Isekai Workshop

JUNE 2019

sikoba
RESEARCH

isekai

A FRAMEWORK FOR ZERO-KNOWLEDGE COMPUTATIONS

- Isekai (https://github.com/sikoba/isekai)
- Prove the execution of arbitrary programs in Zero-Knowledge
- For any language
- Using any Zero-Knowledge Proof scheme
- Work in progress!

ISEKAI ZERO-KNOWLEDGE PROOFS

- A *prover* can prove he executed a function f on some inputs

  u = f(x,w) u,x are public, w are private

1. The *prover* sends the proof π, inputs x, output u and function f to a *verifier*

   Prover  (π,u,x,f)

2. *Verifier* checks (π,u,x,f) is valid and is **convinced** that prover knows w such that u=f(x,w) with very high probability. *Verifiers* does **not** know w.

- Verification is **faster** than executing f(x,w)

ISEKAI HIGH-LEVEL WORKFLOW

Computer program (source code)

Circuit

Equations system

Proof

ISEKAI INPUT

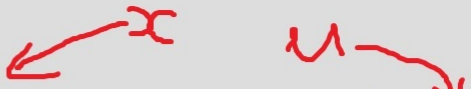
C source code, with a function ‘outsource’ having a specific signature

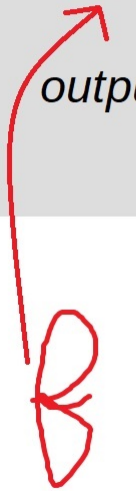
```
#include "eqtest-ifc.h"

void outsource(struct Input *input, struct Output *output)
{
	output->x = (input->a + 5) == (input->b * 2);
}
```

ISEKAI INPUT

C source code, with a function outsource having a specific signature

```
#include "eqtest-ifc.h"

void outsource(struct Input *input, struct Output *output)
{
        output->x = (input->a + 5) == (input->b * 2);
}
```





ISEKAI INPUT

Zero-Knowledge Input

```
void outsource(struct Input *input, struct NzikInput * nzik, struct Output *output)
{
        output->x = (input->a + 5) == (nzik->b * 2);
}
```

ISEKAI INPUT

MyFunction.c

```
void outsource(struct Input *input, struct NzikInput * nzik, struct Output *output)
{
        output->x = (input->a + 5) == (nzik->b * 2);
}
```

MyFunction.c.in

```
3       // input (public)
1       // always 1
4       // nzik (private)
```

ISEKAI HIGH-LEVEL WORKFLOW

Computer program (source code)

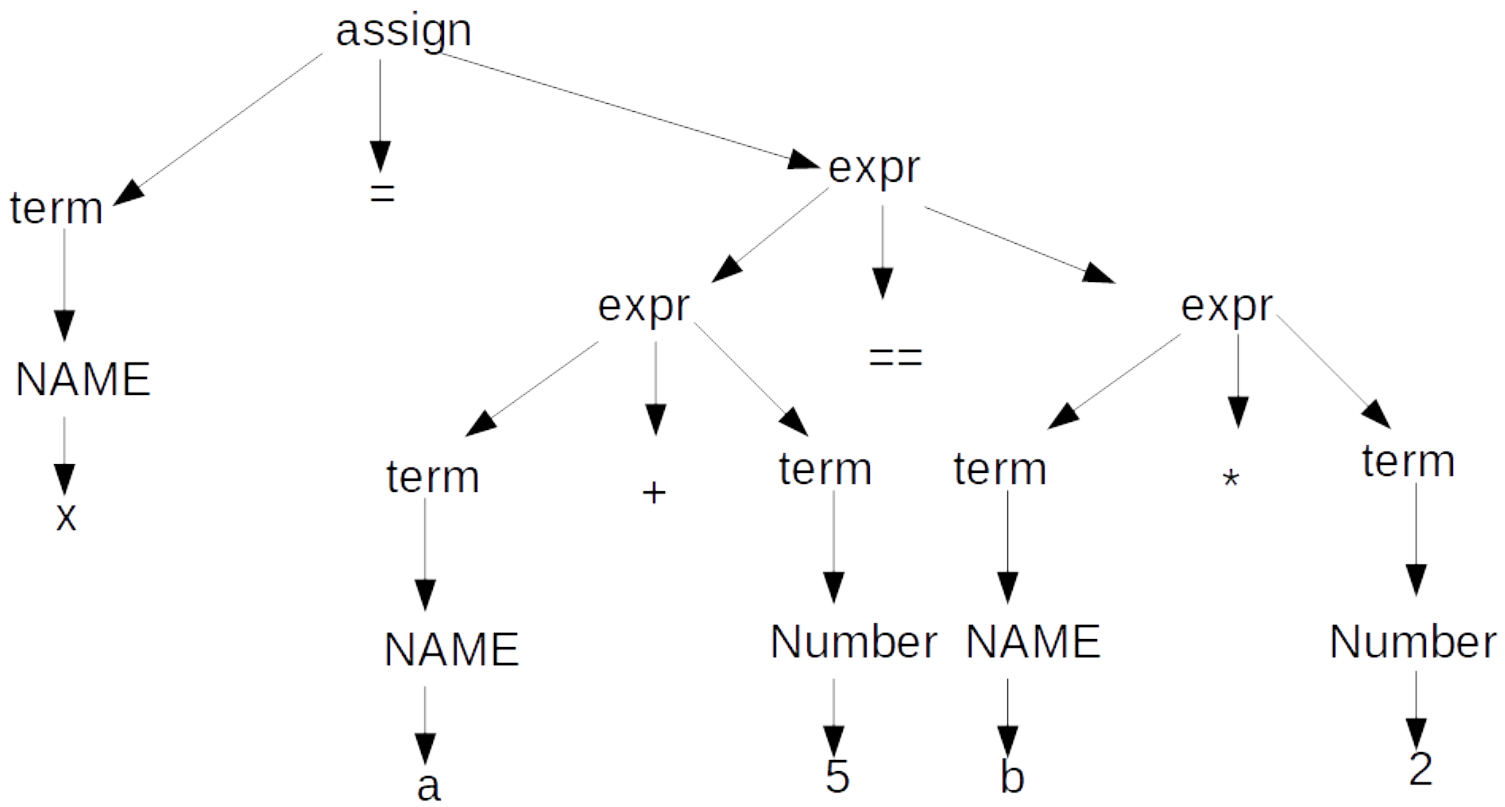
Circuit

Equations system

Proof

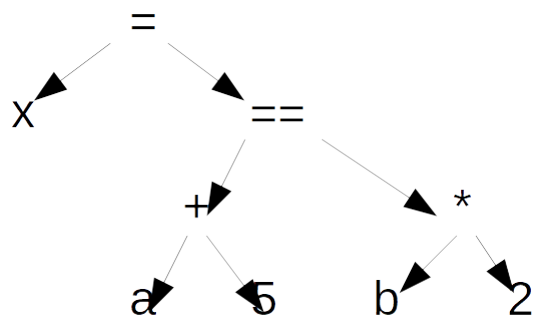
PARSING

x = a+5 == b*2



ABSTRACT SYNTAX TREE

```
x = a+5 == b*2
```



AST

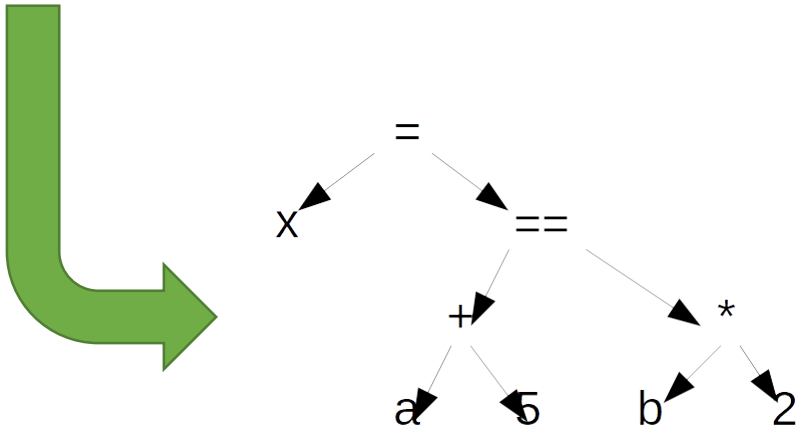
Parent Scope

Key	Value
a	3
b	4
x	True

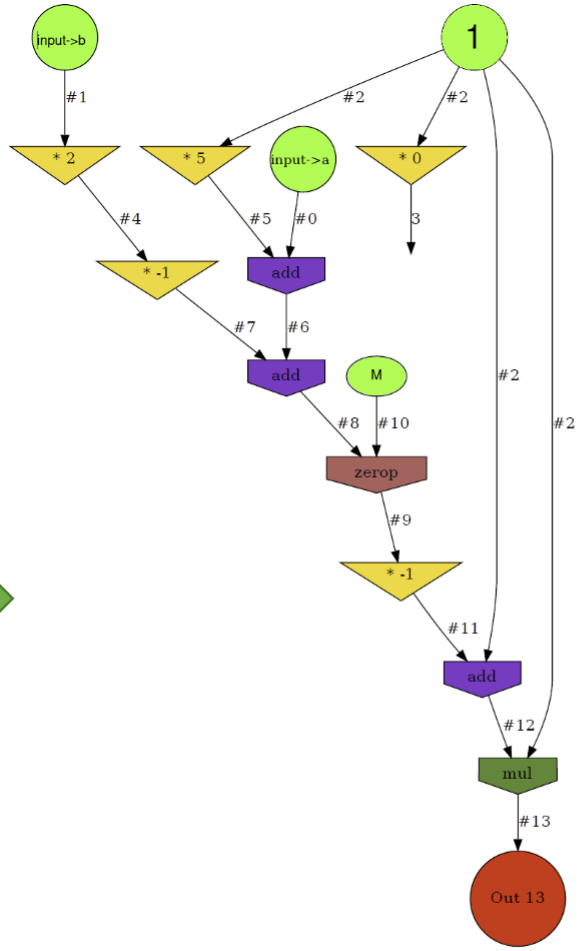
ADT (Symbol Table)

ARITHMETIC CIRCUIT

x = a+5 == b*2



AST



Circuit

GENERATE CIRCUIT WITH ISEKAI

```
Isekai/wk: ~$ ../isekai --arith=ex1.ari ex1.c

total 14
input 0 # input
input 1 # input
input 2 # one-input
const-mul-0 in 1 <2> out 1 <3> # zero
const-mul-2 in 1 <1> out 1 <4> # multiply-by-constant 2
const-mul-5 in 1 <2> out 1 <5> # constant 5
add in 2 <0 5> out 1 <6> # #<Isekai::ArithAddReq:0x7fff2e16db00>
const-mul-neg-1 in 1 <4> out 1 <7> # zerop subtract negative
add in 2 <6 7> out 1 <8> # zerop diff
zerop in 1 <8> out 2 <10 9> # zerop #<Isekai::ArithAddBus:0x7fff2e1ebdc0>
const-mul-neg-1 in 1 <9> out 1 <11> # zerop inverse
add in 2 <2 11> out 1 <12> # zerop result
mul in 2 <2 12> out 1 <13> # output-cast
output 13 #
```

ISEKAI HIGH-LEVEL WORKFLOW

Computer program (source code)

Circuit

Equations system

Proof

ISEKAI HIGH-LEVEL WORKFLOW

Computer program (source code)

Circuit

Equations system

Proof

R1CS

Set of bilinear equations $\langle A,B \rangle = C$

$(a_0 + a_1 * x_1 + \ldots + a_n * x_n) * (b_0 + b_1 * x_1 + \ldots + b_n * x_n) = c_0 + c_1 * x_1 + \ldots + c_n * x_n$

a_i , b_i , c_i are coefficients in a finite field

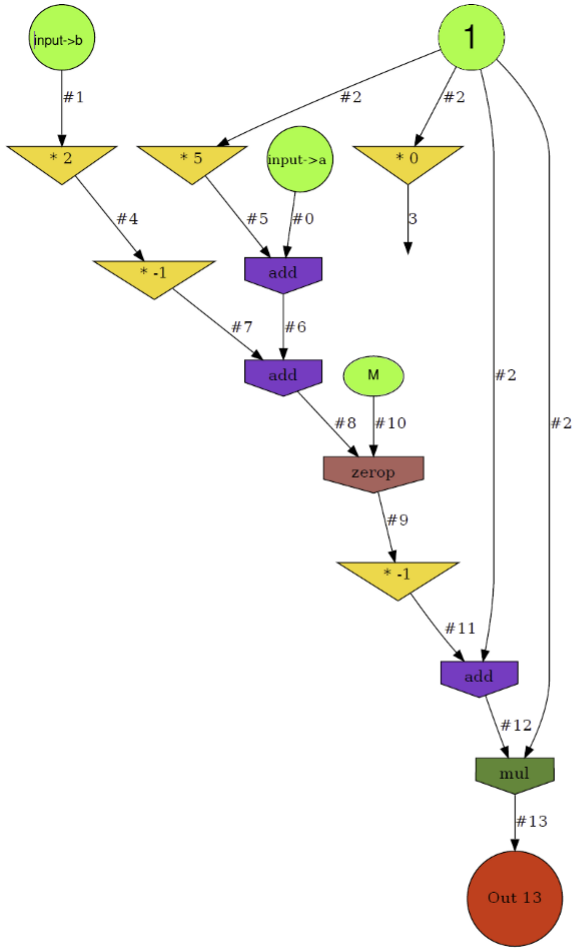
x_i are the variables of the system

RANK-1 CONSTRAINTS SYSTEM

```
#include "eqtest-ifc.h"

void outsource(struct Input *input, struct Output *output)
{
        output->x = (input->a + 5) == (input->b * 2);
}
```

$$x_8 = x_0 + 5 - x_1 * 2$$
$$(1 - x_9) * x_8 = 0$$
$$x_8 * x_{10} - x_9 = 0$$
$$x_{13} = 1 - x_9$$

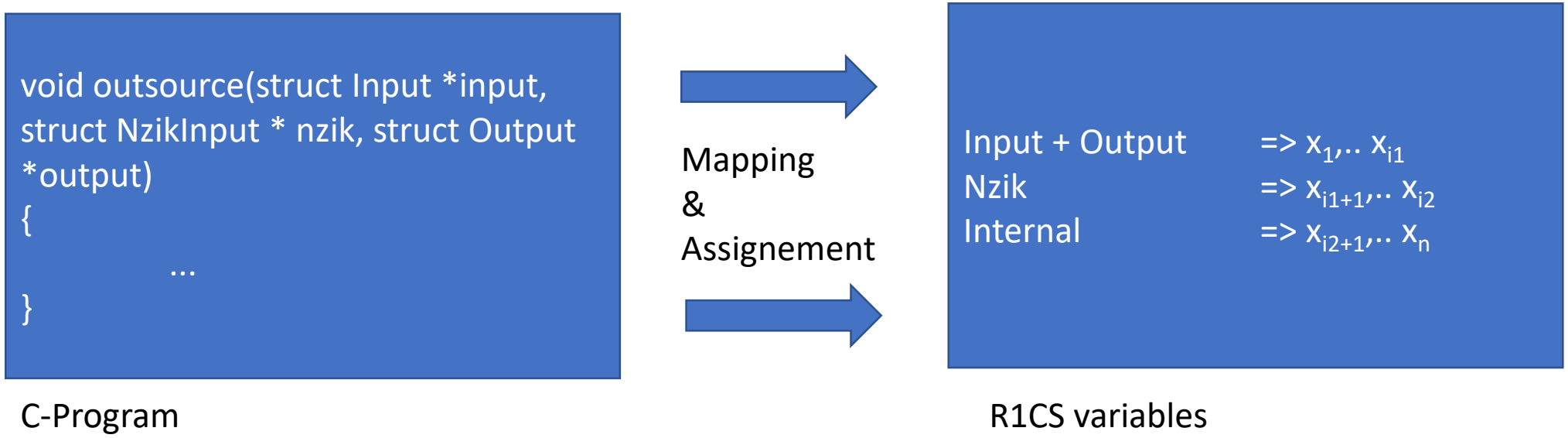
R1CS



Circuit

R1CS REDUCTION

- R1CS variables represent the input data (both public and zero-knowledge) of the high-level statements as well internal variables
- The **mapping** of the variables is done during the R1CS reduction
- The values of the variables (assignment) is a solution to the R1CS and represent an execution of the program



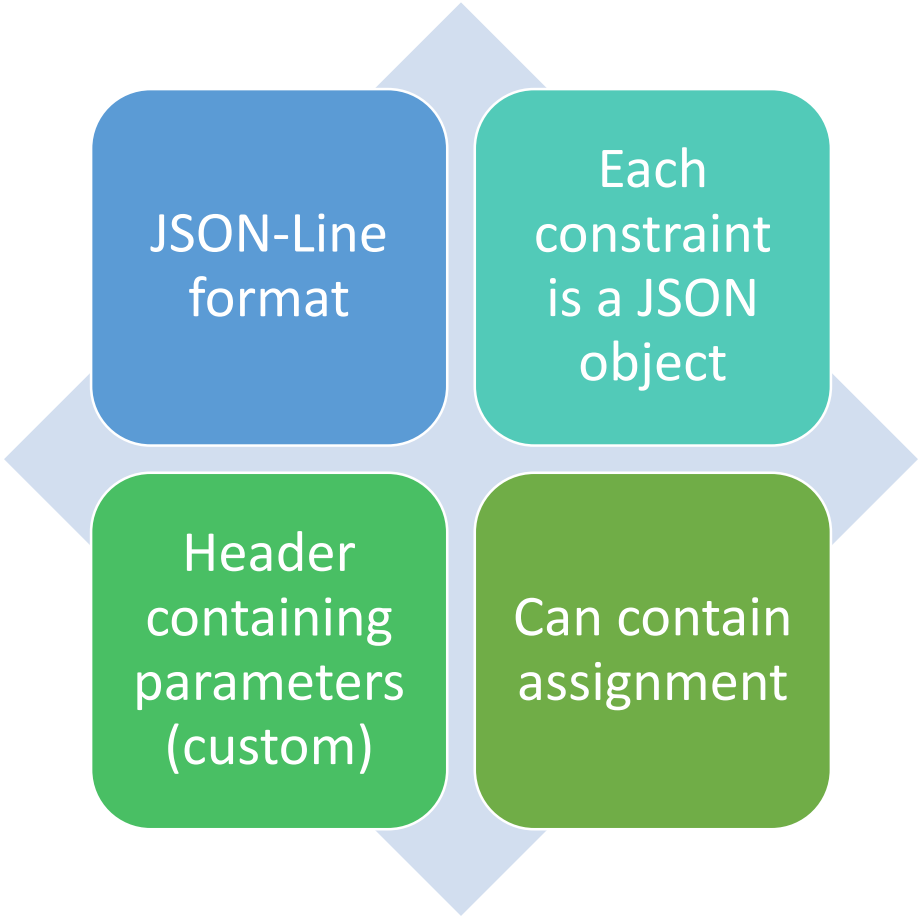
C-Program

R1CS variables

GENERATE R1CS WITH ISEKAI

```
Isekai/wk: ~$ ../isekai --r1cs=ex1.j1 ex1.c

(enter) Parsing and Evaluating the circuit      [           ] (1561037885.1220s
x0.00 from start)
(leave) Parsing and Evaluating the circuit      [0.0023s x0.00]
(1561037885.1243s x0.00 from start)
Translating Constraints ...
      Constraint translation done
      Memory usage for constraint translation: 0 MB
Assignment of values done ..
```

J-R1CS


JSON-Line format
Each constraint is a JSON object
Header containing parameters (custom)
Can contain assignment

PRO

Extensibility

Implementation

Lightweight

Interoperability

Readability

Scalability

J-R1CS

{"r1cs":{"constraint_nb":3,"extension_degree":1,"field_characteristic":1,"instance_nb":4,"version":"1.0","witness_nb":3}}

{"A":[[0,"0"],[1,"1"],[2,"21888242871839275222246405745257275088548364400416034343698204186575808495615"],[3,"5"]],"B":[[0,"1"],[6,"21888242871839275222246405745257275088548364400416034343698204186575808495616"]],"C":[[0,"0"]]}

{"A":[[0,"0"],[1,"1"],[2,"21888242871839275222246405745257275088548364400416034343698204186575808495615"],[3,"5"]],"B":[[0,"0"],[7,"1"]],"C":[[0,"0"],[6,"1"]]}

{"A":[[0,"0"],[3,"1"]],"B":[[0,"0"],[3,"1"],[6,"21888242871839275222246405745257275088548364400416034343698204186575808495616"]],"C":[[0,"0"],[4,"1"]]}

J-R1CS

{"r1cs":{"constraint_nb":3,"extension_degree":1,"field_characteristic":1,"instance_nb":4,"version":"1.0","witness_nb":3}}

{"A":[[0,"0"],[1,"1"],[2,"-2"],[3,"5"]],"B":[[0,"1"],[6,"-1"]],"C":[[0,"0"]]}

{"A":[[0,"0"],[1,"1"],[2,"-2"],[3,"5"]],"B":[[0,"0"],[7,"1"]],"C":[[0,"0"],[6,"1"]]}

{"A":[[0,"0"],[3,"1"]],"B":[[0,"0"],[3,"1"],[6,"-1"]],"C":[[0,"0"],[4,"1"]]}

{"inputs":["3","2","1","0"],"witnesses":["0","1","16416182153879456416684804308942956316411273300312025757773653139931856371713"]}

ISEKAI HIGH-LEVEL WORKFLOW

Computer program (source code)

Circuit

Equations system

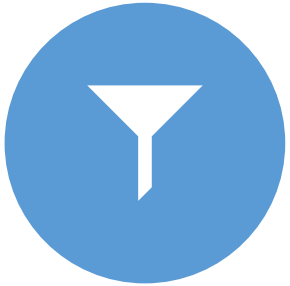
Proof

ZERO KNOWLEDGE PROOF

Source code is transformed into ***Equations*** system

Program execution is transformed into a ***Solution*** (assignment)

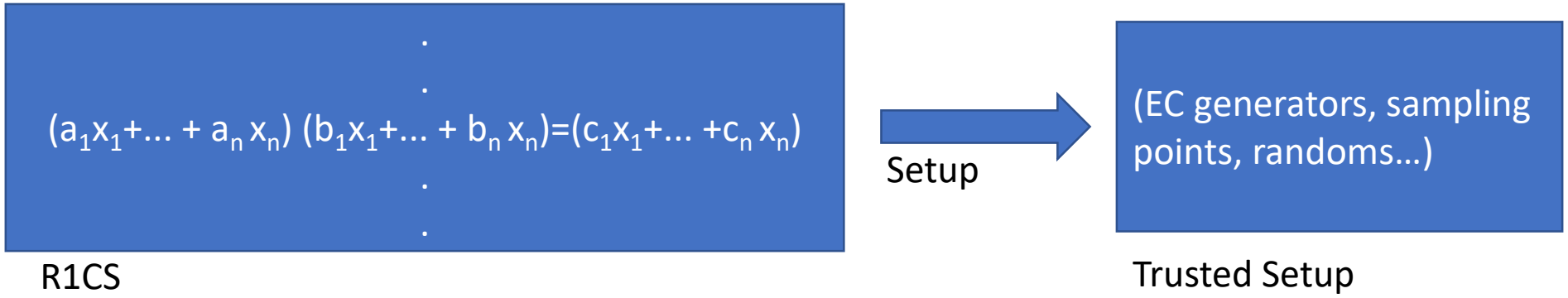
The system of equations provides the **constraint polynomial**



The Solution satisfies the constraints, they are the **roots** of the constraint polynomial

GENERATING THE PROOF

1. Generate the trusted setup from the R1CS



2. Generate the proof from the trusted setup and the assignments

3. Verify the proof

GENERATE PROOF

```
Isekai/wk: ~$ ../isekai --snark=ex1 ex1.j1

* The verification result2 is: PASS
Proved execution successfully with libSnark, generated:
                Trusted setup : ex1.s
                Proof: ex1.p
```

VERIFY PROOF

```
Isekai/wk: ~$ ../isekai --verif=ex1 ex1.j1.in

* The verification result2 is: PASS
Congratulations, the proof is correct!
```

EXAMPLE 2.

- a,b => a+5 < 2*b

```
#include "ex2.h"

void outsource(struct Input *input, struct Output *output)
{
		output->x = (input->a + 5) < (input->b * 2);
}
```

EX3. ZERO-KNOWLEDGE

- ?,? => a+5-2*b

```
#include "ex3.h"

void outsource(struct Input *input, struct NzikInput * nzik, struct Output *output)
{
		output->x = (nzik->a + 5) - (nzik->b * 2);
}
```

MORE ADVANCED EXAMPLES

- Ex4. Median
- Ex5. Hash
- Ex6. Overflow

GUILLAUME DREVON
gd@sikoba.com