“We designed SUAVE to be **the mempool and block builder for all blockchains.**”

OH: “But you do you wanna make money or do you wanna be right”

– 0xPandebug, mevSteward

prefer a used Camry

Puppet Master <-> Meme Lord <-> [CT]

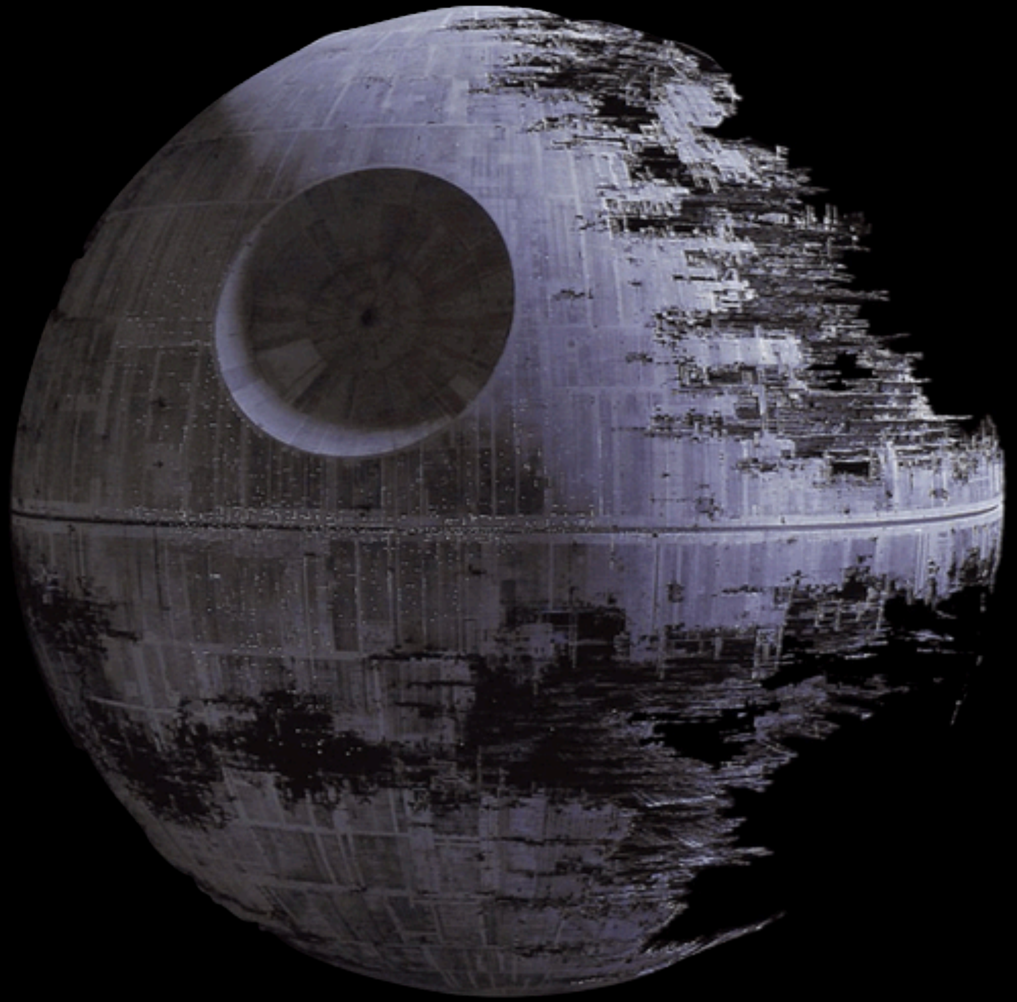
At the core of SUAVE is the concept of *preferences*. A preference is a message that a user signs to express a particular goal and that unlocks a payment if the user's conditions have been met. These **preferences can range from simple transfers or swaps in a single domain to arbitrarily complex sequences of events across multiple blockchains**. You can think of preferences as the native transaction type on *SUAVE*. They can either contain a payload to be executed on a specific domain—such as *Ethereum*—or make a more abstract statement of what the user wants to achieve and leave the optimal routing to the executors.

A SUAVE bid (preference) maps future states of the world to their utility to the bidder.

A bid (preference) is a program (smart contract on SUAVE), such that exec (bid, outputs be where b is a natural number that is paid to address e (the executor who executed the SUAVE transaction)

User/Searchers <-> SUAVE <-> [Executor <-> L1]

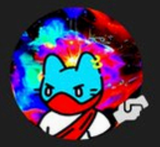
* SUAVE is not yet Operational



Quintus
@0xQuintus

most concrete description of suave i've seen

collective.flashbots.net/t/suave-as-x/1...



apriori
friend

2d

Suave as an air traffic controller for AGI searchers

10:22 AM · Mar 17, 2023 · **1,814** Views

1 Retweet **13** Likes

What is SUAVE?

a shampoo, a Troll, or a single unified auction ?

* Fully decentralized block-builder
* Fully open source, open development
* ETH-Native / EVM-compatible
* Optimal user execution, harnessing MEV
* Full compatibility with Flashbots today
* Cross/multi-chain support
* Maximizes competition and geographic diversity
* Enabling open order-flow for Ethereum's future
* Programmable privacy

SUAVE is a stateful system
SUAVE states are denoted by s1 -> s2 ->s3 ..., where s is the current SUAVE state and S' is the set of all historical/past SUAVE states.

Bids on SUAVE are executable programs that establish the conditions for releasing a payment to the executor. Bids are denoted b and are composed of:

* A deposit k.
* A payee e : S - A.
* An execution predicate P(s).
* An encrypted data blob c.
* An encryption predicate Q(s).
* A set of peekers ∑.

Here, e maps from SUAVE states s € S to the set of SUAVE accounts A, and predicates P(s) and Q (s) are likewise over states of SUAVE.

The result of executing a bid b, if the current state of SUAVE S_cur satisfies P, is to transfer the deposit k to the e(S_cur) account, thus modifying SUAVE state to S_next. If S_cur does not satisfy P, it remains unaltered. We elaborate on c, Q (s), and & in the coming sections.

Example **Bids**; exec(bid, s) outputs b, e where b is a natural number that is paid to address e

If in block 2, transaction OxUltraSound comes at position 0, pay 3TH (b) to the sender of the t× (e) at position 1

Or

If in block 3, transaction OxUltraSound comes before transaction OxZkPanda comes before transaction 0xOpenWaterIce(effective sandwich), pay 3ETH (b) to the sender of the tx following the sandwich (e)

Or

If block 3 is empty, pay the miner (e) 3TH (b)

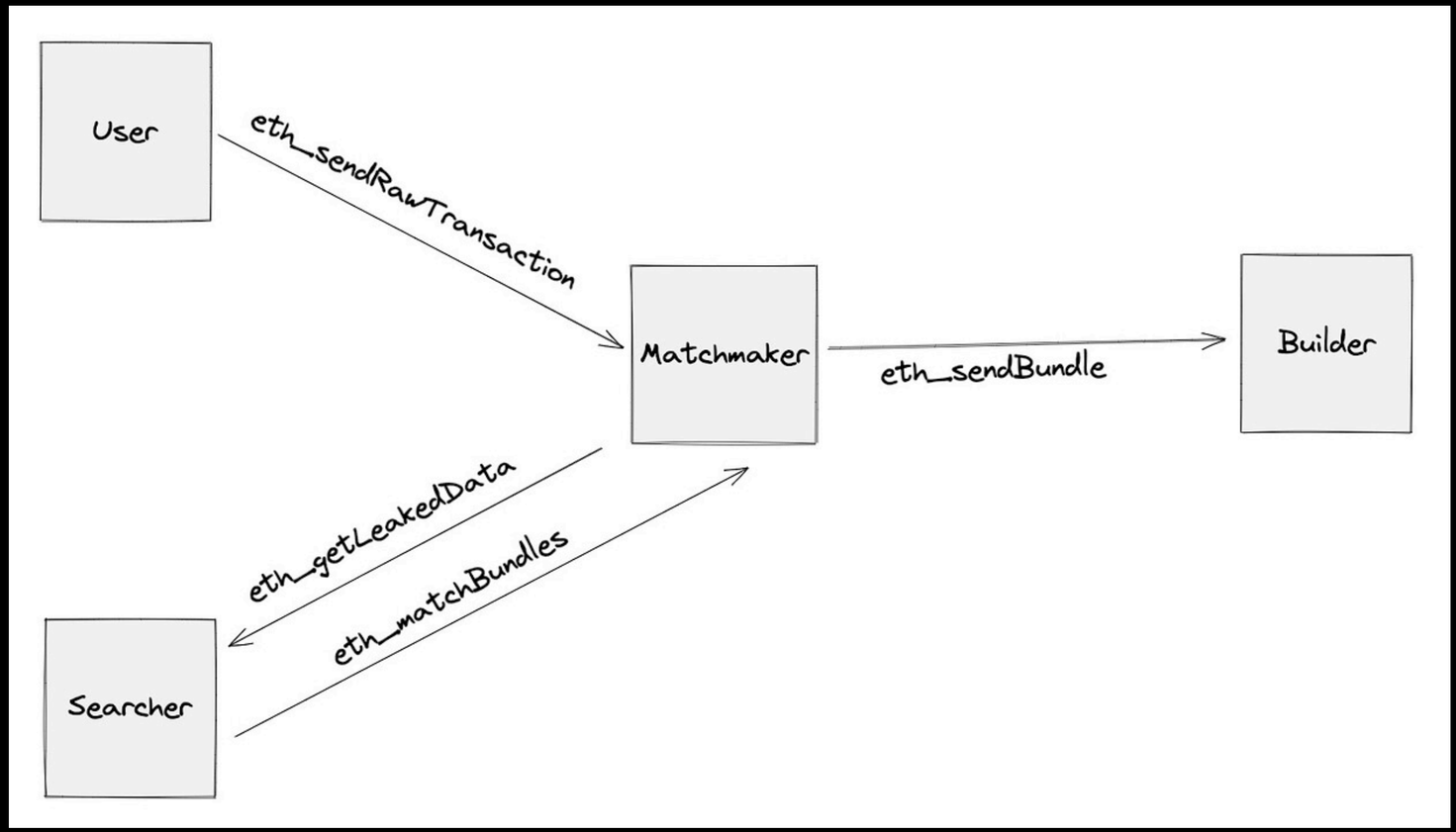
Or

If [x] is in the execution logs (w Merkle proof), pay the originator of the transaction corresponding to [x] (e) 3ETH (b)

SUAVE contains special oracle contracts that are responsible for importing external events into s. One simple example is an ETHOracle contract, which allows SUAVE bids to query ETH history.

Pt.1 MEV-Share

- Users, or wallets acting on their behalf, can program their transaction **privacy preferences** to decide what data to selectively disclose to searchers
- Searchers indicate in their bundles where to **insert private transactions**
- Searchers provide **hints**, which are information that helps to match their bundles against private transactions
- The **matchmaker** attempts to match searcher bundles against private transactions and returns feedback to searchers to help them optimize their bundles
- Any successfully matched bundles are sent on to builders along with a **validity condition** that requires the user be paid ETH



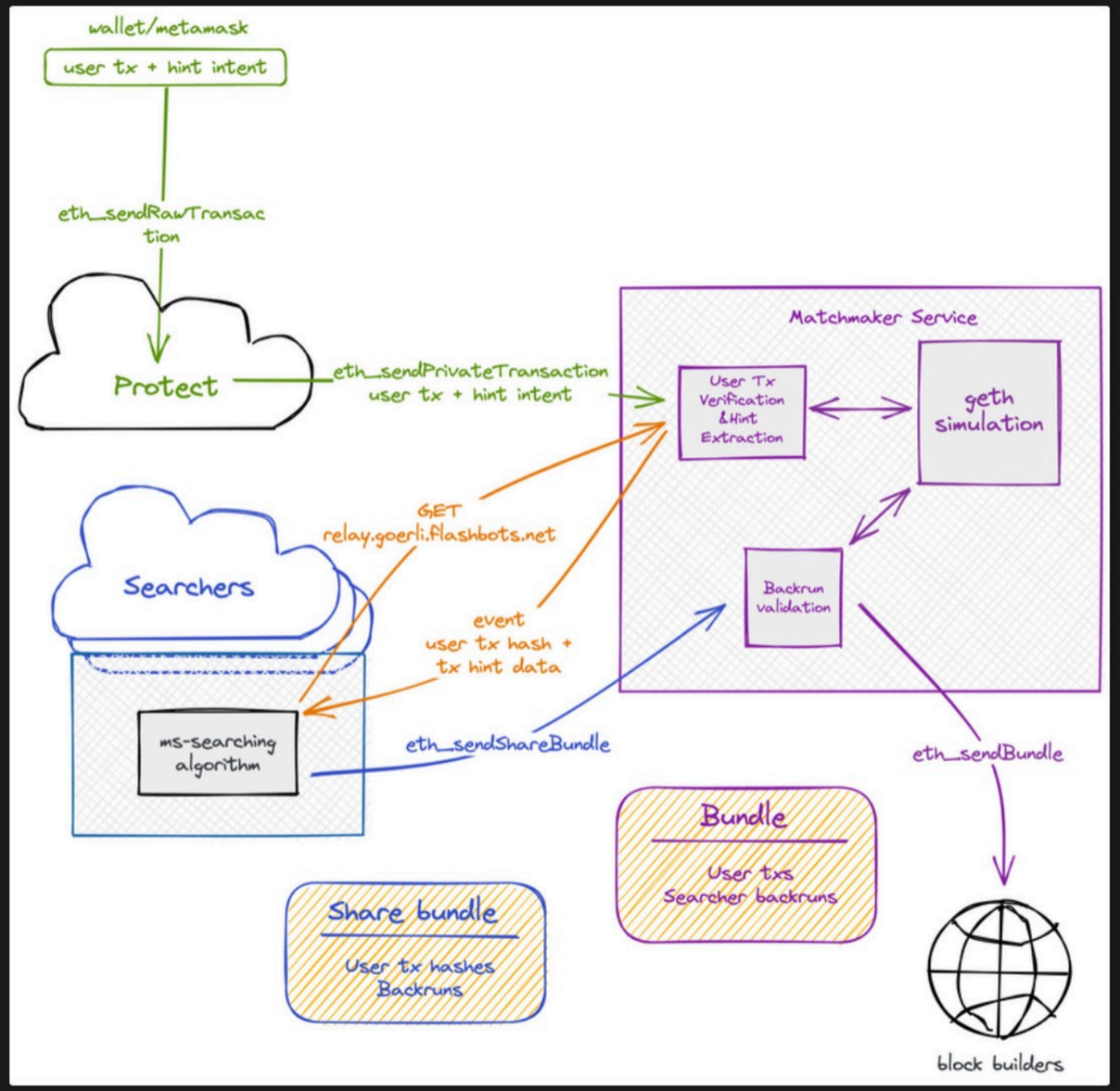
- How much info to leak vs. keep private?
- Most extreme: searcher learns 0 until block is signed
  - Can only reorder/merge encrypted blobs (w own arb txs)
- Least extreme / 0 privacy: send to mempool
- Middle grounds (eg take a Uniswap tx):

  (a) Reveal that TX is Uniswap?

  (b)Reveal pools being traded?

  (c)Reveal trade direction?

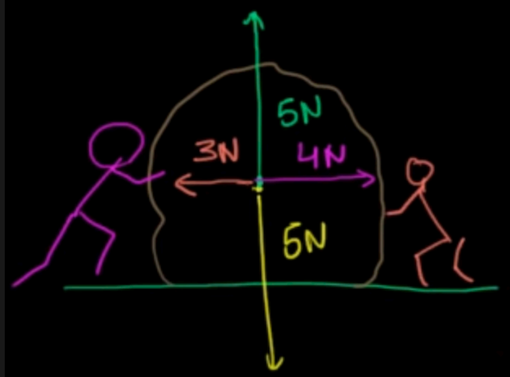
Architecture overview



https://excalidraw.com/#json=h_aUFeChVdC1Yv5FZzplw,rVvIUC4TExS5mFdFGpjUiA

sendBackrunBundle

1. `sendTestBackrunBundle`: This function generates a transaction to backrun a pending MEV-share transaction and sends it to MEV-share. It targets the next `NUM_TARGET_BLOCKS` (5 by default) blocks for the backrun.
2. `handleBackrun`: This async function handles the backrunning process. It receives a pending MEV-share transaction, a provider, a matchmaker, and a Mutex to manage thread synchronization. It calculates the target block and sends backrun bundles. It also watches future blocks for the backrun transaction inclusion, and if it succeeds, it releases the Mutex.
3. `main`: This function initializes the provider, the matchmaker, and sets up the Mutex. It listens for transactions using the `txHandler`, which calls the `handleBackrun` function for every pending transaction. The `blockHandler` sends a transaction on every new block. The main function waits for the Mutex to be released, indicating that the backrun was successful, and then stops listening for transactions and new blocks.



sendTx

```
import { HintPreferences } from '../api'
import { getProvider } from './lib/helpers'
import { sendTx } from './lib/sendTx'

const main = async () => {
    const provider = getProvider()
    const hints: HintPreferences = {
        calldata: true,
        logs: true,
        contractAddress: true,
        functionSelector: true,
    }
    console.log("sending tx to Flashbots Bundle API...")
    await sendTx(provider, hints)
}

main()
```

MEV-GETH in SUAVE

searcher - - I would like [tx1, tx2, tx3] mined
Relay - - Estimate profit to miner for [t×1, t×2, tx3], prune to only send most promising bundles to miner
Sequencer - - Receive relay bundles, re-execute to confirm payment

==========>

Searcher - - I would like [t×1, t×2,t×3] mined, and if they are, I would like to pay 3ETH (b, unconditional)
SUAVE - - Propagate and aggregate searcher input, ensure payments are valid
Executor - - Ensure validators perform economically optimal action

Case 1: Validator is SUAVE-native
Validator can profit switch b against best known mempool block

Case 2: Validator is SUAVE-unaware
Executor gets miner to mine optimal TX order
PGA
MEV-geth-style trusted or native payments

native plugins for Case 1 while allowing anyone to develop Case 2

SUAVE Centauri

- Privacy-aware orderflow auction to return to users the MEV that their transactions create. In this auction, searchers compete for the right to back run a user, thereby bidding up the value returned to them. Initially, the auction assumes trust in Flashbots but is private for users and searchers
- SUAVE Chain testnet for stress testing and community experimentation.

SUAVE Andromeda

- SUAVE Chain mainnet will allow users to express preferences and send them to the Execution Market
- SGX-based orderflow auction to remove trust in Flashbots and make the auction for efficient for searchers
- SGX-based centralized block building to enable open but private orderflow for centralized builders

SUAVE Helios

- SGX-based decentralized building network to allow for permissionless and private collaborative block building across many entities
- Onboard second domain to Suave to address MEV on another domain and provide a foundation for cross-domain MEV
- Cross-domain MEV solutions that allow for expression and execution of cross-domain MEV preferences

But do you wanna be right or do you wanna win? Lol, lmk

smartTransactions.wtf

Smart transactions are defined in terms of the wide scope of their validation capacities, more than the MEV for which their valid traces are optimized, at MEV-time

Smart transactions semantics are innovative in computer science, they reduce costs and have quantitatively qualitatively better risk management than traditional economic transactions, they increase the power of transactions and create very low liability virtual service transaction provider roles, and open up transaction semantics

Appendix

wtfIssaSmartTransaction?

Smart transactions are useful for infrastructure development, for all prior blockchain use cases, and shiny new smart-transaction-only features

Appendix

Like what can you do tho?

- We can build true Access Control Lists w.o. code analysis,
- we can save gas and pay without ETH
- rent as expected MEV to justify state in RAM
- automated collective bargaining
- direct micro/meso/macroeconomic interventions
- just-in-MEV-time LP pools
- 0-capital, 0-credit trading
- “hedge” exact end-of-block costs perfectly
- ….and many more!

Appendix

{Ante bellum}