AMBROSUS

The Ambrosus Sidechannel +PoC

ambrosus.com
info@ambrosus.com

Sidechannel PoC

Contents

Sidechannel PoC 2
Glossary 2
General description 3
PoC functionality 4
PoC usage scenario 5
Architecture description 6
Appendix A: automated sidechannel creation 9
Appendix B: converting mainnet currency to sidechain currency using atomic swapping 10

Glossary

Mainnet (main network) - the main Ambrosus network.

Sidechannel (sidechain) - separate blockchain network that runs in parallel with the main network. It may not be based on the same network as the mainnet, but it must be strictly tied to the main network.

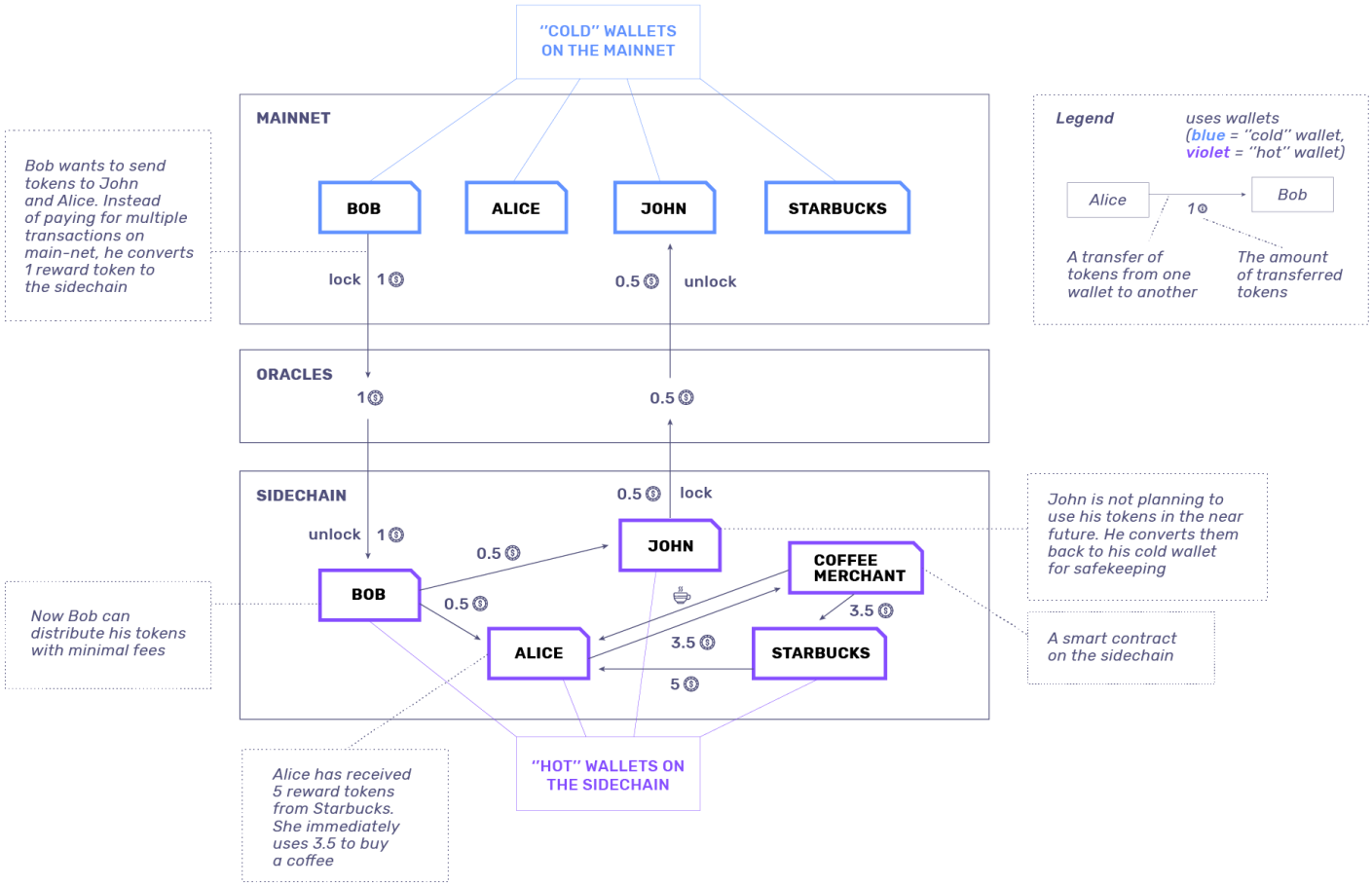
Sidechannel token - the native currency of the sidechannel (similar to ether in Ethereum). Mainnet token - the native Amber currency of the Ambrosus network.

Mainnet asset (custom asset) - custom assets based on smart contracts in the main network (e.g. ERC-20 tokens).

General description

This proof of concept demonstrates potential usage of sidechains for implementing custom assets based on the Ambrosus network. The goal is to provide a platform for companies to create personalized crypto-assets which can be used by their customers without requirements to hold and manage other tokens (e.g. the Ambrosus token). The main problem with achieving this goal is the limitation of the main network which requires all transaction fees to be paid in the native currency (the Ambrosus token). So there is no way to use an asset on the main network without holding at least some Ambrosus tokens. The proof of concept attempts to eliminate transaction fees using sidechannels.

A sidechannel (or a sidechain) is a separate blockchain network that runs in parallel with the main network. It can be mostly a private network, however the sidechannel must be tied to the main network with some form of interoperability between them. The side network can be configured differently to the main network or it may be a different network altogether. This allows for greater flexibility when it comes to transaction fees. Users in the side network can pay fees in that network's native currency. So the proposal is to create assets on the main network and to have the ability to convert them into the native sidechannel currency where transactions can be performed while paying fees using, essentially, the same custom asset. After performing all the necessary transactions in the sidechannel, users can convert their assets back into mainnet assets.



PoC functionality

The proof of concept implements a secure communication bridge between the main network and the side network and uses it to peg the native sidechain currency to a ERC20-based token on the main network. Issuing an asset starts with deployment of a sidechain. It is possible to integrate the deployment functionality into mainnet Parity clients. A possible implementation is described in more detail in **Appendix A**. However, for now, the simple proof of concept uses a private Parity network that was deployed manually as a sidechain. Technically, the PoC doesn't rely on the origin of the network. It will work exactly the same with any pair of Ethereum networks (e.g. Amborsus testnet and a private sidechannel).

During the network creation, a set amount of sidechannel tokens will be issued for a specified account.

Once the side network is set-up, the owner of the network has to deploy a bridge between the "main" network and the "side" network. To deploy the bridge, the owner has to provide an URL to a node on from the sidechannel (the URL for mainnet is hard-coded because it should be the same for all sidechains) and an account that holds both mainnet and sidechain tokens.

This can be done from the "Dashboard" part of the PoC, which represents a common dashboard for all companies who use the Ambrosus network (the Hermes dashboard). The account and its private key is provided through the configuration, and the URL can be entered through the dashboard UI.

In order for the bridge to function, a trusted third party with access to both networks is required. There is no other way to convert tokens for a single user (note: there is a way for two users to exchange tokens without third party involvement which is described in **Appendix B**). Therefore, a trusted “Oracle” is necessary. Fortunately, the Oracle is extremely simple. Its task is to listen for events on one side of the bridge and relay them to the other side. Ideally, this functionality would be integrated into the sidechannel validator. For now, however, the Oracle is a separate entity.

To start an oracle, you only need a funded account in the sidechain and connections to both the sidechain and the mainnet. All of these parameters are provided in the Oracle configuration. Another requirement is that the Oracle account must be marked as trusted by the owner of the bridge.

In order to successfully relay a transaction more than half of all oracles must confirm it. In the PoC there is only one Oracle that is added by default which is enough for the bridge to function.

Once the bridge is deployed and the oracles are ready, the asset smart contracts can be created. In general, any two assets can be pinned between networks. In the PoC, only one asset is created - an ERC20 token on the main network. This asset is pinned to the native sidechain token. The smart contract can be deployed from the Hermes dashboard. In order to deploy it, the Hermes owner has to enter the details about the token (Name, Ticker, Total supply, etc.) and provide the same account that was used during bridge deployment.

After the asset smart contracts are deployed and configured - their addresses will be provided in the dashboard UI. The owner of the Hermes node can use these addresses to create their branded wallet. An example of such branded wallet is provided in the “Wallet” PoC. It has very limited functionality, just for demonstration purposes. The PoC Wallet allows users to convert tokens between main and side networks and to transfer tokens to other accounts in the side network. However, additional features can be easily added as necessary into a production wallet (e.g. exchanging tokens for goods).

PoC usage scenario

With all of the functionality of the PoC in mind, we can describe a more realistic scenario:

Starbucks - is a company which is using the Ambrosus network. Starbucks wants to create a loyalty program where loyalty points will be rewarded for each purchase. Using the Ambrosus platform, Starbucks can create their own sidechannel. Then, from their Hermes dashboard Starbucks can:

1. Create a bridge between the mainnnet and the sidechannel.
2. Launch an oracle to transmit messages using the bridge.
3. Deploy an asset smart contract for the loyalty points.

Starbucks pays GAS fees in Amber tokens for deploying the bridge and the asset smart contracts on the mainnet.

At this point, Starbucks has a supply of loyalty tokens which it can distribute freely. Starbucks can send some tokens to customers who can use the tokens right away in the sidechain. The PoC allows users to transfer tokens to a different account. However, we can easily imagine that transfering to a specific account may mean exchanging goods (e.g. transferring to a cashier account could mean that you have paid for your coffee using the loyalty tokens). These sidechain transfers accept GAS fees in the same loyalty tokens, so the end users don’t need to worry about holding multiple currencies. So the usage case of these tokens can be the following:

1. Starbucks wants to send 10 loyalty tokens to Bob.
2. Starbucks locks 10 tokens on the mainnet, the tokens get unlocked on the sidechannel (Starbucks pays a GAS fee in Amber).

On the sidechannel:

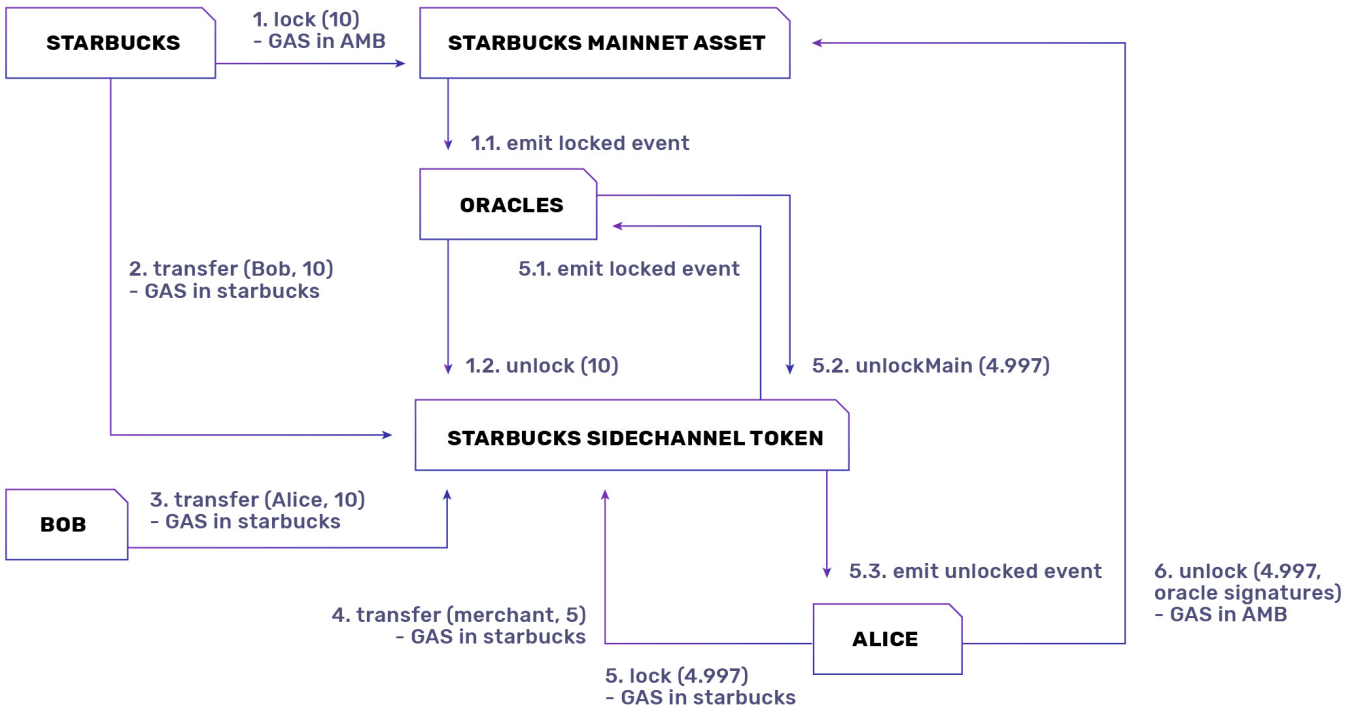
1. Starbucks can send the 10 unlocked tokens to Bob’s address.
2. Bob receives these tokens in his wallet. However, Bob doesn’t really like coffee. Bob wants to transfer these tokens to Alice.
3. Using the wallet, Bob sends his tokens to Alice.
4. Alice receives ~9.998 tokens (minus the GAS fees which were paid in the same tokens).
5. Alice wants to buy a coffee for 5 tokens, so she sends the 5 tokens (plus some GAS fees) to the merchant to buy the coffee (in the same way Bob sent his tokens to Alice).

6. At this point Alice desides to keep the rest of her tokens (4.997) so she transfers them out to the mainnet where they will be safely stored on her account.
7. Alice locks her tokens on the sidechannel.
8. Oracles detect that Alice has locked her tokens and confirm her transaction with their signatures

On the mainnet:

1. Alice unlocks her tokens with Oracle signatures (done automatically by the wallet). Now Alice has 4.997 of the mainnet asset and 0 on the sidechannel balance.

The same process is described in the following diagram (note, "-GAS in ..." denotes that the transaction involves paying a GAS fee in the specified currency).



Architecture description

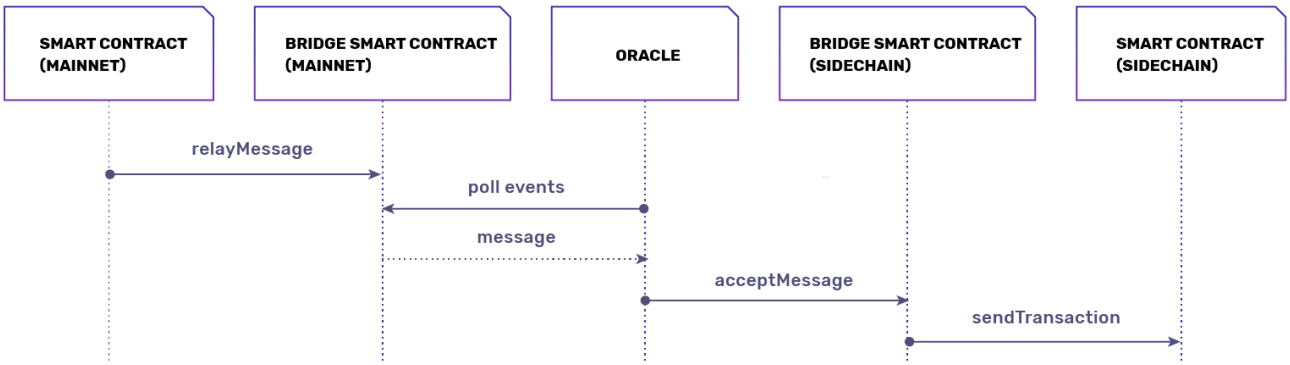
The PoC implements 2-way pegging of assets with the sidechannel - to use an asset in the side-channel, it must be first locked on the mainnet. Similarly, to use assets back on the mainnet, the same assets from the sidechannel must be locked first.

There are two parts of the implementation:

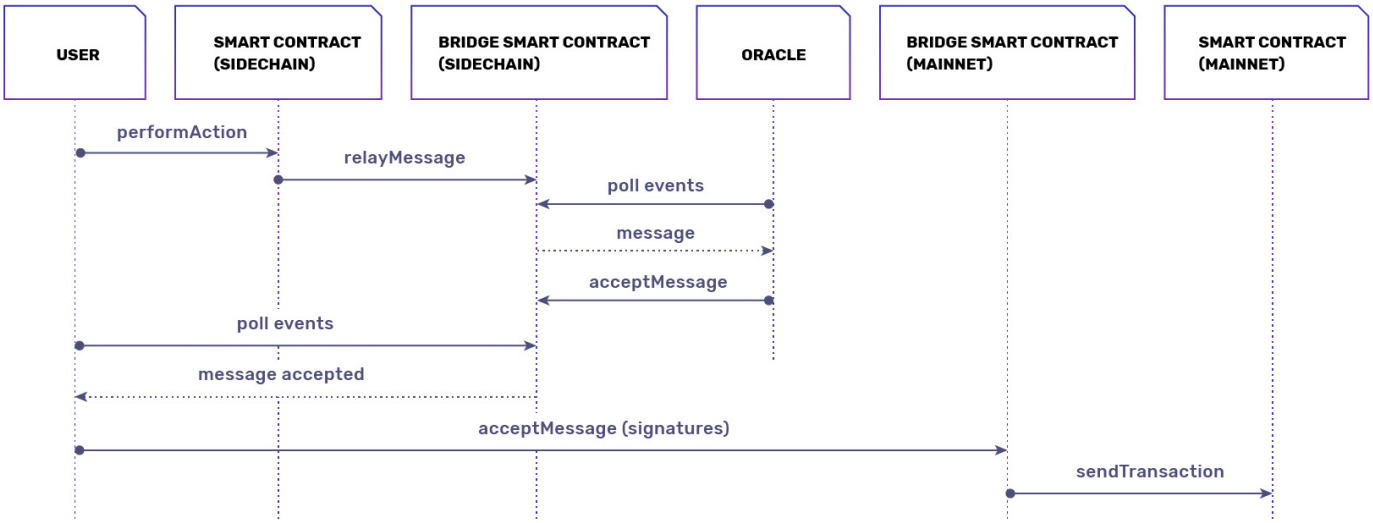
1. The general-use bridge implementation
2. Implementation of tokens that use the bridge

The bridge implementaiton is pretty simple. When a message that needs to be transmitted arrives from the main network, it is emmited as an event. These events are picked up by Oracles and the transaction with the data from the event is commited to the sidechain. The process is slightly different when transferring messages from the sidechannel to the main network. Instead of sending data directly to the mainnet bridge contract, Oracles sign the data and commit it to the sidechannel. Once enough signatures are collected - an event is emmited which contains the data for the user to send to the main network. This way, malicious users cannot abuse conversion to drain funds from Oracles.

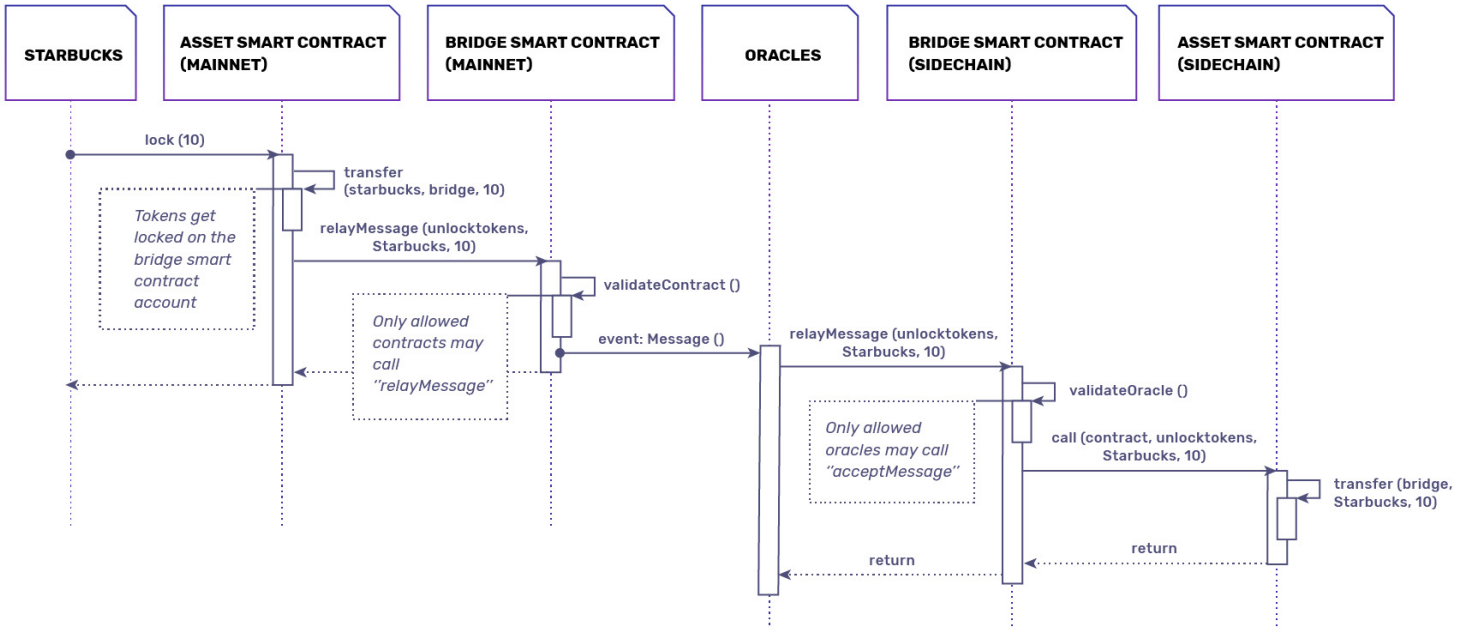
Interaction with bridges from main to side:



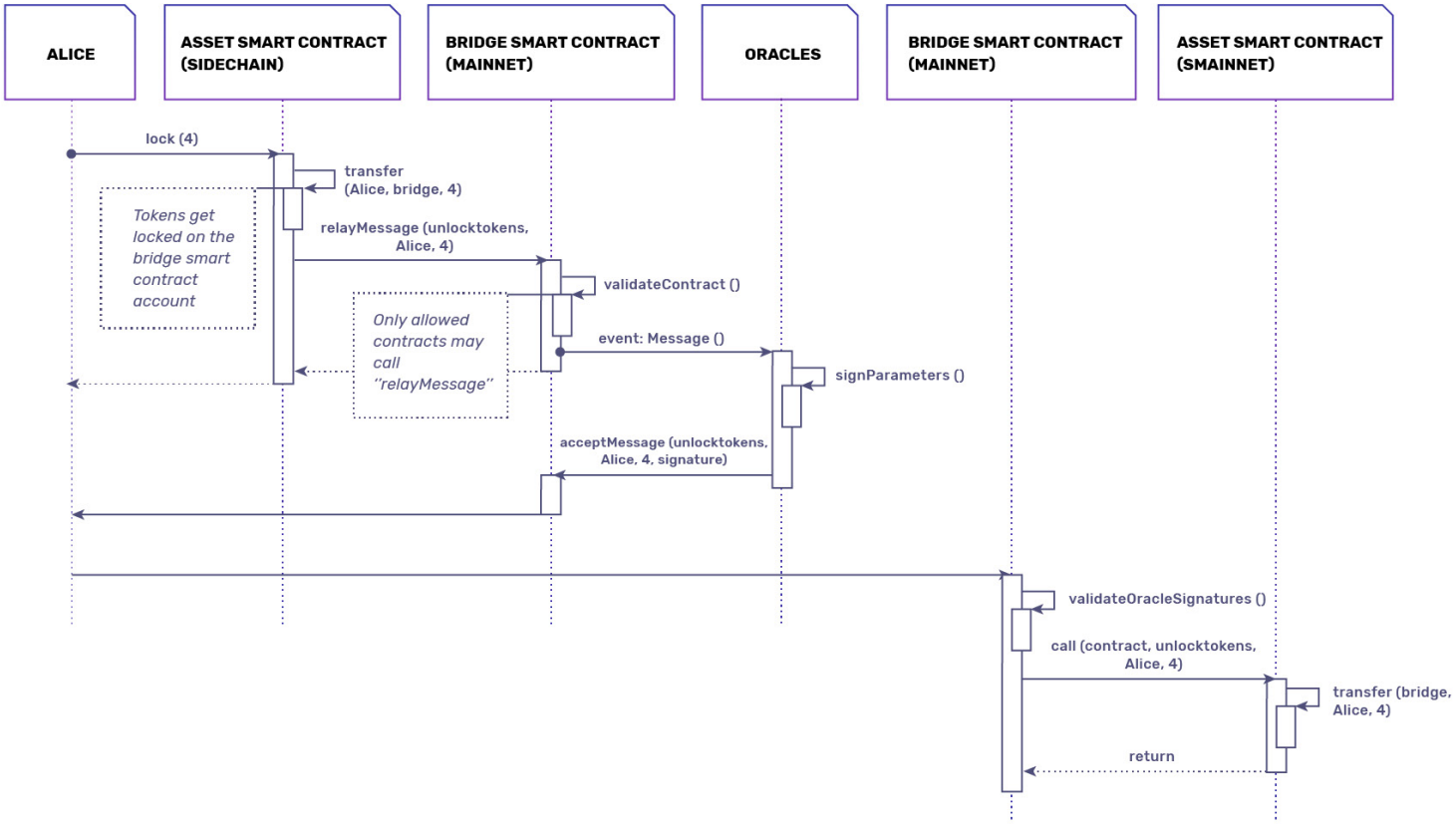
Interaction with bridges from side to main:



These bridges can be used by any smart contract, for example the PoC contains an ERC20 token that can use the bridge to transfer tokens to the sidechain:



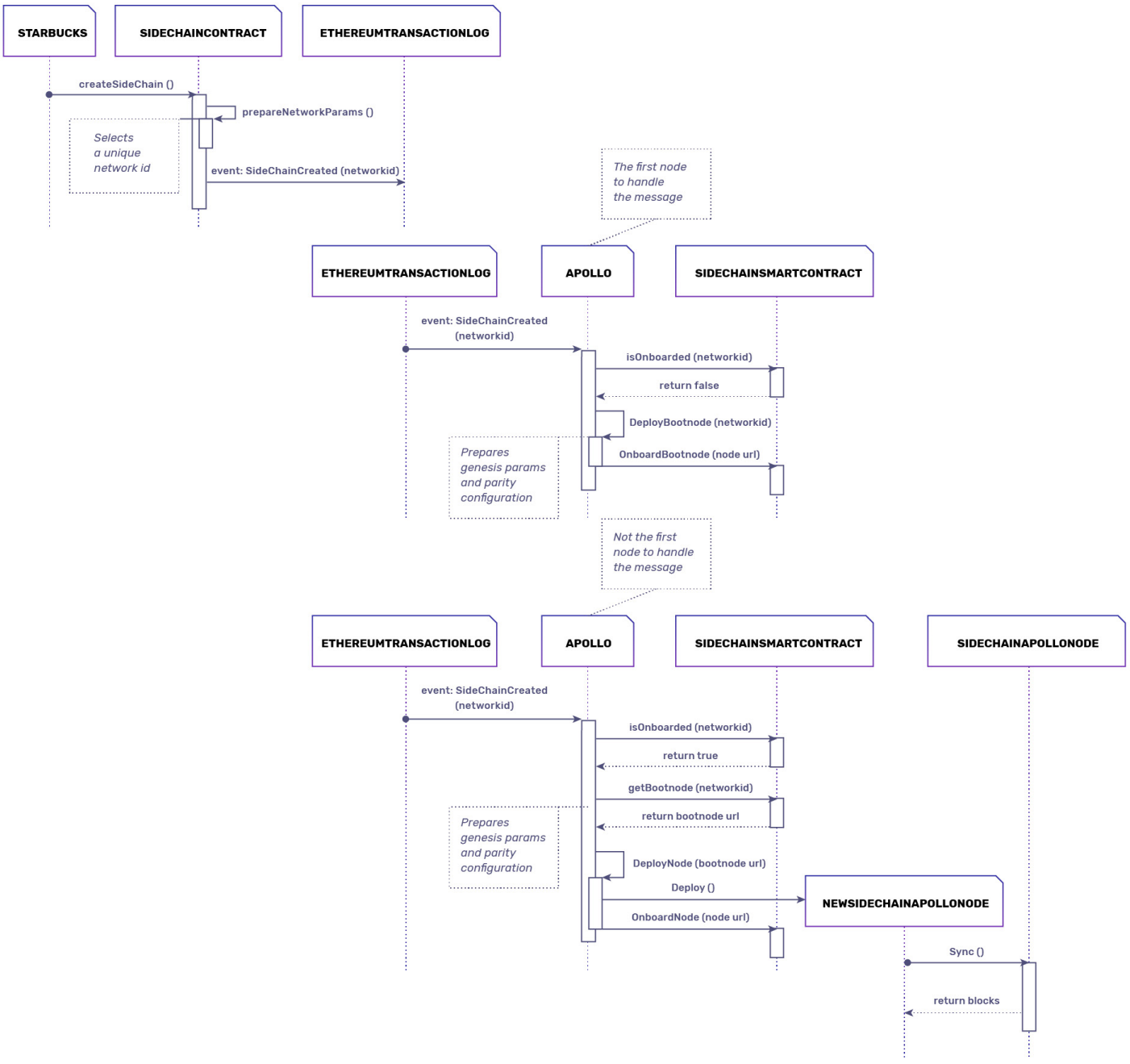
And back from the sidechain:



Appendix A: automated sidechannel creation

This way Apolo nodes on the main net will be responsible for sidechannel deployment. The process should work like this:

1. A company calls the smart contract to create a side-chain and provides a fee for creation.
2. The smart contract records details about the side-chain (network id, number of tokens issued, etc.).
3. The smart contract emits an event that describes the new side-chain.
4. The smart contract event is received by a node.
5. The node subscribes to the side-chain and becomes responsible for handling trans actions on that side-chain.



Sidechannels can be paid for by the creator in periods. The payment can be calculated in the following way:

1. The initial payment = expected GAS usage over 1 period (e.g. a year) * GAS pirce * amount of periods
2. As the sidechain is used, actual GAS usage will become known. Validators of the sidechain will receive rewards based on the actual GAS usage in the sidechain.
3. After the pre-paid period there are three possibilities:
1. The owner of the sidechannel doesnt extend the channel - the channel is deleted.
2. The owner extends the channel and GAS usage was lower than expected - the payment for the next periods = leftover GAS from last period - (expected GAS usage over 1 period (e.g. a year) * GAS pirce * amount of periods).
3. The owner extends the channel and GAS usage was higher than expected - the payment for the next periods = unpaid GAS from last period + (expected GAS usage over 1 period (e.g. a year) * GAS pirce * amount of periods).

Appendix B: converting mainnet currency to sidechain currency using atomic swapping

If two users can arrange a transaction, they can use two-phase locking (or atomic swaps) to safely exchange tokens without using oracles. The idea is that both users first lock their tokens and use a shared secret to unlock tokens on the other network. In case the other user did not lock his tokens, the first user can safely unlock his tokens without losses. While it can be done without involvement of third parties, the main drawback of this approach is that it requires two users who are interested in exchanging the same amount of currency.