- UPDATES -

Rank-1 Constraint System and consequences

- Building block of zkSNARK: R1CS circuits
- Recall: Is a sequence of groups of three vectors

$$\Big((a_1, b_1, c_1), (a_2, b_2, c_2), \ldots, (a_m, b_m, c_m) \Big)$$

whose solution is a vector s s.t. $(a \cdot s) \cdot (b \cdot s) - (c \cdot s) = 0.$

A, B, C matrices

Keeps track of the values that each variable assumes during the computation.

Binds the relationships among these variables.

Rank-1 Constraint System and consequences

- Common programming languages do not transform high-level code into R1CS

Circuits

* State-free
* Non-uniform
* Non-deterministic

Rank-1 Constraint System and consequences

- Common programming languages do not transform high-level code into R1CS

Circuits

* State-free
* Non-uniform
* Non-deterministic

High-level languages

* Stateful
* Uniform

Rank-1 Constraint System and consequences

- Consequences:
  - No dynamic arrays
  - No while loops
  - If-else statement of the form: field y = if $x + 2 == 3$ then 1 else 5 fi
  - Number of iterations must be hard coded in the circuit!

Rank-1 Constraint System and consequences

- Consequences:
  - No dynamic arrays
  - No while loops
  - If-else statement of the form: field y = if $x + 2 == 3$ then 1 else 5 fi
  - Number of iterations must be hard coded in the circuit!
- Specifically designed languages are needed: ZoKrates
- Libraries for python and C: PySNARK, Pequin

Why ZoKrates?

- Language designed to be efficiently compiled into R1CS

~ 30.000 lines of code

3 years old

39 contributors

Why ZoKrates?

- Language designed to be efficiently compiled into R1CS

~ 30.000 lines of code

3 years old

39 contributors

- Good output properties:
  - Small output (efficiency due to optimisations of the R1CS)
  - Accurate output (correctness)

Why ZoKrates?

- Language designed to be efficiently compiled into R1CS: ~ 30.000 lines of code, 3 years old, 39 contributors
- Good output properties:
  - Small output (efficiency due to optimisations of the R1CS)
  - Accurate output (correctness)
- Supports finite fields and conversions between fields and other data types
- Toxic waste management (trusted setup)
- Allows to keep specified variables private

Ethereum block header verification in ZoKrates

- Verification uses Ethash algorithm that includes:
    - Keccak512 hash function

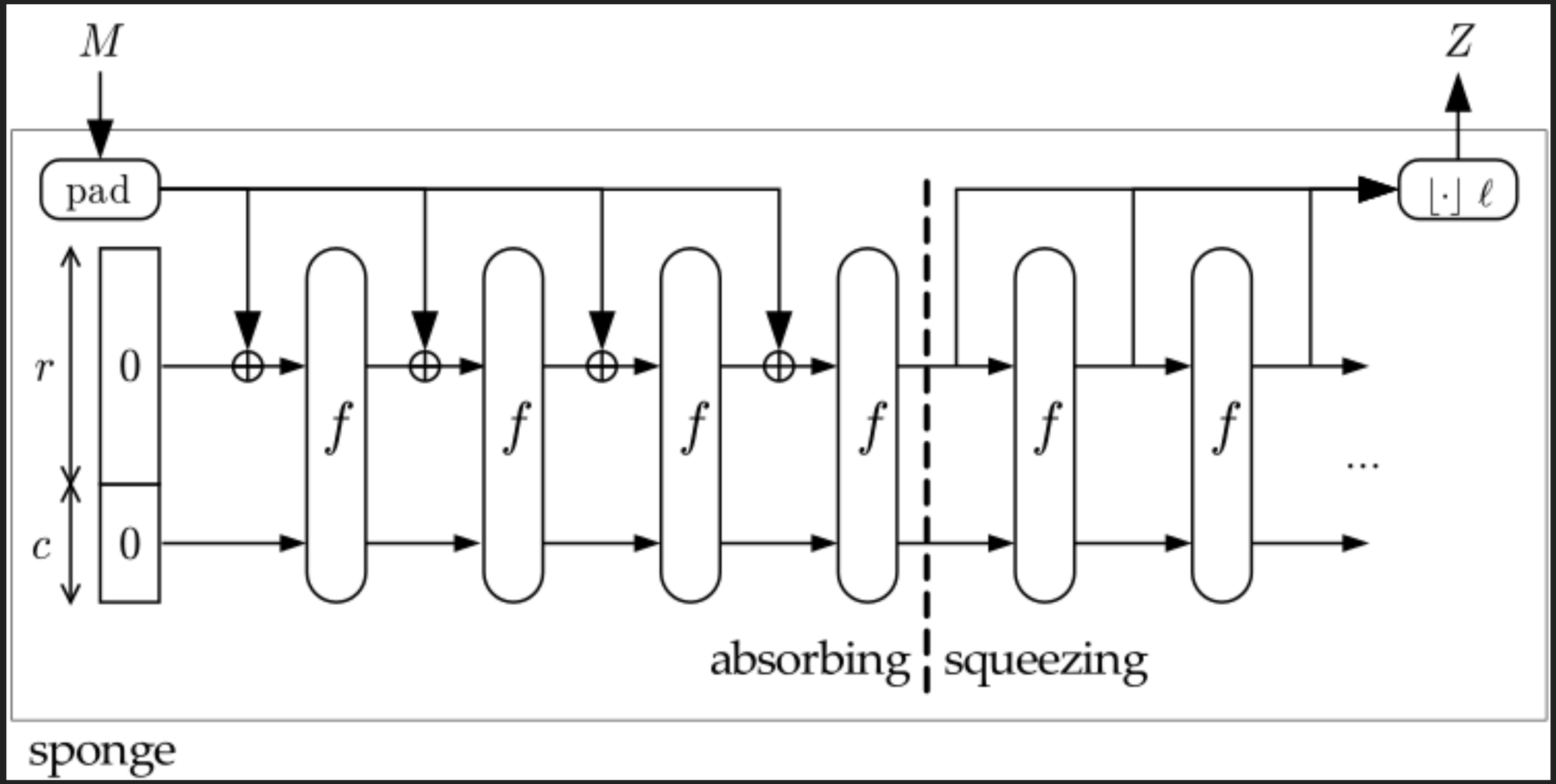
Ethereum block header verification in ZoKrates

- Verification uses Ethash algorithm that includes:
  - Keccak512 hash function
  - Keccak256 hash function

Ethereum block header verification in ZoKrates

- Verification uses Ethash algorithm that includes:
  - Keccak512 hash function
  - Keccak256 hash function
- ZoKrates does not have Keccak hash functions by default (yet)

Keccak hash function construction



Keccak function in ZoKrates

✓ I took the SmartPool solidity implementation as an example

✓ I used booleans as data type (unnecessary casting avoided)

✓ Keccak_f permutation successfully implemented

✓ Sponge construction successfully implemented

✓ 9.5 GB RAM, 5 min compilation time (only for keccak512!)

➡ SmartPool uses a non-standard implementation of the sponge construction .-.

Keccak function in ZoKrates

✓ I took the SmartPool solidity implementation as an example

✓ I used booleans as data type (unnecessary casting avoided)

✓ Keccak_f permutation successfully implemented

✓ Sponge construction successfully implemented

✓ 9.5 GB RAM, 5 min compilation time (only for keccak512!)

➡ SmartPool uses a non-standard implementation of the sponge construction .-.

Weird input, no padding, no bitrate and capacity…What's going on?!?

Keccak function in ZoKrates

✓ I took the SmartPool solidity implementation as an example

✓ I used booleans as data type (unnecessary casting avoided)

✓ Keccak_f permutation successfully implemented

✓ Sponge construction successfully implemented

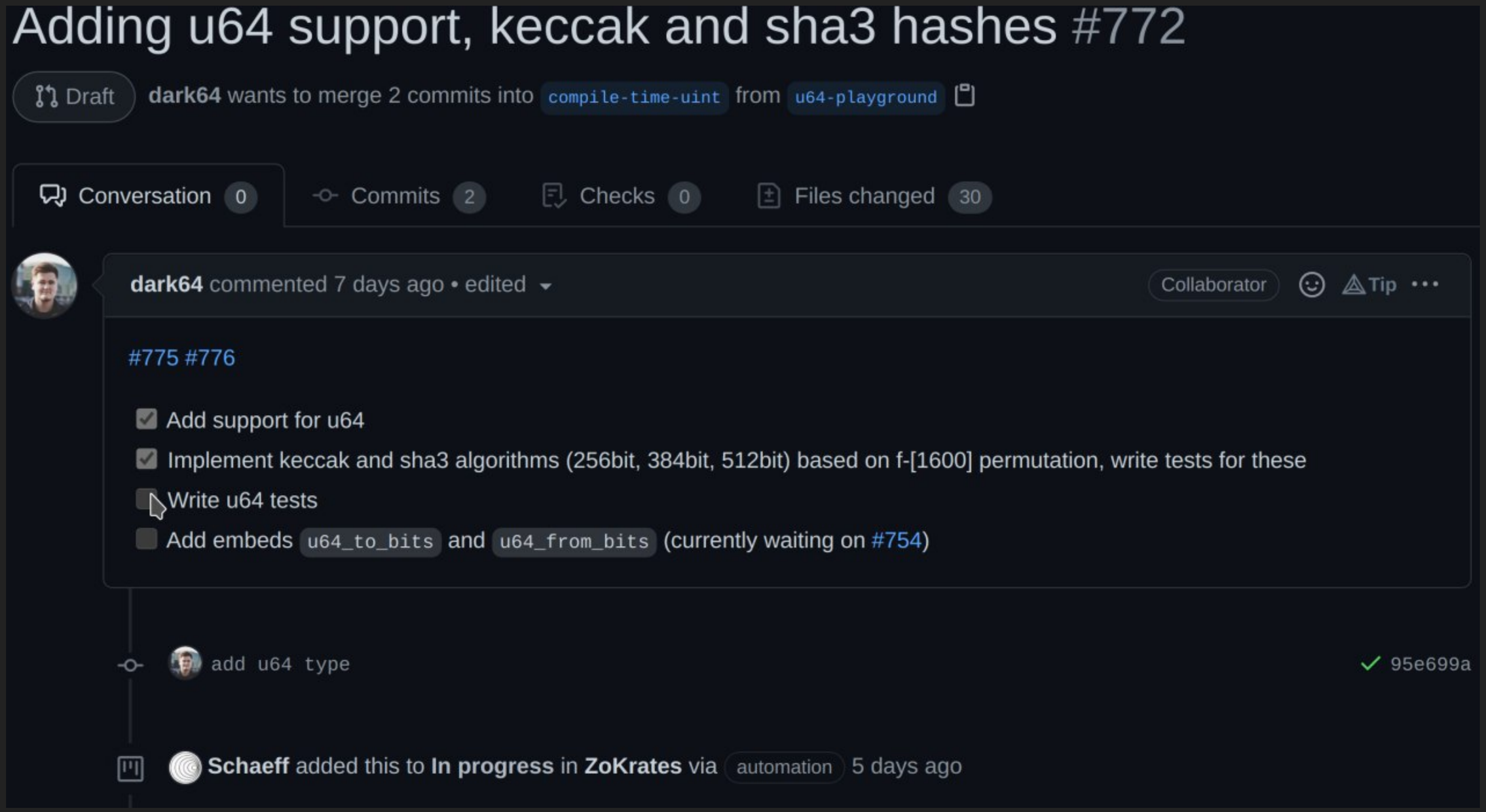
✓ 9.5 GB RAM, 5 min compilation time (only for keccak512!)

➡ SmartPool uses a non-standard implementation of the sponge construction .-.

Weird input, no padding, no bitrate and capacity…What's going on?!?

Construction customised for evaluating Ethash and no inputs of arbitrary size?

A new hope


Adding u64 support, keccak and sha3 hashes #772
Draft
dark64 wants to merge 2 commits into compile-time-uint from u64-playground
Conversation 0
Commits 2
Checks 0
Files changed 30
dark64 commented 7 days ago • edited
Collaborator
Tip
#775 #776
Add support for u64
Implement keccak and sha3 algorithms (256bit, 384bit, 512bit) based on f-[1600] permutation, write tests for these
Write u64 tests
Add embeds u64_to_bits and u64_from_bits (currently waiting on #754)
add u64 type
95e699a
Schaeff added this to In progress in ZoKrates via automation 5 days ago

A new hope

Structure of KECCAK

KECCAK is a family of hash functions that is based on the sponge construction, and hence is a sponge function family. In KECCAK, the underlying function is a permutation chosen in a set of seven KECCAK-f permutations, denoted KECCAK-$f[b]$, where $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ is the width of the permutation. The width of the permutation is also the width of the state in the sponge construction.

The state is organized as an array of 5×5 lanes, each of length $w \in \{1, 2, 4, 8, 16, 32, 64\}$ and $b = 25w$. When implemented on a 64-bit processor, a lane of KECCAK-$f[1600]$ can be represented as a 64-bit CPU word.

We obtain the KECCAK$[r, c]$ sponge function, with parameters capacity c and bitrate r, if we apply the sponge construction to KECCAK-$f[r + c]$ and by applying a specific padding to the message input.

References:

- “Unifying Compilers for SNARKs, SMT, and More”, Ozdemir et al. (2020)
- Zero Knowledge Podcast (YouTube and Spotify): Episode 172: ZK languages with Alex Ozdemir
- https://github.com/Zokrates/ZoKrates
- https://keccak.team/
- https://academy.bit2me.com/en/what-is-the-algorithm-of-ethash-mining/