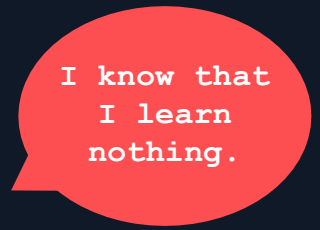
ZoKrates

zkSNARKs for Developers



@schaeff @vanderkriek

Design principles

> High level language

> Compiles to R1CS

> Modular backend implementations

> Heavy use of optimisation

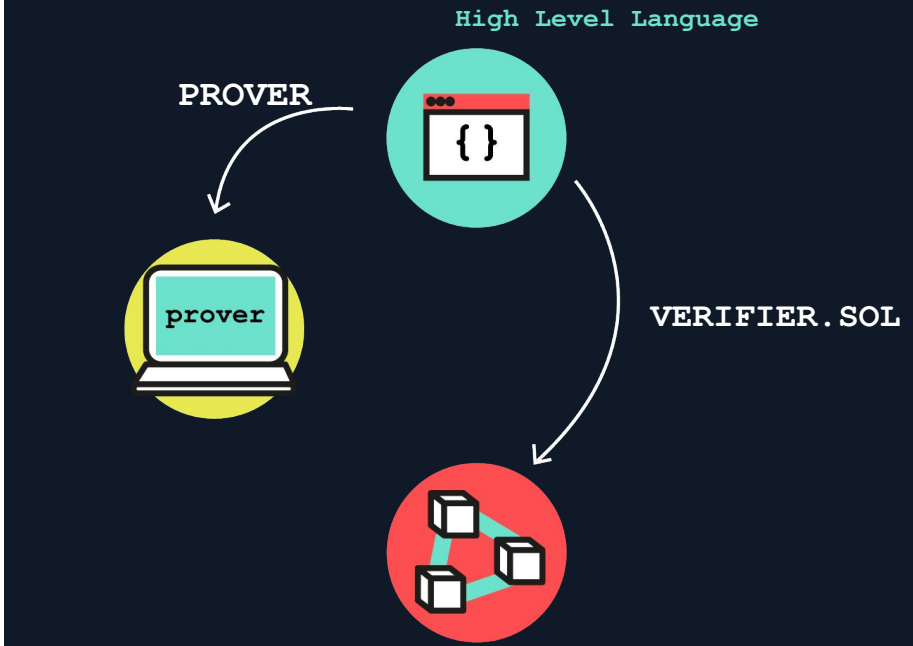
Where does it run?

> Compiler: native, Remix, playground

> Schemes: g16, gm17, marlin, **nova**

> Proving: bellman, ark, bellperson, **snarkjs**

> Verifier: EVM, CLI, js



Hello world

hello.zok

```
def main(private field a, private field b, field c) {
    assert(a*b == c);
}
```

bash

```
> zokrates compile -i hello.zok
> zokrates compute-witness -a 3 3 9
> zokrates generate-proof
> zokrates export-verifier
```

Language features

sha256.zok

```
from "./IVconstants" import IVconstants;
from "./shaRound" import shaRound;

def sha256<K>(u32[K][16] a) -> u32[8] {
    u32[8] current = IVconstants;

    for u32 i in 0..K {
        current = shaRound(a[i], current);
    }

    return current;
}
```

Non-optimal compiler output

sha256round.zok

```
u32 res = (a & b) ^ (a & c) ^ (b & c);

// bc === b * c
// bc - res === (2 * bc - b - c) * a;
```

Compile | user

```
18      bool[32] c_bits = cast(c);
19
20      // create bits of the result
21      bool[32] mut res_bits = [false; 32];
22
23      for u32 i in 0..32 {
24          field a = bool_to_field(a_bits[i]);
25          field b = bool_to_field(b_bits[i]);
26          field c = bool_to_field(c_bits[i]);
27      }
28  }
29
30  def main(u32 a, u32 b, u32 c) -> u32 {
31      return default(a, b, c);
32  }
33
```

Output | Execute | Abi

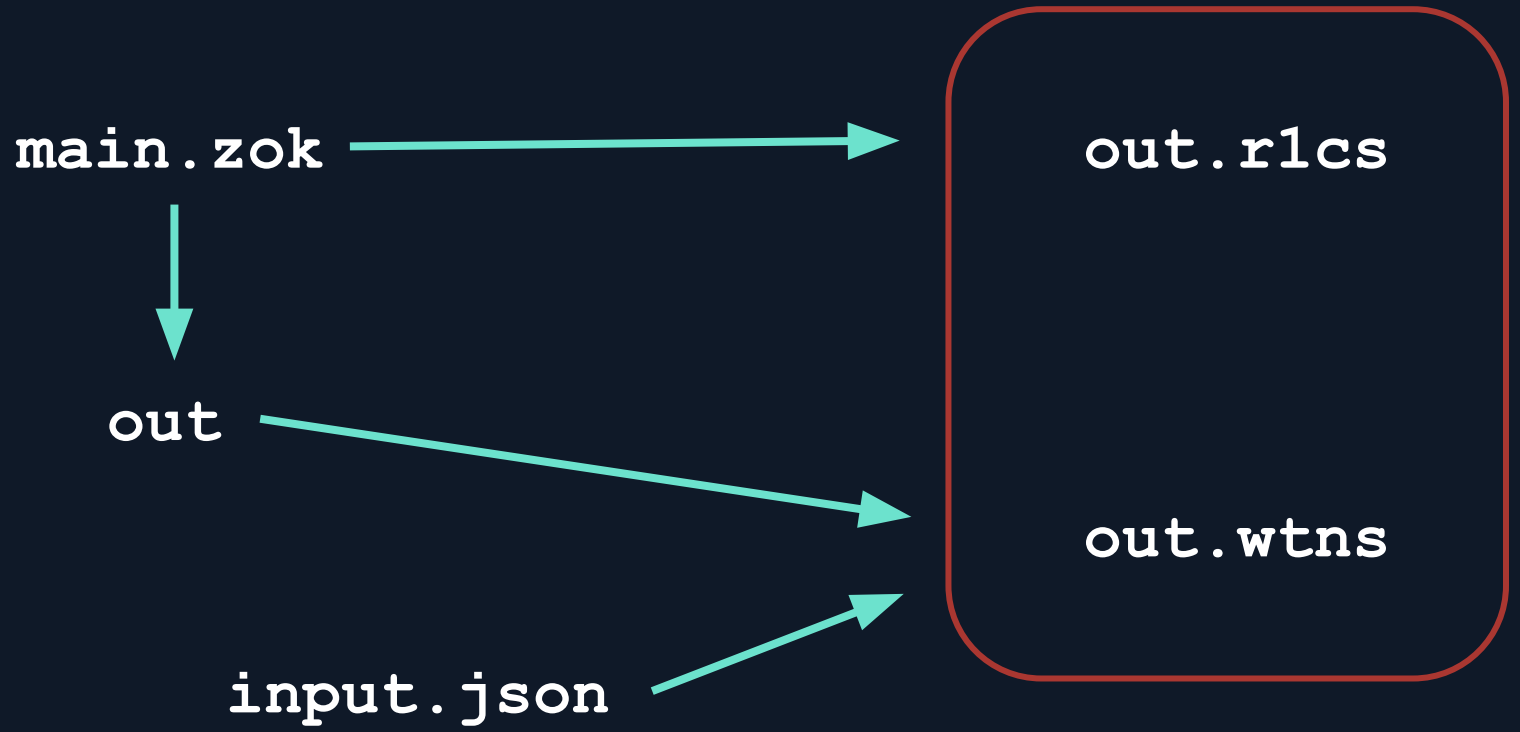
```
[2022-10-08T17:15:49.788Z]
Compilation successful (260
constraints) (took 354.10 ms) ✓
```

Isolating potentially unsafe behavior

> Most ZoKrates code uses guarantees of the compiler

> Performance critical parts in assembly blocks

ZoKrates + snarkjs



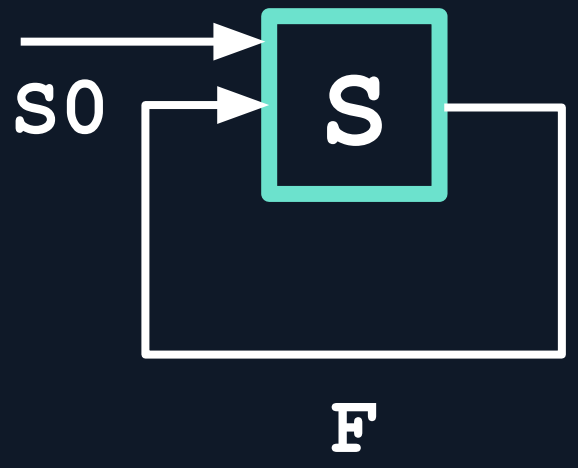
Incrementally verifiable computation

> Start with state S0

> Apply F to find S1

> Repeat!

> Use recursive SNARKs to prove correct execution of the program so far



$$y = F(F(...F(S0)))$$

Some use cases

> Succinctly verifiable blockchains

> VDFs

> and more!

Nova support (soon!)

cube.zok

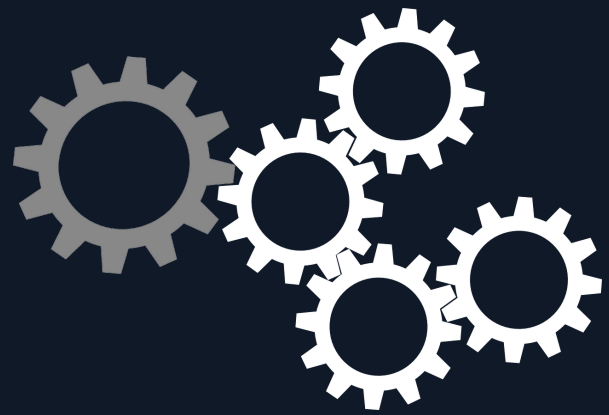
```
def main(field a) -> field {
    return a**3;
}
```

bash

```
> zokrates compile -i cube.zok --curve pallas
> zokrates nova prove --init --steps 3 -a 42
```

In other news

- Powers of tau
- Log statements
- Marlin
- New syntax
- play.zokrat.es



zokrates.github.io

gitter.im/ZoKrates

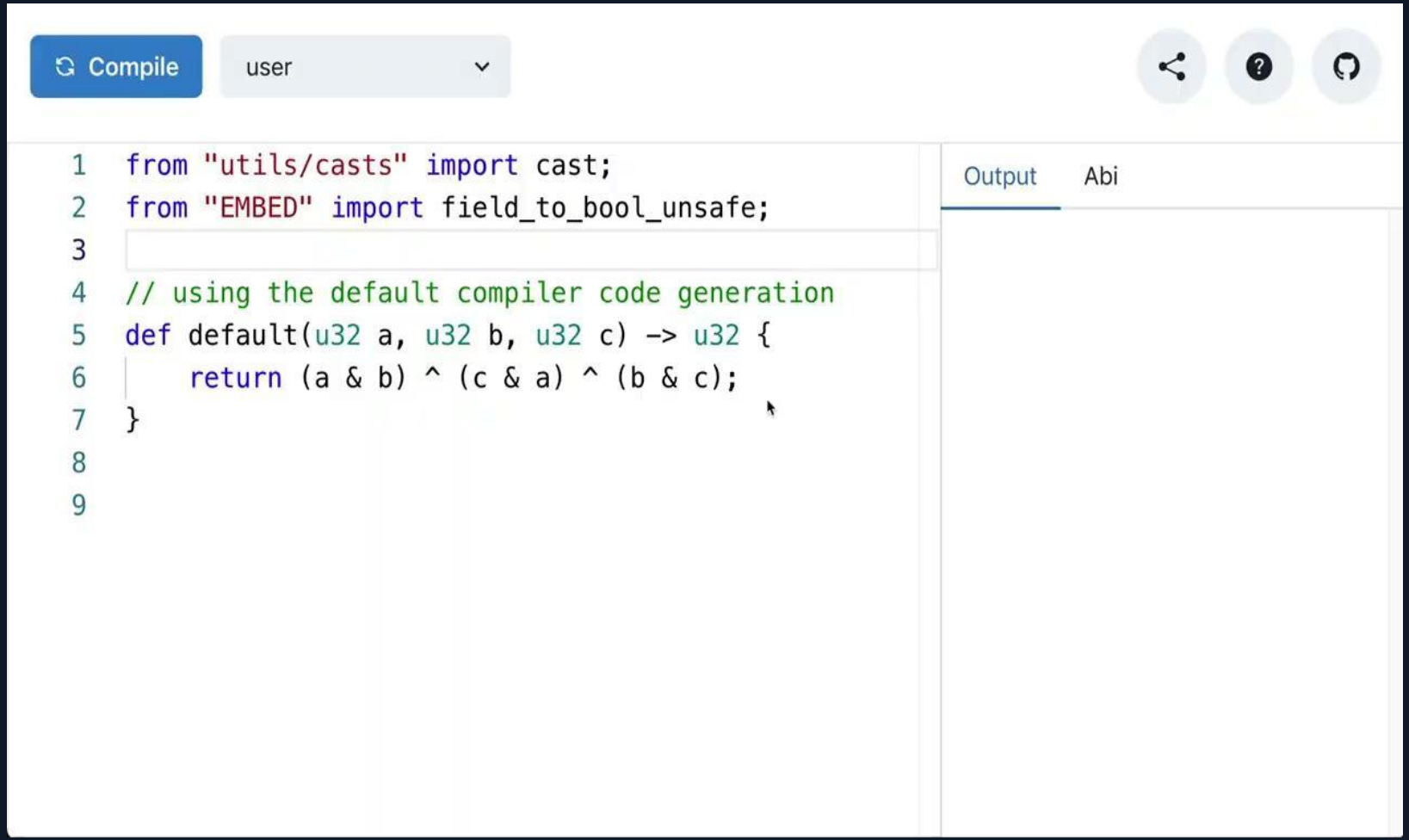

@zokratesproject


@ZoKrates


@vanderkriek


@schaeff

ZoKrates


Compile
user
Output
Abi
1 from "utils/casts" import cast;
2 from "EMBED" import field_to_bool_unsafe;
3
4 // using the default compiler code generation
5 def default(u32 a, u32 b, u32 c) -> u32 {
6 return (a & b) ^ (c & a) ^ (b & c);
7 }
8
9

Assembly

assembly.zok

```
def hand_optimized(u8 a, u8 b, u8 c) -> u8 {
    bool[8] a_bits = cast(a);
    bool[8] b_bits = cast(b);
    bool[8] c_bits = cast(c);

    bool[8] mut res_bits = [false; 8];

    for u32 i in 0..8 {
        field a = bool_to_field(a_bits[i]);
        field b = bool_to_field(b_bits[i]);
        field c = bool_to_field(c_bits[i]);
```

Assembly

assembly.zok

```
        field mut res = 0;
        field bc = b * c;
        asm {
            res <-- bc - (2 * bc - b - c) * a;
            bc - res === (2 * bc - b - c) * a;
        }

        res_bits[i] = field_to_bool_unsafe(res);
    }

    return cast(res_bits);
}
```