

TypeScript SDK

info [SUAVE-Viem](#) is a fork of [viem](#) that will eventually be upstreamed but is currently still in a dynamic state.

Sending Confidential Compute Requests works slightly differently, but most other functionality is similar to interacting with any other EVM chain from viem. This page describes how to work with the SUAVE-viem TypeScript SDK. The SDK simplifies interaction with the SUAVE Chain and provides easy-to-use functions to send transactions and query data. Below, you'll find steps on how to instantiate the library, symlink, and perform some basic actions.

Installation

The [@flashbots/suave-viem](#) package is available on NPM, and can be installed with any NPM-based package manager, such as `npm`, `yarn`, or `bun`.

warning Alpha Release Version

Note: you must use the following reference when importing, or else your project won't import the required types. This is a temporary measure while we fix the tests that are failing in CI, which prevents a full release.

```
@flashbots/suave-viem@main * npm * yarn * bun
```

```
npm i @flashbots/suave-viem@main yarn
```

```
add @flashbots/suave-viem@main bun add @flashbots/suave-viem@main
```

Instantiation

info The rest of this guide assumes you have [SUAVE running locally](#). First, you need to import necessary modules and instantiate the client. In your `index.ts` file, you can copy and paste the following:

```
import
{ http }
from
'@flashbots/suave-viem' ; import
{ getSuaveProvider }
from
'@flashbots/suave-viem/chains/utlis' ;
// connect to your local SUAVE node const
SUAVE_RPC_URL
=
'http://localhost:8545' ; const suaveProvider =
getSuaveProvider ( http ( SUAVE_RPC_URL ) ) ;
```

Wallet Creation

To interact with the SUAVE network, we'll first need a wallet. When running SUAVE locally, there is an account which is set up with funds for you by default. Paste the following the following block to instantiate it in `viem` :

```
// plus other imports from above import
{ Hex }
from
'@flashbots/suave-viem' ; import
{ getSuaveWallet }
from
'@flashbots/suave-viem/chains/utlis' ;
const
DEFAULT_PRIVATE_KEY :
Hex
= '0x91ab9a7e53c220e6210460b65a7a3bb2ca181412a8a7b43ff336b3df1737ce12' ;
const defaultWallet =
getSuaveWallet ( { transport :
http ( SUAVE_RPC_URL ) , privateKey :
DEFAULT_PRIVATE_KEY , } ) ;
console . log ( 'Wallet Address:' , defaultWallet . account . address ) ; You can now run this file:
run index.ts And you should see the following printed to your terminal:
Wallet Address: 0xBE69d72ca5f88aCba033a063dF5DBe43a4148De0
```

Watching Pending Transactions

You can watch for pending transactions and log their details using the following example:

```
// Watch for pending transactions suaveProvider . watchPendingTransactions ( { async
onTransactions ( transactions )
{ for
( const hash of transactions )
{ try
{ const receipt =
await suaveProvider . getTransactionReceipt ( { hash } ) ; console . log ( 'Transaction Receipt:' , receipt ) ; }
catch
( error )
```

```
{ console . error ( 'Error fetching receipt:' , error ) ; } } , ) ;
```

Send a Confidential Compute Request

Let's walk through how to set up and send a Confidential Compute Request:

1. Get Current Gas Price

First, instantiate a new wallet of your own, and fetch the current gas price from the network.

```
const
PRIVATE_KEY :
Hex
=
"; const gasPrice =
await publicClients . suaveLocal . getGasPrice ( ) ;
const wallet =
getSuaveWallet ( { transport :
http ( SUAVE_RPC_URL ) , privateKey :
DEFAULT_PRIVATE_KEY , } ) ;
```

2. Prepare the Fund Transaction

Create a transaction object to fund your new wallet with the required amount.

```
import
{ TransactionRequestSuave }
from
'@flashbots/suave-viem/chains/suave/types' ; // ...
const fundTx :
TransactionRequestSuave
=
{ type :
'0x0' , value :
1000000000000000000n , gasPrice : gasPrice +
10000000000n , chainId : suaveRigil . id , to : wallet . account . address , gas :
21000n , } ;
```

3. Send the Fund Transaction and wait for confirmation

Send the transaction to fund the wallet.

```
const fund =
await defaultWallet . sendTransaction ( fundTx ) ; console . log ( 'sent fund tx' , fund ) ; Use a while loop to periodically check if the transaction has been confirmed.
while
( true )
{ const fundReceipt =
await suaveProvider . getTransactionReceipt ( { hash : fund , } ) ; if
( fundReceipt )
{ console . log ( 'fund tx landed' , fundReceipt ) ; break ; } await
sleep ( 4000 ) ; } If you once again run bun run index.ts , you should see something like the following logged to your terminal:
```

[illegible]

4. Create a Confidential Compute Request

Now, let's set up a CCR with the appropriate parameters.

```
import { TransactionRequestSuave , SuaveTxRequestTypes , } from '@flashbots/suave-viem/chains/utils' ; // ... const ccr : TransactionRequestSuave = { confidentialInputs : "0x00000000000000000000000000000000000000000000000000000000200000000000000000000000000000000000000000000000000000d7b2626c6f636b4e756d626572223a2" , kettleAddress : "0xB5fEAfbDD752ad52Afb7e1bD2E40432A485bBB7F" , // Address of your local Kettle. Use 0x03493869959C866713C33669cA118E774A30A0E5 if on Rigil. to : "0x8f21Fd6B4f4CacD33151777A46c122797c8BF17" , gasPrice :
```

[illegible]