Ingeniería en Informática

Facultad Politécnica

Universidad Nacional de Asunción

Descripción General

Este proyecto implementa un circuito aritmético que verifica la operación :

$$c = (a^2 + b^2) \% p$$

utilizando pruebas de conocimiento cero (ZK-proofs), donde a y b son entradas privadas y p es un número primo público.

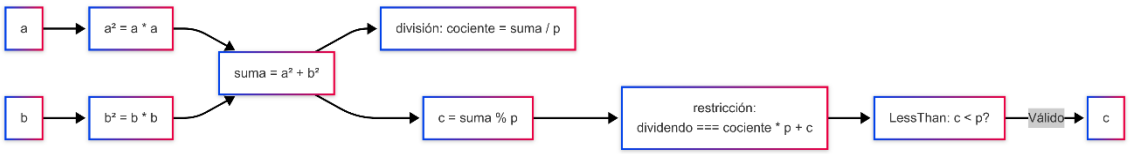
1. Estructura del Circuito

1.1. Señales y operaciones

El circuito implementado en Circom sigue esta lógica:

Tipo	Señal	Descripción
Entrada privada	a	Primera entrada privada.
Entrada privada	b	Segunda entrada privada.
Entrada pública	p	Número primo público (debe ser primo).
Salida pública	c	Resultado de $(a^2 + b^2) \% p$.

1.2. Flujo de Operaciones



1.3. Componente LessThan

- Propósito: Garantiza que el resultado c sea menor que p (crítico para la corrección del módulo).
- Bits (252):
  - El valor 252 se elige porque:
  - Soporta números muy grandes (hasta $\sim 2^{252}$).
  - Es compatible con el campo finito usado en ZK-SNARKs (BN128).

Código relevante:

```
// Usar LessThan para verificar que c < p
component lessThan = LessThan(252); // 252 es el // Usar LessThan para verificar que c < p
component lessThan = LessThan(252); // 252 es el número de bits para comparar
lessThan.in[0] <== c;               // c es la señal que queremos comparar
lessThan.in[1] <== p;               // p es el valor constante
lessThan.out === 1;                 // Asegurar que c < p
```

1.4. Restricciones Clave

- Fórmula de la división:

  Restricción para asegurar que dividendo = cociente * p + c

```
dividendo === cociente * p + c;
```

- Validación de p:

  El circuito **no** verifica que p sea primo.

2. Proceso de Generación de Pruebas

2.1. Archivos Generados

Archivo	Propósito
circuit.wasm	Código compilado para generar el witness.
circuit.r1cs	Restricciones del circuito en formato binario.
input.json	Valores de entrada para el witness
witness.wtns	Señales calculadas a partir de las entradas.
circuit_final.zkey	Claves para pruebas/verificación (Groth16).

2.2. Trusted Setup

- ¿Por qué es necesario?
  - Groth16 requiere una ceremonia inicial (Trusted Setup) para generar claves seguras.
  - El archivo .ptau (pot12_final.ptau) contiene parámetros criptográficos.
- ¿Como se genera?
  - **Fase 1**: Creación inicial del .ptau (usa entropía aleatoria).

```
snarkjs powersoftau new bn128 12 "outputs/pot12_0000.ptau" -v
```

  - **Contribución**: Se añade entropía para descentralizar la confianza.

```
ENTROPY=$(head -c 1024 /dev/urandom | base64)
    echo "$ENTROPY" | snarkjs powersoftau contribute...
```

3. Proceso de verificación

3.1. Verificaciones

Verificación en Node.js	Script: verify.js
Verificación en navegador	Archivo: browser-verifier.html

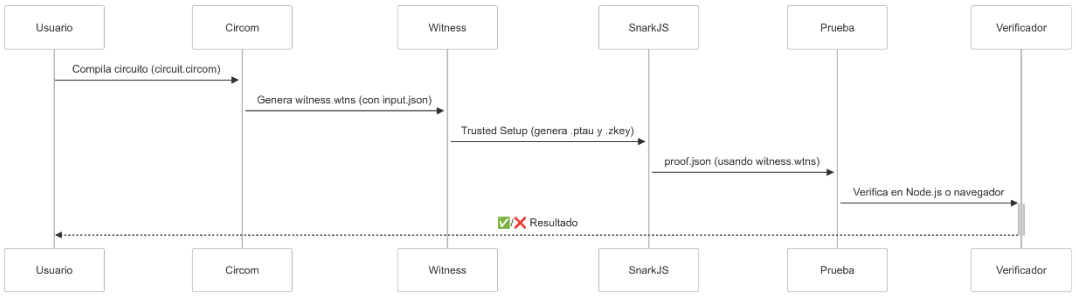
3.2. Contenido de los archivos

Archivo	Ejemplo de Contenido
public.json	["89", "97"] (c, p)
proof.json	{ "pi_a": [...], "pi_b": [...], "pi_c": [...] }

4. Validaciones y casos límites

Caso	Comportamiento del Circuito
p no es primo	El circuito funciona, pero el resultado no es seguro.
a o b son negativos	El módulo se calcula correctamente (ej: $(-3)^2 = 9$).
p = 0	Error en tiempo de ejecución (división por cero).

5. Diagrama de secuencia completo



6. Ejemplo de Uso

Entradas			Cálculo	Salidas	
a	**b**	**p**	**c**	**c**	**p**
3	4	5	(9 + 16) % 5 = 0	0	5
5	8	97	(25 + 64) % 97 = 89	89	97
10	20	7	(100 + 400) % 7 = 3	3	7
0	0	11	(0 + 0) % 11 = 0	0	11

Se incluye **p** como salida ya que estará en el public.json al ser una señal declarada como pública.