eth-account Documentation

Release 0.4.0

The Ethereum Foundation

May 31, 2019

Contents

Sign Ethereum transactions and messages with local private keys

CHAPTER 1

Contents

1.1 eth_account

1.1.1 Account

class eth_account.account.**Account**

The primary entry point for working with Ethereum private keys.

It does **not** require a connection to an Ethereum node.

create(*extra_entropy=''*)

Creates a new private key, and returns it as a `LocalAccount`.

Parameters **extra_entropy** (*str or bytes or int*) – Add extra randomness to whatever randomness your OS can provide

Returns an object with private key and convenience methods

```
>>> from eth_account import Account
>>> acct = Account.create('KEYSMASH FJAFJKLDSKF7JKFDJ 1530')
>>> acct.address
'0x5ce9454909639D2D17A3F753ce7d93fa0b9aB12E'
>>> acct.key
b"\xb2\}\xb3\x1f\xee\xd9\x12''\xbf\t9\xdcv\x9a\x96VK-
↪\xe4\xc4rm\x03[6\xec\xf1\xe5\xb3d"

# These methods are also available: sign_message(), sign_transaction(),␣
↪encrypt()
# They correspond to the same-named methods in Account.*
# but without the private key argument
```

static decrypt(*keyfile_json*, *password*)

Decrypts a private key that was encrypted using an Ethereum client or *encrypt()*.

Parameters

- **keyfile_json** (*dict or str*) – The encrypted key
- **password** (*str*) – The password that was used to encrypt the key

Returns the raw private key

Return type HexBytes

```
>>> encrypted = {
 'address': '5ce9454909639d2d17a3f753ce7d93fa0b9ab12e',
 'crypto': {'cipher': 'aes-128-ctr',
  'cipherparams': {'iv': '78f214584844e0b241b433d7c3bb8d5f'},
  'ciphertext':
↪'d6dbb56e4f54ba6db2e8dc14df17cb7352fdce03681dd3f90ce4b6c1d5af2c4f',
  'kdf': 'pbkdf2',
  'kdfparams': {'c': 1000000,
   'dklen': 32,
   'prf': 'hmac-sha256',
   'salt': '45cf943b4de2c05c2c440ef96af914a2'},
  'mac': 'f5e1af09df5ded25c96fcf075ada313fb6f79735a914adc8cb02e8ddee7813c3'},
 'id': 'b812f3f9-78cc-462a-9e89-74418aa27cb0',
 'version': 3}

>>> import getpass
>>> Account.decrypt(encrypted, getpass.getpass())
HexBytes('0xb25c7db31feed9122727bf0939dc769a96564b2de4c4726d035b36ecf1e5b364')
```

classmethod encrypt(*private_key*, *password*, *kdf=None*, *iterations=None*)

Creates a dictionary with an encrypted version of your private key. To import this keyfile into Ethereum clients like geth and parity: encode this dictionary with `json.dumps()` and save it to disk where your client keeps key files.

Parameters

- **private_key** (hex str, bytes, int or `eth_keys.datatypes.PrivateKey`) – The raw private key
- **password** (*str*) – The password which you will need to unlock the account in your client
- **kdf** (*str*) – The key derivation function to use when encrypting your private key
- **iterations** (*int*) – The work factor for the key derivation function

Returns The data to use in your encrypted file

Return type dict

If kdf is not set, the default key derivation function falls back to the environment variable `ETH_ACCOUNT_KDF`. If that is not set, then 'scrypt' will be used as the default.

```
>>> import getpass
>>> encrypted = Account.encrypt(
    0xb25c7db31feed9122727bf0939dc769a96564b2de4c4726d035b36ecf1e5b364,
    getpass.getpass()
)

{
    'address': '5ce9454909639d2d17a3f753ce7d93fa0b9ab12e',
    'crypto': {
        'cipher': 'aes-128-ctr',
```

(continues on next page)

(continued from previous page)

```
        'cipherparams': {
            'iv': '0b7845a5c3597d3d378bde9b7c7319b7'
        },
        'ciphertext':
↪'a494f1feb3c854e99c1ff01e6aaa17d43c0752009073503b908457dc8de5d2a5',  #␣
↪noqa: E501
        'kdf': 'scrypt',
        'kdfparams': {
            'dklen': 32,
            'n': 262144,
            'p': 8,
            'r': 1,
            'salt': '13c4a48123affaa29189e9097726c698'
        },
        'mac':
↪'f4cfb027eb0af9bd7a320b4374a3fa7bef02cfbafe0ec5d1fd7ad129401de0b1'
    },
    'id': 'a60e0578-0e5b-4a75-b991-d55ec6451a6f',
    'version': 3
}

 >>> with open('my-keyfile', 'w') as f:
     f.write(json.dumps(encrypted))
```

from_key (*private_key*)

Returns a convenient object for working with the given private key.

Parameters **private_key** (hex str, bytes, int or `eth_keys.datatypes.PrivateKey`) – The raw private key

Returns object with methods for signing and encrypting

Return type *LocalAccount*

```
>>> acct = Account.from_key(
  0xb25c7db31feed9122727bf0939dc769a96564b2de4c4726d035b36ecf1e5b364)
>>> acct.address
'0x5ce9454909639D2D17A3F753ce7d93fa0b9aB12E'
>>> acct.key
b"\xb2\}\xb3\x1f\xee\xd9\x12''xbf\t9\xdcv\x9a\x96VK-
↪\xe4\xc4rm\x03[6\xec\xf1\xe5\xb3d"

# These methods are also available: sign_message(), sign_transaction(),␣
↪encrypt()
# They correspond to the same-named methods in Account.*
# but without the private key argument
```

privateKeyToAccount (*private_key*)

> **Caution:** Deprecated for *from_key()*. This method will be removed in v0.5

recoverHash (*message_hash*, *vrs=None*, *signature=None*)

Get the address of the account that signed the message with the given hash. You must specify exactly one of: vrs or signature

> **Caution:** Deprecated for *recover_message()*. This method might be removed as early as v0.5

Parameters

- **message_hash** (*hex str or bytes or int*) – the hash of the message that you want to verify
- **vrs** (*tuple(v, r, s), each element is hex str, bytes or int*) – the three pieces generated by an elliptic curve signature
- **signature** (*hex str or bytes or int*) – signature bytes concatenated as r+s+v

Returns address of signer, hex-encoded & checksummed

Return type str

recoverTransaction(*serialized_transaction*)

> **Caution:** Deprecated for *recover_transaction()*. This method will be removed in v0.5

recover_message(*signable_message: eth_account.messages.SignableMessage, vrs=None, signature=None*)

Get the address of the account that signed the given message. You must specify exactly one of: vrs or signature

Parameters

- **signable_message** – the message that was signed
- **vrs** (*tuple(v, r, s), each element is hex str, bytes or int*) – the three pieces generated by an elliptic curve signature
- **signature** (*hex str or bytes or int*) – signature bytes concatenated as r+s+v

Returns address of signer, hex-encoded & checksummed

Return type str

```
>>> from eth_account.messages import encode_defunct
>>> message = encode_defunct(text="ISF")
>>> vrs = (
      28,
      '0xe6ca9bba58c88611fad66a6ce8f996908195593807c4b38bd528d2cff09d4eb3',
      '0x3e5bfbbf4d3e39b1a2fd816a7680c19ebebaf3a141b239934ad43cb33fcec8ce')
>>> Account.recover_message(message, vrs=vrs)
'0x5ce9454909639D2D17A3F753ce7d93fa0b9aB12E'

# All of these recover calls are equivalent:

# variations on vrs
>>> vrs = (
      '0x1c',
      '0xe6ca9bba58c88611fad66a6ce8f996908195593807c4b38bd528d2cff09d4eb3',
      '0x3e5bfbbf4d3e39b1a2fd816a7680c19ebebaf3a141b239934ad43cb33fcec8ce')
>>> Account.recover_message(message, vrs=vrs)
```

(continues on next page)

(continued from previous page)

```
>>> vrs = (
      b'\x1c',
      b
↪'\\xe6\\xca\\x9b\\xbaX\\xc8\\x86\\x11\\xfa\\xd6jl\\xe8\\xf9\\x96\\x90\\x81\\x95Y8\\x07\\xc
↪',  # noqa: E501
      b'>[\\xfb\\xbfM>
↪9\\xb1\\xa2\\xfd\\x81jv\\x80\\xc1\\x9e\\xbe\\xba\\xf3\\xa1A\\xb29\\x93J\\xd4
↪<\\xb3?\\xce\\xc8\\xce')  # noqa: E501
>>> Account.recover_message(message, vrs=vrs)
>>> # Caution about this approach: likely problems if there are leading 0s
>>> vrs = (
      0x1c,
      0xe6ca9bba58c88611fad66a6ce8f996908195593807c4b38bd528d2cff09d4eb3,
      0x3e5bfbbf4d3e39b1a2fd816a7680c19ebebaf3a141b239934ad43cb33fcec8ce)
>>> Account.recover_message(message, vrs=vrs)

# variations on signature
>>> signature =
↪'0xe6ca9bba58c88611fad66a6ce8f996908195593807c4b38bd528d2cff09d4eb33e5bfbbf4d3e39b1a2fd816
↪'  # noqa: E501
>>> Account.recover_message(message, signature=signature)
>>> signature = b
↪'\\xe6\\xca\\x9b\\xbaX\\xc8\\x86\\x11\\xfa\\xd6jl\\xe8\\xf9\\x96\\x90\\x81\\x95Y8\\x07\\xc
↪[\\xfb\\xbfM>
↪9\\xb1\\xa2\\xfd\\x81jv\\x80\\xc1\\x9e\\xbe\\xba\\xf3\\xa1A\\xb29\\x93J\\xd4
↪<\\xb3?\\xce\\xc8\\xce\\x1c'  # noqa: E501
>>> Account.recover_message(message, signature=signature)
>>> # Caution about this approach: likely problems if there are leading 0s
>>> signature =␣
↪0xe6ca9bba58c88611fad66a6ce8f996908195593807c4b38bd528d2cff09d4eb33e5bfbbf4d3e39b1a2fd816a
↪ # noqa: E501
>>> Account.recover_message(message, signature=signature)
```

recover_transaction(*serialized_transaction*)

Get the address of the account that signed this transaction.

Parameters **serialized_transaction** (*hex str, bytes or int*) – the complete signed transaction

Returns address of signer, hex-encoded & checksummed

Return type str

```
>>> raw_transaction =
↪'0xf86a8086d55698372431831e848094f0109fc8df283027b6285cc889f5aa624eac1f55843b9aca008025a00
↪',  # noqa: E501
>>> Account.recover_transaction(raw_transaction)
'0x2c7536E3605D9C16a7a3D7b1898e529396a65c23'
```

setKeyBackend(*backend*)

Caution: Deprecated for *set_key_backend()*. This method will be removed in v0.5

set_key_backend(*backend*)

Change the backend used by the underlying eth-keys library.

(The default is fine for most users)

Parameters **backend** – any backend that works in eth_keys.KeyApi(backend)

signHash(*message_hash*, *private_key*)

> **Warning:** *Never* sign a hash that you didn't generate, it can be an arbitrary transaction. For example, it might send all of your account's ether to an attacker. Instead, prefer *sign_message()*, which cannot accidentally sign a transaction.

Sign the provided hash.

> **Caution:** Deprecated for *sign_message()*. This method will be removed in v0.5

Parameters

- **message_hash** (*hex str, bytes or int*) – the 32-byte message hash to be signed
- **private_key** (hex str, bytes, int or eth_keys.datatypes.PrivateKey) – the key to sign the message with

Returns Various details about the signature - most importantly the fields: v, r, and s

Return type *AttributeDict*

signTransaction(*transaction_dict*, *private_key*)

> **Caution:** Deprecated for *sign_transaction()*. This method will be removed in v0.5

sign_message(*signable_message: eth_account.messages.SignableMessage*, *private_key*)

Sign the provided message.

This API supports any messaging format that will encode to EIP-191 messages.

If you would like historical compatibility with w3.eth.sign() you can use *encode_defunct()*.

Other options are the "validator", or "structured data" standards. (Both of these are in *DRAFT* status currently, so be aware that the implementation is not guaranteed to be stable). You can import all supported message encoders in eth_account.messages.

Parameters

- **signable_message** – the encoded message for signing
- **private_key** (hex str, bytes, int or eth_keys.datatypes.PrivateKey) – the key to sign the message with

Returns Various details about the signature - most importantly the fields: v, r, and s

Return type *AttributeDict*

```
>>> msg = "ISF"
>>> from eth_account.messages import encode_defunct
>>> msghash = encode_defunct(text=msg)
SignableMessage(version=b'E', header=b'thereum Signed Message:\n6', body=b
↪'I\xe2\x99\xa5SF')
>>> # If you're curious about the internal fields of SignableMessage, take a␣
↪look at EIP-191, linked above
>>> key = "0xb25c7db31feed9122727bf0939dc769a96564b2de4c4726d035b36ecf1e5b364"
>>> Account.sign_message(msghash, key)
{'messageHash': HexBytes(
↪'0x1476abb745d423bf09273f1afd887d951181d25adc66c4834a70491911b7f750'),  #␣
↪noqa: E501
 'r':␣
↪104389933075820307925104709181714897380569894203213074526835978196648170704563,
↪
 's':␣
↪28205917190874851400050446352651915501321657673772411533993420917949420456142,
↪
 'signature': HexBytes(
↪'0xe6ca9bba58c88611fad66a6ce8f996908195593807c4b38bd528d2cff09d4eb33e5bfbbf4d3e39b1a2fd816
↪'),  # noqa: E501
 'v': 28}
```

sign_transaction (*transaction_dict*, *private_key*)

Sign a transaction using a local private key. Produces signature details and the hex-encoded transaction suitable for broadcast using `w3.eth.sendRawTransaction()`.

Create the transaction dict for a contract method with my_contract.functions.my_function().buildTransaction()

Parameters

- **transaction_dict** (*dict*) – the transaction with keys: nonce, chainId, to, data, value, gas, and gasPrice.
- **private_key** (hex str, bytes, int or `eth_keys.datatypes.PrivateKey`) – the private key to sign the data with

Returns Various details about the signature - most importantly the fields: v, r, and s

Return type *AttributeDict*

```
>>> transaction = {
        # Note that the address must be in checksum format or native bytes:
        'to': '0xF0109fC8DF283027b6285cc889F5aA624EaC1F55',
        'value': 1000000000,
        'gas': 2000000,
        'gasPrice': 234567897654321,
        'nonce': 0,
        'chainId': 1
    }
>>> key = '0x4c0883a69102937d6231471b5dbb6204fe5129617082792ae468d01a3f362318'
>>> signed = Account.sign_transaction(transaction, key)
{'hash': HexBytes(
↪'0x6893a6ee8df79b0f5d64a180cd1ef35d030f3e296a5361cf04d02ce720d32ec5'),
 'r':␣
↪44872862617934181798178410248897471157793243053758231102491494799050751740 44,
↪
 'rawTransaction': HexBytes(
↪'0xf86a8086d55698372431831e848094f0109fc8df283027b6285cc889f5aa624eac1f55843b9aca008025a00
↪'),  # noqa: E501
```

(continues on next page)

(continued from previous page)

```
 's':␣
↪30785525769477805655994251009256770582792548537338581640010273753578382951464,
↪
 'v': 37}
>>> w3.eth.sendRawTransaction(signed.rawTransaction)
```

See *Signers* for alternative signers.

1.1.2 AttributeDict

class eth_account.datastructures.**AttributeDict**(**args*, ***kwargs*)

Bases: attrdict.dictionary.AttrDict

See AttrDict docs

This class differs only in that it is made immutable. This immutability is **not** a security guarantee. It is only a style-check convenience.

1.1.3 Messages

class eth_account.messages.**SignableMessage**

Bases: tuple

These are the components of an EIP-191 signable message. Other message formats can be encoded into this format for easy signing. This data structure doesn't need to know about the original message format.

In typical usage, you should never need to create these by hand. Instead, use one of the available encode_* methods in this module, like:

- *encode_structured_data()*
- *encode_intended_validator()*
- *encode_structured_data()*

body

Alias for field number 2

header

Alias for field number 1

version

Alias for field number 0

eth_account.messages.**defunct_hash_message**(*primitive: bytes = None*, ***, *hexstr: str = None*, *text: str = None*) → hexbytes.main.HexBytes

Convert the provided message into a message hash, to be signed.

> **Caution:** Intented for use with the deprecated *eth_account.account.Account.signHash()*. This is for backwards compatibility only. All new implementations should use *encode_defunct()* instead.

Parameters

- **primitive** (*bytes or int*) – the binary message to be signed
- **hexstr** (*str*) – the message encoded as hex

- **text** (*str*) – the message as a series of unicode characters (a normal Py3 str)

Returns The hash of the message, after adding the prefix

eth_account.messages.**encode_defunct**(*primitive: bytes = None*, ***, *hexstr: str = None*, *text: str = None*) → eth_account.messages.SignableMessage

Encode a message for signing, using an old, unrecommended approach.

Only use this method if you must have compatibility with `w3.eth.sign()`.

EIP-191 defines this as "version `E`".

Supply exactly one of the three arguments: bytes, a hex string, or a unicode string.

Parameters

- **primitive** (*bytes or int*) – the binary message to be signed
- **hexstr** (*str*) – the message encoded as hex
- **text** (*str*) – the message as a series of unicode characters (a normal Py3 str)

Returns The EIP-191 encoded message, ready for signing

```
>>> from eth_account.messages import encode_defunct

>>> message_text = "ISF"
>>> encode_defunct(text=message_text)
SignableMessage(version=b'E', header=b'thereum Signed Message:\n6', body=b
↪'I\xe2\x99\xa5SF')

# these four also produce the same hash:
>>> encode_defunct(w3.toBytes(text=message_text))
SignableMessage(version=b'E', header=b'thereum Signed Message:\n6', body=b
↪'I\xe2\x99\xa5SF')

>>> encode_defunct(bytes(message_text, encoding='utf-8'))
SignableMessage(version=b'E', header=b'thereum Signed Message:\n6', body=b
↪'I\xe2\x99\xa5SF')

>>> Web3.toHex(text=message_text)
'0x49e299a55346'
>>> encode_defunct(hexstr='0x49e299a55346')
SignableMessage(version=b'E', header=b'thereum Signed Message:\n6', body=b
↪'I\xe2\x99\xa5SF')

>>> encode_defunct(0x49e299a55346)
SignableMessage(version=b'E', header=b'thereum Signed Message:\n6', body=b
↪'I\xe2\x99\xa5SF')
```

eth_account.messages.**encode_intended_validator**(*validator_address: Union[NewType.<locals>.new_type, str]*, *primitive: bytes = None*, ***, *hexstr: str = None*, *text: str = None*) → eth_account.messages.SignableMessage

Encode a message using the "intended validator" approach (ie~ version 0) defined in EIP-191.

Supply the message as exactly one of these three arguments: bytes as a primitive, a hex string, or a unicode string.

Warning: Note that this code has not gone through an external audit. Also, watch for updates to the format, as the EIP is still in DRAFT.

Parameters

- **validator_address** – which on-chain contract is capable of validating this message, provided as a checksummed address or in native bytes.
- **primitive** (*bytes or int*) – the binary message to be signed
- **hexstr** (*str*) – the message encoded as hex
- **text** (*str*) – the message as a series of unicode characters (a normal Py3 str)

Returns The EIP-191 encoded message, ready for signing

eth_account.messages.**encode_structured_data**(*primitive: Union[bytes, int, collections.abc.Mapping] = None, *, hexstr: str = None, text: str = None*) → eth_account.messages.SignableMessage

Encode a message using the "structured data" approach (ie~ version 1) defined in EIP-712.

Supply the message as exactly one of the three arguments:

- primitive, as a dict that defines the structured data
- primitive, as bytes
- text, as a json-encoded string
- hexstr, as a hex-encoded (json-encoded) string

Warning: Note that this code has not gone through an external audit. Also, watch for updates to the format, as the EIP is still in DRAFT.

Parameters

- **primitive** (*bytes or int or Mapping (eg~ dict)*) – the binary message to be signed
- **hexstr** – the message encoded as hex
- **text** – the message as a series of unicode characters (a normal Py3 str)

Returns The EIP-191 encoded message, ready for signing

1.2 Signers

These classes abstract away the private key, as opposed to *eth_account.account.Account*, which explicitly requires the private key on each usage.

All the signer classes in this package must meet the interface specified by *BaseAccount*.

Currently there is only one Local Signer. Some upcoming alternatives to the basic local signer include hierarchical deterministic (HD) wallets and hardware wallets.

1.2.1 Local Signer

class eth_account.signers.local.**LocalAccount**(*key*, *account*)
Bases: *eth_account.signers.base.BaseAccount*

A collection of convenience methods to sign and encrypt, with an embedded private key.

Variables ***key*** (*bytes*) – the 32-byte private key data

```
>>> my_local_account.address
"0xF0109fC8DF283027b6285cc889F5aA624EaC1F55"
>>> my_local_account.key
b"\x01\x23..."
```

You can also get the private key by casting the account to `bytes`:

```
>>> bytes(my_local_account)
b"\\x01\\x23..."
```

address
The checksummed public address for this account.

```
>>> my_account.address
"0xF0109fC8DF283027b6285cc889F5aA624EaC1F55"
```

encrypt(*password*, *kdf=None*, *iterations=None*)
Generate a string with the encrypted key, as in *encrypt()*, but without a private key argument.

key
Get the private key.

privateKey

Caution: Deprecated for **:var:'~eth_account.signers.local.LocalAccount.key'**. This attribute will be removed in v0.5

signHash(*message_hash*)
Sign the hash of a message, as in *signHash()* but without specifying the private key.

Caution: Deprecated for *sign_message()*. To be removed in v0.5

Parameters **message_hash** (*bytes*) – 32 byte hash of the message to sign

signTransaction(*transaction_dict*)

Caution: Deprecated for `sign_transaction()`.

This method will be removed in v0.5

sign_message(*signable_message*)
Generate a string with the encrypted key, as in *sign_message()*, but without a private key argument.

sign_transaction (*transaction_dict*)
Sign a transaction, as in *sign_transaction()* but without specifying the private key.

Parameters **transaction_dict** (*dict*) – transaction with all fields specified

1.2.2 Abstract Signer

class eth_account.signers.base.**BaseAccount**
Bases: abc.ABC

Abstract class that defines a collection of convenience methods to sign transactions and message hashes.

address
The checksummed public address for this account.

```
>>> my_account.address
"0xF0109fC8DF283027b6285cc889F5aA624EaC1F55"
```

signHash (*message_hash*)
Sign the hash of a message, as in *signHash()* but without specifying the private key.

Caution: Deprecated for *sign_message()*. To be removed in v0.5

Parameters **message_hash** (*bytes*) – 32 byte hash of the message to sign

signTransaction (*transaction_dict*)

Caution: Deprecated for sign_transaction(). This method will be removed in v0.5

sign_message (*signable_message: eth_account.messages.SignableMessage*)
Sign the EIP-191 message, as in *sign_message()* but without specifying the private key.

Parameters **signable_message** – The encoded message, ready for signing

sign_transaction (*transaction_dict*)
Sign a transaction, as in *sign_transaction()* but without specifying the private key.

Parameters **transaction_dict** (*dict*) – transaction with all fields specified

1.3 Release Notes

1.3.1 v0.4.0

Released 2019-05-06

- BREAKING CHANGE: drop python 3.5 (and therefore pypy3 support). #60 (includes other housekeeping)
- New message signing API: *sign_message()* and recover_message. #61
  - New *eth_account.messages.encode_intended_validator()* for EIP-191's Intended Validator message-signing format. #56

  - New *eth_account.messages.encode_structured_data()* for EIP-712's Structured Data message-signing format. #57

- Add optional param iterations to *encrypt()* #52
- Add optional param kdf to *encrypt()*, plus env var ETH_ACCOUNT_KDF. Default kdf switched from hmac-sha256 to scrypt. #38
- Accept "to" addresses formatted as bytes in addition to checksummed, hex-encoded. #36

1.3.2 v0.3.0

Released July 24, 2018

- Support eth_keys.datatypes.PrivateKey in params that accept a private key.
- New docs for *Signers*
- Under the hood: add a new *BaseAccount* abstract class, so that upcoming signing classes can implement it (be on the lookout for upcoming hardware wallet support)

1.3.3 v0.2.3

Released May 27, 2018

- Implement __eq__ and __hash__ for *LocalAccount*, so that accounts can be used in set, or as keys in dict, etc.

1.3.4 v0.2.2

Released Apr 25, 2018

- Compatibility with pyrlp v0 and v1

1.3.5 v0.2.1

Released Apr 23, 2018

- Accept 'from' in signTransaction, if it matches the sending private key's address

1.3.6 v0.2.0 (stable)

Released Apr 19, 2018

- Audit cleanup is complete
- Stopped requiring chainId, until tooling to automatically derive it gets better (Not that transactions without chainId are potentially replayable on fork chains)

1.3.7 v0.2.0-alpha.0

Released Apr 6, 2018

- Ability to sign an already-hashed message
- Moved `eth_sign`-style message hashing to *eth_account.messages.defunct_hash_message()*
- Stricter transaction input validation, and better error messages. Including: *to* field must be checksummed.
- PyPy3 support & tests
- Upgrade dependencies
- Moved non-public interfaces to *internal* module
- Documentation
  - use `getpass` instead of typing in password manually
  - *eth_account.signers.local.LocalAccount* attributes
  - readme improvements
  - more

1.3.8 v0.1.0-alpha.2

- Imported the local signing code from web3.py's `w3.eth.account`
- Imported documentation and added more
- Imported tests and pass them

1.3.9 v0.1.0-alpha.1

- Launched repository, claimed names for pip, RTD, github, etc

CHAPTER 2

Indices and tables

- genindex
- modindex

Python Module Index

e

Index

V