Suave Chain

Table of Contents

Overview

This document outlines the specifications for the SUAVE Rigil chain.

In the context of the SUAVE protocol, the primary purpose of the SUAVE chain is to reach (and maintain) consensus about smart contract code for use cases such as order flow auctions, solvers, block builders, etc. Additionally, the SUAVE chain can also be used to store and broadcast data for better censorship guarantees.

In the initial phases of development, the SUAVE chain runs a proof-of-authority consensus protocol called Clique, over a network of permissioned nodes. We do so to experiment and iterate quickly during protocol development. This will change in later testnets.

Configuration

Network Parameters

- Network ID
- :16813125
- Chain ID
- :16813125

Genesis Settings

Name Value Unit PERIOD 4 block EPOCH 30000 block BLOCK_TIME 3 second GAS_LIMIT 30000000 gas NUM_VALIDATORS 5 Nodes

Consensus Mechanism: Proof-of-Authority (Clique)

Clique, an Ethereum-based Proof-of-Authority consensus protocol definedhere , restricts block minting to a predefined list of trusted signers. Because of this, every block header a client sees can be checked against the list of trusted signers.

Geth Version

Suave-geth is based on geth v1.12.0 (e501b3).

Suave Transaction

The SUAVE protocol adds a new transaction type to the base Ethereum protocol called aSuaveTransaction . The purpose of this new transaction type is to process fees for offchain computation and to support the new data primitives associated with confidential compute.

Blocks on the SUAVE chain consist of lists of SUAVE transactions. This new transaction type facilitates and captures key information involved in Confidential Compute Requests and their subsequent results. AnyConfidentialComputeRequest , signed by the user, specifies anKettleAddress . SUAVE transactions are valid if and only if they are signed by theKettleAddress specified by the user in the originalConfidentialComputeRequest , which is included as theConfidentialComputeRecord .

type SuaveTransaction struct

{ ConfidentialComputeRequest ConfidentialComputeRecord ConfidentialComputeResult [ ] byte

// Kettle's signature ChainID * big . Int V * big . Int R * big . Int S * big . Int }

Suave Transaction Types

SuaveTransaction is the primary transaction type which is returned when requesting transactions from endpoints likeeth_getTransactionByHash oreth_getBlockByNumber , but SUAVE introduces two other important message types:ConfidentialComputeRequest andConfidentialComputeRecord . All new types are detailed in the following table:

Name EIP-2718 Tx Type Description SuaveTransaction 0x50 SUAVE transaction; product of executed ConfidentialComputeRequest ConfidentialComputeRequest 0x43 Sent by users to interact with SUAVE smart contracts using confidential inputs ConfidentialComputeRecord 0x42 Artifact of ConfidentialComputeRequest; represents record that is stored on SUAVE chain

TransactionRequest Serialization & Signing

Messages sent by users of SUAVE can take on two forms:

1. Standard (legacy) Ethereum transaction
2. ConfidentialComputeRequest

Standard transactions are used to transfer SUAVE-ETH and deploy smart contracts to SUAVE chain. ConfidentialComputeRequests are a newEIP-2718 message type, used to interact with SUAVE smart contracts. Note that 'ConfidentialComputeRequests' arenot classified as 'transactions' since 'transaction' implies changes to the state of a database/blockchain. Kettles do not provide persistent storage guarantees and are instead intended to convert inputs to outputs (like transactions targeted at blockchains).

All transactions are encoded with theEIP-2718 RLP-encoding scheme (withEIP-2930 allowed), butConfidentialComputeRequest takes on a special signature scheme that deviates slightly from the traditional method.

ConfidentialComputeRequests adopt this unique signing scheme to keepconfidentialInputs off-chain. The ConfidentialComputeRecord, which is signed by the sender, contains only thehash ofconfidentialInputs . This record is stored on the SUAVE chain, making verification ofconfidentialInputs possible without exposing the actual data on-chain.

In javascript, aConfidentialComputeRequest has the following structure:

const cRequest =

{ confidentialInputs :

'0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000fd7b22626c6f636b4e756d626572223a22
, kettleAddress :

'0xb5feafbdd752ad52afb7e1bd2e40432a485bbb7f' , to :

'0x8f21Fdd6B4f4CacD33151777A46c122797c8BF17' , gasPrice :

10000000000n , gas :

420000n , type :

'0x43' , chainId :

16813125 , data :

'0x236eb5a7000000000000000000000000000000000000000000000000000000000000002000000000000000000000000000000000000000000000000000000000000006000000000000000000000000
, } Note: new fields:confidentialInputs andkettleAddress .

To serialize, sign, and send this request, the client must first RLP-encode the request as aConfidentialComputeRecord and sign its hash.

Note: the following is pseudo-code.rlp ,keccak256 , andwallet.sign implementations may differ.

const

{ nonce , gasPrice , gas , to , value , data , kettleAddress , confidentialInputs , chainId , }

= cRequest const rlpRecord =

rlp ('0x42' ,

[kettleAddress , keccak256 (confidentialInputs) , nonce , gasPrice , gas , to , value , data ,]) const

{ v , r , s }

= wallet . sign (keccak256 (rlpRecord)) cRecord =

{ ... cRecord , v , r , s } Then, the final request is re-encoded with RLP as follows:

// assume (v, r, s) have been added to cRecord const tx =

rlp ('0x43' ,

[cRecord . nonce , cRecord . gasPrice , cRecord . gas , cRecord . to , cRecord . value , cRecord . data , cRecord . kettleAddress , keccak256 (cRequest . confidentialInputs) , cRecord . chainId ,
cRecord . v ===

27n

?

'0x'

:

'0x1' ,

// yParity cRecord . r , cRecord . s ,] , confidentialInputs) This is then sent to SUAVE viaeth_sendRawTransaction :

wallet . request ('eth_sendRawTransaction' ,

[tx])

Node Requirements and Setup

- Hardware
- :* Minimum 8GB RAM, four cores, 50GB SSD (How big do we expect the chain to grow?)
-
  - These requirements will eventually incorporate Trusted Execution Environments (TEEs).
- Software
- :suave-geth
- Setup Steps
- :1. Clone suave-geth
-
  1. Start the devnet:make devnet-up
-
  1. Create transactions:go run suave/devenv/cmd/main.go

Gas and Transaction Fees

The SUAVE chain employs the same gas pricing mechanism as Ethereum pre-Cancun hardfork (no blob transactions) where gas prices adjust based on network demand. Nodes currently track Confidential Compute Request gas usage but only charge a small flat fee for it, and there is no cap for offchain compute.

Currently, SUAVE transactions can only be expressed as Legacy transaction types, but they will get converted into EIP-1559 base fee model under the hood.

Security Considerations

- Security Risk
- : The protocol is unaudited. The protocol currently does not make any guarantees about the confidentiality of data in the network outside of a best effort.
- DoS Risk
- : Nodes have not yet been reviewed, and there may be DoS vectors at this early stage.
- Secure Key Management
- : Storing private keys on Suave is experimental and should be considered insecure.

If you find a security vulnerability in SUAVE, please email us atsecurity@flashbots.net . Edit this page Previous Kettle Next Glossary