Wallet Use Case

By the end of this guide you should know how a cryptocurrency wallet can integrate with the OMG Network.

1. Introduction

1.1 Overview

The main goal of this document is to demonstrate how a cryptocurrency wallet can add OMG Network support for depositing, withdrawing, and transferring ETH and any ERC20 token.

The integration with the OMG Network can significantly help the entire Ethereum ecosystem to offload a certain volume from the root chain, as well as introduce faster and cheaper transactions with the same security guarantees as the Ethereum network.

1.2 Requirements

User Personas

1. Customers - participants and active users of a given cryptocurrency wallet.
2. Integrator - development and management staff of a wallet that evaluates the requirements and delivers the integration with the OMG Network.
3. OMG Network - a technical and management team within the OMG Network that supports the Integrator during the integration process.

User Stories

1. As a customer, I can deposit ERC20 tokens from the Ethereum network onto the OMG Network without leaving my cryptocurrency wallet.
2. As a customer, I can send ERC20 tokens that have been deposited to my wallet to any OMG Network compatible wallet.
3. As a customer, I can withdraw deposited ERC20 tokens from my wallet back to the Ethereum network.

2. Proposed design

The proposed design is based on the lessons learned during the development of the OMG Network mobile wallet, as well as extensive research, interviews, and customer development. The information provided here should be treated as a general suggestion, instead of clear rules that you absolutely must follow.

2.1 Integration scope

The scope for integrating with the OMG Network by a given cryptocurrency wallet includes several steps. The current scope is very similar to the Exchange Use Case Integration Scope, thus some parts will be used as references to that use case to avoid redundant information.

There are two ways a wallet can integrate with the OMG Network:

1. Natively: separate implementations for Android and iOS using Java/Kotlin or Objective-C/Swift respectively.
2. Hybrid: single implementation based on React Native.

At the moment, we do not provide SDKs for Android and iOS yet, thus the first option can be more complicated because some of the Watcher Info API endpoints may not be available, and we cannot guarantee that external dependencies (web3, crypto, encoding/decoding utils, etc.) will behave the same way as they do with JavaScript libraries.

We do provide React Native support for omg-js so you shouldn't have any issues using this approach. If possible, we suggest using the latter option because it is well tested by our team.

2.1.1 Query blockchain

Querying blockchain is the most basic thing every integrator needs to know how to do. A wallet provider might be interested in retrieving data about transactions, accounts, deposits, and exits. You can find the full list of commands for this step in the Exchange Use Case.

Implementation details:

The information about the token and its price currently have to be retrieved from different sources (Ethereum node, such as Infura, and one of the public price feeds, such as Coinmarketcap, Coingecko, AmberData, or your existing price source), which makes it more difficult to display the full token details and the balance in different fiat currencies.

2.1.2 Make Deposits

It is required to have funds deposited into OMG Network first before making any transfer. You can implement this functionality by following Deposit Funds guide.

Implementation details:

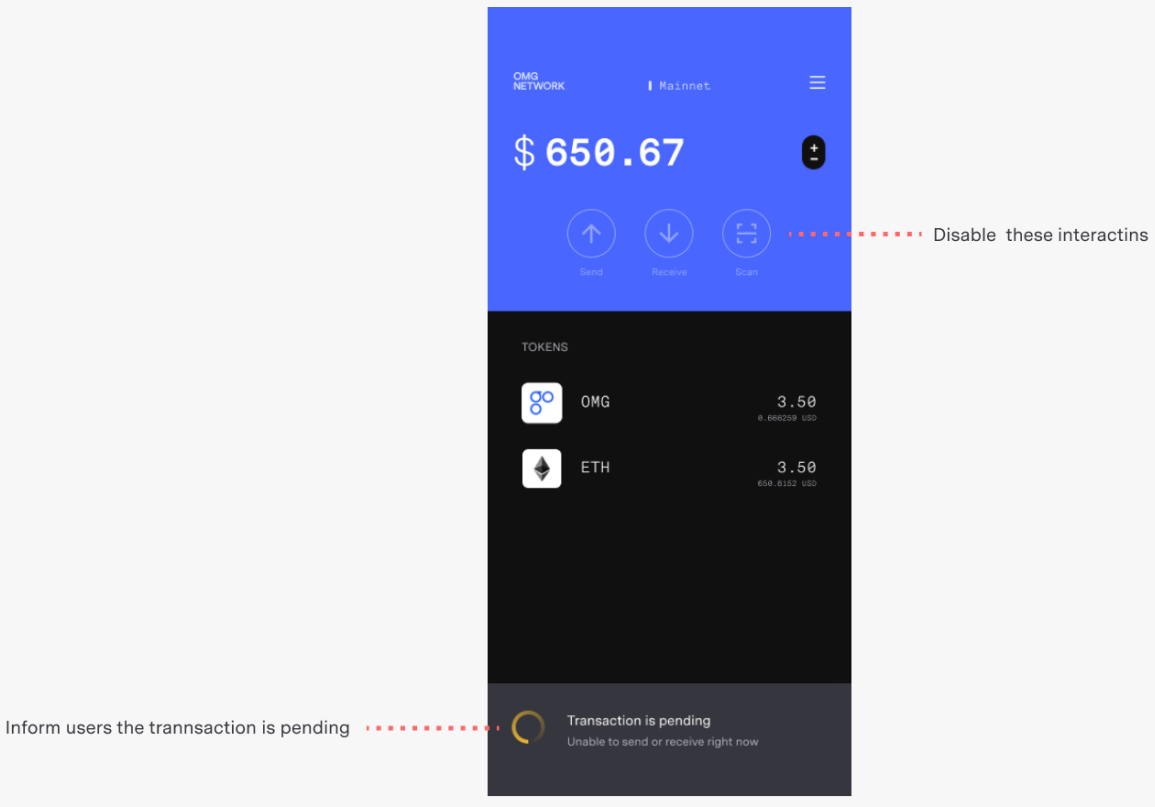
Each deposit has a deposit finality period before it can be used on the network. Currently, this period equals 10 Ethereum blocks. The confirmations are counted on the Ethereum because the OMG Network relies on rootchain (Ethereum) security and creates blocks only when new transactions are formed (i.e. on-demand). You may use the `waitForChildchainBalance` function from omg-js documentation or implement your custom helper/service for polling the expected amount after the deposit.

2.1.3 Make Transfers

Making transfers on the OMG Network has the same behavior as on other networks, blockchains, or payment systems. Currently, the network supports only payment transactions, more advanced types will be available in the future. This operation will occupy the majority of data load, thus should be treated with more focus. You can implement a standard transfer functionality using Make a Transfer guide.

Implementation details:

When you send a transaction, you will often use `account/get_utxos` endpoint to get information about the latest UTXOs. Current implementation returns only confirmed UTXOs that are not used for the pending transaction. One of the ways to mitigate this limitation is to block/disable your wallet UI to prevent sending other transactions until the current transaction is confirmed. Otherwise, you may encounter a `UTXO not found` issue. Alternatively, you can keep track of all of the UTXOs used for your transactions but this will create additional complexity for your application.



2.1.4 Make Withdrawals (Exits)

Using OMG Network as a value transfer layer offers a faster and cheaper alternative if your work requires frequent operations on the Ethereum network. Most of the time you will hold a certain amount of assets on the OMG Network, and periodically withdraw your funds back to Ethereum. This is the most complicated technical and UX flow. You can implement exits functionality using Start a Standard Exit guide.

Implementation details:

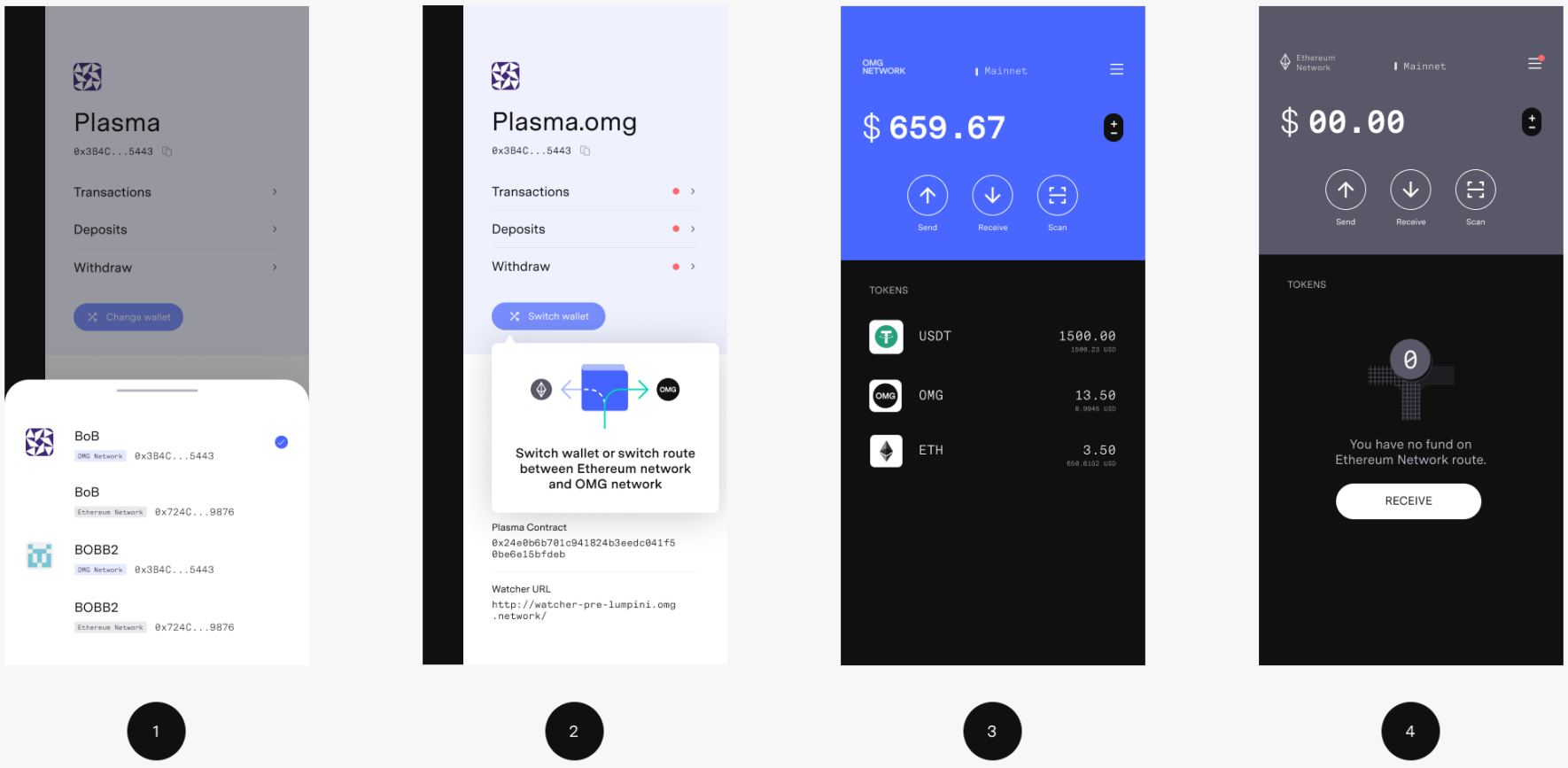
1. The entire exit flow for mobile can be very complicated. It involves dealing with challenges, invalid exits, covering standard and in-flight exits. Thus, if you don't have enough time and resources for implementing such functionality, we recommend not to include it in your wallet at all. Exits are perceived as a less frequent activity on a network and can be accomplished with a Web Wallet provided by the OMG Network team.
2. There's no endpoint to query exit data in the Watcher Info API. It is available only in omg-js library.

2.2 Design Guideline

Layer 2 mobile UI and UX can be challenging and confusing, thus our job is to make the user flows intuitive, appealing, and easy to navigate. Below you can see recommendations from our team.

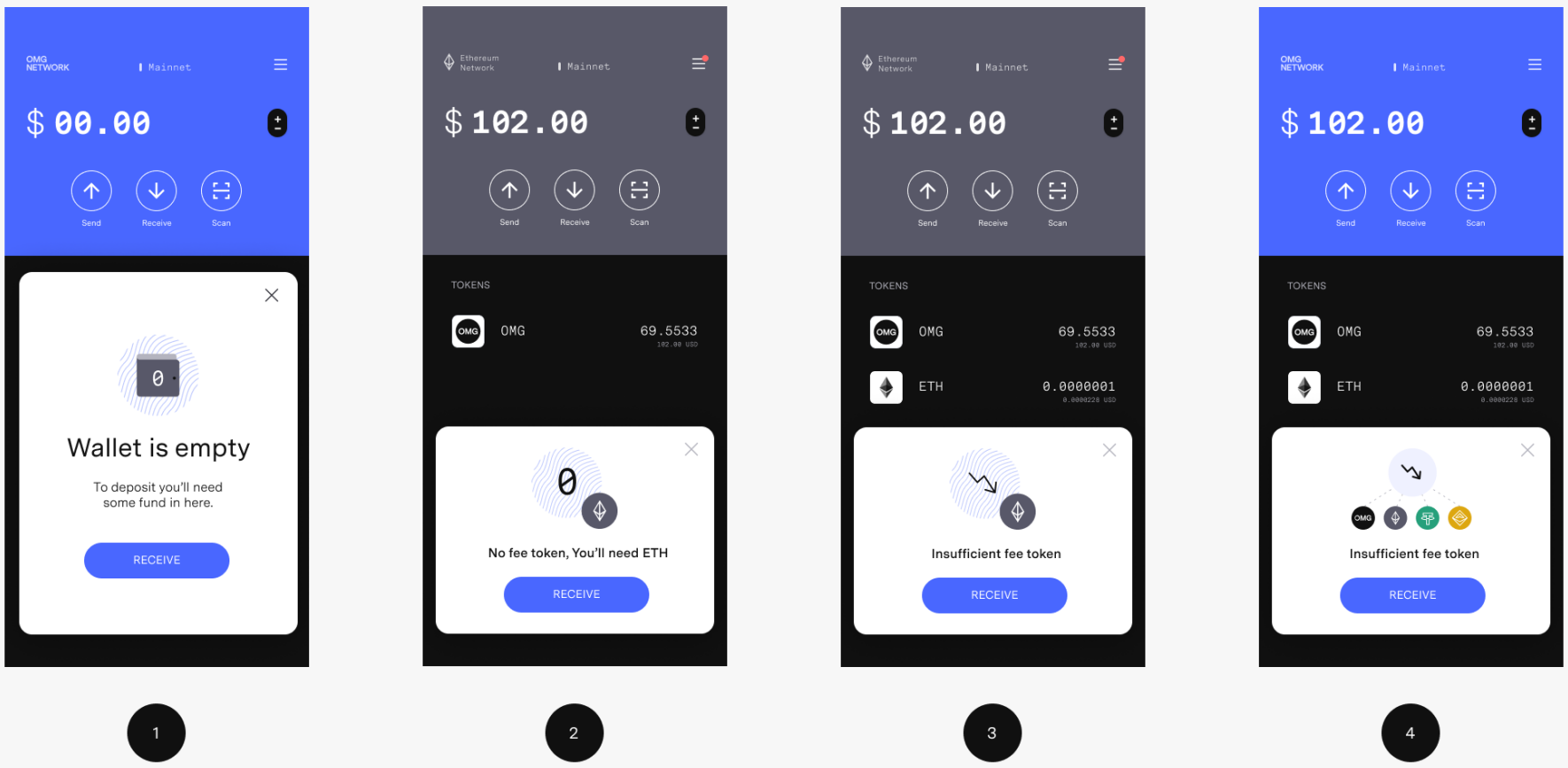
Network Distinction

It's a good practice to have a clear visual distinction between the Ethereum path and the OMG Network path. It's not recommended to display token's balance on both networks together. Setting an obvious separate path helps users to interact with tokens on each network. Additionally, you can set up a switch action (e.g. button) that will help to switch between OMG Network and Ethereum.

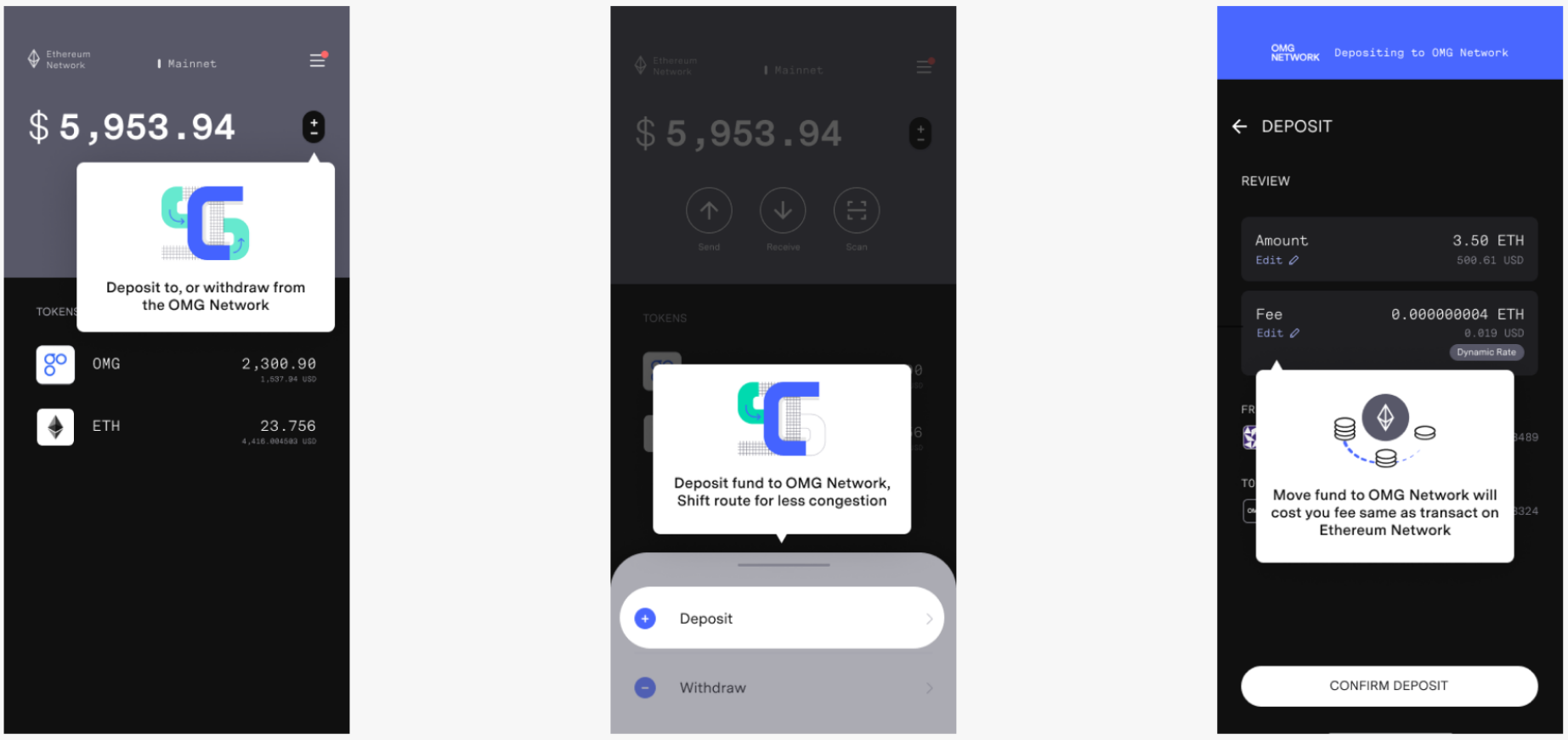


Deposits

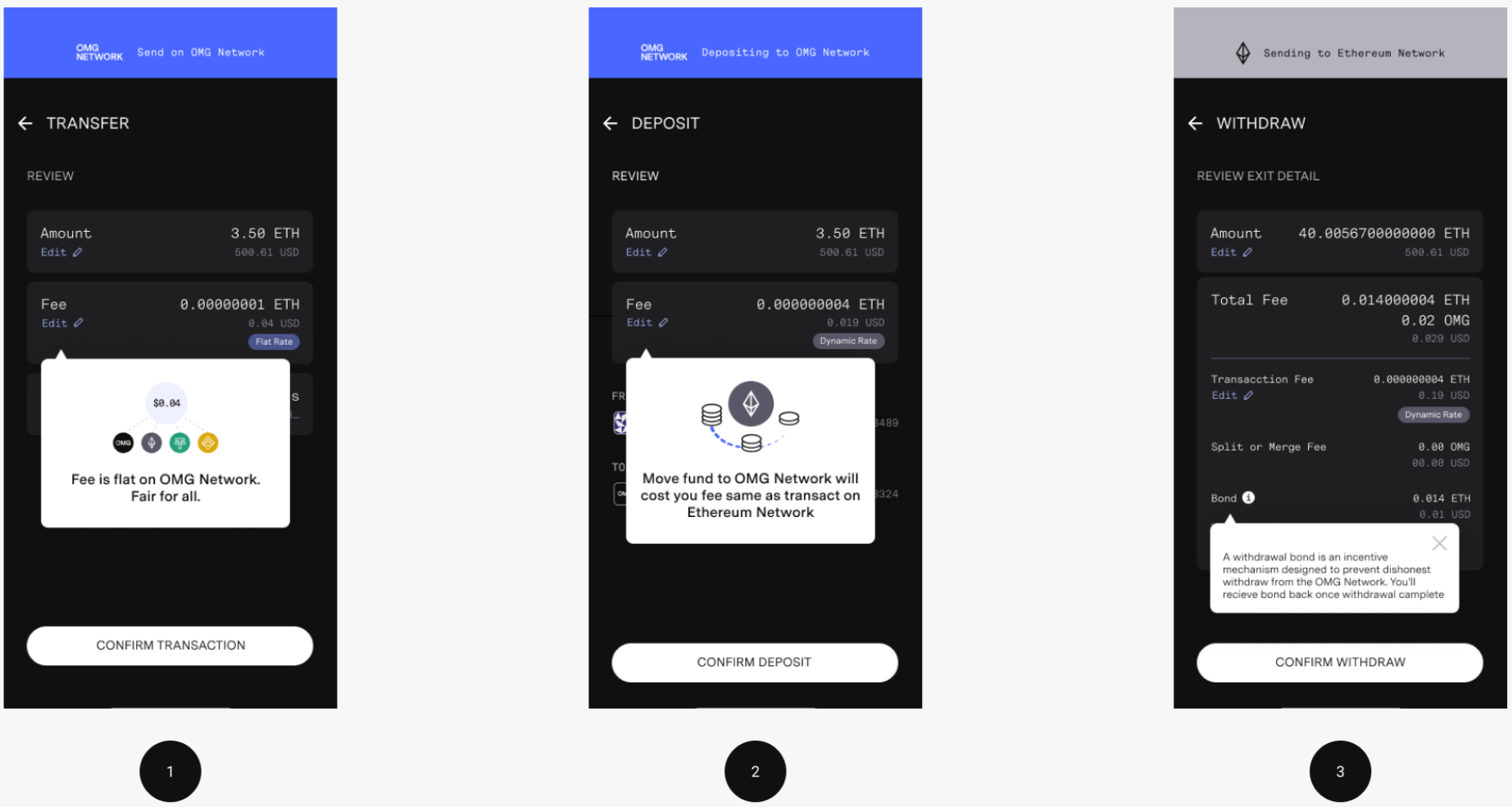
1. A user needs to have some ETH in their cryptocurrency wallet to cover the deposit fee. It's recommended to encourage users to deposit fee tokens first and inform about the fee model upfront. This can be mentioned in the tour/onboarding screens or your main screen as follows:



2. Deposit is not as frequent operation as transfer, thus you may not need to add it to the quick actions panel. However, it has to be easily accessible and people should be aware of this functionality through a dedicated menu, intuitive UX flows, or onboarding.

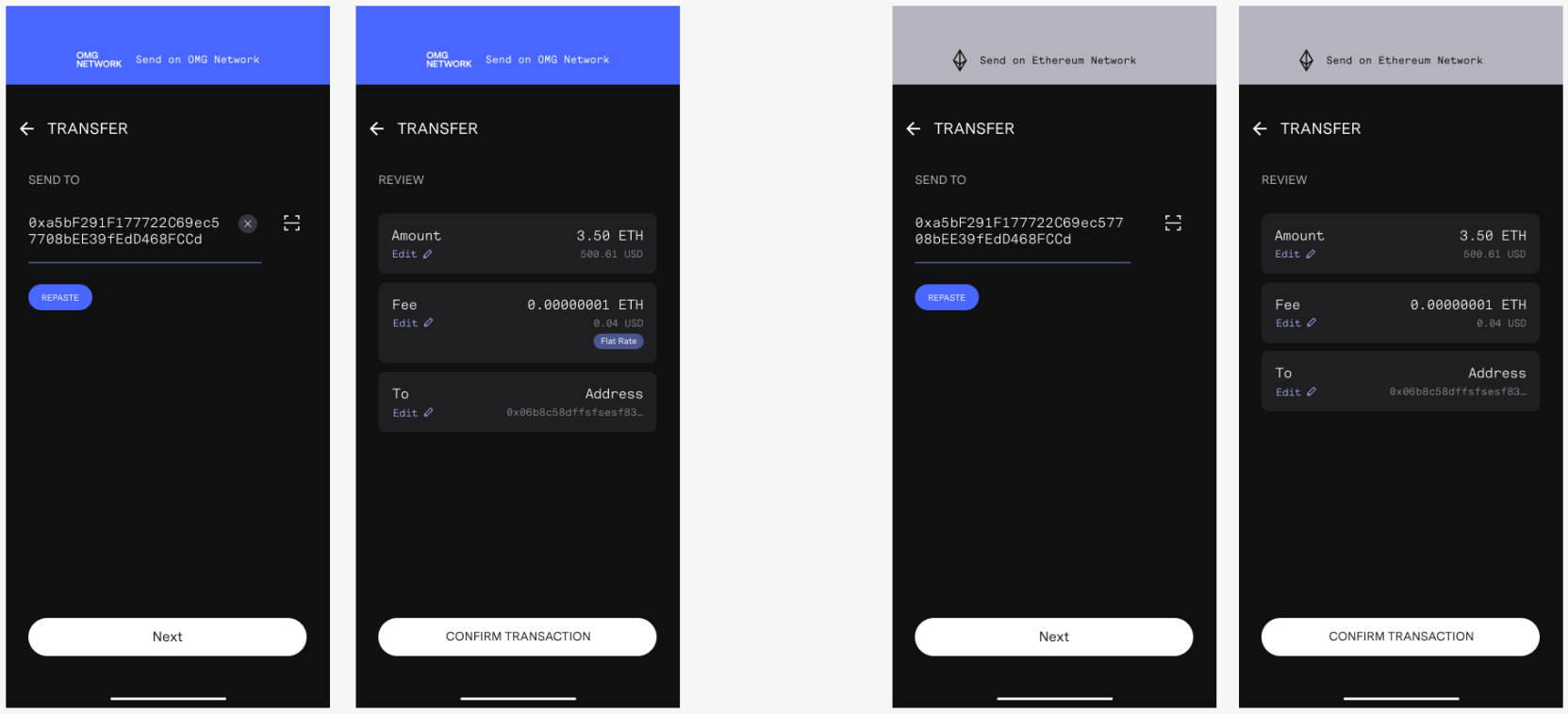


3. The OMG Network has a different [fee amount] for different types of operations, including deposit fee, transfer fee, exit fee, and [exit bond]. Make sure to be explicit about those, as well as mention if the fee has a flat rate or dynamic rate. You might also set a fee with the highest amount as a default option. Additionally, you may mention that the transfer fee is charged in OMG token, the rest of the fees are in ETH.



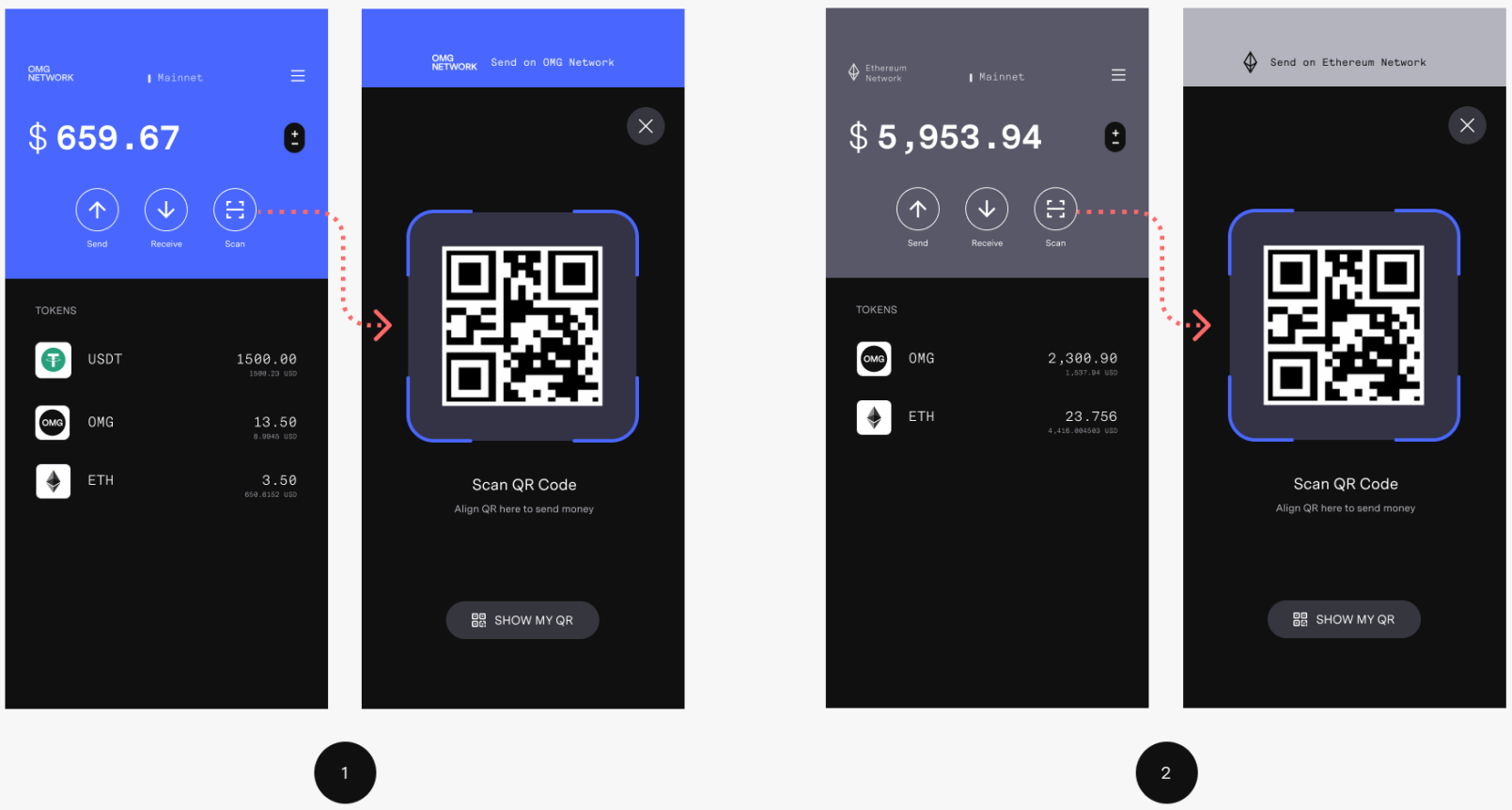
Transfers

1. It might be confusing which network you're using while making a transfer. It's recommended to distinguish between Ethereum and OMG Network with different colors, labels, icons, etc. according to your brand book.



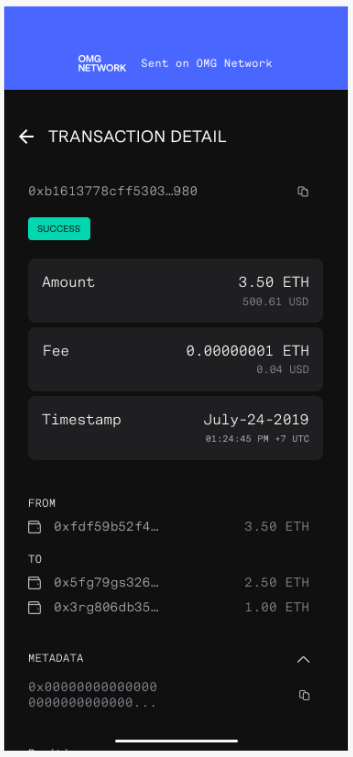
1 2

2. It's a good practice to set a default `Scan QR code` button for the network a user is currently on (Ethereum or OMG). It's not recommended to use a global scan as it creates an extra decision for a user to make. If possible, have a separate flow for each of the networks to create a clear and consistent association and mental model for a particular user.

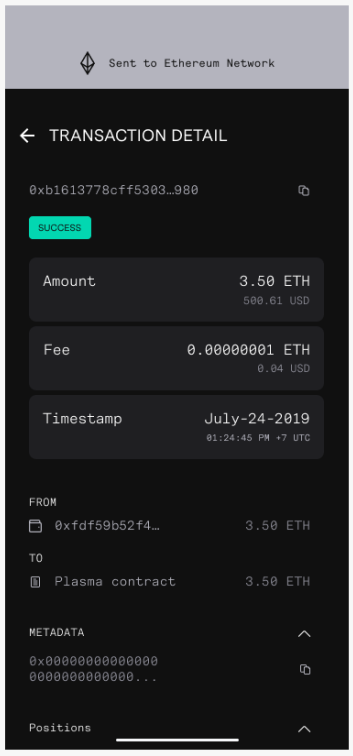


1 2

3. Before making a transaction to an exchange or custodian wallet, ensure that the provider is already on the OMG Network. This can be accomplished by providing a validation message with corresponding color and text message.
4. It is possible to send existing funds on the OMG Network to any non-custodial Ethereum address. Thus, make sure to inform your users of such a possibility. The funds can be easily accessed with any OMG Network compatible wallet afterward.
5. It's a good practice to have a clear distinction in transaction history for both Ethereum and OMG Network.



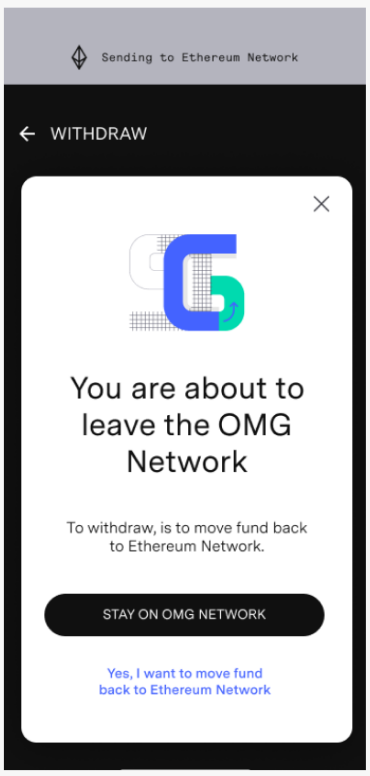
1



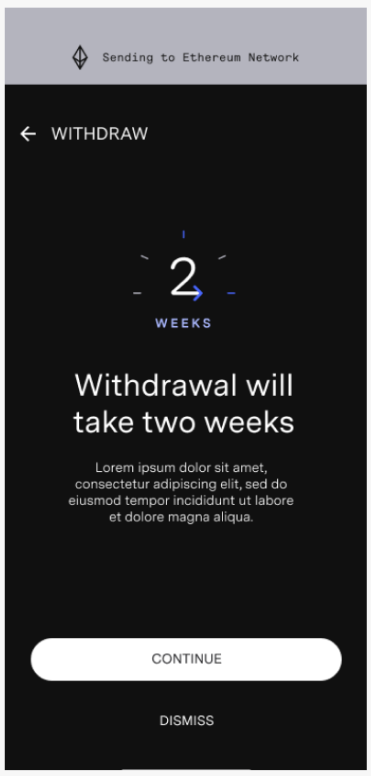
2

Withdrawals

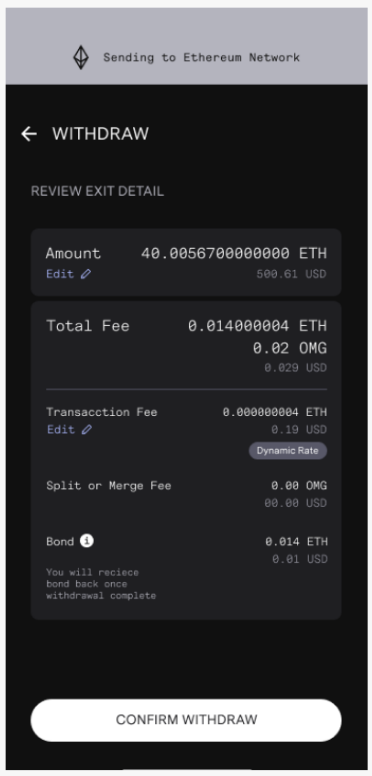
1. Make sure to inform withdrawal condition before allowing a user to proceed with exit submission.



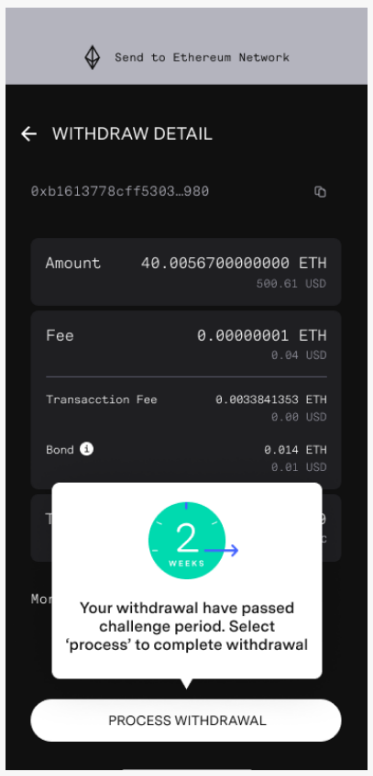
1



2

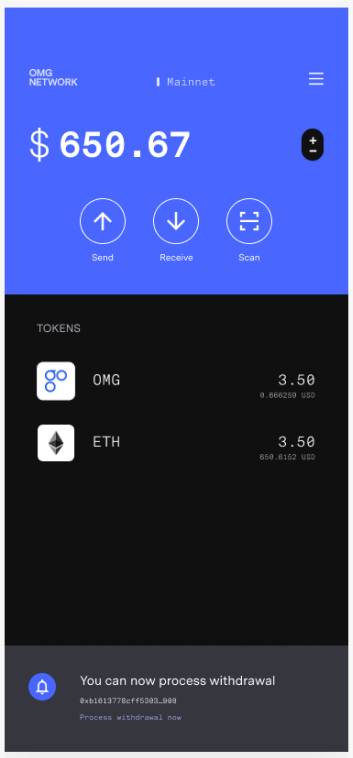


3



4

2. Each withdrawal has to be processed separately. That's why suggesting your users merging UTXOs first is a great UX pattern. Note, you can't mix different ERC20 tokens during a single UTXO merge so plan your UI accordingly to match this behavior.
3. A good practice for Layer 2 UX is to notify users (via both UI and push notification) when the challenge period has already passed and you can withdraw your funds back to the Ethereum network.



1

2

3. Setup a Watcher

Watcher is a service that guarantees data availability, secures the network, and allows to maintain the trustlessness of the OMG Network. It's recommended for every client to run a separate Watcher instance to rely on their own information source about the incoming deposits, transactions, exits, and byzantine events that happen on the OMG Network.

It also means you fully trust the OMG Network operator with everything that occurs on the network. Running your own Watcher ensures that only valid transactions and blocks are created and confirmed by the childchain operator.

There are several ways to deploy a Watcher. You can use the one you prefer the most via Run a Watcher guide.