RISC Zero Proof System: A Concrete Example

This document offers a concrete example of the RISC Zero Proof System; you can find a PDF version on www.RISCZero.com or peek behind the formulas with the Google Sheet Version.

For questions, corrections, conversation, and collaboration, find us on Twitter or Discord.

When any code executes in the RISC Zero virtual machine, each step of that execution is recorded in an **Execution Trace**. We show a simplified example, computing 4 steps of a Fibonacci sequence modulo 97, using two user-specified inputs. In this sheet, we introduce the columns of a RISC Zero Execution Trace. In the following sheets, we will demonstrate a concrete example of how RISC Zero proves the validity of an Execution Trace without revealing any knowledge.

In this example, our trace consists of 6 columns. Each of the first three columns is a record of the internal state of a register at each clock cycle from initialization until termination. We call these **Data Columns**. The next three columns are **Control Columns,** which we use to mark initialization and termination points.

In the full RISC Zero protocol, the **Data Columns** hold the state of the RISC-V processor, including ISA registers, the program counter, and various microarchitecture details such as instruction decoding data, ALU registers, etc., while the **Control Columns** handle system initialization and shutdown, the initial program code to load into memory before execution, and other control signals that don't depend on the programs execution.

Input 1	24
Input 2	30

Asserted Output	28

Modulo	97

	Clock Cycle	Data Column 1	Data Column 2	Data Column 3	Control Column - Initialization	Control Column - Transition	Control Column - Termination
Execution Trace - Initialization	0	24	30	54	1	0	0
Execution Trace - Transition	1	30	54	84	0	1	0
Execution Trace - Transition	2	54	84	41	0	1	0
Execution Trace - Termination	3	84	41	28	0	1	1

Checking the Trace

In this sheet, we introduce a number of rule-checking cells in order to demonstrate the validity of the Execution In this example, we show six rules. In the full RISC-V implementation, we check over 100 rules in order to validate the execution trace.

Each rule check is written as the product of two terms, modulo 97. The first term equals zero when the rule holds. The second term equals zero when we don't want to enforce the rule.

Each rule checking column can be expressed as a multi-input, single-output polynomial, where the inputs are some combination of entries in the trace; we call these **Rule-Checking Polynomials**.

Input 1	24
Input 2	30

Asserted Output	28

Modulo	97

	Clock Cycle	Data Column 1	Data Column 2	Data Column 3	Control Column - Initialization	Control Column - Transition	Control Column - Termination	Does Fibonacci relation hold?	Does Initialization for Data Column 1 match User Input 1?	Does Initialization for Data Column 2 match User Input 2?	Does Termination value for Data Column 3 match Output?	Does entry i from Input Column 1 match entry i-1 from Input Column 2?	Does entry i from Input Column 2 match entry i-1 from Output Column?
Execution Trace - Initialization	0	24	30	54	1	0	0	0	0	0	0	0	0
Execution Trace - Transition	1	30	54	84	0	1	0	0	0	0	0	0	0
Execution Trace - Transition	2	54	84	41	0	1	0	0	0	0	0	0	0
Execution Trace - Termination	3	84	41	28	0	1	1	0	0	0	0	0	0

Adding Random Padding

Before encoding each column as a polynomial, we append random padding to the end of the Execution Trace, which allows for a zero-knowledge protocol. This random noise is generated by the host system's cryptographically secure pseudorandom number generator. We set the Control columns to 0 for these random noise rows, in order to turn off our rule checks.

Input 1	24
Input 2	30

Asserted Output	28

Modulo	97

	Clock Cycle	Data Column 1	Data Column 2	Data Column 3	Control Column - Initialization	Control Column - Transition	Control Column - Termination	Does Fibonacci relation hold?	Does Initialization for Data Column 1 match User Input 1?	Does Initialization for Data Column 2 match User Input 2?	Does Termination value for Data Column 3 match Output?	Does entry i from Input Column 1 match entry i-1 from Input Column 2?	Does entry i from Input Column 2 match entry i-1 from Output Column?
Execution Trace - Initialization	0	24	30	54	1	0	0	0	0	0	0	0	0
Execution Trace - Transition	1	30	54	84	0	1	0	0	0	0	0	0	0
Execution Trace - Transition	2	54	84	41	0	1	0	0	0	0	0	0	0
Execution Trace - Termination	3	84	41	28	0	1	1	0	0	0	0	0	0
Random Padding	4	78	55	69	0	0	0	0	0	0	0	0	0
Random Padding	5	15	58	2	0	0	0	0	0	0	0	0	0
Random Padding	6	29	25	56	0	0	0	0	0	0	0	0	0
Random Padding	7	50	92	25	0	0	0	0	0	0	0	0	0

Interpolating Trace Polynomials

Let's remove the rule-checking columns for a minute and turn our attention toward encoding our Trace data in terms of polynomials. Just as any two points allow you to draw a line, any 8 points allow you to determine a degree 7 polynomial. The standard techniques for constructing a polynomial through a given set of points are Lagrange Interpolation and Finite Fourier Transforms (FFTs). In our context, we use NTTs (Number Theoretic Transforms), which are essentially just a finite field equivalent of an FFT. Since we're working modulo 97, our polynomials have values and coefficients in the finite field F_97.

The polynomial encoding technique we use in RISC Zero is called Reed-Solomon encoding; RS codes are ubiquitous in the world of digital signal processing as a method of providing redundance for error checking and error correction purposes.

Reed-Solomon Codes are built using points of the form (a^0, x_0), (a^1, x_1), ... In this example, we use powers of 28 as the inputs (and the entries of the trace as the outputs).

We use this technique to write a **Trace Polynomial** for each column of the trace. Running an iNTT on the 8 entries from Data Column 1, we use iNTT(column, modulus) to generate a polynomial whose evaluations agree with the trace data. Then, we evaluate this polynomial over an **Expanded Domain** to construct the Reed Solomon encoding of the column.

In Python using sympy, intt([**24**, **30**, **54**, **84**, **78**, **15**, **29**, **50**], prime=97) returns [*94*, *68*, *41*, *69*, *25*, *72*, *85*, *55*].

The 8 entries of this iNTT input array are shown in **boldface** in Data Column 1 below; with each entry corresponding to a row of the Padded Trace.

We use the entries of the output array as the coefficients of the associated Trace Polynomial. In this case, d_1(x) = *94 + 68x + 41x^2 + 69x^3 + 25x^4 +72x^5 +85x^6 +55x^7* (modulo 97).

The key feature of d_1(x) is that for z=5^0, 5^12, 5^24, ... the evaluations d_1(z) agree with the values in Data Column 1: d_1(5^0) = 24, d_1(5^12)=30, d_1(5^24) = 54, etc.

We proceed similarly on each data column and each control column, generating one Trace Polynomial for each column of our Trace.

A quick note about F_97: every element of this field can be written as a power of 5. In other words, the elements of F_97 are 0, 5^0, 5^1, ..., and 5^95. Written in this form, we can view our Reed-Solomon inputs as every third power of 5: 5^0, 5^3, 5^6, etc.

For a brief introduction to finite fields as they relate to the RISC Zero proof system, see here.

Input 1	24
Input 2	30

Asserted Output	28

Modulo	97

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**Data Column 2**	**Data Column 3**	**Control Column - Initialization**	**Control Column - Transition**	**Control Column - Termination**
Execution Trace - Initialization	28^0 mod 97	5^0 mod 97	1	0	**24**	30	54	1	0	0
RS Redundancy Row	28^1 mod 97	5^3 mod 97	28		27	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row	28^2 mod 97	5^6 mod 97	8		74	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row	28^3 mod 97	5^9 mod 97	30		77	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Execution Trace - Transition			64	1	**30**	54	84	0	1	0
RS Redundancy Row			46		37	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			27		62	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			77		3	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Execution Trace - Transition			22	2	**54**	84	41	0	1	0
RS Redundancy Row			34		42	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			79		96	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			78		69	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Execution Trace - Termination			50	3	**84**	41	28	0	0	1
RS Redundancy Row			42		36	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			12		26	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			45		37	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Random Padding			96	4	**78**	95	77	0	0	0
RS Redundancy Row			69		71	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			89		24	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			67		70	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Random Padding			33	5	**15**	52	7	0	0	0
RS Redundancy Row			51		31	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			70		38	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			20		71	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Random Padding			75	6	**29**	82	12	0	0	0
RS Redundancy Row			63		40	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			18		54	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			19		19	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
Random Padding			47	7	**50**	12	26	0	0	0
RS Redundancy Row			55		80	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row			85		87	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
RS Redundancy Row	28^31 mod 97	5^93 mod 97	52		18	For each column, compute iNTT(Trace Column, modulus) to fill in this data				
	28^32 mod 97	5^96 mod 97	1		24					

Interpolating Trace Polynomials (cont.)

This sheet shows our 6 trace polynomials, each evaluated at 32 points.

intt(TraceColumn, prime=97) returns the coefficients of the trace polynomial.

Column	Python Code (from sympy import intt)	Code Output (Coefficients of Trace Polynomials)								Trace Polynomials
d1	d1 = intt([24, 30, 54, 84, 78, 15, 29, 50], prime=97)	94	68	41	69	25	72	85	55	d1(x)=94+68x+41x^2+69x^3=25x^4+72x^5+85x^6+55x^7
d2	d2 = intt([30, 54, 84, 41, 2, 77, 21, 36], prime=97)	31	31	0	87	76	66	6	24	d2(x)=31+31x+0x^2+87x^3=76x^4+66x^5+6x^6+24x^7
d3	d3 = intt([54, 84, 41, 28, 71, 17, 92, 33], prime=97)	4	14	83	44	12	44	12	35	d3(x)=4+14x+83x^2+44x^3=12x^4+44x^5+12x^6+35x^7
c1	c1 = intt([1, 0, 0, 0, 0, 0, 0, 0], prime=97)	85	85	85	85	85	85	85	85	c1(x)=85+85x+85x^2+85x^3=85x^4+85x^5+85x^6+85x^7
c2	c2 = intt([0, 1, 1, 1, 0, 0, 0, 0], prime=97)	61	80	12	37	12	60	12	17	c2(x)=61+80x+12x^2+37x^3=12x^4+60x^5+12x^6+17x^7
c3	c3 = intt([0, 0, 0, 1, 0, 0, 0, 0], prime=97)	85	89	27	18	12	8	70	79	c3(x)=85+89x+27x^2+18x^3=12x^4+8x^5+70x^6+79x^7

Input 1	24
Input 2	30

Asserted Output	28

Modulo	97

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**Data Column 2**	**Data Column 3**	**Control Column - Initialization**	**Control Column - Transition**	**Control Column - Termination**
Execution Trace - Initialization	28^0 mod 97	5^0 mod 97	1	**0**	**24**	**30**	**54**	**1**	**0**	**0**
RS Redundancy Row	28^1 mod 97	5^3 mod 97	28		27	33	43	23	35	26
RS Redundancy Row	28^2 mod 97	5^6 mod 97	8		74	14	27	45	31	13
RS Redundancy Row	28^3 mod 97	5^9 mod 97	30		77	31	88	53	63	86
Execution Trace - Transition			64	**1**	**30**	**54**	**84**	**0**	**1**	**0**
RS Redundancy Row			46		37	76	67	72	32	46
RS Redundancy Row			27		62	85	34	83	63	87
RS Redundancy Row			77		3	84	63	70	30	80
Execution Trace - Transition			22	**2**	**54**	**84**	**41**	**0**	**1**	**0**
RS Redundancy Row			34		42	14	11	10	58	60
RS Redundancy Row			79		96	18	86	60	59	28
RS Redundancy Row			78		69	3	29	25	20	91
Execution Trace - Termination			50	**3**	**84**	**41**	**28**	**0**	**1**	**1**
RS Redundancy Row			42		36	15	54	53	8	23
RS Redundancy Row			12		26	16	31	11	91	45
RS Redundancy Row			45		37	77	1	68	51	53
Random Padding			96	**4**	**78**	**2**	**71**	**0**	**0**	**0**
RS Redundancy Row			69		71	90	82	2	38	72
RS Redundancy Row			89		24	15	14	62	57	83
RS Redundancy Row			67		70	66	75	13	66	70
Random Padding			33	**5**	**15**	**77**	**17**	**0**	**0**	**0**
RS Redundancy Row			51		31	8	76	26	65	10
RS Redundancy Row			70		38	20	67	13	36	60
RS Redundancy Row			20		71	19	14	86	9	25
Random Padding			75	**6**	**29**	**21**	**92**	**0**	**0**	**0**
RS Redundancy Row			63		40	43	42	46	81	53
RS Redundancy Row			18		54	72	72	87	86	11
RS Redundancy Row			19		19	92	90	80	70	68
Random Padding			47	**7**	**50**	**36**	**33**	**0**	**0**	**0**
RS Redundancy Row			55		80	66	45	60	74	2
RS Redundancy Row			85		87	8	89	28	65	62
RS Redundancy Row	28^31 mod 97		52		18	70	60	91	82	13
	28^32 mod 97		1							

Committing Trace Polynomials

The next step is for the Prover to commit the **Trace Polynomials** into a **Merkle Tree**. In order to maintain a Zero-Knowledge protocol, the Prover evaluates each Trace Polynomial over a "shifted evaluation domain." Specifically, we evaluate each d_i(x) at x=5, 5^4, 5^7, ..., 5^93.

Notice that because of our shifted evaluation domain, the yellow and blue cells in Data Columns 1, 2, and 3 no longer match the Inputs and Asserted Outputs. In fact, this shift in the evaluation domain disguises *all* the Trace Data. We only reveal information about the disguised trace, and the random padding we appended is sufficient to prevent an attacker from deducing any connection between the disguised trace and the actual trace.

Trace Polynomials
d1(x)=94+68x+41x^2+69x^3=25x^4+72x^5+85x^6+55x^7
d2(x)=31+31x+0x^2+87x^3=76x^4+66x^5+6x^6+24x^7
d3(x)=4+14x+83x^2+44x^3=12x^4+44x^5+12x^6+35x^7
c1(x)=85+85x+85x^2+85x^3=85x^4+85x^5+85x^6+85x^7
c2(x)=61+80x+12x^2+37x^3=12x^4+60x^5+12x^6+17x^7
c3(x)=85+89x+27x^2+18x^3=12x^4+8x^5+70x^6+79x^7

Input 1	24
Input 2	30

Modulo	97

Asserted Output	28

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**Data Column 2**	**Data Column 3**	**Control Column - Initialization**	**Control Column - Transition**	**Control Column - Termination**
Disguised Execution Trace	5*28^0 mod 97	5^1 mod 97	5	**0**	**31**	**39**	**12**	**82**	**81**	**2**
Disguised RS Redundancy Row	5*28^1 mod 97	5^4 mod 97	43		15	36	11	18	72	32
Disguised RS Redundancy Row	5*28^2 mod 97	5^7 mod 97	40		96	65	6	32	73	65
Disguised RS Redundancy Row	5*28^3 mod 97	5^10 mod 97	53		79	41	88	68	10	22
Disguised Execution Trace			29	**1**	**69**	**35**	**49**	**81**	**64**	**32**
Disguised RS Redundancy Row			36		31	85	50	41	58	16
Disguised RS Redundancy Row			38		16	41	69	18	40	38
Disguised RS Redundancy Row			94		71	24	89	86	56	50
Disguised Execution Trace			13	**2**	**35**	**77**	**46**	**92**	**16**	**47**
Disguised RS Redundancy Row			73		10	40	9	59	83	24
Disguised RS Redundancy Row			7		53	54	58	14	20	67
Disguised RS Redundancy Row			2		28	7	95	44	92	35
Disguised Execution Trace			56	**3**	**67**	**81**	**80**	**43**	**61**	**82**
Disguised RS Redundancy Row			16		26	2	73	31	21	18
Disguised RS Redundancy Row			60		0	54	76	8	64	32
Disguised RS Redundancy Row			31		45	36	80	92	4	68
Disguised | Random Padding			92	**4**	**91**	**59**	**35**	**10**	**22**	**81**
Disguised RS Redundancy Row			54		94	39	57	71	34	41
Disguised RS Redundancy Row			57		18	82	19	50	40	18
Disguised RS Redundancy Row			44		80	36	44	89	28	86
Disguised | Random Padding			68	**5**	**54**	**12**	**61**	**2**	**48**	**92**
Disguised RS Redundancy Row			61		39	46	78	32	64	59
Disguised RS Redundancy Row			59		16	16	16	65	72	14
Disguised RS Redundancy Row			3		19	49	95	22	31	44
Disguised | Random Padding			84	**6**	**57**	**23**	**47**	**32**	**55**	**43**
Disguised RS Redundancy Row			24		74	89	18	16	37	31
Disguised RS Redundancy Row			90		40	96	42	38	26	8
Disguised RS Redundancy Row			95		43	54	72	50	9	92
Disguised | Random Padding			41	**7**	**57**	**19**	**90**	**47**	**44**	**10**
Disguised RS Redundancy Row			81		75	8	27	24	22	71
Disguised RS Redundancy Row			37		28	34	37	67	56	50
Disguised RS Redundancy Row	5*28^31 mod 97		66		96	1	51	35	64	89
	28^32 mod 97		5							

Introducing Constraint Polynomials

Now that we've encoded our Trace data into Trace polynomials, let's return to our original Reed-Solomon domain and add back in our Rule Checking cells.

Of course, we shouldn't expect these rule checks to evaluate to 0 in the redundancy rows, as they're not directly associated with the data from the trace.

Conveniently, by writing these rule checks in terms of our trace polynomials, we can convert our multi-input rule checking polynomials into single-input polynomials, which we call **Constraint Polynomials.**

Note that each Constraint Polynomials will evaluate to 0 at the RS input values that are associated with actual trace data.

Input 1	24
Input 2	30

Asserted Output	28

Modulo	97

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**Data Column 2**	**Data Column 3**	**Control Column - Initialization**	**Control Column - Transition**	**Control Column - Termination**	**Does Fibonacci relation hold?**	**Does Initialization for Data Column 1 match User Input 1?**	**Does Initialization for Data Column 2 match User Input 2?**	**Does Termination value for Data Column 3 match Output?**	**Does entry i from Input Column 1 match entry i-1 from Input Column 2?**	**Does entry i from Input Column 2 match entry i-1 from Output Column?**
Execution Trace - Initialization	28^0 mod 97	5^0 mod 97	1	**0**	**24**	**30**	**54**	**1**	**0**	**0**	0	0	0	0	0	0
RS Redundancy Row	28^1 mod 97	5^3 mod 97	28		27	33	43	23	35	26	27	69	69	2	90	65
RS Redundancy Row	28^2 mod 97	5^6 mod 97	8		74	14	27	45	31	13	3	19	56	84	9	3
RS Redundancy Row	28^3 mod 97	5^9 mod 97	30		77	31	88	53	63	86	34	93	53	19	53	16
Execution Trace - Transition			64	**1**	**30**	**54**	**84**	**0**	**1**	**0**	0	0	0	0	0	0
RS Redundancy Row			46		37	76	67	72	32	46	84	63	14	48	31	86
RS Redundancy Row			27		62	85	34	83	63	87	55	50	6	37	17	65
RS Redundancy Row			77		3	84	63	70	30	80	45	82	94	84	33	74
Execution Trace - Transition			22	**2**	**54**	**84**	**41**	**0**	**1**	**0**	0	0	0	0	0	0
RS Redundancy Row			34		42	14	11	10	58	60	60	83	34	47	65	30
RS Redundancy Row			79		96	18	86	60	59	28	55	52	56	72	67	26
RS Redundancy Row			78		69	3	29	25	20	91	69	58	4	91	88	61
Execution Trace - Termination			50	**3**	**84**	**41**	**28**	**0**	**1**	**1**	0	0	0	0	0	0
RS Redundancy Row			42		36	15	54	53	8	23	58	54	78	16	79	32
RS Redundancy Row			12		26	16	31	11	91	45	32	22	40	38	49	32
RS Redundancy Row			45		37	77	1	68	51	53	61	11	92	24	85	23
Random Padding			96	**4**	**78**	**2**	**71**	**0**	**0**	**0**	0	0	0	0	0	0
RS Redundancy Row			69		71	90	82	2	38	72	76	94	23	8	91	10
RS Redundancy Row			89		24	15	14	62	57	83	91	0	40	2	68	58
RS Redundancy Row			67		70	66	75	13	66	70	29	16	80	89	23	22
Random Padding			33	**5**	**15**	**77**	**17**	**0**	**0**	**0**	0	0	0	0	0	0
RS Redundancy Row			51		31	8	76	26	65	10	51	85	10	92	45	40
RS Redundancy Row			70		38	20	67	13	36	60	11	85	64	12	52	22
RS Redundancy Row			20		71	19	14	86	9	25	95	65	24	38	45	78
Random Padding			75	**6**	**29**	**21**	**92**	**0**	**0**	**0**	0	0	0	0	0	0
RS Redundancy Row			63		40	43	42	46	81	53	89	57	16	63	70	43
RS Redundancy Row			18		54	72	72	87	86	11	55	88	65	96	14	42
RS Redundancy Row			19		19	92	90	80	70	68	78	85	13	45	0	28
Random Padding			47	**7**	**50**	**36**	**33**	**0**	**0**	**0**	0	0	0	0	0	0
RS Redundancy Row			55		80	66	45	60	74	2	38	62	26	34	22	30
RS Redundancy Row			85		87	8	89	28	65	62	40	18	63	96	5	11
RS Redundancy Row	28^31 mod 97		52		18	70	60	91	82	13	30	36	51	28	43	9
	28^32 mod 97		1													

Mixing Constraint Polynomials

In this sheet, we add one new column, which Mixes our **Constraint Polynomials** into a single **Constraint Polynomial**.

After the Prover sends a Merkle root for each trace polynomial, the verifier responds with a **Constraint Mixing Parameter,** *a*.

Letting c_i denote the constraint polynomials for i=0,1,2,3,4,5, we write C(x) = a^0 * c_0(x) + a^1 c_1(x) + ... + a^5 *c_5(x)

Note that if each c_i evaluates to 0 at some input *x*, then C will also evaluate to 0 for that input.

In this example, the degree of the Mixed Constraint Polynomial is equal to the degree of the Trace Polynomials, because the Rule-Checking involved is particularly simple. In more complicated examples, composing our Rule Checking Polynomials with our Trace Polynomials would yield "High Degree Constraint Polynomials." In that case, we'd add an extra step at this point to split our "High Degree Mixed Constraint Polynomial" into a few "Low Degree Mixed Constraint Polynomials."

Input 1	24
Input 2	30

Asserted Output	28
Constraint Mixing Parameter	3

Modulo	97

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**Data Column 2**	**Data Column 3**	**Control Column - Initialization**	**Control Column - Transition**	**Control Column - Termination**	**Mixed Constraint Polynomial**	**Does Fibonacci relation hold?**	**Does Initialization for Data Column 1 match User Input 1?**	**Does Initialization for Data Column 2 match User Input 2?**	**Does Termination value for Data Column 3 match Output?**	**Does entry i from Input Column 1 match entry i-1 from Input Column 2?**	**Does entry i from Input Column 2 match entry i-1 from Output Column?**
Execution Trace - Initialization	28^0 mod 97	5^0 mod 97	5	**0**	**31**	**39**	**12**	**82**	**81**	**2**	52	33	89	59	65	2	40
RS Redundancy Row	28^1 mod 97	5^3 mod 97	43		15	36	11	18	72	32	47	67	32	11	38	19	66
RS Redundancy Row	28^2 mod 97	5^6 mod 97	40		96	65	6	32	73	65	45	34	73	53	25	64	7
RS Redundancy Row	28^3 mod 97	5^9 mod 97	53		79	41	88	68	10	22	32	1	54	69	59	4	94
Execution Trace - Transition			29	**1**	**69**	**35**	**49**	**81**	**64**	**32**	86	62	56	17	90	77	17
RS Redundancy Row			36		31	85	50	41	58	16	77	73	93	24	61	1	24
RS Redundancy Row			38		16	41	69	18	40	38	95	85	50	4	6	77	42
RS Redundancy Row			94		71	24	89	86	56	50	62	12	65	66	43	31	5
Execution Trace - Transition			13	**2**	**35**	**77**	**46**	**92**	**16**	**47**	80	52	42	56	70	0	60
RS Redundancy Row			73		10	40	9	59	83	24	61	81	47	8	29	80	43
RS Redundancy Row			7		53	54	58	14	20	67	6	95	18	45	70	46	88
RS Redundancy Row			2		28	7	95	44	92	35	45	75	79	55	17	77	22
Execution Trace - Termination			56	**3**	**67**	**81**	**80**	**43**	**61**	**82**	27	59	6	59	93	69	1
RS Redundancy Row			16		26	2	73	31	21	18	54	46	62	5	34	94	47
RS Redundancy Row			60		0	54	76	8	64	32	73	57	2	95	81	36	35
RS Redundancy Row			31		45	36	80	92	4	68	23	30	89	67	44	55	55
Random Padding			92	**4**	**91**	**59**	**35**	**10**	**22**	**81**	79	3	88	96	82	26	23
RS Redundancy Row			54		94	39	57	71	34	41	63	59	23	57	25	24	8
RS Redundancy Row			57		18	82	19	50	40	18	43	79	88	78	32	15	46
RS Redundancy Row			44		80	36	44	89	28	86	44	31	37	49	18	68	29
Random Padding			68	**5**	**54**	**12**	**61**	**2**	**48**	**92**	16	66	60	61	29	51	60
RS Redundancy Row			61		39	46	78	32	64	59	65	79	92	27	40	0	72
RS Redundancy Row			59		16	16	16	65	72	14	52	9	62	60	26	1	75
RS Redundancy Row			3		19	49	95	22	31	44	18	0	84	30	38	55	58
Random Padding			84	**6**	**57**	**23**	**47**	**32**	**55**	**43**	4	75	86	67	41	50	44
RS Redundancy Row			24		74	89	18	16	37	31	18	42	24	71	78	66	19
RS Redundancy Row			90		40	96	42	38	26	8	68	22	26	83	15	42	43
RS Redundancy Row			95		43	54	72	50	9	92	7	8	77	36	71	43	19
Random Padding			41	**7**	**57**	**19**	**90**	**47**	**44**	**10**	4	56	96	65	38	41	29
RS Redundancy Row			81		75	8	27	24	22	71	22	44	60	54	26	80	71
RS Redundancy Row			37		28	34	37	67	56	50	62	40	74	74	62	72	37
RS Redundancy Row	28^31 mod 97		66		96	1	51	35	64	89	57	82	95	52	10	69	15
	28^32 mod 97		5														

The Core of the RISC Zero STARK

The Prover constructs the **Validity Polynomial** by dividing the **Constraint Polynomial** from the previous sheet by the publicly known **Zeros Polynomial**.

V(x) = C(x) / Z(x)

In our example, the **Zeros Polynomial** is Z(x) = (x-1)*(x-47)*(x-75)*(x-33)*(x-96)*(x-50)*(x-22)*(x-64)

Normally when we divide two low degree polynomials, we don't expect to get another low degree polynomial. But for an honest prover, it's not hard to see that V(x) will be lower degree than C(x), since the roots of Z(x) line up perfectly with roots of C(x) (see previous page)

The Prover evaluates V(x) over the "shifted evaluation domain" shown below, commits the values to a Merkle Tree, and sends the Merkle root to the verifier.

The construction of these polynomials is the core conceptual thrust of RISC Zero's Proof of Trace Validity. All of the information necessary to confirm the validity of the original Execution trace can be described in the following assertions about these polynomials:

(i) V(x) = C(x) / Z(x) for all x

(ii) The degree of the Validity Polynomial and each Trace Polynomials are less than or equal to 7.

The FRI protocol is the technique we use for proving (ii). Those details are omitted from this simplified example.

In the original STARK protocol, the Verifier tests (i) at a number of test points; the soundness of the protocol depends on the number of tests. The DEEP-ALI technique allows us to achieve a high degree of soundness with a single test. The details of DEEP are described in the following sheet.

Input 1	24
Input 2	30

Asserted Output	28
Constraint Mixing Parameter	3

Modulo	97

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**Data Column 2**	**Data Column 3**	**Control Column - Initialization**	**Control Column - Transition**	**Control Column - Termination**	**Mixed Constraint Polynomial**	**Zeros Polynomial, Z (x)**	**Validity Polynomial, V (x) = C(x) / Z(x)**
Execution Trace - Initialization	28^0 mod 97	5^0 mod 97	5	**0**	**31**	**39**	**12**	**82**	**81**	**2**	52	5	88
RS Redundancy Row	28^1 mod 97	5^3 mod 97	43		15	36	11	18	72	32	47	34	67
RS Redundancy Row	28^2 mod 97	5^6 mod 97	40		96	65	6	32	73	65	45	90	49
RS Redundancy Row	28^3 mod 97	5^9 mod 97	53		79	41	88	68	10	22	32	61	53
Execution Trace - Transition			29	**1**	**69**	**35**	**49**	**81**	**64**	**32**	86	5	56
RS Redundancy Row			36		31	85	50	41	58	16	77	34	85
RS Redundancy Row			38		16	41	69	18	40	38	95	90	28
RS Redundancy Row			94		71	24	89	86	56	50	62	61	36
Execution Trace - Transition			13	**2**	**35**	**77**	**46**	**92**	**16**	**47**	80	5	16
RS Redundancy Row			73		10	40	9	59	83	24	61	34	56
RS Redundancy Row			7		53	54	58	14	20	67	6	90	13
RS Redundancy Row			2		28	7	95	44	92	35	45	61	23
Execution Trace - Termination			56	**3**	**67**	**81**	**80**	**43**	**61**	**82**	27	5	83
RS Redundancy Row			16		26	2	73	31	21	18	54	34	13
RS Redundancy Row			60		0	54	76	8	64	32	73	90	45
RS Redundancy Row			31		45	36	80	92	4	68	23	61	29
Random Padding			92	**4**	**91**	**59**	**35**	**10**	**22**	**81**	79	5	74
RS Redundancy Row			54		94	39	57	71	34	41	63	34	96
RS Redundancy Row			57		18	82	19	50	40	18	43	90	77
RS Redundancy Row			44		80	36	44	89	28	86	44	61	85
Random Padding			68	**5**	**54**	**12**	**61**	**2**	**48**	**92**	16	5	42
RS Redundancy Row			61		39	46	78	32	64	59	65	34	39
RS Redundancy Row			59		16	16	16	65	72	14	52	90	48
RS Redundancy Row			3		19	49	95	22	31	44	18	61	48
Random Padding			84	**6**	**57**	**23**	**47**	**32**	**55**	**43**	4	5	59
RS Redundancy Row			24		74	89	18	16	37	31	18	34	69
RS Redundancy Row			90		40	96	42	38	26	8	68	90	18
RS Redundancy Row			95		43	54	72	50	9	92	7	61	51
Random Padding			41	**7**	**57**	**19**	**90**	**47**	**44**	**10**	4	5	59
RS Redundancy Row			81		75	8	27	24	22	71	22	34	52
RS Redundancy Row			37		28	34	37	67	56	50	62	90	5
RS Redundancy Row	28^31 mod 97		66		96	1	51	35	64	89	57	61	55
	28^32 mod 97		5										

Constructing the DEEP Polynomials

In this sheet, we use the Trace Polynomials and the Validity Polynomial(s) to construct the DEEP Polynomials. The DEEP polynomials allow the Verifier to test V(x) = C(x) / Z(x) outside the original Merkle tree commitments, which substantially improves the robustness of the Verifier's test.

With commitments of the trace polynomials and the validity polynomial in place, the Verifier picks a random field element z. We use z=93 in this example. The Verifier would like to be able to compute the Mixed constraint polynomial, C(93). The Prover sends V(z) and the **necessary evaluations** of the Trace Polynomials to allow the Verifer to compute C(93).

In order to enable the Verifier to make this computation, the Prover supplies the **necessary evaluations**: d_1(93), d_2(93), d_3(93), c_1(93), c_2(93), c_3(93), d_2(93*5^-12), d_3(93*5^-12). This 5^-12 is a pointer backwards 1 computational step and allows for checking the rules that span multiple clock-cycles.

These 8 points are called the **taps of the trace** at 93. These 8 points, together with the publicly known rule-checking functions, allow the Verifier to manually compute C(93) and therefore V(93).

The Prover also constructs the DEEP polynomials, interpolates each one, and sends the coefficients of each DEEP polynomial to the Verifier. The DEEP polynomials are defined as follows:

d`_1(x) = (d_1(x) - d_1(93)) / (x - 93)

d`_2(x) = (d_2(x) - dbar_2(x)) / ((x-93)(x-6)) where dbar_2(x) is constructed by interpolating (6, d_2(6)) and (93, d_2(93)

d`_3(x) = (d_3(x) - dbar_3(x)) / ((x-93)(x-6)) where dbar_3(x) is constructed by interpolating (6, d_3(6)) and (93, d_3(93)

c`_1(x) = (c_1(x) - c_1(93)) / (x - 93)

c`_2(x) = (c_2(x) - c_2(93)) / (x - 93)

c`_3(x) = (c_3(x) - c_3(93)) / (x - 93)

V`(x) = (V(x) - V(93)) / (x - 93) where the Prover computes V(93) by running iNTT(ValidityColumn) and then evaluating at 93.

intt([88,67,49,53,56,85,28,36,16,56,13,23,83,13,45,29,74,96,77,85,42,39,48,48,59,69,18,51,59,52,5,55], prime=97)

V coefficients [96, 83, 47, 58, 36, 60, 96, 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]

Without the DEEP technique, the Prover would assert that d_1, d_2, d_3, c_1, c_2, c_3, and V were all low degree polynomials.

With the DEEP technique, the Prover argues instead that d`_1, d`_2, d`_3, c`_1, c`_2, c`_3, and V` are low degree polynomials

Input 1	24
Input 2	30

Modulo	97

Asserted Output	28
Constraint Mixing Parameter	3
Random Test Point	93

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input** (Simplified)	**Clock Cycle**	**Data Column 1**	**DEEP Polynomial 1**	**Data Column 2**	**DEEP Polynomial 2**	**Data Column 3**	**DEEP Polynomial 3**	**Control Column - Initialization**	**DEEP Polynomial 4**	**Control Column - Transition**	**DEEP Polynomial 5**	**Control Column - Termination**	**DEEP Polynomial 6**	**Validity Polynomial, V (x) = C(x) / Z(x)**	**DEEP Validity Polynomial**
Disguised Execution Trace	5*28^0 mod 97	5^1 mod 97	5	0	1	67	83	33	29	28	82	47	76	88	43	38	88	53
Disguised RS Redundancy Row	5*28^1 mod 97	5^4 mod 97	43		30	60	69	29	60	16	18	4	3	48	31	8	67	4
Disguised RS Redundancy Row	5*28^2 mod 97	5^7 mod 97	40		94	1	81	72	30	38	32	79	32	44	8	58	49	60
Non-Merkle Points	5^2*28^2 mod 97	5^8 mod 97	6				21		1			5	3	4	7	50		
		5^9 mod 97																
Disguised RS Redundancy Row	5*28^3 mod 97	5^10 mod 97	53		27	37	46	16	20	1	68	64	33	84	92	23	53	26
Disguised Execution Trace		5^13 mod 97	29	1	88	52	19	96	28	9	81	2	38	90	10	41	56	49
Disguised RS Redundancy Row		5^16 mod 97	36		21	88	46	94	2	30	41	51	90	36	71	56	85	45
Disguised RS Redundancy Row	5*28^6 mod 97	5^19 mod 97	38		60	96	11	20	27	86	18	4	22	32	50	89	28	93
Non-Merkle Points	5^2*28^6 mod 97	5^20 mod 97	93		77		57		69		47		60		89		96	
		5^21 mod 97	77															
Disguised RS Redundancy Row		5^22 mod 97	94		68	88	4	8	45	79	86	39	39	76	89	0	36	37
Disguised Execution Trace			13	2	12	19	61	92	87	27	92	54	53	11	2	12	16	1
Disguised RS Redundancy Row			73		29	20	5	93	74	93	59	92	5	31	32	48	56	49
Disguised RS Redundancy Row			7		36	58	83	28	3	37	14	94	58	88	65	86	13	63
Disguised RS Redundancy Row			2		94	19	28	15	40	41	44	48	84	4	22	5	23	4
Disguised Execution Trace			56	3	1	64	29	64	76	49	43	34	12	20	32	48	83	62

(For readability, we've abbreviated the trace.)

The FRI Polynomial

After using the DEEP polynomials to check the relation between the Trace Polynomial, the Validity Polynomial, and the Zeros Polynomial at z=93, the only thing left for the Prover to do is to show that the DEEP polynomials are low-degree.

The FRI protocol provides a mechanism for the Verifier to confirm the low-degree-ness of polynomials, with very little computation required of the Verifier. In order to reduce this assertion of low-degree-ness to a single application of FRI, the Prover mixes the DEEP polynomials into a single FRI polynomial, using the DEEP Mixing parameter.

Letting c`_1, c`_2, c`_3, d`_1, d`_2, d`_3, and V` denote the DEEP polynomials, we mix the DEEP polynomials to construct the FRI polynomial,
f(x) = b^0 * c`_1(x) + b^1 c`_2(x) + ... + b^6 *V`(x)

To complete the argument, the Prover constructs a FRI proof that f(x) is a low degree polynomial.

With this process, the Prover has constructed a zero-knowledge argument of computational integrity that can be verified incredibly quickly.

Input 1	24
Input 2	30

Modulo	97

Asserted Output	28
Constraint Mixing Parameter	3
Random Test Point	93
DEEP Mixing Parameter	21

	Reed Solomon Input (Exponent Notation in terms of 28)	**Reed Solomon Input** (Exponent Notation in terms of 5)	**Reed Solomon Input (Simplified)**	**Clock Cycle**	**DEEP Polynomial 1**	**DEEP Polynomial 2**	**DEEP Polynomial 3**	**DEEP Polynomial 4**	**DEEP Polynomial 5**	**DEEP Polynomial 6**	**DEEP Validity Polynomial**	**FRI Polynomial**
Disguised Execution Trace	5*28^0 mod 97	5^1 mod 97	5	0	67	67	1	47	88	38	53	64
Disguised RS Redundancy Row	5*28^1 mod 97	5^4 mod 97	43		96	82	72	84	38	71	84	27
Disguised RS Redundancy Row	5*28^2 mod 97	5^7 mod 97	40		7	89	82	68	17	18	32	16
Disguised RS Redundancy Row	5*28^3 mod 97	5^10 mod 97	53		74	11	31	31	71	46	52	36
Disguised Execution Trace		5^13 mod 97	29	1	65	1	88	51	64	27	37	26
Disguised RS Redundancy Row		5^16 mod 97	36		18	39	95	92	25	82	7	80
Disguised RS Redundancy Row	5*28^6 mod 97	5^19 mod 97	38		25	45	48	94	73	6	3	4
Disguised RS Redundancy Row		5^22 mod 97	94		88	28	34	39	76	0	37	4
Disguised Execution Trace			13	2	19	27	87	54	11	12	1	15
Disguised RS Redundancy Row			73		80	7	7	77	27	95	2	90
Disguised RS Redundancy Row			7		58	28	37	94	88	86	63	64
Disguised RS Redundancy Row			2		19	50	72	48	4	5	4	86
Disguised Execution Trace			56	3	44	94	75	84	38	33	79	79
Disguised RS Redundancy Row			16		4	60	18	38	7	40	88	61
Disguised RS Redundancy Row			60		21	53	42	10	63	28	28	33
Disguised RS Redundancy Row			31		67	60	35	29	7	46	84	82
Disguised Random Padding			92	4	51	16	56	37	26	42	22	59
Disguised RS Redundancy Row			54		70	95	86	74	60	34	0	44
Disguised RS Redundancy Row			57		58	7	48	8	50	6	14	27
Disguised RS Redundancy Row			44		12	44	90	13	73	11	22	36
Disguised Random Padding			68	5	65	42	78	60	7	74	72	10
Disguised RS Redundancy Row			61		28	66	44	52	37	78	23	91
Disguised RS Redundancy Row			59		24	79	44	28	70	73	87	41
Disguised RS Redundancy Row			3		18	57	28	38	59	94	7	69
Disguised Random Padding			84	6	79	39	94	34	57	44	58	51
Disguised RS Redundancy Row			24		34	69	26	37	34	26	51	64
Disguised RS Redundancy Row			90		82	56	53	3	84	56	26	55
Disguised RS Redundancy Row			95		53	70	78	50	70	47	26	60
Disguised Random Padding			41	7	80	86	18	0	26	13	66	68
Disguised RS Redundancy Row			81		43	84	71	10	89	51	36	63
Disguised RS Redundancy Row			37		1	75	10	62	66	10	38	66
Disguised RS Redundancy Row	5*28^31 mod 97	5^93 mod 97	66		42	91	28	22	54	34	59	24
	5*28^32 mod 97	5^96 mod 97	5									