Swap

Release 0.2.3

Meheret Tesfaye

Oct 23, 2020

CONTENTS

CHAPTER

ONE

SWAP

Cryptocurrencies were created to make it possible for advanced, encrypted payments to be made between two or more people digitally, without the parties involved having to trust each other for the payment be completed. In other words, cryptocurrencies make it possible to send money reliably to other people over the internet without the money being double spent, and without people getting scammed out of their money when they try to make these digital payments.

Note: Hash Time Lock Contracts (HTLCs) are a perfect example of a payment technology for cryptocurrencies which makes all of the aforementioned things possible.

Swap is a python library for cross-chain atomic swap between the networks of two cryptocurrencies. Cross-chain atomic swap are the cheapest and most secure way to swap cryptocurrencies. It's a brand new decentralized payment environment based on Hash Time Lock Contracts (HTLCs) protocol.

CHAPTER

TWO

WHAT IS A HTLC?

A Hash Time Lock contract (HTLC) is essentially a type of payment in which two people agree to a financial arrangement where one party will pay the other party a certain amount of Cryptocurrency, such as Bitcoin or Bytom assets. However, because these contracts are Time Locked, the receiving party only has a certain amount of time to accept the payment, otherwise the money can be returned to the sender.

Hash time lock contracts can help to eliminate the need for third parties in contracts between two parties. Third parties that are often involved in contracts are lawyers, banks, etc. Lawyers are often required to draw up contracts, and banks are often required to help store money and then transfer it to the receiving party in the contract.

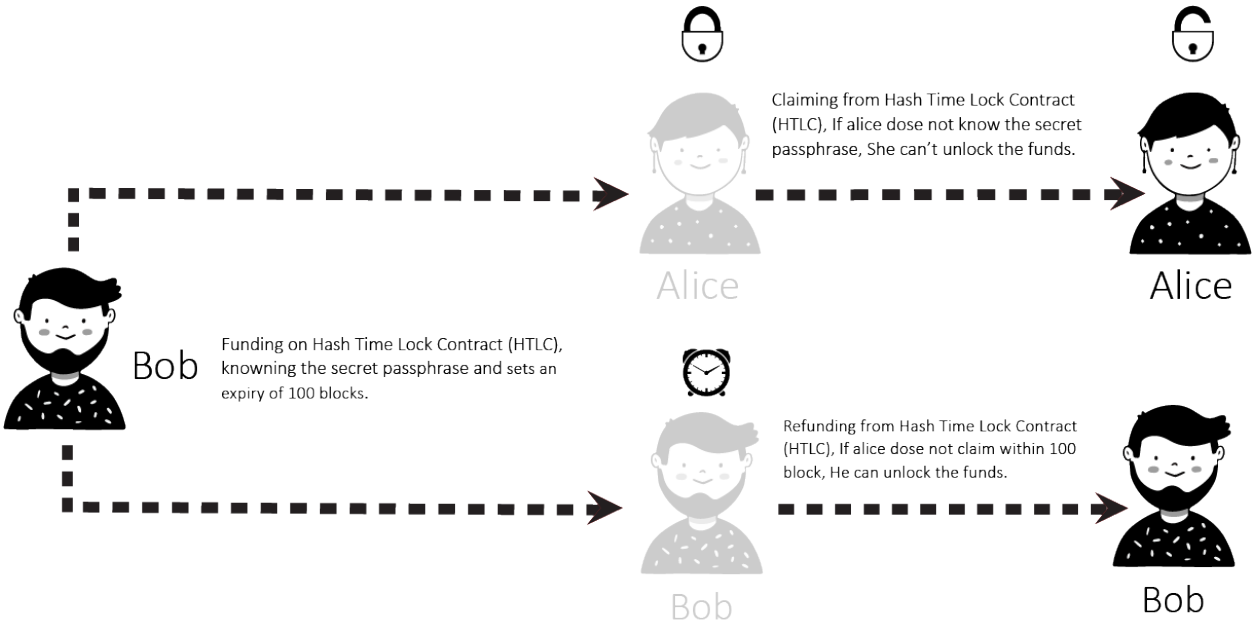
With hash time lock contracts, two parties could hypothetically set up contracts and transfer money without the need for third parties. This is because the sending party could create the conditional payment, and then the receiving party could agree to it, receive it, and help validate the transaction in the process.

This could potentially revolutionize the way that many businesses interact with one another and dramatically speed up the time that it takes for business deals to be set up.

2.1 How do HTLC work?

The way that Hash Time Lock Contracts work is that the person who will be making the payment sets up a specific hash, which represents the amount of money that will be paid. To receive the payment, the recipient will have to create a cryptographic proof of payment, and he or she will have to do this within the specified amount of time. The amount of time that the recipient has to accept the payment can vary significantly from one Time Locked contract to the next. If the recipient meets the deadline, then the money will be theirs, if he or she fails to meet the deadline, it won't. So, there is an often a lot at stake when it comes to meeting deadlines from hash Time Locked contracts, when Cryptocurrencies are being exchanged.

The amount of time that the recipient has to accept the payment can vary significantly from one Time Locked contract to the next. If the recipient meets the deadline, then the money will be theirs, if he or she fails to meet the deadline, it won't. So, there is an often a lot at stake when it comes to meeting deadlines from hash Time Locked contracts, when Cryptocurrencies are being exchanged.

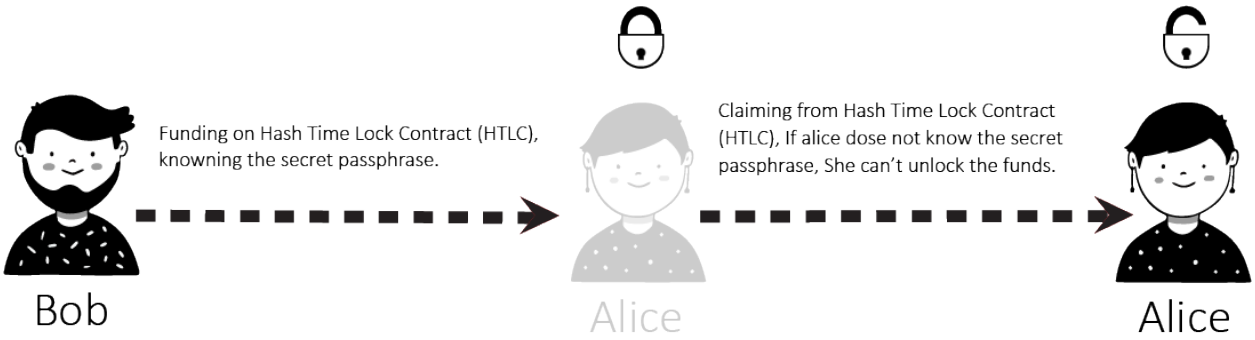


A Hash Time Lock Contract or HTLC is a class of payments that uses Hash Locked and Time Locked to require that the receiver of a payment either acknowledge receiving the payment prior to a deadline by generating cryptographic proof of payment or forfeit the ability to claim the payment, returning(refunding) it to the payer.

Hash Time Lock Contracts (HTLCs) allow payments to be securely routed across multiple payment channels which is super important because it is not optimal for a person to open a payment channel with everyone he/she is transacting with.

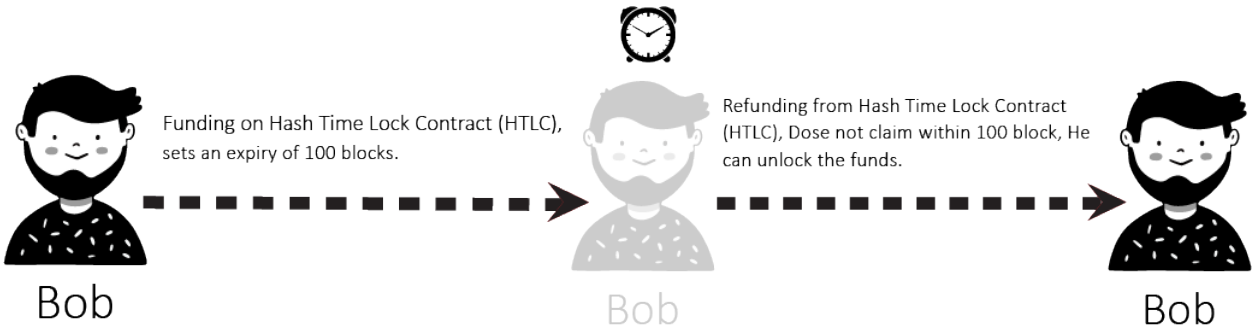
2.1.1 Hash Locked

A Hash Locked functions like "two-factor authentication" (2FA). It requires the intended recipient to provide the correct secret passphrase to claim the funds.



2.1.2 Time Locked

A Time Locked adds a "timeout" expiration date to a payment. It requires the intended recipient to claim the funds prior to the expiry. Otherwise, the transaction defaults to enabling the original sender of funds to claim a refund.



2.2 Benefits of HTLC's

There are many benefits to these types of contracts. First, because they are time sensitive, it prevents the person who is making the payment from having to wait indefinitely to find out whether or not his or her payment goes through. Second, the person who makes the payment will not have to waste his or her money if the payment is not accepted. It will simply be returned.

2.2.1 Time Sensitivity

The time sensitive nature of the transaction prevents the sender from having to wait forever to find out whether their payment went through. If the time runs out, the funds will just be sent back to the sender, so they don't have to worry and can wait for the process to unfold.

2.2.2 Trustless system

As is the case with all smart contracts, trust is not needed as the rules are already coded into the contract itself. Hash Time Lock Contracts take this one step further by implementing a time limit for recipients to acknowledge the payment.

2.2.3 Validation of the Blockchain

Transactions are validated because of the cryptographic proof of payment required by the receiver.

2.2.4 Private Information's

There are no complicated account setups or KYC/AML restrictions. Trade directly from your wallet with a counterparty of your choice. Only the parties involved know the details of the trade.

2.2.5 Trading across multiple Cryptocurrencies

HTLC makes Cross-chain transactions easier and more secure than ever. Cross chain transactions are the next step in the evolution of Cryptocurrency adoption. The easier it becomes to unite the hundreds of blockchain's that currently exist in silos, the faster the technology as a whole can begin to scale and achieve mass adoption.

CHAPTER

THREE

INSTALLING SWAP

The easiest way to install Swap is via pip:

```
$ pip install swap
```

If you want to run the latest version of the code, you can install from git:

```
$ pip install git+git://github.com/meherett/swap.git
```

For the versions available, see the tags on this repository.

3.1 Development

We welcome pull requests. To get started, just fork this github repository, clone it locally, and run:

```
$ pip install -e . -r requirements.txt
```

Once you have installed, type `swap` to verify that it worked:

```
$ swap
Usage: swap [OPTIONS] COMMAND [ARGS]...

Options:
  -v, --version  Show Swap version and exit.
  -h, --help     Show this message and exit.

Commands:
  bitcoin  Select Bitcoin provider.
  bytom    Select Bytom provider.
```

3.2 Dependencies

Swap has the following dependencies:

- bytom-wallet-desktop - version 1.1.0 or greater.
- pip - To install packages from the Python Package Index and other indexes
- python3 version 3.6 or greater, python3-dev

CHAPTER

FOUR

COMMAND LINE INTERFACE (CLI)

After you have installed, type `swap` to verify that it worked:

```
$ swap
Usage: swap [OPTIONS] COMMAND [ARGS]...

Options:
  -v, --version  Show Shuttle version and exit.
  -h, --help     Show this message and exit.

Commands:
  bitcoin  Select Bitcoin provider.
  bytom    Select Bytom provider.
```

4.1 swap

```
swap [OPTIONS] COMMAND [ARGS]...
```

Options

-v, --version
Show Swap version and exit.

4.1.1 bitcoin

Select Bitcoin provider.

```
swap bitcoin [OPTIONS] COMMAND [ARGS]...
```

claim

Select Bitcoin Claim transaction builder.

```
swap bitcoin claim [OPTIONS]
```

Options

-a, --address <address>
Required Set Bitcoin recipient address.

-ti, --transaction-id <transaction_id>
Required Set Bitcoin funded transaction id/hash.

-am, --amount <amount>
Required Set Bitcoin amount (SATOSHI).

-n, --network <network>
Set Bitcoin network.

Default mainnet

-v, --version <version>
Set Bitcoin transaction version.

Default 2

decode

Select Bitcoin transaction raw decoder.

```
swap bitcoin decode [OPTIONS]
```

Options

-tr, --transaction-raw <transaction_raw>
Required Set Bitcoin transaction raw.

-i, --indent <indent>
Set json indent.

Default 4

-o, --offline <offline>
Set Offline decode transaction raw.

Default True

fund

Select Bitcoin Fund transaction builder.

```
swap bitcoin fund [OPTIONS]
```

Options

-a, --address <address>
Required Set Bitcoin sender address.

-ha, --htlc-address <htlc_address>
Required Set Bitcoin Hash Time Lock Contract (HTLC) address.

-am, --amount <amount>
Required Set Bitcoin amount (SATOSHI).

-n, --network <network>
Set Bitcoin network.

Default mainnet

-v, --version <version>
Set Bitcoin transaction version.

Default 2

htlc

Select Bitcoin Hash Time Lock Contract (HTLC) builder.

```
swap bitcoin htlc [OPTIONS]
```

Options

-sh, --secret-hash <secret_hash>
Required Set secret 256 hash.

-ra, --recipient-address <recipient_address>
Required Set Bitcoin recipient address.

-sa, --sender-address <sender_address>
Required Set Bitcoin sender address.

-s, --sequence <sequence>
Set Bitcoin sequence/expiration block.

Default 1000

-n, --network <network>
Set Bitcoin network.

Default mainnet

refund

Select Bitcoin Refund transaction builder.

```
swap bitcoin refund [OPTIONS]
```

Options

-a, --address <address>
Required Set Bitcoin sender address.

-ti, --transaction-id <transaction_id>
Required Set Bitcoin funded transaction id/hash.

-am, --amount <amount>
Required Set Bitcoin amount (SATOSHI).

-n, --network <network>
Set Bitcoin network.

Default mainnet

-v, --version <version>
Set Bitcoin transaction version.

Default 2

sign

Select Bitcoin transaction raw signer.

```
swap bitcoin sign [OPTIONS]
```

Options

-rxk, --root-xprivate-key <root_xprivate_key>
Required Set Bitcoin root xprivate key.

-tr, --transaction-raw <transaction_raw>
Required Set Bitcoin unsigned transaction raw.

-b, --bytecode <bytecode>
Set Bitcoin witness HTLC bytecode. [default: None]

-sk, --secret-key <secret_key>
Set secret key. [default: None]

-s, --sequence <sequence>
Set Bitcoin sequence/expiration block.

Default 1000

-ac, --account <account>
Set Bitcoin derivation from account.

Default 1

-ch, --change <change>
Set Bitcoin derivation from change.

Default False

-ad, --address <address>
Set Bitcoin derivation from address.

Default 1

-p, --path <path>
Set Bitcoin derivation from path. [default: None]

-v, --version <version>
Set Bitcoin transaction version.

Default 2

submit

Select Bitcoin transaction raw submitter.

```
swap bitcoin submit [OPTIONS]
```

Options

-tr, --transaction-raw <transaction_raw>
Required Set signed Bitcoin transaction raw.

4.1.2 bytom

Select Bytom provider.

```
swap bytom [OPTIONS] COMMAND [ARGS]...
```

claim

Select Bytom Claim transaction builder.

```
swap bytom claim [OPTIONS]
```

Options

-a, --address <address>
Required Set Bytom recipient address.

-ti, --transaction-id <transaction_id>
Required Set Bytom funded transaction id/hash.

-am, --amount <amount>
Required Set Bytom amount (NEU).

-as, --asset <asset>
Set Bytom asset id.

Default ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

-n, --network <network>
Set Bitcoin network.

Default mainnet

decode

Select Bytom transaction raw decoder.

```
swap bytom decode [OPTIONS]
```

Options

-tr, --transaction-raw <transaction_raw>
Required Set Bytom transaction raw.

-i, --indent <indent>
Set json indent.

Default 4

fund

Select Bytom Fund transaction builder.

```
swap bytom fund [OPTIONS]
```

Options

-a, --address <address>
Required Set Bytom sender address.

-ha, --htlc-address <htlc_address>
Required Set Bytom Hash Time Lock Contract (HTLC) address.

-am, --amount <amount>
Required Set Bytom amount (NEU).

-as, --asset <asset>
Set Bytom asset id.

Default ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

-n, --network <network>
Set Bitcoin network.

Default mainnet

htlc

Select Bytom Hash Time Lock Contract (HTLC) builder.

```
swap bytom htlc [OPTIONS]
```

Options

-sh, --secret-hash <secret_hash>
Required Set secret 256 hash.

-rpk, --recipient-public-key <recipient_public_key>
Required Set Bytom recipient public key.

-spk, --sender-public-key <sender_public_key>
Required Set Bytom sender public key.

-s, --sequence <sequence>
Set Bytom sequence/expiration block.

Default 1000

-n, --network <network>
Set Bytom network.

Default mainnet

refund

Select Bytom Refund transaction builder.

```
swap bytom refund [OPTIONS]
```

Options

-a, --address <address>
Required Set Bytom sender address.

-ti, --transaction-id <transaction_id>
Required Set Bytom funded transaction id/hash.

-am, --amount <amount>
Required Set Bytom amount (NEU).

-as, --asset <asset>
Set Bytom asset id.

Default ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff

-n, --network <network>
Set Bitcoin network.

Default mainnet

sign

Select Bytom transaction raw signer.

```
swap bytom sign [OPTIONS]
```

Options

-xk, --xprivate-key <xprivate_key>
Required Set Bytom xprivate key.

-tr, --transaction-raw <transaction_raw>
Required Set Bytom unsigned transaction raw.

-b, --bytecode <bytecode>
Set Bytom witness HTLC bytecode. [default: None]

-sk, --secret-key <secret_key>
Set secret key. [default: None]

-ac, --account <account>
Set Bytom derivation from account.

Default 1

-ch, --change <change>
Set Bytom derivation from change.

Default False

-ad, --address <address>
Set Bytom derivation from address.

Default 1

-p, --path <path>
Set Bytom derivation from path. [default: None]

-i, --indexes <indexes>
Set Bytom derivation from indexes. [default: None]

submit

Select Bytom transaction raw submitter.

```
swap bytom submit [OPTIONS]
```

Options

-tr, --transaction-raw <transaction_raw>
Required Set signed Bytom transaction raw.

CHAPTER

FIVE

UTILS

swap.utils.**generate_passphrase**(*length: int = 32*) → str
Generate entropy hex string.

Parameters **length** (*int*) – Passphrase length, default to 32.

Returns str – Passphrase hex string.

```
>>> from swap.utils import generate_passphrase
>>> generate_passphrase(length=32)
"N39rPfa3QvF2Tm2nPyoBpXNiBFXJywTz"
```

swap.utils.**generate_entropy**(*strength: int = 128*) → str
Generate entropy hex string.

Parameters **strength** (*int*) – Entropy strength, default to 128.

Returns str – Entropy hex string.

```
>>> from swap.utils import generate_entropy
>>> generate_entropy(strength=128)
"ee535b143b0d9d1f87546f9df0d06b1a"
```

swap.utils.**generate_mnemonic**(*language: str = 'english'*, *strength: int = 128*) → str
Generate 12 word mnemonic.

Parameters

- **language** (*str*) – Mnemonic language, default to english.
- **strength** (*int*) – Entropy strength, default to 128.

Returns str – Mnemonic words.

```
>>> from swap.utils import generate_mnemonic
>>> generate_mnemonic(language="french")
"sceptre capter sequence girafe absolu relatif fleur zoologie muscle sirop␣
↪saboter parure"
```

swap.utils.**is_mnemonic**(*mnemonic: str*, *language: Optional[str] = None*) → bool
Check mnemonic.

Parameters

- **mnemonic** (*str*) – Mnemonic words.
- **language** (*str*) – Mnemonic language, default to None.

Returns bool – Mnemonic valid/invalid.

```
>>> from swap.utils import is_mnemonic
>>> is_mnemonic(mnemonic="sceptre capter sequence girafe absolu relatif fleur␣
↪zoologie muscle sirop saboter parure")
True
```

swap.utils.**get_mnemonic_language**(*mnemonic: str*) → str
Get mnemonic language.

Parameters **mnemonic** (*str*) – Mnemonic words.

Returns str – Mnemonic language.

```
>>> from swap.utils import get_mnemonic_language
>>> get_mnemonic_language(mnemonic="sceptre capter sequence girafe absolu relatif␣
↪fleur zoologie muscle sirop saboter parure")
"french"
```

swap.utils.**sha256**(*data: Union[str, bytes]*) → str
SHA256 hash.

Parameters **data** (*str, bytes*) – Any string/bytes data.

Returns str – SHA256 hash.

```
>>> from swap.utils import sha256
>>> sha256(data="Hello Meheret!")
"3a26da82ead15a80533a02696656b14b5dbfd84eb14790f2e1be5e9e45820eeb"
```

swap.utils.**double_sha256**(*data: Union[str, bytes]*) → str
Double SHA256 hash.

Parameters **data** (*str, bytes*) – Any string/bytes data.

Returns str – Double SHA256 hash.

```
>>> from swap.utils import double_sha256
>>> double_sha256(data="Hello Meheret!")
"821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e0158"
```

swap.utils.**clean_transaction_raw**(*transaction_raw: str*) → str
Clean transaction raw.

Parameters **transaction_raw** (*str*) – Any transaction raw.

Returns str – Cleaned transaction raw.

```
>>> from swap.utils import clean_transaction_raw
>>> clean_transaction_raw(transaction_raw=
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZnU2d
↪")

↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZnU2d
↪"
```

CHAPTER

SIX

BITCOIN

Bitcoin is a Cryptocurrency. It is a decentralized digital currency without a central bank or single administrator that can be sent from user to user on the peer-to-peer bitcoin network without the need for intermediaries.

6.1 Wallet

The implementation of Hierarchical Deterministic (HD) wallets generator for Bitcoin blockchain.

class swap.providers.bitcoin.wallet.**Wallet**(*network: str = 'mainnet'*)

Bitcoin Wallet class.

Parameters **network** (*str*) – Bitcoin network, defaults to mainnet.

Returns Wallet – Bitcoin wallet instance.

Note: Bitcoin has only two networks, `mainnet` and `mainnet`.

from_entropy(*entropy: str*, *passphrase: str = None*, *language: str = 'english'*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from entropy.

Parameters

- **entropy** (*str*) – Bitcoin wallet entropy.
- **passphrase** (*str*) – Bitcoin wallet passphrase, default to None.
- **language** (*str*) – Bitcoin wallet language, default to english.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_mnemonic(*mnemonic: str*, *passphrase: str = None*, *language: str = None*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from mnemonic.

Parameters

- **mnemonic** (*str*) – Bitcoin wallet mnemonic.
- **passphrase** (*str*) – Bitcoin wallet passphrase, default to None.

- **language** (*str*) – Bitcoin wallet language, default to english.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_mnemonic("extend length miss suit broken rescue around harbor␣
↪vehicle vicious jelly quality")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_seed(*seed: str*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from seed.

Parameters **seed** (*str*) – Bitcoin wallet seed.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_seed(
↪"51a0f6fb9abd5e5aa27f42dd375d8e4fc6944c704c859454e557fc419d3979e5a50273743c93e5035244adb09
↪")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_root_xprivate_key(*root_xprivate_key: str*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from root xprivate key.

Parameters **root_xprivate_key** (*str*) – Bitcoin wallet root xprivate key.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_root_xprivate_key(
↪"xprv9s21ZrQH143K4QXLfi9Ht3fz7CciYxE2MuTdNxvDs8kRQRyPByvJLRSvfNBa3kh6twMksiJtZuyT2Cor7aLAA
↪")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_xprivate_key(*xprivate_key: str*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from xprivate key.

Parameters **xprivate_key** (*str*) – Bitcoin wallet xprivate key.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_xprivate_key(
↪"xprvA2WJkML27XMaNmtrsuVcvCCXTR5LiBKAwkV6LapBFmw7eGFkHHzVa57gPRjBToTnvr2PpkN4s1reiDW6Ay9yX
↪")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_wif(*wif: str*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from wallet important format (WIF).

Parameters **wif** (*str*) – Bitcoin wallet important format.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_wif("L4p5duRK9PZVP22rPLTZ8Zar77JQ1Pc6dz3Js5drL89wPRH1kz6R")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_private_key(*private_key*) → *swap.providers.bitcoin.wallet.Wallet*

Initialize wallet from private key.

Parameters **private_key** (*str*) – Bitcoin wallet private key.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_private_key(
↪"e28afe15f98501502fac7da75939d41a0c8d074aeb76d0131f5a5c5ce3132a79")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_path(*path: str*) → *swap.providers.bitcoin.wallet.Wallet*

Drive Bitcoin wallet from path.

Parameters **path** (*str*) – Bitcoin wallet path.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

from_index(*index: int*, *harden: bool = False*) → *swap.providers.bitcoin.wallet.Wallet*

Drive Bitcoin wallet from index.

Parameters

- **index** (*int*) – Bitcoin wallet index.
- **harden** (*bool*) – Use harden, default to False.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_index(44, harden=True)
>>> wallet.from_index(0, harden=True)
>>> wallet.from_index(0, harden=True)
>>> wallet.from_index(0)
>>> wallet.from_index(0)
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
```

clean_derivation() → *swap.providers.bitcoin.wallet.Wallet*

Clean derivation Bitcoin wallet.

Returns Wallet – Bitcoin wallet instance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
```

(continues on next page)

(continued from previous page)

```
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.path()
"m/44'/0'/0'/0/0"
>>> wallet.clean_derivation()
<swap.providers.bitcoin.wallet.Wallet object at 0x040DA268>
>>> wallet.path()
None
```

entropy() → Optional[str]
Get Bitcoin wallet entropy.

Returns str – Bitcoin wallet entropy.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.entropy()
"50f002376c81c96e430b48f1fe71df57"
```

mnemonic() → Optional[str]
Get Bitcoin wallet mnemonic.

Returns str – Bitcoin wallet mnemonic.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.mnemonic()
"extend length miss suit broken rescue around harbor vehicle vicious jelly␣
↪quality"
```

passphrase() → Optional[str]
Get Bitcoin wallet passphrase.

Returns str – Bitcoin wallet passphrase.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57", passphrase=
↪"meherett")
>>> wallet.passphrase()
"meherett"
```

language() → Optional[str]
Get Bitcoin wallet language.

Returns str – Bitcoin wallet language.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.language()
"english"
```

seed() → Optional[str]
Get Bitcoin wallet seed.

Returns str – Bitcoin wallet seed.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.seed()

↪"51a0f6fb9abd5e5aa27f42dd375d8e4fc6944c704c859454e557fc419d3979e5a50273743c93e5035244adb09
↪"
```

root_xprivate_key (*encoded: bool = True*) → Optional[str]
Get Bitcoin wallet root xprivate key.

Parameters **encoded** (*bool*) – Encoded root xprivate key, default to True.

Returns str – Bitcoin wallet root xprivate key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.root_xprivate_key()

↪"xprv9s21ZrQH143K4QXLfi9Ht3fz7CciYxE2MuTdNxvDs8kRQRyPByvJLRSvfNBa3kh6twMksiJtZuyT2Cor7aLAA
↪"
```

root_xpublic_key (*encoded: bool = True*) → Optional[str]
Get Bitcoin wallet root xpublic key.

Parameters **encoded** (*bool*) – Encoded root xprivate key, default to True.

Returns str – Bitcoin wallet root xpublic key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.root_xpublic_key()

↪"xpub661MyMwAqRbcGtbomjgJFBcifETCxQwsj8PEBMKqRUHQHEJXjXEYtDmQWdoFYwDCpQWXhUt7Ce6D34r9gq7os
↪"
```

xprivate_key (*encoded=True*) → Optional[str]
Get Bitcoin wallet xprivate key.

Parameters **encoded** (*bool*) – Encoded xprivate key, default to True.

Returns str – Bitcoin wallet xprivate key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.xprivate_key()

↪"xprvA2WJkML27XMaNmtrsuVcvCCXTR5LiBKAwkV6LapBFmw7eGFkHHzVa57gPRjBToTnvr2PpkN4s1reiDW6Ay9yX
↪"
```

xpublic_key (*encoded: bool = True*) → Optional[str]
Get Bitcoin wallet xpublic key.

Parameters **encoded** (*bool*) – Encoded xprivate key, default to True.

Returns str – Bitcoin wallet xpublic key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.xpublic_key()

↪"xpub6FVf9rruwtusbFyKyw2dHL9G1Suq7e32JyQh8yDnp7U6X4atpqJk7sSAEgfR45VFNs64tsRF67XnGFjuHER3S
↪"
```

uncompressed() → str
Get Bitcoin wallet uncompressed public key.

Returns str – Bitcoin wallet uncompressed public key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.uncompressed()

↪"d5fb6799738d146d7558ac0b14c74cc66f879bd846231b64296fb7cc7c9d974fc6e73408a18811a99906fd10f
↪"
```

compressed() → str
Get Bitcoin wallet compressed public key.

Returns str – Bitcoin wallet compressed public key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.compressed()
"02d5fb6799738d146d7558ac0b14c74cc66f879bd846231b64296fb7cc7c9d974f"
```

chain_code() → str
Get Bitcoin wallet chain code.

Returns str – Bitcoin wallet chain code.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.chain_code()
"baa85729cc8400fd0321ec6df70e7f976a601b133c1aae91a5ec2638fa748017"
```

private_key() → str
Get Bitcoin wallet private key.

Returns str – Bitcoin wallet private key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.private_key()
"e28afe15f98501502fac7da75939d41a0c8d074aeb76d0131f5a5c5ce3132a79"
```

public_key (*private_key: str = None*) → str
Get Bitcoin wallet public key.

Returns str – Bitcoin wallet public key.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.public_key()
"02d5fb6799738d146d7558ac0b14c74cc66f879bd846231b64296fb7cc7c9d974f"
```

path() → Optional[str]
Get Bitcoin wallet path.

Returns str – Bitcoin wallet path.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.path()
"m/44'/0'/0'/0/0"
```

address() → str
Get Bitcoin wallet address.

Returns str – Bitcoin wallet address.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.address()
"18Ac1AiZuNU7ywC1qP6Ref3hGbdRM74Rxv"
```

wif() → str
Get Bitcoin wallet important format (WIF).

Returns str – Bitcoin wallet important format.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.wif()
"L4p5duRK9PZVP22rPLTZ8Zar77JQ1Pc6dz3Js5drL89wPRH1kz6R"
```

hash() → str
Get Bitcoin wallet public key/address hash.

Returns str – Bitcoin wallet public key/address hash.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.hash()
"4e99d7d43ebb41a620426aa836dbd4c4fa85667e"
```

p2pkh() → str
Get Bitcoin wallet public key/address p2pkh.

Returns str – Bitcoin wallet public key/address p2pkh.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.p2pkh()
"76a9144e99d7d43ebb41a620426aa836dbd4c4fa85667e88ac"
```

balance() → int
Get Bitcoin wallet balance.

Returns int – Bitcoin wallet balance.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.balance()
1000000
```

utxos(*limit: int = 15*) → list
Get Bitcoin wallet unspent transaction output (UTXO's).

Parameters **limit** (*int*) – Bitcoin balance, default is 15.

Returns list – Bitcoin unspent transaction outputs.

```
>>> from swap.providers.bitcoin.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44'/0'/0'/0/0")
>>> wallet.utxos()
[{'index': 0, 'hash':
↪'be3466266281996089267 92d775381e54d8632c14b3ce702f90639481722392c', 'output_
↪index': 1, 'amount': 12340, 'script':
↪'76a9146bce65e58a50b97989930e9a4ff1ac1a77515ef188ac'}]
```

6.2 Hash Time Lock Contract (HTLC)

Bitcoin Hash Time Lock Contract (HTLC).

class swap.providers.bitcoin.htlc.**HTLC**(*network: str = 'mainnet'*)
Bitcoin Hash Time Lock Contract (HTLC).

Parameters **network** (*str*) – Bitcoin network, defaults to testnet.

Returns HTLC – Bitcoin HTLC instance.

Note: Bitcoin has only two networks, `mainnet` and `testnet`.

---

build_htlc(*secret_hash: str*, *recipient_address: str*, *sender_address: str*, *sequence: int = 1000*) → *swap.providers.bitcoin.htlc.HTLC*
Build Bitcoin Hash Time Lock Contract (HTLC).

Parameters

- **secret_hash** (*str*) – secret sha-256 hash.
- **recipient_address** (*str*) – Bitcoin recipient address.
- **sender_address** (*str*) – Bitcoin sender address.
- **sequence** (*int*) – Bitcoin sequence number of expiration block, defaults to 1000.

Returns HTLC – Bitcoin Hash Time Lock Contract (HTLC) instance.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="testnet")
>>> htlc.build_htlc(secret_hash=sha256("Hello Meheret!"), recipient_address=
↪"mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF", sender_address=
↪"mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC", sequence=1000)
<swap.providers.bitcoin.htlc.HTLC object at 0x0409DAF0>
```

from_opcode(*opcode: str*) → *swap.providers.bitcoin.htlc.HTLC*

Initiate Bitcoin Hash Time Lock Contract (HTLC) from opcode script.

Parameters **opcode** (*str*) – Bitcoin opcode script.

Returns HTLC – Bitcoin Hash Time Lock Contract (HTLC) instance.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> htlc = HTLC(network="testnet")
>>> opcode = "OP_IF OP_HASH256␣
↪821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e0158 OP_
↪EQUALVERIFY OP_DUP OP_HASH160 0e259e08f2ec9fc99a92b6f66fdfcb3c7914fd68 OP_
↪EQUALVERIFY OP_CHECKSIG OP_ELSE e803 OP_CHECKSEQUENCEVERIFY OP_DROP OP_DUP␣
↪OP_HASH160 33ecab3d67f0e2bde43e52f41ec1ecbdc73f11f8 OP_EQUALVERIFY OP_
↪CHECKSIG OP_ENDIF"
>>> htlc.from_opcode(opcode=opcode)
<swap.providers.bitcoin.htlc.HTLC object at 0x0409DAF0>
```

from_bytecode(*bytecode: str*) → *swap.providers.bitcoin.htlc.HTLC*

Initialize Bitcoin Hash Time Lock Contract (HTLC) from bytecode.

Parameters **bytecode** (*str*) – Bitcoin bytecode.

Returns HTLC – Bitcoin Hash Time Lock Contract (HTLC) instance.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> htlc = HTLC(network="testnet")
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> htlc.from_bytecode(bytecode=bytecode)
<swap.providers.bitcoin.htlc.HTLC object at 0x0409DAF0>
```

bytecode() → str

Get Bitcoin Hash Time Lock Contract (HTLC) bytecode.

Returns str – Bitcoin HTLC bytecode.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="testnet")
```

(continues on next page)

(continued from previous page)

```
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF", "mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",␣
↪1000)
>>> htlc.bytecode()

↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
```

opcode() → str
Get Bitcoin Hash Time Lock Contract (HTLC) OP_Code.

Returns str – Bitcoin HTLC opcode.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="testnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF", "mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",␣
↪1000)
>>> htlc.opcode()
"OP_IF OP_HASH256␣
↪821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e0158 OP_
↪EQUALVERIFY OP_DUP OP_HASH160 0e259e08f2ec9fc99a92b6f66fdfcb3c7914fd68 OP_
↪EQUALVERIFY OP_CHECKSIG OP_ELSE e803 OP_CHECKSEQUENCEVERIFY OP_DROP OP_DUP␣
↪OP_HASH160 33ecab3d67f0e2bde43e52f41ec1ecbdc73f11f8 OP_EQUALVERIFY OP_
↪CHECKSIG OP_ENDIF"
```

hash() → str
Get Bitcoin HTLC hash.

Returns str – Bitcoin Hash Time Lock Contract (HTLC) hash.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="testnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF", "mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",␣
↪1000)
>>> htlc.hash()
"a9149418feed4647e156d6663db3e0cef7c050d0386787"
```

address() → str
Get Bitcoin Hash Time Lock Contract (HTLC) address.

Returns str – Bitcoin HTLC address.

```
>>> from swap.providers.bitcoin.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="testnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF", "mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",␣
↪1000)
>>> htlc.address()
"2N6kHwQy6Ph5EdKNgzGrcW2WhGHKGfmP5ae"
```

6.3 Transaction

Bitcoin transaction in blockchain network.

class swap.providers.bitcoin.transaction.**Transaction**(*network: str = 'mainnet'*, *version: int = 2*)

Bitcoin Transaction.

Parameters

- **network** (*str*) – Bitcoin network, defaults to testnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns Transaction – Bitcoin transaction instance.

Note: Bitcoin has only two networks, `mainnet` and `testnet`.

---

fee() → int

Get Bitcoin transaction fee.

Returns int – Bitcoin transaction fee.

```
>>> from swap.providers.bitcoin.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("testnet")
>>> claim_transaction.build_transaction("mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> claim_transaction.fee()
576
```

hash() → str

Get Bitcoin transaction hash.

Returns str – Bitcoin transaction id/hash.

```
>>> from swap.providers.bitcoin.transaction import FundTransaction
>>> fund_transaction = FundTransaction("testnet")
>>> fund_transaction.build_transaction("mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",
↪"2N6kHwQy6Ph5EdKNgzGrcW2WhGHKGfmP5ae", 10000)
>>> fund_transaction.hash()
"9cc0524fb8e7b2c5fecaee4eb91d43a3dc5cc18e9906abcb35a5732ff52efcc7"
```

json() → dict

Get Bitcoin transaction json format.

Returns dict – Bitcoin transaction json format.

```
>>> from swap.providers.bitcoin.transaction import RefundTransaction
>>> refund_transaction = RefundTransaction("testnet")
>>> refund_transaction.build_transaction("mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",
↪ "1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> refund_transaction.json()
{"hex":
↪"02000000012c392217483906f902e73c4bc132864de58153772d7926896099816226663 4be0100000000ffff
↪", "txid": "9cc0524fb8e7b2c5fecaee4eb91d43a3dc5cc18e9906abcb35a5732ff52efcc7
↪", "hash": "9cc0524fb8e7b2c5fecaee4eb91d43a3dc5cc18e9906abcb35a5732ff52efcc7
↪", "size": 117, "vsize": 117, "version": 2, "locktime": 0, "vin": [{"txid":
↪"be34662662819960892 6792d775381e54d8632c14b3ce702f90639481722392c", "vout":
↪1, "scriptSig": {"asm": "", "hex": ""}, "sequence": "4294967295"}], "vout":
↪[{"value": "0.00001000", "n": 0, "scriptPubKey": {"asm": "OP_HASH160
↪971894c58d85981c16c2059d422bcde0b156d044 OP_EQUAL", "hex":
↪"a914971894c58d85981c16c2059d422bcde0b156d04487", "type": "p2sh", "address
↪": "2N729UBGZB3xjsGFRgKivy4bSjkaJGMVSpB"}}, {"value": "0.00010662", "n": 1,
↪"scriptPubKey": {"asm": "OP_DUP OP_HASH160
↪6bce65e58a50b97989930e9a4ff1ac1a77515ef1 OP_EQUALVERIFY OP_CHECKSIG", "hex
↪": "76a9146bce65e58a50b97989930e9a4ff1ac1a77515ef188ac", "type": "p2pkh",
```

(continues on next page)

(continued from previous page)

raw() → str
Get Bitcoin main transaction raw.

Returns str – Bitcoin transaction raw.

```
>>> from swap.providers.bitcoin.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("testnet")
>>> claim_transaction.build_transaction("mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> claim_transaction.raw()

↪"02000000012c392217483906f902e73c4bc132864de58153772d792689609981622666634be0100000000ffff
↪"
```

type() → str
Get Bitcoin signature transaction type.

Returns str – Bitcoin signature transaction type.

```
>>> from swap.providers.bitcoin.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("testnet")
>>> claim_transaction.build_transaction("mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> claim_transaction.type()
"bitcoin_claim_unsigned"
```

6.3.1 FundTransaction

class swap.providers.bitcoin.transaction.**FundTransaction**(*network: str = 'mainnet'*, *version: int = 2*)
Bitcoin Fund transaction.

Parameters

- **network** (*str*) – Bitcoin network, defaults to testnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns FundTransaction – Bitcoin fund transaction instance.

> **Warning:** Do not forget to build transaction after initialize fund transaction.

build_transaction(*address: str*, *htlc_address: str*, *amount: int*, *locktime: int = 0*) → *swap.providers.bitcoin.transaction.FundTransaction*
Build Bitcoin fund transaction.

Parameters

- **address** (*str*) – Bitcoin sender address.
- **htlc_address** (*str*) – Bitcoin Hash Time Lock Contract (HTLC) address.
- **amount** (*int*) – Bitcoin amount to fund.
- **locktime** (*int*) – Bitcoin transaction lock time, defaults to 0.

Returns FundTransaction – Bitcoin fund transaction instance.

```
>>> from swap.providers.bitcoin.transaction import FundTransaction
>>> fund_transaction = FundTransaction("testnet")
>>> fund_transaction.build_transaction(address=
↪"mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC", htlc_address=
↪"2N6kHwQy6Ph5EdKNgzGrcW2WhGHKGfmP5ae", amount=10000)
<swap.providers.bitcoin.transaction.FundTransaction object at 0x0409DAF0>
```

sign(*solver:* swap.providers.bitcoin.solver.FundSolver) → *swap.providers.bitcoin.transaction.FundTransaction*

Sign Bitcoin fund transaction.

Parameters **solver** (bitcoin.solver.FundSolver) – Bitcoin fund solver.

Returns FundTransaction – Bitcoin fund transaction instance.

```
>>> from swap.providers.bitcoin.transaction import FundTransaction
>>> from swap.providers.bitcoin.solver import FundSolver
>>> from swap.providers.bitcoin.wallet import Wallet
>>> sender_wallet = Wallet("testnet").from_mnemonic("indicate warm sock␣
↪mistake code spot acid ribbon sing over taxi toast").from_path("m/44'/0'/0'/
↪0/0")
>>> fund_solver = FundSolver(sender_wallet.root_xprivate_key())
>>> fund_transaction = FundTransaction("testnet").build_transaction(sender_
↪wallet.address(), "2N6kHwQy6Ph5EdKNgzGrcW2WhGHKGfmP5ae", 10000)
>>> fund_transaction.sign(solver=fund_solver)
<swap.providers.bitcoin.transaction.FundTransaction object at 0x0409DAF0>
```

transaction_raw() → str

Get Bitcoin fund transaction raw.

Returns str – Bitcoin fund transaction raw.

```
>>> from swap.providers.bitcoin.transaction import FundTransaction
>>> fund_transaction = FundTransaction("testnet")
>>> fund_transaction.build_transaction("mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",
↪"2N6kHwQy6Ph5EdKNgzGrcW2WhGHKGfmP5ae", 10000)
>>> fund_transaction.transaction_raw()

↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTJjMzkyMjE3NDgzOTA2ZjkwMmU3M2M0YmMxMzI4NjRkZTU4M
↪"
```

6.3.2 ClaimTransaction

class swap.providers.bitcoin.transaction.**ClaimTransaction**(*network: str = 'mainnet', version: int = 2*)

Bitcoin Claim transaction.

Parameters

- **network** (*str*) – Bitcoin network, defaults to testnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns Transaction – Bitcoin transaction instance.

Warning: Do not forget to build transaction after initialize claim transaction.

build_transaction(*address: str*, *transaction_id: str*, *amount: int*, *locktime: int = 0*) → *swap.providers.bitcoin.transaction.ClaimTransaction*

Build Bitcoin claim transaction.

Parameters

- **address** (*str*) – Bitcoin recipient address.
- **transaction_id** (*str*) – Bitcoin fund transaction id to redeem.
- **amount** (*int*) – Bitcoin amount to withdraw.
- **locktime** (*int*) – Bitcoin transaction lock time, defaults to 0.

Returns ClaimTransaction – Bitcoin claim transaction instance.

```
>>> from swap.providers.bitcoin.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("testnet")
>>> claim_transaction.build_transaction(address=
↪"mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF", transaction_id=
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd",␣
↪amount=10000)
<swap.providers.bitcoin.transaction.ClaimTransaction object at 0x0409DAF0>
```

sign(*solver:* swap.providers.bitcoin.solver.ClaimSolver) → *swap.providers.bitcoin.transaction.ClaimTransaction*

Sign Bitcoin claim transaction.

Parameters **solver** (bitcoin.solver.ClaimSolver) – Bitcoin claim solver.

Returns ClaimTransaction – Bitcoin claim transaction instance.

```
>>> from swap.providers.bitcoin.transaction import ClaimTransaction
>>> from swap.providers.bitcoin.solver import ClaimSolver
>>> from swap.providers.bitcoin.wallet import Wallet
>>> recipient_wallet = Wallet("testnet").from_mnemonic("hint excuse upgrade␣
↪sleep easily deputy erase cluster section other ugly limit").from_path("m/44
↪'/0'/0'/0/0")
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> claim_solver = ClaimSolver(recipient_wallet.root_xprivate_key(), "Hello␣
↪Meheret!", bytecode)
>>> claim_transaction = ClaimTransaction("testnet")
>>> claim_transaction.build_transaction(recipient_wallet.address(),
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> claim_transaction.sign(solver=claim_solver)
<swap.providers.bitcoin.transaction.ClaimTransaction object at 0x0409DAF0>
```

transaction_raw() → str

Get Bitcoin claim transaction raw.

Returns str – Bitcoin claim transaction raw.

```
>>> from swap.providers.bitcoin.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("testnet")
>>> claim_transaction.build_transaction("mgokpSJoX7npmAK1Zj8ze1926CLxYDt1iF",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> claim_transaction.transaction_raw()

↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTJjMzkyMjE3NDgzOTA2ZjkwMmU3M2M0YmMxMzI4NjRkZTU4M
↪"
```

6.3.3 RefundTransaction

class swap.providers.bitcoin.transaction.**RefundTransaction**(*network: str = 'mainnet', version: int = 2*)

Bitcoin Refund transaction.

Parameters

- **network** (*str*) – Bitcoin network, defaults to testnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns Transaction – Bitcoin transaction instance.

> **Warning:** Do not forget to build transaction after initialize refund transaction.

build_transaction(*address: str, transaction_id: str, amount: int, locktime: int = 0*) → *swap.providers.bitcoin.transaction.RefundTransaction*

Build Bitcoin refund transaction.

Parameters

- **address** (*str*) – Bitcoin sender address.
- **transaction_id** (*str*) – Bitcoin fund transaction id to redeem.
- **amount** (*int*) – Bitcoin amount to withdraw.
- **locktime** (*int*) – Bitcoin transaction lock time, defaults to 0.

Returns RefundTransaction – Bitcoin refund transaction instance.

```
>>> from swap.providers.bitcoin.transaction import RefundTransaction
>>> refund_transaction = RefundTransaction("testnet")
>>> refund_transaction.build_transaction(address=
↪"mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC", transaction_id=
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd",␣
↪amount=10000)
<swap.providers.bitcoin.transaction.RefundTransaction object at 0x0409DAF0>
```

sign(*solver:* swap.providers.bitcoin.solver.RefundSolver) → *swap.providers.bitcoin.transaction.RefundTransaction*

Sign Bitcoin refund transaction.

Parameters **solver** (bitcoin.solver.RefundSolver) – Bitcoin refund solver.

Returns RefundTransaction – Bitcoin refund transaction instance.

```
>>> from swap.providers.bitcoin.transaction import RefundTransaction
>>> from swap.providers.bitcoin.solver import RefundSolver
>>> from swap.providers.bitcoin.wallet import Wallet
>>> sender_wallet = Wallet("testnet").from_mnemonic("indicate warm sock␣
↪mistake code spot acid ribbon sing over taxi toast").from_path("m/44'/0'/0'/
↪0/0")
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> refund_solver = RefundSolver(sender_wallet.root_xprivate_key(), bytecode,␣
↪sequence=1000)
>>> refund_transaction = RefundTransaction("testnet")
>>> refund_transaction.build_transaction(sender_wallet.address(),
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
```

(continues on next page)

(continued from previous page)

```
>>> refund_transaction.sign(solver=refund_solver)
<swap.providers.bitcoin.transaction.RefundTransaction object at 0x0409DAF0>
```

transaction_raw() → str

Get Bitcoin refund transaction raw.

Returns str – Bitcoin refund transaction raw.

```
>>> from swap.providers.bitcoin.transaction import RefundTransaction
>>> refund_transaction = RefundTransaction("testnet")
>>> refund_transaction.build_transaction("mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC",
↪ "1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000)
>>> refund_transaction.transaction_raw()

↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTJjMzkyMjE3NDgzOTA2ZjkwMmU3M2M0YmMxMzI4NjRkZTU4M
↪"
```

6.4 Solver

Bitcoin solver.

6.4.1 FundSolver

class swap.providers.bitcoin.solver.**FundSolver**(*root_xprivate_key: str*, *account: int = 0*, *change: bool = False*, *address: int = 0*, *path: Optional[str] = None*)

Bitcoin Fund solver.

Parameters

- **root_xprivate_key** (*str*) – Bitcoin sender root xprivate key.
- **account** (*int*) – Bitcoin derivation account, defaults to 0.
- **change** (*bool*) – Bitcoin derivation change, defaults to False.
- **address** (*int*) – Bitcoin derivation address, defaults to 0.
- **path** (*str*) – Bitcoin derivation path, defaults to None.

Returns FundSolver – Bitcoin fund solver instance.

```
>>> from swap.providers.bitcoin.solver import FundSolver
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky25Bp
↪"
>>> fund_solver = FundSolver(root_xprivate_key=sender_root_xprivate_key)
<swap.providers.bitcoin.solver.FundSolver object at 0x03FCCA60>
```

6.4.2 ClaimSolver

class swap.providers.bitcoin.solver.**ClaimSolver**(*root_xprivate_key: str*, *secret_key: str*, *bytecode: str*, *account: int = 0*, *change: bool = False*, *address: int = 0*, *path: Optional[str] = None*)

Bitcoin Claim solver.

Parameters

- **root_xprivate_key** (*str*) – Bitcoin sender root xprivate key.
- **secret_key** (*str*) – Secret password/passphrase.
- **bytecode** (*str*) – Bitcoin witness HTLC bytecode..
- **account** (*int*) – Bitcoin derivation account, defaults to 0.
- **change** (*bool*) – Bitcoin derivation change, defaults to False.
- **address** (*int*) – Bitcoin derivation address, defaults to 0.
- **path** (*str*) – Bitcoin derivation path, defaults to None.

Returns ClaimSolver – Bitcoin claim solver instance.

```
>>> from swap.providers.bitcoin.solver import ClaimSolver
>>> recipient_root_xprivate_key =
↪"xprv9s21ZrQH143K4Kpce43z5guPyxLrFoc2i8aQAq835Zzp4Rt7i6nZaMCnVSDyHT6MnmJJGKHMrCUqaYpGojrug1ZN5
↪"
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2ec9fc
↪"
>>> claim_solver = ClaimSolver(wallet=recipient_root_xprivate_key, secret_key=
↪"Hello Meheret!", bytecode=bytecode)
<swap.providers.bitcoin.solver.ClaimSolver object at 0x03FCCA60>
```

6.4.3 RefundSolver

class swap.providers.bitcoin.solver.**RefundSolver**(*root_xprivate_key: str*, *bytecode: str*, *sequence: int = 1000*, *account: int = 0*, *change: bool = False*, *address: int = 0*, *path: Optional[str] = None*)

Bitcoin Refund solver.

Parameters

- **root_xprivate_key** (*str*) – Bitcoin sender root xprivate key.
- **bytecode** (*str*) – Bitcoin witness HTLC bytecode..
- **sequence** (*int*) – Bitcoin witness sequence number(expiration block), defaults to 1000.
- **account** (*int*) – Bitcoin derivation account, defaults to 0.
- **change** (*bool*) – Bitcoin derivation change, defaults to False.
- **address** (*int*) – Bitcoin derivation address, defaults to 0.
- **path** (*str*) – Bitcoin derivation path, defaults to None.

Returns RefundSolver – Bitcoin refund solver instance.

```
>>> from swap.providers.bitcoin.solver import RefundSolver
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky25Bp
↪"
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2ec9fc
↪"
>>> refund_solver = RefundSolver(root_xprivate_key=sender_root_xprivate_key,␣
↪bytecode=bytecode sequence=1000)
<swap.providers.bitcoin.solver.RefundSolver object at 0x03FCCA60>
```

6.5 Signature

Bitcoin signature.

class swap.providers.bitcoin.signature.**Signature**(*network: str = 'mainnet'*, *version: int = 2*)

Bitcoin Signature.

Parameters

- **network** (*str*) – Bitcoin network, defaults to mainnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns Transaction – Bitcoin transaction instance.

Note: Bitcoin has only two networks, `mainnet` and `testnet`.

---

fee() → int

Get Bitcoin transaction fee.

Returns int – Bitcoin transaction fee.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwY
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
>>> fund_solver = FundSolver(sender_root_xprivate_key)
>>> signature = Signature("testnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.fee()
678
```

hash() → str

Get Bitcoin signature transaction hash.

Returns str – Bitcoin signature transaction hash or transaction id.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import ClaimSolver
>>> unsigned_claim_transaction_raw =
↪"eyJmZWUiOiA1NzYsICJyYXciOiAiMDIwMDAwMDAwMTE4MjNmMzlhOGM1ZjZmMjc4NDVkZDExYTY1ZTAzZmUyZWY1M
↪"
```

(continues on next page)

(continued from previous page)

```
>>> recipient_root_xprivate_key =
↪"xprv9s21ZrQH143K4Kpce43z5guPyxLrFoc2i8aQAq835Zzp4Rt7i6nZaMCnVSDyHT6MnmJJGKHMrCUqaYpGojruq
↪"
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> claim_solver = ClaimSolver(recipient_root_xprivate_key, "Hello Meheret!",␣
↪bytecode)
>>> signature = Signature("testnet")
>>> signature.sign(unsigned_claim_transaction_raw, claim_solver)
>>> signature.hash()
"29c7ac0ec049687e1b952cefdaf2f1f52957e6f42f35826af21ec6bd3edf60ce"
```

json() → dict

Get Bitcoin signature transaction json format.

Returns str – Bitcoin signature transaction json format.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwY
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
>>> fund_solver = FundSolver(sender_root_xprivate_key)
>>> signature = Signature("testnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.json()
{"hex":
↪"02000000010825e00ba596ab11126cd89203b882bce60a7db019e51217056c471f510cfd85000000006b48304
↪", "txid": "29c7ac0ec049687e1b952cefdaf2f1f52957e6f42f35826af21ec6bd3edf60ce
↪", "hash": "29c7ac0ec049687e1b952cefdaf2f1f52957e6f42f35826af21ec6bd3edf60ce
↪", "size": 224, "vsize": 224, "version": 2, "locktime": 0, "vin": [{"txid":
↪"85fd0c511f476c051712e519b07d0ae6bc82b80392d86c1211ab96a50be02508", "vout":␣
↪0, "scriptSig": {"asm":
↪"30450221009ac6afb68728eee53050ea7a301b6fb836e13b782cd52c29be2f8b0cc71f4427022069671a0a3df
↪02065e8cb5fa76699079860a450bddd0e37e0ad3dbf2ddfd01d7b600231e6cde8e", "hex":
↪"4830450221009ac6afb68728eee53050ea7a301b6fb836e13b782cd52c29be2f8b0cc71f4427022069671a0a3
↪"}, "sequence": "4294967295"}], "vout": [{"value": "0.00010000", "n": 0,
↪"scriptPubKey": {"asm": "OP_HASH160␣
↪4695127b1d17c454f4bae9c41cb8e3cdb5e89d24 OP_EQUAL", "hex":
↪"a9144695127b1d17c454f4bae9c41cb8e3cdb5e89d2487", "type": "p2sh", "address
↪": "2MygRsRs6En1RCj8a88FfsK1QBeissBTswL"}}, {"value": "0.00089322", "n": 1,
↪"scriptPubKey": {"asm": "OP_DUP OP_HASH160␣
↪33ecab3d67f0e2bde43e52f41ec1ecbdc73f11f8 OP_EQUALVERIFY OP_CHECKSIG", "hex
↪": "76a91433ecab3d67f0e2bde43e52f41ec1ecbdc73f11f888ac", "type": "p2pkh",
↪"address": "mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC"}}]}
```

raw() → str

Get Bitcoin main transaction raw.

Returns str – Bitcoin signature transaction raw.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import RefundSolver
```

(continues on next page)

(continued from previous page)

```
>>> unsigned_refund_transaction_raw =
↪"eyJmZWUiOiA1NzYsICJyYXciOiAiMDIwMDAwMDAwMTE4MjNmMzlhOGM1ZjZmMjc4NDVkZDEzYTY1ZTAzZmUyZWY1M
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> refund_solver = RefundSolver(sender_root_xprivate_key, bytecode, 1000)
>>> signature = Signature("testnet")
>>> signature.sign(unsigned_refund_transaction_raw, refund_solver)
>>> signature.raw()

↪"02000000011823f39a8c5f6f27845dd13a65e03fe2ef5108d235e7a36edb6eb267b0459c5a00000000ca47304
↪"
```

type() → str

Get Bitcoin signature transaction type.

Returns str – Bitcoin signature transaction type.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwY
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
>>> fund_solver = FundSolver(sender_root_xprivate_key)
>>> signature = Signature("testnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.type()
"bitcoin_fund_signed"
```

sign(*transaction_raw: str*, *solver: Union[*swap.providers.bitcoin.solver.FundSolver, swap.providers.bitcoin.solver.ClaimSolver, swap.providers.bitcoin.solver.RefundSolver*]*) → Union[*swap.providers.bitcoin.signature.FundSignature*, *swap.providers.bitcoin.signature.ClaimSignature*, *swap.providers.bitcoin.signature.RefundSignature*]

Sign unsigned transaction raw.

Parameters

- **transaction_raw** (*str*) – Bitcoin unsigned transaction raw.
- **solver** (bitcoin.solver.FundSolver, bitcoin.solver.ClaimSolver, bitcoin.solver.RefundSolver) – Bitcoin solver

Returns FundSignature, ClaimSignature, RefundSignature – Bitcoin signature instance.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwY
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
```

(continues on next page)

(continued from previous page)

```
>>> fund_solver = FundSolver(sender_root_xprivate_key)
>>> signature = Signature("testnet")
>>> signature.sign(transaction_raw=unsigned_fund_transaction_raw, solver=fund_
↪solver)
<swap.providers.bitcoin.signature.FundSignature object at 0x0409DAF0>
```

transaction_raw() → str
Get Bitcoin transaction raw.

Returns str – Bitcoin transaction raw.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwY
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
>>> fund_solver = FundSolver(sender_root_xprivate_key)
>>> signature = Signature("testnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.transaction_raw()

↪"eyJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwYTdkYjAxOWU1MTIxN
↪"
```

6.5.1 FundSignature

class swap.providers.bitcoin.signature.**FundSignature**(*network: str = 'mainnet'*, *version: int = 2*)
Bitcoin Fund signature.

Parameters

- **network** (*str*) – Bitcoin network, defaults to mainnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns FundSignature – Bitcoin fund signature instance.

sign(*transaction_raw: str*, *solver:* swap.providers.bitcoin.solver.FundSolver) → *swap.providers.bitcoin.signature.FundSignature*
Sign unsigned fund transaction raw.

Parameters

- **transaction_raw** (*str*) – Bitcoin unsigned fund transaction raw.
- **solver** (bitcoin.solver.FundSolver) – Bitcoin fund solver.

Returns FundSignature – Bitcoin fund signature instance.

```
>>> from swap.providers.bitcoin.signature import Signature
>>> from swap.providers.bitcoin.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTA4MjVlMDBiYTU5NmFiMTExMjZjZDg5MjAzYjg4MmJjZTYwY
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
```

(continues on next page)

(continued from previous page)

```
>>> fund_solver = FundSolver(sender_root_xprivate_key)
>>> fund_signature = FundSignature("testnet")
>>> fund_signature.sign(unsigned_fund_transaction_raw, fund_solver)
<swap.providers.bitcoin.signature.FundSignature object at 0x0409DAF0>
```

6.5.2 ClaimSignature

class swap.providers.bitcoin.signature.**ClaimSignature**(*network: str = 'mainnet'*, *version: int = 2*)

Bitcoin Claim signature.

Parameters

- **network** (*str*) – Bitcoin network, defaults to mainnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns ClaimSignature – Bitcoin claim signature instance.

sign(*transaction_raw: str*, *solver:* swap.providers.bitcoin.solver.ClaimSolver) → *swap.providers.bitcoin.signature.ClaimSignature*

Sign unsigned claim transaction raw.

Parameters

- **transaction_raw** (*str*) – Bitcoin unsigned claim transaction raw.
- **solver** (bitcoin.solver.ClaimSolver) – Bitcoin claim solver.

Returns ClaimSignature – Bitcoin claim signature instance.

```
>>> from swap.providers.bitcoin.signature import ClaimSignature
>>> from swap.providers.bitcoin.solver import ClaimSolver
>>> unsigned_claim_transaction_raw =
↪"eyJmZWUiOiA1NzYsICJyYXciOiAiMDIwMDAwMDAwMTE4MjNmMzlhOGM1ZjZmMjc4NDVkZDEzYTY1ZTAzZmUyZWY1M
↪"
>>> recipient_root_xprivate_key =
↪"xprv9s21ZrQH143K4Kpce43z5guPyxLrFoc2i8aQAq835Zzp4Rt7i6nZaMCnVSDyHT6MnmJJGKHMrCUqaYpGojruq
↪"
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> claim_solver = ClaimSolver(recipient_root_xprivate_key, "Hello Meheret!",␣
↪bytecode)
>>> claim_signature = ClaimSignature("testnet")
>>> claim_signature.sign(transaction_raw=unsigned_claim_transaction_raw,␣
↪solver=claim_solver)
<swap.providers.bitcoin.signature.ClaimSignature object at 0x0409DAF0>
```

6.5.3 RefundSignature

class swap.providers.bitcoin.signature.**RefundSignature**(*network: str = 'mainnet', version: int = 2*)

Bitcoin Refund signature.

Parameters

- **network** (*str*) – Bitcoin network, defaults to mainnet.
- **version** (*int*) – Bitcoin transaction version, defaults to 2.

Returns RefundSignature – Bitcoin refund signature instance.

sign(*transaction_raw: str, solver:* swap.providers.bitcoin.solver.RefundSolver) → *swap.providers.bitcoin.signature.RefundSignature*

Sign unsigned refund transaction raw.

Parameters

- **transaction_raw** (*str*) – Bitcoin unsigned refund transaction raw.
- **solver** (bitcoin.solver.RefundSolver) – Bitcoin refund solver.

Returns RefundSignature – Bitcoin refund signature instance.

```
>>> from swap.providers.bitcoin.signature import RefundSignature
>>> from swap.providers.bitcoin.solver import RefundSolver
>>> unsigned_refund_transaction_raw =
↪"eyJmZWUiOiA1NzYsICJyYXciOiAiMDIwMDAwMDAwMTE4MjNmMzlhOGM1ZjZmMjc4NDVkZDEzYTY1ZTAzZmUyZWY1M
↪"
>>> sender_root_xprivate_key =
↪"xprv9s21ZrQH143K3XihXQBN8Uar2WBtrjSzK2oRDEGQ25pA2kKAADoQXaiiVXht163ZTrdtTXfM4GqNRE9gWQHky
↪"
>>> bytecode =
↪"63aa20821124b554d13f247b1e5d10b84e44fb1296f18f38bbaa1bea34a12c843e01588876a9140e259e08f2e
↪"
>>> refund_solver = RefundSolver(sender_root_xprivate_key, bytecode, 1000)
>>> refund_signature = RefundSignature("testnet")
>>> refund_signature.sign(transaction_raw=unsigned_refund_transaction_raw,␣
↪solver=refund_solver)
<swap.providers.bitcoin.signature.RefundSignature object at 0x0409DAF0>
```

6.6 Remote Procedure Call (RPC)

Bitcoin remote procedure call.

swap.providers.bitcoin.rpc.**get_balance**(*address: str, network: str = 'mainnet', headers: dict = {}, timeout: int = 60*) → int

Get Bitcoin balance.

Parameters

- **address** (*str*) – Bitcoin address.
- **network** (*str*) – Bitcoin network, defaults to testnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns int – Bitcoin balance (SATOSHI amount).

```
>>> from swap.providers.bitcoin.rpc import get_balance
>>> get_balance(address="mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC", network="testnet")
25800000
```

swap.providers.bitcoin.rpc.**get_utxos**(*address: str*, *network: str = 'mainnet'*, *include_script: bool = True*, *limit: int = 15*, *headers: dict = {}*, *timeout: int = 60*) → list

Get Bitcoin unspent transaction outputs (UTXO's).

Parameters

- **address** (*str*) – Bitcoin address.
- **network** (*str*) – Bitcoin network, defaults to testnet.
- **include_script** (*bool*) – Bitcoin include script, defaults to True.
- **limit** (*int*) – Bitcoin utxo's limit, defaults to 15.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns list – Bitcoin unspent transaction outputs (UTXO's).

```
>>> from swap.providers.bitcoin.rpc import get_utxos
>>> get_utxos(address="mkFWGt4hT11XS8dJKzzRFsTrqjjAwZfQAC", network="testnet")
[...]
```

swap.providers.bitcoin.rpc.**get_transaction**(*transaction_id: str*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → dict

Get Bitcoin transaction detail.

Parameters

- **transaction_id** (*str*) – Bitcoin transaction id/hash.
- **network** (*str*) – Bitcoin network, defaults to testnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bitcoin transaction detail.

```
>>> from swap.providers.bitcoin.rpc import get_transaction
>>> get_transaction(transaction_id=
↪"4e91bca76db112d3a356c17366df93e364a4922993414225f65390220730d0c1", network=
↪"testnet")
{...}
```

swap.providers.bitcoin.rpc.**decode_raw**(*raw: str*, *network: str = 'mainnet'*, *offline: bool = True*, *headers: dict = {}*, *timeout: int = 60*) → dict

Decode original Bitcoin raw.

Parameters

- **raw** (*str*) – Bitcoin transaction raw.
- **network** (*str*) – Bitcoin network, defaults to testnet.
- **offline** (*bool*) – Offline decode, defaults to True.
- **headers** (*dict*) – Request headers, default to common headers.

- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bitcoin decoded transaction raw.

```
>>> from swap.providers.bitcoin.rpc import decode_raw
>>> decode_raw(raw=
↪"02000000011823f39a8c5f6f27845dd13a65e03fe2ef5108d235e7a36edb6eb267b0459c5a010000006a473044022
↪", network="testnet")
{...}
```

swap.providers.bitcoin.rpc.**submit_raw**(*raw: str*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → str

Submit original Bitcoin raw into blockchain.

Parameters

- **raw** (*str*) – Bitcoin transaction raw.
- **network** (*str*) – Bitcoin network, defaults to testnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bitcoin submitted transaction id/hash.

```
>>> from swap.providers.bitcoin.rpc import submit_raw
>>> submit_raw(raw=
↪"02000000011823f39a8c5f6f27845dd13a65e03fe2ef5108d235e7a36edb6eb267b0459c5a010000006a473044022
↪", network="testnet")
"167faa4043ff622e7860ee5228d1ad6d763c5a6cfce79dbc3b9b5fc7bded6394"
```

6.7 Utils

Bitcoin Utils.

swap.providers.bitcoin.utils.**amount_converter**(*amount: Union[int, float]*, *symbol: str = 'SATOSHI2BTC'*) → Union[int, float]

Amount converter

Parameters

- **amount** (*Union[int, float]*) – Bitcoin amount.
- **symbol** (*str*) – Bitcoin symbol, default to SATOSHI2BTC.

Returns float – BTC asset amount.

```
>>> from swap.providers.bitcoin.utils import amount_converter
>>> amount_converter(amount=10_000_000, symbol="SATOSHI2BTC")
0.1
```

swap.providers.bitcoin.utils.**fee_calculator**(*transaction_input: int = 1*, *transaction_output: int = 1*) → int

Bitcoin fee calculator.

Parameters

- **transaction_input** (*int*) – transaction input numbers, defaults to 1.
- **transaction_output** (*int*) – transaction output numbers, defaults to 1.

Returns int – Bitcoin fee (SATOSHI amount).

```
>>> from swap.providers.bitcoin.utils import fee_calculator
>>> fee_calculator(2, 9)
1836
```

swap.providers.bitcoin.utils.**is_network**(*network: str*) → bool

Check Bitcoin network.

Parameters **network** (*str*) – Bitcoin network.

Returns bool – Bitcoin valid/invalid network.

```
>>> from swap.providers.bitcoin.utils import is_network
>>> is_network("testnet")
True
```

swap.providers.bitcoin.utils.**is_address**(*address: str*, *network: Optional[str] = None*) → bool

Check Bitcoin address.

Parameters

- **address** (*str*) – Bitcoin address.
- **network** (*str*) – Bitcoin network, defaults to None.

Returns bool – Bitcoin valid/invalid address.

```
>>> from swap.providers.bitcoin.utils import is_address
>>> is_address("mrmtGq2HMmqAogSsGDjCtXUpxrb7rHThFH", "testnet")
True
```

swap.providers.bitcoin.utils.**is_transaction_raw**(*transaction_raw: str*) → bool

Check Bitcoin transaction raw.

Parameters **transaction_raw** (*str*) – Bitcoin transaction raw.

Returns bool – Bitcoin valid/invalid transaction raw.

```
>>> from swap.providers.bitcoin.utils import is_transaction_raw
>>> is_transaction_raw("...")
True
```

swap.providers.bitcoin.utils.**decode_transaction_raw**(*transaction_raw: str*, *offline: bool = True*, *headers: dict = {}*, *timeout: int = 60*) → dict

Decode Bitcoin transaction raw.

Parameters

- **transaction_raw** (*str*) – Bitcoin transaction raw.
- **offline** (*bool*) – Offline decode, defaults to True.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Decoded Bitcoin transaction raw.

```
>>> from swap.providers.bitcoin.utils import decode_transaction_raw
>>> transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTg4OGJlN2VjMDY1MDk3ZDk1NjY0NzYzZjI3NmQ0MjU1NTJkNzM1Z
↪"
>>> decode_transaction_raw(transaction_raw)
{'fee': 678, 'type': 'bitcoin_fund_unsigned', 'tx': {'hex':
↪'0200000001888be7ec065097d95664763f276d425552d735fb1d974ae78bf72106dca0f391010000000ffffffff0
↪', 'txid': 'abc70fd3466aec9478ea3115200a84f993204ad1f614fe08e92ecc5997a0d3ba',
↪'hash': 'abc70fd3466aec9478ea3115200a84f993204ad1f614fe08e92ecc5997a0d3ba',
↪'size': 117, 'vsize': 117, 'version': 2, 'locktime': 0, 'vin': [{'txid':
↪'91f3a0dc0621f78be74a971dfb35d75255426d273f766456d9975006ece78b88', 'vout': 1,
↪'scriptSig': {'asm': '', 'hex': ''}, 'sequence': '4294967295'}], 'vout': [{
↪'value': '0.00010000', 'n': 0, 'scriptPubKey': {'asm': 'OP_HASH160␣
↪2bb013c3e4beb08421dedcf815cb65a5c388178b OP_EQUAL', 'hex':
↪'a9142bb013c3e4beb08421dedcf815cb65a5c388178b87', 'type': 'p2sh', 'address':
↪'2MwEDybGC34949zgzWX4M9FHmE3crDSUydP'}}, {'value': '0.00974268', 'n': 1,
↪'scriptPubKey': {'asm': 'OP_DUP OP_HASH160␣
↪64a8390b0b1685fcbf2d4b457118dc8da92d5534 OP_EQUALVERIFY OP_CHECKSIG', 'hex':
↪'76a91464a8390b0b1685fcbf2d4b457118dc8da92d553488ac', 'type': 'p2pkh', 'address
↪': 'mphBPZf15cRFcL5tUq6mCbE84XobZ1vg7Q'}}]}, 'network': 'testnet'}
```

swap.providers.bitcoin.utils.**submit_transaction_raw**(*transaction_raw: str*, *headers: dict = {}*, *timeout: int = 60*) → dict

Submit transaction raw to Bitcoin blockchain.

Parameters

- **transaction_raw** (*str*) – Bitcoin transaction raw.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bitcoin transaction id, fee, type and date.

```
>>> from swap.providers.bitcoin.utils import submit_transaction_raw
>>> transaction_raw =
↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTg4OGJlN2VjMDY1MDk3ZDk1NjY0NzYzZjI3NmQ0MjU1NTJkNzM1Z
↪"
>>> submit_transaction_raw(transaction_raw)
{'fee': '...', 'type': '...', 'transaction_id': '...', 'network': '...', 'date':
↪'...'}
```

swap.providers.bitcoin.utils.**get_address_type**(*address: str*) → str

Get Bitcoin address type.

Parameters **address** (*str*) – Bitcoin address.

Returns str – Bitcoin address type (P2PKH, P2SH).

```
>>> from swap.providers.bitcoin.utils import get_address_type
>>> get_address_type(address="mrmtGq2HMmqAogSsGDjCtXUpxrb7rHThFH")
"p2pkh"
```

swap.providers.bitcoin.utils.**get_address_hash**(*address: str*, *script: bool = False*) → Union[str, btcpy.structs.script.P2pkhScript, btcpy.structs.script.P2shScript]

Get Bitcoin address hash.

Parameters

- **address** (*str*) – Bitcoin address.
- **script** (*bool*) – Return script (P2pkhScript, P2shScript), default to False.

Returns str – Bitcoin address hash.

```
>>> from swap.providers.bitcoin.utils import get_address_hash
>>> get_address_hash(address="mrmtGq2HMmqAogSsGDjCtXUpxrb7rHThFH", script=False)
"7b7c4431a43b612a72f8229935c469f1f6903658"
```

CHAPTER

SEVEN

BYTOM

Bytom is a protocol of multiple byte assets. Heterogeneous byte-assets operate in different forms on the Bytom Blockchain and atomic assets (warrants, securities, dividends, bonds, intelligence information, forecasting information and other information that exist in the physical world) can be registered, exchanged, gambled via Bytom.

7.1 Wallet

The implementation of Hierarchical Deterministic (HD) wallets generator for Bytom blockchain.

class swap.providers.bytom.wallet.**Wallet**(*network: str = 'mainnet'*)

Bytom Wallet class.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns Wallet – Bytom wallet instance.

Note: Bytom has only two networks, `mainnet`, `solonet` and `testnet`.

from_entropy(*entropy: str*, *passphrase: Optional[str] = None*, *language: str = 'english'*) → *swap.providers.bytom.wallet.Wallet*

Initiate Bytom wallet from entropy.

Parameters

- **entropy** (*str*) – Bytom wallet entropy.
- **passphrase** (*str*) – Bytom wallet passphrase, default to None.
- **language** (*str*) – Bytom wallet language, default to english.

Returns Wallet – Bytom wallet instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_mnemonic(*mnemonic: str*, *passphrase: Optional[str] = None*, *language: Optional[str] = None*) → *swap.providers.bytom.wallet.Wallet*

Initialize Bytom wallet from mnemonic.

Parameters

- **mnemonic** (*str*) – Bytom wallet mnemonic.
- **passphrase** (*str*) – Bytom wallet passphrase, default to None.

- **language** (*str*) – Bytom wallet language, default to english.

Returns Wallet – Bytom wallet class instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_mnemonic("extend length miss suit broken rescue around harbor␣
↪vehicle vicious jelly quality")
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_seed(*seed: str*) → *swap.providers.bytom.wallet.Wallet*

Initialize Bytom wallet from seed.

Parameters **seed** (*str*) – Bytom wallet seed.

Returns Wallet – Bytom wallet class instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_seed(
↪"51a0f6fb9abd5e5aa27f42dd375d8e4fc6944c704c859454e557fc419d3979e5a50273743c93e5035244adb09
↪")
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_xprivate_key(*xprivate_key: str*) → *swap.providers.bytom.wallet.Wallet*

Initiate Bytom wallet from xprivate key.

Parameters **xprivate_key** (*str.*) – Bytom wallet xprivate key.

Returns Wallet – Bytom wallet instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_xprivate_key(
↪"d0d4862706cfe7d2ffdf53d00fba1d524587972e2eb0226ce9fff3ca58e5a14f031f74b091a04f3ff6b172254
↪")
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_private_key(*private_key: str*) → *swap.providers.bytom.wallet.Wallet*

Initialize Bytom wallet from private key.

Parameters **private_key** (*str.*) – Bytom private key.

Returns Wallet – Bytom wallet instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_private_key(
↪"40d171e524c5d366c87f789e293e9e8d63ab95be796b3c04b63db29321eaa14f92de5a98859ca593b63f9e421
↪")
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_path(*path: str*) → *swap.providers.bytom.wallet.Wallet*

Drive Bytom wallet from path.

Parameters **path** (*str*) – Bytom wallet path.

Returns Wallet – Bytom wallet instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_indexes(*indexes: List[str]*) → *swap.providers.bytom.wallet.Wallet*

Drive Bytom wallet from indexes.

Parameters **indexes** (*list.*) – Bytom derivation indexes.

Returns Wallet – Bytom wallet class instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_xprivate_key(
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪")
>>> wallet.from_indexes(["2c000000", "99000000", "01000000", "00000000",
↪"01000000"])
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

from_index(*index: int*, *harden: bool = False*) → *swap.providers.bytom.wallet.Wallet*

Drive Bytom wallet from index.

Parameters

- **index** (*int*) – Bytom wallet index.
- **harden** (*bool*) – Use harden, default to False.

Returns Wallet – Bytom wallet instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_index(44)
>>> wallet.from_index(153)
>>> wallet.from_index(1)
>>> wallet.from_index(0)
>>> wallet.from_index(1)
<swap.providers.bytom.wallet.Wallet object at 0x040DA268>
```

clean_derivation() → *swap.providers.bytom.wallet.Wallet*

Clean derivation Bytom wallet.

Returns Wallet – Bytom wallet instance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.indexes()
["2c000000", "99000000", "01000000", "00000000", "01000000"]
>>> wallet.path()
"m/44/153/1/0/1"
>>> wallet.clean_derivation()
>>> wallet.indexes()
[]
```

(continues on next page)

(continued from previous page)

```
>>> wallet.path()
None
```

entropy() → Optional[str]
Get Bytom wallet entropy.

Returns str – Bytom wallet entropy.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.entropy()
"50f002376c81c96e430b48f1fe71df57"
```

mnemonic() → Optional[str]
Get Bytom wallet mnemonic.

Returns str – Bytom wallet mnemonic.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.mnemonic()
"extend length miss suit broken rescue around harbor vehicle vicious jelly␣
↪quality"
```

passphrase() → Optional[str]
Get Bytom wallet passphrase.

Returns str – Bytom wallet passphrase.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57", passphrase=
↪"meherett")
>>> wallet.passphrase()
"meherett"
```

language() → Optional[str]
Get Bytom wallet language.

Returns str – Bytom wallet language.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.language()
"english"
```

seed() → Optional[str]
Get Bytom wallet seed.

Returns str – Bytom wallet seed.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
```

(continues on next page)

(continued from previous page)

```
>>> wallet.seed()

↪"51a0f6fb9abd5e5aa27f42dd375d8e4fc6944c704c859454e557fc419d3979e5a50273743c93e5035244adb09
↪"
```

path() → Optional[str]
Get Bytom wallet derivation path.

Returns str – Bytom derivation path.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet", change=True, address=3)
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.path()
"m/44/153/1/0/1"
```

indexes() → list
Get Bytom wallet derivation indexes.

Returns list – Bytom derivation indexes.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.indexes()
['2c000000', '99000000', '01000000', '00000000', '01000000']
```

xprivate_key() → Optional[str]
Get Bytom wallet xprivate key.

Returns str – Bytom xprivate key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.xprivate_key()

↪"d0d4862706cfe7d2ffdf53d00fba1d524587972e2eb0226ce9fff3ca58e5a14f031f74b091a04f3ff6b172254
↪"
```

xpublic_key() → Optional[str]
Get Bytom wallet xpublic key.

Returns str – Bytom xpublic key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.xpublic_key()

↪"5086c8522be8c3b8674d72a6b9aa19eef43ef1992a482e71f389d99159accc39031f74b091a04f3ff6b172254
↪"
```

expand_xprivate_key() → Optional[str]
Get Bytom wallet expand xprivate key.

Returns str – Bytom expand xprivate key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.expand_xprivate_key()

↪"d0d4862706cfe7d2ffdf53d00fba1d524587972e2eb0226ce9fff3ca58e5a14f7c15b70c1b0fc7a393fdb443c
↪"
```

child_xprivate_key() → Optional[str]
Get Bytom child wallet xprivate key.

Returns str – Bytom child xprivate key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.child_xprivate_key()

↪"40d171e524c5d366c87f789e293e9e8d63ab95be796b3c04b63db29321eaa14f92de5a98859ca593b63f9e421
↪"
```

child_xpublic_key() → Optional[str]
Get Bytom child wallet xpublic key.

Returns str – Bytom child xpublic key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.child_xpublic_key()

↪"ffbbd79031060ef98fee4deda59818732e7665de15df34dff209d1f6f9a1443992de5a98859ca593b63f9e421
↪"
```

guid() → Optional[str]
Get Bytom wallet Blockcenter GUID.

Returns str – Bytom Blockcenter GUID.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.guid()
"e34d612c-f1f9-42c6-8b14-3d93c5b21715"
```

private_key() → str
Get Bytom wallet private key.

Returns str – Bytom private key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.private_key()

↪"40d171e524c5d366c87f789e293e9e8d63ab95be796b3c04b63db29321eaa14f92de5a98859ca593b63f9e421
↪"
```

public_key() → str
Get Bytom wallet public key.

Returns str – Bytom public key.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.public_key()
"ffbbd79031060ef98fee4deda59818732e7665de15df34dff209d1f6f9a14439"
```

program()
Get Bytom wallet control program.

Returns str – Bytom control program.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.program()
"00144eaaa3205545eac08fb3a2d1b1570b67c3b46016"
```

address(*network: Optional[str] = 'mainnet'*) → str
Get Bytom wallet address.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns str – Bytom wallet address.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_indexes(["2c000000", "99000000", "01000000", "00000000",
↪"01000000"])
>>> wallet.address(network="mainnet")
"bm1qf642xgz4gh4vpran5tgmz4ctvlpmgcqkmhn2le"
```

balance(*asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*) → int
Get Bytom wallet balance.

Parameters **asset** (*str*) – Bytom asset id, defaults to BTM asset.

Returns int – Bytom wallet balance.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.balance()
2450000000
```

utxos(*asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*, *limit: int = 15*) → list
Get Bytom wallet unspent transaction output (UTXO's).

Parameters

- **asset** (*str*) – Bytom asset id, defaults to BTM asset.
- **limit** (*int*) – Bytom balance, default is 15.

Returns list – Bytom unspent transaction outputs.

```
>>> from swap.providers.bytom.wallet import Wallet
>>> wallet = Wallet(network="mainnet")
>>> wallet.from_entropy("50f002376c81c96e430b48f1fe71df57")
>>> wallet.from_path("m/44/153/1/0/1")
>>> wallet.utxos()
[{'hash': 'e152f88d33c6659ad823d15c5c65b2ed946d207c42430022cba9bb9b9d70a7a4',
↪'asset': 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
↪ 'amount': 587639800}, {'hash':
↪'88289fa4c7633574931be7ce4102aeb24def0de20e38e7d69a5ddd6efc116b95', 'asset
↪': 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
↪'amount': 8160000}, {'hash':
↪'f71c68f921b434cc2bcd469d26e7927aa6db7500e4cdeef814884f11c10f5de2', 'asset
↪': 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
↪'amount': 10000}, {'hash':
↪'e46cfecc1f1a26413172ce81c78affb19408e613915642fa5fb04d3b0a4ffa65', 'asset
↪': 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff',
↪'amount': 100}]
```

7.2 Hash Time Lock Contract (HTLC)

Bytom Hash Time Lock Contract (HTLC).

class swap.providers.bytom.htlc.**HTLC**(*network: str = 'mainnet'*)

Bytom Hash Time Lock Contract (HTLC).

Parameters **network** (*str*) – Bytom network, defaults to testnet.

Returns HTLC – Bytom HTLC instance.

Note: Bytom has only three networks, `mainnet`, `solonet` and `testnet`.

---

build_htlc(*secret_hash: str*, *recipient_public_key: str*, *sender_public_key: str*, *sequence: int = 1000*, *use_script: bool = False*) → *swap.providers.bytom.htlc.HTLC*

Build Bytom Hash Time Lock Contract (HTLC).

Parameters

- **secret_hash** (*str*) – secret sha-256 hash.
- **recipient_public_key** (*str*) – Bytom recipient public key.
- **sender_public_key** (*str*) – Bytom sender public key.
- **sequence** (*int*) – Bytom sequence number(expiration block), defaults to Bytom config sequence.
- **use_script** (*bool*) – Initialize HTLC by using script, default to False.

Returns HTLC – Bytom Hash Time Lock Contract (HTLC) instance.

```
>>> from swap.providers.bytom.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="mainnet")
```

(continues on next page)

(continued from previous page)

```
>>> htlc.build_htlc(secret_hash=sha256("Hello Meheret!"), recipient_public_
↪key="3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e",␣
↪sender_public_key=
↪"91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2",␣
↪sequence=1000, use_script=False)
<swap.providers.bytom.htlc.HTLC object at 0x0409DAF0>
```

from_bytecode(*bytecode: str*) → *swap.providers.bytom.htlc.HTLC*

Initialize Bytom Hash Time Lock Contract (HTLC) from bytecode.

Parameters **bytecode** (*str*) – Bytom bytecode.

Returns HTLC – Bytom Hash Time Lock Contract (HTLC) instance.

```
>>> from swap.providers.bytom.htlc import HTLC
>>> htlc = HTLC(network="testnet")
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> htlc.from_bytecode(bytecode=bytecode)
<swap.providers.bitcoin.htlc.HTLC object at 0x0409DAF0>
```

bytecode() → str

Get Bytom Hash Time Lock Contract (HTLC) bytecode.

Returns str – Bytom HTLC bytecode.

```
>>> from swap.providers.bytom.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="mainnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e",
↪"91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2", 1000,␣
↪False)
>>> htlc.bytecode()

↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
```

opcode() → Optional[str]

Get Bytom Hash Time Lock Contract (HTLC) OP_Code.

Returns str – Bytom HTLC opcode.

```
>>> from swap.providers.bytom.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="mainnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e",
↪"91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2", 1000,␣
↪False)
>>> htlc.opcode()
"0xe803 0x91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2␣
↪0x3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e␣
↪0x3a26da82ead15a80533a02696656b14b5dbfd84eb14790f2e1be5e9e45820eeb DEPTH␣
↪0x547a6416000000557aa888537a7cae7cac631f000000537acd9f6972ae7cac FALSE␣
↪CHECKPREDICATE"
```

hash() → str
Get Bytom Hash Time Lock Contract (HTLC) hash.

Returns str – Bytom HTLC hash.

```
>>> from swap.providers.bytom.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="mainnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e",
↪"91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2", 1000,␣
↪False)
>>> htlc.hash()
"4f8f0e88d0a44b3d884b07b6dd4536518ffcbb596a91ca0e6b2f37e96463bbfc"
```

address() → str
Get Bytom Hash Time Lock Contract (HTLC) address.

Returns str – Bytom HTLC address.

```
>>> from swap.providers.bytom.htlc import HTLC
>>> from swap.utils import sha256
>>> htlc = HTLC(network="mainnet")
>>> htlc.build_htlc(sha256("Hello Meheret!"),
↪"3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e",
↪"91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2", 1000,␣
↪False)
>>> htlc.address()
"bm1qf78sazxs539nmzztq7md63fk2x8lew6ed2gu5rnt9um7jerrh07q3yf5q8"
```

7.3 Transaction

Bitcoin transaction in blockchain network.

class swap.providers.bytom.transaction.**Transaction**(*network: str = 'mainnet'*)
Bytom Transaction.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns Transaction – Bytom transaction instance.

Note: Bytom has only three networks, `mainnet`. `solonet` and `mainnet`.

---

fee() → int
Get Bitcoin transaction fee.

Returns int – Bitcoin transaction fee.

```
>>> from swap.providers.bytom.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("mainnet")
>>> claim_transaction.build_transaction(
↪"bm1q3plwvmvy4qhjmp5zffzmk50aagpujt6f5je85p",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> claim_transaction.fee()
10000000
```

hash() → str
Get Bytom transaction hash.

Returns str – Bytom transaction id/hash.

```
>>> from swap.providers.bytom.transaction import FundTransaction
>>> fund_transaction = FundTransaction("mainnet")
>>> fund_transaction.build_transaction(
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7",
↪"bm1qf78sazxs539nmzztq7md63fk2x8lew6ed2gu5rnt9um7jerrh07q3yf5q8", 10000)
>>> fund_transaction.hash()
"2993414225f65390220730d0c1a356c14e91bca76db112d37366df93e364a492"
```

json() → dict
Get Bytom transaction json format.

Returns dict – Bytom transaction json format.

```
>>> from swap.providers.bytom.transaction import RefundTransaction
>>> refund_transaction = RefundTransaction("mainnet")
>>> refund_transaction.build_transaction(
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7",
↪"481c00212c552fbdf537fcc88c1006a69bdd3130f593965f6ff4f91818a1c6e1", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> refund_transaction.json()
{"hash": "2993414225f65390220730d0c1a356c14e91bca76db112d37366df93e364a492",
↪"status_fail": false, "size": 379, "submission_timestamp": 0, "memo": "",
↪"inputs": [{"script": "00142cda4f99ea8112e6fa61cdd26157ed6dc408332a",
↪"address": "bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", "asset":
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff", "amount
↪": 2450000000, "type": "spend"}], "outputs": [{"utxo_id":
↪"5edccebe497893c289121f9e365fdeb34c97008b9eb5a9960fe9541e7923aabc", "script
↪":
↪"01642091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e220ac13c0bb1445423a6
↪", "address": "smart contract", "asset":
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff", "amount
↪": 1000, "type": "control"}, {"utxo_id":
↪"f8cfbb692db1963be88b09c314adcc9e19d91c6c019aa556fb7cb76ba8ffa1fa", "script
↪": "00142cda4f99ea8112e6fa61cdd26157ed6dc408332a", "address":
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", "asset":
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff", "amount
↪": 2439999000, "type": "control"}], "fee": 10000000, "balances": [{"asset":
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff", "amount
↪": "-10001000"}], "types": ["ordinary"]}
```

raw() → str
Get Bytom transaction raw.

Returns str – Bytom transaction raw.

```
>>> from swap.providers.bytom.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("mainnet")
>>> claim_transaction.build_transaction(
↪"bm1q3plwvmvy4qhjmp5zffzmk50aagpujt6f5je85p",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> claim_transaction.raw()

↪"070100010160015e7f2d7ecec3f61d30d0b2968973a3ac8448f0599ea20dce883b48c903c4d6e87fffffffff
↪"
```

(continues on next page)

(continued from previous page)

type() → str

Get Bitcoin signature transaction type.

Returns str – Bitcoin signature transaction type.

```
>>> from swap.providers.bytom.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("mainnet")
>>> claim_transaction.build_transaction(
↪"bm1q3plwvmvy4qhjmp5zffzmk50aagpujt6f5je85p",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> claim_transaction.type()
"bitcoin_claim_unsigned"
```

unsigned_datas(*detail: bool = False*) → list

Get Bytom transaction unsigned datas(messages) with instruction.

Parameters **detail** (*bool*) – Bytom unsigned datas to see detail, defaults to False.

Returns list – Bytom transaction unsigned datas.

```
>>> from swap.providers.bytom.transaction import FundTransaction
>>> from swap.providers.bytom.solver import FundSolver
>>> from swap.providers.bytom.wallet import Wallet
>>> sender_wallet = Wallet("mainnet").from_mnemonic("indicate warm sock␣
↪mistake code spot acid ribbon sing over taxi toast").from_path("m/44/153/1/
↪0/1")
>>> fund_solver = FundSolver(sender_wallet.xprivate_key())
>>> fund_transaction = FundTransaction("mainnet")
>>> fund_transaction.build_transaction(sender_wallet.address(),
↪"bm1qf78sazxs539nmzztq7md63fk2x81ew6ed2gu5rnt9um7jerrh07q3yf5q8", 10000)
>>> fund_transaction.unsigned_datas(solver=fund_solver)
[{'datas': ['38601bf7ce08dab921916f2c723acca0451d8904649bbec16c2076f1455dd1a2
↪'], 'public_key':
↪'91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2', 'network
↪': 'mainnet', 'path': 'm/44/153/1/0/1'}]
```

sign(**args*, ***kwargs*)

Bytom sign unsigned transaction datas.

Parameters

- **private_key** (*str*) – Bytom private key, default to None.
- **xprivate_key** (*str*) – Bytom xprivate key, default to None.
- **account** (*int*) – Bytom derivation account, defaults to 1.
- **change** (*bool*) – Bytom derivation change, defaults to False.
- **address** (*int*) – Bytom derivation address, defaults to 1.
- **path** (*str.*) – Bytom derivation path, default to None.
- **indexes** (*list.*) – Bytom derivation indexes, default to None.

Returns Transaction – Bytom transaction instance.

```
>>> from pybytom.transaction import Transaction
>>> transaction = Transaction(network="mainnet")
>>> transaction.build_transaction("f0ed6ddd-9d6b-49fd-8866-a52d1083a13b",␣
↪inputs=[...], outputs=[...])
>>> transaction.sign(xprivate_key)
<pybytom.transaction.transaction.Transaction object at 0x0409DAF0>
```

signatures() → list

Get Bytom transaction signatures(signed datas).

Returns list – Bytom transaction signatures.

```
>>> from swap.providers.bytom.transaction import FundTransaction
>>> from swap.providers.bytom.solver import FundSolver
>>> from swap.providers.bytom.wallet import Wallet
>>> sender_wallet = Wallet("mainnet").from_mnemonic("indicate warm sock␣
↪mistake code spot acid ribbon sing over taxi toast").from_path("m/44/153/1/
↪0/1")
>>> fund_solver = FundSolver(sender_wallet.xprivate_key())
>>> fund_transaction = FundTransaction("mainnet")
>>> fund_transaction.build_transaction(sender_wallet.address(),
↪"bm1qf78sazxs539nmzztq7md63fk2x8lew6ed2gu5rnt9um7jerrh07q3yf5q8", 10000)
>>> fund_transaction.sign(solver=fund_solver)
>>> fund_transaction.signatures()
[[
↪'8ca69a01def05118866681bc7008971efcff40895285297e0d6bd791220a36d6ef85a11abc48438de21f0256c
↪']]
```

7.3.1 FundTransaction

class swap.providers.bytom.transaction.**FundTransaction**(*network: str = {'blockcenter': {'v2': 'https://bcapi.bystack.com/api/v2/btm', 'v3': 'https://bcapi.bystack.com/bytom/v3'}, 'blockmeta': 'https://blockmeta.com/api/v3', 'bytom-core': 'http://localhost:9888'}*)

Bytom Fund transaction.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns FundTransaction – Bytom fund transaction instance.

Warning: Do not forget to build transaction after initialize fund transaction.

build_transaction(*address: str*, *htlc_address: str*, *amount: int*, *asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*) → *swap.providers.bytom.transaction.FundTransaction*

Build Bytom fund transaction.

Parameters

- **address** (*str*) – Bytom sender wallet address.

- **htlc_address** (*str*) – Bytom Hash Time Lock Contract (HTLC) address.
- **amount** (*int*) – Bytom amount to fund.
- **asset** (*str*) – Bytom asset id, defaults to BTM asset.

Returns FundTransaction – Bytom fund transaction instance.

```
>>> from swap.providers.bytom.transaction import FundTransaction
>>> fund_transaction = FundTransaction("mainnet")
>>> fund_transaction.build_transaction(address=
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", htlc_address=
↪"bm1qf78sazxs539nmzztq7md63fk2x8lew6ed2gu5rnt9um7jerrh07q3yf5q8",␣
↪amount=10000)
<swap.providers.bytom.transaction.FundTransaction object at 0x0409DAF0>
```

sign(*solver:* swap.providers.bytom.solver.FundSolver) → *swap.providers.bytom.transaction.FundTransaction*

Sign Bytom fund transaction.

Parameters **solver** (bytom.solver.FundSolver) – Bytom fund solver.

Returns FundTransaction – Bytom fund transaction instance.

```
>>> from swap.providers.bytom.transaction import FundTransaction
>>> from swap.providers.bytom.solver import FundSolver
>>> from swap.providers.bytom.wallet import Wallet
>>> sender_wallet = Wallet("mainnet").from_mnemonic("indicate warm sock␣
↪mistake code spot acid ribbon sing over taxi toast").from_path("m/44/153/1/
↪0/1")
>>> fund_solver = FundSolver(sender_wallet.xprivate_key())
>>> fund_transaction = FundTransaction("mainnet")
>>> fund_transaction.build_transaction(sender_wallet.address(),
↪"bm1qf78sazxs539nmzztq7md63fk2x8lew6ed2gu5rnt9um7jerrh07q3yf5q8", 10000)
>>> fund_transaction.sign(solver=fund_solver)
<swap.providers.bytom.transaction.FundTransaction object at 0x0409DAF0>
```

transaction_raw() → str

Get Bytom fund transaction raw.

Returns str – Bytom fund transaction raw.

```
>>> from swap.providers.bytom.transaction import FundTransaction
>>> fund_transaction = FundTransaction("mainnet")
>>> fund_transaction.build_transaction(
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7",
↪"bm1qf78sazxs539nmzztq7md63fk2x8lew6ed2gu5rnt9um7jerrh07q3yf5q8", 10000)
>>> fund_transaction.transaction_raw()

↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTJjMzkyMjE3NDgzOTA2ZjkwMmU3M2M0YmMxMzI4NjRkZTU4M
↪"
```

7.3.2 ClaimTransaction

class swap.providers.bytom.transaction.**ClaimTransaction**(*network: str = 'mainnet'*)
Bytom Claim transaction.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns ClaimTransaction – Bytom claim transaction instance.

> **Warning:** Do not forget to build transaction after initialize claim transaction.

build_transaction(*address: str*, *transaction_id: str*, *amount: int*, *asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*) → *swap.providers.bytom.transaction.ClaimTransaction*
Build Bytom claim transaction.

Parameters

- **address** (*str*) – Bytom recipient wallet address.
- **transaction_id** (*str*) – Bytom fund transaction id to redeem.
- **amount** (*int*) – Bytom amount to withdraw.
- **asset** (*str*) – Bytom asset id, defaults to BTM asset.

Returns ClaimTransaction – Bytom claim transaction instance.

```
>>> from swap.providers.bytom.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("mainnet")
>>> claim_transaction.build_transaction(address=
↪"bm1q3plwvmvy4qhjmp5zffzmk50aagpujt6f5je85p", transaction_id=
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd",␣
↪amount=10000, asset=
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
<swap.providers.bytom.transaction.ClaimTransaction object at 0x0409DAF0>
```

sign(*solver:* swap.providers.bytom.solver.ClaimSolver) → *swap.providers.bytom.transaction.ClaimTransaction*
Sign Bytom claim transaction.

Parameters **solver** (bytom.solver.ClaimSolver) – Bytom claim solver.

Returns ClaimTransaction – Bytom claim transaction instance.

```
>>> from swap.providers.bytom.transaction import ClaimTransaction
>>> from swap.providers.bytom.solver import ClaimSolver
>>> from swap.providers.bytom.wallet import Wallet
>>> recipient_wallet = Wallet("mainnet").from_mnemonic("hint excuse upgrade␣
↪sleep easily deputy erase cluster section other ugly limit").from_path("m/
↪44/153/1/0/1")
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> claim_solver = ClaimSolver(recipient_wallet.xprivate_key(), "Hello␣
↪Meheret!", bytecode=bytecode)
>>> claim_transaction = ClaimTransaction("mainnet")
>>> claim_transaction.build_transaction(recipient_wallet.address(),
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
```

(continues on next page)

(continued from previous page)

```
>>> claim_transaction.sign(solver=claim_solver)
<swap.providers.bytom.transaction.ClaimTransaction object at 0x0409DAF0>
```

transaction_raw() → str

Get Bytom claim transaction raw.

Returns str – Bytom claim transaction raw.

```
>>> from swap.providers.bytom.transaction import ClaimTransaction
>>> claim_transaction = ClaimTransaction("mainnet")
>>> claim_transaction.build_transaction(
↪"bm1q3plwvmvy4qhjmp5zffzmk50aagpujt6f5je85p",
↪"1006a6f537fcc4888c65f6ff4f91818a1c6e19bdd3130f59391c00212c552fbd", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> claim_transaction.transaction_raw()

↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTJjMzkyMjE3NDgzOTA2ZjkwMmU3M2M0YmMxMzI4NjRkZTU4M
↪"
```

7.3.3 RefundTransaction

class swap.providers.bytom.transaction.**RefundTransaction**(*network: str = 'mainnet'*)

Bytom Refund transaction.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns RefundTransaction – Bytom refund transaction instance.

Warning: Do not forget to build transaction after initialize refund transaction.

build_transaction(*address: str*, *transaction_id: str*, *amount: int*, *asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*) → *swap.providers.bytom.transaction.RefundTransaction*

Build Bytom refund transaction.

Parameters

- **address** (*str*) – Bytom sender wallet address.
- **transaction_id** (*str*) – Bytom fund transaction id to redeem.
- **amount** (*int*) – Bytom amount to withdraw.
- **asset** (*str*) – Bytom asset id, defaults to BTM asset.

Returns RefundTransaction – Bytom refund transaction instance.

```
>>> from swap.providers.bytom.transaction import RefundTransaction
>>> refund_transaction = RefundTransaction("mainnet")
>>> refund_transaction.build_transaction(address=
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", transaction_id=
↪"481c00212c552fbdf537fcc88c1006a69bdd3130f593965f6ff4f91818a1c6e1",␣
↪amount=10000, asset=
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
<swap.providers.bytom.transaction.RefundTransaction object at 0x0409DAF0>
```

sign(*solver:* swap.providers.bytom.solver.RefundSolver) → *swap.providers.bytom.transaction.RefundTransaction*
Sign Bytom refund transaction.

Parameters **solver** (bytom.solver.RefundSolver) – Bytom refund solver.

Returns RefundTransaction – Bytom refund transaction instance.

```
>>> from swap.providers.bytom.transaction import RefundTransaction
>>> from swap.providers.bytom.solver import RefundSolver
>>> from swap.providers.bytom.wallet import Wallet
>>> sender_wallet = Wallet("mainnet").from_mnemonic("indicate warm sock␣
↪mistake code spot acid ribbon sing over taxi toast").from_path("m/44/153/1/
↪0/1")
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> refund_solver = RefundSolver(sender_wallet.xprivate_key(),␣
↪bytecode=bytecode)
>>> refund_transaction = RefundTransaction("mainnet")
>>> refund_transaction.build_transaction(sender_wallet.address(),
↪"481c00212c552fbdf537fcc88c1006a69bdd3130f593965f6ff4f91818a1c6e1", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> refund_transaction.sign(solver=refund_solver)
<swap.providers.bytom.transaction.RefundTransaction object at 0x0409DAF0>
```

transaction_raw() → str
Get Bytom refund transaction raw.

Returns str – Bytom refund transaction raw.

```
>>> from swap.providers.bytom.transaction import RefundTransaction
>>> refund_transaction = RefundTransaction("mainnet")
>>> refund_transaction.build_transaction(
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7",
↪"481c00212c552fbdf537fcc88c1006a69bdd3130f593965f6ff4f91818a1c6e1", 10000,
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff")
>>> refund_transaction.transaction_raw()

↪"eyJmZWUiOiA2NzgsICJyYXciOiAiMDIwMDAwMDAwMTJjMzkyMjE3NDgzOTA2ZjkwMmU3M2M0YmMxMzI4NjRkZTU4M
↪"
```

7.4 Solver

Bytom solver.

7.4.1 FundSolver

class swap.providers.bytom.solver.**FundSolver**(*xprivate_key: str, account: int = 1, change: bool = False, address: int = 1, path: Optional[str] = None, indexes: Optional[List[str]] = None*)
Bytom Fund solver.

Parameters

- **xprivate_key** (*str*) – Bytom sender xprivate key.

- **account** (*int*) – Bytom derivation account, defaults to 1.
- **change** (*bool*) – Bytom derivation change, defaults to False.
- **address** (*int*) – Bytom derivation address, defaults to 1.
- **path** (*str*) – Bytom derivation path, defaults to None.
- **indexes** (*list*) – Bytom derivation indexes, defaults to None.

Returns FundSolver – Bytom fund solver instance.

```
>>> from swap.providers.bytom.solver import FundSolver
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f99711367189245
↪"
>>> fund_solver = FundSolver(xprivate_key=sender_xprivate_key)
<swap.providers.bytom.solver.FundSolver object at 0x03FCCA60>
```

7.4.2 ClaimSolver

class swap.providers.bytom.solver.**ClaimSolver**(*xprivate_key: str, secret_key: str, bytecode: str, account: int = 1, change: bool = False, address: int = 1, path: Optional[str] = None, indexes: Optional[List[str]] = None*)

Bytom Claim solver.

Parameters

- **xprivate_key** (*str*) – Bytom sender xprivate key.
- **secret_key** (*str*) – Secret password/passphrase.
- **bytecode** (*str*) – Bytom witness HTLC bytecode, defaults to None.
- **account** (*int*) – Bytom derivation account, defaults to 1.
- **change** (*bool*) – Bytom derivation change, defaults to False.
- **address** (*int*) – Bytom derivation address, defaults to 1.
- **path** (*str*) – Bytom derivation path, defaults to None.
- **indexes** (*list*) – Bytom derivation indexes, defaults to None.

Returns ClaimSolver – Bytom claim solver instance.

```
>>> from swap.providers.bytom.solver import ClaimSolver
>>> recipient_xprivate_key =
↪"58dd4094155bbebf2868189231c47e4e0edbd9f74545f843c9537259e1d7a656983aef283d0ccebecc2d33577a9f6
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa031d45
↪"
>>> claim_solver = ClaimSolver(xprivate_key=recipient_xprivate_key, secret_key=
↪"Hello Meheret!", bytecode=bytecode)
<swap.providers.bytom.solver.ClaimSolver object at 0x03FCCA60>
```

7.4.3 RefundSolver

class swap.providers.bytom.solver.**RefundSolver**(*xprivate_key: str*, *bytecode: str*, *account: int = 1*, *change: bool = False*, *address: int = 1*, *path: Optional[str] = None*, *indexes: Optional[List[str]] = None*)

Bytom Refund solver.

Parameters

- **xprivate_key** (*str*) – Bytom sender xprivate key.
- **bytecode** (*str*) – Bytom witness HTLC bytecode, defaults to None.
- **account** (*int*) – Bytom derivation account, defaults to 1.
- **change** (*bool*) – Bytom derivation change, defaults to False.
- **address** (*int*) – Bytom derivation address, defaults to 1.
- **path** (*str*) – Bytom derivation path, defaults to None.
- **indexes** (*list*) – Bytom derivation indexes, defaults to None.

Returns RefundSolver – Bytom refund solver instance.

```
>>> from swap.providers.bytom.solver import RefundSolver
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f99711367189245
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa031d45
↪"
>>> refund_solver = RefundSolver(xprivate_key=sender_xprivate_key,␣
↪bytecode=bytecode)
<swap.providers.bytom.solver.RefundSolver object at 0x03FCCA60>
```

7.5 Signature

Bytom signature.

class swap.providers.bytom.signature.**Signature**(*network: str = 'mainnet'*)

Bytom Signature.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns Transaction – Bytom transaction instance.

Note: Bytom has only three networks, `mainnet`, `solonet` and `testnet`.

---

fee() → int

Get Bytom transaction fee.

Returns int – Bytom transaction fee.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.fee()
10000000
```

hash() → str

Get Bytom signature transaction hash.

Returns str – Bytom signature transaction hash or transaction id.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import ClaimSolver
>>> unsigned_claim_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTNwbHd2bXZ5NHFoam1wNXpmZnptazUwYWFncHVqdDZmN
↪"
>>> recipient_xprivate_key =
↪"58dd4094155bbebf2868189231c47e4e0edbd9f74545f843c9537259e1d7a656983aef283d0ccebecc2d33577
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> claim_solver = ClaimSolver(recipient_xprivate_key, "Hello Meheret!",␣
↪bytecode)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_claim_transaction_raw, claim_solver)
>>> signature.hash()
"d544ad2d08f9dda33b78953c74eede9c9eb5d80835695310b242d5796cfb91d6"
```

json() → dict

Get Bytom signature transaction json format.

Returns dict – Bytom signature transaction json format.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.json()
{"tx_id": "50b336ab6e055d9d4d65a9f2295b53270abd3816c23ba4c954841f399aa772d5",
↪"version": 1, "size": 405, "time_range": 0, "inputs": [{"type": "spend",
↪"asset_id":
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff", "asset_
↪definition": {}, "amount": 8160000, "control_program":
↪"00142cda4f99ea8112e6fa61cdd26157ed6dc408332a", "address":
↪"bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", "spent_output_id":
↪"88289fa4c7633574931be7ce4102aeb24def0de20e38e7d69a5ddd6efc1
↪id": "49e97e1685d5b08b82713e6acb6747bd176177141cb5618aeecca418c3afd03a",
↪"witness_arguments": [
↪"91ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2"], "sign_
↪data": "f7d3aa18b295cda6f2b1132c4231933cc92f3baca705974c5de378f9b695f0e2"}
```

(continues on next page)

(continued from previous page)

raw() → str
Get Bytom signature transaction raw.

Returns str – Bytom signature transaction raw.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.raw()

↪"07010002015f015d82e65f964d3c3532548dfde938462f566c95d3c90e6a3a182a0b3bdae46aa790fffffffff
↪"
```

type() → str
Get Bytom signature transaction type.

Returns str – Bytom signature transaction type.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import RefundSolver
>>> unsigned_refund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> refund_solver = RefundSolver(sender_xprivate_key, bytecode, 1000)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_refund_transaction_raw, refund_solver)
>>> signature.type()
"bytom_refund_signed"
```

sign(*transaction_raw: str*, *solver: Union[*swap.providers.bytom.solver.FundSolver, swap.providers.bytom.solver.ClaimSolver, swap.providers.bytom.solver.RefundSolver*]*) → Union[*swap.providers.bitcoin.signature.FundSignature*, *swap.providers.bitcoin.signature.ClaimSignature*, *swap.providers.bitcoin.signature.RefundSignature*]
Sign unsigned transaction raw.

Parameters

- **transaction_raw** (*str*) – Bytom unsigned transaction raw.
- **solver** (bytom.solver.FundSolver, bytom.solver.ClaimSolver, bytom.solver.RefundSolver) – Bytom solver

Returns FundSignature, ClaimSignature, RefundSignature – Bytom signature instance.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
>>> signature = Signature("mainnet")
>>> signature.sign(transaction_raw=unsigned_fund_transaction_raw, solver=fund_
↪solver)
<swap.providers.bytom.signature.FundSignature object at 0x0409DAF0>
```

unsigned_datas(**args*, ***kwargs*) → list

Get Bytom transaction unsigned datas with instruction.

Returns list – Bytom transaction unsigned datas.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import ClaimSolver
>>> unsigned_claim_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTNwbHd2bXZ5NHFoam1wNXpmZnptazUwYWFncHVqdDZmN
↪"
>>> recipient_xprivate_key =
↪"58dd4094155bbebf2868189231c47e4e0edbd9f74545f843c9537259e1d7a656983aef283d0ccebecc2d3357
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> claim_solver = ClaimSolver(recipient_xprivate_key, "Hello Meheret!",␣
↪bytecode)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_claim_transaction_raw, claim_solver)
>>> signature.unsigned_datas()
[{"datas": ["5172290a9858a4a07c603c741f6fd8e86715a8a4470eb237d0a2d8325c1706b7
↪"], "network": "mainnet", "path": null}, {"datas": [
↪"e41ab964701f20a23473340b11d5cbcfba9a373cedf284f809c0c61ce7d715da"],
↪"public_key":
↪"3e0a377ae4afa031d4551599d9bb7d5b27f4736d77f78cac4d476f0ffba5ae3e", "network
↪": "mainnet", "path": "m/44/153/1/0/1"}]
```

signatures() → list

Get Bytom transaction signatures(signed datas).

Returns list – Bytom transaction signatures.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
```

(continues on next page)

(continued from previous page)

```
>>> signature.signatures()
[[
↪"00c005bc114ec5f89b49e48526f90312b6f1a5274efd252049880023aeb8e7998c15e0baa4ff10fabbdae702f
↪"], [
↪"fbfb123ef062c9068dad22ce28de2a4e72f82076b6f98cb7e0909c11856260e7020aecbdca639f0b6e39d345c
↪"]]
```

transaction_raw() → str
Get Bytom signed transaction raw.

Returns str – Bytom signed transaction raw.

```
>>> from swap.providers.bytom.signature import Signature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
>>> signature = Signature("mainnet")
>>> signature.sign(unsigned_fund_transaction_raw, fund_solver)
>>> signature.transaction_raw()

↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
```

7.5.1 FundSignature

class swap.providers.bytom.signature.**FundSignature**(*network: str = 'mainnet'*)
Bytom Fund signature.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns FundSignature – Bytom fund signature instance.

sign(*transaction_raw: str*, *solver:* swap.providers.bytom.solver.FundSolver) → *swap.providers.bytom.signature.FundSignature*
Sign unsigned fund transaction raw.

Parameters

- **transaction_raw** (*str*) – Bytom unsigned fund transaction raw.
- **solver** (bytom.solver.FundSolver) – Bytom fund solver.

Returns FundSignature – Bytom fund signature instance.

```
>>> from swap.providers.bytom.signature import FundSignature
>>> from swap.providers.bytom.solver import FundSolver
>>> unsigned_fund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> fund_solver = FundSolver(sender_xprivate_key)
```

(continues on next page)

(continued from previous page)

```
>>> fund_signature = FundSignature("mainnet")
>>> fund_signature.sign(unsigned_fund_transaction_raw, fund_solver)
<swap.providers.bytom.signature.FundSignature object at 0x0409DAF0>
```

7.5.2 ClaimSignature

class swap.providers.bytom.signature.**ClaimSignature**(*network: str = 'mainnet'*)

Bytom Claim signature.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns ClaimSignature – Bytom claim signature instance.

sign(*transaction_raw: str*, *solver:* swap.providers.bytom.solver.ClaimSolver) → *swap.providers.bytom.signature.ClaimSignature*

Sign unsigned claim transaction raw.

Parameters

- **transaction_raw** (*str*) – Bytom unsigned claim transaction raw.
- **solver** (bytom.solver.ClaimSolver) – Bytom claim solver.

Returns ClaimSignature – Bytom claim signature instance.

```
>>> from swap.providers.bytom.signature import ClaimSignature
>>> from swap.providers.bytom.solver import ClaimSolver
>>> unsigned_claim_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTNwbHd2bXZ5NHFoam1wNXpmZnptazUwYWFncHVqdDZmN
↪"
>>> recipient_xprivate_key =
↪"58dd4094155bbebf2868189231c47e4e0edbd9f74545f843c9537259e1d7a656983aef283d0ccebecc2d33577
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> claim_solver = ClaimSolver(recipient_xprivate_key, "Hello Meheret!",␣
↪bytecode)
>>> claim_signature = ClaimSignature("mainnet")
>>> claim_signature.sign(transaction_raw=unsigned_claim_transaction_raw,␣
↪solver=claim_solver)
<swap.providers.bytom.signature.ClaimSignature object at 0x0409DAF0>
```

7.5.3 RefundSignature

class swap.providers.bytom.signature.**RefundSignature**(*network: str = 'mainnet'*)

Bytom Refund signature.

Parameters **network** (*str*) – Bytom network, defaults to mainnet.

Returns RefundSignature – Bytom claim signature instance.

sign(*transaction_raw: str*, *solver:* swap.providers.bytom.solver.RefundSolver) → *swap.providers.bytom.signature.RefundSignature*

Sign unsigned refund transaction raw.

Parameters

- **transaction_raw** (*str*) – Bytom unsigned refund transaction raw.
- **solver** (bytom.solver.RefundSolver) – Bytom refund solver.

Returns RefundSignature – Bytom refund signature instance.

```
>>> from swap.providers.bytom.signature import RefundSignature
>>> from swap.providers.bytom.solver import RefundSolver
>>> unsigned_refund_transaction_raw =
↪"eyJmZWUiOiAxMDAwMDAwMCwgImFkZHJlc3MiOiAiYm0xcTluZHlseDAyc3lmd2Q3bnBlaGZ4ejRsZGRoenFzdmUyZ
↪"
>>> sender_xprivate_key =
↪"205b15f70e253399da90b127b074ea02904594be9d54678207872ec1ba31ee51ef4490504bd2b6f997113671
↪"
>>> bytecode =
↪"02e8032091ff7f525ff40874c4f47f0cab42e46e3bf53adad59adef9558ad1b6448f22e2203e0a377ae4afa03
↪"
>>> refund_solver = RefundSolver(sender_xprivate_key, bytecode, 1000)
>>> refund_signature = RefundSignature("mainnet")
>>> refund_signature.sign(transaction_raw=unsigned_refund_transaction_raw,␣
↪solver=refund_solver)
<swap.providers.bytom.signature.RefundSignature object at 0x0409DAF0>
```

7.6 Remote Procedure Call (RPC)

Bytom remote procedure call.

swap.providers.bytom.rpc.**get_balance**(*address: str*, *asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → int

Get Bytom balance.

Parameters

- **address** (*str*) – Bytom address.
- **asset** (*str*) – Bytom asset, default to BTM asset.
- **network** (*str*) – Bytom network, defaults to mainnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 15.

Returns int – Bytom asset balance (NEU amount).

```
>>> from swap.providers.bytom.rpc import get_balance
>>> get_balance(address="bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", asset=
↪"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff", network=
↪"mainnet")
2580000000
```

swap.providers.bytom.rpc.**build_transaction**(*address: str*, *transaction: dict*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → dict

Build Bytom transaction.

Parameters

- **address** (*str*) – Bytom address.

- **transaction** (*dict*) – Bytom transaction.
- **network** (*str*) – Bytom network, defaults to mainnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bytom built transaction.

```
>>> from swap.providers.bytom.rpc import build_transaction
>>> build_transaction(address="bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7",␣
↪transaction={...}, network="mainnet")
{...}
```

swap.providers.bytom.rpc.**get_utxos**(*program: str*, *network: str = 'mainnet'*, *asset: str = 'ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff'*, *limit: int = 15*, *by: str = 'amount'*, *order: str = 'desc'*, *headers: dict = {}*, *timeout: int = 60*) → list

Get Bytom unspent transaction outputs (UTXO's).

Parameters

- **program** (*str*) – Bytom control program.
- **network** (*str*) – Bytom network, defaults to mainnet.
- **asset** (*str*) – Bytom asset id, defaults to BTM asset.
- **limit** (*int*) – Bytom utxo's limit, defaults to 15.
- **by** (*str*) – Sort by, defaults to amount.
- **order** (*str*) – Sort order, defaults to desc.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns list – Bytom unspent transaction outputs (UTXO's).

```
>>> from swap.providers.bytom.rpc import get_utxos
>>> get_utxos(program="00142cda4f99ea8112e6fa61cdd26157ed6dc408332a", network=
↪"mainnet")
[...]
```

swap.providers.bytom.rpc.**get_transaction**(*transaction_id: str*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → dict

Get Bytom transaction detail.

Parameters

- **transaction_id** (*str*) – Bytom transaction id.
- **network** (*str*) – Bytom network, defaults to mainnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bytom transaction detail.

```
>>> from swap.providers.bytom.rpc import get_transaction
>>> get_transaction(transaction_id=
↪"4e91bca76db112d3a356c17366df93e364a4922993414225f65390220730d0c1", network=
↪"mainnet")
{...}
```

swap.providers.bytom.rpc.**decode_raw**(*raw: str*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → dict

Decode original Bytom raw.

Parameters

- **raw** (*str*) – Bytom transaction raw.
- **network** (*str*) – Bytom network, defaults to mainnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bytom decoded transaction raw.

```
>>> from swap.providers.bytom.rpc import decode_raw
>>> decode_raw(raw="...", network="testnet")
{...}
```

swap.providers.bytom.rpc.**submit_raw**(*address: str*, *raw: str*, *signatures: list*, *network: str = 'mainnet'*, *headers: dict = {}*, *timeout: int = 60*) → str

Submit original Bytom raw into blockchain.

Parameters

- **address** (*str*) – Bytom address.
- **raw** (*str*) – Bytom transaction raw.
- **signatures** (*list*) – Bytom signed massage datas.
- **network** (*str*) – Bytom network, defaults to mainnet.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns str – Bytom submitted transaction id/hash.

```
>>> from swap.providers.bytom.rpc import submit_raw
>>> submit_raw(address="...", raw="...", signatures=[[...], ...], network="...")
"2993414225f65390220730d0c1a356c14e91bca76db112d37366df93e364a492"
```

7.7 Utils

Bytom Utils.

swap.providers.bytom.utils.**amount_converter**(*amount: Union[int, float]*, *symbol: str = 'NEU2BTM'*) → Union[int, float]

Amount converter

Parameters

- **amount** (*int, float*) – Bytom amount.
- **symbol** (*str*) – Bytom symbol, default to NEU2BTM.

Returns float – BTM asset amount.

```
>>> from swap.providers.bytom.utils import amount_converter
>>> amount_converter(amount=10_000_000, symbol="NEU2BTM")
0.1
```

swap.providers.bytom.utils.**is_network**(*network: str*) → bool

Check Bytom network.

Parameters **network** (*str*) – Bytom network.

Returns bool – Bytom valid/invalid network.

```
>>> from swap.providers.bytom.utils import is_network
>>> is_network("solonet")
True
```

swap.providers.bytom.utils.**is_address**(*address: str*, *network: Optional[str] = None*) → bool

Check Bytom address.

Parameters

- **address** (*str*) – Bytom address.
- **network** (*str*) – Bytom network, defaults to None.

Returns bool – Bytom valid/invalid address.

```
>>> from swap.providers.bytom.utils import is_address
>>> is_address("bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7", "mainnet")
True
```

swap.providers.bytom.utils.**is_transaction_raw**(*transaction_raw: str*) → bool

Check Bytom transaction raw.

Parameters **transaction_raw** (*str*) – Bytom transaction raw.

Returns bool – Bytom valid/invalid transaction raw.

```
>>> from swap.providers.bytom.utils import is_transaction_raw
>>> is_transaction_raw("...")
True
```

swap.providers.bytom.utils.**decode_transaction_raw**(*transaction_raw: str*, *headers: dict = {}*, *timeout: int = 60*) → dict

Decode Bytom transaction raw.

Parameters

- **transaction_raw** (*str*) – Bytom transaction raw.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Decoded Bytom transaction raw.

```
>>> from swap.providers.bytom.utils import decode_transaction_raw
>>> decode_transaction_raw(transaction_raw)
{'fee': ..., 'type': '...', 'address': '...', 'transaction': {...}, 'unsigned_
↪datas': [...], 'signatures': [...], 'network': '...'}
```

swap.providers.bytom.utils.**submit_transaction_raw**(*transaction_raw: str*, *headers: dict = {}*, *timeout: int = 60*) → dict

Submit Bytom transaction raw.

Parameters

- **transaction_raw** (*str*) – Bytom transaction raw.
- **headers** (*dict*) – Request headers, default to common headers.
- **timeout** (*int*) – Request timeout, default to 60.

Returns dict – Bytom submitted.

```
>>> from swap.providers.bytom.utils import submit_transaction_raw
>>> submit_transaction_raw(transaction_raw)
{'fee': ..., 'type': '...', 'transaction_id': '...', 'network': '...', 'date': '..
↪.'}
```

swap.providers.bytom.utils.**get_address_type**(*address: str*) → Optional[str]

Get Bytom address type.

Parameters **address** (*str*) – Bytom address.

Returns str – Bytom address type (P2WPKH, P2WSH).

```
>>> from swap.providers.bytom.utils import get_address_type
>>> get_address_type(address="bm1q9ndylx02syfwd7npehfxz4lddhzqsve2fu6vc7")
"p2wpkh"
```

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

W

X