



Distcom

Alumno: Agustín Cambiano

Tutor: Leandro Ferrigno

Sistema de donación de cómputo: World Community Grid



- La mayoría de las personas tienen dispositivos tecnológicos más potentes de lo necesario
- En vez de donar dinero a asociaciones, puede donarse el resultado del cálculo de operaciones



Este sistema tiene un gran problema

Un donante malicioso podría mentir en su resultado, contaminando así los datos, o forzando a las organizaciones a solicitar ayuda únicamente para problemas con resultados fácilmente verificables

Posible solución



ethereum

- Imitar parcialmente a ethereum y armar una red de cómputo con redundancia
- Debido a la poca cantidad de usuarios que usarán la red (por tratarse de donaciones), esta sería muy susceptible a un ataque del 51%



Solución real: Pruebas de integridad de cómputo

- Estas nos permiten demostrar matemáticamente con alta probabilidad que el resultado de una operación es verídico
- Mezclan muchos conceptos matemáticos
- Permiten generar un archivo de prueba que luego puede ser verificado por quien utiliza el resultado



Proyecto implementado



Distcom

Es una red que conecta organizaciones y donantes

Las organizaciones suben código para ejecutar

Los donantes lo descargan y suben el resultado junto con pruebas de integridad

Las organizaciones descargan las pruebas y verifican si son válidas antes de aceptar los resultados donados



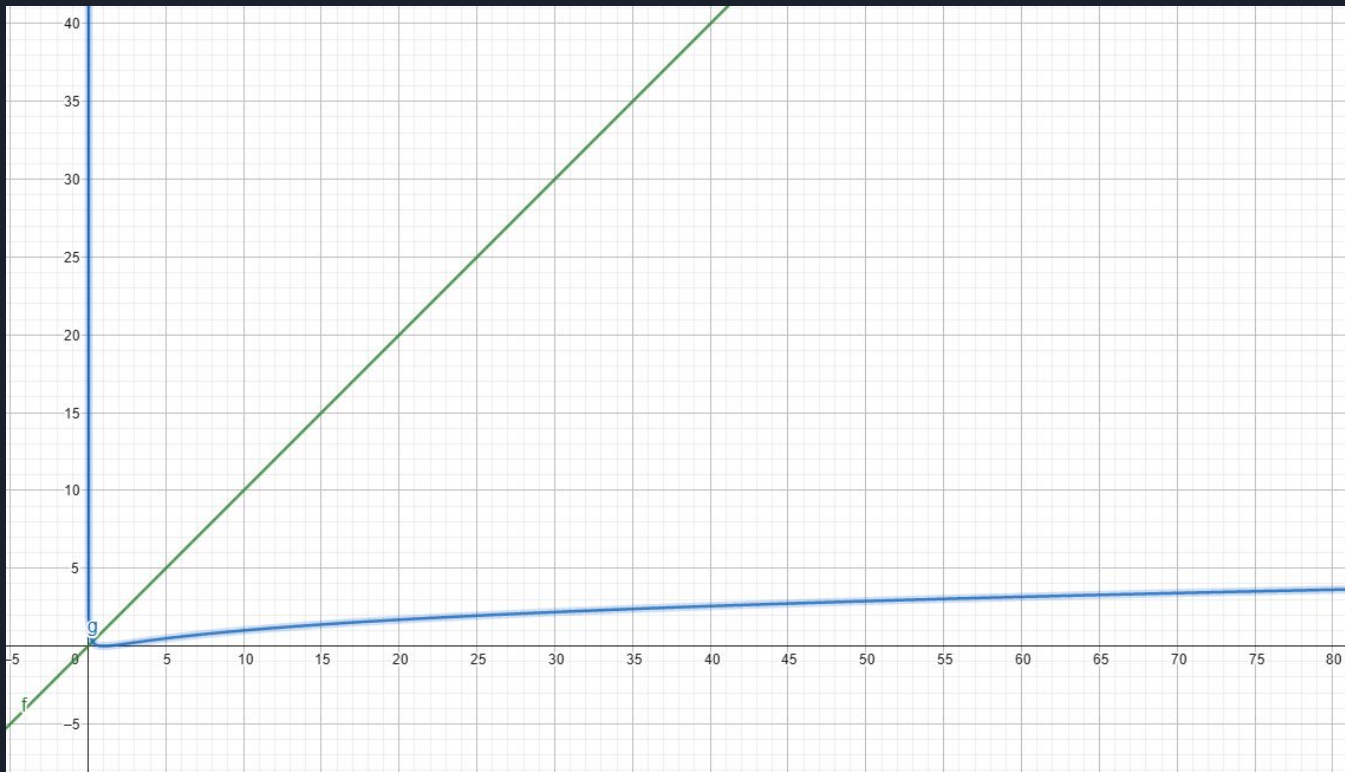
Razones de uso del sistema

Aunque el orden de generación de la prueba es $n \cdot \text{polylog}(n)$, el de verificación es $\text{polylog}(n)$, siendo n la cantidad de instrucciones ejecutadas normalmente en el programa

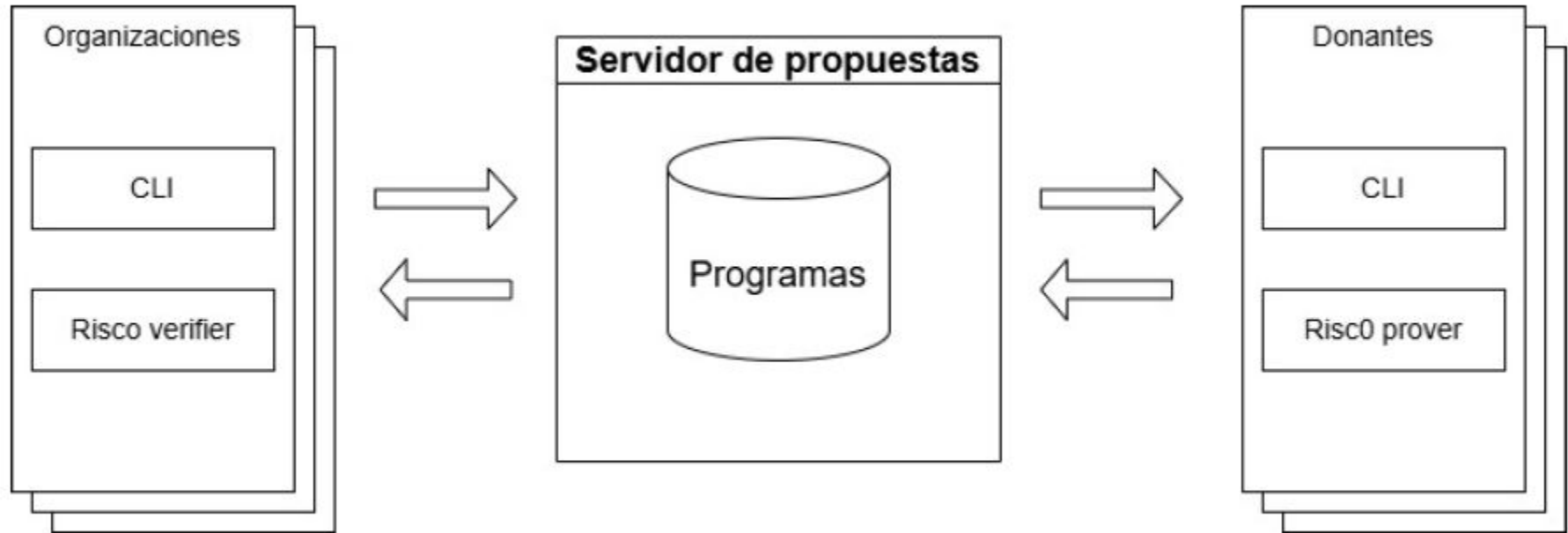
Al tardar idealmente la verificación menos que la ejecución normal, conviene a la organización delegar el cómputo a donantes para luego verificarlo en vez de calcularlo por su cuenta

Esto permite recibir donaciones de resultados difíciles de corroborar como válidos, ya que se podrá simplemente verificar la prueba

x vs $\log^2(x)$



Se implementó un cliente para las organizaciones, otro para los donantes, y un servidor para que estos clientes puedan interactuar entre sí. Todos los programas se encuentran dockerizados para facilidad de uso.





Ciente de organizaciones

Please execute a command:

help upload

Upload of the methods folder of the code the user wants to upload

Usage: upload --path <FOLDER_NAME> --name <NAME> --description <DESCRIPTION> --timeout <EXECUTION_TIMEOUT>

Options:

-p, --path <FOLDER_NAME>	Name of the methods folder in the uploads directory
-n, --name <NAME>	Name for the program what will be displayed for the provers
-d, --description <DESCRIPTION>	Explanation of the objectives of the program or any additional information considered necessary
-t, --timeout <EXECUTION_TIMEOUT>	How many seconds an input group of this program will be blocked before another prover can request it for execution
-h, --help	Print help

Es un programa de consola implementado en rust, permite subir el código a ejecutar, subir múltiples posibles inputs para ese código, y luego verificar las pruebas de los resultados de esos inputs una vez subidas al servidor. Utiliza Risc0 para la verificación de las pruebas.



Cliente de donantes

Please execute a command:

help

Usage: <COMMAND>

Commands:

organizations Displays a list with the information of the organizations stored in the program distributor, moves the execution to another commands set

all-programs Displays a list with the information of the programs stored in the program distributor without taking the uploader into account, moves the execution to another commands set

exit Exits the program

help Print this message or the help of the given subcommand(s)

Options:

-h, --help Print help

-V, --version Print version

Es un programa de consola implementado en rust, descargar el código a ejecutar, generar la prueba luego de obtener el resultado, y luego subirla al servidor. Utiliza Risc0 para la generación de las pruebas con una virtual machine.



Servidor

Es un servidor HTTP que almacena las cuentas de las organizaciones

Maneja las reservas de los inputs (para prevenir doble cálculo de parte de donantes)

Recibe y retorna las pruebas de integridad de cómputo, utilizando AWS para su almacenamiento

No hace uso de Risc0



Demo



Conceptos teóricos



Zero Knowledge proofs

Procedimiento por el cual un prover demuestra a un verifier que cierto statement es verdad sin revelar ningún tipo de información adicional.

En otras palabras, es demostrar que $f(x) = y$, sin revelar x , es decir, probando únicamente que cierto elemento desconocido posee cierta propiedad sin revelar el elemento en sí.



Ejemplo: pelotas de colores

Luego de n pruebas, la probabilidad de que haya mentido es
 $(\frac{1}{2})^n$

Por lo que la probabilidad de que la prueba haya sido exitosa es

$$1 - (\frac{1}{2})^n$$

$$\text{Si } n = 10 \Rightarrow 1 - (\frac{1}{2})^{10} = 0,999 \Rightarrow 99,9\%$$




Teoría de polinomios




Corroboración de grado de polinomios

- Se quiere ver si los puntos de una lista pertenecen a un polinomio de grado $d-1$
- Un polinomio de grado $d-1$ está definido por d puntos
- Dado un set de $m > d$ puntos, se necesita utilizar al menos $d+1$ puntos para ver si pertenecen a un polinomio de grado d
- Abandonando el determinismo, se puede optar por una elección de un punto aleatorio luego de definir el polinomio con d puntos. Dada una proporción de p puntos inválidos, la probabilidad de un falso positivo es $1-p$



Corroboración de grado de polinomios: reducción de puntos de testeo

- Dados dos polinomios A y B de grado $d-1$, el polinomio $C = A+B$ probablemente también tendrá grado $d-1$
- Al tener C probablemente un grado $d-1$, se deberán probar $d+1$ puntos, en vez de los $2(d+1)$ puntos que se deberían probar para testear A y B por separado
- El tradeoff es la posibilidad de que A y B sean maliciosos y tengan grados mayores y se cancelen, pero es un caso extremadamente poco probable



Corroboración de grado de polinomios: reducción de grado

Esto puede utilizarse para reducir a la mitad el grado de un polinomio, separando los coeficientes pares e impares en dos polinomios y reduciendo su grado a la mitad para que sean sumados. Así al sumarlos se genera un polinomio de mitad de grado, mucho más fácil de interpolar

Se genera así otro set de puntos pertenecientes al nuevo polinomio de mitad de grado, con la mitad de cantidad de elementos contenidos

$$Q_0(x) = pares_0(x) + x.impares_0(x)$$

$$pares_0(x) = a_0 + a_2X^2 + a_4X^4 + a_6X^6 + \dots$$

$$x.impares_0(x) = x(a_1 + a_3X^2 + a_5X^4 + a_7X^6 + \dots)$$

$$pares_0(x) = P_0(X^2)$$

$$impares_0(x) = I_0(X^2)$$

$$Q_1(x) = P_0(x) + I_0(x)$$

$$P_0(x^2) = \frac{Q_0(x) + Q_0(-x)}{2}$$

$$I_0(x^2) = \frac{Q_0(x) - Q_0(-x)}{2x}$$



Restricciones polinomiales

Dado un polinomio $A(x)$, pueden corroborarse propiedades al hacer una composición con otro polinomio $B(x)$

Al hacer $B(A(x))$, se ve que los puntos en los que se quiere que se corrobore la propiedad generen un cero

Si se quiere corroborar que $A(x)$ es 1, 2 o 3 para $1 \leq x \leq 100$, la restricción es

$$B(x) = (1 - x)(2 - x)(3 - x)$$

Y se corrobora que $B(A(x)) = 0$ para $1 \leq x \leq 100$



División de polinomios

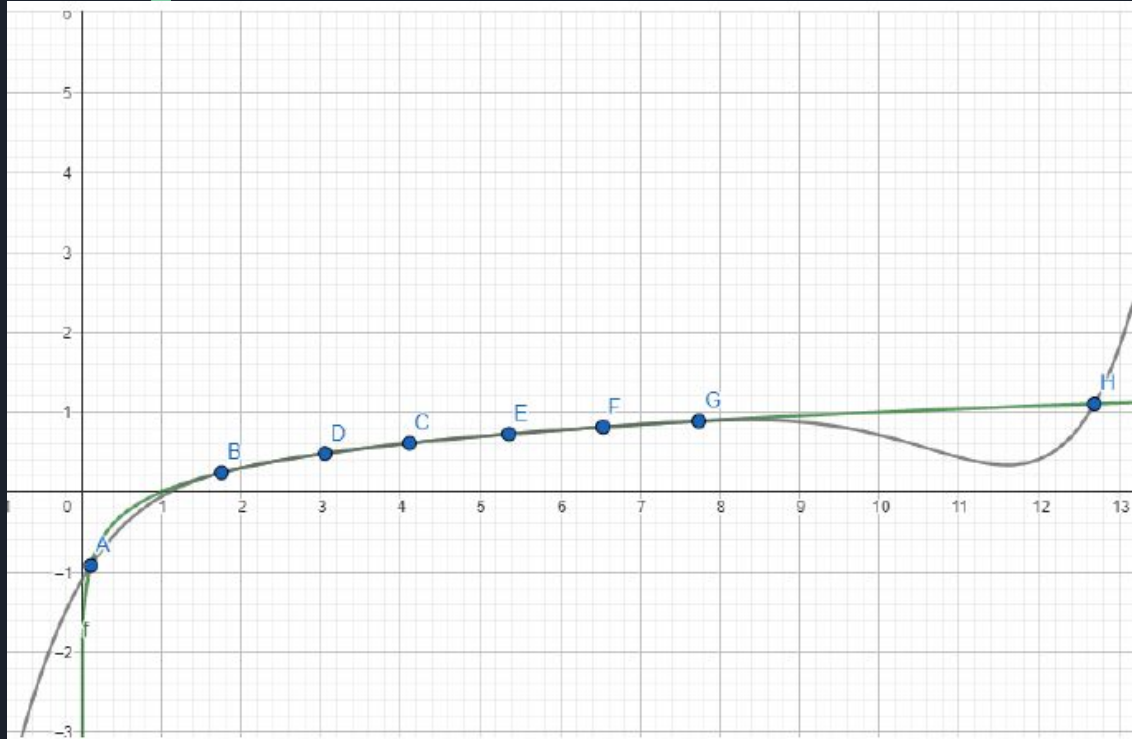
Todo polinomio puede ser expresado como producto de un escalar y los ceros que lo componen

$$P(x) = a \cdot \prod (x - x_i)$$

Se ve que un polinomio puede dividirse únicamente por sus ceros, si se divide por otra cosa el resultado no será un polinomio

Toda serie de puntos puede ser representada por un polinomio, por eso en realidad se ve si los puntos resultantes de la división son de un polinomio de grado alto

Polinomio vs logaritmo



Con una muestra no se puede saber si esos puntos vienen de una función específica o de un polinomio de grado lo suficientemente alto



STARKS

- Es una ZKP que demuestra la ejecución correcta de un cómputo
- Se generan en orden $n \cdot \text{polylog}(n)$ y se verifican en $\text{polylog}(n)$, la verificación es en teoría más rápida que la ejecución



Explicación básica

Dada una o varias listas de valores de los pasos de un cómputo, se quiere corroborar que las transiciones entre estos sean siempre válidas

Para esto se definen restricciones que relacionan las distintas celdas entre sí, y deben ser cumplidas

El conjunto de estas listas se denomina traza de ejecución

Traza ejemplificada: restricciones de factura de compra

Item	Precio	Total actual
Producto 1	1,15	0
Producto 2	2,36	1,15
Producto 3	9,64	3,51
Producto 4	2	13,15
Total	15,15	15,15

$T_{a,b}$: celda de columna a, fila b

Se verifica con operaciones sobre las celdas que todos los cálculos son válidos. Se tiene una columna con los precios y otra con la suma acumulada.


Suma de valor actual y acumulado previo es válida:

$$T_{1,i} + T_{2,i} - T_{2,i+1} = 0$$

El acumulador empieza en cero: $T_{1,2} = 0$

El total es igual a la suma acumulada final:

$$T_{5,1} - T_{5,2} = 0$$



Corroborar que se cumplan las restricciones para todas las celdas resultaría muy costoso (igual o más que realizar el cómputo nuevamente)

Realizar un muestreo de celdas a las cuales se les realiza una corroboración de restricciones probablemente ignoraría una transición inválida, contaminando el resto del cómputo

Para solucionar esto, se introducen polinomios para incrementar la cantidad de puntos erróneos en caso de que estos existan, permitiendo un muestreo más útil y eficiente



Composición real de la traza

Cada lista se convierte en una columna de la traza

Cada columna conforma un polinomio, en el que los valores de sus celdas serán la imagen, interpolada sobre puntos de la forma g^i , $i=0\dots$

De esta forma pueden aplicarse las restricciones sobre la traza corroborando que en los puntos de interpolación correspondientes hay ceros



Puntos de cumplimiento de las restricciones

Un polinomio solo puede ser dividido por sus ceros

Para ver si una restricción se cumple donde se debe, puede dividirse por los puntos en los que debe cumplirse, es decir, sus ceros

División polinomial de la traza de la factura:

$$D_0(x) = \frac{C_0(x)}{Z_0(x)} = \frac{t_1(x) + t_2(x) - t_2(x.g)}{(x - g)(x - g^2)(x - g^3)(x - g^4)}$$

$$D_2(x) = \frac{C_2(x)}{Z_2(x)} = \frac{t_1(x) - t_2(x)}{x - g^5}$$

$$D_1(x) = \frac{C_1(x)}{Z_1(x)} = \frac{t_2(x) - 0}{x - g}$$



Generación de polinomio de composición

Dadas las restricciones y los puntos en los que deben cumplirse, se tendrán n polinomios $D_i(x)$

Para que se cumplan todas las restricciones, todos los D_i deben ser polinomios de grado bajo, por lo que la suma de estos también debe serlo, se define así el polinomio de composición $H(x)$

$$H(x) = \sum \alpha_i D_i$$

$$H(x) = H_1(x^2) + x.H_2(x^2)$$



Generación final de la prueba

Si se corrobora que $H_0(x)$ tiene el grado esperado (que depende de las restricciones), entonces todas las restricciones son válidas y también los valores finales de la prueba (también definidos con restricciones)

Así se define la prueba como válida



Generación final de la prueba

Se utiliza el algoritmo ya mencionado para determinación del grado de un polinomio, agregando un escalar para agregar aleatoriedad

$$H'_{i+1}(x) = P_i(x) + \alpha \cdot I_i(x)$$

Si $\text{grado}(H'_0(x)) = d$, luego de $\log_2(d)$ pasos se debería llegar a una constante, sino el grado era mayor al esperado y fallará la prueba

En cada iteración de generación de $H'_i(x)$ se realizará un commit sobre los valores



Verificación de la prueba

Una vez hecho todo esto por parte del prover, el verifier realizará queries sobre los distintos polinomios H' , eligiendo puntos aleatorios y corroborando que en todos los pasos el cálculo de la imagen del siguiente paso sea correcto.

Por cada test realizado sobre H'_0 se ganan L bits de seguridad, siendo L la potencia de 2 que genera el blowup factor (blowup factor = 2^L). Un blowup factor de 4 con 50 queries genera aproximadamente 100 bits de seguridad



ZKVM (Zero Knowledge Virtual Machine)

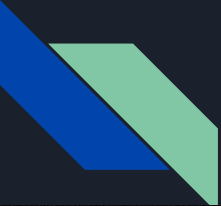
Es una aplicación específica de STARKS, con restricciones que definen el funcionamiento normal de una computadora

Es una ejecución de STARKS que demuestra que se ejecutó un programa específico en vez de generar un set de restricciones para cada programa a resolver, que todas las transiciones de los valores en la memoria y los registros son válidos, y que el resultado generado pertenece a la memoria

$$\text{res} = f_{\text{RES_ADD}} \cdot (\text{op0} + \text{op1}) + f_{\text{RES_MUL}} \cdot \text{mul} + (1 - f_{\text{RES_ADD}} - f_{\text{RES_MUL}}) \cdot \text{op1}$$



Mediciones y conclusión



Comparación de eficiencia

```
Starting proof verification of program with id "bd0c650d-1043-4541-9ec8-c90e17d452b9" with input group with id "f21267a0-b9ac-49cd-8570-8909cbalf7f1"
downloaded_guest: Starting build for riscv32im-risc0-zkvm-elf
downloaded_guest:   Compiling downloaded_guest v0.1.0 (/app/src/runner/methods/guest)
downloaded_guest: warning: unused variable: `cast_base`
downloaded_guest:   --> src/main.rs:28:9
downloaded_guest:   |
downloaded_guest: 28 |       let cast_base = base as u64;
downloaded_guest:   |           ^^^^^^^^^ help: if this is intentional, prefix it with an underscore: `_cast_base`
downloaded_guest:   |
downloaded_guest:   = note: `#[warn(unused_variables)]` on by default
downloaded_guest: warning: `downloaded_guest` (bin "downloaded_guest") generated 1 warning
downloaded_guest:   Finished `release` profile [optimized] target(s) in 0.41s
Proof was verified, total seconds passed: 3
```

El tiempo de verificación de un test de primalidad miller rabin es de 3 segundos

Comparación de eficiencia

```
normal_examples
└─ miller_rabin_test
   └─ inputs
      ├── 53_miller_rabin_big_endian.csv
      ├── 561_miller_rabin_big_endian.csv
      ├── 562_miller_rabin_big_endian.csv
      ├── 563_miller_rabin_big_endian.csv
      ├── 564_miller_rabin_big_endian.csv
      ├── 565_miller_rabin_big_endian.csv
      ├── 566_miller_rabin_big_endian.csv
      ├── 567_miller_rabin_big_endian.csv
      ├── 568_miller_rabin_big_endian.csv
      ├── 2371_miller_rabin_big_endian.csv
      ├── 2373_miller_rabin_big_endian.csv
      ├── 7237_miller_rabin_big_endian.csv
      ├── 7243_miller_rabin_big_endian.csv
      ├── 7247_miller_rabin_big_endian.csv
      ├── 7253_miller_rabin_big_endian.csv
      ├── 7283_miller_rabin_big_endian.csv
      ├── 7297_miller_rabin_big_endian.csv
      ├── 7307_miller_rabin_big_endian.csv
      ├── 7309_miller_rabin_big_endian.csv
      ├── 7321_miller_rabin_big_endian.csv
      └── 7331_miller_rabin_big_endian.csv

47  fn execute_miller_rabin(input: Vec<u8>) -> Outputs {
87  };
88  outputs

agustin@LAPTOP-M43B0G0S: X Windows PowerShell
agustin@LAPTOP-M43B0G0S:/mnt/c/Users/agusc/OneDrive/Documents/Facultad/distcom/normal_examples/miller_rabin_te
st$ time cargo run
warning: unused imports: `thread` and `time::Duration`
--> src/main.rs:1:29
1 | use std::{fs::{self, File}, thread, time::Duration};
  |                                     ^^^^^^ ^^^^^^^^^^^^^^^^^
= note: `#[warn(unused_imports)]` on by default

warning: `miller_rabin_test` (bin "miller_rabin_test") generated 1 warning (run `cargo fix --bin "miller_rabin_test"` to apply 1 suggestion)
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.43s
Running `target/debug/miller_rabin_test`

real    0m0.739s
user    0m0.031s
sys     0m0.083s
agustin@LAPTOP-M43B0G0S:/mnt/c/Users/agusc/OneDrive/Documents/Facultad/distcom/normal_examples/miller_rabin_te
st$ |
112 }
113 assert!(counter == 1, "There is more than one element per line");
```

El tiempo de ejecución de 29 tests de primalidad de miller rabin seguidos sin verificación es de 739 milisegundos



Conclusión

Actualmente el proyecto no es viable, para la verificación de una única ejecución se tardó más que para la ejecución de 29 ejemplos seguidos

Para la organización es mejor ejecutar el programa por su cuenta que verificar el resultado donado, necesitan programas muy computacionalmente intensos para obtener una mejora real

Sin embargo, hay mejoras pendientes en Risc 0 que podrían solucionar esto en el futuro

Upgrade	Est Perf Bump	Description
rv32im GPU optimizations	2x-4x	GPU improvements targeting our RISC-V circuit
recursion GPU optimizations	2x-4x	GPU improvements targeting our recursion circuit
rv32im v2 Circuit	2x+++	A new, more efficient and flexible RISC-V circuit



Posibles desarrollos futuros del proyecto

- UI más amigable
- Mejoras en el manejo de cuentas de organizaciones en el servidor
- Permitir la verificación y ejecución sin descargar el código de donación para compilar
- Implementación de más interfaces de manejo de archivos además de la de AWS
- **Implementación de generación de pruebas en batches, lo cual agrega una gran eficiencia en cuanto a la generación y verificación de pruebas**



Fin