SUAVE Rigil Testnet

This repository hosts the current SUAVE Rigil testnet specifications and design docs.

⚠ The SUAVE protocol is still in a state where the code is the most up-to-date protocol spec and open questions are being researched. The goal of these notes is to gradually evolve into an implementation-agnostic specification. ⚠

Table of Contents

Specs

We recommend that you read the specs in the following order:

- Suave Chain
- MEVM
- Precompiles
- Confidential Data Store
- Kettle
- Bridge

About SUAVE

SUAVE - Single Unifying Auction for Value Expression - is a platform for building MEV applications such as OFAs, block builders, and intent executors in a decentralized and private way. SUAVE does not replace other blockchains: it is intended to aggregate and coordinate all the things that ultimately change the state of other chains.

Read more about SUAVE:

- https://writings.flashbots.net/the-future-of-mev-is-suave
- https://writings.flashbots.net/mevm-suave-centauri-and-beyond

Rigil Overview

This set of specs outlines the Rigil Testnet, a continuation of the star system theme (Centauri, Andromeda, Helios) laid out in The Future of MEV ; and the first in a series of SUAVE testnets based on stars in the (Alpha) Centauri system : Rigil Kentaurus (Alpha Centauri A), Toliman (B) and Proxima Centauri (C).

The Rigil Testnet is a developer focused sandbox for creating SUAPPs (MEV applications) in a way that's both decentralized and confidential. It features the MEVM, a variant of the EVM, which equips developers with the ability to write SUAPPs as smart contracts by giving them access to unique MEV-specific precompiles. SUAPPs can send transactions and intents confidentially to a network of searchers, solvers, block builders, and more.

Rigil provides a live, Flashbots-hosted test network for rapid prototyping that uses Goerli ETH for gas and operates with a proof-of-authority consensus mechanism.

Rigil's architecture is composed of several parts:

- SUAVE Kettles: a network of actors that provide confidential computation for SUAPPs.

- Confidential Data Storage: a private place to store data (e.g. user transactions).
- SUAVE Chain: a public place to store data (e.g. intentionally leaked information) and SUAPP logic (e.g. deployed smart contracts).
- MEVM: a modified EVM that exposes confidential computation and storage APIs to developers

The goal of the Rigil testnet is to gather feedback on developer experience and harden the overall SUAVE software stack. The testnet is not intended to be a long-lived network and will be decommissioned after the launch of the next testnet Toliman.

Users

The Rigil testnet is initially focused on a specific set of actors:

1. Developers
2.
    - create smart contracts on SUAVE Chain that define rules for SUAPPs like order flow auctions, block building, and intent executors.
3. Transaction Originators
4.
    - leverage unique applications on SUAVE, e.g. to send private transactions or execute your intents.
5. Proposers
6.
    - outsource block building to SUAVE. Initially, SUAVE is focused only on Ethereum.
7. Block Builders
8.
    - can be implemented as smart contracts inside Suave. In the Rigil Testnet, SUAPPs can submit bundles to external builders to help with transaction inclusion during the early stages of development.
9. Auction Protocols
10.
    - can program their auctions as a smart contract.

Rigil Design Goals

1. Permissionless
2.
    - Allow anyone to deploy smart contracts on SUAVE.
3. Easy to use
4.
    - Create an environment that is as easy to use and test as possible, enabling rapid prototyping.
5. SGX UX Closeness
6.
    - The DevEx on Rigil should be forward-compatible with future constraints imposed by trusted enclaves (i.e. SGX).

Design Decisions

Here is a list of design decisions made for the Rigil testnet and associated reasoning:

- Decision1
- :Proof-of-Authority Consensus
-
    - reason
-
    - :SUAVE consensus
-
    - is an active open question which, whether answered or not, does not drastically impact the UX of users onRigil Testnet
-
    - .
- Decision2
- :No SGX Nodes (yet)
-
    - reason: SGX SUAVE Kettles are an active area of research and development and do not drastically impact the UX of users onRigil Testnet
-
    - .
- Decision3
- :Weak Data Availability Guarantees

-
  - Reason: The Confidential Data Store currently only keeps private data available for one dayCompute Output Validity and Heterogenous DA
-
  - are active open questions which, whether answered or not, does not drastically impact the UX on Rigil Testnet.
- Decision4
- :Centralized Builder Interoperability
-
  - reason: Blocks emitted from SUAVE Kettles will have unpredictable inclusion in early development so Rigil Testnet supports a precompile to send bundles to off-SUAVE block builders.

Glossary

Seeglossary.md .

Architecture

SUAVE Kettles house all components necessary to perform confidential compute and are the main protocol actor in the SUAVE protocol. Below is a high-level architectural overview.

A broad-level view of how a SUAPP gets onchain and utilizes SUAVE core components is as follows:

1. Developers
2. create contracts, which contain the logic for their SUAPP. A typical flow might look like: intake and validate user L1 transaction, simulate it on L1 state, then do something based on the simulation results. These contracts are deployed to the SUAVE chain by sending to aKettle
3. .
4. Users
5. sendConfidential Compute Requests
6. directed to aKettle
7. or multiple.
8. Inside theKettle
9. :* Requests are routed using theRPC
10.
    - to the MEVM or regular EVM, depending on the context.
11.
    - TheMEVM
12.
    - processes the confidential computation or smart contract call.
13.
    - Data might be stored in or retrieved from theSuave Chain State
14.
    - orConfidential Data Store
15.
    - based on SUAPP needs.
16.
    - Precompiles
17.
    - aid in the efficient execution of certain functions.
18.
    - Domain-Specific Services
19.
    - handle execution on different domains (such as simulation against Ethereum state) and return results toMEVM
20.
    - .
21. TheSuave PoA Chain
22. stores results of confidential computation.

Open Questions

The are multiple open questions that need to be solved in the long term about SUAVE architecture. A non-exhaustive list of questions are:

- Consensus Requirements
- Output Validity and Heterogenous DA
- SUAVE Economic Security Models
- Bridge Request For Comments

Example Flows

The example flows in the following sections are used to illustrate some of the possibilities for SUAPPs on SUAVE. Flows start at a high level showing how an OFA and Block Builder SUAPPs work together to emit a block from a SUAVE Kettle. Then to understand the Confidential Compute Request Flow more deeply, there is a detailed data flow diagram, followed by two deeper dives into the specifics of how the OFA and Block Builder flows work individually.

High Level - OFA + Block Builder

Below we can see the journey of order flow from transaction, to searcher back-run, to a block emitted from SUAVE.

1. A user sends their L1 transaction, EIP-712 message, UserOp, or Intent into a SUAVE Kettle.
2. MEVM processes this L1 transaction, extracts a hint (data intentionally leaked by the contract), and does two things:* Stores the L1 transaction in the confidential data store
3.
    - Sends a SUAVE transaction to the mempool which when executed emits the hint as a log
4. Searchers will be listening on two different lanes for hints:* The fast lane which is the mempool
5.
    - The global lane which is the SUAVE chain, is slower but will surface any hints that may have been censored by your specific peer in the mempool
6. Once a hint is received, searchers craft a backrun transaction and send it to a SUAVE Kettle.
7. SUAVE Kettles will process the backrun, combine it into a bundle with the original transaction, include the bundle in a block, and then submit the block to a relay.

Optionally , bundles can be sent straight to a centralized block builder.

Confidential Compute Request Flow

The SUAVE Kettle and the MEVM support multiple new data types, which are all specified in the Kettle spec .

The diagram below showcases how these different types interact to enable confidential computation on SUAVE Kettles.

Transaction Flow:

1. User sends a Confidential Compute Request to the RPC - Confidential Compute Requests are made up of two components, Compute Transaction and Confidential Inputs. The Compute request will reference data inside of the Confidential Inputs that the MEVM is able to use during computation.
2. Once the RPC receives the Confidential Compute Request, it will extract the confidential inputs and send them to the confidential data store. It will then send the Compute Request to the MEVM to process.
3. Confidential Compute Phase - During this phase, the MEVM will process the compute transaction similar to an EVM transaction except it will also have access to the confidential inputs. After doing the initial computation with the confidential data, it will then grab the results and information from the Compute Transaction and put them into their final home a SUAVE transaction.
4. Block Inclusion - Once the SUAVE transaction has been created it will then quickly be included in a block by a SUAVE proposer.

OFA Example

If we consider a specific use case, like an order flow auction, the high-level series of steps taken to complete the auction can represented as below:

1. The user sends a Confidential Compute Request interacting with a SUAPP by calling itsnewTransaction
2. function. Included in this request is also the user's L1 transaction as a confidential Input.
3. The Kettle will receive the transaction and process it. To do so it first runs the offchain logic associated withnewTransaction
4. which will extract the transaction's data and then return a callback:

return bytes . concat (this . emitDataRecord . selector , abi . encode (dataRecord)) ; which points to another function:

function emitDataRecord (Suave . DataRecord calldata dataRecord) public { emit DataRecordEvent (dataRecord . id , dataRecord . decryptionCondition , dataRecord . allowedPeekers) ; } 1. The callback is inserted into the calldata of a SUAVE transaction and then shipped off to the SUAVE mempool. 2. The transaction will get picked up, inserted in a SUAVE block, and propagated to SUAVE Kettles. 3. From here a searcher monitoring the chain and this specific OFA will see the log emitted and begin processing. 4. Once the searcher has a backrun crafted for the opportunity it will send it to the Kettle as a Confidential Compute Request with the backrun transaction in the confidential inputs. 5. The Kettle will receive and process the searcher's Confidential Compute Request based on the contract's logic. In this case, it will:* Grab the referenced User Transaction to be placed behind 6. * Construct a bundle object with the two transactions 7. * Submit to domain-specific service for simulation and validation 8. From there, in this example, the MEVM will then forward the bundle to pre-configured off-SUAVE block builders, but could as easily also forward to onchain block builders.

The important thing to note here is that this Confidential Compute pattern makes it such that no sensitive data gets leaked onchain except for what the smart contract specifies, and thus creates programmable information leakage.

Block Building Example

Blocks built from SUAVE will have unpredictable inclusion in the beginning, but adventurers are invited to hook into the block building flow already achievable today. Below is a walkthrough of a block being built via a solidity smart contract.

1. The user sends a Confidential Compute Request that specifies callingbuildBlock
2. on the onchain block builder contract.
3. The Kettle receives and processes the transaction; specifically, the logic will:* grab all bundles that are stored in the block builders' confidential data store
4.
    - simulate all bundles
5.
    - sort bundles via arbitrary logic but in this gas by effective gas price
6.
    - compute state root and package into a block
7. Optional: Similar to the above, the confidential compute result can be a callback which will emit a log of the block's bid value onchain as well as a header that a validator can view.
8. Similar to sending to a centralized block builder, the MEVM will then send the block to a centralized relay where it is free to access by validators. Edit this page Previous README Next Kettle