STARKWARE

STARK-friendly crypto primitives wish-list

Eli Ben-Sasson

April 2023

Wishlist for crypto primitives

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

Wish-list for Primitive P:

1. Minimize t x w, |F|, d (also s, e)
2. Minimize CPU time of computing P
3. Make it field agnostic, and safe!

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

Wish-list for Primitive P:

1. Minimize t x w, |F|, d (also s, e)
2. Minimize CPU time of computing P
3. Make it field agnostic, and safe!

ethSTARK (Rescue):

1. t x w = 120, log |F| ~ 62, d=3
2. CPU time large, esp in large F (cube root!)
3. Relatively safe, AES-like (say the experts)

Poseidon in Starknet/Ex:

1. t x w = 226, log |F|~252, d=2
2. CPU time better than Rescue (100 microsec)
3. Relatively safe (though less than Rescue)

AIR Parameters: *t, w, F, d, s, e*

STARKWARE

Keccak:

1. t x w = 70,000-90,000, log |F| ~ 252, d=2
2. CPU time amazing (1 microsecond)
3. Very safe

Wish-list for Primitive P:

1. Minimize t x w, |F|, d (also s, e)
2. Minimize CPU time of computing P
3. Make it field agnostic, and safe!

ethSTARK (Rescue):

1. t x w = 120, log |F| ~ 62, d=3
2. CPU time large, esp in large F (cube root!)
3. Relatively safe, AES-like (say the experts)

Poseidon in Starknet/Ex:

1. t x w = 226, log |F|~252, d=2
2. CPU time better than Rescue (100 microsec)
3. Relatively safe (though less than Rescue)

Overview

- STARKs - Integrity Through Math
- Arithmetization
- Wish list for crypto primitives

Integrity Through Math

STARKWARE

STARK

Integrity* via Math

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Integrity Through Math

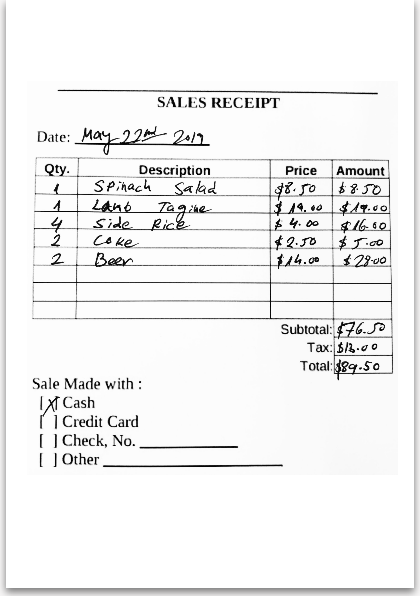
STARK

Integrity* via Math

Claim
total=$89.50

PROVER
Party producing proof
(Grocer)

VERIFIER
Party checking proof
(Customer)



SALES RECEIPT

Date: May 22nd 2019

Qty.	Description	Price	Amount
1	Spinach Salad	$8.50	$8.50
1	Lamb Tagine	$19.00	$19.00
4	Side Rice	$4.00	$16.00
2	Coke	$2.50	$5.00
2	Beer	$14.00	$28.00

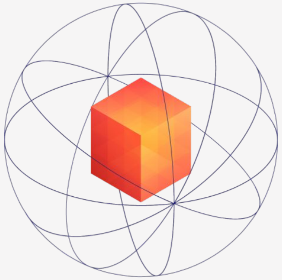
Subtotal: $76.50
Tax: $13.00
Total: $89.50

Sale Made with :
[X] Cash
[ ] Credit Card
[ ] Check, No. ________
[ ] Other ________

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math



STARK

Integrity* via Math

Privacy (Zero Knowledge, ZK)
Prover’s private inputs are shielded

Scalability
Exponentially small verifier running time*
Nearly linear prover running time*

Universality
Applicability to general computation

Transparency
No toxic waste (i.e. no trusted setup)

Lean & Battle-Hardened Cryptography
e.g. post-quantum secure

*With respect to size of computation

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Integrity Through Math

STARK

Integrity* via Math

1992

Checking Computations in Polylogarithmic Time

ászló Babai[1] — Chicago[6] and iv., Budapest

Lance Fortnow[2] — Dept. Comp. Sci. Univ. of Chicago[6]

Leonid A. Levin[3] — Dept. Comp. Sci. Boston University[4]

Mario Szegedy[5] — Dept. Comp. Sci. Univ. of Chicago

Abstract. Motivated by Manuel Blum's concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x, representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S.

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log\log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired*. (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the "theorem-candidate" is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log\|P\|)^{O(1/\varepsilon)}$.

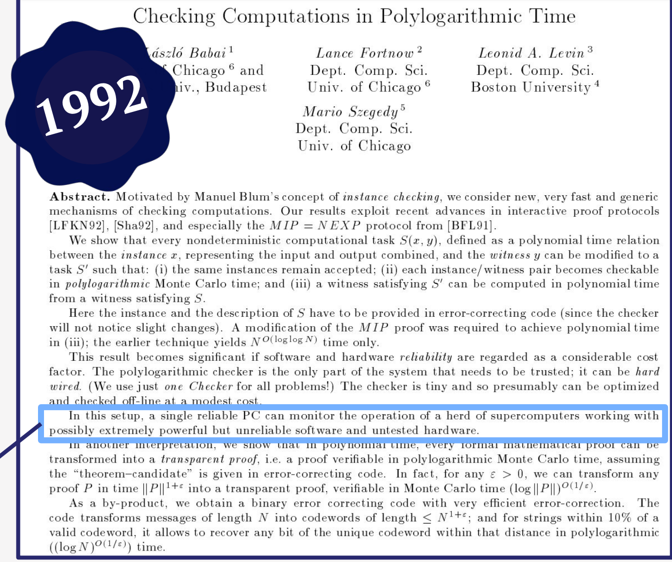
As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\le N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Integrity Through Math

STARK

Integrity* via Math


1992

Checking Computations in Polylogarithmic Time

László Babai [1]
Univ. of Chicago [6] and
Eötvös Univ., Budapest

Lance Fortnow [2]
Dept. Comp. Sci.
Univ. of Chicago [6]

Leonid A. Levin [3]
Dept. Comp. Sci.
Boston University [4]

Mario Szegedy [5]
Dept. Comp. Sci.
Univ. of Chicago

Abstract. Motivated by Manuel Blum's concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x, representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S.

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log\log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired.* (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the "theorem–candidate" is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log\|P\|)^{O(1/\varepsilon)}$.

As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\leq N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Integrity Through Math

STARKWARE

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P,** reached state hash **y***

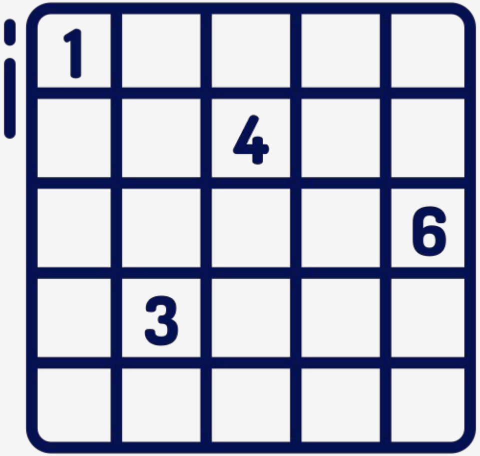
"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**



PCP

Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P,*** *reached state hash* ***y***

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by *x, y, P,* and #tx (=1,000,000)

1	1	3	2	6
4	1	4	7	3
8	5	8	3	6
6	3	5	9	7
5	2	8	7	6

PCP

Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P,*** *reached state hash* ***y***

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

Prover submits solution

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**

1	1	3	2	6
4	1	4	7	3
8	5	8	3	6
6	3	5	9	7
5	2	8	7	6

PCP

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P,** reached state hash **y***

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

Prover submits solution

Verifier samples and checks a single constraint

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**

1	1	3	2	6
4	1	4	7	3
8	5	8	3	6
6	3	5	9	7
5	2	8	7	6

PCP

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P**, reached state hash **y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A “proof” of a false claim satisfies < 1% of constraints!

“…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …”

Prover submits solution

Verifier samples and checks a single constraint

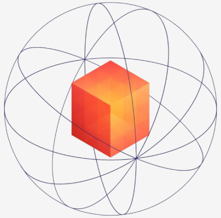
* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**

1	1	3	2	6
4	1	4	7	3
8	5	8	3	6
6	3	5	9	7
5	2	8	7	6

STARK



Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P,*** *reached state hash* ***y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A "proof" of a false claim satisfies < 1% of constraints!

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

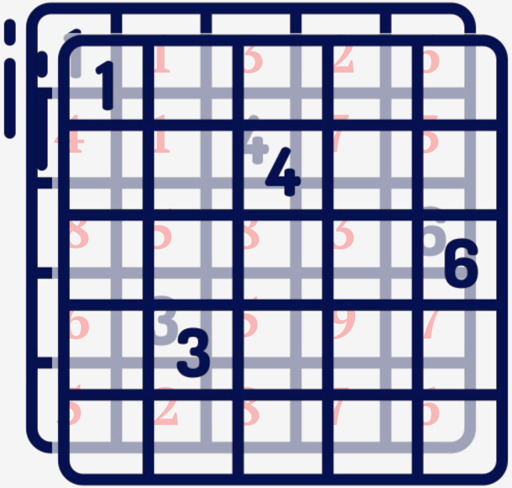
Prover submits solution

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**

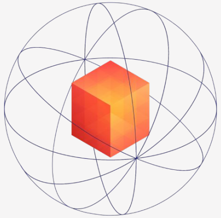


Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P**, reached state hash **y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A “proof” of a false claim satisfies < 1% of constraints!

“…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …”

STARK



Prover submits solution

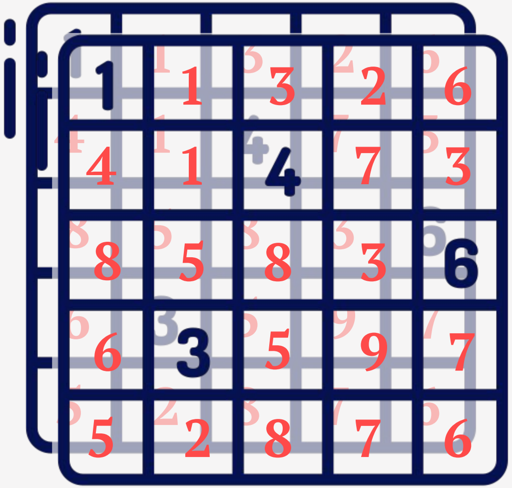
Verifier posts another (random) sudoku puzzle

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

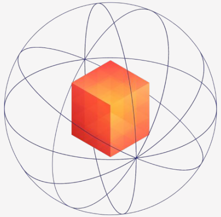
STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**



STARK



Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P****, reached state hash* ***y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A "proof" of a false claim satisfies < 1% of constraints!

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

Prover submits solution

Verifier posts another (random) sudoku puzzle

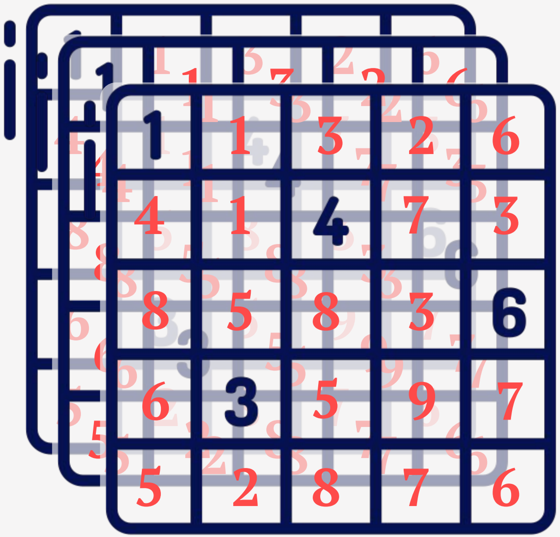
* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**

Claim: *Starting @ state hash **x**, after **1,000,000** txs processed by program **P**, reached state hash **y***



STARK

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A “proof” of a false claim satisfies < 1% of constraints!

Prover submits solution

“...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ...”

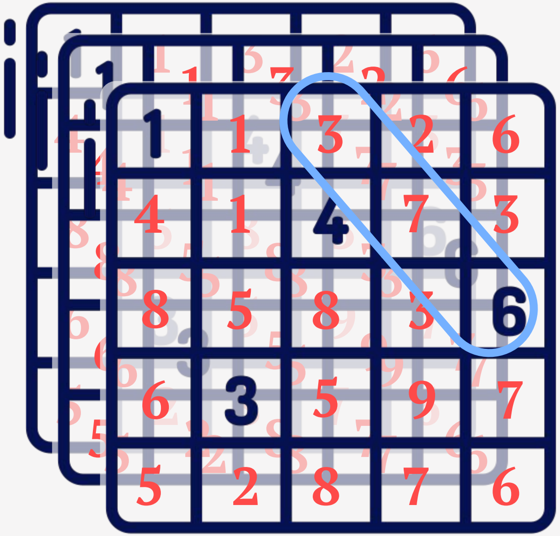
Verifier posts another (random) sudoku puzzle

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

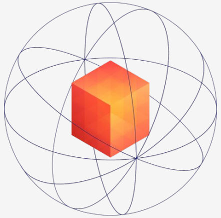
STARKWARE

Integrity Through Math

A sudoku-like set of constraints is implied by the statement proved, by ***x, y, P,*** and **#tx (=1,000,000)**



STARK



Claim: *Starting @ state hash* ***x****, after* ***1,000,000*** *txs processed by program* ***P,*** *reached state hash* ***y***

Magic (aka Math)
- Sampling constraints takes exponentially small time!
- Good proofs satisfy ALL constraints!
- A "proof" of a false claim satisfies < 1% of constraints!

"...a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware ..."

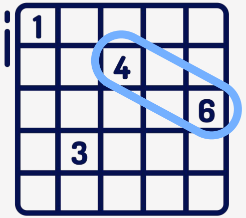
Prover submits solution

Verifier posts another (random) sudoku puzzle

Verifier samples and checks a single constraint

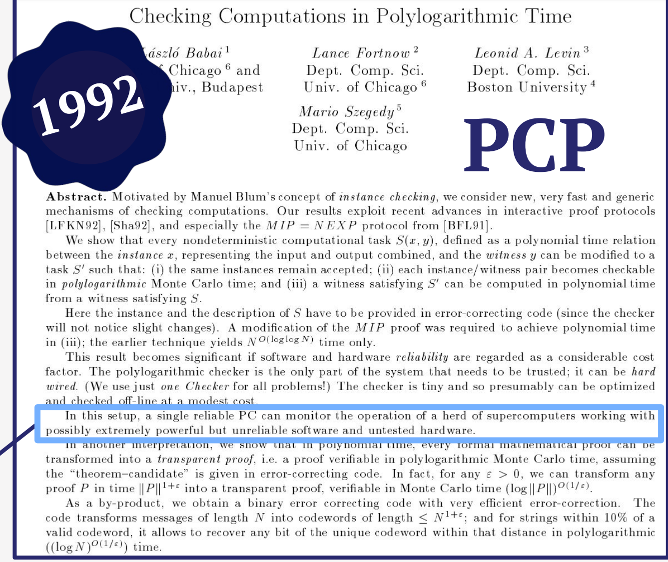
* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

Integrity Through Math



PCP

Integrity* via Math
(impractical)



Checking Computations in Polylogarithmic Time

ászló Babai [1]
f Chicago [6] and
niv., Budapest

Lance Fortnow [2]
Dept. Comp. Sci.
Univ. of Chicago [6]

Leonid A. Levin [3]
Dept. Comp. Sci.
Boston University [4]

Mario Szegedy [5]
Dept. Comp. Sci.
Univ. of Chicago

Abstract. Motivated by Manuel Blum's concept of *instance checking*, we consider new, very fast and generic mechanisms of checking computations. Our results exploit recent advances in interactive proof protocols [LFKN92], [Sha92], and especially the $MIP = NEXP$ protocol from [BFL91].

We show that every nondeterministic computational task $S(x, y)$, defined as a polynomial time relation between the *instance* x, representing the input and output combined, and the *witness* y can be modified to a task S' such that: (i) the same instances remain accepted; (ii) each instance/witness pair becomes checkable in *polylogarithmic* Monte Carlo time; and (iii) a witness satisfying S' can be computed in polynomial time from a witness satisfying S.

Here the instance and the description of S have to be provided in error-correcting code (since the checker will not notice slight changes). A modification of the MIP proof was required to achieve polynomial time in (iii); the earlier technique yields $N^{O(\log\log N)}$ time only.

This result becomes significant if software and hardware *reliability* are regarded as a considerable cost factor. The polylogarithmic checker is the only part of the system that needs to be trusted; it can be *hard wired*. (We use just *one Checker* for all problems!) The checker is tiny and so presumably can be optimized and checked off-line at a modest cost.

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with possibly extremely powerful but unreliable software and untested hardware.

In another interpretation, we show that in polynomial time, every formal mathematical proof can be transformed into a *transparent proof*, i.e. a proof verifiable in polylogarithmic Monte Carlo time, assuming the "theorem–candidate" is given in error-correcting code. In fact, for any $\varepsilon > 0$, we can transform any proof P in time $\|P\|^{1+\varepsilon}$ into a transparent proof, verifiable in Monte Carlo time $(\log\|P\|)^{O(1/\varepsilon)}$.

As a by-product, we obtain a binary error correcting code with very efficient error-correction. The code transforms messages of length N into codewords of length $\le N^{1+\varepsilon}$; and for strings within 10% of a valid codeword, it allows to recover any bit of the unique codeword within that distance in polylogarithmic $((\log N)^{O(1/\varepsilon)})$ time.

"…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …"

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]

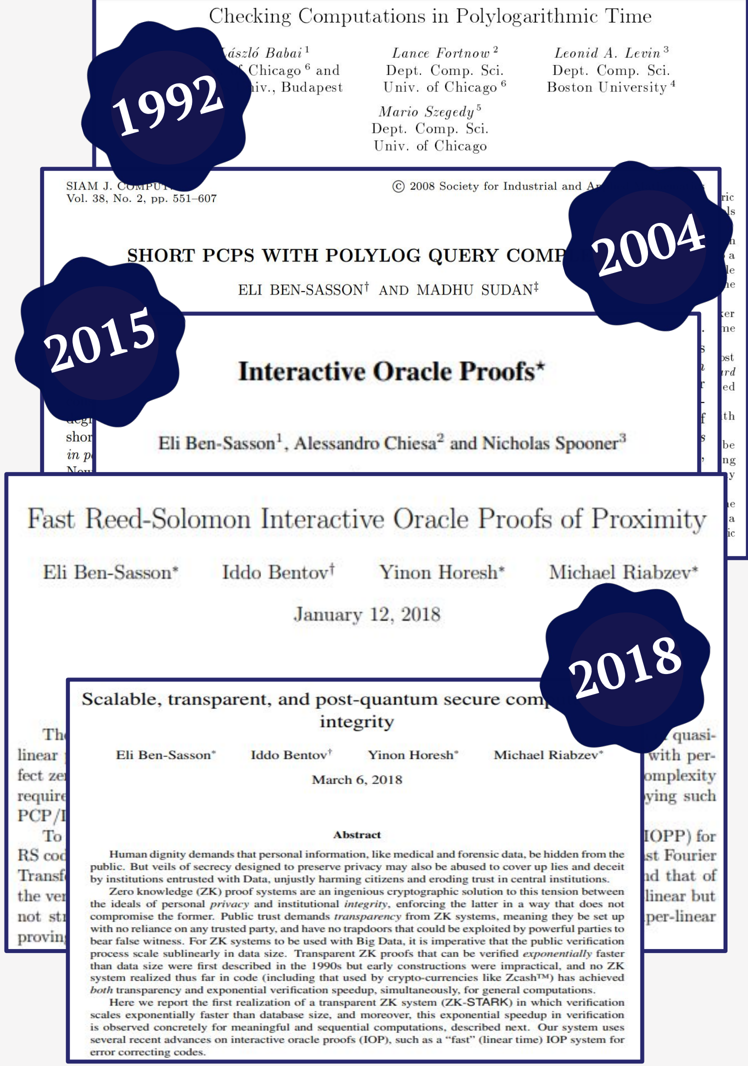
Integrity Through Math

STARK

Integrity* via Math
(~~im~~practical)

> *“…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …”*

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]



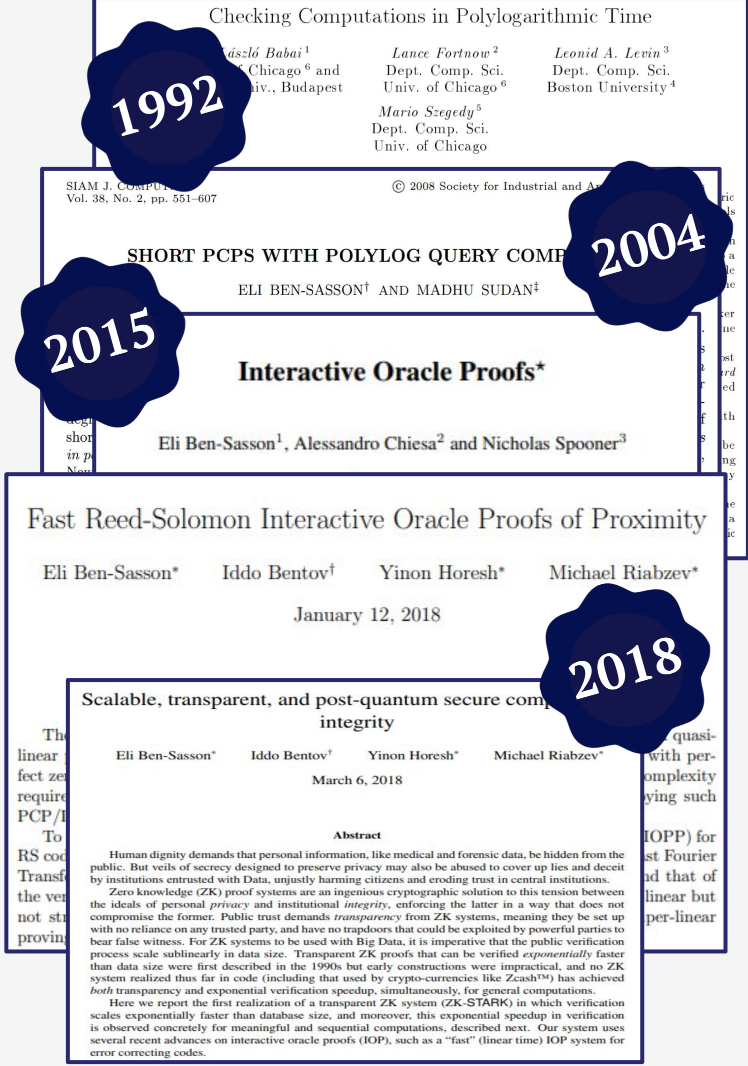
Integrity Through Math

STARK

Integrity* via Math
(~~im~~practical)

> *“…a single reliable PC can monitor the operation of a herd of supercomputers with powerful but unreliable software and untested hardware …”*

* *Integrity means doing the right thing, even when no one is watching* [C.S. Lewis]



Arithmetization

Arithmetization

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Claim:** Starting @ state hash **x**, after **1,000,000** txs processed by program **P,** reached state hash **y

Arithmetization

STARKWARE

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Pre-arithmetization claim

Starting @ state hash ***x****, after* ***1,000,000*** *txs processed by program* ***P****, reached state hash* ***y***

Arithmetization

STARKWARE

Arithmetization Converts ("reduces") Computational Integrity problems to problems about local relations between a bunch of polynomials

Pre-arithmetization claim	**Reduction**
Starting @ state hash ***x****, after* ***1,000,000*** *txs processed by program* ***P****, reached state hash* ***y***	*produces 2 polynomials:* ***Q(X,Y,T,W), R(X)*** *and degree bound* ***d***

Arithmetization

STARKWARE

Arithmetization Converts (“reduces”) Computational Integrity problems to problems about local relations between a bunch of polynomials

Pre-arithmetization claim	Reduction	Post-arithmetization claim
Starting @ state hash ***x****, after* ***1,000,000*** *txs processed by program* ***P****, reached state hash* ***y***	*produces 2 polynomials:* ***Q(X,Y,T,W), R(X)*** *and degree bound* ***d***	*I know 4 polynomials of degree* ***d*** *- A(x), B(x), C(x), D(X) - such that:* $Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)$

Arithmetization

Arithmetization Converts (“reduces”) Computational Integrity problems to problems about local relations between a bunch of polynomials

Pre-arithmetization claim	Reduction	Post-arithmetization claim	Theorem
*Starting @ state hash **x**, after **1,000,000** txs processed by program **P**, reached state hash **y***	*produces 2 polynomials: **Q(X,Y,T,W), R(X)** and degree bound **d***	*I know 4 polynomials of degree **d** - A(x), B(x), C(x), D(X) - such that:* *Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)*	*If A, B, C, D do not satisfy THIS,* *then nearly all x expose Bob’s lie*

Arithmetization

STARKWARE

Assuming Theorem, we get a scalable proof system for Bob's original claim:

1. Apply reduction, ask Bob to provide access to A,B,C,D of degree-d
2. Sample random x and accept Bob's claim iff equality holds for this x

New problem: Force Bob to
(1) commit to degree d polynomials, then
(2) answer queries to the precommitted polys

Post-arithmetization claim

I know 4 polynomials of degree d - $A(x)$, $B(x)$, $C(x)$, $D(X)$ - such that:

$Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)$

Theorem

If A, B, C, D do not satisfy THIS,

then nearly all x expose Bob's lie

Arithmetization

STARKWARE

Assuming Theorem, we get a scalable proof system for Bob's original claim:

1. Apply reduction, ask Bob to provide access to A,B,C,D of degree-d
2. Sample random x and accept Bob's claim iff equality holds for this x

Post-arithmetization claim

I know 4 polynomials of degree d - A(x), B(x), C(x), D(X) - such that:

*Q(X, A(X), B(X+1), C(2*X))=D(X) * R(X)*

Theorem

If A, B, C, D do not satisfy THIS,

then nearly all x expose Bob's lie

New problem: Force Bob to
(1) commit to degree *d* polynomials, then
(2) answer queries to the precommitted polys

AIR (Algebraic Intermediate Representation)

Computation:

A set of rules for how to sequentially evolve a state, starting with an input and ending up with an output.

AIR (Algebraic Intermediate Representation)

STARKWARE

python:

```
a = 1
b = 0
for i in range(n):
    a, b = a + b, a
return a
```

AIR (Algebraic Intermediate Representation)

STARKWARE

python:

```
a = 1
b = 0
for i in range(n):
    a, b = a + b, a
return a
```

trace:

$a_0=1$	$b_0=0$
$a_1=1$	$b_1=1$
$a_2=2$	$b_2=1$
$\vdots$	$\vdots$
a_n	b_n

AIR (Algebraic Intermediate Representation)

python:

```
a = 1
b = 0
for i in range(n):
    a, b = a + b, a
return a
```

trace:

$a_0=1$	$b_0=0$
$a_1=1$	$b_1=1$
$a_2=2$	$b_2=1$
$\vdots$	$\vdots$
a_n	b_n

constraints:

$$a_0 = 1$$

$$b_0 = 0$$

$$a_{i+1} = a_i + b_i$$

$$b_{i+1} = a_i$$

$$a_n = fib(n)$$

AIR

STARKWARE

python:

```python
a = 1
b = 0
for i in range(n):
    a, b = a + b, a
return a
```

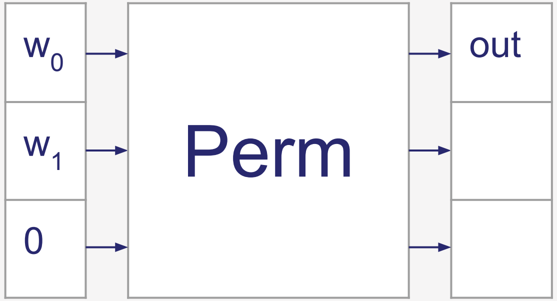
trace:

$a_0=1$	$b_0=0$
$a_1=1$	$b_1=1$
$a_2=2$	$b_2=1$
$\vdots$	$\vdots$
a_n	b_n

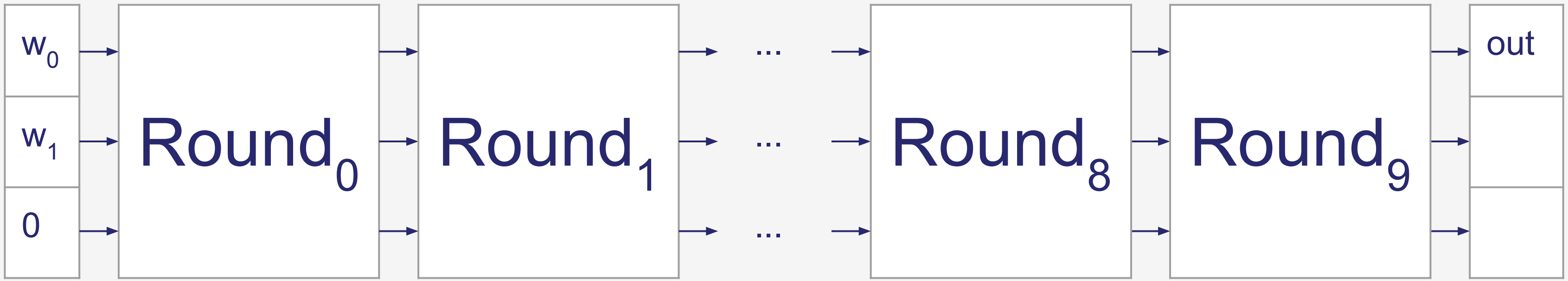
constraints: / domain:

constraints:	domain:
$a_0 = 1$	First row
$b_0 = 0$	First row
$a_{i+1} = a_i + b_i$	$0 \leq i < n$
$b_{i+1} = a_i$	$0 \leq i < n$
$a_n = fib(n)$	Row n

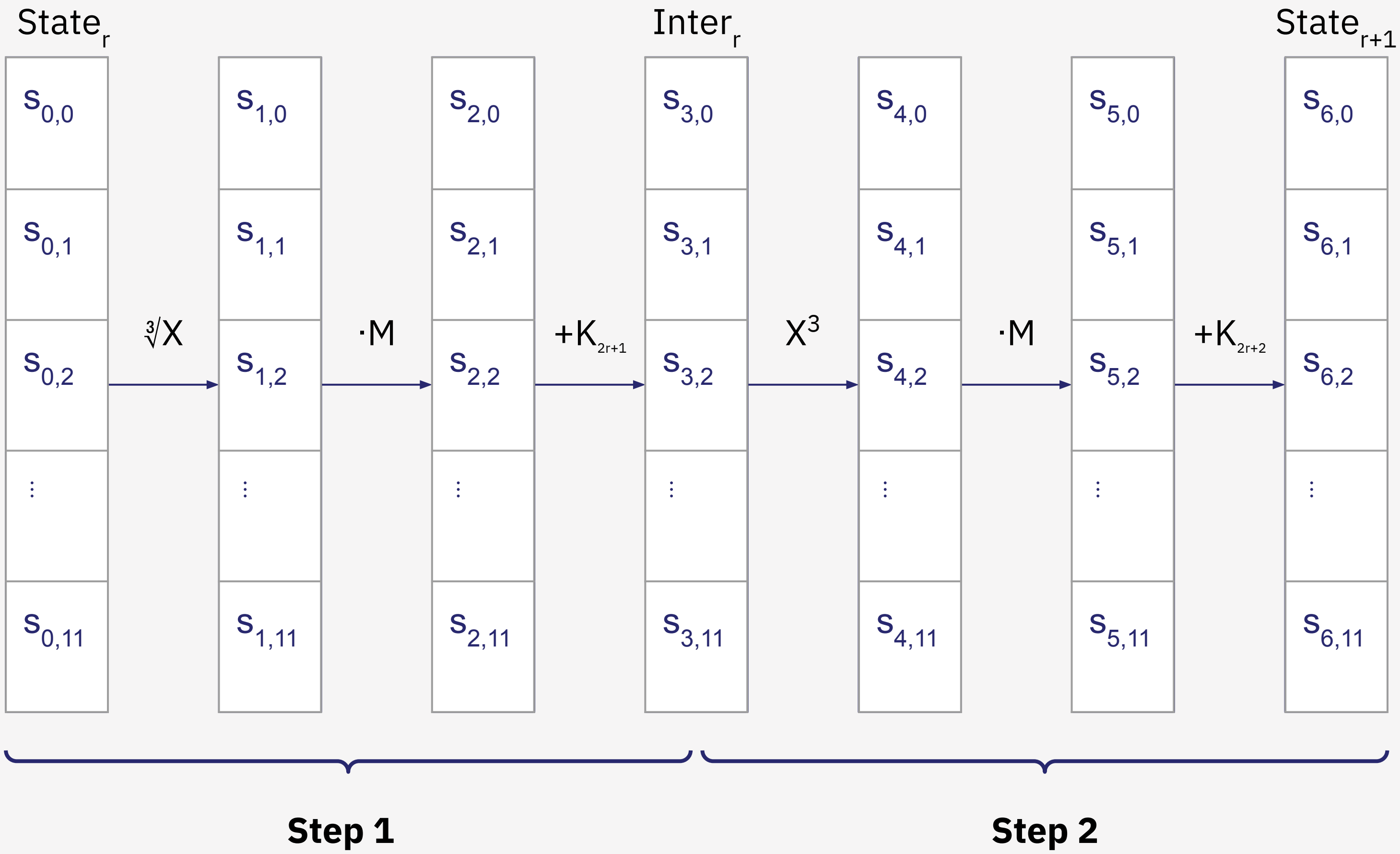
Rescue hash function



Rescue hash function



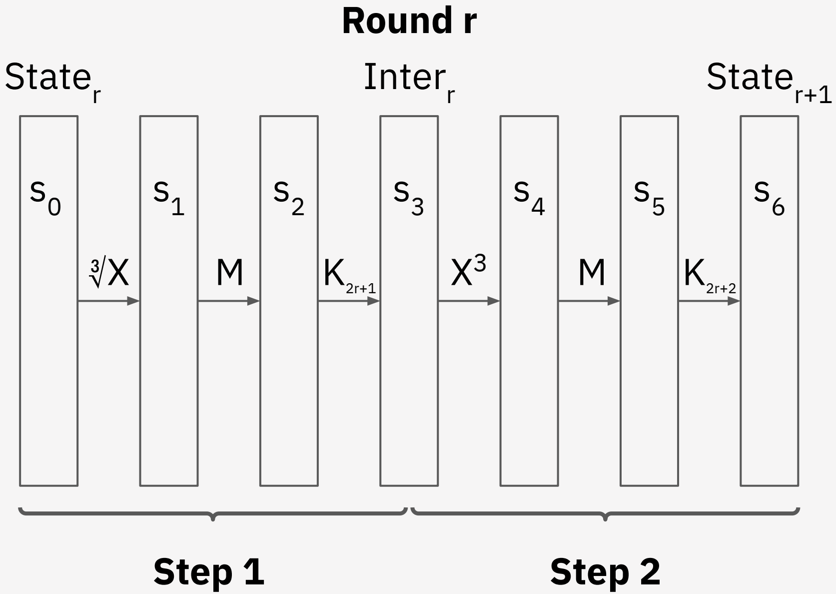
Round r



State_r
Inter_r
State_r+1
s0,0
s0,1
s0,2
s0,11
s1,0
s1,1
s1,2
s1,11
s2,0
s2,1
s2,2
s2,11
s3,0
s3,1
s3,2
s3,11
s4,0
s4,1
s4,2
s4,11
s5,0
s5,1
s5,2
s5,11
s6,0
s6,1
s6,2
s6,11
∛X
·M
+K2r+1
X3
·M
+K2r+2
Step 1
Step 2

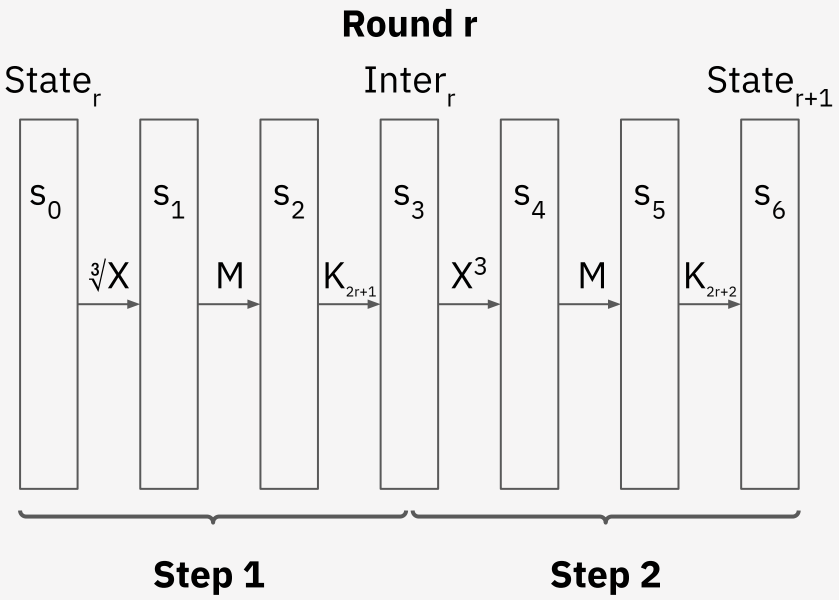
Rescue AIR

$$s_{1,i}^3 = s_{0,i}$$



Rescue AIR

$$s_{1,i}^3 = s_{0,i}$$
$$s_{2,i} = \sum_j M_{ij} s_{1,j} \qquad (s_2 = M \cdot s_1)$$

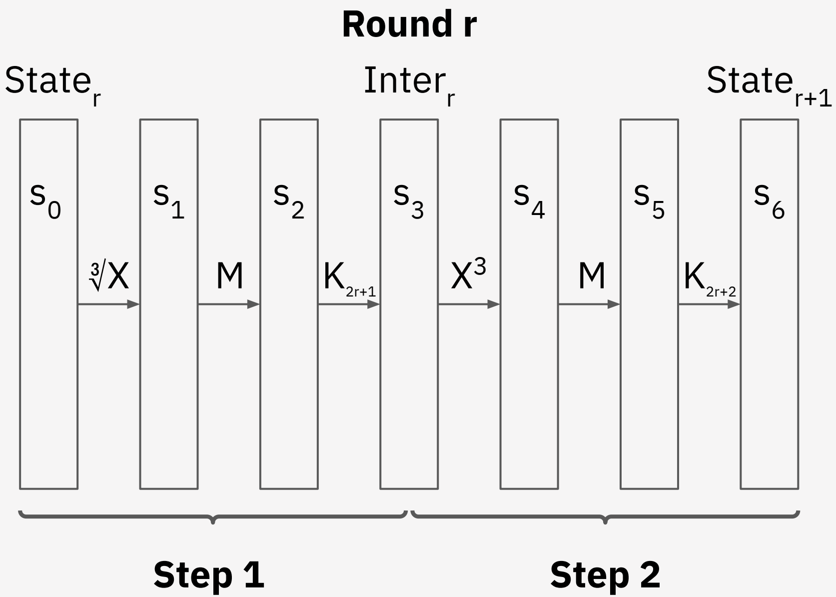


Rescue AIR

$$s_{1,i}^3 = s_{0,i}$$

$$s_{2,i} = \sum_j M_{ij} s_{1,j} \qquad (s_2 = M \cdot s_1)$$

$$s_3 = s_2 + K_{2r+1}$$



Rescue AIR

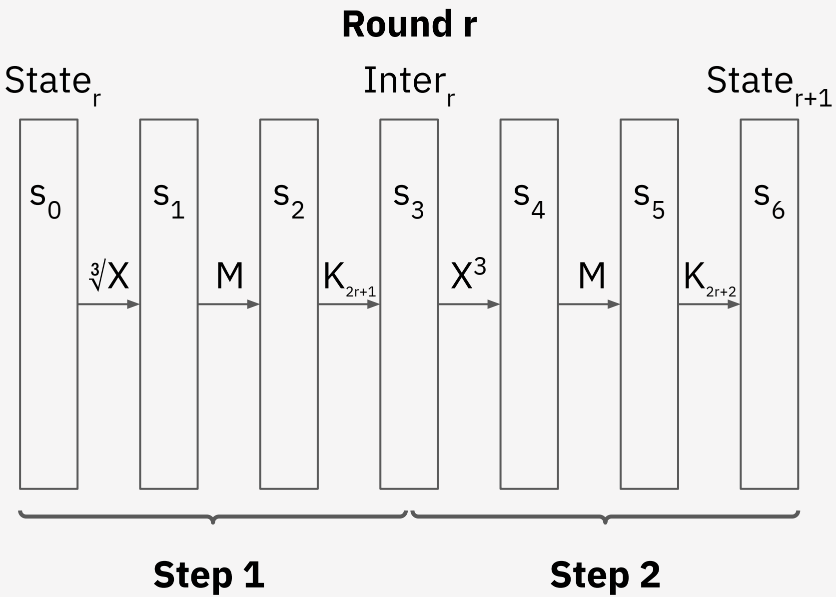
STARKWARE

$$s_{1,i}^3 = s_{0,i}$$

$$s_{2,i} = \sum_j M_{ij} s_{1,j} \qquad (s_2 = M \cdot s_1)$$

$$s_3 = s_2 + K_{2r+1}$$

$$s_{4,i} = s_{3,i}^3$$



Rescue AIR

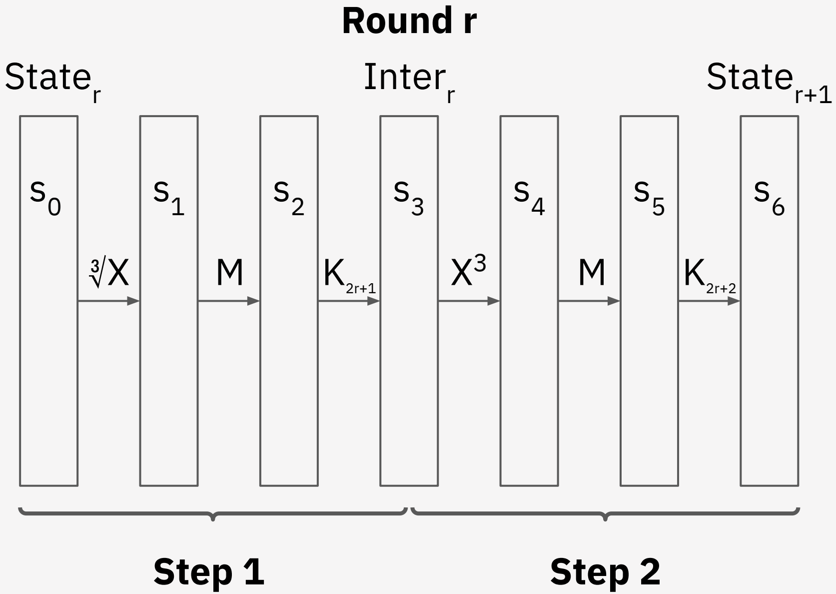
$$s_{1,i}^3 = s_{0,i}$$

$$s_{2,i} = \sum_j M_{ij} s_{1,j} \qquad (s_2 = M \cdot s_1)$$

$$s_3 = s_2 + K_{2r+1}$$

$$s_{4,i} = s_{3,i}^3$$

$$s_5 = M \cdot s_4$$



Rescue AIR

$$s_{1,i}^3 = s_{0,i}$$

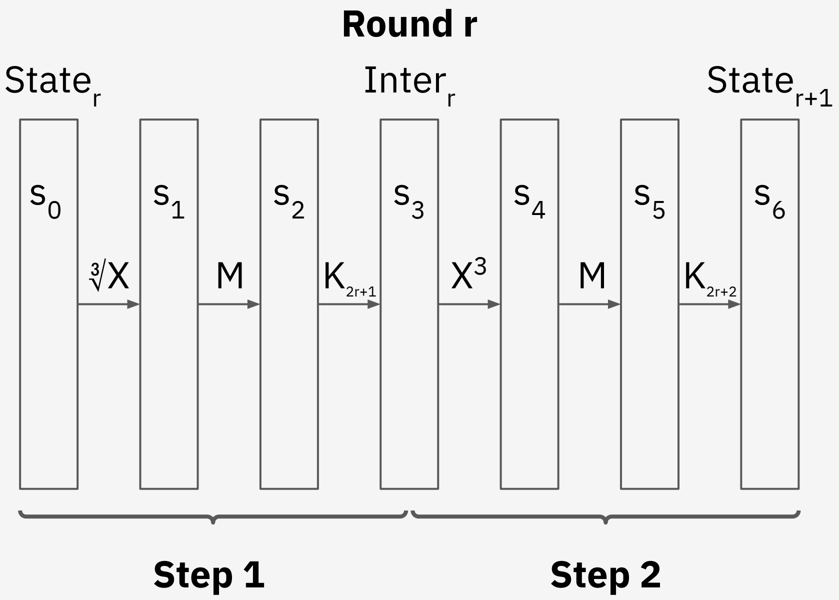
$$s_{2,i} = \sum_j M_{ij} s_{1,j} \qquad (s_2 = M \cdot s_1)$$

$$s_3 = s_2 + K_{2r+1}$$

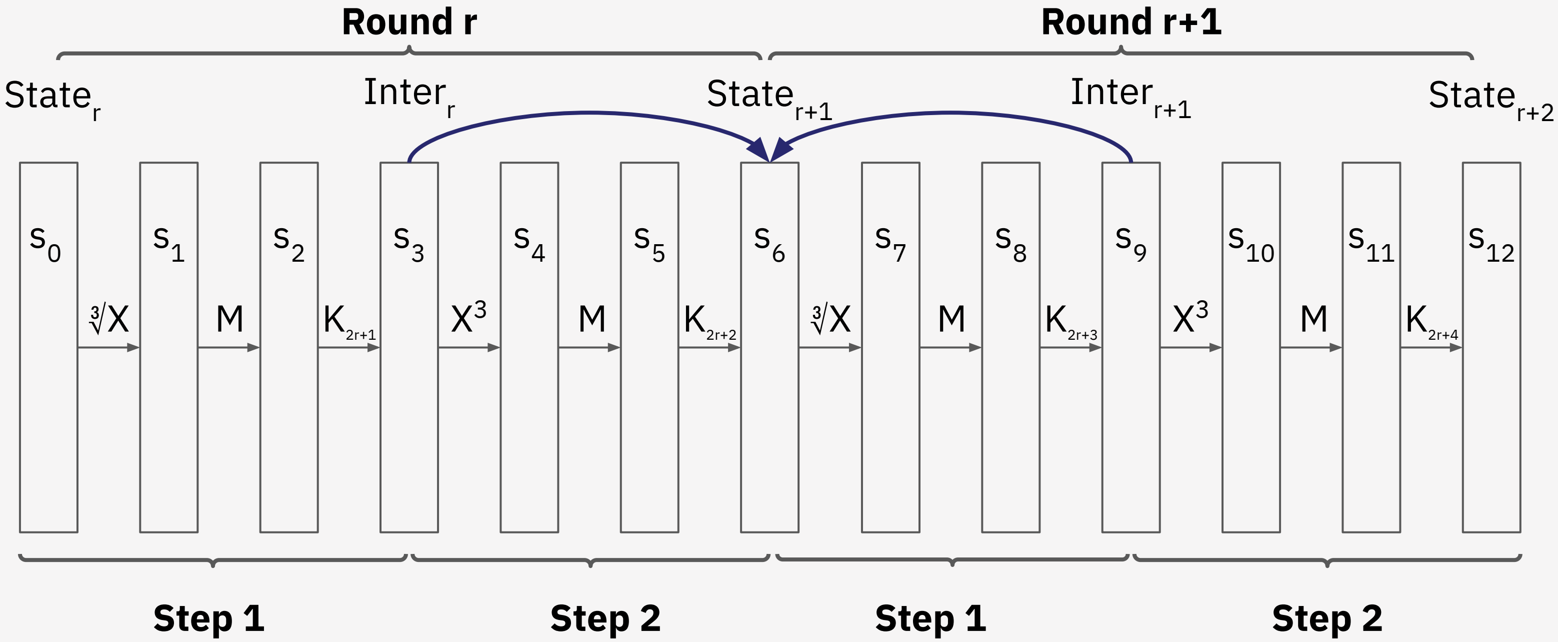
$$s_{4,i} = s_{3,i}^3$$

$$s_5 = M \cdot s_4$$

$$s_6 = s_5 + K_{2r+2}$$



Rescue AIR



$$\left(\sum_j M_{ij} s_{3,j}^3\right) + K_{2r+2,i} = s_{6,i} = \left(\sum_j (M^{-1})_{ij}(s_{9,j} - K_{2r+3,j})\right)^3$$

Wishlist for crypto primitives

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
    - Fib(n): t=n, w=2
    - Rescue: t= #rounds, w = |state|

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

Wish-list for Primitive P:

1. Minimize t x w, |F|, d (also s, e)
2. Minimize CPU time of computing P
3. Make it field agnostic, and safe!

AIR Parameters: *t, w, F, d, s, e*

- Execution trace size: ***t*** rows, ***w*** columns
  - Fib(n): t=n, w=2
  - Rescue: t= #rounds, w = |state|
- Trace entries in field ***F***. Which field?
  - **Any** field! [BCKL22] (FFT-friendly better)
- Constraints
  - Maximal degree: ***d***
  - # constraints: ***s***
  - Enforcement domain complexity: ***e***

Wish-list for Primitive P:

1. Minimize t x w, |F|, d (also s, e)
2. Minimize CPU time of computing P
3. Make it field agnostic, and safe!

ethSTARK (Rescue):

1. t x w = 120, log |F| ~ 62, d=3
2. CPU time large, esp in large F (cube root!)
3. Relatively safe, AES-like (say the experts)

Poseidon in Starknet/Ex:

1. t x w = 226, log |F|~252, d=2
2. CPU time better than Rescue (100 microsec)
3. Relatively safe (though less than Rescue)

AIR Parameters: *t, w, F, d, s, e*

Keccak:

1. t x w = 70,000-90,000, log |F| ~ 252, d=2
2. CPU time amazing (1 microsecond)
3. Very safe

Wish-list for Primitive P:

1. Minimize t x w, |F|, d (also s, e)
2. Minimize CPU time of computing P
3. Make it field agnostic, and safe!

ethSTARK (Rescue):

1. t x w = 120, log |F| ~ 62, d=3
2. CPU time large, esp in large F (cube root!)
3. Relatively safe, AES-like (say the experts)

Poseidon in Starknet/Ex:

1. t x w = 226, log |F|~252, d=2
2. CPU time better than Rescue (100 microsec)
3. Relatively safe (though less than Rescue)

STARK-friendly crypto primitives wish-list

Eli Ben-Sasson

April 2023