

SIEVE Circuit IR Specification

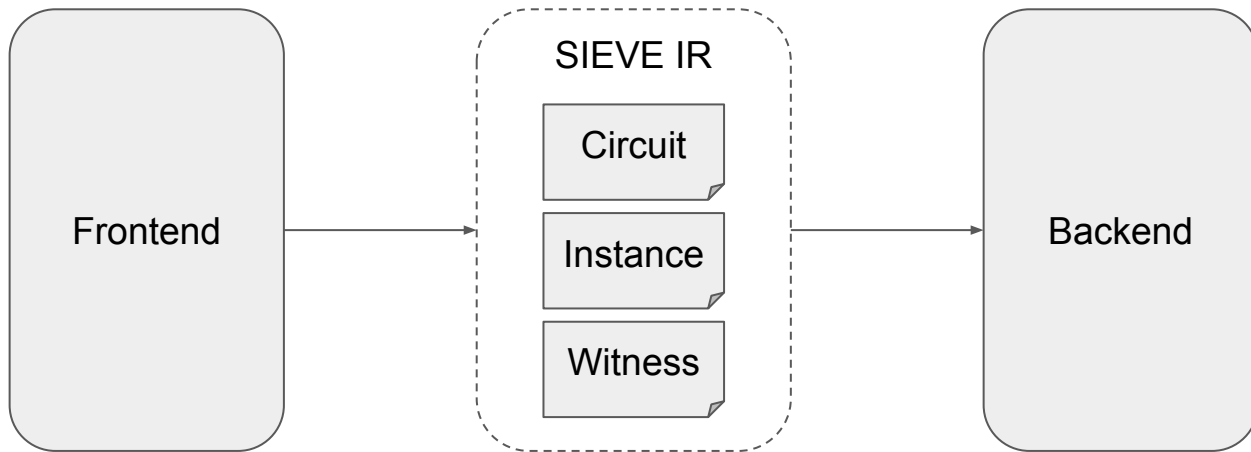
ZKProof 5.5 - 2 August 2023

James Parker (james@galois.com), **Kimberlee Model** (kimee@stealthsoftwareinc.com), Paul Bunn, David Darais, Daniel Genkin, Steve Lu, Tarik Riviere, Muthuramakrishnan Venkitasubramaniam, Xiao Wang, Steven Eker, Karim Eldefrawy, Stephane Graham-Lengrand, Vitor Pereira, Hadas Zeilberger, Michael Adjedj, Daniel Benarroch Guenun, Eran Tromer, Aurelien Nicolas, Constance Beguier, Alex Malozemoff, Chris Phifer, Mayank Varia

Background: SIEVE

- DARPA program focused on improving the utility and efficiency of (mostly interactive) ZK protocols
- Teams are split into two categories:
 - Frontends - Produce ZK statements for different applications
 - Backends - Efficiently verify ZK statements generated by frontends
- There are many frontend and backend teams, and they need a standard way to represent ZK statements

SIEVE Circuit IR



- Standardized circuit representation of ZK proofs
- Designed by a committee made up of all the SIEVE teams
- Enables backends to run all* ZK statements produced by frontends

Target Applications

- Proofs of software vulnerabilities
- Early focus on “really big” proofs
- Mobile <-> cloud proofs
- Policy Enforcement
 - Healthcare
 - Government Regulations
 - Transparency
 - Geo-fencing and location history

Why Circuits?

- **R1CS?**
 - Theoretically equivalent to Circuits
 - Practically not so equivalent
 - Round trip translation grows about 10x
- **Plonk-ish?**
 - Didn't have name recognition when we started
- **Memory Efficiency**
 - Statement Streaming
 - Delete temporary wires
- **Multi Field Proofs (“field switching”)**
- **Flexibility & Scalability**
 - Expectation: Frontend research yields smaller circuits
 - Reality: Frontend research yields additional features with higher circuit density

Language Overview: Types

```
// Index 0: Boolean  
@type field 2;
```

```
// Index 1: GF(2^63) with polynomial modulus x^63 + x + 1  
@type ext field 0 63 0x80000000000000003;
```

```
// Index 2: Unsigned 32-bit integers  
@type ring 32;
```

For some prime p and polynomial P , with coefficients $c_{0\dots n}$ we encode:

$$P = \sum c_i * p^i$$

- Supported types: prime fields, extension fields, rings
 - Circuits can contain multiple types in the same statement
 - Types are defined in the circuit header
 - Types are referenced by index

Language Overview: Gates

```
$0 <- @private(0);  
$1 <- @private(0);  
  
$2 <- @mul($0, $0);  
$3 <- @mul($1, $1);  
$4 <- @add($2, $3);  
  
@assert_zero($4);
```

- Basic gates: @add, @mul, @assert_zero
- Read from witness or instance: @private, @public
 - Values are streamed from a separate file
- Text and flatbuffer representations

Language Overview: Functions

```
@function(dot_prod_10, @out: 1:1, @in: 1:10, 1:10)
    $21 <- @mul($1, $11);
    // ...
@end

$25 <- @call(dot_prod_10, $0 ... $9, $10 ... $19);
```

- Users can define and call function gates
 - Useful for frequently used subcircuits

Language Overview: Plugin Extensions

```
@plugin(ram, init,  <size>)  
@plugin(ram, read,  <addr>)  
@plugin(ram, write, <addr>, <value>)
```

- ZK is moving quickly, so the IR needs a way to support the latest features and protocols
- IR supports plugins, which allow users to define their own gadgets
 - Typically useful when backends have efficient protocols for specific operations
- Official plugins: RAM, permutations, iterators, vectors, arithmetic
- Unofficial plugins: Polynomial equality checks, disjunctions

Language Overview: New and Delete

```
@new($10... $20);
```

```
// Assignments and usage of $10, $11, ... $20
```

```
@delete($10 ... $20);
```

- Allocate a block of contiguous wires with @new
 - Subsequent gates may assign within the block
- Delete a block of wires

Memory Management: Function Parameter Rules

```
@new($0 ... $9); @new($10 ... $19);
```

```
$20 <- @call(dot_prod_10, $0 ... $9, $10 ... $19);
```

```
@new($21 ... $40);
```

```
$41 ... $50 <- @call(vec_mul_10, $21 ... $30, $31 ... $40);
```

- Each Parameter Range must span within a single allocation
 - Subsets are okay
- Output parameters may implicitly allocate
 - May not conflict with prior allocations, even if unassigned

Memory Management: Why?

```
@function(dot_prod_10, @out: 1:1, @in: 1:10, 1:10)  
  // gates...  
@end
```

```
$25 <- @call(dot_prod_10, $0 ... $9, $10 ... $19);
```

- **Consecutive wire numbers**
- **Discontiguous wire storage (e.g. distinct allocations)**
- **“shape” mismatch between caller and function**

Function Scope

Out: \$0

In: \$1 ... \$10

In: \$11 ... \$20

Locals...

Calling Scope

Local: \$0 ... \$7

Local: \$8 ... \$15

Local: \$16 ... \$23

Local: \$24 ... \$32

Conversion Gates

Convert a value from one type to another

- Enable conversions in the Circuit's Header

```
@type field 2; // type 0: Booleans
@type field 2305843009213693951; // type 1: Mersenne 61
// Allow bidirectional conversions
@convert(@out: 0:61, @in: 1:1);
@convert(@out: 1:1, @in: 0:61);
```

Allow overflow, or forbid it
with `@no_modulus` (default)

- Use conversions within the circuit's body (not the type indexes)

Big-endian

```
$0 <- 1: <12345>; // (assign a constant)
0: $0 ... $60 <- @convert(1: $0, @modulus); // Convert to boolean
```

- Compose or decompose as bits or digits, with the prime as a base.

SIEVE Circuit IR Ecosystem

Frontends

- Proofs of vulnerability: Cheesecloth (Galois), Reverie (Trail of Bits)
- General Purpose/Policy Enforcement: zk-SecreC (Cyberbernetica), PicoZK (Stealth/UVM)

“Mid-ends” (SIEVE IR tools/libraries)

- WizToolKit
- zkInterface-sieve

Backends

- VOLE: Mac’n’cheese (Galois), EMP (Stealth/NWU)
- LIGERO: LIGERO (Ligero)
- LPZK: (Stealth), Formally-Verified (SRI)

Lessons learned

- When designing the IR by committee, people had different visions and priorities
 - Human vs machine readable
 - Debuggable vs efficient
 - Featureful vs simpler abstractions
- DARPA required interoperability between all teams
- Suggestions for future standards:
 - Attempt to understand everyone's vision and priorities
 - Define a process for proposing, debating, and agreeing on designs and features
 - Assign an unbiased arbiter to carry out the process

Beyond SIEVE

- SIEVE is ending in a year
- Hopefully the SIEVE Circuit IR standard will be useful outside of SIEVE
- Perhaps the standard will be adopted by ZKProof and continue to evolve with the wider community
 - PLONK plugin extension?
 - Compiler from SIEVE Circuit IR to PLONK?

Summary

- SIEVE Circuit IR has proven to be a useful language for encoding ZK statements
 - Enabled interoperability between ZK frontends and backends
- Consider using the SIEVE Circuit IR in your next project!

<https://github.com/sieve-zk/ir/>

- Resources
 - Blog: <https://stealthsoftwareinc.github.io/wizkit-blog/blog.html>
 - WizToolKit: <https://stealthsoftwareinc.github.io/wiztoolkit/>
 - zkInterface: https://crates.io/crates/zki_sieve