Introduction to the SIEVE Intermediate Representation

See Section 4 for list of Contributors

Last Updated: 2023-07-17

This material is based upon work supported by DARPA under Contracts No. HR001120C0087, HR001120C0086, HR001120C0085 and Agreement No. HR00112020021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

Identification and History

Version	Date	Notable Changes
0.1.0	2020-09-19	Initial version as proposed jointly by WizKit and FROMAGER teams on DARPA SIEVE.
0.1.1	2020-12-16	A number of bugfixes on the prior version. • Fix typographical errors and omissions. • Add checks for matching header, `num_wires`, and etc., across multiple resources. • Eliminate unused type-checking and arity-checking attributes from the attribute grammar, because IR0 has only one meaningful type. • Add `num_wires` to the instance and witness in the text and binary grammars.
0.2.0	2021-03-19	Remove the `num_wires` and associated input size data and add a `@delete` directive.
1.0.0	2021-07-20	Add "uniformity" features: function gates, public-index for loops, and private-condition switch-case statements.

1.0.1	2022-04-29	Fix the following issues: • Fix switch-statement semantics which required a nested switch-statement where the outer case is inactive and the inner switch-statement has no active branch to cause an unexpected witness-statement invalidity. • Constrain for-loop bounds and wire-ranges such that the first must be less than or equal to the last iteration or wire. • Add a warning that the switch-statement conversion algorithm may produce a poorly-formed relation with certain forms of switch-statement and for-loop nesting. Add a suggestion to unroll for-loops in those conditions. • Assorted typo fixes and performance improvements.
2.0.0	2022-11-08	The 2.0.0 revision is a significant departure from prior revisions, regressing some of the haphazardly developed high level features and preparing for the development of a distinct high-level (translation), and low-level (circuit) IR layers. This release was used during the SIEVE Phase II testing event, and specifies the Circuit-IR – characterized as follows. • Flat circuit with numbered wires. • Function's encapsulating circuit fragments. • Explicit allocation directives enable memory efficiency. • A new plugin system allows access to backend-defined functionality.
2.1.0	2023-07-17	Adds ring and extension field types to the Circuit-IR and multi-wire convenience modes for copy gates and input gates.

Contents

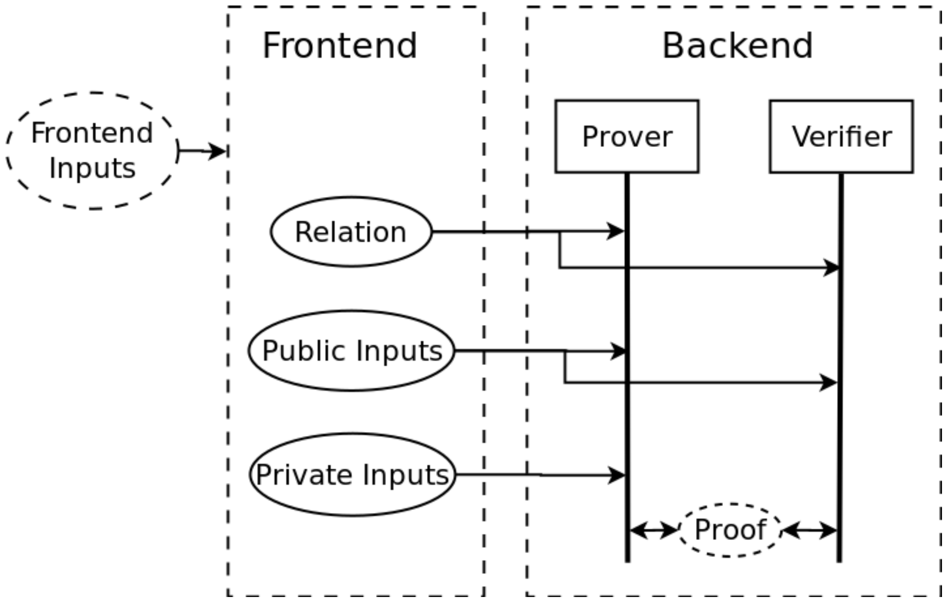
1 Introduction

This document specifies the current version of the SIEVE Intermediate Representation (IR). Throughout the DARPA SIEVE program, changes to this specification were to be discussed and modified only through the decision process outlined in the SIEVE IR Agreement document. The IR's developers are currently seeking a venue to host further development and standardization after the SIEVE program ends.

The IR is an interaction between the front-end and the back-end portions of a Zero Knowledge (ZK) proof pipeline. The frontend transforms high level statements in a target domain into the IR. It is the producer of the three resources which are the subject of this document. The backend is the consumer of them: it is an interaction between a Prover and a Verifier, where the Prover wishes to prove a statement to the Verifier, without revealing a secret component of the proof. The primary goal for this specification is to define a standard format for ZK statements, so that there is interoperability between multiple frontends and backends.

At a high level, ZK proofs involve three resources provided by the frontend and used by the backend as the fact being proven.

- Relation – a mathematical relationship between the inputs.
- Public inputs – inputs to the relation given to both the Prover and Verifier.
- Private inputs – inputs to the relation given only to the Prover.



1.1 Genesis

In the course of the SIEVE Program, performers experimented with a variety of representations ranging from very high level through very low level. The SIEVE Program has proved statements on every platform from mobile web to gigantic cloud instances, with runtime taking anywhere from milliseconds to days to prove. Additionally, SIEVE backends have preferred a wide variety of intermediate representations – including R1CS, C++ compilation, and other custom solutions. The greatest challenge in developing the IR has been making it flexible enough to accomodate so many requirements.

The program's first attempt at a flexible IR, the SIEVE IR v1.0, was to essentially overlay high-level features upon a flat circuit with numbered wires. This was a failure for backends wanting just flat circuits because they were forced to unroll high level features. It was also a failure for the higher level backends because the numbered wires were insufficient to convey high level meaning and long stretches of flat circuitry would cause slowdowns for them.

1.2 Flexibility and Scalability

To resolve this issue, the latest SIEVE IR revision has two layers. The lower layer, called the Circuit-IR, defines a flat circuit, augmented with flat functions. While it is not completely flat, it does strike a balance between statement size and interpreter complexity – important because many SIEVE backends bottlenecked on disk IO while reading completely flat circuits.

A great strength of the Circuit-IR is circuit streaming. Gates are processed one after the other, and their outputs (wires) are stored until needed for further computation. Ordinarily, this would cause memory usage to increase until either the circuit ends or the available memory is exhausted. The Circuit-IR implements memory management operations and restrictions which enables a circuit to declare what wires it will produce and when it is finished with them, so that its memory may be recycled.

The higher layer, called Translation-IR, is a fairly high level language in terms of control flow. For maximal compatibility, SIEVE Performers are also developing a translation into the Circuit-IR, hence the name Translation-IR. But the Translation-IR need not be translated just to the Circuit-IR, one of its goals is that it be translatable to non-SIEVE IRs such as the aforementioned R1CS, C++ compilation, or other custom solutions. At the time of writing, the Circuit-IR recently survived the program's Phase II Testing Event, while the Translation-IR is currently in development.

1.3 Multi Field Circuits

To most practitioners of ZK, a single prime field is chosen at the beginning of a proof and used throughout. However, for some applications it is desirable to use multiple primes for different elements within a single larger proof. For example a large and expensive prime may be needed to verify public-key signatures, while a medium sized prime is necessary for large scale business logic.

To accommodate these applications, the IR must allow for multiple fields within a single relation. To the frontend each field must describe the type of a wire, while to the backend these wires actually belong to multiple independent proofs. An analogy to the real world might be a circuit card with transistor logic on one side and high voltage on the other.

Occasionally information from one field will be required in another. The IR models this using a conversion gate with inputs in one field and outputs in another. To continue the analogy, a relay would allow information to flow from transistor logic into high voltage, or in reverse, an analog-digital converter. In ZK, methodologies must be developed and used to show equivalence of inputs and outputs across independent proofs or even across different proof systems.

1.4 Resources of the SIEVE IR

Each file or connection in the SIEVE IR is referred to as a resource, a stream of bytes to be consumed by the backend. A relation is a resource, and so are the inputs to the relation. There is also a special configuration resource which helps frontends and backends negotiate compatibility.

The SIEVE IR introduces two types of relation: the Circuit-IR and the Translation-IR.

- The Circuit-IR is defined by flat lists of gates and wires; functions may be defined and reused within the circuit.
- The Translation-IR is a program which outputs a relation in the Circuit-IR format. The backend is given some amount of control over how the Translation-IR is translated, and is free to reimplement common or standardized libraries, translate to an alternative IR, or even prove statements directly from the Translation-IR.

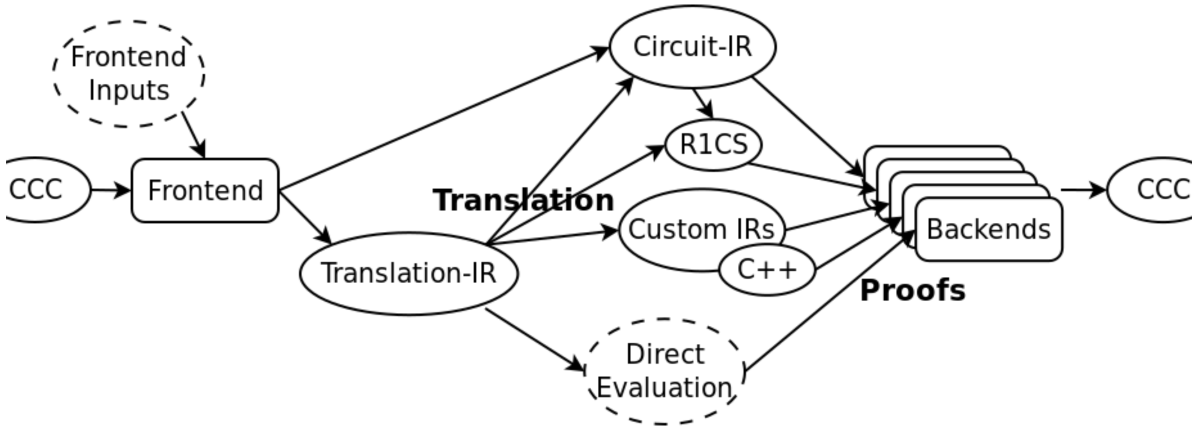
In recognition that many ZK backends can improve upon simple circuits' performance when particular functionality is desired, both layers of the IR define a **plugin** interface allowing calls from the IR into backend specific functionality.

To ease the frontends choice of configuration constants (such as prime fields and plugin parameters), a Circuit Configuration Communication occurs so that the backend may feed optimal constants to the frontend.

In total, the IR winds up with five resources: four in the frontend to backend direction, and a configuration which flows in the opposite direction.

- Relation
  - Translation-IR – high level program which emits a flat circuit
  - Circuit-IR – flat circuit which is easy for backends to ingest
- Public inputs – a list of inputs known to both the Prover and Verifier
- Private inputs – a list of inputs known only to the Prover
- Circuit Configuration Communication (CCC) - hints from the backend to the frontend describing features supported by the backend.

This illustration shows how the backends start by producing the CCC – typically checked in with its source code – enabling a frontend to choose types and plugins suitable to a backend. Then the frontend can produce SIEVE IR enabling a web of available translations before the backend can finally prove the statement.



2 Headers

All SIEVE IR files start with a header. The header contains a version number for quick recognition of the IR and a resource type. Each resource type may define additional header declarations that appear after the version and resource type, but before the `@begin` keyword.

`circuit` Circuit-IR

`translation` Translation-IR

`public_input` Public input streams (the *instance*)

`private_input` Private input streams (the *witness*)

`configuration` the Circuit Configuration Communication

Here is an example header.

```
version 2.0.0;
private_input;
```

This indicates the file is a private input resource for version 2.0.0 of the SIEVE IR.

3 Subdocuments of the SIEVE IR

This spec is organized into multiple subdocuments.

- The Circuit-IR document includes formats and semantics for the public and private input streams and the CCC.
- The Translation IR document is currently in development, and unreleased.
- the Plugins directory contains sub documents for each plugin which has been standardized (although backends are free to develop their own plugins).

4 Contributors

Authors

- Team Wizkit: Paul Bunn, David Darais, Daniel Genkin, Steve Lu, Kimberlee Model, Tarik Riviere, Muthuramakrishnan Venkitasubramaniam, Xiao Wang
- Team Emphasize: Steven Eker, Karim Eldefrawy, Stéphane Graham-Lengrand, Vitor Pereira, Hadas Zeilberger
- Team Fromager: Michael Adjedj, Daniel Benarroch Guenun, Eran Tromer, Aurélien Nicolas, Constance Beguier, Alex Malozemoff, James Parker, Chris Phifer
- Team Oracles: Mayank Varia

The following individuals from the T&E Team were instrumental in moderating the IR's development: David A. Wilson (PI), R. Nicholas Cunningham, J. Parker Diamond, Hanson Duan, Ariel Hamlin, Noah Luther, Richard Shay.

Disclaimers

Distribution Statement "A": Approved for Public Release, Distribution Unlimited.

This material is based upon work supported by DARPA under Contracts No. HR001120C0087, HR001120C0086, HR001120C0085 and Agreement No. HR00112020021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

The authors of the SIEVE IR have made every effort that the SIEVE IR Specification, if adopted by a recognized standards body, would be compatible with the OSI Open Standard Requirement.

No Intentional Secrets We have made every effort to ensure that the IR contains no necessary secrets. Further, several of the SIEVE Performers have or will open-source their IR implementations.

Availability The SIEVE Performers have agreed to release the text of the IR Specification under the Creative Commons Attribution 4.0 International (CC-BY 4.0) terms.

Patents Although we are not aware of any current patents which may be necessary to implement the IR, the SIEVE Performers have agreed not to enforce any intellectual property rights which may be necessary to implement the IR.

Dependencies To our knowledge, all technologies dependent by the IR are already themselves open-source.