

# Bridging ERC-20 Tokens to OP Mainnet With the Optimism SDK

This tutorial explains how you can use the [Optimism SDK\(opens in a new tab\)](#) to bridge ERC-20 tokens from L1 (Ethereum or Sepolia) to L2 (OP Mainnet or OP Sepolia). The Optimism SDK is an easy way to add bridging functionality to your javascript-based application. It also provides some safety rails to prevent common mistakes that could cause tokens to be made inaccessible.

Behind the scenes, the Optimism SDK uses the [Standard Bridge](#) contracts to transfer tokens. Make sure to check out the [Standard Bridge guide](#) if you want to learn more about how the bridge works under the hood.

The Standard Bridge does not support [fee on transfer tokens \(opens in a new tab\)](#) or [rebasing tokens \(opens in a new tab\)](#) because they can cause bridge accounting errors.

## Supported Networks

The Optimism SDK supports any of the [Superchain networks](#). [Some Superchain networks\(opens in a new tab\)](#) are already included in the SDK by default. If you want to use a network that isn't included by default you can simply [instantiate the SDK with the appropriate contract addresses](#).

## Dependencies

- [node\(opens in a new tab\)](#)
- [pnpm\(opens in a new tab\)](#)

## Create a Demo Project

You're going to use the Optimism SDK for this tutorial. Since the Optimism SDK is a [Node.js\(opens in a new tab\)](#) library, you'll need to create a Node.js project to use it.

### Make a Project Folder

```
mkdir
op-sample-project cd
op-sample-project
```

### Initialize the Project

```
pnpm
init
```

### Install the Optimism SDK

```
pnpm
add
@eth-optimism/sdk
```

### Install ethers.js

```
pnpm
add
```

ethers@5 Want to create a new wallet for this tutorial? If you have [cast \(opens in a new tab\)](#) installed you can run `cast wallet new` in your terminal to create a new wallet and get the private key.

## Get ETH on Sepolia and OP Sepolia

This tutorial explains how to bridge tokens from Sepolia to OP Sepolia. You will need to get some ETH on both of these testnets.

You can use [this faucet\(opens in a new tab\)](#) to get ETH on Sepolia. You can use the [Superchain Faucet\(opens in a new tab\)](#) to get ETH on OP Sepolia.

## Add a Private Key to Your Environment

You need a private key in order to sign transactions. Set your private key as an environment variable with the `export` command. Make sure this private key corresponds to an address that has ETH on both Sepolia and OP Sepolia.

```
export TUTORIAL_PRIVATE_KEY = 0 x...
```

## Start the Node REPL

You're going to use the Node REPL to interact with the Optimism SDK. To start the Node REPL run the following command in your terminal:

```
node
```

This will bring up a Node REPL prompt that allows you to run javascript code.

## Import Dependencies

You need to import some dependencies into your Node REPL session.

### Import the Optimism SDK

```
const
optimism
=
require ( "@eth-optimism/sdk" )
```

### Import ethers.js

```
const
ethers
=
require ( "ethers" )
```

## Set Session Variables

You'll need a few variables throughout this tutorial. Let's set those up now.

### Load your private key

```
const
privateKey
=
process . env . TUTORIAL_PRIVATE_KEY
```

### Create the RPC providers and wallets

```
const
I1Provider
=
new
ethers . providers . StaticJsonRpcProvider ( "https://rpc.ankr.com/eth_sepolia" ) const
I2Provider
```

```

=
new
ethers . providers . StaticJsonRpcProvider ( "https://sepolia.optimism.io" ) const
l1Wallet
=
new
ethers . Wallet ( privateKey , l1Provider ) const
l2Wallet
=
new
ethers . Wallet ( privateKey , l2Provider )

```

## Set the L1 and L2 ERC-20 addresses

This tutorial uses existing ERC-20 tokens that have been deployed on Sepolia and OP Sepolia. These tokens are designed for testing the bridging process.

```

const
l1Token
=
"0x5589BB8228C07c4e15558875fAf2B859f678d129" const
l2Token
=

```

"0xD08a2917653d4E460893203471f0000826fb4034" If you're coming from the [Bridging Your Standard ERC-20 Token to OP Mainnet Using the Standard Bridge](#) or [Bridging Your Custom ERC-20 Token to OP Mainnet Using the Standard Bridge](#) tutorials, you can use the addresses of your own ERC-20 tokens here instead.

## Get L1 Tokens

You're going to need some tokens on L1 that you can bridge to L2. The L1 testing token located at [0x5589BB8228C07c4e15558875fAf2B859f678d129 \(opens in a new tab\)](#) has a faucet function that makes it easy to get tokens.

## Set the ERC20 ABI

```

const
erc20ABI
= [{ constant :
true , inputs : [{ name :
"_owner" , type :
"address" }] , name :
"balanceOf" , outputs : [{ name :
"balance" , type :
"uint256" }] , type :
"function" } , { inputs : [] , name :
"faucet" , outputs : [] , stateMutability :

```

"nonpayable" , type :

"function" ]}]

## Create a Contract instance for the L1 token

```
const
I1ERC20
=
new
ethers.Contract (I1Token , erc20ABI , I1Wallet)
```

## Request some tokens

**tx**

```
await
I1ERC20.faucet () await
tx.wait ()
```

## Check your token balance

```
console.log (( await
I1ERC20.balanceOf ( I1Wallet.address)) .toString ()))
```

## Deposit Tokens

Now that you have some tokens on L1 you can deposit those tokens into theL1StandardBridge contract. You'll then receive the same number of tokens on L2 in return.

## Define the amount to deposit

The testing token has 18 decimal places, so you'll want to define a variable that represents one token.

```
const
oneToken
=
1000000000000000000 n
```

## Create a CrossChainMessenger instance

The Optimism SDK exports aCrossChainMessenger class that makes it easy to interact with theL1StandardBridge contract.

Create an instance of theCrossChainMessenger class:

```
const
messenger
=
new
optimism.CrossChainMessenger ({ I1ChainId :
11155111 ,
// 11155111 for Sepolia, 1 for Ethereum I2ChainId :
```

```
11155420 ,
```

```
// 11155420 for OP Sepolia, 10 for OP Mainnet I1SignerOrProvider : I1Wallet , I2SignerOrProvider : I2Wallet , })
```

## Allow the Standard Bridge to access your tokens

Before you can deposit your tokens, you'll need to give the Standard Bridge contract an allowance of tokens on L1. This will allow the Standard Bridge to pull these tokens out of your address and escrow inside the bridge.

**tx**

```
await
```

```
messenger .approveERC20 (I1Token , I2Token , oneToken) await
```

```
tx .wait ()
```

## Deposit your tokens

Now you can deposit your tokens into the Standard Bridge contract.

**tx**

```
await
```

```
messenger .depositERC20 (I1Token , I2Token , oneToken) await
```

```
tx .wait ()
```

Using a smart contract wallet? As a safety measure, depositERC20 will fail if you try to deposit ETH from a smart contract wallet without specifying a recipient . Add the recipient option to the depositERC20 call to fix this. Check out the [Optimism SDK docs \(opens in a new tab\)](#) for more info on the options you can pass to depositERC20 .

## Wait for the deposit to be relayed

You can use the waitForMessageStatus function to wait for the deposit to be relayed to L2.

```
await
```

```
messenger .waitForMessageStatus ( tx .hash ,
```

```
optimism . MessageStatus . RELAYED )
```

## Check your token balance on L1

You should now have one less token on L1.

```
console .log (( await
```

```
I1ERC20 .balanceOf ( I1Wallet .address)) .toString ())
```

## Create a Contract instance for the L2 token

```
const
```

```
I2ERC20
```

```
=
```

```
new
```

```
ethers .Contract (I2Token , erc20ABI , I2Wallet)
```

## Check your token balance on L2

You should now have one more token on L2.

```
console .log (( await
```

```
I2ERC20 .balanceOf ( I2Wallet .address)) .toString ())
```

## Withdraw Tokens

You just bridged some tokens from L1 to L2. Nice! Now you're going to repeat the process in reverse to bridge some tokens from L2 to L1.

### Start your withdrawal on L2

The first step to withdrawing tokens from L2 to L1 is to start the withdrawal on L2.

```
const
withdrawal
=
await
messenger .withdrawERC20 (I1Token , I2Token , oneToken) await
withdrawal .wait ()
```

### Check your token balance on L2

You should now have one less token on L2, but your token balance on L1 will not have changed yet.

```
console .log (( await
I2ERC20 .balanceOf ( I2Wallet .address)) .toString ())
```

### Wait until the withdrawal is ready to prove

The second step to withdrawing tokens from L2 to L1 is to prove to the bridge on L1 that the withdrawal happened on L2. You first need to wait until the withdrawal is ready to prove.

```
await
messenger .waitForMessageStatus ( withdrawal .hash ,
optimism . MessageStatus . READY_TO_PROVE ) This step can take a few minutes. Feel free to take a quick break while you wait.
```

### Prove the withdrawal on L1

Once the withdrawal is ready to be proven, you'll send an L1 transaction to prove that the withdrawal happened on L2.

```
await
messenger .proveMessage ( withdrawal .hash)
```

### Wait until the withdrawal is ready for relay

The final step to withdrawing tokens from L2 to L1 is to relay the withdrawal on L1. This can only happen after the fault proof period has elapsed. On OP Sepolia, this is only a few seconds. On OP Mainnet, this takes 7 days.

```
await
messenger .waitForMessageStatus ( withdrawal .hash ,
optimism . MessageStatus . READY_FOR_RELAY )
```

### Relay the withdrawal on L1

Once the withdrawal is ready to be relayed you can finally complete the withdrawal process.

```
await
messenger .finalizeMessage ( withdrawal .hash)
```

### Wait until the withdrawal is relayed

Now you simply wait until the message is relayed.

await

```
messenger .waitForMessageStatus ( withdrawal .hash ,  
optimism . MessageStatus . RELAYED )
```

## Check your token balance on L1

You should now have one more token on L1.

```
console .log (( await  
I1ERC20 .balanceOf ( I1Wallet .address)) .toString ()))
```

## Next Steps

Congrats! You've just deposited and withdrawn tokens using the Optimism SDK. You should now be able to write applications that use the Optimism SDK to transfer ERC-20 tokens between L1 and L2. Although this tutorial used Sepolia and OP Sepolia, the same process works for Ethereum and OP Mainnet.

[Bridging ETH With the Optimism SDK](#) [Bridging Your Custom ERC-20 Token to OP Mainnet](#)