Plonk

¿Qué es Plonk?

Ariel Gabizon, Zachary J. Williamson, Oana Ciobotaru

Permutations over
Lagrange-bases for
Oecumenical
Noninteractive arguments of
Knowledge

Protocolo zkSNARK

Forma de aritmetización

¿Qué podemos hacer con Plonk?

¿Qué NO es Plonk?

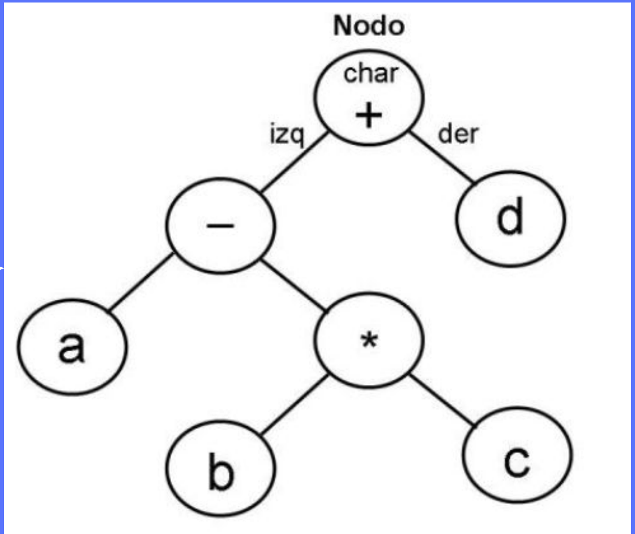
No es Plonky2

No es una implementación, es un protocolo (hay implementaciones)

No es una forma de expresar un programa arbitrario como “circuitos aritméticos zk”

¿Qué **NO** es Plank?

```cpp
#include <iostream>
#include <string>

template <typename Traits>
class Information{
    private:
        Traits pf_name;         // fp == public field name
        Traits pf_family;       // fp == public field family

    public:
        void set_information(Traits arg_name, Traits arg_family){
            pf_name = arg_name;
            pf_family == arg_family;
        }

        void get_information(){
            std::cout << "Your name is " << pf_name << " " << pf_family << std::endl;
        }
};

auto main(int argc, char* argv[]) -> decltype(0){
    Information<std::string> o_person;

    o_person.set_information("Milad", "Kahsari Alhadi");
    o_person.get_information();

    return 0;
}
```



¿Qué vamos a ver hoy?

Qué componentes tiene la aritmetización

En qué consiste un circuito de Plonk

Qué restricciones establece

Cómo expresar (casi) todo esto en polinomios

Protocolo (versión incompleta)

¿Qué NO vamos a ver hoy?

Argumento de permutación ← Miércoles

Cómo funcionan los oráculos:
Polinomial Commitment Schemes (KZG, FRI) ← Jueves

Protocolo de principio a fin ← Viernes


A LOS
BIFES
GETING STRAIGHT TO THE POINT

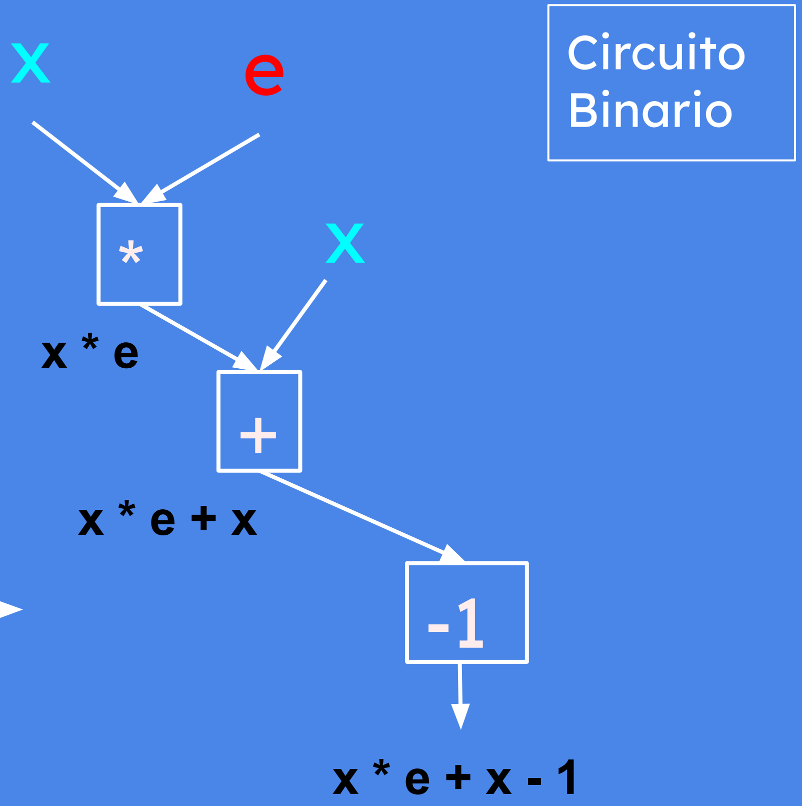
Partimos de un programa básico

PUBLIC INPUT: x
PRIVATE INPUT: e
OUTPUT: e * x + x - 1

Programa público

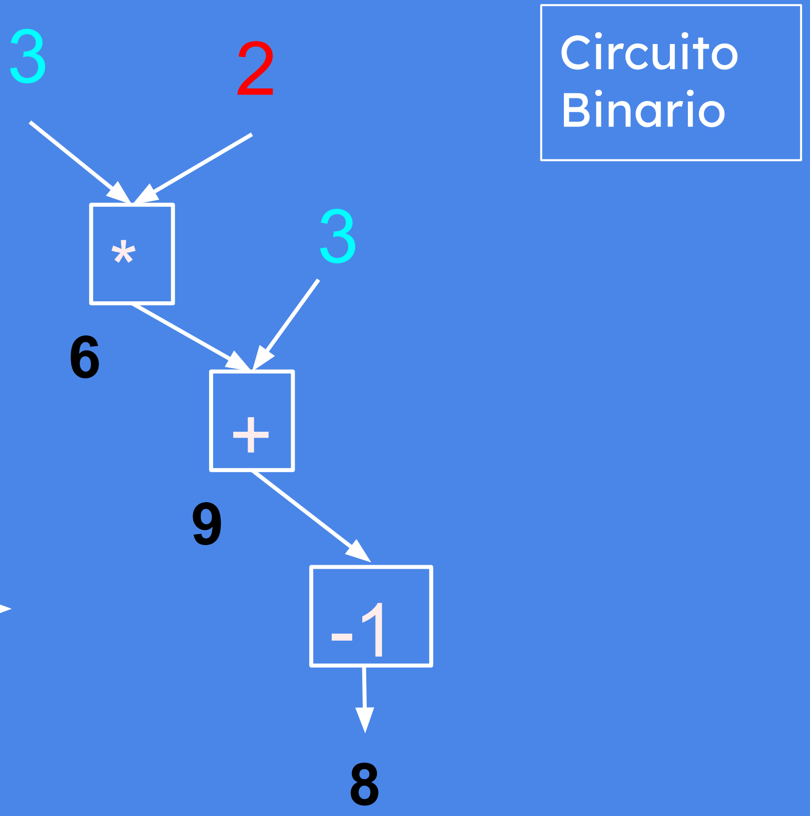
Circuitos en abstracto

- PUBLIC INPUT: x
- PRIVATE INPUT: e
- OUTPUT: e * x + x - 1



Circuitos con valores concretos

- PUBLIC INPUT: 3
- PRIVATE INPUT: 2
- OUTPUT: 2 * 3 + 3 - 1



La traza

A	B	C
2	3	6
6	3	9
9	-	8

También llamado “witness” o “testigo”

Valores concretos que toma el circuito

Una fila por cada compuerta

Depende de la ejecución concreta

La matriz Q

Columnas: $Q_L \, Q_R \, Q_M \, Q_O \, Q_C$

Restricción: $Q_L A + Q_R B + Q_M AB + Q_O C + Q_C = 0$

Determina nuestro circuito, independientemente de los inputs

La traza se relaciona con la traza a través de la restricción

Cada fila de la matriz representa una compuerta aritmética

Multiplicación

Traza

Q_L	Q_R	Q_M	Q_O	Q_C
0	0	1	-1	0

A	B	C
2	3	6

2*0 + 3*0 + 2*3*1 + 6*(-1) + 0 = 0

⟷

2*3*1 + 6*(-1) = 0

Suma

Traza

Q_L	Q_R	Q_M	Q_O	Q_C
1	1	0	-1	0

A	B	C
6	3	9

6*1 + 3*1 + 6*3*0 + 9*(-1) + 0 = 0

6*1 + 3*1 + 9*(-1) = 0

Constantes

Q_L	Q_R	Q_M	Q_O	Q_C
1	0	0	-1	-1

Traza

A	B	C
9	-	8

$9*1 + - *0 + 9* - *0 + 8*(-1) + (-1) = 0$

$9*1 + 8*(-1) + (-1) = 0$

Matrices Q y Traza de la mano

Programa

Q_L	Q_R	Q_M	Q_O	Q_C
0	0	1	-1	0
1	1	0	-1	0
1	0	0	-1	-1

Satisface la ecuación:

Traza

A	B	C
2	3	6
6	3	9
9	-	8

Las restricciones se cumplen individualmente para cada fila... ¿falta algo?

Matrices Q y T... no son suficientes

Programa

Q_L	Q_R	Q_M	Q_O	Q_C
0	0	1	-1	0
1	1	0	-1	0
1	0	0	-1	-1

Satisface la ecuación:

Traza

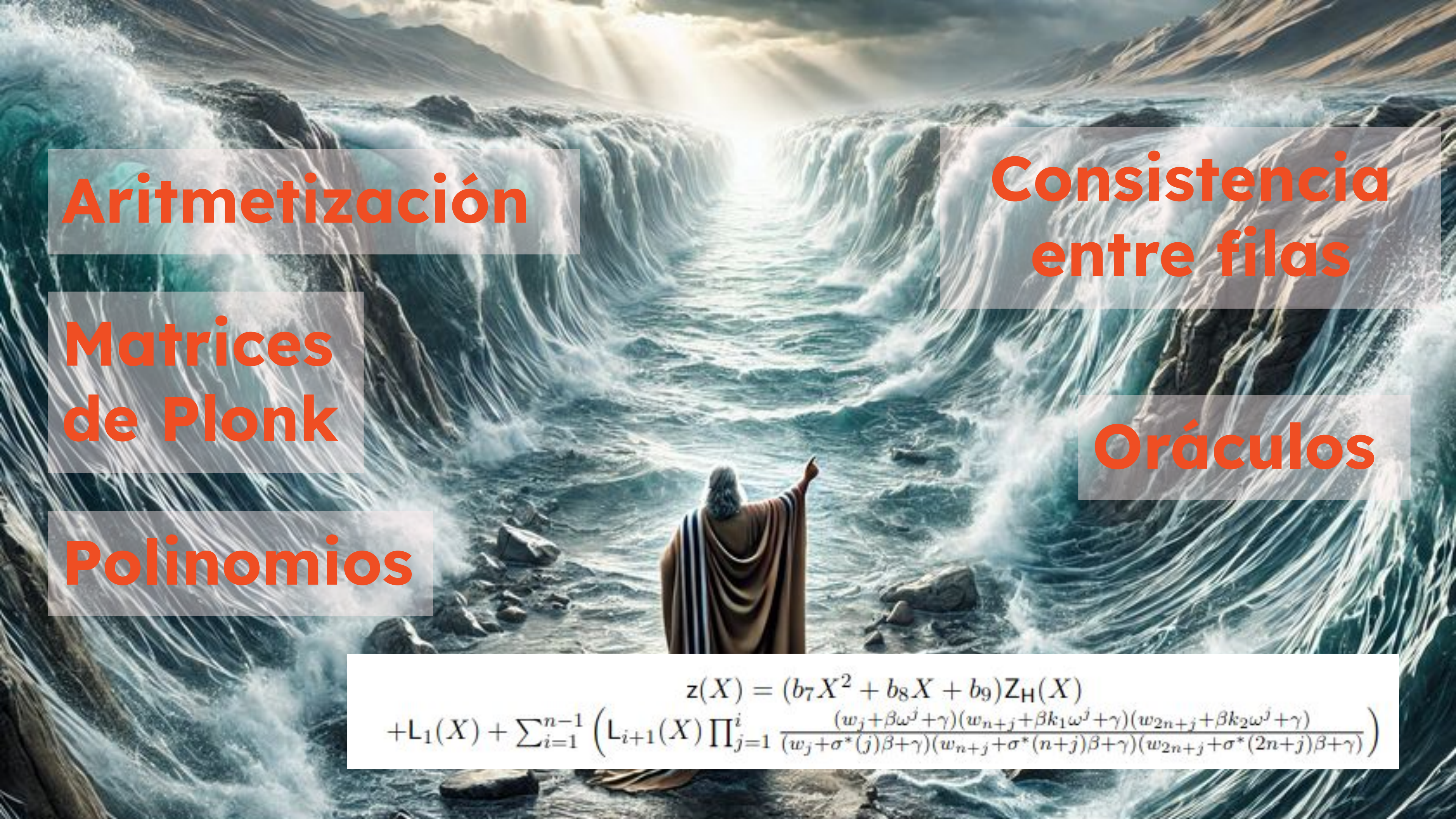
A	B	C
2	3	6
0	0	0
20	-	19

No se bajen .png de google imágenes

No tenemos forma de relacionar las filas entre sí... todavía

Consistencia entre filas

A	B	C
2	3	6
6	3	9
9	-	8



Aritmetización

Matrices de Plonk

Polinomios

Consistencia entre filas

Oráculos

$$
\begin{aligned}
\mathsf{z}(X) &= (b_7X^2 + b_8X + b_9)\mathsf{Z}_{\mathsf{H}}(X) \\
&+\mathsf{L}_1(X) + \sum_{i=1}^{n-1}\left(\mathsf{L}_{i+1}(X)\prod_{j=1}^{i}\frac{(w_j+\beta\omega^j+\gamma)(w_{n+j}+\beta k_1\omega^j+\gamma)(w_{2n+j}+\beta k_2\omega^j+\gamma)}{(w_j+\sigma^*(j)\beta+\gamma)(w_{n+j}+\sigma^*(n+j)\beta+\gamma)(w_{2n+j}+\sigma^*(2n+j)\beta+\gamma)}\right)
\end{aligned}
$$

La matriz V

Columnas: `L R O`

Restricción: $\forall i, j, k, l \; V_{i,j} = V_{k,l} \implies T_{i,j} = T_{k,l}$

Determina nuestro circuito, independientemente de los inputs

La traza se relaciona con la traza a través de la restricción

Cada fila de la matriz V representa un “nombramiento” de variables

Mucha data junta

Ahora pensemos un poco

¿Cómo podemos comprimir nuestro programa en una única compuerta de Plonk?

PUBLIC INPUT: x
PRIVATE INPUT: e
OUTPUT: e * x + x - 1

Queremos que nuestro programa tenga una sola fila

¿Q?

Traza

A	B	C
2	3	8

PUBLIC INPUT: x
PRIVATE INPUT: e
OUTPUT: e * x + x - 1

$$A_i Q_{Li} + B_i Q_{Ri} + A_i B_i Q_{Mi} + C_i Q_{Oi} + Q_{Ci} = 0$$

Queremos que nuestro programa tenga una sola fila

Q

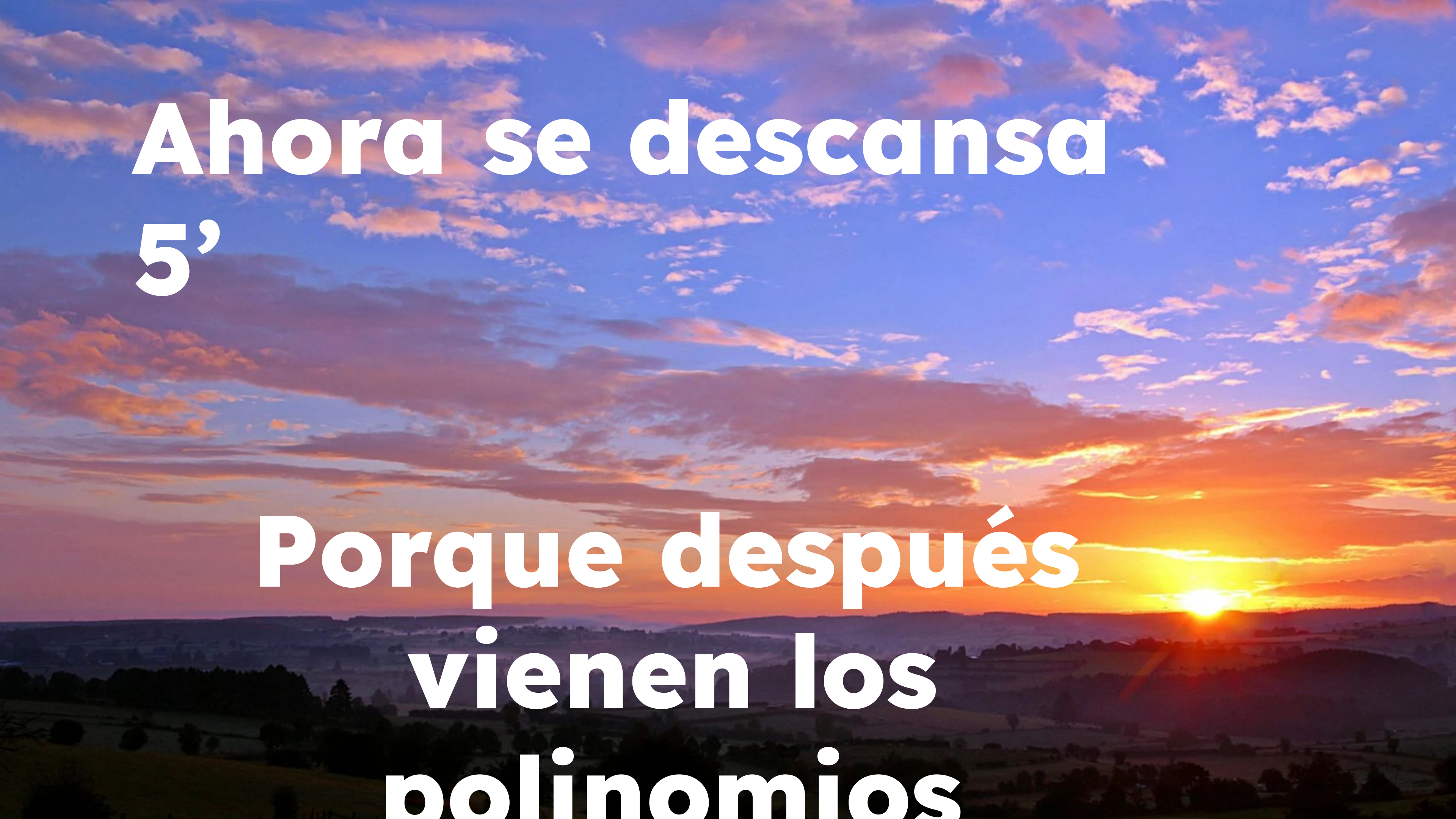
Q_L	Q_R	Q_M	Q_O	Q_C
1	1	1	-1	-1

Traza

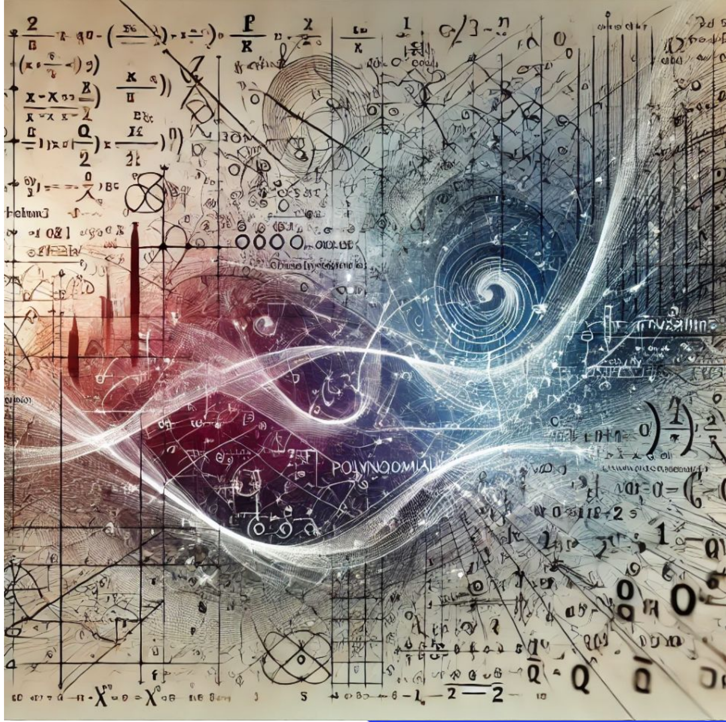
A	B	C
2	3	8

$2*1+3*1+2*3*1+8*(-1)+(-1)=0$

PUBLIC INPUT: x
PRIVATE INPUT: e
OUTPUT: e * x + x - 1


Ahora se descansa 5'
Porque después vienen los polinomios

Polinomios



¿Qué queremos?

- Demostrar que conocemos una traza T que cumple las restricciones dadas por Q (nuestro programa)

- No revelar los Private Inputs

- Que la verificación sea rápida

Público

Q (programa)

Cuerpo finito: F_p

Privado

T (traza)

Vamos a interpolar

Interpolamos sobre el dominio H

- Las columnas A, B y C de la traza
- Las columnas Q_L, Q_R, Q_M, Q_O, Q_C de la matriz Q

Obtenemos 8 polinomios a, b, c, q_L, q_R, q_M, q_O, q_C

$a(0) = A[i]$

$b(1) = B[i]$

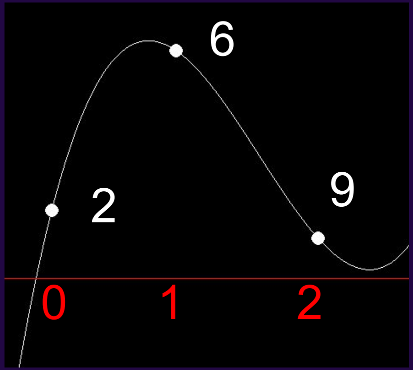
$c(2) = C[i]$

A	B	C
2	3	6
6	3	9
9	-	8

$a(0) = 2$

$a(2) = 6$

$a(2) = 9$



Settings del protocolo

```
F = GF(11)
Polynomial = PolynomialRing(F, 'X')
dominio = [0,1,2]
```

Programa

```
QL = [0,1,1]
QR = [0,1,0]
QM = [1,0,0]
QO = [-1,-1,-1]
QC = [0,0,-1]
```

Q_L	Q_R	Q_M	Q_O	Q_C
0	0	1	-1	0
1	1	0	-1	0
1	0	0	-1	-1

Traza

```
A = [2,6,9]
B = [3,3,0]
C = [6,9,8]
```

A	B	C
2	3	6
6	3	9
9	-	8

Interpolamos

```
a = Polynomial.lagrange_polynomial(zip(dominio,A))
b = Polynomial.lagrange_polynomial(zip(dominio,B))
c = Polynomial.lagrange_polynomial(zip(dominio,C))
qL = Polynomial.lagrange_polynomial(zip(dominio,QL))
qR = Polynomial.lagrange_polynomial(zip(dominio,QR))
qM = Polynomial.lagrange_polynomial(zip(dominio,QM))
qO = Polynomial.lagrange_polynomial(zip(dominio,QO))
qC = Polynomial.lagrange_polynomial(zip(dominio,QC))
```

```
a: 5*X^2 + 10*X + 2
b: 4*X^2 + 7*X + 3
c: 9*X^2 + 5*X + 6
qL: 5*X^2 + 7*X
qR: 10*X^2 + 2*X
qM: 6*X^2 + 4*X + 1
qO: 10
qC: 5*X^2 + 6*X
```

```
f = a*qL + b*qR + a*b*qM + c*qO + qC
```

```
f: 10*X^6 + 2*X^5 + 8*X^4 + 5*X^3 + 7*X^2 + X
```

```
print(f(dominio[0]), f(dominio[1]), f(dominio[2]))
```

```
0 0 0
```

¿Qué dominio usamos?

Tomamos un generador de Fp de orden N: w

$$D = \{w^i : 0 \leq i \leq N\}$$

Un único polinomio

Definimos el polinomio f como

$$f = q_L \cdot a + q_r \cdot b + q_M \cdot a \cdot b + q_O \cdot c + q_c$$

¿Les suena de algo?

$$Q_L A + Q_R B + Q_M AB + Q_O C + Q_C = 0$$

Entonces podemos decir que si la traza es válida entonces

$$q_L(x)a(x) + q_R(x)b(x) + q_M(x)a(x)b(x) + q_O(x)c(x) + q_C(x) = 0 \;\forall x \in D$$

$$\Longleftrightarrow \quad f(x) = 0 \;\forall\, x \in \{w^i : 0 \leq i < N\}$$

Para convencernos

$$a(w^i) = A[i]$$

Fórmulas dump

$$f(x) = 0 \; sii \, x \in \{w^i : 0 \leq i < N\}$$

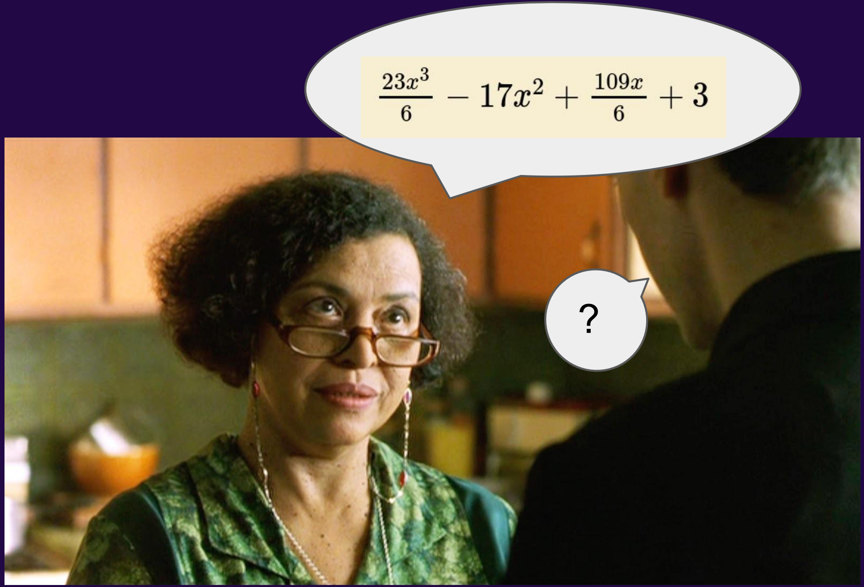
$$\left(x - w^i\right) \Big| \; f(x)$$

$$\prod_{i=0}^{N-1} \left(x - w^i\right) \quad \Big| \; f(x)$$

$$z_D = \prod_{i=0}^{N-1} \left(x - w^i\right) = x^n - 1$$

$$\exists t : f = z_D \cdot t$$

Oráculos de polinomios



Llamamos [a] al oŕaculo de un polinomio a(x)

[a] nos permite evaluar a(x) solo una vez y en un valor aleatorio

Llamamos $[a]_z$ al valor de a(z) dado por el oráculo

Schwartz–Zippel lemma

Dado un cuerpo finito Fp con p grande y 2 polinomios P y Q de grado $n<<p$ y sus respectivas funciones asociadas $p,q: Fp \to Fp$, dado un valor aleatorio z tomado de una distribución uniforme sobre Fp, decimos que si $p(z)=q(z)$ entonces con altísima probabilidad $P = Q$.

¿Cuál es la probabilidad de que NO sean el mismo? p y q se cortan en a lo sumo n puntos (por su grado), y el dominio tiene p puntos, entonces la probabilidad de que justo sea uno de esos puntos es n/p.

Protocolo (parcial)

Prover → le da al verifier los oráculos [a], [b], [c] y [t]

¿Qué pasa con q_R, q_L, q_M, q_o y q_C?
Son públicos, los puede calcular el verifier por su cuenta

Verifier → interpola las matrices y obtiene q_R, q_L, q_M, q_O y q_C.

A partir de esto, alcanza con hacer la verificación…

$$q_L(z)[a]_Z + q_R(z)[b]_z + q_M(z)[a]_z[b]_z + q_O(z)[c]_z + q_C(z) \;=\; z_D(z)[t]_z$$

…para convencerse de que el prover tiene una traza válida.

Nota sobre complejidad

¿Qué operaciones estamos haciendo?

Prover:

- Interpolación de las trazas
- División de polinomios

Verifier:

- Cuentas en Fp

Oráculos:

- ??????

¿Y los inputs?

¿Dónde están los inputs?

Inputs... públicos? privados? output?

Necesitamos una forma de demostrar que el programa se ejecutó con ciertos valores públicos (conocidos por ambas partes)

Los outputs del programa son valores públicos

No estamos ejecutando nada, estamos demostrando que conocemos una traza válida.

Los valores privados están metidos en la traza

Versión completa

Programa

Q_L	Q_R	Q_M	Q_O	Q_C
-1	0	0	0	0
-1	0	0	0	0
1	1	1	-1	1
1	-1	0	0	0

Ejecución

PI
3
8
0
0

A	B	C
3	-	-
8	-	-
2	3	8
8	8	-

Restricción actualizada

$$A_i Q_{Li} + B_i Q_{Ri} + A_i B_i Q_{Mi} + C_i Q_{Oi} + Q_{Ci} + \underline{PI_i} = 0$$

Momento de Plonk Adventures

1

Estamos haciendo la versión acotada del protocolo, sin inputs ni consistencia entre filas

2

La implementación de los oráculos va a ser mucho más simple que lo que en realidad son

3

El programa no va a tener inputs


¡Gracias!