Zero-Knowledge Proofs II
zk-SNARKs

Oct. 21, 2019

Overview

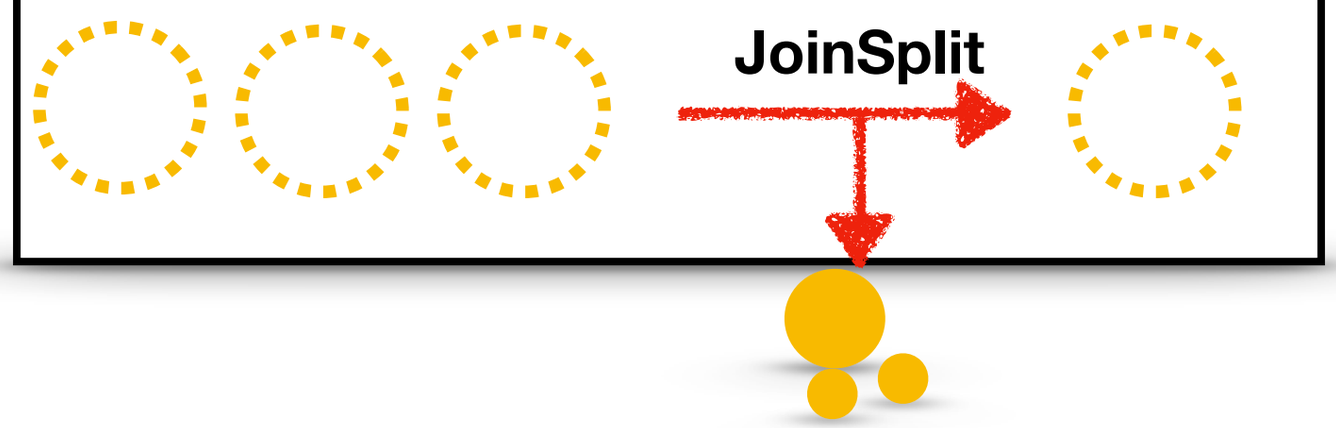
- Recap Lelantus
  - One efficient way to do 1-in-N proofs
- zk-SNARKs
  - A general way to prove anything in Zero-Knowledge
    - (if you don’t know how to do it any other way, use zk-SNARKs)

Lelantus

Used serial#

```
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
…
```

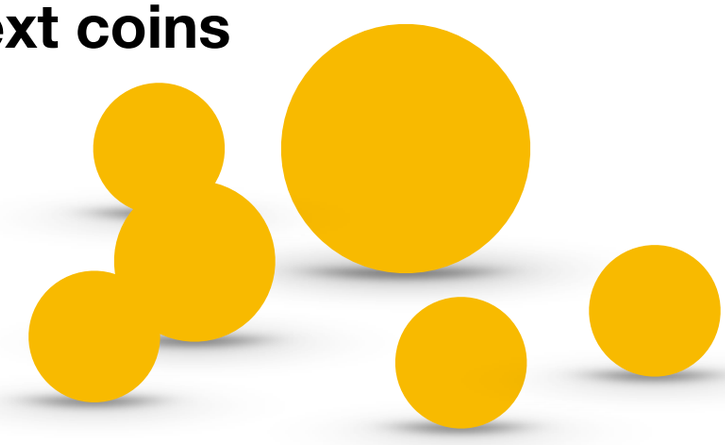
hidden coins (Pedersen Commitments)



Mint

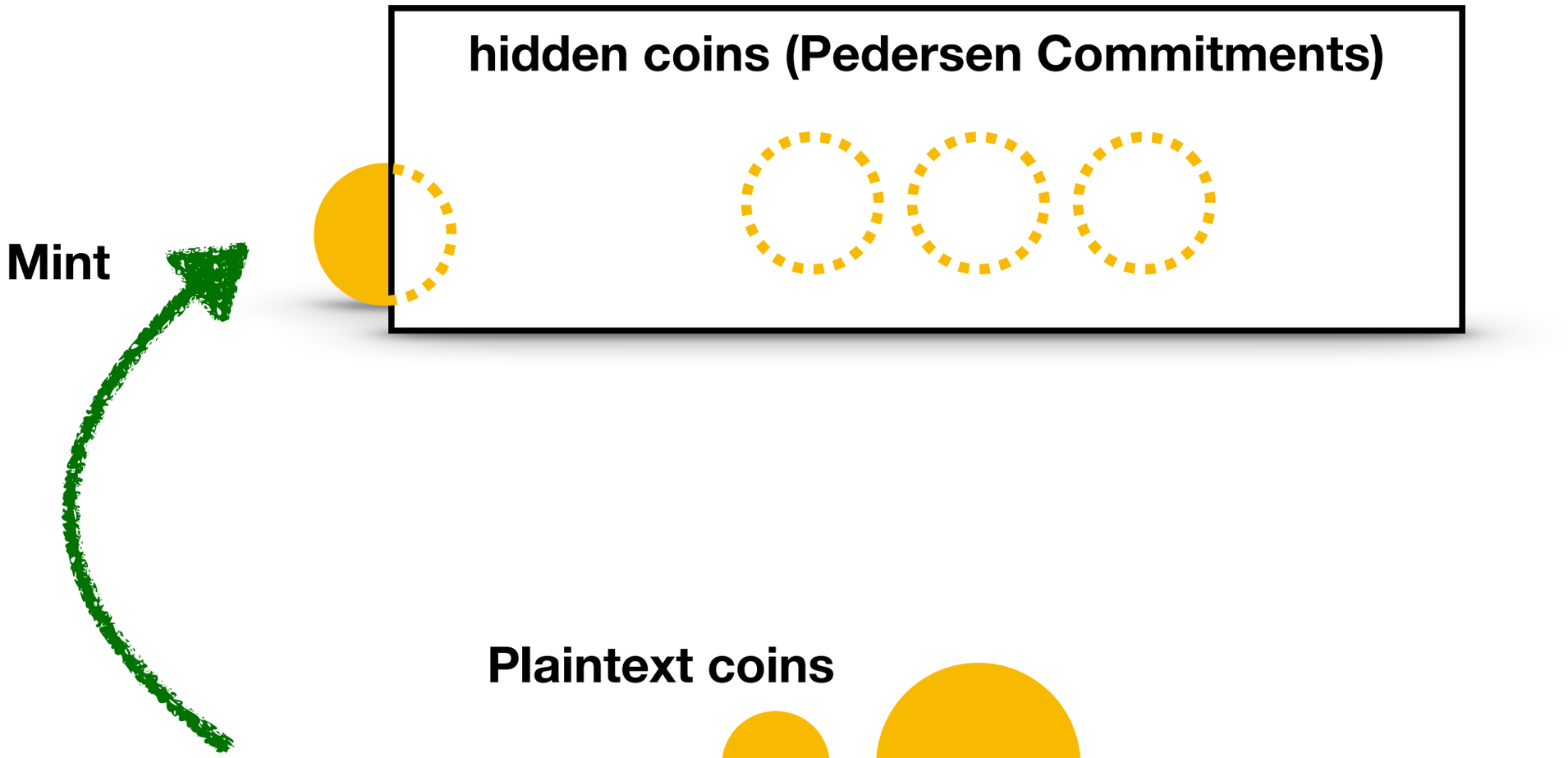
Spend

Plaintext coins

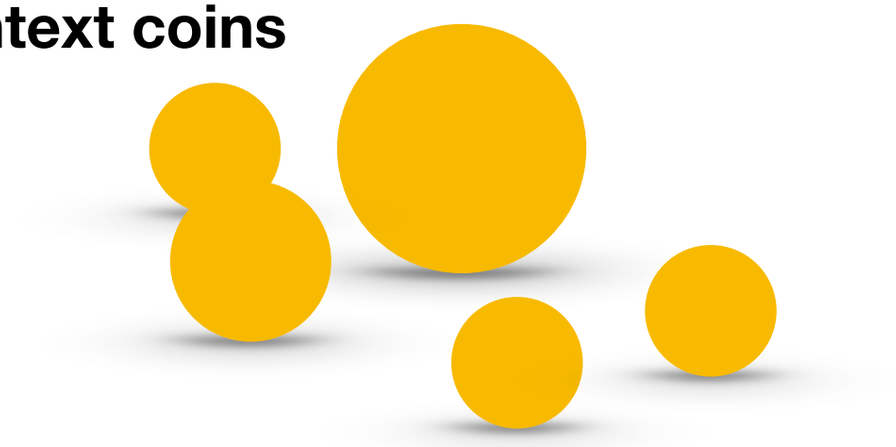


Lelantus Mint

Proof: Pedersen Commitment valid



Plaintext coins

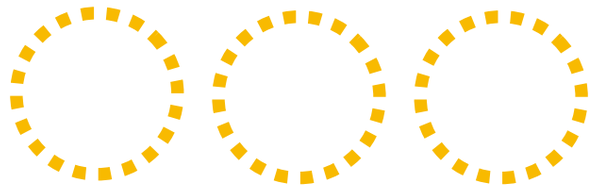


Lelantus Spend

Used serial#

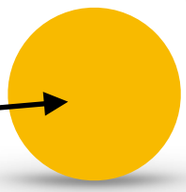
```
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
73f143adf73708de491ff9d
…
```

hidden coins (Pedersen Commitments)



Spend

Proof: Serial number
amount

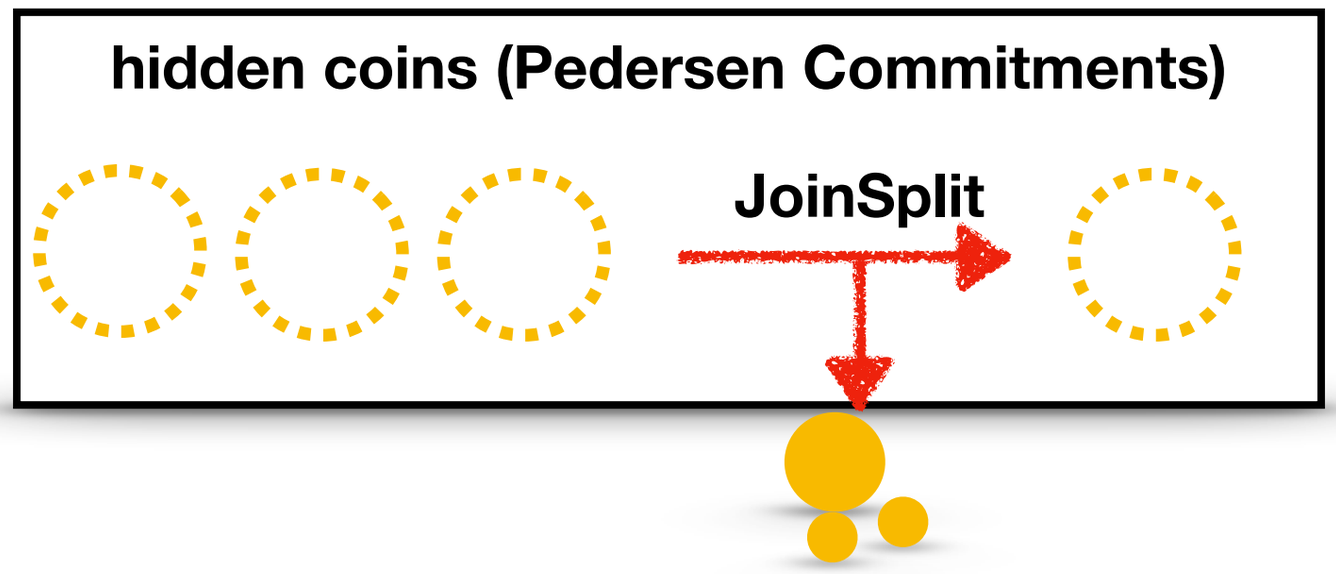


Plaintext coins

Lelantus JoinSplit

Used serial#

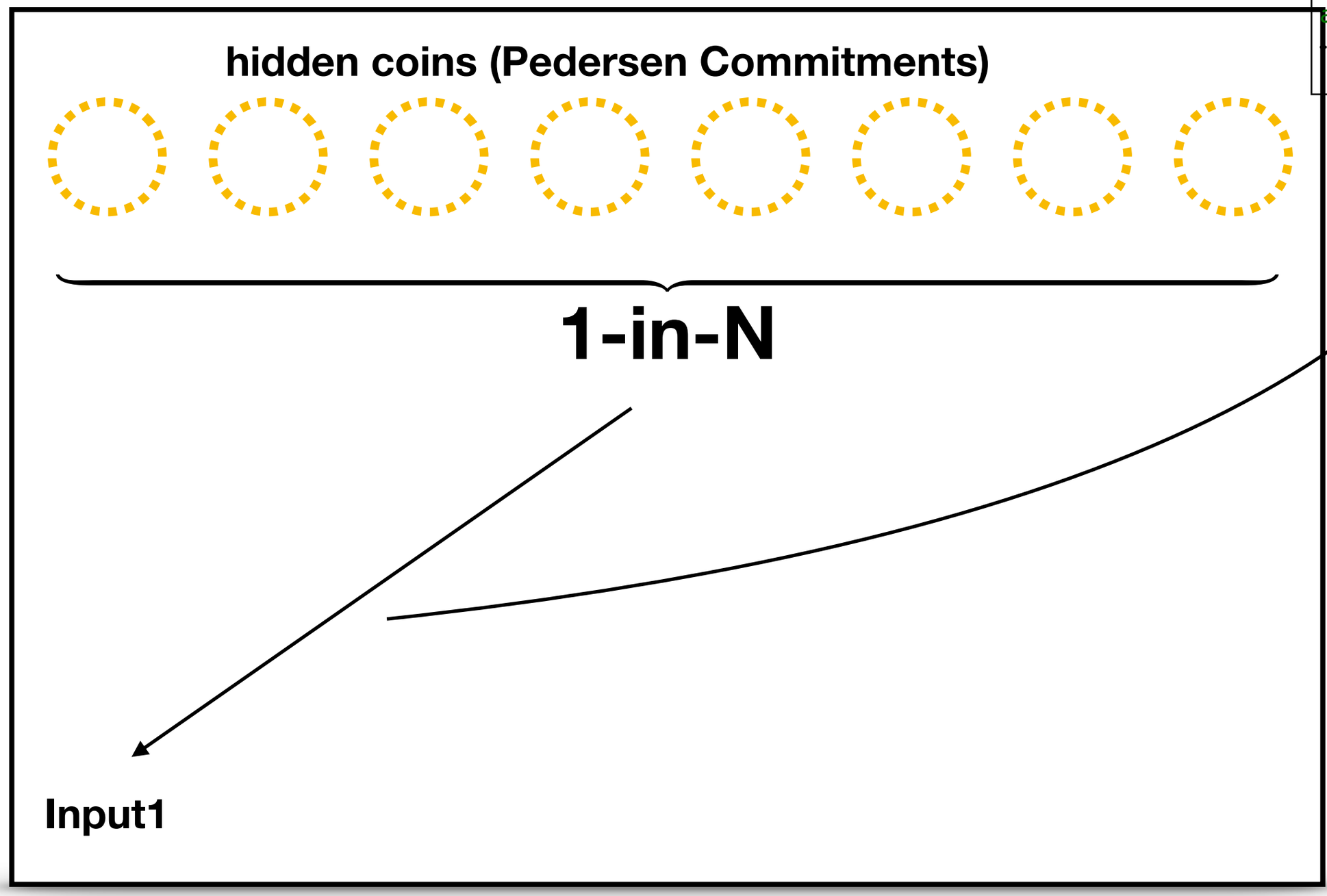
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
…



Lelantus JoinSplit

Used serial#

e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
...

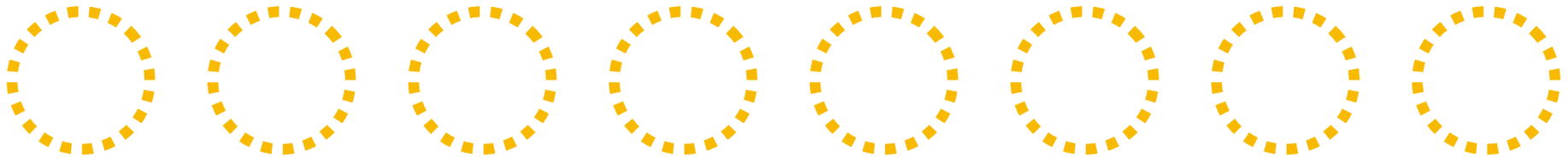


Lelantus JoinSplit

Used serial#

e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
73f143adf73708de491ff9d
...

hidden coins (Pedersen Commitments)



1-in-N

Input1 + Input2

Lelantus JoinSplit

Used serial#

```
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
73f143adf73708de491ff9d
95b96411c8dc99f6be2b443
...
```

hidden coins (Pedersen Commitments)

1-in-N

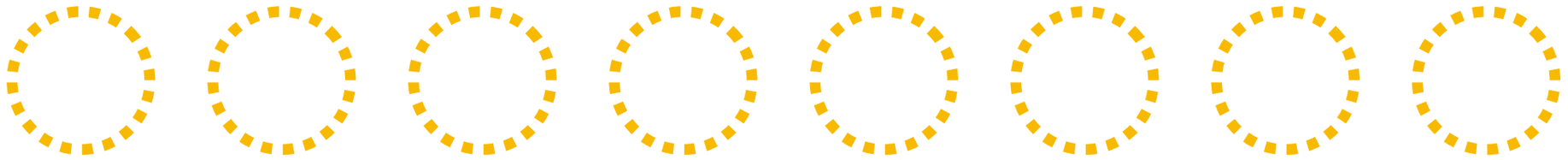
Input1 + Input2 + Input3

Lelantus JoinSplit

Used serial#

e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
73f143adf73708de491ff9d
95b96411c8dc99f6be2b443
...

hidden coins (Pedersen Commitments)



Proof: valid Pedersen Commitment

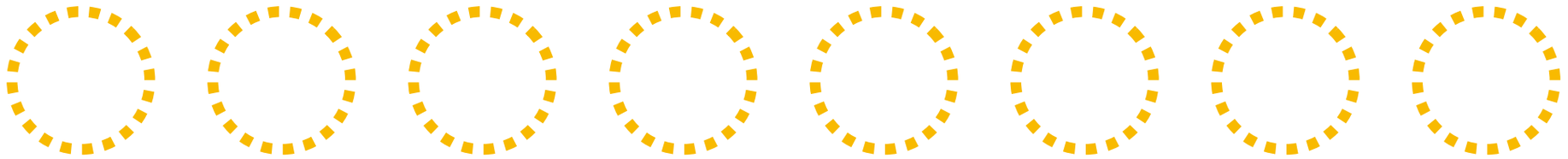
Input1 + Input2 + Input3 + Output1

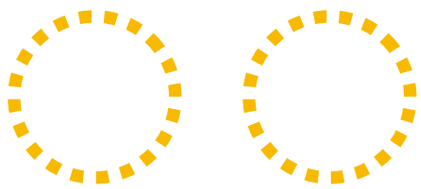
Lelantus JoinSplit

Used serial#

```
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
73f143adf73708de491ff9d
95b96411c8dc99f6be2b443
...
```

hidden coins (Pedersen Commitments)





Proof: valid Pedersen Commitment

Input1 + Input2 + Input3 + Output1 + Output2

Lelantus JoinSplit

Used serial#

```
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
73f143adf73708de491ff9d
95b96411c8dc99f6be2b443
...
```

hidden coins (Pedersen Commitments)

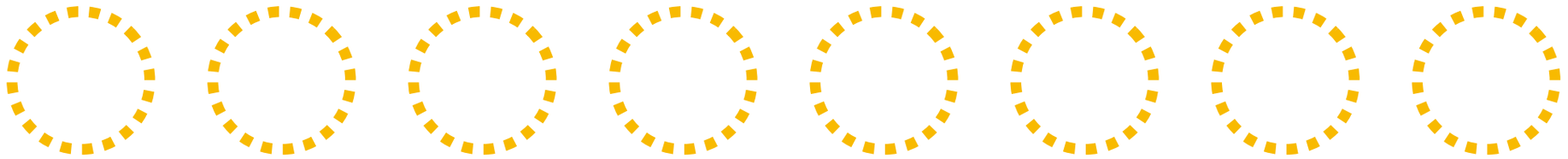
Input1 + Input2 + Input3 + Output1 + Output2 + ExtraCashOut

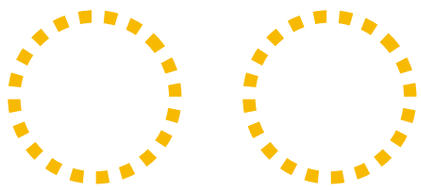
Lelantus JoinSplit

Used serial#

```
e8fb04ab61cfdd9ab54d9b1
ea6a1728b274a7e3c667523
cdcb04f2b45a6dd3c13e90c
050cf72a2c4ff1f4df4084a
5a35670340e4107632e4629
f59cc4cef45a8063e4afb65
2d28e9bb87f78a5c0b6b008
1c4433bd43daafa3806759b
4f587540daa9bcb002b3699
a6e434bb929b8c4d9adf1fb
73f143adf73708de491ff9d
95b96411c8dc99f6be2b443
...
```

hidden coins (Pedersen Commitments)





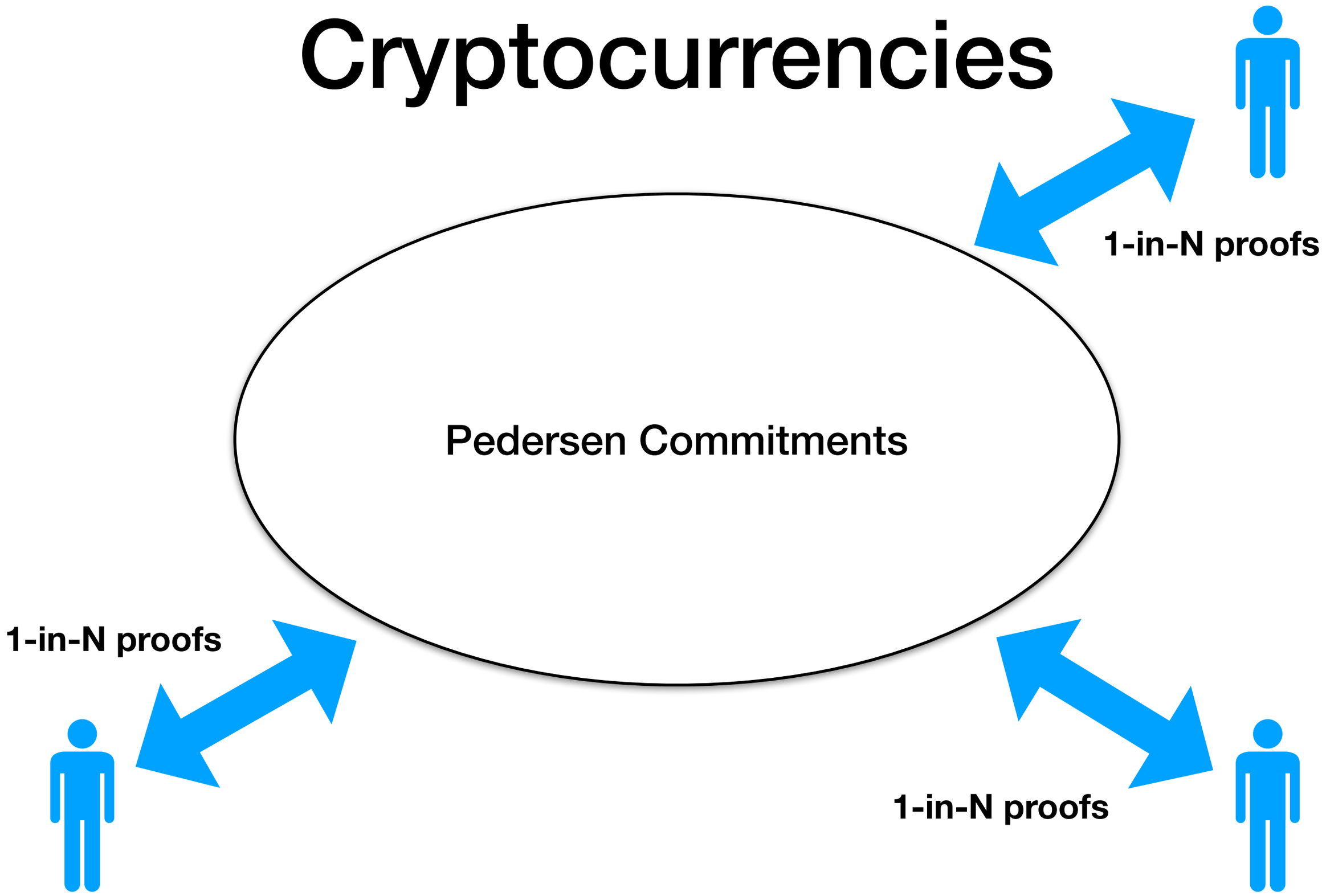
Proof of valid transaction: (c, d, α)

Input1 + Input2 + Input3 + Output1 + Output2 + ExtraCashOut=T

If $c = \mathscr{H}(T \,|\, cT + dH + \alpha F)$, then T can be described as a factor of only H and F:

- does not have any G components = no money was created or destroyed

Anonymous Cryptocurrencies



zk-SNARKs

Zero-Knowledge
Succinct Non-Interactive Argument of Knowledge

zk-SNARK

- A general purpose zero-knowledge tool for any computation
  - Need to prove that you know the pre-image of a hash
    - => zk-SNARK
  - Need to build a secret cryptocurrency (e.g. Zerocoin)
    - => zk-SNARK
  - Need to prove that you know XYZ?
    - => zk-SNARK

zk-SNARK

- A general purpose zero-knowledge tool for any computation
- Very useful, highly relevant, but quite complicated
- We will give a high-level overview of how this works
    - a complete discussion could be an entire semester

zk-SNARK

- Perform the computation storing any intermediate value
  - All values of all variables, called the *witness*
- We encode the witness as a polynomial function $w(x)$
- We show that $w(x)$ can divide $c(x)$, the constraint polynomial
  - Only if $a(x) \cdot w(x) = c(x)$ is the *witness* valid
  - If the witness is valid, the program was executed correctly

zk-SNARK

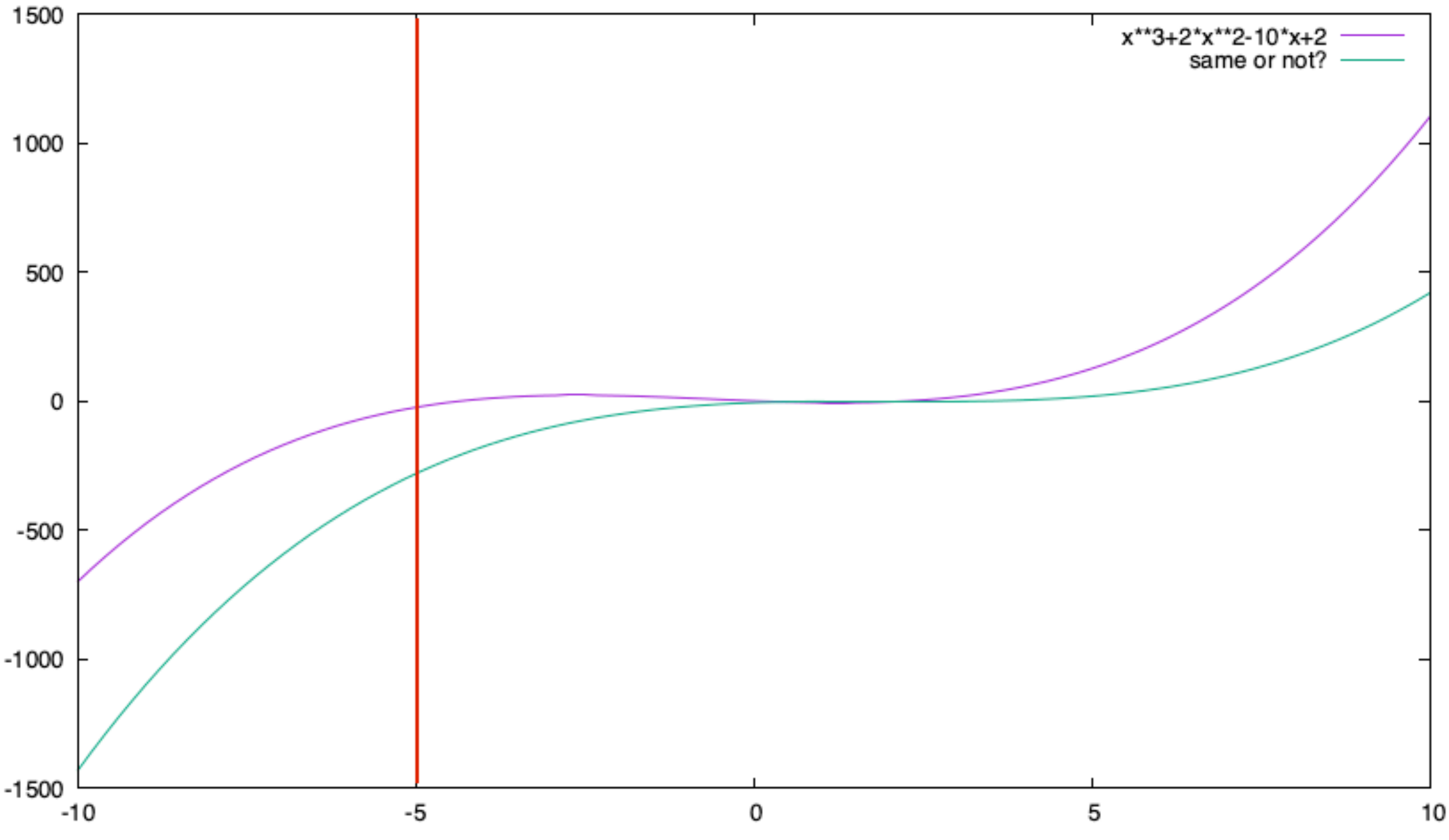
- The trick is showing $a(x) \cdot w(x) = c(x)$
  - We show $a(x) \cdot w(x) - c(x) = 0$ at a secret position x
    - Encode polynomials a, w, c and position x via ECC

zk-SNARK

- Alice wants to convince Bob that she executed a program
- Alice creates the witness $w(x)$
- Bob choses a position x_{eval} and verifies

$$a(x_{\text{eval}}) - w(x_{\text{eval}}) - c(x_{\text{eval}}) = 0$$

Evaluating two polynomials at a random position is enough to check for equality



All that’s left to do

- Represent the proof of executing a program as a proof that I know a divisor of a polynomial

- Encode the proof $w(x)a(x) = c(x)$ in ECC

Proof of Knowledge of Division

- Points can be added and multiplied
- given 3 points $A = aG, B = bB, C = cG, D = dG$, I can encode the polynomial $ax^3 + bx^2 + cx + d$

$$x^3A + x^2B + xC + D$$

- The details on how to do the polynomial checks are beyond the scope of today's lecture

Proof of execution

- Computers run on hardware
    - Theoretically, we can simulate any program with looking at the binary circuits

1. Represent the computation as a binary circuit
    - Or algebraic circuit for pure math problems
2. Reduction to a *Rank 1 Constraint System* (R1CS)
3. Representation as a Quadratic Assignment Problem (QAP)

Program Representation

- Assume we want to prove that we know a value x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

- Other applications:
  - I know a value x so that $\mathscr{H}(x) = 23d23e1\ldots$ (proof of knowledge of preimage)

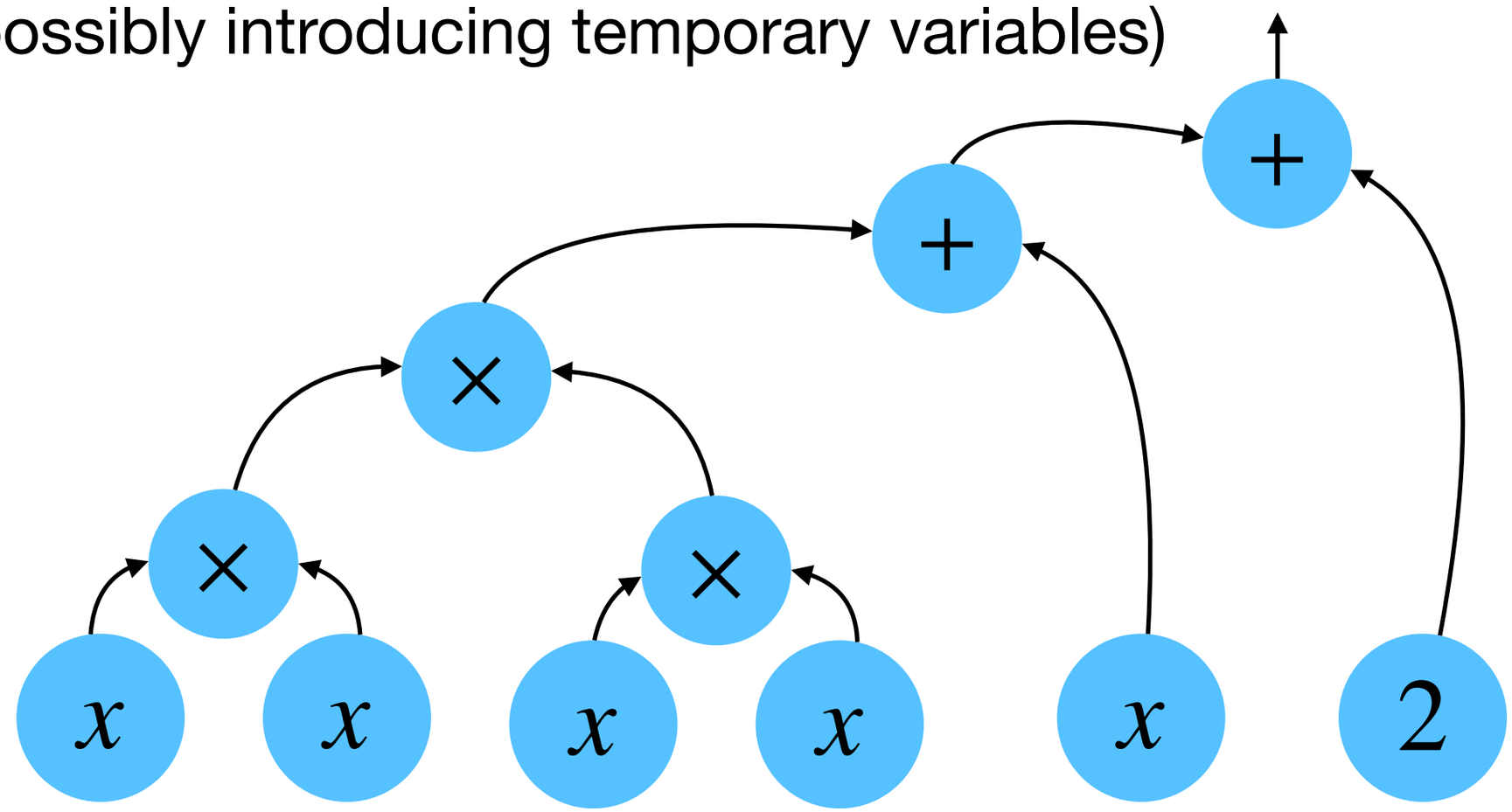
  - A secret blockchain: I know a transaction T so that
    - T is the blockchain
    - I know the private key/serial# of T
    - The output is not yet spend

Flattening the computation

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

- We can verify all basic operations (+,-,*,assignment)
  - We need to represent the computation as a sequence of basic steps (possibly introducing temporary variables)

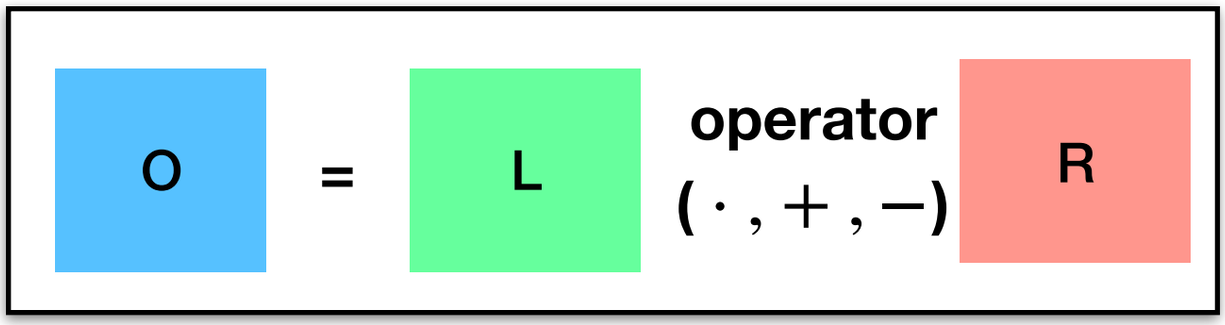
1. $a = x \cdot x$
2. $b = a \cdot a$
3. $c = b + x$
4. $\text{out} = c + 2$



Flattening the computation

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

1. $a = x \cdot x$
2. $b = a \cdot a$
3. $c = b + x$
4. out $= c + 2$



List of all variables:

$1, x, a, b, c$, **out**

1 instead of 2 as basic unit for all constants

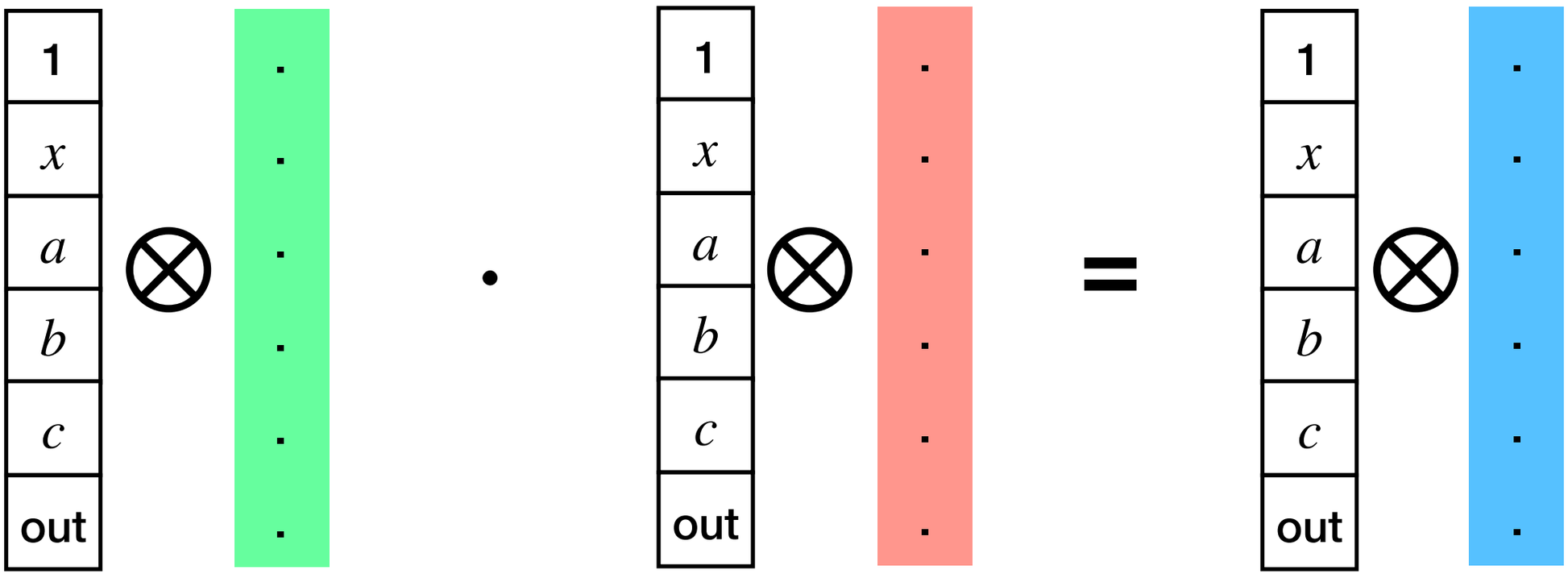
Each operation as vector

L **operator** $(\cdot, +, -)$ R = O

List of all variables:

$1, x, a, b, c$, out

We can generalize all operations using 3 vectors:



Each operation as vector

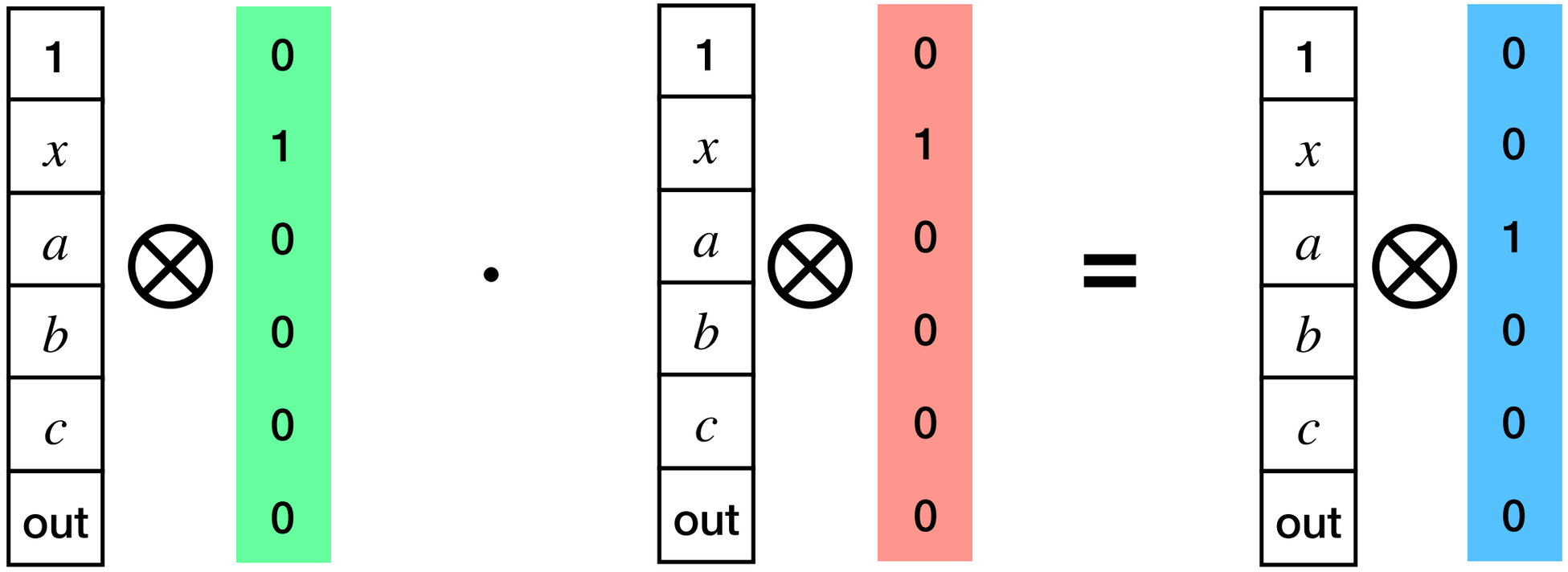
L **operator** $(\cdot, +, -)$ R = O

List of all variables:

$1, x, a, b, c$, **out**

We can generalize all operations using 3 vectors:

Multiplication: (Example $a = x \cdot x$)



$1 \cdot x \quad \cdot \quad 1 \cdot x \quad = \quad a \cdot 1$

Each operation as vector

L	operator $(\cdot, +, -)$	R	=	O

List of all variables:

$1, x, a, b, c$, out

We can generalize all operations using 3 vectors:

Addition: (Example $b = x + 7$)

1		1		1		7		1		0		1		0
x		0		x		1		x		0		x		0
a	$\otimes$	0	$\cdot$	a	$\otimes$	0	=	a	$\otimes$	0		a	$\otimes$	0
b		0		b		0		b		1		b		1
c		0		c		0		c		0		c		0
out		0		out		0		out		0		out		0

$1 \cdot 1 \quad \cdot \quad (1 \cdot 7) + (x \cdot 1) \quad = \quad b \cdot 1$

Each operation as vector

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

1. $a = x \cdot x$
2. $b = a \cdot a$
3. $c = b + x$
4. $\text{out} = c + 2$

L	operator $(\cdot, +, -)$	R	=	O

List of all variables:

$1, x, a, b, c,$ **out**

1	0		1	0		1	0
x	1		x	1		x	0
a	0	$\cdot$	a	0	=	a	1
b	0		b	0		b	0
c	0		c	0		c	0
out	0		out	0		out	0

Each operation as vector

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

1. $a = x \cdot x$
2. $b = a \cdot a$
3. $c = b + x$
4. $\text{out} = c + 2$

L	operator $(\cdot, +, -)$	R	=	O

List of all variables:

$1, x, a, b, c,$ **out**

1	0		1	0		1	0
x	0		x	0		x	0
a	1	$\cdot$	a	1	=	a	0
b	0		b	0		b	1
c	0		c	0		c	0
out	0		out	0		out	0

Each operation as vector

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

1. $a = x \cdot x$
2. $b = a \cdot a$
3. $c = b + x$
4. $\text{out} = c + 2$

L	operator $(\cdot, +, -)$	R	=	O

List of all variables:

$1, x, a, b, c,$ **out**

1	1		1	0		1	0
x	0		x	1		x	0
a	0	$\cdot$	a	0	=	a	0
b	0		b	1		b	0
c	0		c	0		c	1
out	0		out	0		out	0

Each operation as vector

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

1. $a = x \cdot x$
2. $b = a \cdot a$
3. $c = b + x$
4. $\text{out} = c + 2$

L	operator $(\cdot, +, -)$	R	=	O

List of all variables:

$1, x, a, b, c,$ **out**

1	1		1	2		1	0
x	0		x	0		x	0
a	0	$\cdot$	a	0	=	a	0
b	0		b	0		b	0
c	0		c	1		c	0
out	0		out	0		out	1

Summarized Constraints

Proof: We know x so that $x^4 + x + 2 = 86$ (hint $x = 3$)

1	0	0	1	1	0	0	0	2	0	0	0	0
x	1	0	0	0	1	0	1	0	0	0	0	0
a	0	1	0	0	0	1	0	0	1	0	0	0
b	0	0	0	0	0	0	1	0	0	1	0	0
c	0	0	0	0	0	0	0	1	0	0	1	0
out	0	0	0	0	0	0	0	0	0	0	0	1

1st constraint $x \cdot x = a$

3rd constraint $b + x = c$

Witness

- To proof that we executed the computation for $x^4 + x + 2 = 86$ (hint $x = 3$) we create the following witness

1	1
3	x
9	a
81	b
84	c
86	out

Witness fulfills all Constraints

1
x
a
b
c
out

0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	2
1	0	1	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$$(3 \cdot 1) \cdot (3 \cdot 1) = (9 \cdot 1)$$

1	0
3	1
9	0
81	0
84	0
86	0

$\cdot$

1	0
3	1
9	0
81	0
84	0
86	0

$=$

1	0
3	0
9	1
81	0
84	0
86	0

Witness fulfills all Constraints

1
x
a
b
c
out

0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	2
1	0	1	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$$(9 \cdot 1) \cdot (9 \cdot 1) = (81 \cdot 1)$$

1	0
3	0
9	1
81	0
84	0
86	0

$\cdot$

1	0
3	0
9	1
81	0
84	0
86	0

$=$

1	0
3	0
9	0
81	1
84	0
86	0

Witness fulfills all Constraints

1
x
a
b
c
out

0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	2
1	0	1	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$$(1 \cdot 1) \cdot (3 \cdot 1 + 81 \cdot 1) = (84 \cdot 1)$$

1	1
3	0
9	0
81	0
84	0
86	0

$\cdot$

1	0
3	1
9	0
81	1
84	0
86	0

$=$

1	0
3	0
9	0
81	0
84	1
86	0

Witness fulfills all Constraints

1
x
a
b
c
out

0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	2
1	0	1	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$$(1 \cdot 1) \cdot (2 \cdot 1 + 84 \cdot 1) = (86 \cdot 1)$$

1	1
3	0
9	0
81	0
84	0
86	0

$\cdot$

1	2
3	0
9	0
81	0
84	1
86	0

$=$

1	0
3	0
9	0
81	0
84	0
86	1

Witness

Naive approach

- Alice: Create the 3 groups of vectors as constraints
- Bob: Runs the computation, creates the witness
- Alice: checks that the witness fulfills all constraints

- Works, but is slow and uses lots of data

Quadratic Assignment Problem

- We encode the constraints and witnesses as polynomials

1
x
a
b
c
out

$L_1(t)$

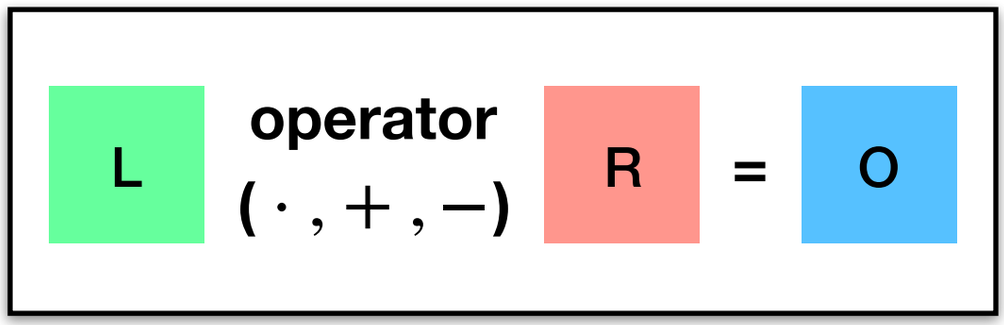
0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	2
1	0	1	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

0	0	0	0
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

Quadratic Assignment Problem

- We encode the constraints and witnesses as polynomials



1	$L_1(t)$	$R_1(t)$	$O_1(t)$
x	$L_x(t)$	$R_x(t)$	$O_x(t)$
a	$L_a(t)$	$R_a(t)$	$O_a(t)$
b	$L_b(t)$	$R_b(t)$	$O_b(t)$
c	$L_c(t)$	$R_c(t)$	$O_c(t)$
out	$L_{\text{out}}(t)$	$R_{\text{out}}(t)$	$O_{\text{out}}(t)$

Quadratic Assignment Problem

1	$L_1(t)$	$R_1(t)$	$O_1(t)$
x	$L_x(t)$	$R_x(t)$	$O_x(t)$
a	$L_a(t)$	$R_a(t)$	$O_a(t)$
b	$L_b(t)$	$R_b(t)$	$O_b(t)$
c	$L_c(t)$	$R_c(t)$	$O_c(t)$
out	$L_{\text{out}}(t)$	$R_{\text{out}}(t)$	$O_{\text{out}}(t)$

- Value $t = 1,2,3,4$ encodes the step of the program
- We show that for the polynomial $L\_, R\_, O\_$:

$$L\_(t) \otimes \text{Witness} \cdot R\_(t) \otimes \text{Witness} = O\_(t) \otimes \text{Witness}$$

  - This implies that it hold for all t, i.e. all computational steps were done correctly

Quadratic Assignment Problem

1
x
a
b
c
out

$L_1(t)$
$L_x(t)$
$L_a(t)$
$L_b(t)$
$L_c(t)$
$L_{\text{out}}(t)$

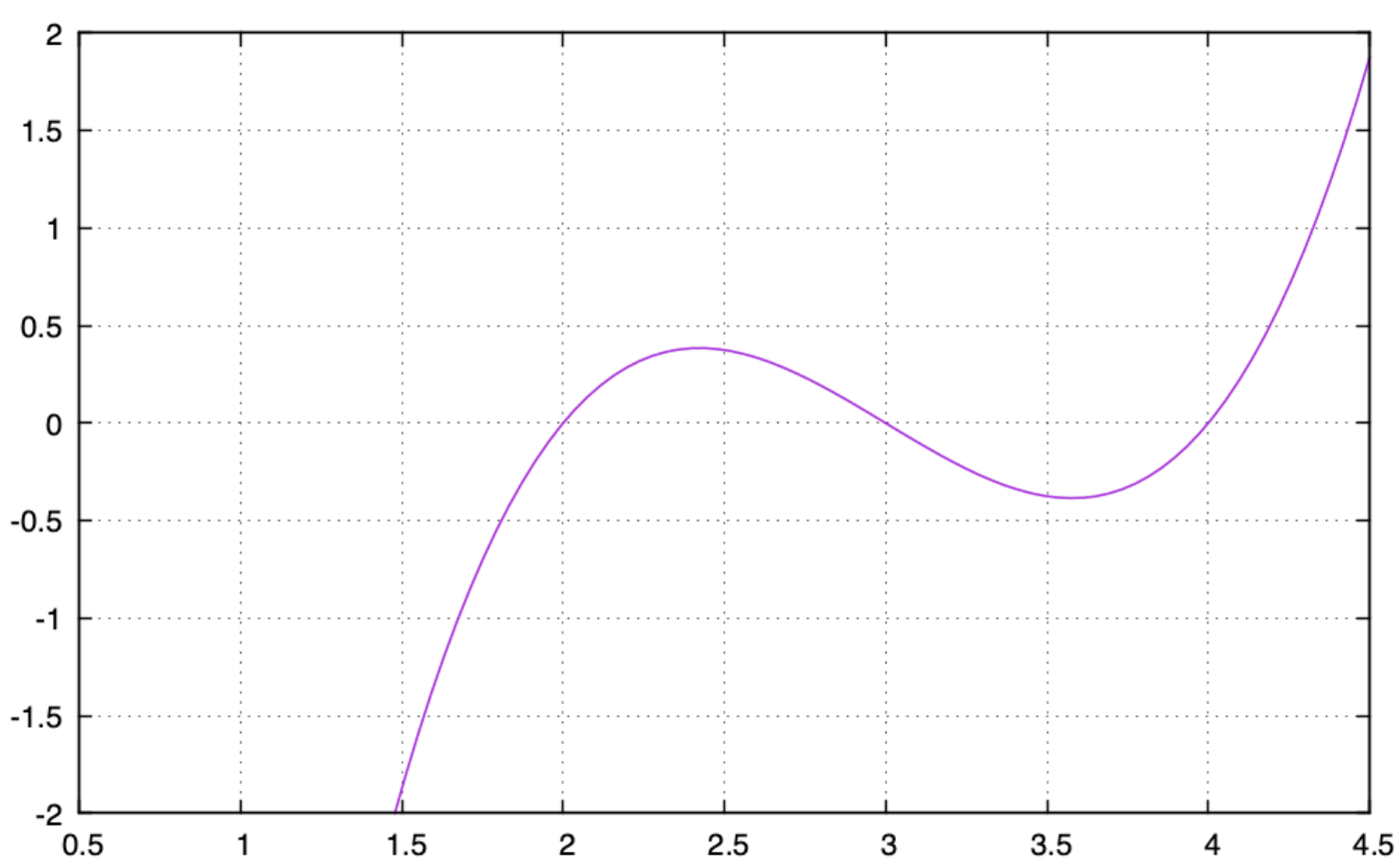
0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$$L_1(1) = 0 \qquad L_1(2) = 0 \qquad L_1(3) = 1 \qquad L_1(4) = 1$$

Creating Polynoms

$$\underline{P(1) = w}, \qquad P(2) = x, \qquad P(3) = y, \qquad P(4) = z$$

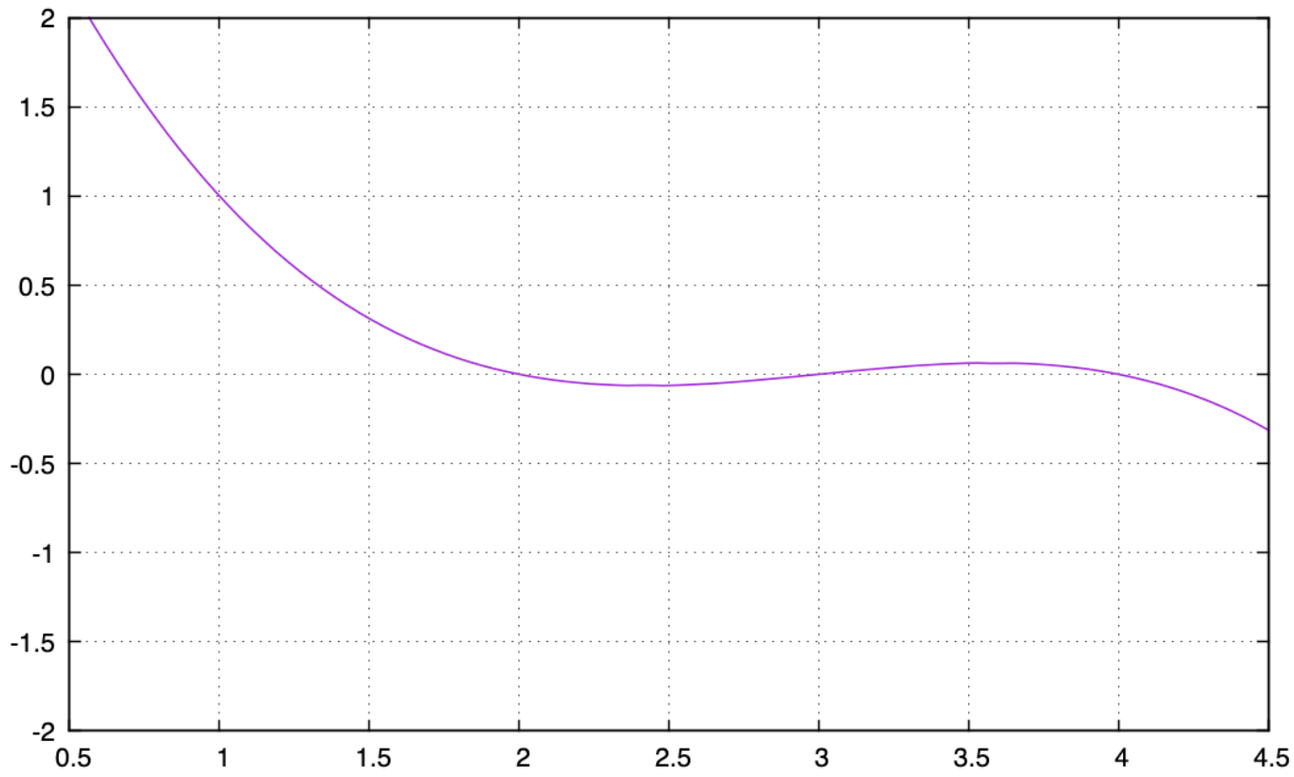
$$\text{buildingBlock}_1(t) = (t-2)(t-3)(t-4)$$



Creating Polynoms

$$P(1) = w, \qquad P(2) = x, \qquad P(3) = y, \qquad P(4) = z$$

$$\text{buildingBlock}_2(t) = \frac{(t-2)(t-3)(t-4)}{(1-2)(1-3)(1-4)}$$



1 at $t = 1$,
0 at $t = 2,3,4$

Creating Polynoms

- Assume we want

$$\underline{P(1) = w}, \qquad P(2) = x, \qquad P(3) = y, \qquad P(4) = z$$

$$\text{buildingBlock}_w(t) = w \cdot \frac{(t-2)(t-3)(t-4)}{(1-2)(1-3)(1-4)}$$

  - w at $t = 1$, 0 at $t = 2,3,4$

Creating Polynoms

- Assume we want

$$P(1) = w, \quad \underline{P(2) = x}, \quad P(3) = y, \quad P(4) = z$$

$$\text{buildingBlock}_x(t) = x \cdot \frac{(t-1)(t-3)(t-4)}{(2-1)(2-3)(2-4)}$$

  - x at $t = 2$, 0 at $t = 1,3,4$

Creating Polynoms

- Assume we want

$$P(1) = w, \quad P(2) = x, \quad \underline{P(3) = y}, \quad P(4) = z$$

$$\text{buildingBlock}_y(t) = y \cdot \frac{(t-1)(t-2)(t-4)}{(3-1)(3-2)(3-4)}$$

- y at $t = 3$, 0 at $t = 1,2,4$

Creating Polynoms

- Assume we want

$$P(1) = w, \quad P(2) = x, \quad P(3) = y, \quad \underline{P(4) = z}$$

$$\text{buildingBlock}_z(t) = z \cdot \frac{(t-1)(t-2)(t-3)}{(4-1)(4-2)(4-3)}$$

  - z at $t = 4$, 0 at $t = 1,2,3$

Creating Polynoms

- $P(t) = \text{buildingBlock}_w(t) + \text{buildingBlock}_x(t) +$

$$\text{buildingBlock}_y(t) + \text{buildingBlock}_z(t)$$

- $P(1) = w, \qquad P(2) = x, \qquad P(3) = y, \qquad P(4) = z$

Quadratic Assignment Problem

1
x
a
b
c
out

$L_1(t)$
$L_x(t)$
$L_a(t)$
$L_b(t)$
$L_c(t)$
$L_{\text{out}}(t)$

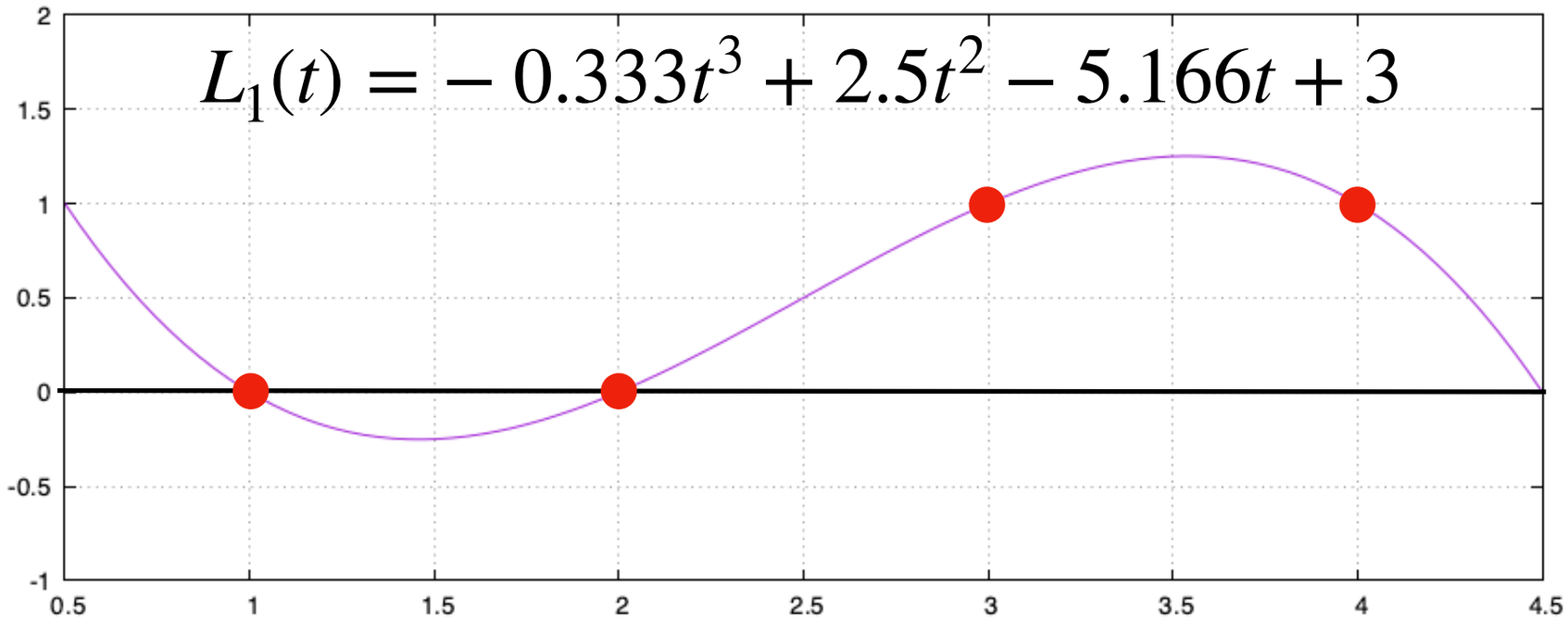
0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$$L_1(1) = 0$$

$$L_1(2) = 0$$

$$L_1(3) = 1$$

$$L_1(4) = 1$$



Quadratic Assignment Problem

1
x
a
b
c
out

$L_1(t)$
$L_x(t)$
$L_a(t)$
$L_b(t)$
$L_c(t)$
$L_{\text{out}}(t)$

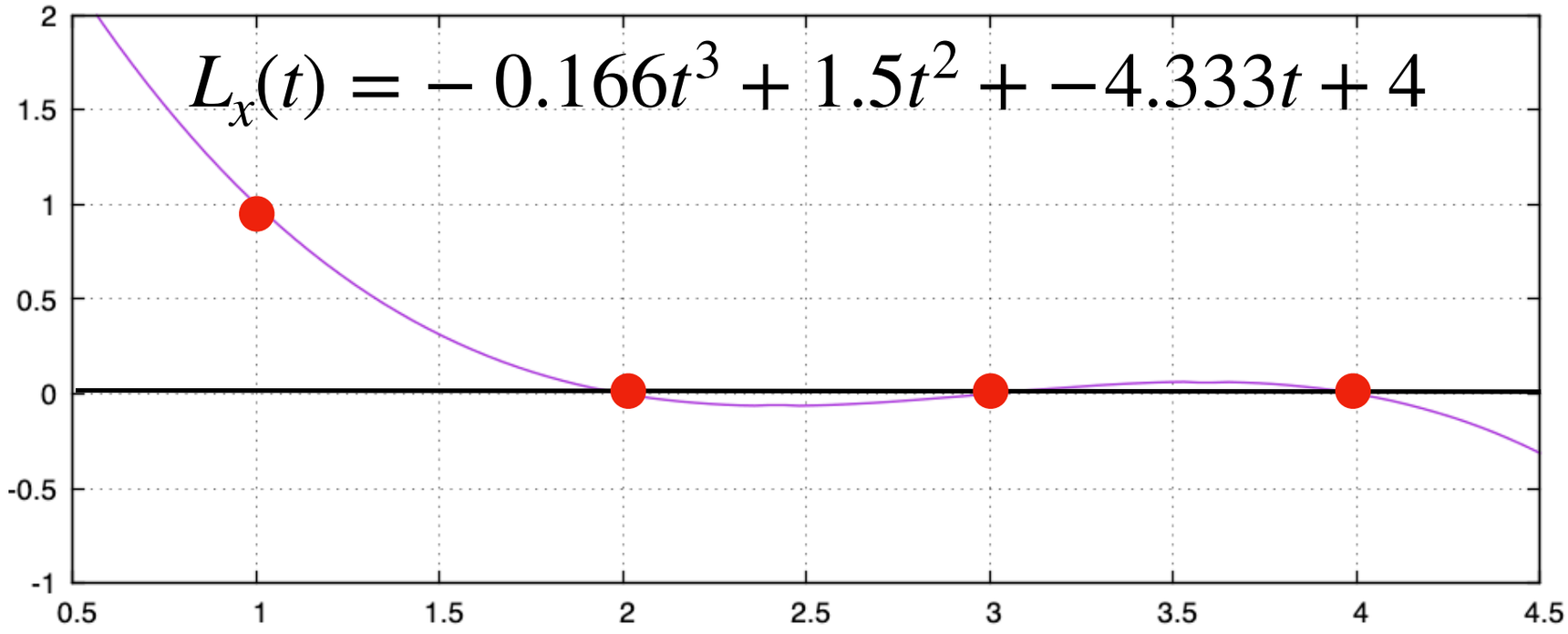
0	0	1	1
1	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$$L_x(1) = 1$$

$$L_x(2) = 0$$

$$L_x(3) = 0$$

$$L_x(4) = 0$$



Quadratic Assignment Problem

1	$L_1(t)$	0	0	1	1
x	$L_x(t)$	1	0	0	0
a	$L_a(t)$	0	1	0	0
b	$L_b(t)$	0	0	0	0
c	$L_c(t)$	0	0	0	0
out	$L_{\text{out}}(t)$	0	0	0	0

coefficients

t^3	t^2	t	1
-0.333	2.5	-5.166	3.0
-0.166	1.5	-4.333	4.0
0.5	-4.0	9.5	-6.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Quadratic Assignment Problem

1	$R_1(t)$	0	0	0	2
x	$R_x(t)$	1	0	1	0
a	$R_a(t)$	0	1	0	0
b	$R_b(t)$	0	0	1	0
c	$R_c(t)$	0	0	0	1
out	$R_{\text{out}}(t)$	0	0	0	0

coefficients

t^3	t^2	t	1
0.333	-2.0	3.666	-2.0
-0.666	5.0	-11.33	8.0
0.5	-4.0	9.5	-6.0
-0.5	3.5	-7.0	4.0
0.166	-1.0	1.833	-1.0
0.0	0.0	0.0	0.0

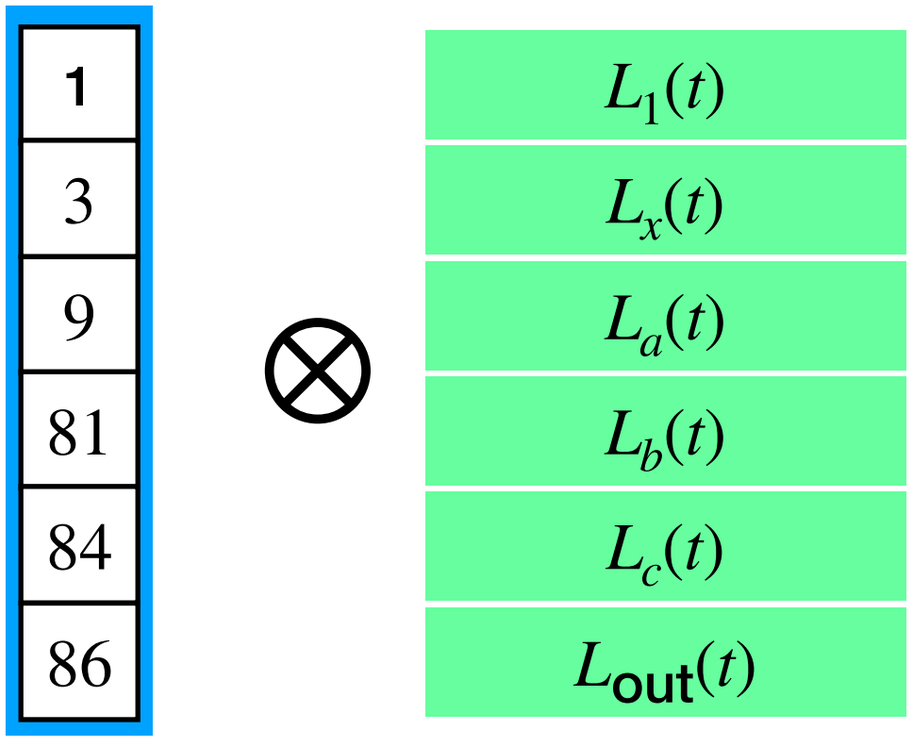
Quadratic Assignment Problem

1	$O_1(t)$	0	0	0	0
x	$O_x(t)$	0	0	0	0
a	$O_a(t)$	1	0	0	0
b	$O_b(t)$	0	1	0	0
c	$O_c(t)$	0	0	1	0
out	$O_{\text{out}}(t)$	0	0	0	1

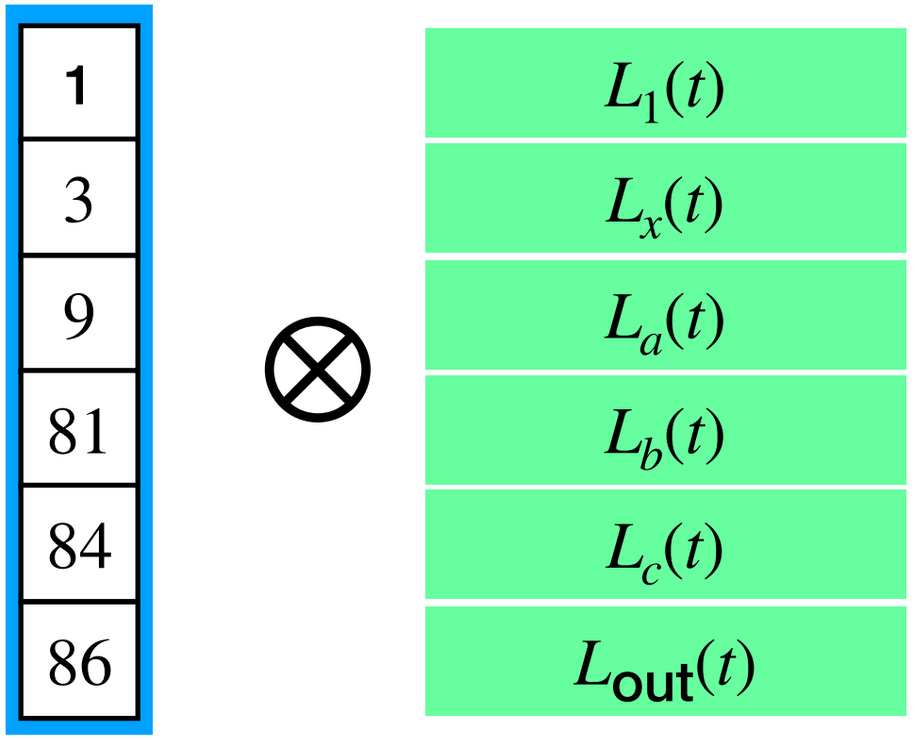
coefficients

t^3	t^2	t	1
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
-0.166	1.5	-4.333	4.0
0.5	-4.0	9.5	-6.0
-0.5	3.5	-7.0	4.0
0.166	-1.0	1.833	-1.0

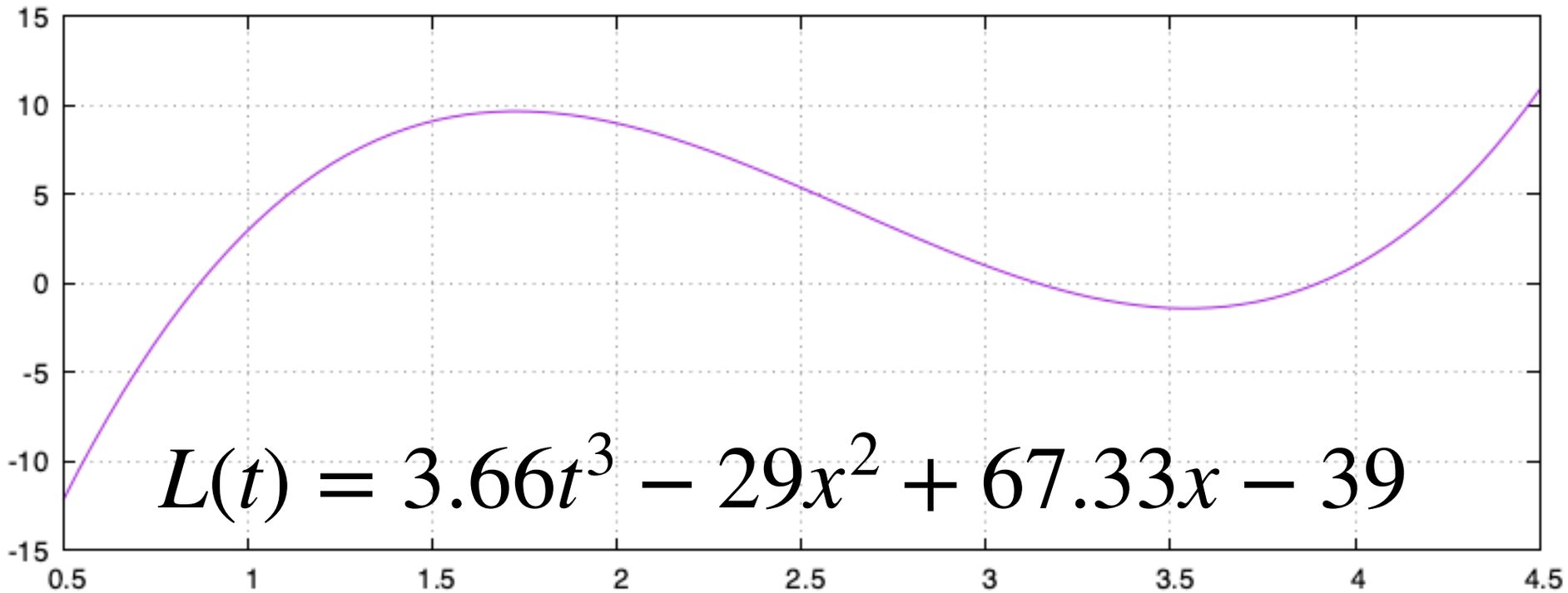
Adding the Witness



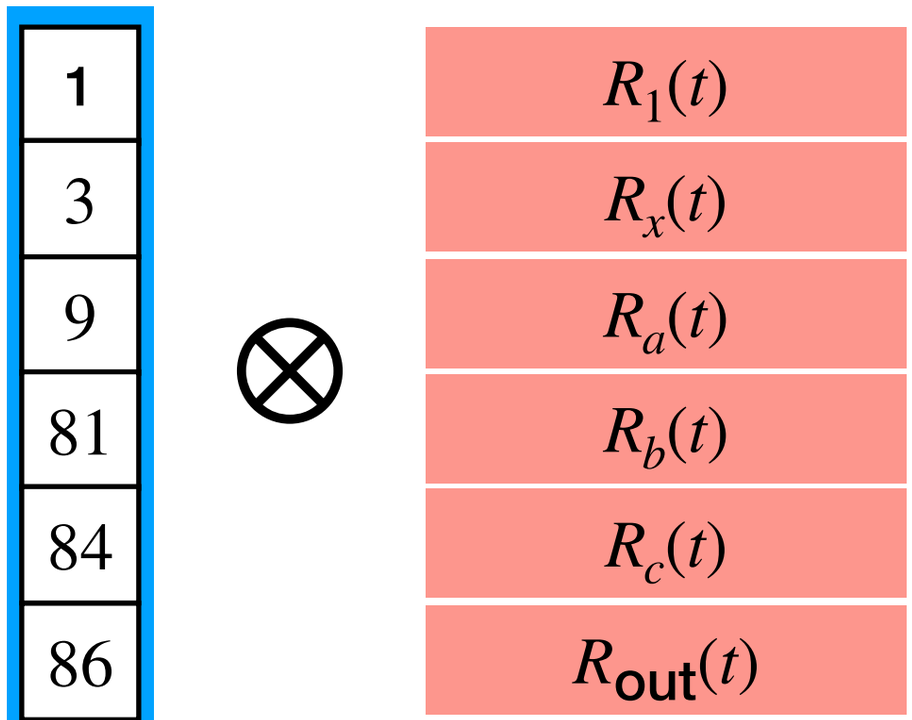
Adding the Witness



this is also a polynomial

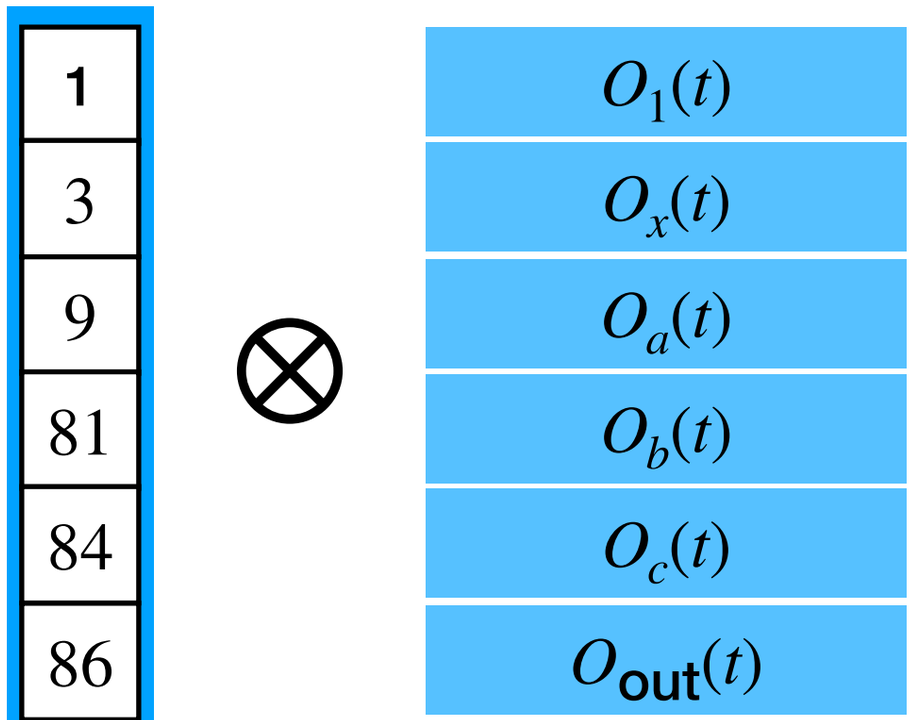


Adding the Witness



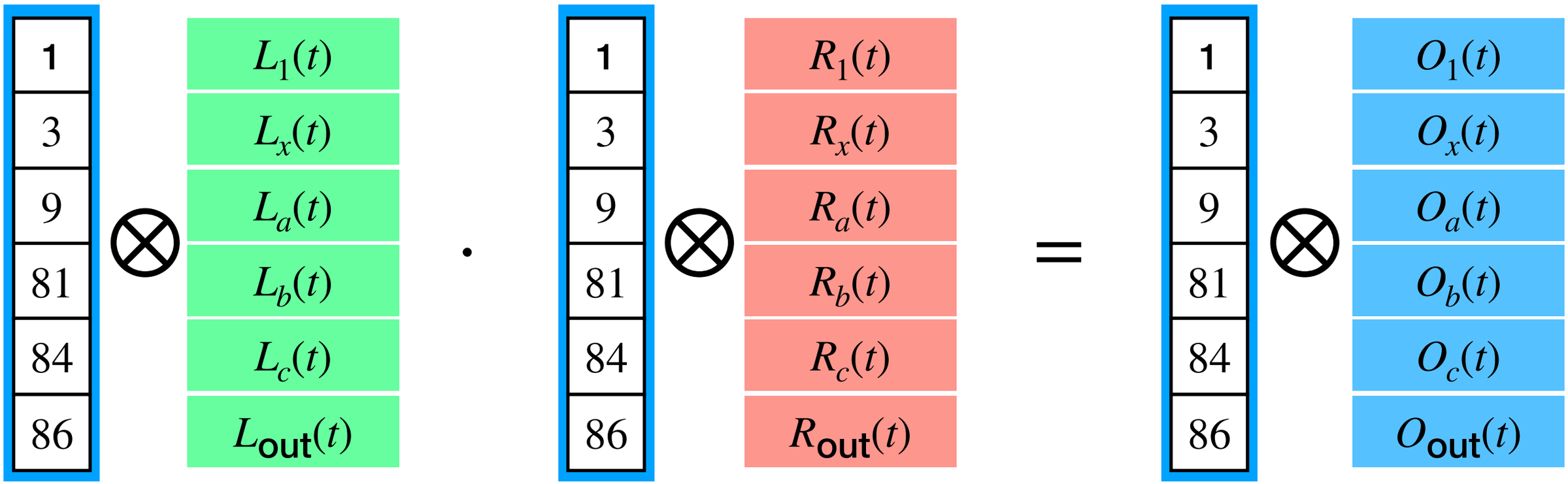
$$R(t) = 22.166t^3 + 167.5x^2 - 341.33x + 199$$

Adding the Witness



$$O(t) = 11.333t^3 - 102.5x^2 + 300.166x - 200$$

Check all Constraints



$$L(t) \cdot R(t) = O(t)$$

For $t = 1,2,3,4$

Check all Constraints

$$L(t) \quad \cdot \quad R(t) \quad = \quad O(t)$$

For $t = 1,2,3,4$

- We don’t make any assumption for the values $t \neq 1,2,3,4$
- More generalized, we can write $X(t) = L(t)R(t) - O(t)$

Polynomials with same roots

- If a polynomial $p_1(x)$ has the same roots than another $p_2(x)$, they divide each other without residue
  - $p_1(x) = c \cdot p_2(x)$
- To check that our polynomial $X(t) = L(t)R(t) - O(t)$ is zero at $t = 1,2,3,4$
  - We construct $Z(t) = (t-1)(t-2)(t-3)(t-4)$
  - Verify that $\frac{X(t)}{Z(t)} = H(t)$

Polynomials with same roots

- We compute $X(t) = L(t)R(t) - O(t)$
- We show $Z(t) = (t-1)(t-2)(t-3)(t-4)$ is a divisor of $X(t)$
  - $X(t)$ is 0 at $t = 1,2,3,4$
    - Thus $L(t) \cdot R(t) = O(t)$ at $t = 1,2,3,4$
- Compute $H(t) = \frac{X(t)}{Z(t)}$
  - If the witness were fake, this division leaves a residue
- All that's left to prove is $H(t)Z(t) = X(t)$

To check all constraints

$$X(t) = L(t)R(t) - O(t)$$

$$Z(t) = (t-1)(t-2)(t-3)(t-4)$$

$$H(t) = \frac{X(t)}{Z(t)}$$

- Instead of $H(t)Z(t) = X(t)$, show $H(t)Z(t) - X(t) = 0$ everywhere

- Instead of $H(t)Z(t) - X(t) = 0$ everywhere, pick a secret t and evaluate the 3 functions there (with ECC math)

Summary

- Alice:
  - List an arbitrary computation as a set of basic operations
  - Create $L\_(t), R\_(t), O\_(t)$ polynomials for each input, temporary variables, output and the constant 1
- Bob:
  - Creates the witness vector
  - Computes $L(t) = W \otimes L\_(t), R(t) = \ldots, O(t) = \ldots$
  - Divides $H(t) = X(t)/Z(t)$
- Alice:
  - Evaluates the equation $H(t)Z(t) = X(t)$ at a point of her choosing, accepts if 0

Trusted Setup

- This is done non-interactively if Alice encrypts the point t as $T = tG$, and Bob proves that $H(T)Z(T) - X(T) = 0$
- If Bob can break the encryption (or if he breaks into Alices computer), he can find t
  - knowing at which point Alice evaluates $H(t)Z(t) = X(t)$, he can fake a solution
- Coda, Zerocoin, Zerocash, and others use zk-SNARKS
  - We need to trust that the creators do not collaborate with some users and share the secret value t