBT to remove the input from.
- **index** – The zero-based index of the input to remove.

Returns See *Error Codes*

int **wally_psbt_add_output_at**(struct wally_psbt **psbt*, uint32_t *index*, uint32_t *flags*, const struct wally_tx_output **output*)
Add a transaction output to PBST at a given position.

Parameters

- **psbt** – The PSBT to add the output to.
- **index** – The zero-based index of the position to add the output at.
- **flags** – Flags controlling output insertion. Must be 0.
- **output** – The transaction output to add.

Returns See *Error Codes*

int **wally_psbt_remove_output**(struct wally_psbt **psbt*, uint32_t *index*)
Remove a transaction output from a PBST.

Parameters

- **psbt** – The PSBT to remove the output from.
- **index** – The zero-based index of the output to remove.

Returns See *Error Codes*

int **wally_psbt_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, struct wally_psbt ***output*)
Create a PSBT from its serialized bytes.

Parameters

- **bytes** – Bytes to create the PSBT from.
- **bytes_len** – Length of `bytes` in bytes.
- **output** – Destination for the resulting PSBT.

Returns See *Error Codes*

int **wally_psbt_get_length**(const struct wally_psbt **psbt*, uint32_t *flags*, size_t **written*)
Get the length of a PSBT when serialized to bytes.

Parameters

- **psbt** – the PSBT.
- **flags** – Flags controlling length determination. Must be 0.
- **written** – Destination for the length in bytes when serialized.

Returns See *Error Codes*

int **wally_psbt_to_bytes**(const struct wally_psbt **psbt*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Serialize a PSBT to bytes.

Parameters

- **psbt** – the PSBT to serialize.
- **flags** – Flags controlling serialization. Must be 0.
- **bytes_out** – Bytes to create the transaction from.
- **len** – Length of `bytes` in bytes (use *`wally_psbt_get_length`*).
- **written** – number of bytes written to bytes_out.

Returns See *Variable Length Output Buffers*

int **wally_psbt_from_base64**(const char **base64*, struct wally_psbt ***output*)
Create a PSBT from its serialized base64 string.

Parameters

- **base64** – Base64 string to create the PSBT from.
- **output** – Destination for the resulting PSBT.

Returns See *Error Codes*

int **wally_psbt_to_base64**(const struct wally_psbt **psbt*, uint32_t *flags*, char ***output*)
Serialize a PSBT to a base64 string.

Parameters

- **psbt** – the PSBT to serialize.
- **flags** – Flags controlling serialization. Must be 0.
- **output** – Destination for the resulting serialized PSBT.

Returns See *Error Codes*

int **wally_psbt_combine**(struct wally_psbt **psbt*, const struct wally_psbt **src*)
Combine the metadata from a source PSBT into another PSBT.

Parameters

- **psbt** – the PSBT to combine into.
- **source** – the PSBT to copy data from.

Returns See *Error Codes*

int **wally_psbt_clone_alloc**(const struct wally_psbt **psbt*, uint32_t *flags*, struct wally_psbt ***output*)
Clone a PSBT into a newly allocated copy.

Parameters

- **psbt** – the PSBT to clone.
- **flags** – Flags controlling PSBT creation. Must be 0.
- **output** – Destination for the resulting cloned PSBT.

Returns See *Error Codes*

int **wally_psbt_sign**(struct wally_psbt **psbt*, const unsigned char **key*, size_t *key_len*, uint32_t *flags*)
Sign a PSBT using the simple signer algorithm.

Parameters

- **psbt** – PSBT to sign. Directly modifies this PSBT.
- **key** – Private key to sign PSBT with.
- **key_len** – Length of key in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **flags** – Flags controlling sigining. Must be 0 or EC_FLAG_GRIND_R.

Note: See https://github.com/bitcoin/bips/blob/master/bip-0174.mediawiki#simple-signer-algorithm for a description of the simple signer algorithm.

Returns See *Error Codes*

int **wally_psbt_finalize**(struct wally_psbt **psbt*)
Finalize a PSBT.

Parameters

- **psbt** – PSBT to finalize. Directly modifies this PSBT.

Returns See *Error Codes*

int **wally_psbt_extract**(const struct wally_psbt **psbt*, struct wally_tx ***output*)
Extract a network transaction from a finalized PSBT.

Parameters

- **psbt** – PSBT to extract from.
- **output** – Destination for the resulting transaction.

Returns See *Error Codes*

int **wally_psbt_is_elements**(const struct wally_psbt **psbt*, size_t **written*)
Determine if a PSBT is an elements PSBT.

Parameters

- **psbt** – The PSBT to check.
- **written** – 1 if the PSBT is an elements PSBT, otherwise 0.

Returns See *Error Codes*

int **wally_psbt_elements_init_alloc**(uint32_t *version*, size_t *inputs_allocation_len*, size_t *outputs_allocation_len*, size_t *global_unknowns_allocation_len*, struct wally_psbt ***output*)
Allocate and initialize a new elements PSBT.

Parameters

- **version** – The version of the PSBT. Must be 0.
- **inputs_allocation_len** – The number of inputs to pre-allocate space for.
- **outputs_allocation_len** – The number of outputs to pre-allocate space for.
- **global_unknowns_allocation_len** – The number of global unknowns to allocate space for.
- **output** – Destination for the resulting PSBT output.

Returns See *Error Codes*

int **wally_psbt_input_set_value**(struct wally_psbt_input **input*, uint64_t *value*)
Set the value in an elements input.

Parameters

- **input** – The input to update.
- **value** – The value for this input.

Returns See *Error Codes*

int **wally_psbt_input_clear_value**(struct wally_psbt_input **input*)
Clear the value in an elements input.

Parameters

- **input** – The input to update.

Returns See *Error Codes*

int **wally_psbt_input_set_vbf**(struct wally_psbt_input **input*, const unsigned char **vbf*, size_t *vbf_len*)

Set the value blinding factor in an elements input.

Parameters

- **input** – The input to update.
- **vbf** – The value blinding factor.
- **vbf_len** – Length of `vbf`. Must be `BLINDING_FACTOR_LEN`.

Returns See *Error Codes*

int **wally_psbt_input_set_asset**(struct wally_psbt_input **input*, const unsigned char **asset*, size_t *asset_len*)

Set the asset in an elements input.

Parameters

- **input** – The input to update.
- **asset** – The asset for this input.
- **asset_len** – Length of `asset` in bytes.

Returns See *Error Codes*

int **wally_psbt_input_set_abf**(struct wally_psbt_input **input*, const unsigned char **abf*, size_t *abf_len*)

Set the asset blinding factor in an elements input

Parameters

- **input** – The input to update.
- **abf** – The asset blinding factor.
- **abf_len** – Length of `abf` in bytes. Must be `BLINDING_FACTOR_LEN`.

Returns See *Error Codes*

int **wally_psbt_input_set_pegin_tx**(struct wally_psbt_input **input*, const struct wally_tx **pegin_tx*)

Set the peg in tx in an input.

Parameters

- **input** – The input to update.
- **pegin_tx** – The peg in tx for this input if it exists.

Returns See *Error Codes*

int **wally_psbt_input_set_txoutproof**(struct wally_psbt_input **input*, const unsigned char **proof*, size_t *proof_len*)

Set the txout proof in an elements input.

Parameters

- **input** – The input to update.
- **proof** – The txout proof for this input.
- **proof_len** – Length of `proof` in bytes.

Returns See *Error Codes*

int **wally_psbt_input_set_genesis_blockhash**(struct wally_psbt_input **input*, const unsigned char **genesis_blockhash*, size_t *genesis_blockhash_len*)

Set the genesis hash in an elements input.

Parameters

- **input** – The input to update.
- **genesis_blockhash** – The genesis hash for this input.
- **genesis_blockhash_len** – Length of `genesis_blockhash` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_psbt_input_set_claim_script**(struct wally_psbt_input **input*, const unsigned char **script*, size_t *script_len*)

Set the claim script in an elements input.

Parameters

- **input** – The input to update.
- **script** – The claim script for this input.
- **script_len** – Length of `script` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_blinding_pubkey**(struct wally_psbt_output **output*, const unsigned char **pub_key*, size_t *pub_key_len*)

Set the blinding pubkey in an elements output.

Parameters

- **output** – The output to update.
- **pub_key** – The blinding pubkey for this output.
- **pub_key_len** – Length of `pub_key` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_value_commitment**(struct wally_psbt_output **output*, const unsigned char **commitment*, size_t *commitment_len*)

Set the value commitment in an elements output.

Parameters

- **output** – The output to update.
- **commitment** – The value commitment for this output.
- **commitment_len** – Length of `commitment` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_vbf**(struct wally_psbt_output **output*, const unsigned char **vbf*, size_t *vbf_len*)

Set the value blinding factor in an elements output.

Parameters

- **output** – The output to update.
- **vbf** – The value blinding factor.
- **vbf_len** – Length of `vbf`. Must be `BLINDING_FACTOR_LEN`.

Returns See *Error Codes*

int **wally_psbt_output_set_asset_commitment**(struct wally_psbt_output **output*, const unsigned char **commitment*, size_t *commitment_len*)

Set the asset commitment in an elements output.

Parameters

- **output** – The output to update.
- **commitment** – The asset commitment for this output.
- **commitment_len** – Length of `commitment` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_abf**(struct wally_psbt_output **output*, const unsigned char **abf*, size_t *abf_len*)

Set the asset blinding factor in an elements output.

Parameters

- **output** – The output to update.
- **abf** – The asset blinding factor.
- **abf_len** – Length of `abf` in bytes. Must be `BLINDING_FACTOR_LEN`.

Returns See *Error Codes*

int **wally_psbt_output_set_nonce**(struct wally_psbt_output **output*, const unsigned char **nonce*, size_t *nonce_len*)

Set the nonce commitment in an elements output.

Parameters

- **output** – The output to update.
- **nonce** – The commitment used to create the nonce (with the blinding key) for the range proof.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_CT_NONCE_LEN`.

Returns See *Error Codes*

int **wally_psbt_output_set_rangeproof**(struct wally_psbt_output **output*, const unsigned char **proof*, size_t *proof_len*)

Set the range proof in an elements output.

Parameters

- **output** – The output to update.
- **proof** – The range proof for this output.
- **proof_len** – Length of `proof` in bytes.

Returns See *Error Codes*

int **wally_psbt_output_set_surjectionproof**(struct wally_psbt_output **output*, const unsigned char **proof*, size_t *proof_len*)

Set the surjection proof in an elements output.

Parameters

- **output** – The output to update.
- **proof** – The surjection proof for this output.

- **proof_len** – Length of `proof` in bytes.

Returns See *Error Codes*

CHAPTER 8

Script Functions

int **wally_scriptpubkey_get_type**(const unsigned char **bytes*, size_t *bytes_len*, size_t **written*)

Determine the type of a scriptPubkey script.

Parameters

- **bytes** – Bytes of the scriptPubkey.
- **bytes_len** – Length of `bytes` in bytes.
- **written** – Destination for the `WALLY_SCRIPT_TYPE_` script type.

Returns See *Error Codes*

int **wally_scriptpubkey_p2pkh_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a P2PKH scriptPubkey.

Parameters

- **bytes** – Bytes to create a scriptPubkey for.
- **bytes_len** – The length of `bytes` in bytes. If `WALLY_SCRIPT_HASH160` is given in `flags`, `bytes` is a public key to hash160 before creating the P2PKH, and `bytes_len` must be `EC_PUBLIC_KEY_LEN` or `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`. Otherwise, `bytes_len` must be `HASH160_LEN` and `bytes` must contain the hash160 to use.
- **flags** – `WALLY_SCRIPT_HASH160` or 0.
- **bytes_out** – Destination for the resulting scriptPubkey.
- **len** – Length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_scriptsig_p2pkh_from_sig**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **sig*, size_t *sig_len*, uint32_t *sighash*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a P2PKH scriptSig from a pubkey and compact signature.

This function creates the scriptSig by converting `sig` to DER encoding, appending the given sighash, then calling *wally_scriptsig_p2pkh_from_der*.

Parameters

- **pub_key** – The public key to create a scriptSig with.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN` or `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`.
- **sig** – The compact signature to create a scriptSig with.
- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_LEN`.
- **sighash** – `WALLY_SIGHASH_` flags specifying the type of signature desired.
- **bytes_out** – Destination for the resulting scriptSig.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_witness_p2wpkh_from_sig**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **sig*, size_t *sig_len*, uint32_t *sighash*, struct wally_tx_witness_stack ***witness*)

Create a P2WPKH witness from a pubkey and compact signature.

Parameters

- **pub_key** – The public key to create a witness with.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN` or `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`.
- **sig** – The compact signature to create a witness with.
- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_LEN`.
- **sighash** – `WALLY_SIGHASH_` flags specifying the type of signature desired.
- **witness** – Destination for the newly created witness.

Returns See *Error Codes*

int **wally_scriptsig_p2pkh_from_der**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **sig*, size_t *sig_len*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a P2PKH scriptSig from a pubkey and DER signature plus sighash.

Parameters

- **pub_key** – The public key to create a scriptSig with.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN` `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`.
- **sig** – The DER encoded signature to create a scriptSig, with the sighash byte appended to it.
- **sig_len** – The length of `sig` in bytes.

- **bytes_out** – Destination for the resulting scriptSig.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_witness_p2wpkh_from_der**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **sig*, size_t *sig_len*, struct wally_tx_witness_stack ***witness*)

Create a P2WPKH witness from a pubkey and DER signature plus sighash.

Parameters

- **pub_key** – The public key to create a witness with.
- **pub_key_len** – Length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN` `EC_PUBLIC_KEY_UNCOMPRESSED_LEN`.
- **sig** – The DER encoded signature to create a witness, with the sighash byte appended to it.
- **sig_len** – The length of `sig` in bytes.
- **witness** – Destination for the newly created witness.

Returns See *Error Codes*

int **wally_scriptpubkey_op_return_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create an OP_RETURN scriptPubkey.

Parameters

- **bytes** – Bytes to create a scriptPubkey for.
- **bytes_len** – Length of `bytes` in bytes. Must be less than or equal to `WALLY_MAX_OP_RETURN_LEN`.
- **flags** – Currently unused, must be 0.
- **bytes_out** – Destination for the resulting scriptPubkey.
- **len** – The length of `bytes_out` in bytes. Passing `WALLY_SCRIPTPUBKEY_OP_RETURN_MAX_LEN` will ensure there is always enough room for the resulting scriptPubkey.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_scriptpubkey_p2sh_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a P2SH scriptPubkey.

Parameters

- **bytes** – Bytes to create a scriptPubkey for.
- **bytes_len** – Length of `bytes` in bytes.
- **flags** – `WALLY_SCRIPT_HASH160` or 0.
- **bytes_out** – Destination for the resulting scriptPubkey.

- **len** – The length of `bytes_out` in bytes. If `WALLY_SCRIPT_HASH160` is given, `bytes` is a script to hash160 before creating the P2SH. Otherwise, bytes_len must be `HASH160_LEN` and `bytes` must contain the hash160 to use.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_scriptpubkey_multisig_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *threshold*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a multisig scriptPubkey.

Parameters

- **bytes** – Compressed public keys to create a scriptPubkey from.
- **bytes_len** – Length of `bytes` in bytes. Must be a multiple of `EC_PUBLIC_KEY_LEN`.
- **threshold** – The number of signatures that must match to satisfy the script.
- **flags** – Must be `WALLY_SCRIPT_MULTISIG_SORTED` for BIP67 sorting or 0.
- **bytes_out** – Destination for the resulting scriptPubkey.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Note: A maximum of 15 keys are allowed to be passed.

Returns See *Variable Length Output Buffers*

int **wally_scriptsig_multisig_from_bytes**(const unsigned char **script*, size_t *script_len*, const unsigned char **bytes*, size_t *bytes_len*, const uint32_t **sighash*, size_t *sighash_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a multisig scriptSig.

Parameters

- **script** – The redeem script this scriptSig provides signatures for.
- **script_len** – The length of `script` in bytes.
- **bytes** – Compact signatures to place in the scriptSig.
- **bytes_len** – Length of `bytes` in bytes. Must be a multiple of `EC_SIGNATURE_LEN`.
- **sighash** – `WALLY_SIGHASH_` flags for each signature in `bytes`.
- **sighash_len** – The number of sighash flags in `sighash`.
- **flags** – Must be zero.
- **bytes_out** – Destination for the resulting scriptSig.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_witness_multisig_from_bytes**(const unsigned char **script*, size_t *script_len*, const unsigned char **bytes*, size_t *bytes_len*, const uint32_t **sighash*, size_t *sighash_len*, uint32_t *flags*, struct wally_tx_witness_stack ***witness*)

Create a multisig scriptWitness.

Parameters

- **script** – The witness script this scriptWitness provides signatures for.
- **script_len** – The length of `script` in bytes.
- **bytes** – Compact signatures to place in the scriptWitness.
- **bytes_len** – Length of `bytes` in bytes. Must be a multiple of `EC_SIGNATURE_LEN`.
- **sighash** – `WALLY_SIGHASH_` flags for each signature in `bytes`.
- **sighash_len** – The number of sighash flags in `sighash`.
- **flags** – Must be zero.
- **witness** – Destination for newly allocated witness.

Returns See *Error Codes*

int **wally_scriptpubkey_csv_2of2_then_1_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *csv_blocks*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a CSV 2of2 multisig with a single key recovery scriptPubkey.

The resulting output can be spent at any time with both of the two keys given, and by the last (recovery) key alone, `csv_blocks` after the output confirms.

Parameters

- **bytes** – Compressed public keys to create a scriptPubkey from. The second key given will be used as the recovery key.
- **bytes_len** – Length of `bytes` in bytes. Must 2 * `EC_PUBLIC_KEY_LEN`.
- **csv_blocks** – The number of blocks before the recovery key can be used. Must be between 17 and 65536.
- **flags** – Must be zero.
- **bytes_out** – Destination for the resulting scriptPubkey.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_scriptpubkey_csv_2of2_then_1_from_bytes_opt**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *csv_blocks*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create an optimised CSV 2of2 multisig with a single key recovery scriptPubkey.

Works like `wally_scriptpubkey_csv_2of2_then_1_from_bytes` but produces a script that is smaller and compatible with the miniscript expression

"and(pk(key_user),or(99@pk(key_service),older(<csv_blocks>)))". :return: See *Variable Length Output Buffers*

int **wally_scriptpubkey_csv_2of3_then_2_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *csv_blocks*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a CSV 2of3 multisig with two key recovery scriptPubkey.

The resulting output can be spent at any time with any two of the three keys given, and by either of the last two (recovery) keys alone, `csv_blocks` after the output confirms.

Parameters

- **bytes** – Compressed public keys to create a scriptPubkey from. The second and third keys given will be used as the recovery keys.
- **bytes_len** – Length of `bytes` in bytes. Must 3 * `EC_PUBLIC_KEY_LEN`.
- **csv_blocks** – The number of blocks before the recovery keys can be used. Must be between 17 and 65536.
- **flags** – Must be zero.
- **bytes_out** – Destination for the resulting scriptPubkey.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_script_push_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a bitcoin script that pushes data to the stack.

Parameters

- **bytes** – Bytes to create a push script for.
- **bytes_len** – Length of `bytes` in bytes.
- **flags** – `WALLY_SCRIPT_HASH160` or `WALLY_SCRIPT_SHA256` to hash `bytes` before pushing it.
- **bytes_out** – Destination for the resulting push script.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_varint_get_length**(uint64_t *value*, size_t **written*)

Get the length of an integer serialized to a varint.

Parameters

- **value** – The integer value to be find the length of.
- **written** – Destination for the length of the integer when serialized.

Returns See *Error Codes*

int **wally_varint_to_bytes**(uint64_t *value*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Serialize an integer to a buffer as a varint.

Parameters

- **value** – The integer value to be serialized.
- **bytes_out** – Destination for the resulting serialized varint.
- **len** – The length of `bytes_out` in bytes. Must be at least wally_varint_get_length(value).
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_varbuff_get_length**(const unsigned char **bytes*, size_t *bytes_len*, size_t **written*)

Get the length of a buffer serialized to a varbuff (varint size, followed by the buffer).

Parameters

- **bytes** – The buffer to get the length of.
- **bytes_len** – Length of `bytes` in bytes.
- **written** – Destination for the length of the buffer when serialized.

Returns See *Error Codes*

int **wally_varbuff_to_bytes**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Serialize a buffer to a varbuff (varint size, followed by the buffer).

Parameters

- **bytes** – The buffer to be serialized.
- **bytes_len** – Length of `bytes` in bytes.
- **bytes_out** – Destination for the resulting serialized varbuff.
- **len** – The length of `bytes_out` in bytes. Must be at least wally_varbuff_get_length(bytes, bytes_len).
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_witness_program_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a segwit witness program from a script or hash.

Parameters

- **bytes** – Script or hash bytes to create a witness program from.
- **bytes_len** – Length of `bytes` in bytes. Must be `HASH160_LEN` or `SHA256_LEN` if neither `WALLY_SCRIPT_HASH160` or `WALLY_SCRIPT_SHA256` is given.
- **flags** – `WALLY_SCRIPT_HASH160` or `WALLY_SCRIPT_SHA256` to hash the input script before using it. `WALLY_SCRIPT_AS_PUSH` to generate a push of the generated script as used for the scriptSig in p2sh-p2wpkh and p2sh-p2wsh.
- **bytes_out** – Destination for the resulting witness program.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_witness_program_from_bytes_and_version**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *version*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a segwit witness program from a script or hash using witness version.

Parameters

- **bytes** – Script or hash bytes to create a witness program from.
- **bytes_len** – Length of `bytes` in bytes.
- **version** – Witness version to create a witness program from. Specify a value of 16 or less.
- **flags** – `WALLY_SCRIPT_HASH160` or `WALLY_SCRIPT_SHA256` to hash the input script before using it. `WALLY_SCRIPT_AS_PUSH` to generate a push of the generated script as used for the scriptSig in p2sh-p2wpkh and p2sh-p2wsh.
- **bytes_out** – Destination for the resulting witness program.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_elements_pegout_script_size**(size_t *genesis_blockhash_len*, size_t *mainchain_script_len*, size_t *sub_pubkey_len*, size_t *whitelistproof_len*, size_t **written*)

Get the pegout script size.

Parameters

- **genesis_blockhash_len** – Length of `genesis_blockhash` in bytes. Must be `SHA256_LEN`.
- **mainchain_script_len** – Length of `mainchain_script` in bytes.
- **sub_pubkey_len** – Length of `sub_pubkey` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **whitelistproof_len** – The length of `whitelistproof` in bytes.
- **written** – Destination for the number of bytes required to hold the pegout script.

Returns See *Error Codes*

int **wally_elements_pegout_script_from_bytes**(const unsigned char **genesis_blockhash*, size_t *genesis_blockhash_len*, const unsigned char **mainchain_script*, size_t *mainchain_script_len*, const unsigned char **sub_pubkey*, size_t *sub_pubkey_len*, const unsigned char **whitelistproof*, size_t *whitelistproof_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a pegout script.

Parameters

- **genesis_blockhash** – The genesis blockhash of the parent chain.
- **genesis_blockhash_len** – Length of `genesis_blockhash` in bytes. Must be `SHA256_LEN`.

- **mainchain_script** – The parent chain script.
- **mainchain_script_len** – Length of `mainchain_script` in bytes.
- **sub_pubkey** – The whitelisted public key.
- **sub_pubkey_len** – Length of `sub_pubkey` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **whitelistproof** – The whitelist proof.
- **whitelistproof_len** – The length of `whitelistproof` in bytes.
- **flags** – Must be zero.
- **bytes_out** – Destination for the resulting pegout script.
- **len** – The length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_elements_pegin_contract_script_from_bytes**(const unsigned char *redeem_script*, size_t *redeem_script_len*, const unsigned char **script*, size_t *script_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Create a script for P2CH pegin transactions.

Parameters

- **redeem_script** – The federation redeem script.
- **redeem_script_len** – Length of `redeem_script` in bytes.
- **script** – The claim script.
- **script_len** – Length of `script` in bytes.
- **flags** – Must be zero.
- **bytes_out** – Destination for the resulting script.
- **len** – Length of `bytes_out` in bytes.
- **written** – Destination for the number of bytes written to `bytes_out`.

Returns See *Variable Length Output Buffers*

CHAPTER 9

Symmetric Functions

int **wally_symmetric_key_from_seed**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

Create a new symmetric parent key from entropy.

Parameters

- **bytes** – Entropy to use.
- **bytes_len** – Size of `bytes` in bytes. Must be one of `BIP32_ENTROPY_LEN_128`, `BIP32_ENTROPY_LEN_256` or `BIP32_ENTROPY_LEN_512`.
- **bytes_out** – Destination for the resulting parent key.
- **len** – Size of `bytes_out` in bytes. Must be `HMAC_SHA512_LEN`.

Returns See *Error Codes*

int **wally_symmetric_key_from_parent**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *version*, const unsigned char **label*, size_t *label_len*, unsigned char **bytes_out*, size_t *len*)

Create a new child symmetric key from a parent key.

Parameters

- **bytes** – Parent key to use.
- **bytes_len** – Size of `bytes` in bytes. Must be `HMAC_SHA512_LEN`.
- **version** – Version byte to prepend to `label`. Must be zero.
- **label** – Label to use for the child.
- **label_len** – Size of `label` in bytes.
- **bytes_out** – Destination for the resulting key.
- **len** – Size of `bytes_out` in bytes. Must be `HMAC_SHA512_LEN`.

Returns See *Error Codes*

CHAPTER 10

Transaction Functions

int **wally_tx_witness_stack_init_alloc**(size_t *allocation_len*, struct wally_tx_witness_stack **output*)

Allocate and initialize a new witness stack.

Parameters

- **allocation_len** – The number of items to pre-allocate space for.
- **output** – Destination for the resulting witness stack.

Returns See *Error Codes*

int **wally_tx_witness_stack_clone_alloc**(const struct wally_tx_witness_stack **stack*, struct wally_tx_witness_stack **output*)

Create a copy of a witness stack.

Parameters

- **stack** – The witness stack to copy.
- **output** – Destination for the resulting copy.

Returns See *Error Codes*

int **wally_tx_witness_stack_add**(struct wally_tx_witness_stack **stack*, const unsigned char **witness*, size_t *witness_len*)

Add a witness to a witness stack.

Parameters

- **stack** – The witness stack to add to.
- **witness** – The witness data to add to the stack.
- **witness_len** – Length of `witness` in bytes.

Returns See *Error Codes*

int **wally_tx_witness_stack_add_dummy**(struct wally_tx_witness_stack **stack*, uint32_t *flags*)

Add a dummy witness item to a witness stack.

Parameters

- **stack** – The witness stack to add to.
- **flags** – WALLY_TX_DUMMY_ Flags indicating the type of dummy to add.

Returns See *Error Codes*

int **wally_tx_witness_stack_set**(struct wally_tx_witness_stack **stack*, size_t *index*, const unsigned char **witness*, size_t *witness_len*)

Set a witness item to a witness stack.

Parameters

- **stack** – The witness stack to add to.
- **index** – Index of the item to set. The stack will grow if needed to this many items.
- **witness** – The witness data to add to the stack.
- **witness_len** – Length of witness in bytes.

Returns See *Error Codes*

int **wally_tx_witness_stack_set_dummy**(struct wally_tx_witness_stack **stack*, size_t *index*, uint32_t *flags*)

Set a dummy witness item to a witness stack.

Parameters

- **stack** – The witness stack to add to.
- **index** – Index of the item to set. The stack will grow if needed to this many items.
- **flags** – WALLY_TX_DUMMY_ Flags indicating the type of dummy to set.

Returns See *Error Codes*

int **wally_tx_witness_stack_free**(struct wally_tx_witness_stack **stack*)

Free a transaction witness stack allocated by *wally_tx_witness_stack_init_alloc*.

Parameters

- **stack** – The transaction witness stack to free.

Returns See *Error Codes*

int **wally_tx_input_init_alloc**(const unsigned char **txhash*, size_t *txhash_len*, uint32_t *utxo_index*, uint32_t *sequence*, const unsigned char **script*, size_t *script_len*, const struct wally_tx_witness_stack **witness*, struct wally_tx_input ***output*)

Allocate and initialize a new transaction input.

Parameters

- **txhash** – The transaction hash of the transaction this input comes from.
- **txhash_len** – Size of txhash in bytes. Must be WALLY_TXHASH_LEN.
- **utxo_index** – The zero-based index of the transaction output in txhash that this input comes from.
- **sequence** – The sequence number for the input.
- **script** – The scriptSig for the input.
- **script_len** – Size of script in bytes.
- **witness** – The witness stack for the input, or NULL if no witness is present.

- **output** – Destination for the resulting transaction input.

Returns See *Error Codes*

int **wally_tx_input_free**(struct wally_tx_input *input)
Free a transaction input allocated by `wally_tx_input_init_alloc`.

Parameters

- **input** – The transaction input to free.

Returns See *Error Codes*

int **wally_tx_output_init**(uint64_t *satoshi*, const unsigned char *script, size_t *script_len*, struct wally_tx_output *output)
Initialize a new transaction output.

:param satoshi The amount of the output in satoshi. :param script: The scriptPubkey for the output. :param script_len: Size of `script` in bytes. :param output: Transaction output to initialize. :return: See *Error Codes*

int **wally_tx_output_init_alloc**(uint64_t *satoshi*, const unsigned char *script, size_t *script_len*, struct wally_tx_output **output)
Allocate and initialize a new transaction output.

:param satoshi The amount of the output in satoshi. :param script: The scriptPubkey for the output. :param script_len: Size of `script` in bytes. :param output: Destination for the resulting transaction output. :return: See *Error Codes*

int **wally_tx_output_clone_alloc**(const struct wally_tx_output *tx_output_in, struct wally_tx_output **output)
Create a new copy of a transaction output.

Parameters

- **tx_output_in** – The transaction output to clone.
- **output** – Destination for the resulting transaction output copy.

Returns See *Error Codes*

int **wally_tx_output_clone**(const struct wally_tx_output *tx_output_in, struct wally_tx_output *output)
Create a new copy of a transaction output in place.

Parameters

- **tx_output_in** – The transaction output to clone.
- **output** – Destination for the resulting transaction output copy.

Note: `output` is overwritten in place, and not cleared first.

Returns See *Error Codes*

int **wally_tx_output_free**(struct wally_tx_output *output)
Free a transaction output allocated by `wally_tx_output_init_alloc`.

Parameters

- **output** – The transaction output to free.

Returns See *Error Codes*

int **wally_tx_init_alloc**(uint32_t *version*, uint32_t *locktime*, size_t *inputs_allocation_len*, size_t *outputs_allocation_len*, struct wally_tx **output*)

Allocate and initialize a new transaction.

Parameters

- **version** – The version of the transaction.
- **locktime** – The locktime of the transaction.
- **inputs_allocation_len** – The number of inputs to pre-allocate space for.
- **outputs_allocation_len** – The number of outputs to pre-allocate space for.
- **output** – Destination for the resulting transaction output.

Returns See *Error Codes*

int **wally_tx_clone_alloc**(const struct wally_tx **tx*, uint32_t *flags*, struct wally_tx **output*)

Create a new copy of a transaction.

Parameters

- **tx** – The transaction to clone.
- **flags** – Flags controlling transaction creation. Must be 0.
- **output** – Destination for the resulting transaction copy.

Returns See *Error Codes*

int **wally_tx_add_input**(struct wally_tx **tx*, const struct wally_tx_input **input*)

Add a transaction input to a transaction.

Parameters

- **tx** – The transaction to add the input to.
- **input** – The transaction input to add to `tx`.

Returns See *Error Codes*

int **wally_tx_add_input_at**(struct wally_tx **tx*, uint32_t *index*, const struct wally_tx_input **input*)

Add a transaction input to a transaction at a given position.

Parameters

- **tx** – The transaction to add the input to.
- **index** – The zero-based index of the position to add the input at.
- **input** – The transaction input to add to `tx`.

Returns See *Error Codes*

int **wally_tx_add_raw_input**(struct wally_tx **tx*, const unsigned char **txhash*, size_t *txhash_len*, uint32_t *utxo_index*, uint32_t *sequence*, const unsigned char **script*, size_t *script_len*, const struct wally_tx_witness_stack **witness*, uint32_t *flags*)

Add a transaction input to a transaction.

Parameters

- **tx** – The transaction to add the input to.
- **txhash** – The transaction hash of the transaction this input comes from.
- **txhash_len** – Size of `txhash` in bytes. Must be `WALLY_TXHASH_LEN`.

- **utxo_index** – The zero-based index of the transaction output in `txhash` that this input comes from.
- **sequence** – The sequence number for the input.
- **script** – The scriptSig for the input.
- **script_len** – Size of `script` in bytes.
- **witness** – The witness stack for the input, or NULL if no witness is present.
- **flags** – Flags controlling input creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_add_raw_input_at**(struct wally_tx **tx*, uint32_t *index*, const unsigned char **txhash*, size_t *txhash_len*, uint32_t *utxo_index*, uint32_t *sequence*, const unsigned char **script*, size_t *script_len*, const struct wally_tx_witness_stack **witness*, uint32_t *flags*)

Add a transaction input to a transaction in a goven position.

Parameters

- **tx** – The transaction to add the input to.
- **index** – The zero-based index of the position to add the input at.
- **txhash** – The transaction hash of the transaction this input comes from.
- **txhash_len** – Size of `txhash` in bytes. Must be `WALLY_TXHASH_LEN`.
- **utxo_index** – The zero-based index of the transaction output in `txhash` that this input comes from.
- **sequence** – The sequence number for the input.
- **script** – The scriptSig for the input.
- **script_len** – Size of `script` in bytes.
- **witness** – The witness stack for the input, or NULL if no witness is present.
- **flags** – Flags controlling input creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_remove_input**(struct wally_tx **tx*, size_t *index*)

Remove a transaction input from a transaction.

Parameters

- **tx** – The transaction to remove the input from.
- **index** – The zero-based index of the input to remove.

Returns See *Error Codes*

int **wally_tx_set_input_script**(const struct wally_tx **tx*, size_t *index*, const unsigned char **script*, size_t *script_len*)

Set the scriptsig for an input in a transaction.

Parameters

- **tx** – The transaction to operate on.
- **index** – The zero-based index of the input to set the script on.
- **script** – The scriptSig for the input.

- **script_len** – Size of `script` in bytes.

Returns See *Error Codes*

int **wally_tx_set_input_witness**(const struct wally_tx **tx*, size_t *index*, const struct wally_tx_witness_stack **stack*)

Set the witness stack for an input in a transaction.

Parameters

- **tx** – The transaction to operate on.
- **index** – The zero-based index of the input to set the witness stack on.
- **stack** – The transaction witness stack to set.

Returns See *Error Codes*

int **wally_tx_add_output**(struct wally_tx **tx*, const struct wally_tx_output **output*)

Add a transaction output to a transaction.

Parameters

- **tx** – The transaction to add the output to.
- **output** – The transaction output to add to `tx`.

Returns See *Error Codes*

int **wally_tx_add_output_at**(struct wally_tx **tx*, uint32_t *index*, const struct wally_tx_output **output*)

Add a transaction output to a transaction at a given position.

Parameters

- **tx** – The transaction to add the output to.
- **index** – The zero-based index of the position to add the output at.
- **output** – The transaction output to add to `tx`.

Returns See *Error Codes*

int **wally_tx_add_raw_output**(struct wally_tx **tx*, uint64_t *satoshi*, const unsigned char **script*, size_t *script_len*, uint32_t *flags*)

Add a transaction output to a transaction.

Parameters

- **tx** – The transaction to add the output to.
- **satoshi** – The amount of the output in satoshi.
- **script** – The scriptPubkey for the output.
- **script_len** – Size of `script` in bytes.
- **flags** – Flags controlling output creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_add_raw_output_at**(struct wally_tx **tx*, uint32_t *index*, uint64_t *satoshi*, const unsigned char **script*, size_t *script_len*, uint32_t *flags*)

Add a transaction output to a transaction at a given position.

Parameters

- **tx** – The transaction to add the output to.
- **index** – The zero-based index of the position to add the output at.

- **satoshi** – The amount of the output in satoshi.
- **script** – The scriptPubkey for the output.
- **script_len** – Size of `script` in bytes.
- **flags** – Flags controlling output creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_remove_output**(struct wally_tx **tx*, size_t *index*)
Remove a transaction output from a transaction.

Parameters

- **tx** – The transaction to remove the output from.
- **index** – The zero-based index of the output to remove.

Returns See *Error Codes*

int **wally_tx_get_witness_count**(const struct wally_tx **tx*, size_t **written*)
Get the number of inputs in a transaction that have witness data.

Parameters

- **tx** – The transaction to get the witnesses count from.
- **written** – Destination for the number of witness-containing inputs.

Returns See *Error Codes*

int **wally_tx_free**(struct wally_tx **tx*)
Free a transaction allocated by *wally_tx_init_alloc*.

Parameters

- **tx** – The transaction to free.

Returns See *Error Codes*

int **wally_tx_get_txid**(const struct wally_tx **tx*, unsigned char **bytes_out*, size_t *len*)
Return the txid of a transaction.

Parameters

- **tx** – The transaction to compute the txid of.
- **bytes_out** – Destination for the txid.
- **len** – Size of `bytes_out` in bytes. Must be `WALLY_TXHASH_LEN`.

Note: The txid is expensive to compute.

Returns See *Error Codes*

int **wally_tx_get_length**(const struct wally_tx **tx*, uint32_t *flags*, size_t **written*)
Return the length of transaction once serialized into bytes.

Parameters

- **tx** – The transaction to find the serialized length of.
- **flags** – `WALLY_TX_FLAG_` Flags controlling serialization options.
- **written** – Destination for the length of the serialized bytes.

Returns See *Error Codes*

int **wally_tx_from_bytes**(const unsigned char **bytes*, size_t *bytes_len*, uint32_t *flags*, struct wally_tx ***output*)
Create a transaction from its serialized bytes.

Parameters

- **bytes** – Bytes to create the transaction from.
- **bytes_len** – Length of `bytes` in bytes.
- **flags** – `WALLY_TX_FLAG_` Flags controlling serialization options.
- **output** – Destination for the resulting transaction.

Returns See *Error Codes*

int **wally_tx_from_hex**(const char **hex*, uint32_t *flags*, struct wally_tx ***output*)
Create a transaction from its serialized bytes in hexadecimal.

Parameters

- **hex** – Hexadecimal string containing the transaction.
- **flags** – `WALLY_TX_FLAG_` Flags controlling serialization options.
- **output** – Destination for the resulting transaction.

Returns See *Error Codes*

int **wally_tx_to_bytes**(const struct wally_tx **tx*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*, size_t **written*)
Serialize a transaction to bytes.

Parameters

- **tx** – The transaction to serialize.
- **flags** – `WALLY_TX_FLAG_` Flags controlling serialization options.
- **bytes_out** – Destination for the serialized transaction.
- **len** – Size of `bytes_out` in bytes.
- **written** – Destination for the length of the serialized transaction.

Returns See *Variable Length Output Buffers*

int **wally_tx_to_hex**(const struct wally_tx **tx*, uint32_t *flags*, char ***output*)
Serialize a transaction to hex.

Parameters

- **tx** – The transaction to serialize.
- **flags** – `WALLY_TX_FLAG_` Flags controlling serialization options.
- **output** – Destination for the resulting hexadecimal string.

Note: The string returned should be freed using *`wally_free_string`*.

Returns See *Error Codes*

int **wally_tx_get_weight**(const struct wally_tx **tx*, size_t **written*)
Get the weight of a transaction.

Parameters

- **tx** – The transaction to get the weight of.
- **written** – Destination for the weight.

Returns See *Error Codes*

int **wally_tx_get_vsize**(const struct wally_tx **tx*, size_t **written*)
Get the virtual size of a transaction.

Parameters

- **tx** – The transaction to get the virtual size of.
- **written** – Destination for the virtual size.

Returns See *Error Codes*

int **wally_tx_vsize_from_weight**(size_t *weight*, size_t **written*)
Compute transaction vsize from transaction weight.

Parameters

- **weight** – The weight to convert to a virtual size.
- **written** – Destination for the virtual size.

Returns See *Error Codes*

int **wally_tx_get_total_output_satoshi**(const struct wally_tx **tx*, uint64_t **value_out*)
Compute the total sum of all outputs in a transaction.

Parameters

- **tx** – The transaction to compute the total from.
- **value_out** – Destination for the output total.

Returns See *Error Codes*

int **wally_tx_get_btc_signature_hash**(const struct wally_tx **tx*, size_t *index*, const unsigned char **script*, size_t *script_len*, uint64_t *satoshi*, uint32_t *sighash*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)
Create a BTC transaction for signing and return its hash.

Parameters

- **tx** – The transaction to generate the signature hash from.
- **index** – The input index of the input being signed for.
- **script** – The (unprefixed) scriptCode for the input being signed.
- **script_len** – Size of `script` in bytes.
- **satoshi** – The amount spent by the input being signed for. Only used if flags includes `WALLY_TX_FLAG_USE_WITNESS`, pass 0 otherwise.
- **sighash** – `WALLY_SIGHASH_` flags specifying the type of signature desired.
- **flags** – `WALLY_TX_FLAG_USE_WITNESS` to generate a BIP 143 signature, or 0 to generate a pre-segwit Bitcoin signature.
- **bytes_out** – Destination for the signature hash.
- **len** – Size of `bytes_out` in bytes. Must be at least `SHA256_LEN`.

Returns See *Error Codes*

int **wally_tx_get_signature_hash**(const struct wally_tx **tx*, size_t *index*, const unsigned char **script*, size_t *script_len*, const unsigned char **extra*, size_t *extra_len*, uint32_t *extra_offset*, uint64_t *satoshi*, uint32_t *sighash*, uint32_t *tx_sighash*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Create a transaction for signing and return its hash.

Parameters

- **tx** – The transaction to generate the signature hash from.
- **index** – The input index of the input being signed for.
- **script** – The (unprefixed) scriptCode for the input being signed.
- **script_len** – Size of `script` in bytes.
- **extra** – Extra bytes to include in the transaction preimage.
- **extra_len** – Size of `extra` in bytes.
- **extra_offset** – Offset with the preimage to store `extra`. To store it at the end of the preimage, use 0xffffffff.
- **satoshi** – The amount spent by the input being signed for. Only used if flags includes `WALLY_TX_FLAG_USE_WITNESS`, pass 0 otherwise.
- **sighash** – `WALLY_SIGHASH_` flags specifying the type of signature desired.
- **tx_sighash** – The 32bit sighash value to include in the preimage to hash. This must be given in host CPU endianess; For normal Bitcoin signing the value of `sighash` should be given.
- **flags** – `WALLY_TX_FLAG_USE_WITNESS` to generate a BIP 143 signature, or 0 to generate a pre-segwit Bitcoin signature.
- **bytes_out** – Destination for the signature hash.
- **len** – Size of `bytes_out` in bytes. Must be at least `SHA256_LEN`.

Returns See *Error Codes*

int **wally_tx_is_coinbase**(const struct wally_tx **tx*, size_t **written*)

Determine if a transaction is a coinbase transaction.

Parameters

- **tx** – The transaction to check.
- **written** – 1 if the transaction is a coinbase transaction, otherwise 0.

Returns See *Error Codes*

int **wally_tx_elements_input_issuance_set**(struct wally_tx_input **input*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **entropy*, size_t *entropy_len*, const unsigned char **issuance_amount*, size_t *issuance_amount_len*, const unsigned char **inflation_keys*, size_t *inflation_keys_len*, const unsigned char **issuance_amount_rangeproof*, size_t *issuance_amount_rangeproof_len*, const unsigned char **inflation_keys_rangeproof*, size_t *inflation_keys_rangeproof_len*)

Set issuance data on an input.

Parameters

- **input** – The input to add to.
- **nonce** – Asset issuance or revelation blinding factor.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **entropy** – Entropy for the asset tag calculation.
- **entropy_len** – Size of `entropy` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **issuance_amount** – The (blinded) issuance amount.
- **issuance_amount_len** – Size of `issuance_amount` in bytes.
- **inflation_keys** – The (blinded) token reissuance amount.
- **inflation_keys_len** – Size of `inflation_keys` in bytes.
- **issuance_amount_rangeproof** – Issuance amount rangeproof.
- **issuance_amount_rangeproof_len** – Size of `issuance_amount_rangeproof` in bytes.
- **inflation_keys_rangeproof** – Inflation keys rangeproof.
- **inflation_keys_rangeproof_len** – Size of `inflation_keys_rangeproof` in bytes.

Returns See *Error Codes*

int **wally_tx_elements_input_issuance_free**(struct wally_tx_input **input*)

Free issuance data on an input.

Parameters

- **input** – The input issuance data to free.

Returns See *Error Codes*

int **wally_tx_elements_input_init_alloc**(const unsigned char **txhash*, size_t *txhash_len*, uint32_t *utxo_index*, uint32_t *sequence*, const unsigned char **script*, size_t *script_len*, const struct wally_tx_witness_stack **witness*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **entropy*, size_t *entropy_len*, const unsigned char **issuance_amount*, size_t *issuance_amount_len*, const unsigned char **inflation_keys*, size_t *inflation_keys_len*, const unsigned char **issuance_amount_rangeproof*, size_t *issuance_amount_rangeproof_len*, const unsigned char **inflation_keys_rangeproof*, size_t *inflation_keys_rangeproof_len*, const struct wally_tx_witness_stack **pegin_witness*, struct wally_tx_input ***output*)

Allocate and initialize a new elements transaction input.

Parameters

- **txhash** – The transaction hash of the transaction this input comes from.
- **txhash_len** – Size of `txhash` in bytes. Must be `WALLY_TXHASH_LEN`.

- **utxo_index** – The zero-based index of the transaction output in `txhash` that this input comes from.
- **sequence** – The sequence number for the input.
- **script** – The scriptSig for the input.
- **script_len** – Size of `script` in bytes.
- **witness** – The witness stack for the input, or NULL if no witness is present.
- **nonce** – Asset issuance or revelation blinding factor.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **entropy** – Entropy for the asset tag calculation.
- **entropy_len** – Size of `entropy` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **issuance_amount** – The (blinded) issuance amount.
- **issuance_amount_len** – Size of `issuance_amount` in bytes.
- **inflation_keys** – The (blinded) token reissuance amount.
- **inflation_keys_len** – Size of `inflation_keys` in bytes.
- **issuance_amount_rangeproof** – Issuance amount rangeproof.
- **issuance_amount_rangeproof_len** – Size of `issuance_amount_rangeproof` in bytes.
- **inflation_keys_rangeproof** – Inflation keys rangeproof.
- **inflation_keys_rangeproof_len** – Size of `inflation_keys_rangeproof` in bytes.
- **pegin_witness** – The pegin witness stack for the input, or NULL if no witness is present.
- **output** – Destination for the resulting transaction input.

Returns See *Error Codes*

int **wally_tx_elements_input_is_pegin**(const struct wally_tx_input **input*, size_t **written*)

Determine if an input is a pegin.

Parameters

- **input** – The input to check.
- **written** – 1 if the input is a pegin, otherwise 0.

Returns See *Error Codes*

int **wally_tx_elements_output_commitment_set**(struct wally_tx_output **output*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **value*, size_t *value_len*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **surjectionproof*, size_t *surjectionproof_len*, const unsigned char **rangeproof*, size_t *rangeproof_len*)

Set commitment data on an output.

Parameters

- **output** – The output to add to.

- **asset** – The commitment to a possibly blinded asset.
- **asset_len** – Size of `asset` in bytes. Must be `WALLY_TX_ASSET_CT_ASSET_LEN`.
- **value** – The commitment to a possibly blinded value.
- **value_len** – Size of `value` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_LEN` or `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.
- **nonce** – The commitment used to create the nonce (with the blinding key) for the range proof.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_CT_NONCE_LEN`.
- **surjectionproof** – surjection proof.
- **surjectionproof_len** – Size of `surjectionproof` in bytes.
- **rangeproof** – rangeproof.
- **rangeproof_len** – Size of `rangeproof` in bytes.

Returns See *Error Codes*

int **wally_tx_elements_output_commitment_free**(struct wally_tx_output **output*)

Free commitment data on an output.

Parameters

- **output** – The output with the commitment data to free.

Returns See *Error Codes*

int **wally_tx_elements_output_init**(const unsigned char **script*, size_t *script_len*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **value*, size_t *value_len*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **surjectionproof*, size_t *surjectionproof_len*, const unsigned char **rangeproof*, size_t *rangeproof_len*, struct wally_tx_output **output*)

Initialize a new elements transaction output in place.

Parameters

- **script** – The scriptPubkey for the output.
- **script_len** – Size of `script` in bytes.
- **asset** – The asset tag of the output.
- **asset_len** – Size of `asset` in bytes. Must be `WALLY_TX_ASSET_CT_ASSET_LEN`.
- **value** – The commitment to a possibly blinded value.
- **value_len** – Size of `value` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_LEN` or `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.
- **nonce** – The commitment used to create the nonce (with the blinding key) for the range proof.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_CT_NONCE_LEN`.
- **surjectionproof** – The surjection proof.
- **surjectionproof_len** – Size of `surjectionproof` in bytes.
- **rangeproof** – The range proof.
- **rangeproof_len** – Size of `rangeproof` in bytes.

- **output** – Destination for the resulting transaction output copy.

Note: `output` is overwritten in place, and not cleared first.

Returns See *Error Codes*

int **wally_tx_elements_output_init_alloc**(const unsigned char **script*, size_t *script_len*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **value*, size_t *value_len*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **surjectionproof*, size_t *surjectionproof_len*, const unsigned char **rangeproof*, size_t *rangeproof_len*, struct wally_tx_output ***output*)

Allocate and initialize a new elements transaction output.

Parameters

- **script** – The scriptPubkey for the output.
- **script_len** – Size of `script` in bytes.
- **asset** – The asset tag of the output.
- **asset_len** – Size of `asset` in bytes. Must be `WALLY_TX_ASSET_CT_ASSET_LEN`.
- **value** – The commitment to a possibly blinded value.
- **value_len** – Size of `value` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_LEN` or `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.
- **nonce** – The commitment used to create the nonce (with the blinding key) for the range proof.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_CT_NONCE_LEN`.
- **surjectionproof** – The surjection proof.
- **surjectionproof_len** – Size of `surjectionproof` in bytes.
- **rangeproof** – The range proof.
- **rangeproof_len** – Size of `rangeproof` in bytes.
- **output** – Destination for the resulting transaction output.

Returns See *Error Codes*

int **wally_tx_add_elements_raw_input**(struct wally_tx **tx*, const unsigned char **txhash*, size_t *txhash_len*, uint32_t *utxo_index*, uint32_t *sequence*, const unsigned char **script*, size_t *script_len*, const struct wally_tx_witness_stack **witness*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **entropy*, size_t *entropy_len*, const unsigned char **issuance_amount*, size_t *issuance_amount_len*, const unsigned char **inflation_keys*, size_t *inflation_keys_len*, const unsigned char **issuance_amount_rangeproof*, size_t *issuance_amount_rangeproof_len*, const unsigned char **inflation_keys_rangeproof*, size_t *inflation_keys_rangeproof_len*, const struct wally_tx_witness_stack **pegin_witness*, uint32_t *flags*)

Add an elements transaction input to a transaction.

Parameters

- **tx** – The transaction to add the input to.
- **txhash** – The transaction hash of the transaction this input comes from.
- **txhash_len** – Size of `txhash` in bytes. Must be `WALLY_TXHASH_LEN`.
- **utxo_index** – The zero-based index of the transaction output in `txhash` that this input comes from.
- **sequence** – The sequence number for the input.
- **script** – The scriptSig for the input.
- **script_len** – Size of `script` in bytes.
- **witness** – The witness stack for the input, or NULL if no witness is present.
- **nonce** – Asset issuance or revelation blinding factor.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **entropy** – Entropy for the asset tag calculation.
- **entropy_len** – Size of `entropy` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **issuance_amount** – The (blinded) issuance amount.
- **issuance_amount_len** – Size of `issuance_amount` in bytes.
- **inflation_keys** – The (blinded) token reissuance amount.
- **inflation_keys_len** – Size of `inflation_keys` in bytes.
- **issuance_amount_rangeproof** – Issuance amount rangeproof.
- **issuance_amount_rangeproof_len** – Size of `issuance_amount_rangeproof` in bytes.
- **inflation_keys_rangeproof** – Inflation keys rangeproof.
- **inflation_keys_rangeproof_len** – Size of `inflation_keys_rangeproof` in bytes.
- **pegin_witness** – The pegin witness stack for the input, or NULL if no witness is present.
- **flags** – Flags controlling input creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_add_elements_raw_input_at**(struct wally_tx *tx, uint32_t *index*, const unsigned char *txhash, size_t *txhash_len*, uint32_t *utxo_index*, uint32_t *sequence*, const unsigned char *script, size_t *script_len*, const struct wally_tx_witness_stack *witness, const unsigned char *nonce, size_t *nonce_len*, const unsigned char *entropy, size_t *entropy_len*, const unsigned char *issuance_amount, size_t *issuance_amount_len*, const unsigned char *inflation_keys, size_t *inflation_keys_len*, const unsigned char *issuance_amount_rangeproof, size_t *issuance_amount_rangeproof_len*, const unsigned char *inflation_keys_rangeproof, size_t *inflation_keys_rangeproof_len*, const struct wally_tx_witness_stack *pegin_witness, uint32_t *flags*)

Add an elements transaction input to a transaction at a given position.

Parameters

- **tx** – The transaction to add the input to.
- **index** – The zero-based index of the position to add the input at.
- **txhash** – The transaction hash of the transaction this input comes from.
- **txhash_len** – Size of `txhash` in bytes. Must be `WALLY_TXHASH_LEN`.
- **utxo_index** – The zero-based index of the transaction output in `txhash` that this input comes from.
- **sequence** – The sequence number for the input.
- **script** – The scriptSig for the input.
- **script_len** – Size of `script` in bytes.
- **witness** – The witness stack for the input, or NULL if no witness is present.
- **nonce** – Asset issuance or revelation blinding factor.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **entropy** – Entropy for the asset tag calculation.
- **entropy_len** – Size of `entropy` in bytes. Must be `WALLY_TX_ASSET_TAG_LEN`.
- **issuance_amount** – The (blinded) issuance amount.
- **issuance_amount_len** – Size of `issuance_amount` in bytes.
- **inflation_keys** – The (blinded) token reissuance amount.
- **inflation_keys_len** – Size of `inflation_keys` in bytes.
- **issuance_amount_rangeproof** – Issuance amount rangeproof.
- **issuance_amount_rangeproof_len** – Size of `issuance_amount_rangeproof` in bytes.
- **inflation_keys_rangeproof** – Inflation keys rangeproof.
- **inflation_keys_rangeproof_len** – Size of `inflation_keys_rangeproof` in bytes.
- **pegin_witness** – The pegin witness stack for the input, or NULL if no witness is present.

- **flags** – Flags controlling input creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_add_elements_raw_output**(struct wally_tx **tx*, const unsigned char **script*, size_t *script_len*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **value*, size_t *value_len*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **surjectionproof*, size_t *surjectionproof_len*, const unsigned char **rangeproof*, size_t *rangeproof_len*, uint32_t *flags*)

Add a elements transaction output to a transaction.

Parameters

- **tx** – The transaction to add the output to.
- **script** – The scriptPubkey for the output.
- **script_len** – Size of `script` in bytes.
- **asset** – The asset tag of the output.
- **asset_len** – Size of `asset` in bytes. Must be `WALLY_TX_ASSET_CT_ASSET_LEN`.
- **value** – The commitment to a possibly blinded value.
- **value_len** – Size of `value` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_LEN` or `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.
- **nonce** – The commitment used to create the nonce (with the blinding key) for the range proof.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_CT_NONCE_LEN`.
- **surjectionproof** – The surjection proof.
- **surjectionproof_len** – Size of `surjectionproof` in bytes.
- **rangeproof** – The range proof.
- **rangeproof_len** – Size of `rangeproof` in bytes.
- **flags** – Flags controlling output creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_add_elements_raw_output_at**(struct wally_tx **tx*, uint32_t *index*, const unsigned char **script*, size_t *script_len*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **value*, size_t *value_len*, const unsigned char **nonce*, size_t *nonce_len*, const unsigned char **surjectionproof*, size_t *surjectionproof_len*, const unsigned char **rangeproof*, size_t *rangeproof_len*, uint32_t *flags*)

Add a elements transaction output to a transaction at a given position.

Parameters

- **tx** – The transaction to add the output to.
- **index** – The zero-based index of the position to add the output at.
- **script** – The scriptPubkey for the output.
- **script_len** – Size of `script` in bytes.

- **asset** – The asset tag of the output.
- **asset_len** – Size of `asset` in bytes. Must be `WALLY_TX_ASSET_CT_ASSET_LEN`.
- **value** – The commitment to a possibly blinded value.
- **value_len** – Size of `value` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_LEN` or `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.
- **nonce** – The commitment used to create the nonce (with the blinding key) for the range proof.
- **nonce_len** – Size of `nonce` in bytes. Must be `WALLY_TX_ASSET_CT_NONCE_LEN`.
- **surjectionproof** – The surjection proof.
- **surjectionproof_len** – Size of `surjectionproof` in bytes.
- **rangeproof** – The range proof.
- **rangeproof_len** – Size of `rangeproof` in bytes.
- **flags** – Flags controlling output creation. Must be 0.

Returns See *Error Codes*

int **wally_tx_is_elements**(const struct wally_tx **tx*, size_t **written*)

Determine if a transaction is an elements transaction.

Parameters

- **tx** – The transaction to check.
- **written** – 1 if the transaction is an elements transaction, otherwise 0.

Returns See *Error Codes*

int **wally_tx_confidential_value_from_satoshi**(uint64_t *satoshi*, unsigned char **bytes_out*, size_t *len*)

Convert satoshi to an explicit confidential value representation.

Parameters

- **satoshi** – The value in satoshi to convert.
- **bytes_out** – Destination for the confidential value bytes.
- **len** – Size of `bytes_out` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.

Returns See *Error Codes*

int **wally_tx_confidential_value_to_satoshi**(const unsigned char **value*, size_t *value_len*, uint64_t **value_out*)

Convert an explicit confidential value representation to satoshi.

Parameters

- **value** – The confidential value bytes.
- **value_len** – Size of `value` in bytes. Must be `WALLY_TX_ASSET_CT_VALUE_UNBLIND_LEN`.
- **value_out** – The converted value in satoshi.

Returns See *Error Codes*

int **wally_tx_get_elements_signature_hash**(const struct wally_tx **tx*, size_t *index*, const unsigned char **script*, size_t *script_len*, const unsigned char **value*, size_t *value_len*, uint32_t *sighash*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Create a Elements transaction for signing and return its hash.

Parameters

- **tx** – The transaction to generate the signature hash from.
- **index** – The input index of the input being signed for.
- **script** – The (unprefixed) scriptCode for the input being signed.
- **script_len** – Size of `script` in bytes.
- **value** – The (confidential) value spent by the input being signed for. Only used if flags includes `WALLY_TX_FLAG_USE_WITNESS`, pass NULL otherwise.
- **value_len** – Size of `value` in bytes.
- **sighash** – `WALLY_SIGHASH_` flags specifying the type of signature desired.
- **flags** – `WALLY_TX_FLAG_USE_WITNESS` to generate a BIP 143 signature, or 0 to generate a pre-segwit Bitcoin signature.
- **bytes_out** – Destination for the signature hash.
- **len** – Size of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_tx_elements_issuance_generate_entropy**(const unsigned char **txhash*, size_t *txhash_len*, uint32_t *index*, const unsigned char **contract_hash*, size_t *contract_hash_len*, unsigned char **bytes_out*, size_t *len*)

Calculate the asset entropy from a prevout and the Ricardian contract hash.

Parameters

- **txhash** – The prevout transaction hash.
- **txhash_len** – Size of `txhash` in bytes. Must be `WALLY_TXHASH_LEN`.
- **utxo_index** – The zero-based index of the transaction output in `txhash` to use.
- **contract_hash** – The issuer specified Ricardian contract hash.
- **contract_hash_len** – Size of `contract hash` in bytes. Must be `SHA256_LEN`.
- **bytes_out** – Destination for the asset entropy.
- **len** – Size of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_tx_elements_issuance_calculate_asset**(const unsigned char **entropy*, size_t *entropy_len*, unsigned char **bytes_out*, size_t *len*)

Calculate the asset from the entropy.

Parameters

- **entropy** – The asset entropy.
- **entropy_len** – Size of `entropy` in bytes. Must be `SHA256_LEN`.

- **bytes_out** – Destination for the asset tag.
- **len** – Size of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

int **wally_tx_elements_issuance_calculate_reissuance_token**(const unsigned char **entropy*, size_t *entropy_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Calculate a re-issuance token from an asset's entropy.

Parameters

- **entropy** – The asset entropy.
- **entropy_len** – Size of `entropy` in bytes. Must be `SHA256_LEN`.
- **flags** – `WALLY_TX_FLAG_BLINDED_INITIAL_ISSUANCE` if initial issuance was blinded, pass 0 otherwise.
- **bytes_out** – Destination for the re-issuance token.
- **len** – Size of `bytes_out` in bytes. Must be `SHA256_LEN`.

Returns See *Error Codes*

CHAPTER 11

Elements Functions

int **wally_asset_generator_from_bytes**(const unsigned char **asset*, size_t *asset_len*, const unsigned char **abf*, size_t *abf_len*, unsigned char **bytes_out*, size_t *len*)

Create a blinded Asset Generator from an Asset Tag and Asset Blinding Factor.

Parameters

- **asset** – Asset Tag to create a blinding generator for.
- **asset_len** – Length of `asset` in bytes. Must be `ASSET_TAG_LEN`.
- **abf** – Asset Blinding Factor (Random entropy to blind with).
- **abf_len** – Length of `abf` in bytes. Must be `BLINDING_FACTOR_LEN`.
- **bytes_out** – Destination for the resulting Asset Generator.
- **len** – The length of `bytes_out` in bytes. Must be `ASSET_GENERATOR_LEN`.

Returns See *Error Codes*

int **wally_asset_final_vbf**(const uint64_t **values*, size_t *values_len*, size_t *num_inputs*, const unsigned char **abf*, size_t *abf_len*, const unsigned char **vbf*, size_t *vbf_len*, unsigned char **bytes_out*, size_t *len*)

Generate the final value blinding factor required for blinding a confidential transaction.

Parameters

- **values** – Array of transaction input values in satoshi
- **values_len** – Length of `values`, also the number of elements in all three of the input arrays, which is equal to `num_inputs` plus the number of outputs.
- **num_inputs** – Number of elements in the input arrays that represent transaction inputs. The number of outputs is implicitly `values_len` - `num_inputs`.
- **abf** – Array of bytes representing `values_len` asset blinding factors.
- **abf_len** – Length of `abf` in bytes. Must be `values_len` * `BLINDING_FACTOR_LEN`.

- **vbf** – Array of bytes representing (values_len - 1) value blinding factors.
- **vbf_len** – Length of vbf in bytes. Must be (values_len - 1) * BLINDING_FACTOR_LEN.
- **bytes_out** – Buffer to receive the final value blinding factor.
- **len** – Length of bytes_out. Must be BLINDING_FACTOR_LEN.

Returns See *Error Codes*

int **wally_asset_value_commitment**(uint64_t *value*, const unsigned char **vbf*, size_t *vbf_len*, const unsigned char **generator*, size_t *generator_len*, unsigned char **bytes_out*, size_t *len*)

Calculate the value commitment for a transaction output.

Parameters

- **value** – Output value in satoshi.
- **vbf** – Value Blinding Factor.
- **vbf_len** – Length of vbf. Must be BLINDING_FACTOR_LEN.
- **generator** – Asset generator from *wally_asset_generator_from_bytes*.
- **generator_len** – Length of generator. Must be ASSET_GENERATOR_LEN.
- **bytes_out** – Buffer to receive value commitment.
- **len** – Length of bytes_out. Must be ASSET_COMMITMENT_LEN.

Returns See *Error Codes*

int **wally_asset_rangeproof**(uint64_t *value*, const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **abf*, size_t *abf_len*, const unsigned char **vbf*, size_t *vbf_len*, const unsigned char **commitment*, size_t *commitment_len*, const unsigned char **extra*, size_t *extra_len*, const unsigned char **generator*, size_t *generator_len*, uint64_t *min_value*, int *exp*, int *min_bits*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Generate a rangeproof for a transaction output.

Parameters

- **value** – Value of the output in satoshi.
- **pub_key** – Public blinding key for the output. See *wally_confidential_addr_to_ec_public_key*.
- **pub_key_len** – Length of pub_key. Must be EC_PUBLIC_KEY_LEN
- **priv_key** – Pivate ephemeral key. Should be randomly generated for each output.
- **priv_key_length** – Length of priv_key.
- **asset** – Asset id of output.
- **asset_len** – Length of asset. Must be ASSET_TAG_LEN.
- **abf** – Asset blinding factor. Randomly generated for each output.
- **abf_len** – Length of abf. Must be BLINDING_FACTOR_LEN.
- **vbf** – Value blinding factor. Randomly generated for each output except the last, which is generate by calling *wally_asset_final_vbf*.
- **vbf_len** – Length of vbf. Must be BLINDING_FACTOR_LEN.

- **commitment** – Value commitment from *wally_asset_value_commitment*.
- **commitment_len** – Length of `commitment`. Must be `ASSET_COMMITMENT_LEN`.
- **extra** – Set this to the script pubkey of the output.
- **extra_len** – Length of `extra`, i.e. script pubkey.
- **generator** – Asset generator from *wally_asset_generator_from_bytes*.
- **generator_len** – Length of `generator`. Must be `ASSET_GENERATOR_LEN`.
- **min_value** – Recommended value 1.
- **exp** – Exponent value. -1 >= `exp` >= 18. Recommended value 0.
- **min_bits** – 0 >= min_bits >= 64. Recommended value 52.
- **bytes_out** – Buffer to receive rangeproof.
- **len** – Length of `bytes_out`. See `ASSET_RANGEPROOF_MAX_LEN`.
- **written** – Number of bytes actually written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_asset_surjectionproof_size**(size_t *num_inputs*, size_t **written*)

Return the required buffer size for receiving a surjection proof

Parameters

- **num_inputs** – Number of transaction inputs.
- **written** – Destination for the surjection proof size.

Returns See *Error Codes*

int **wally_asset_surjectionproof**(const unsigned char **output_asset*, size_t *output_asset_len*, const unsigned char **output_abf*, size_t *output_abf_len*, const unsigned char **output_generator*, size_t *output_generator_len*, const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **asset*, size_t *asset_len*, const unsigned char **abf*, size_t *abf_len*, const unsigned char **generator*, size_t *generator_len*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Generate a surjection proof for a transaction output

Parameters

- **output_asset** – asset id for the output.
- **output_asset_len** – Length of `asset`. Must be `ASSET_TAG_LEN`.
- **output_abf** – Asset blinding factor for the output. Generated randomly for each output.
- **output_abf_len** – Length of `output_abf`. Must be `BLINDING_FACTOR_LEN`.
- **output_generator** – Asset generator from *wally_asset_generator_from_bytes*.
- **output_generator_len** – Length of `output_generator`. Must be `ASSET_GENERATOR_LEN`.
- **bytes** – Must be generated randomly for each output.
- **bytes_len** – Length of `bytes`. Must be 32.
- **asset** – Array of input asset tags.

- **asset_len** – Length of `asset`. Must be ``ASSET_TAG_LEN * number of inputs.
- **abf** – Array of asset blinding factors from the transaction inputs.
- **abf_len** – Length of `abf`. Must be `BLINDING_FACTOR_LEN` * number of inputs.
- **generator** – Array of asset generators from transaction inputs.
- **generator_len** – Length of `generator`. Must be `ASSET_GENERATOR_LEN` * number of inputs.
- **bytes_out** – Buffer to receive surjection proof.
- **bytes_out_len** – Length of `bytes_out`. See *wally_asset_surjectionproof_size*.
- **written** – Number of bytes actually written to `bytes_out`.

Returns See *Variable Length Output Buffers*

int **wally_asset_unblind_with_nonce**(const unsigned char **nonce_hash*, size_t *nonce_hash_len*, const unsigned char **proof*, size_t *proof_len*, const unsigned char **commitment*, size_t *commitment_len*, const unsigned char **extra*, size_t *extra_len*, const unsigned char **generator*, size_t *generator_len*, unsigned char **asset_out*, size_t *asset_out_len*, unsigned char **abf_out*, size_t *abf_out_len*, unsigned char **vbf_out*, size_t *vbf_out_len*, uint64_t **value_out*)

Unblind a confidential transaction output.

Parameters

- **nonce_hash** – SHA-256 hash of the generated nonce.
- **nonce_hash_len** – Length of `nonce_hash`. Must be `SHA256_LEN`.
- **proof** – Rangeproof from `wally_tx_get_output_rangeproof()`.
- **proof_len** – Length of `proof`.
- **commitment** – Value commitment from `wally_tx_get_output_value()`.
- **commitment_len** – Length of `commitment`.
- **extra** – Script pubkey from `wally_tx_get_output_script()`.
- **extra_len** – Length of `extra`.
- **generator** – Asset generator from `wally_tx_get_output_asset()`.
- **generator_len** – Length of `generator`. Must be `ASSET_GENERATOR_LEN`.
- **asset_out** – Buffer to receive unblinded asset id.
- **asset_out_len** – Length of `asset_out`. Must be `ASSET_TAG_LEN`.
- **abf_out** – Buffer to receive asset blinding factor.
- **abf_out_len** – Length of `abf_out`. Must be `BLINDING_FACTOR_LEN`.
- **vbf_out** – Buffer to receive asset blinding factor.
- **vbf_out_len** – Length of `vbf_out`. Must be `BLINDING_FACTOR_LEN`.
- **value_out** – Destination for unblinded transaction output value.

Returns See *Error Codes*

int **wally_asset_unblind**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **proof*, size_t *proof_len*, const unsigned char **commitment*, size_t *commitment_len*, const unsigned char **extra*, size_t *extra_len*, const unsigned char **generator*, size_t *generator_len*, unsigned char **asset_out*, size_t *asset_out_len*, unsigned char **abf_out*, size_t *abf_out_len*, unsigned char **vbf_out*, size_t *vbf_out_len*, uint64_t **value_out*)

Unblind a confidential transaction output.

Parameters

- **pub_key** – From `wally_tx_get_output_nonce()`.
- **pub_key_len** – Length of `pub_key`. Must be `EC_PUBLIC_KEY_LEN`.
- **priv_key** – Private blinding key corresponding to public blinding key used to generate destination address. See *wally_asset_blinding_key_to_ec_private_key()*.
- **proof** – Rangeproof from `wally_tx_get_output_rangeproof()`.
- **proof_len** – Length of `proof`.
- **commitment** – Value commitment from `wally_tx_get_output_value()`.
- **commitment_len** – Length of `commitment`.
- **extra** – Script pubkey from `wally_tx_get_output_script()`.
- **extra_len** – Length of `extra`.
- **generator** – Asset generator from `wally_tx_get_output_asset()`.
- **generator_len** – Length of `generator`. Must be `ASSET_GENERATOR_LEN`.
- **asset_out** – Buffer to receive unblinded asset id.
- **asset_out_len** – Length of `asset_out`. Must be `ASSET_TAG_LEN`.
- **abf_out** – Buffer to receive asset blinding factor.
- **abf_out_len** – Length of `abf_out`. Must be `BLINDING_FACTOR_LEN`.
- **vbf_out** – Buffer to receive asset blinding factor.
- **vbf_out_len** – Length of `vbf_out`. Must be `BLINDING_FACTOR_LEN`.
- **value_out** – Destination for unblinded transaction output value.

Returns See *Error Codes*

int **wally_asset_blinding_key_from_seed**(const unsigned char **bytes*, size_t *bytes_len*, unsigned char **bytes_out*, size_t *len*)

Generate a master blinding key from a seed as specified in SLIP-0077.

Parameters

- **bytes** – Seed value. See *bip39_mnemonic_to_seed()*.
- **bytes_len** – Length of `seed`. Must be one of `BIP32_ENTROPY_LEN_128`, `BIP32_ENTROPY_LEN_256` or `BIP32_ENTROPY_LEN_512`.
- **bytes_out** – Buffer to receive master blinding key. The master blinding key can be used to generate blinding keys for specific outputs by passing it to *wally_asset_blinding_key_to_ec_private_key*.
- **len** – Length of `bytes_out`. Must be `HMAC_SHA512_LEN`.

Returns See *Error Codes*

int **wally_asset_blinding_key_to_ec_private_key**(const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **script*, size_t *script_len*, unsigned char **bytes_out*, size_t *len*)

Generate a blinding key for a script pubkey.

Parameters

- **bytes** – Master blinding key from *wally_asset_blinding_key_from_seed*.
- **bytes_len** – Length of `bytes`. Must be `HMAC_SHA512_LEN`.
- **script** – The script pubkey for the confidential output address.
- **script_len** – Length of `script`.
- **bytes_out** – Buffer to receive blinding key.
- **len** – Length of `bytes_out`. Must be `EC_PRIVATE_KEY_LEN`.

Returns See *Error Codes*

int **wally_asset_pak_whitelistproof_size**(size_t *num_keys*, size_t **written*)

Calculate the size in bytes of the whitelist proof.

Parameters

- **num_keys** – The number of offline/online keys.
- **written** – Destination for the number of bytes needed for the proof.

Returns See *Error Codes*

int **wally_asset_pak_whitelistproof**(const unsigned char **online_keys*, size_t *online_keys_len*, const unsigned char **offline_keys*, size_t *offline_keys_len*, size_t *key_index*, const unsigned char **sub_pubkey*, size_t *sub_pubkey_len*, const unsigned char **online_priv_key*, size_t *online_priv_key_len*, const unsigned char **summed_key*, size_t *summed_key_len*, unsigned char **bytes_out*, size_t *len*, size_t **written*)

Generate the whitelist proof for the pegout script.

Parameters

- **online_keys** – The list of online keys.
- **online_keys_len** – Length of `online_keys_len` in bytes. Must be a multiple of `EC_PUBLIC_KEY_LEN`.
- **offline_keys** – The list of offline keys.
- **offline_keys_len** – Length of `offline_keys_len` in bytes. Must be a multiple of `EC_PUBLIC_KEY_LEN`.
- **key_index** – The index in the PAK list of the key signing this whitelist proof
- **sub_pubkey** – The key to be whitelisted.
- **sub_pubkey_len** – Length of `sub_pubkey` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **online_priv_key** – The secret key to the signer's online pubkey.
- **online_priv_key_len** – Length of `online_priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **summed_key** – The secret key to the sum of (whitelisted key, signer's offline pubkey).

- **summed_key_len** – Length of `summed_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes_out** – Destination for the resulting whitelist proof.
- **len** – Length of `bytes_out` in bytes.
- **written** – Number of bytes actually written to `bytes_out`.

Returns See *Variable Length Output Buffers*

CHAPTER 12

Anti-Exfil Functions

int **wally_ae_host_commit_from_bytes**(const unsigned char **entropy*, size_t *entropy_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Create the initial commitment to host randomness.

Parameters

- **entropy** – Randomness to commit to. It must come from a cryptographically secure RNG. As per the protocol, this value must not be revealed to the client until after the host has received the client commitment.
- **entropy_len** – The length of `entropy` in bytes. Must be `WALLY_S2C_DATA_LEN`.
- **flags** – Must be `EC_FLAG_ECDSA`.
- **bytes_out** – Destination for the resulting compact signature.
- **len** – The length of `bytes_out` in bytes. Must be `WALLY_HOST_COMMITMENT_LEN`.

Returns See *Error Codes*

int **wally_ae_signer_commit_from_bytes**(const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **commitment*, size_t *commitment_len*, uint32_t *flags*, unsigned char **s2c_opening_out*, size_t *s2c_opening_out_len*)

Compute signer's original nonce.

Parameters

- **priv_key** – The private key used for signing.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes** – The message hash to be signed.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **commitment** – Randomness commitment from the host.

- **commitment_len** – The length of `commitment` in bytes. Must be `WALLY_HOST_COMMITMENT_LEN`.
- **flags** – Must be `EC_FLAG_ECDSA`.
- **s2c_opening_out** – Destination for the resulting opening information.
- **s2c_opening_out_len** – The length of `s2c_opening_out` in bytes. Must be `WALLY_S2C_OPENING_LEN`.

Returns See *Error Codes*

int **wally_ae_sig_from_bytes**(const unsigned char **priv_key*, size_t *priv_key_len*, const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **entropy*, size_t *entropy_len*, uint32_t *flags*, unsigned char **bytes_out*, size_t *len*)

Same as `wally_ec_sig_from_bytes`, but commits to the host randomness.

Parameters

- **priv_key** – The private key to sign with.
- **priv_key_len** – The length of `priv_key` in bytes. Must be `EC_PRIVATE_KEY_LEN`.
- **bytes** – The message hash to sign.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **entropy** – Host provided randomness.
- **entropy_len** – The length of `entropy` in bytes. Must be `WALLY_S2C_DATA_LEN`.
- **flags** – Must be `EC_FLAG_ECDSA`.
- **bytes_out** – Destination for the resulting compact signature.
- **len** – The length of `bytes_out` in bytes. Must be `EC_SIGNATURE_LEN`.

Returns See *Error Codes*

int **wally_ae_verify**(const unsigned char **pub_key*, size_t *pub_key_len*, const unsigned char **bytes*, size_t *bytes_len*, const unsigned char **entropy*, size_t *entropy_len*, const unsigned char **s2c_opening*, size_t *s2c_opening_len*, uint32_t *flags*, const unsigned char **sig*, size_t *sig_len*)

Verify a signature was correctly constructed using the Anti-Exfil Protocol.

Parameters

- **pub_key** – The public key to verify with.
- **pub_key_len** – The length of `pub_key` in bytes. Must be `EC_PUBLIC_KEY_LEN`.
- **bytes** – The message hash to verify.
- **bytes_len** – The length of `bytes` in bytes. Must be `EC_MESSAGE_HASH_LEN`.
- **entropy** – Randomness provided by the host.
- **entropy_len** – The length of `entropy` in bytes. Must be `WALLY_S2C_DATA_LEN`.
- **s2c_opening** – Opening information provided by the signer.
- **s2c_opening_len** – The length of `s2c_opening` in bytes. Must be `WALLY_S2C_OPENING_LEN`.
- **flags** – Must be `EC_FLAG_ECDSA`.
- **sig** – The compact signature of the message in `bytes`.

- **sig_len** – The length of `sig` in bytes. Must be `EC_SIGNATURE_LEN`.

Returns See *Error Codes*

CHAPTER 13

Library Conventions

13.1 Error Codes

The following values can be returned by library functions:

Code	Meaning
`WALLY_OK`	The function completed without error. See *Variable Length Output Buffers* if applicable for the function.
`WALLY_ERROR`	An internal or unexpected error happened in the library. In some cases this code may indicate a specific error condition which will be documented with the function.
`WALLY_EINVAL`	One or more parameters passed to the function is not valid. For example, a required buffer value is NULL or of the wrong length.
`WALLY_ENOMEM`	The function required memory allocation but no memory could be allocated from the O/S.

13.2 Variable Length Output Buffers

Some functions write output that can vary in length to user supplied buffers. In these cases, the number of written bytes is placed in the `written` output parameter when the function completes.

If the user supplied buffer is of insufficient size, these functions will still return `WALLY_OK`, but will place the required size in the `written` output parameter.

Callers must check not only that the function succeeds, but also that the number of bytes written is less than or equal to the supplied buffer size. If the buffer is too small, it should be resized to the returned size and the call retried.

CHAPTER 14

Liquid

The following walkthrough demonstrates how to use libwally to create a transaction spending a confidential liquid utxo. For documentation of the Blockstream Liquid network please go to Blockstream

The example code here is written in python using the generated python swig wrappers.

14.1 Generating a confidential address

Start by creating a standard p2pkh address. Assume that we have defined `mnemonic` as a 24 word mnemonic for the wallet we want to use. From this we can derive bip32 keys depending on the requirements of the wallet.

```
_, seed = wally.bip39_mnemonic_to_seed512(mnemonic, '')
wallet_master_key = wally.bip32_key_from_seed(
    seed,
    wally.BIP32_VER_TEST_PRIVATE, 0)
wallet_derived_key = wally.bip32_key_from_parent(
    wallet_master_key,
    1,
    wally.BIP32_FLAG_KEY_PRIVATE)
address = wally.bip32_key_to_address(
    wallet_derived_key,
    wally.WALLY_ADDRESS_TYPE_P2PKH,
    wally.WALLY_ADDRESS_VERSION_P2PKH_LIQUID_REGTEST)
```

For each new receive address a blinding key should be deterministically derived from a master blinding key, itself derived from the bip39 mnemonic for the wallet. wally provides the function *wally_asset_blinding_key_from_seed()* which can be used to derive a master blinding key from a mnemonic.

```
master_blinding_key = wally.asset_blinding_key_from_seed(seed)
script_pubkey = wally.address_to_scriptpubkey(
    address,
    wally.WALLY_NETWORK_LIQUID_REGTEST)
```

(continues on next page)

(continued from previous page)

```
private_blinding_key = wally.asset_blinding_key_to_ec_private_key(
    master_blinding_key,
    script_pubkey)
public_blinding_key = wally.ec_public_key_from_private_key(
    private_blinding_key)
```

Finally call the wally function *wally_confidential_addr_from_addr()* to combine the non-confidential address with the public blinding key to create a confidential address. We also supply a blinding prefix indicating the network version.

```
confidential_address = wally.confidential_addr_from_addr(
    address,
    wally.WALLY_CA_PREFIX_LIQUID_REGTEST,
    public_blinding_key)
```

The confidential address can now be passed to liquid-cli to receive funds. We'll send 1.1 BTC to our confidential address and save the raw hex transaction for further processing.

```
$ liquid-cli getrawtransaction $(sendtoaddress <confidential address> 1.1)
```

14.2 Receiving confidential assets

On receiving confidential (blinded) utxos you can use libwally to unblind and inspect them. Take the hex transaction returned by getrawtransaction above and create a libwally tx.

```
tx = wally.tx_from_hex(
    tx_hex,
    wally.WALLY_TX_FLAG_USE_ELEMENTS | wally.WALLY_TX_FLAG_USE_WITNESS)
```

The transaction will likely have three outputs: the utxo paying to our confidential address, a change output and an explicit fee output(Liquid transactions differ from standard bitcoin transaction in that the fee is an explicit output). Iterate over the transaction outputs and unblind any addressed to us.

```
asset_generators_in = b''
asset_ids_in = b''
values_in = []
abfs_in = b''
vbfs_in = b''
script_pubkeys_in = []
vouts_in = []
num_outputs = wally.tx_get_num_outputs(tx)
for vout in range(num_outputs):
    script_pubkey_out = wally.tx_get_output_script(tx, vout)
    if script_pubkey_out != script_pubkey:
        continue

    vouts_in.append(vout)

    sender_ephemeral_pubkey = wally.tx_get_output_nonce(tx, vout)
    rangeproof = wally.tx_get_output_rangeproof(tx, vout)
    script_pubkey = wally.tx_get_output_script(tx, vout)
    asset_id = wally.tx_get_output_asset(tx, vout)
    value_commitment = wally.tx_get_output_value(tx, vout)
```

(continues on next page)

(continued from previous page)

```
    script_pubkeys_in.append(script_pubkey)

    value, asset_id, abf, vbf = wally.asset_unblind(
        sender_ephemeral_pubkey,
        private_blinding_key,
        rangeproof,
        value_commitment,
        script_pubkey,
        asset_id)

    asset_generator = wally.asset_generator_from_bytes(asset_id, abf)

    asset_generators_in += asset_generator
    asset_ids_in += asset_id
    values_in.append(value)
    abfs_in += abf
    vbfs_in += vbf
```

We have now unblinded the values and asset ids of the utxos. We've also saved the abfs (asset blinding factors) and vbfs (value binding factors) because they are needed for the next step: spending the utxos.

14.3 Spending confidential assets

The wallet logic will define the transaction outputs, values and fees. Here we assume that we're only dealing with a single asset and a single confidential recipient address `destination_address` to which we'll pay the input amount less some fixed `fee`.

```
total_in = sum(values_in)
output_values = [total_in - fee,]
confidential_output_addresses = [destination_address,]
output_asset_ids = asset_ids_in[:wally.ASSET_TAG_LEN]
```

Generate blinding factors for the outputs. These are asset blinding factors (abf) and value blinding factors (vbf). The blinding factors are random except for the final vbf which must be calculated by calling *wally_asset_final_vbf()*.

```
num_inputs = len(values_in)
num_outputs = 1
abfs_out = os.urandom(32 * num_outputs)
vbfs_out = os.urandom(32 * (num_outputs - 1))
vbfs_out += wally.asset_final_vbf(
    values_in + output_values,
    num_inputs,
    abfs_in + abfs_out,
    vbfs_in + vbfs_out)
```

A confidential output address can be decomposed into a standard address plus the public blinding key, and the libwally function *wally_address_to_scriptpubkey()* will provide the corresponding script pubkey.

```
blinding_pubkeys = [
    wally.confidential_addr_to_ec_public_key(
        confidential_address, address_prefix)
    for confidential_address in confidential_output_addresses]
```

(continues on next page)

(continued from previous page)

```
non_confidential_addresses = [
    wally.confidential_addr_to_addr(
        confidential_address, address_prefix)
    for confidential_address in confidential_output_addresses]

script_pubkeys = [
    wally.address_to_scriptpubkey(
        non_confidential_address, network)
    for non_confidential_address in non_confidential_addresses]
```

Create a new transaction

```
version = 2
locktime = 0
output_tx = wally.tx_init(version, locktime, 0, 0)
```

Iterate over the outputs and calculate the value commitment, rangeproof and surjectionproofs. This requires generating a random ephemeral public/private ec key pair for each blinded output.

```
for value, blinding_pubkey, script_pubkey in zip(
        output_values, blinding_pubkeys, script_pubkeys):

    abf, abfs_out = abfs_out[:32], abfs_out[32:]
    vbf, vbfs_out = vbfs_out[:32], vbfs_out[32:]
    asset_id, output_asset_ids = output_asset_ids[:32], output_asset_ids[32:]

    generator = wally.asset_generator_from_bytes(asset_id, abf)

    value_commitment = wally.asset_value_commitment(
        value, vbf, generator)

    ephemeral_privkey = os.urandom(32)
    ephemeral_pubkey = wally.ec_public_key_from_private_key(
        ephemeral_privkey)

    rangeproof = wally.asset_rangeproof(
        value,
        blinding_pubkey,
        ephemeral_privkey,
        asset_id,
        abf,
        vbf,
        value_commitment,
        script_pubkey,
        generator,
        1, # min_value
        0, # exponent
        36 # bits
    )

    surjectionproof = wally.asset_surjectionproof(
        asset_id,
        abf,
        generator,
        os.urandom(32),
```

(continues on next page)

(continued from previous page)

```
        asset_ids_in,
        abfs_in,
        asset_generators_in
    )

    wally.tx_add_elements_raw_output(
        output_tx,
        script_pubkey,
        generator,
        value_commitment,
        ephemeral_pubkey,
        surjectionproof,
        rangeproof,
        0
    )
```

Finally the fee output must be explicitly added (unlike standard Bitcoin transactions where the fee is implicit). The fee is always payable in the bitcoin asset. The wallet logic will determine the fee amount.

```
BITCOIN = "5ac9f65c0efcc4775e0baec4ec03abdde22473cd3cf33c0419ca290e0751b225"
BITCOIN = wally.hex_to_bytes(BITCOIN)[::-1]

wally.tx_add_elements_raw_output(
    output_tx,
    None,
    bytearray([0x01]) + BITCOIN,
    wally.tx_confidential_value_from_satoshi(fee),
    None, # nonce
    None, # surjection proof
    None, # range proof
    0)
```

Sign the transaction inputs.

```
flags = 0
prev_txid = wally.tx_get_txid(tx)
for vout in vouts_in:

    wally.tx_add_elements_raw_input(
        output_tx,
        prev_txid,
        vout,
        0xffffffff,
        None, # scriptSig
        None, # witness
        None, # nonce
        None, # entropy
        None, # issuance amount
        None, # inflation keys
        None, # issuance amount rangeproof
        None, # inflation keys rangeproof
        None, # pegin witness
        flags)

for vin, script_pubkey in enumerate(script_pubkeys_in):
```

(continues on next page)

(continued from previous page)

```
    privkey = wally.bip32_key_get_priv_key(wallet_derived_key)
    pubkey = wally.ec_public_key_from_private_key(privkey)

    sighash = wally.tx_get_elements_signature_hash(
        output_tx, vin, script_pubkey, None, wally.WALLY_SIGHASH_ALL, 0)

    signature = wally.ec_sig_from_bytes(
        privkey, sighash, wally.EC_FLAG_ECDSA)

    scriptsig = wally.scriptsig_p2pkh_from_sig(
        pubkey, signature, wally.WALLY_SIGHASH_ALL)

    wally.tx_set_input_script(output_tx, vin, scriptsig)
```

The transaction is now ready to be broadcast, the hex value is easily retrieved by calling *wally_tx_to_hex()*.

```
tx_hex = wally.tx_to_hex(output_tx, wally.WALLY_TX_FLAG_USE_WITNESS)
```

CHAPTER 15

Anti-Exfil Protocol

The following walkthrough demonstrates how to use libwally to implement the ECDSA Anti-Exfil Protocol to prevent a signing device from exfiltrating the secret signing keys through biased signature nonces. For the full details, see here.

The example code here is written in python using the generated python swig wrappers.

15.1 Step 1

The host draws randomness `rho` and computes a commitment to it:

```
host_commitment = wally.ae_host_commit_from_bytes(rho, wally.EC_FLAG_ECDSA)
```

Host sends `host_commitment` to the signer.

15.2 Step 2

The signing device computes the original nonce `R`, i.e. `signer commitment`:

```
signer_commitment = wally.ae_signer_commit_from_bytes(priv_key, message_hash, host_
↪commitment, wally.EC_FLAG_ECDSA)
```

Signing device sends `signer_commitment` to the host.

Warning: If, at any point from this step onward, the hardware device fails, it is okay to restart the protocol using **exactly the same** `rho` and checking that the hardware device proposes **exactly the same** `R`. Otherwise, the hardware device may be selectively aborting and thereby biasing the set of nonces that are used in actual signatures.

It takes many (>100) such aborts before there is a plausible attack, given current knowledge in 2020. However such aborts accumulate even across a total replacement of all relevant devices (but not across replacement of the actual signing keys with new independently random ones).

In case the hardware device cannot be made to sign with the given `rho`, `R` pair, wallet authors should alert the user and present a very scary message implying that if this happens more than even a few times, say 20 or more times EVER, they should change hardware vendors and perhaps sweep their coins.

15.3 Step 3

The host replies with `rho` generated at *Step 1*.

15.4 Step 4

The signing device signs and provide the signature to the host:

```
signature = wally.ae_sig_from_bytes(priv_key, message_hash, rho, wally.EC_FLAG_ECDSA)
```

15.5 Step 5

The host verifies that the signature's public nonce matches the signer commitment `R` from *Step 2* and its original randomness `rho`:

```
wally.ae_verify(pub_key, message_hash, rho, signer_commitment, wally.EC_FLAG_ECDSA,␣
↪signature)
```

CHAPTER 16

Indices and tables

- genindex
- search

Index

B

W