SUAVE smart contract programming model: TEE-based smart contracts for block building

Explaining SUAVE to Comp. Sec. researchers in 10 minutes

Caveats

The following explanation gets many details wrong or oversimplified. Some over simplification is deliberate intended to dodge rabbit holes and save time.

I am mainly inhabiting a very simplified mental model where there’s nothing about the blockchain except for uniswap v2 trades. I think this is sufficient to understand the main phenomena.

Plenty of this is conjecture, personal interpretation of everyone else’s notion drafts, etc.!

First let’s talk about goals&policy, separate from how to do it

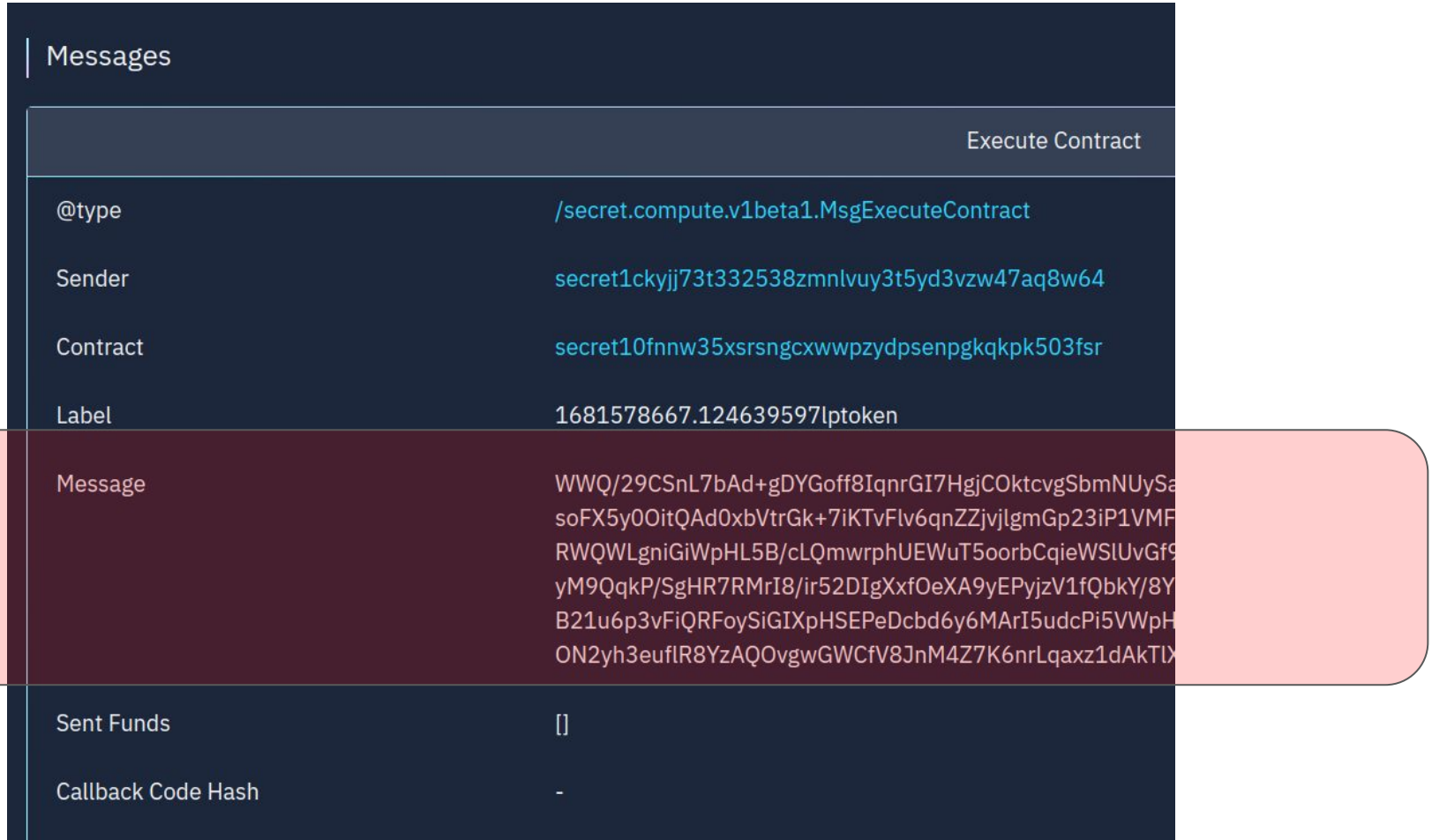
“Flashbots just wants to extract MEV from users.”

No. Flashbots wants to (and already does) the following:

1. Prevent front running

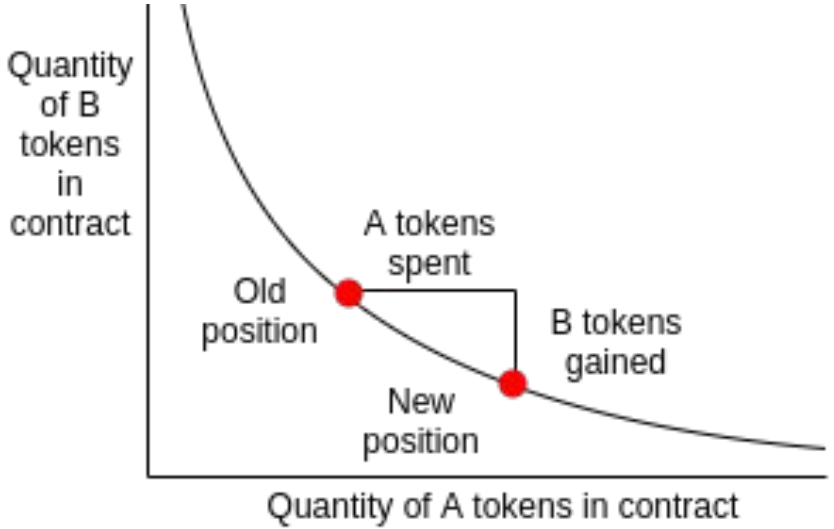
2. Make back running (arbitrage) benefit ***users*** rather than just searchers, without bogging down the network.

Secret Network is a bit like SUAVE



In TEE-based smart contracts, we already have encrypted mempools, and therefore fair ordering.

However, the remaining arbitrage opportunity cannot be captured by users, must be mitigated



SiennaSwap

completion. Furthermore, due to the utilization of the Secret Network and its Secret Contracts, the transactional information for transactions occurring on Sienna Network are private - eliminating the merest possibility of front-running with the purpose of shorting the market on Sienna Network.

Unsophisticated trades create mess *someone* must clean up

In a world with Fair Ordering only (e.g., Secret today) the following would happen:

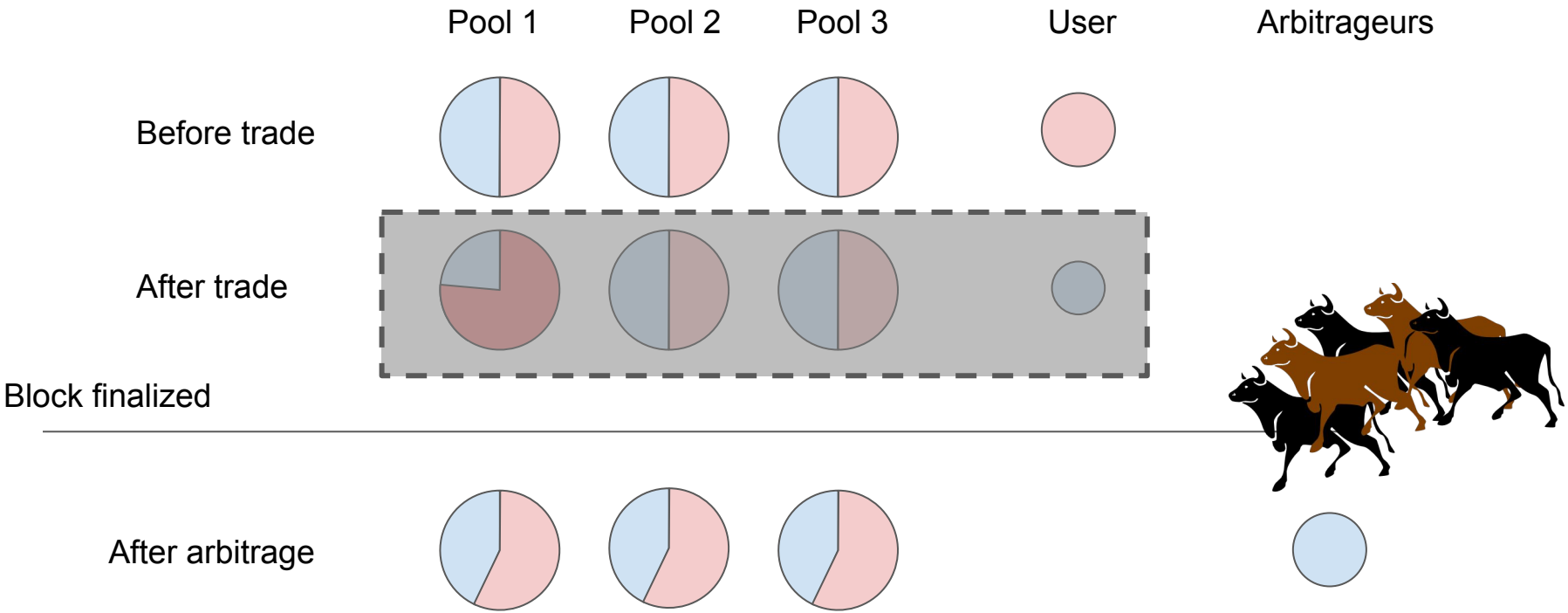
Unsophisticated trades create arbitrage opportunities.

These are only revealed to arbitrageurs in the next block.

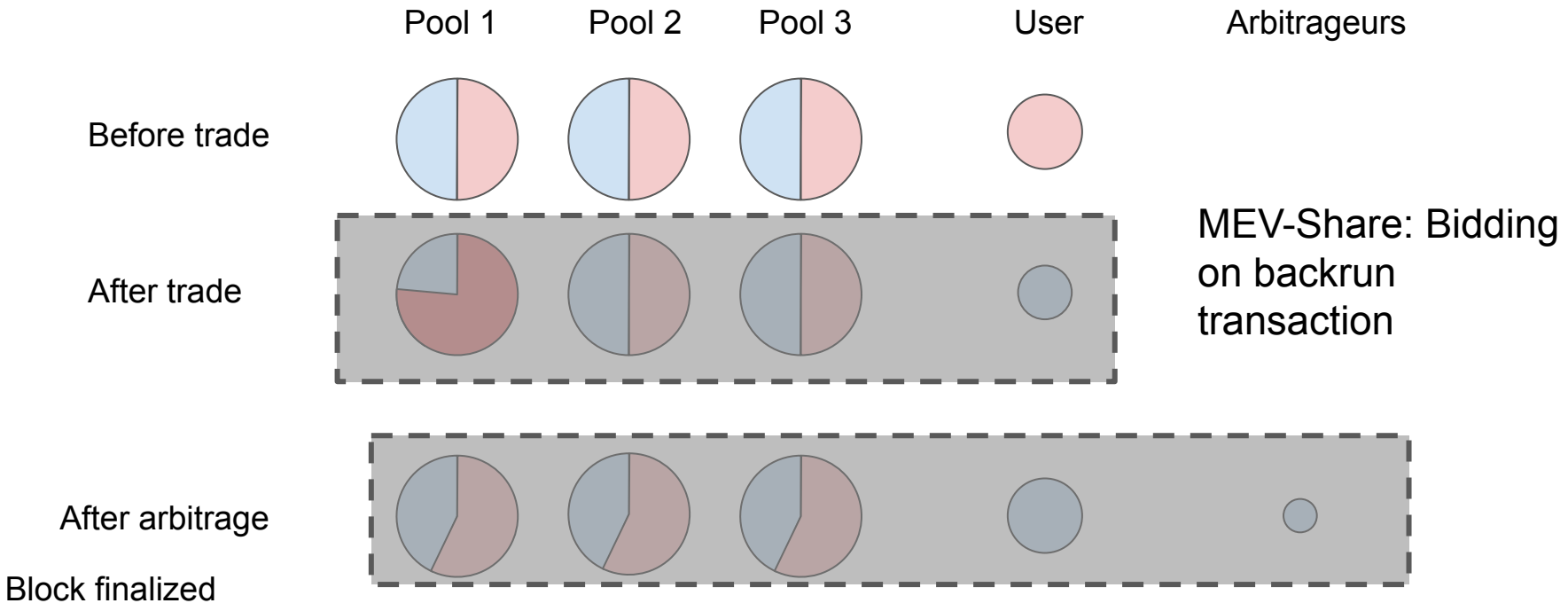
Users capture zero of the benefit when the arbitrage eventually occurs.

A PGA battle could break out as arbitrageurs fight over top of block position.

Unsophisticated trades create mess *someone* must clean up



MEV-Share allows users to outsource their arbitrage



What is SUAVE, compared to what exists today?

Three things:

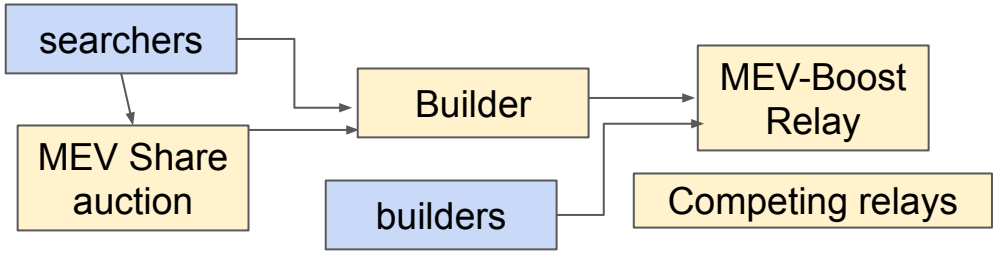
1. Replace trust in the operator with trust in ***TEEs***, and/or cryptography and threshold mpc

2. Make the operation of the service decentralized, geographically and administratively

3. User programmable, based on smart contracts. An open, contestable, marketplace for mechanisms.

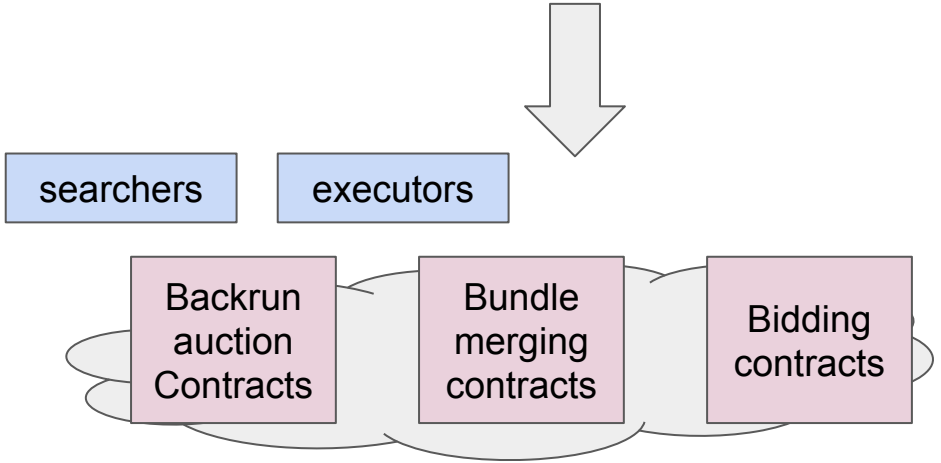
SUAVE: unifying the MEV transaction supply chain

Today's market heavily relies on ***trust*** in service providers



With SUAVE Market for mechanisms based on ***Smart Contracts,*** we have transparency, user empowerment. Competing contracts can fulfill different roles.

Contracts require a ***privacy*** oriented ***programming*** model.



SUAVE's strategy was anticipated by StrategyProof Computing from Ng, Parkes, Seltzer 03.

[A4 open] Systems should be *open* to allow for innovation and competition in the design of new services. This principle applies to the methods of resource allocation and arbitration, just as it applies, for example, to the design of web servers and the design of distributed file caches. A successful open system requires a *lightweight* infrastructure in which we only introduce universal and minimal components, to enable the deployment of services without preventing the future deployment of unanticipated applications.

[A5 decentralized] The control structure in distributed computing systems must be decentralized, both to respect the autonomy of the nodes that own the property rights to resources, but also for reasons of computational scale and timeliness of information.

The SPC infrastructure must provide support for multiple users to design and deploy competing LSP mechanisms, in a ***market for mechanisms.*** With this, we achieve the second set of design principles of *open* and *decentralized* systems. Our belief is that an open marketplace will naturally lead to mechanisms with the "right scope" and the "right complexity". This decision represents a tradeoff between providing a large enough scope to sufficiently simplify the game-theoretic decision facing a participant—for example, bringing resources that are *complementary* for a large number of users into the same scope —while maintaining a small enough scope to build computationally reasonable resource allocation mechanisms. The degree to which market forces lead to the emergence of mechanisms with the right scope is an important research question.

SUAVE's strategy was anticipated by StrategyProof Computing from Ng, Parkes, Seltzer 03.

SUAVE aims to fulfill this vision of Strategy Proof Computing, in having an open marketplace for mechanisms… even just for these blockspace / orderflow auctions.

Emphasis on "openness".... SUAVE ensures ***market innovation occurs in the open, through published smart contracts***… rather than in private arenas with deep moats.

The SPC infrastructure must provide support for multiple users to design and deploy competing LSP mechanisms, in a ***market for mechanisms.*** With this, we achieve the second set of design principles of *open* and *decentralized* systems. Our belief is that an open marketplace will naturally lead to mechanisms with the "right scope" and the "right complexity". This decision represents a tradeoff between providing a large enough scope to sufficiently simplify the game-theoretic decision facing a participant—for example, bringing resources that are *complementary* for a large number of users into the same scope —while maintaining a small enough scope to build computationally reasonable resource allocation mechanisms. The degree to which market forces lead to the emergence of mechanisms with the right scope is an important research question.

SUAVE design principles: (heavily borrowed from Ng, Parkes, Seltzer):

1. Guide the market towards ***open innovation competition***.
2. Open system design.
3. Decentralized implementation.
4. Programmable privacy.

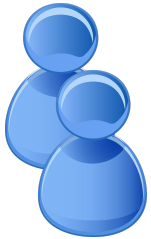
SUAVE as an Ideal Functionality

On-Chain block building in the spec

Market design innovators

define contracts

Users, wallets



bids, private preferences, authorize contracts

hints, leakage, *side channels*

Searchers, solvers, executors

Arbitrage, strategy, solving

SUAVE contracts

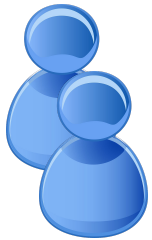
SUAVE contracts

SUAVE contracts

Confidential storage

Blocks to validators

SUAVE as decentralized TEE-based smart contracts

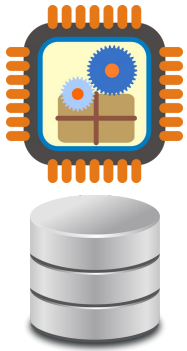


bids, private preferences,
authorize contracts

SUAVE
contracts

L1
Blocks

SUAVE Chain
- Mempool of pending bids
- Committed bids
- Committed smart contracts
- Confidential storage

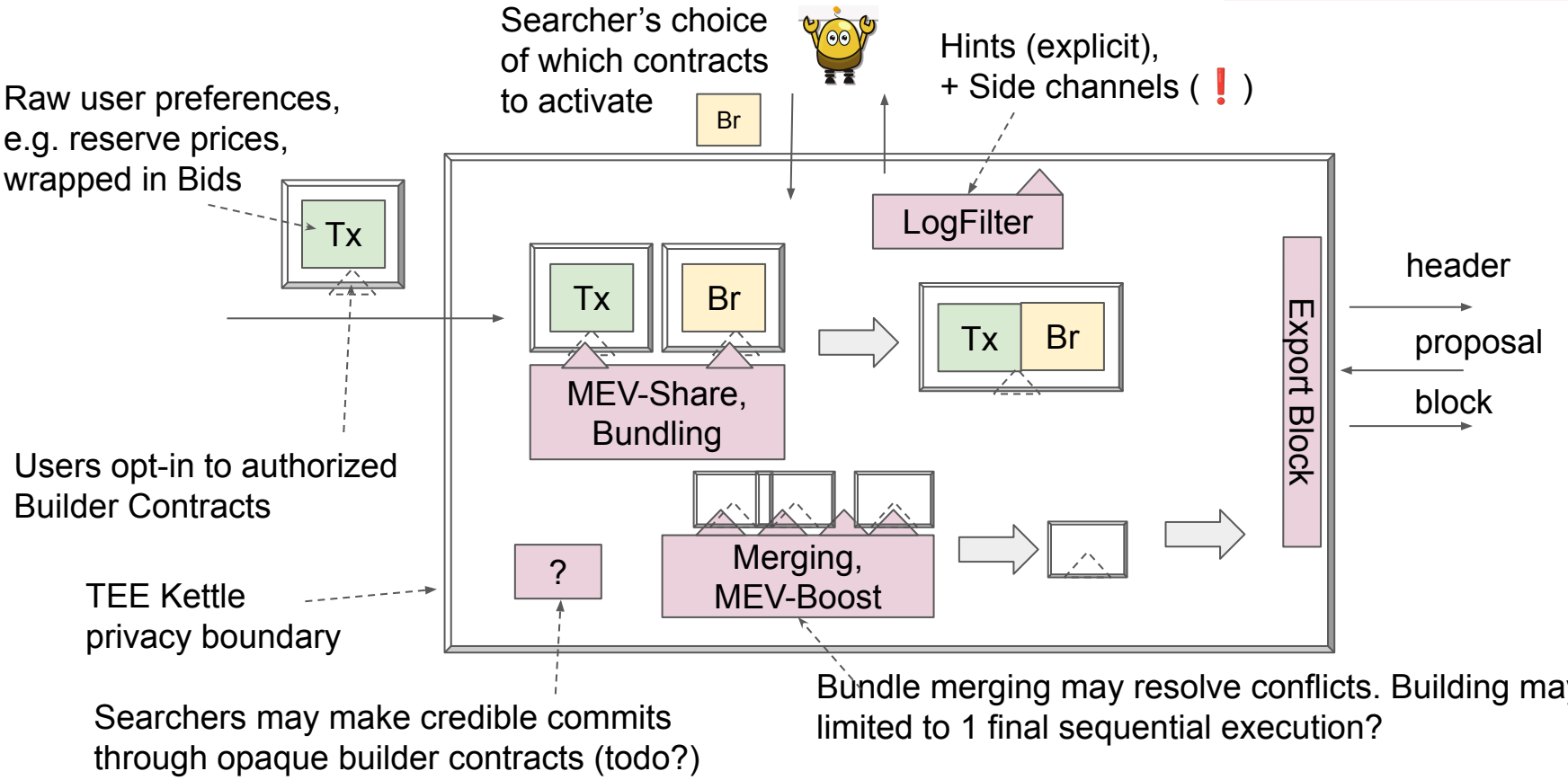


TEE Kettles
- Executors have TEEs
- Able to execute off-chain queries defined by contracts
- Local encrypted storage

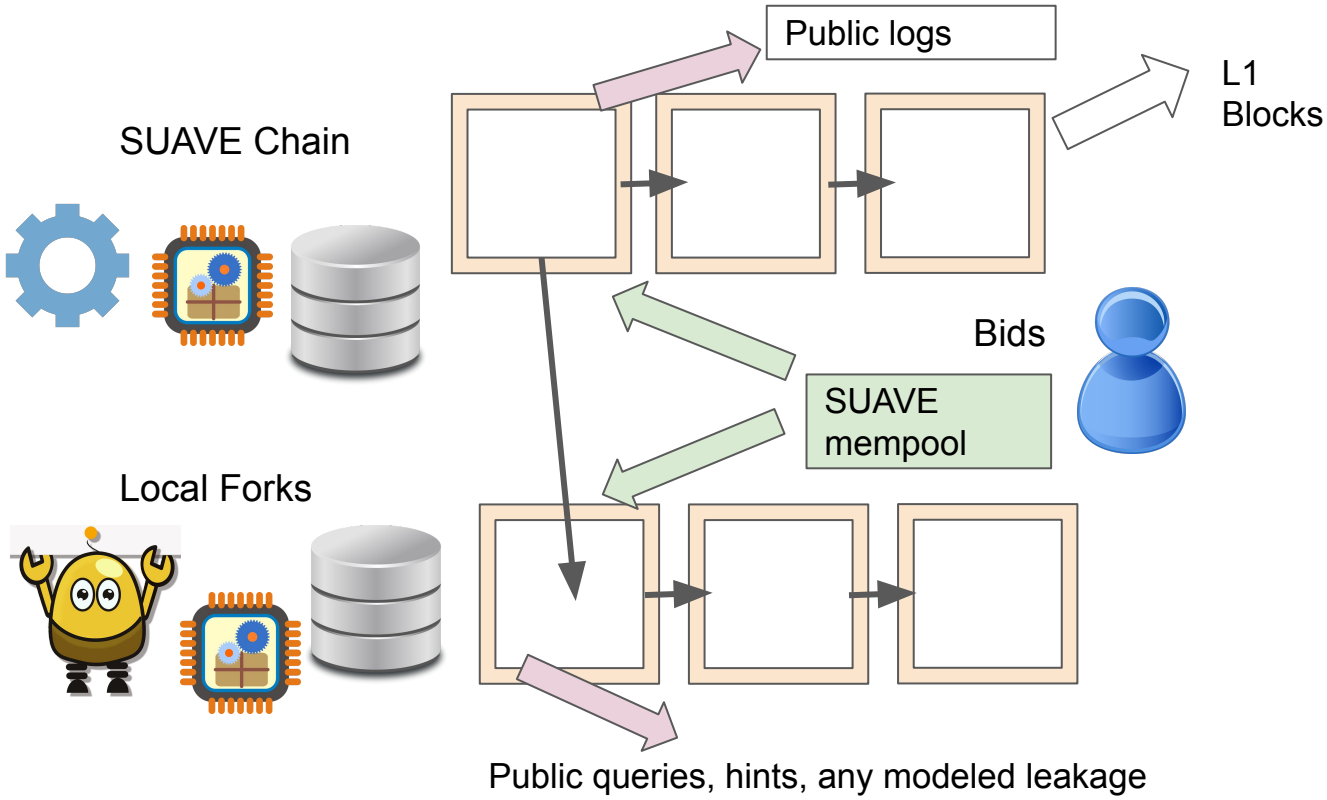
Off-Chain block building in realization

Ecosystem of SUAVE Contracts

Innovators in the mechanism marketplace propose new SUAVE Contracts, compete for user adoption.

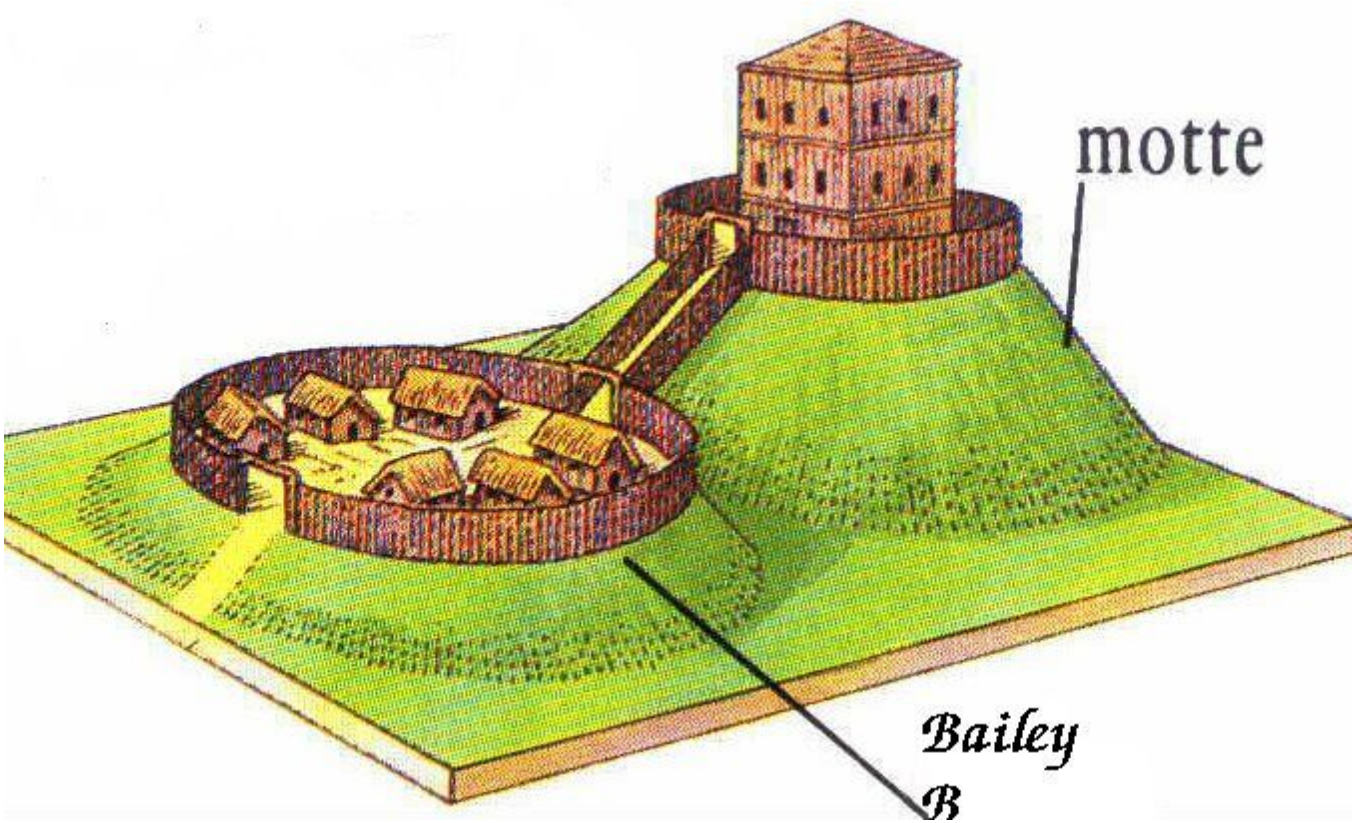


Decentralized Execution through TEE Kettle network



TEE Kettles create local simulated forks, as specified by ***SUAVE contract applications.***

The TEE Deployment Motte & Bailey



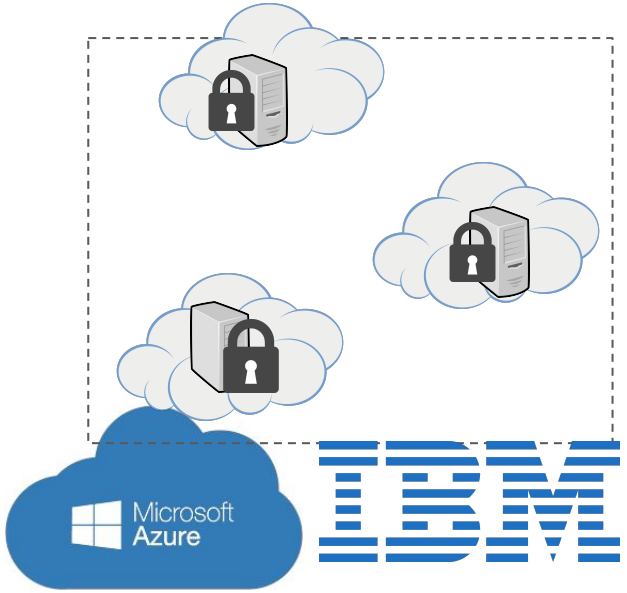
The TEE Deployment Motte & Bailey

Self-hosted or cloud TEEs
++ Convenience
++ Distributed resources

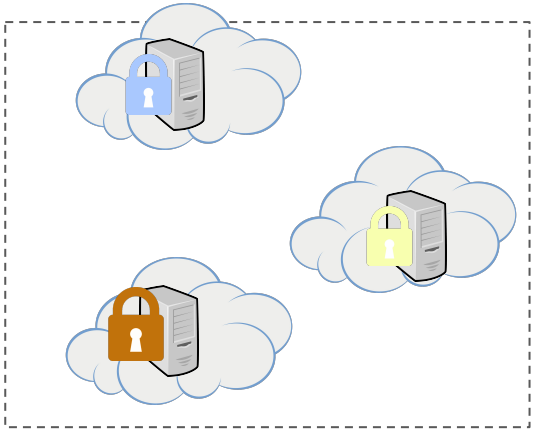
- - Trust model, physical access

Cloud instances (e.g., Azure, IBM, ...),
+ Convenience
+ Physical security, incentive alignment(?)

- Often opaque policies, incentive alignment(?)



MPC / Threshold FHE
+ Best security, must break TEE *AND* multiple hosts
- Cost, not practical yet



SUAVE Programming Examples

Modelling Flashbots Builder as SUAVE contract

```
function mevboost.build() {

    // Pass #1: Estimate EGP of each bundle based on top-of-block sim
    For each bundle in all_bundles();
        st :=L1.topofblock.simulateEVM(bundle)
        EGP[bundle] := st.queryEGP()

    st := L1.topofblock; block := []
    // Pass #2: Revisit in order by Pass#1 estimate
    While blockspace remains, for each bundle by EGP:
        st := st.simulateEVM(bundle)
        EGP' := st.queryEGP()
        If EGP' decreases, or bundle fails: discard
        Otherwise: block.append(bundle)

    exportBlock(block)
}
```

Pass 1: Top of block simulation.
(EGP: Effective Gas Price, a measure of the the MEV obtained)

Pass 2: In-order simulation

Modelling WalruSuave as a SUAVE Contract

```
function submit_bid(uint qty, uint rsv) {
    // Phase #1: On-chain bidding ensures data availability
    //   This could also be passed as signed messages
    bids[msg.sender].push(Bid(qty,rsv));
}

function process_bids() {
    // Phase #2: Use bisection to determine the strike price,
    // Passing the residual demand to the AMM.
    bundle = settle_and_residual(self.bids, self.amm)
    mevboost.submit_bundle(bundle) // Submit onward to an auction
}
```

Modelling MEV-Share as a SUAVE contract

This contract allows searchers to submit a backrun. Bundles are submitted with user defined hints about the bundle.

```
function submit_bundle(Bundle b, hints) public {
    // Pass 1: on ingest… simulated top of block only
    all_bundles[b.ID] := (b, hints)
    emit (hints,hints(b))     // Leak user-configured hints
}

function match_backrun(Bundle b, Tx br, hints) {
    // Pass 2: build a bundle based on this backrun
    b’,hints := flatten(all_bundles[b.id], br, hints)
    all_bundles[b’.ID] := b’; mevboost.send_bundle(b’)
}
```

Modelling MEV-Share as a SUAVE contract

We also need to modify the builder rule, to respect the MEV-share redist policy

```
function mevboost.build_plus_mevshare() {
    // Pass #1: Estimate EGP
    For each bundle in all_mevshare_bundles();
        st :=L1.topofblock.simulateEVM(bundle)
        EGP[bundle] := st.queryEGP()
        bundle.append(Generate Kickback transaction)


    // Pass #2: Revisit in order by Pass#1 estimate
    ...

    exportBlock(block)
}
```

PROF as a SUAVE contract - a simulation bypass

PROF bundles can be added to a block and exported, but they are NOT allowed to be simulated EXCEPT in the very final step of exporting a block.
In other words, this bypasses the EGP simulation step of the sorting builder.

```
function mevboost_plusprof.build() {
    // Ordinary mevboost.build
    …

    // Prof blocks
    While blockspace still remains, for each prof bundle:
        block.append(bundle)

    exportBlock(block)
}
```

TEE Smart contract insights

Rollups for Free using TEEs

If we are using TEEs, we might as well also incorporate their use for attestation. We can get the same functionality as zk SNARKs, assuming the TEE functions correctly.

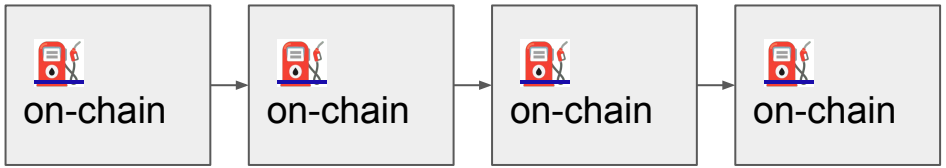
So far, TEE-based smart contracts have NOT made use of this. Instead they have preferred replication. This importantly means that if the TEE is compromised, it does not change the integrity guarantees, i.e. printing tokens.

In the case of the SUAVE auction, what's at stake is short lived confidentiality anyway, so this tradeoff makes sense. ***This is more applicable for SUAVE than for user-privacy in general TEE Smart Contract applications.***

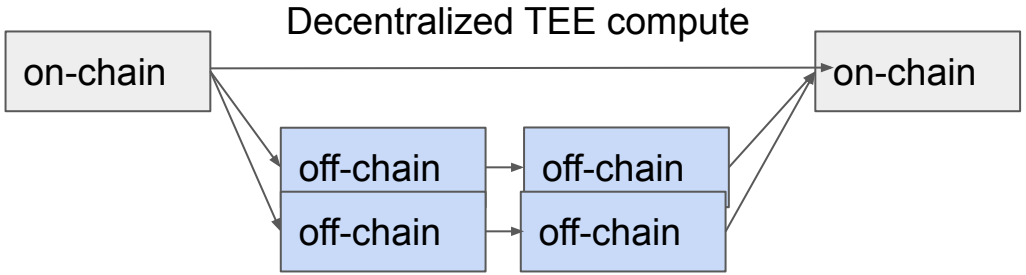
Interesting new programming concept for TEE contracts

Hackin' insight: TEE-based contracts like Secret and Oasis can turn their "view" queries into off-chain programs. The reason is that these views are smart contract-defined, but since they run in TEE they can have private state.

It makes sense to allow them to use encryption and message authentication codes, in order to chain off-chain views together. ***It's a TEE-based rollup.***

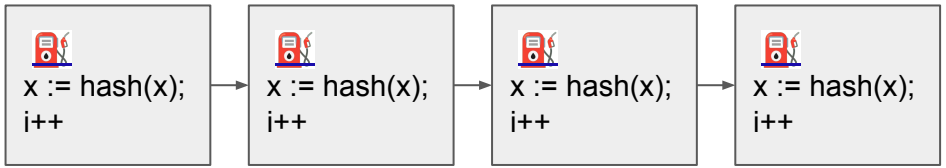


On-chain computing:
slow, gassy

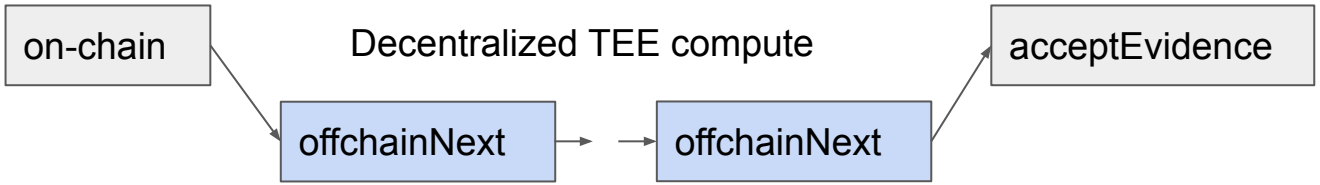


Off-chain computing:
- Makes use of contract-defined ***view*** functions.
- View functions can access TEE-contract MAC and encryption.
- ***Localmutable*** for local indexes

Off-chain computing example




```
contract {
        // Mutable function iterates
        function computeNext() …
}
```




```
contract {
        // Off-chain function TEE kettles compute locally
        function offchainNext(EncState) returns(EncState) view ….
        // On-chain function to accept result after off-chain compute
        function acceptEvidence(EncState);
}
```

Side channels? Make them explicit… *then* mitigate them

In this ideal functionality, the use of **simulateEVM** ***explicitly*** generates an ***access pattern trace***. In fact this trace is quite rich as a starting point, and essentially follows along with the program counter. Practically only ***signatures*** and, say, the ***value of a random 32-byte field*** are protected. In this way the trace is a faithful over-approximation of the actual side channels when run at full speed. It's ***explicit***, so the market ***must*** price it in.

~~Counterintuitive~~: *Troll thought: this is side channel accelerationism*.

Next, we can aim to remove side channels for specific recognized applications, through ORAM or by computing on an abstracted form of the calldata (see mpc backrunning)

Open Questions

- Mechanism design and analysis. This has been describing a platform, but not at all making progress on the “Locally Strategy Proof” regime of compositional mechanism analysis. Start with MEV-share+boost, which is very concrete?

Conjecture: MEV-share outperforms PROF for every participating user

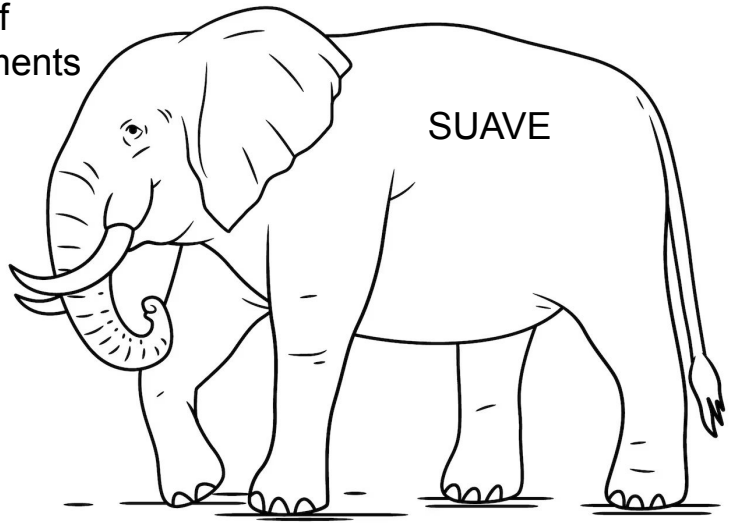
- Security analysis. This has been a sketch of an ideal functionality model. Can we instantiate it from an Ideal Functionality of TEE-based smart contracts?

- Builder contracts programming model. Haven’t finished these examples yet, but already the precompiles design questions are interesting. Modular designs for building rules? Off-chain programming this needs more abstraction.

- **Hybrid operation:** interoperate with cryptography alternatives to TEE

Block building using a smart contracts programming model

Mechanism design guided by logic of credible commitments



Public innovation marketplace

Distributed Execution based on TEEs and cryptography

Pragmatic generalization of MEV-Boost, MEV-Share