BPX Bridge

BPX Bridge documentation

Introduction

BPX Bridge is the official cross-chain bridge between the BPX chain and other EVM-compatible networks.

The bridge can be used to transfer popular cryptocurrencies and stablecoins to the BPX chain as a 1:1 wrapped tokens, as well as transfer the BPX coin to other chains. This makes it possible to use reputable tokens in decentralized applications running on the BPX chain, as well as trading the BPX coin on popular DEXes, like Uniswap, Pancakeswap.

BPX Bridge is fully decentralized and non-custodial. It means that:

- It does not rely on any centralized infrastructure under the control of developers. Everything is implemented using on-chain smart contracts and trustless P2P communication.
- Anyone can participate in the validation of transactions passing through the bridge, earning money from transaction fees.
- Project developers do not have access to users funds, they do not store these funds in their wallets, the funds are locked in smart contracts.
- In the near future, after implementing governance contracts, BPX holders will decide about the protocol updates, supported networks and assets.

The official instance of the Bridge client application is available at:

https://bridge.bpxchain.cc

Chains list

BPX Bridge currently operates on the following chains:

Chain ID	Name	Bridge contract address

Assets list

BPX Bridge currently supports the following assets:

Original asset	Original chain	Wrapped asset	Wrapped asset chain
BPX Chain (**BPX**) *native token*	BPX Chain	BPX Chain (**BPX**) *0x602d550a4cA5eAe83195486AC85dC40032dAA787*	Arbitrum
BPX Chain (**BPX**) *native token*	BPX Chain	BPX Chain (**BPX**) *0x602d550a4cA5eAe83195486AC85dC40032dAA787*	Polygon
BPX Chain (**BPX**) *native token*	BPX Chain	BPX Chain (**BPX**) *0x602d550a4cA5eAe83195486AC85dC40032dAA787*	Avalanche C-Chain
Ethereum (**ETH**) *native token*	Arbitrum	□□ Ethereum (**ETH**) *0x8EF2a6c2a7f6ed5ef63D4d9667C1CCd09C2721C8*	BPX Chain
Polygon (**MATIC**) *native token*	Polygon	□□ Polygon (**MATIC**) *0x43A478B2A63098De753b345f0dFCBa24Da06849d*	BPX Chain
Avalanche (**AVAX**) *native token*	Avalanche C-Chain	□□ Avalanche (**AVAX**) *0xA303b26580e0eB63fAcEC99E26B4A8a6b549e550*	BPX Chain
Tether USD (**USDT**) *0xfd086bc7cd5c481dcc9c85ebe478a1c0b69fcbb9*	Arbitrum	□□ Tether USD (**USDT**) *0x67c7950934833E001cfc8c62E903EaeC69D34790*	BPX Chain
Tether USD (**USDT**) *0xc2132D05D31c914a87C6611C10748AEb04B58e8F*	Polygon	□□ Tether USD (**USDT**) *0x67c7950934833E001cfc8c62E903EaeC69D34790*	BPX Chain
Tether USD (**USDT**) *0x9702230A8Ea53601f5cD2dc00fDBc13d4dF4A8c7*	Avalanche C-Chain	□□ Tether USD (**USDT**) *0x67c7950934833E001cfc8c62E903EaeC69D34790*	BPX Chain

USD Coin (**USDC**) 0xaf88d065e77c8cC2239327C5EDb3A432268e5831	Arbitrum	🌉 USD Coin (**USDC**) *0x0b2ffE5CC332B72293Ca87d48ae9400bFd9Ba97e*	BPX Chain
USD Coin (**USDC**) *0x3c499c542cEF5E3811e1192ce70d8cC03d5c3359*	Polygon	🌉 USD Coin (**USDC**) *0x0b2ffE5CC332B72293Ca87d48ae9400bFd9Ba97e*	BPX Chain
USD Coin (**USDC**) *0xB97EF9Ef8734C71904D8002F8b6Bc66Dd9c48a6E*	Avalanche C-Chain	🌉 USD Coin (**USDC**) *0x0b2ffE5CC332B72293Ca87d48ae9400bFd9Ba97e*	BPX Chain

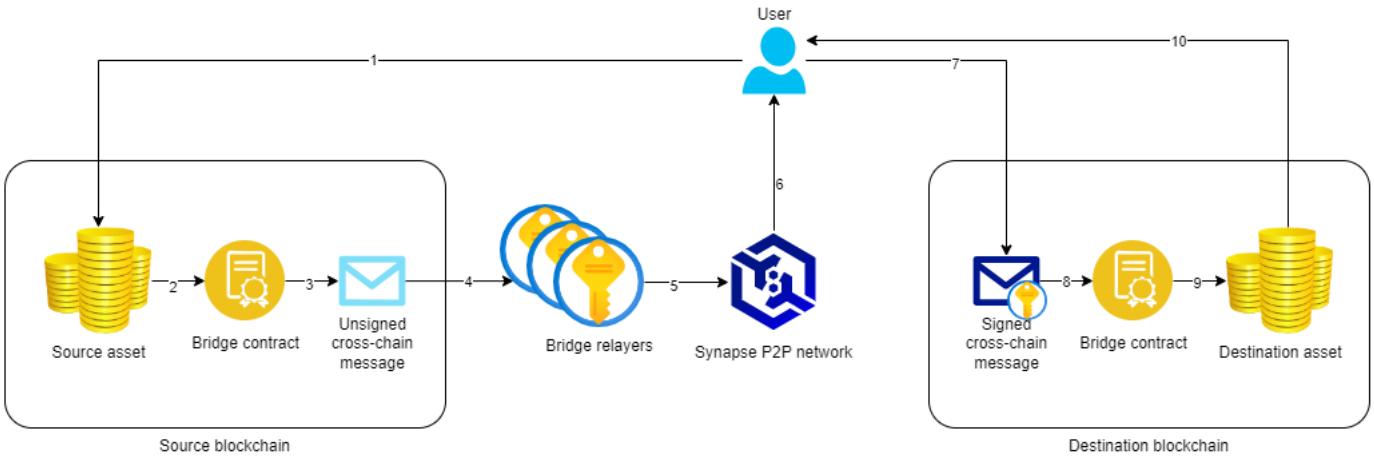
We have established a convention that all tokens wrapped by the bridge on the BPX chain have the "bridge at night" emoji (🌉) at the beginning of their original name and original token symbol.

However, wrapped BPX on externals chains is always named "BPX Chain" and has the original symbol "BPX".

We do not add any prefixes, like WBPX, WETH, unless the original asset already have one.

Transaction flow

Each bridge transaction proceeds according to the following scheme:



1. The user initiates a source asset transfer to the bridge contract running on the source blockchain. If the source asset is a native blockchain token, the user transacts to the bridge contract's `transfer()` method. If the source asset is an ERC-20 token, the user transacts to the token's `approve()` method, then transacts to the bridge contract's `transferERC20()` method.
2. If the user sent the original token to get wrapped one, the token is locked in the bridge contract. If the user sent the wrapped token to recover the original asset, the wrapped token is burned.
3. The bridge contract generates a cross-chain message - a bytes string that contains all transaction details. The `MessageCreated()` event is emitted.
4. Bridge relayers constantly listen for bridge contract events. After receiving information about a new message, each relayer performs a checks according to a deterministic algorithm whether it should sign this message or not. Only 8 relayers out of all will be delegated to sign a certain message.
5. Selected relayers sign the message with their private key and send it to the user using the Synapse network.
6. The user waits for all 8 signatures to be received and checks their validity and compliance with the algorithm rules.
7. The user sends a cross-chain message with 8 signatures to the bridge contract running on the destination chain by transacting to the `messageProcess()` method. In the same transaction, the user pays a bridge fee.
8. The bridge contract validates the message, checks whether the signatures are correct and come from exactly those relayers that were selected by the algorithm.
9. If the user unwraps the wrapped tokens, the bridge contract sends the original assets back to the user's wallet. If the user wraps assets, the bridge contract mints the wrapped token to the user's wallet. Additionally the bridge contract divides the received fee into 8

equal amounts and adds them to the balances of relayers involved in a given transaction processing.

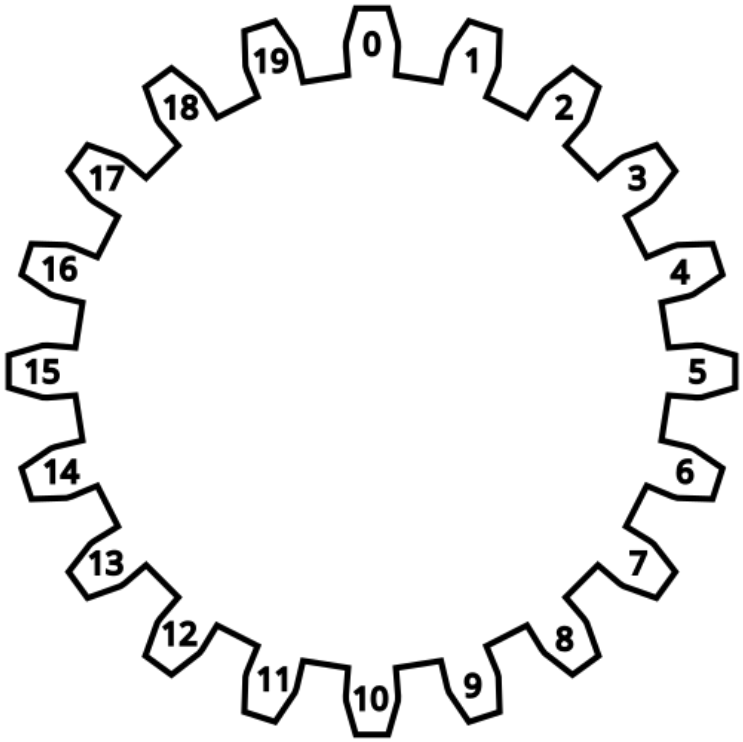
10. The user receives the destination assets on the destination blockchain.

Relayer selection algorithm

BPX Bridge associates each transaction and signature to a so-called epoch - 20-minute period of time. When the epoch changes, for the same transaction, the set of selected validators will be completely different. So if a transaction cannot be completed in a given epoch due to a relayer failure, the algorithm will select a different set of relayers within 20 minutes at the latest and the transaction will not be stuck forever.

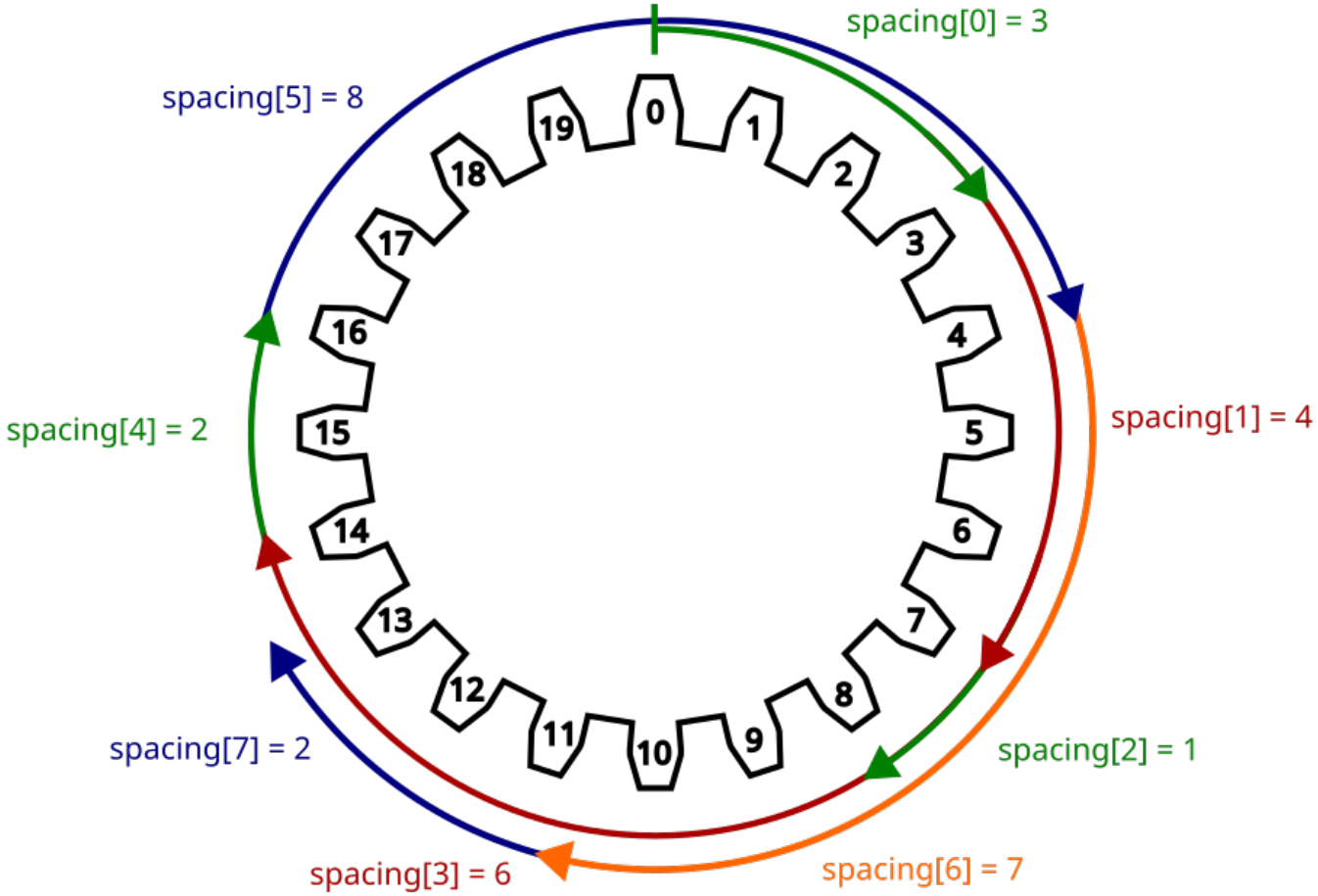
The algorithm for delegating 8 relayers for a transaction going through the bridge works as follows:

1. The bridge smart contract keeps a list of all wallet addresses that have registered as a relayer. Let’s visualize this list as a circle, because the algorithm is based on infinite shifting of the relayers list. For example, if there are 20 relayers in total, we will be adding a number 10, or 20, or 100, etc. to the index 15 to jump over the end of the list and get the relayer with index of 5 again, instead of getting non-existing relayer 25.

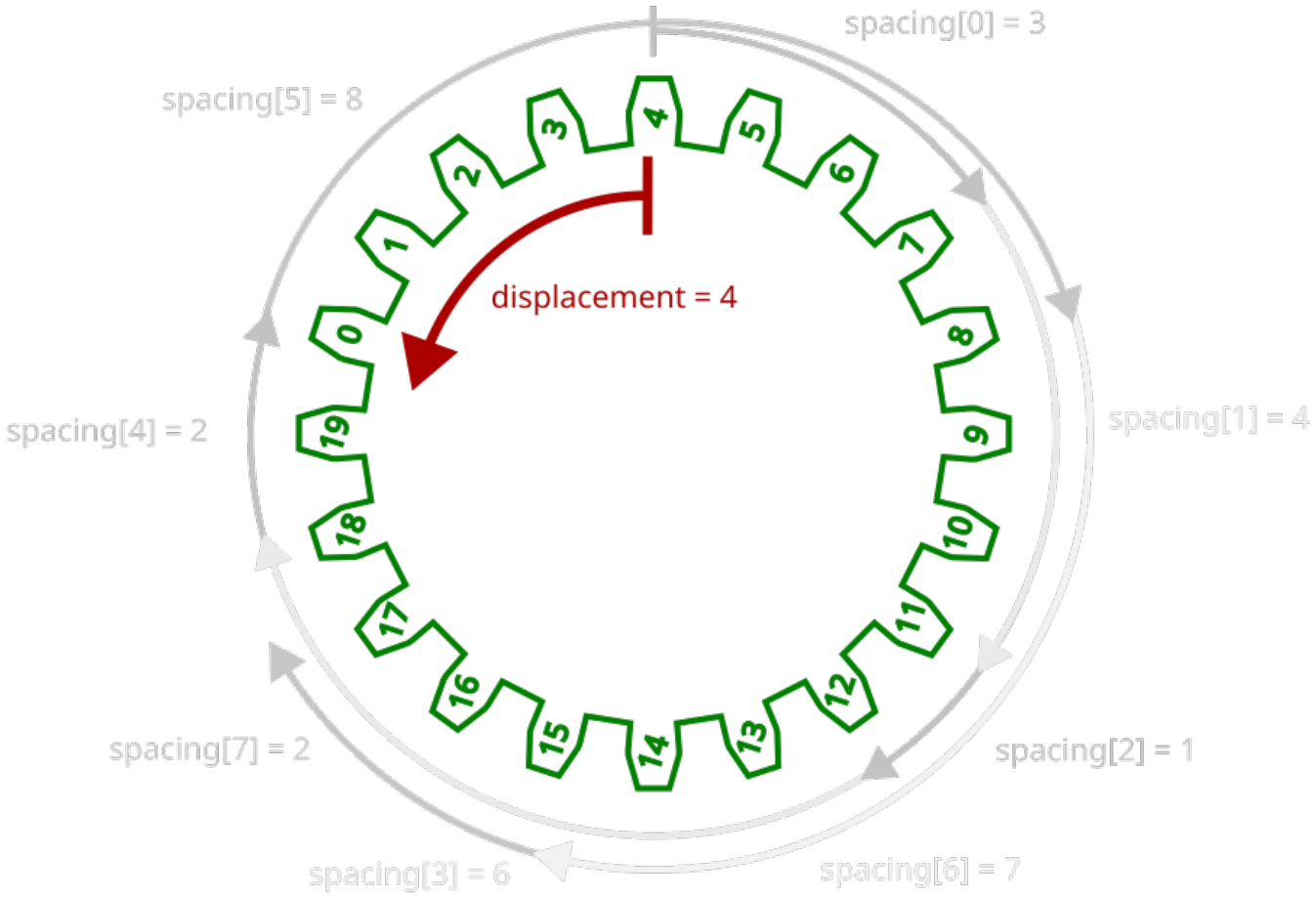


2. The smart contract picks up 8 individual spacings between relayers. These values will remain constant throughout the entire bridge epoch. To calculate these spacings, we are using transaction-independent pseudorandom data – hashes of previous blocks in the chain. Therefore

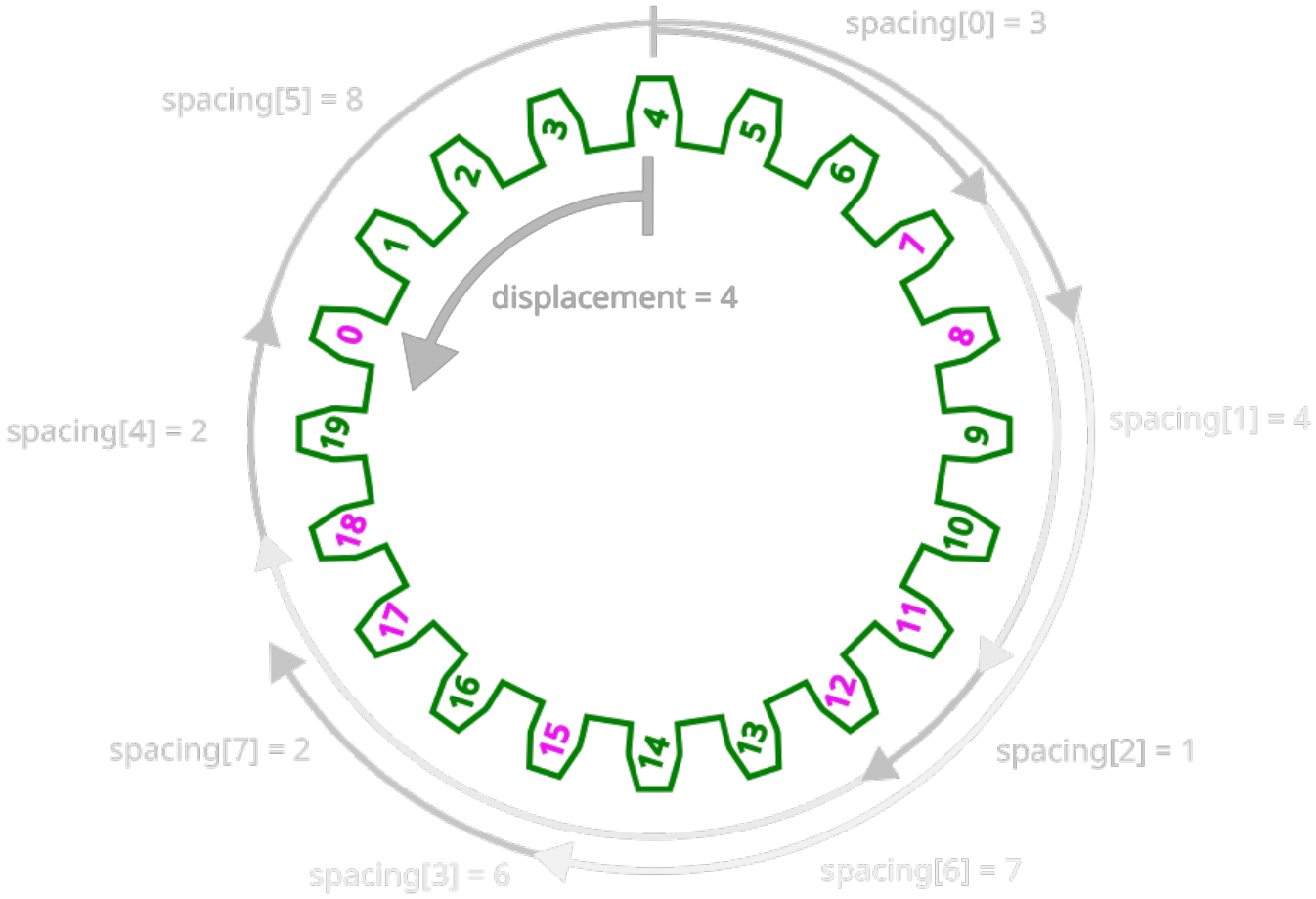
the bridge user has no influence on the calculated values. If we relied on transaction-dependent data, a hostile user could craft the transaction parameters in a such way, that only hostile relayers under his full control were delegated to validate the transaction. In our example, the calculated spacings have the following values: 3, 4, 1, 6, 2, 8, 7, 2.



3. For each transaction, the displacement of relayers list is calculated. We are shifting the entire list by a value based on transaction-dependent data, so each transaction, even in the same epoch, has a different set of relayers. Potential hostile bridge user can influence the calculated displacement by modifying the transaction parameters, but is still unable to modify the spacings from the previous step.



4. After applying the displacement, relayers selected to validate the transaction are as follows: 7, 11, 12, 18, 0, 8, 15, 17. The order is important - the signatures will be invalid if they are swapped.



If the algorithm happens to select the same relayer twice or more, the next available relayer in the list is selected instead.

Smart contracts

All BPX Bridge smart contracts are open source and available in the public GitHub repository:

https://github.com/bpx-chain/bridge-contracts

BridgeHome.sol

This contract is deployed only on the BPX chain. In addition to the standard bridge logic, it contains a map of all supported chains and assets and a set of methods that resolve the local token contract address to the remote one and vice versa.

BridgeForeign.sol

This contract is deployed on all chains except BPX. It contains the standard bridge logic, but it does not contain a map of supported chains and assets. The only allowed opposite chain is BPX and in all generated messages it includes local addresses of the token contracts, which must be resolved by the BridgeHome contract on the opposite side of the bridge.

Proxy.sol

Typical Solidity proxy contract that enables future updates of the BridgeHome and BridgeForeign contracts.

BridgedToken.sol

The smart contract of any token wrapped by the bridge. A classic ERC-20 implementation that does not perform any additional functions apart from minting and burning by the owner. The owner is always the bridge contract.

Bridge contract methods

These are all the public methods exposed by the underlying Bridge contract from which both the BridgeHome and BridgeForeign contracts inherit.

Asset methods

```
function assetResolve(uint256 chainId, address contractLocal) view returns (address)
```

In the BridgeHome contract (running on the BPX chain), it queries the assets database and returns the remote token contract address by provided remote chain ID and local token contract address.

In the BridgeForeign contract (running on other chains), it just returns the `contractLocal` parameter.

Message methods

```
function messageCheckSignatures(uint256 chainId, bytes32 messageHash, tuple(uint8 v, bytes32 r, bytes32 s)[8] signatures, uint64 sigEpoch) view returns (address[8])
```

Checks the signatures of a cross-chain message before actually posting it for execution.

```
function messageGetRelayers(uint256 chainId, bytes32 messageHash, uint64 epoch) view returns (address[8])
```

Returns addresses of 8 relayers delegated to sign a given message in the given epoch.

```
function messageProcess(bytes message, tuple(uint8 v, bytes32 r, bytes32 s)[8] signatures, uint64 sigEpoch) payable
```

Executes a signed cross-chain message and transfers funds to the destination wallet. The function is payable and requires a bridge fee which will go to the relayers as their earnings. The fee amount must meet both conditions:

- The fee must be greater than or equal to the gas price of the current transaction (`tx.gasprice`) multiplied by 21000
- The fee must be divisible by 8

Relayer methods

```
function relayerActivate(uint256 chainId) payable
```

Activates the caller wallet as a bridge relayer. Activation takes effect from the next epoch after the current one. The function is payable and requires a relayer stake deposit. The amount of the required stake can be checked by the `relayerGetStake` method.

```
function relayerDeactivate(uint256 chainId)
```

Deactivates the caller wallet as a bridge relayer. Deactivation takes effect from the next epoch after the current one.

```
function relayerGetBalance(uint256 chainId, address relayerAddr) view returns (uint256)
```

Returns the entire relayer balance (deposited stake + all earnings).

```
function relayerGetStake(address relayerAddr) view returns (uint256)
```

Returns the stake amount required to become a bridge relayer.

```
function relayerGetStatus(uint256 chainId, address relayerAddr) view returns (bool, uint64)
```

Returns whether the relayer is active or not (`bool`) and from which epoch it has its current activation status (`uint64`).

```
function relayerGetWithdrawalMax(uint256 chainId, address relayerAddr) view returns (uint256)
```

Returns the relayer balance available for withdrawal. For an active relayer, these will be be only earnings. For an inactive relayer, it will be a stake + all earnings.

```
function relayerWithdraw(uint256 chainId, address to, uint256 value)
```

Withdraws funds from the caller relayer balance. The amount of funds that can be withdrawn can be checked using the `relayerGetWithdrawalMax` method.

Transfer methods

```
function transfer(uint256 dstChainId, address dstAddress) payable
```

Initiates the transfer of a native token. This function is payable and transaction should have the value to be transferred by the bridge.

```
function transferERC20(address srcContract, uint256 dstChainId, address dstAddress, uint256 value)
```

Initiates an ERC-20 token transfer. The bridge contract must be approved to perform an ERC-20 transfer from caller wallet (ERC-20 `approve()` method).

Becoming a relayer

Anyone can become a BPX Bridge relayer and validate transactions passing through the bridge. For helping maintain the bridge secure, relayers are paid in each blockchain native currencies.

To become a relayer, the following conditions must be met:

- You must have a fast and stable Internet connection, uninterrupted power supply and a server running 24/7 with Windows, Linux or Mac OS. VPS in the dedicated hosting company is also allowed.
- You must have the RPC access to full nodes of BPX Chain and other blockchains for which you want to relay transactions. Using the public RPC endpoints is strongly discouraged, local nodes are preferred.
- You must activate your relayer wallet in bridge smart contracts and thus deposit a relayer stake for each relay direction. For example, if you only want to relay transactions coming from the BPX network to Arbitrum, you must only deposit on the Arbitrum network. If you want to relay transactions going both ways, you need to deposit on both networks. If you want to validate transactions between the BPX, Arbitrum and Polygon in all allowed directions, you must deposit on the Arbitrum network, deposit on the Polygon network, and deposit on the BPX network **twice**. Of course, after unregistering the relayer, you could immediately withdraw all deposited funds, and for the whole time deposits are locked by a smart contracts. BPX developers are not their custodians.
- You must run the open source relayer software for each relay direction and ensure it is always running without any issues.

Current relayer stake amounts in all supported chains:

Chain	**Relayer stake**
BPX Chain	3,000,000 BPX
Arbitrum	0.25 ETH
Polygon	1500 MATIC
Avalanche C-Chain	25 AVAX

> Relayer stake is a crucial mechanism for ensuring bridge security. Without this deposit, one person could register millions of relayer wallets and thus gain a huge probability of being able to sign transactions themselves, without other honest relayers. Additionally, the deposit enforces relayer to unregister from the smart contract to end their relaying activity which protects the bridge against stucked transactions because of lack of signature from "abandoned" relayer.

Relayer software

The official JavaScript (node.js) implementation of the relayer software is available here:

https://github.com/bpx-chain/bridge-relayer

The software works on all popular operating systems. The only requirements are Node.js and npm installed.

Installation

```
git clone https://github.com/bpx-chain/bridge-relayer
cd bridge-relayer
npm install
```

Usage

```
$ node bbrelay.js -h
Usage: bbrelay [options]

BPX Bridge relayer

Options:
  -s, --src-rpc <url>     Source chain RPC URL
  -d, --dst-rpc <url>     Destination chain RPC URL
  -k, --wallet-key <key>  Relayer wallet private key
  -h, --help              display help for command
```