



High-Assurance ZK

Building foundations you can trust



Mamy Ratsimbazafy

ZK Engineering @ Taiko



mratsim



m_ratsim

A quick technical presentation of Taiko

- ❖ **Type 1 zkEVM (Ethereum-equivalent)** → no changes to hashes, state trees, transaction trees, precompiles or any other in-consensus logic. Full implementation of the Ethereum (execution layer) yellow paper specifications
- ❖ **Based rollup** → block sequencing driven by L1 validators, thereby inheriting Ethereum's level of decentralization
- ❖ **Permissionless** → proposers & provers can join or leave the network at any time, thereby maximizing censorship resistance
- ❖ **Deterministic block execution** → finality is achieved immediately after a block is proposed, i.e. all block properties are immutable from that point on. Thereby Taiko L2 transactions are finalized after only a single L1 confirmation

High-Assurance software



High-Assurance?

ISO 15408 - Common Criteria for Information Technology Security Evaluation
(1999)

EAL: Evaluation Assurance Level

- EAL1: Functionally Tested
- EAL2: Structurally Tested
- EAL3: Methodically Tested and Checked
- EAL4: Methodically Designed, Tested and Reviewed
- EAL5: Semiformally Designed and Tested
- EAL6: Semiformally Verified Design and Tested
- EAL7: Formally Verified Design and Tested

High-Assurance?

Software that you can trust to build
a multi-billion dollars ecosystem on

Software failures



What do Ariane 5, Boeing 787 and Gangnam Style have in common?

<https://www.bbc.com/future/article/20150505-the-numbers-that-lead-to-disaster>

r



What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Ariane 5: <https://www-users.cse.umn.edu/~arnold/disasters/ariane5rep.html>

“The internal SRI [Inertial Reference System] software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value.”

What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Boeing 787



Fiora

@FioraAeterna

...

248 days == 2^{31} 100ths of a second.

even in 2015, our airplanes have integer overflow bugs



Ben Goldacre @bengoldacre · May 1, 2015

If you leave your Boeing 787 switched on for 248 days the power shuts off and you fall out of the sky. Epic bug. theguardian.com/business/2015/...

2:06 PM · May 1, 2015

960 Reposts 8 Quotes 486 Likes 2 Bookmarks

What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Psy - Gangnam Style

In 2014, Gangnam Style exceeded Youtube view limits of 2,147,483,647

What do Ariane 5, Boeing 787 and Gangnam Style have in common?

Psy - Gangnam Style

In 2014, Gangnam Style exceeded Youtube view limits of 2,147,483,647

Radiation therapy gone wrong

The Therac-25:

<https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/>

Between 1983 and 1987, 6 patients died from radiation overdose.

“With the technician running the machine, the two were able to pinpoint the issue. The VT-100 console used to enter Therac-25 prescriptions allowed cursor movement via cursor up and down keys. If the user selected X-ray mode, the machine would begin setting up the machine for high-powered X-rays. This process took about 8 seconds. If the user switched to Electron mode within those 8 seconds, the turntable would not switch over to the correct position, leaving the turntable in an unknown state.”

Shuffling Ethereum validators

Code

4

Commits

35

Issues

72

Wikis

0

States

65

Closed

8

Open

8

Advanced search

Cheat sheet

73 issues in [ethereum/eth2.0-specs](#)

Sort: Best match

🚩

Use custom types in data structures

#667

Opened by NashatYrev 9 days ago • 4 comments

🐛

fix committee assignment bugs

#699

2 bugs: - We were modifying current `shuffling_epoch` and then calling `get_current_epoch_committee_count` expecting that it was still getting the committee count from the previous value of ...
Opened by djrtwo 2 days ago • 5 comments

🐛

[WIP] Break up `crosslink_committees_at_slot`

#707

should we add an assert to check `shuffling_epoch` out of bound condition? assert `get_previous_epoch(state) <= shuffling_epoch <= get_current_epoch(state)+1`
Opened by djrtwo 7 hours ago • 8 comments

🚩

committee shufflings take at least 2 epochs to change

#436

... validating and not all shards are represented in each `shuffling`. - This slows down the max rate of activations and exits by a factor of 2.
Opened by djrtwo on Jan 14 • 3 comments

🚩

Light client proposal

#459

... , maintained similarly to the randao roots - To compute the seed used in `get_shuffling`, use seed = `sha3(get_randao_mix(slot), get_active_index_root(slot))` where `get_active_index_root` is defined ...
Opened by vbuterin on Jan 17 • 3 comments

🚩

get_crosslink_committees reads previous_seed during genesis_epoch

#639

Problem Summary During `GENESIS_EPOCH`, `get_crosslink_committees_at_slot(...)` reads state.previous_shuffling_epoch when epoch == current_epoch. Detail `get_crosslink_committees_at_slot(...)` first ...
Opened by paulhauner 13 days ago

🐛

Fix out-of-bounds in `get_shuffling`

#641

What Change `get_shuffling(...)` so it gives shuffles using an index of `active_validator_indices` instead of a value of `active_validator_indices`. Why The following fails with a `IndexError: list index ...`

🚩

Remove Record suffix

#434

no, `get_shuffling` just returns the `shuffling` split across slots, `get_shard_committees_at_slot` returns an array of arrays of (validators, shard) tuples. Arguably, the naming and return value of these ...
Opened by silfoc on Jan 13 • 24 comments

🚩

Keep latest 2^n RANDAO mixes in the state

#295

Also ensures that we have the seed of the current `shuffling` in state. Right now, we discard the current `shuffling` seed immediately after performing the `shuffling`. Not very friendly for reconstructing the `shuffling` outside the specific state transition.
Opened by JustinDrake on Dec 12, 2018 • 4 comments

🚩

"proposer_slots" -> "proposer_nonce"

#493

Opened by paulhauner on Jan 24 • 1 comment

🚩

RFC: drop `shard_and_committee_for_slots` from state

#191

Opened by ametheduck on Nov 29, 2018 • 4 comments

🚩

Understanding the `get_new_shuffling()` function

#127

For the past several weeks, as part of my hacktenship, I have been trying to understand the `get_new_shuffling()` function. In its current iteration, the `get_new_shuffling()` function is in charge of ...
Opened by Mikerah on Nov 14, 2018 • 3 comments

🚩

initial assignment of 'state.persistent_committees'

#239

Issue We are currently assigning state.persistent_committees as a `shuffling` of ValidatorRecords rather than just `validator_indices` persistent_committees=split(shuffle(initial_validator_registry ...
Opened by djrtwo on Dec 5, 2018 • 5 comments

🚩

Remove MIN_VALIDATOR_REGISTRY_CHANGE_INTERVAL

#340

... in times of non-finality (attacks, short range forks, etc), I worry we end up making the `shuffling` extremely subjective and ultimately an attack vector. (for example: easier to construct reasonable looking blocks when the proposer for that slot is not entirely certain)
Opened by vbuterin on Dec 19, 2018 • 5 comments

🚩

assertion in `'get_active_index_root'` too strong

#515

Issue In validator registry and `shuffling` seed data we set `state.current_calculation_epoch = next_epoch` and then do `state.current_epoch_seed = generate_seed(state, state.current_calculation_epoch ...`
Opened by djrtwo on Jan 29

🚩

Delay exits with penalty

#350

Delaying exits with penalty by 1+epsilon epochs ensures that self-slashing single validators does not change the `shuffling` for the next epoch and so cannot (normally) be used as a way of manipulating the `shuffling`.
Opened by vbuterin on Dec 21, 2018 • 8 comments

🚩

Introduce swap-or-not shuffle

#576

See #563 for discussion. Here is a more efficient implementation for `shuffling` an entire set; it can live here until we come up with an explicit "efficient implementation" doc: `def shuffle ...`
Opened by vbuterin 23 days ago • 20 comments

🚩

Mitigating attacks on light clients

#403

... `shuffling`. Note that alternative `shuffling` algos do not fix this problem, because the step of filtering out inactive validators still requires a pass through the entire validator set. Second, it is ...
Opened by vbuterin on Jan 7 • 8 comments

🚩

helpers and notes for `shuffling` lookahead

#520

beacon chain spec changes: - update `get_crosslink_committees_at_slot` to be able to get potential committees for slots from the next epoch. add `registry_change` param to get next epoch committees ...
Opened by djrtwo on Jan 30 • 7 comments

🚩

Possible alternative numer-theoretic `shuffling` algorithm

#323

Motivation Construct a `shuffling` algorithm where you can compute the value in the shuffled list at any specific position relatively cheaply without computing all of the other values at the same time ...
Opened by vbuterin on Dec 14, 2018 • 13 comments

🚩

non-determinism in `shuffling` from 'SEED_LOOKAHEAD'

#405

Issue shufflings are calculated using a seed from `SEED_LOOKAHEAD` slots ago `get_shuffling(state.latest_randao_mixes[(state.slot - SEED_LOOKAHEAD) % LATEST_RANDAO_MIXES_LENGTH ...`
Opened by djrtwo on Jan 7 • 3 comments

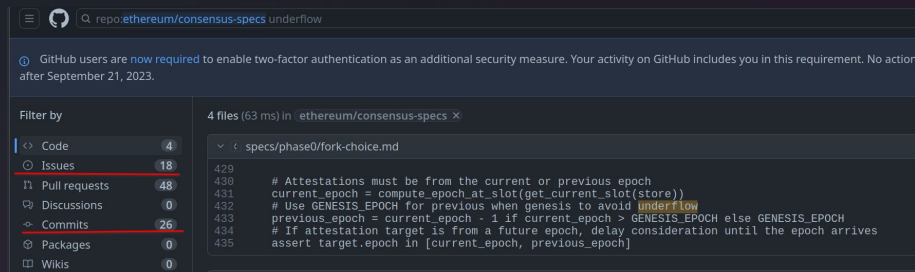
What could go wrong?

A Ethereum 2 spec bug that plagued us, enforcing the invariant:

`assert x <= y - z`

Often x and y are epoch, x represents a justified, finalized or RANDAO epoch, y is the current epoch. z is often 1.

And on Genesis (epoch 0) the assertion which should have caught a bug (signed $0 \leq -1$) doesn't because of unsigned underflow ($0 \leq 2^{64}$)



What could go wrong?

We have seen:

- Boundary / edge conditions (overflows and underflows)
- State machine bugs
- Design bugs

There are many more classes of bugs

And smart-contracts introduce even more (<https://rekt.news/>)

The stakes



What's at stake?

- Your reputation
- Your (hopefully) billion-dollar TVL
- Your future revenue (from fees)

...

Your sleep

Catching the bugs



Testing

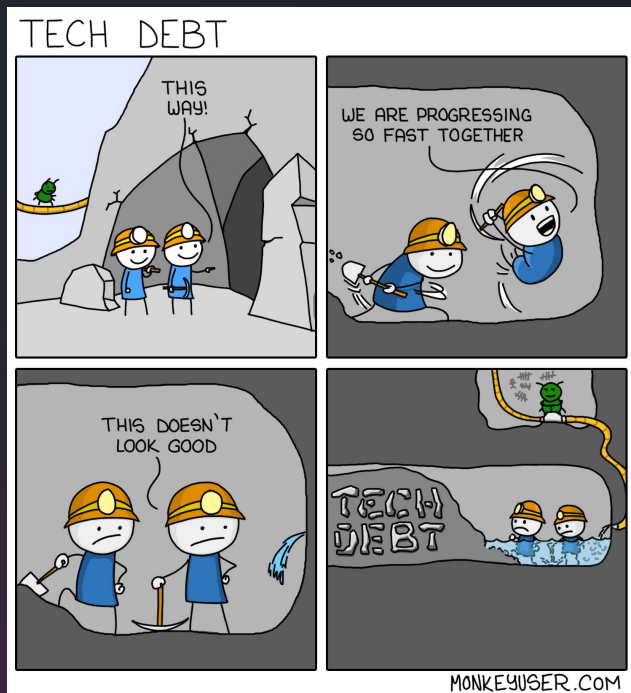
Are there projects still skipping on tests?

Wealth of resources on testing:

- Popularized by test-driven development
- Test coverage tools (beware “You become what you measure”)
- Unit testing and integration testing
- Negative and positive tests
- Property-based testing
- Anti-regression tests
- Continuous Integration

Testing

Are there projects still skipping on tests?



Auditing

Book your auditors early



Fuzzing

Charles Darwin meets software engineering

- Random population of inputs
- Can your software accommodate them?
- Mutate the population
- Can your software accommodate them?
-
- Crash your software with unlikely very specialized inputs

Fuzzing

- Blackbox fuzzing
- Whitebox fuzzing
- Greybox fuzzing / coverage-guided

Symbolic Execution

“In computer science, symbolic execution (also symbolic evaluation or symbex) is a means of analyzing a program to determine what inputs cause each part of a program to execute. An interpreter follows the program, assuming symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would.

It thus arrives at expressions in terms of those symbols for expressions and variables in the program, and constraints in terms of those symbols for the possible outcomes of each conditional branch.

Finally, the possible inputs that trigger a branch can be determined by solving the constraints.”

Consider the program below, which reads in a value and fails if the input is 6.

```
1  int f() {  
2      ...  
3      y = read();  
4      z = y * 2;  
5      if (z == 12) {  
6          fail();  
7      } else {  
8          printf("OK");  
9      }  
10 }
```


Formal verification

Model-checking

- Model checking is brute-forcing all possible states of your are of interest and ensuring that all pre-conditions, post-conditions and invariants are maintained.
- Necessary even for supposedly “safe” language, example <https://github.com/tokio-rs/loom>
To ensure that concurrent data structures have no race conditions (which are different from data races that the Rust borrow checker addresses)
- Can spot design bugs

Formal verification

Deductive verification

- Armed with a proof assistant, you provide proofs, for example of your input ranges or array bounds, when the assistant cannot do the proof itself.

Formal verification

Correct-by-construction

- Using a formally proven compiler, you generate your code from a formally proven design.
You have no bugs.

Example:

- <https://github.com/mit-plv/fiat-crypto>
- <https://github.com/hacl-star/hacl-star>

What's next for ZK?

- Fuzzing (we're working on zkEVM fuzzing at Taiko!)
- Using symbolic execution, SAT or SMT solver to narrow down on edge cases that humans (including auditors) can miss.

Pro Tip: Auditors always start with fuzzing if it's possible. Saves a lot of manpower to find buggy areas.



High-Assurance ZK

Building foundations you can trust



Mamy Ratsimbazafy

ZK Engineering @ Taiko



mratsim



m_ratsim