UNIVERSIDAD NACIONAL DE ASUNCIÓN
FACULTAD POLITÉCNICA

INFORME PRIMER PARCIAL - BLOCKCHAIN

ALUMNO: HADI FAOUR

Abril 2025

CAMPUS UNIVERSITARIO DE LA UNA

SAN LORENZO - PARAGUAY

1. **La estructura del circuito.**

   El circuito implementado en Circom verifica la operación $c = (a^2 + b^2) \% p$, donde a y b son entradas privadas, p es un número primo público, y c es la salida pública. La estructura del circuito se compone de los siguientes componentes:

   Componentes Principales:

   - Potencia: Calcula el cuadrado de una entrada (out = in * in).
   - Suma: Realiza la suma de dos entradas (out = a + b).
   - Módulo: Calcula el módulo de una entrada respecto a un primo p (out = in % p).
   - IsZero: Verifica si una entrada es cero, utilizado para restricciones.

   Circuito Principal:

   - Entradas: a y b (privadas).
   - Salida: c (pública).

   Flujo de Operaciones:

   - Calcula a^2 y b^2 usando el componente Potencia.
   - Suma los resultados con el componente Suma.
   - Aplica el módulo p al resultado usando el componente Módulo.
   - Asegura que a y b sean menores que p mediante restricciones con IsZero.
   - Prime usado: 21888242871839275222246405745257275088548364400416034343698204186575808495617.

2. **El proceso de generación de pruebas.**

   La generación de pruebas se realiza mediante snarkjs y sigue los siguientes pasos:

   a. Compilación del Circuito:
      - El archivo .circom se compila a .r1cs (Restricted Rank-1 Constraint System) y .wasm (WebAssembly) para ejecución eficiente.
   b. Ceremonia de Confianza (Trusted Setup):
      - Se genera un archivo .ptau (Powers of Tau) para la configuración inicial.
      - Se contribuye a la ceremonia para descentralizar la confianza.
   c. Generación de Claves:
      - Se crea una clave de prueba (proving key) y una clave de verificación (verification key).
   d. Generación del Testigo (Witness):
      - Se calcula el testigo usando input.json y el archivo .wasm.
   e. Generación de la Prueba:
      - Se produce un archivo proof.json que contiene la prueba ZK-SNARK.

3. **El proceso de verificación.**
    a. Verificación con snarkjs:
        - Usa la clave de verificación (verification_key.json), las señales públicas (public.json) y la prueba (proof.json).
    b. Verificación en Navegador:
        - Se genera un archivo HTML (verifier.html) que carga snarkjs y permite verificar la prueba interactivamente.
    c. Verificador Solidity:
        - Se genera un contrato inteligente (verifier.sol) para integración con Ethereum.

4. **Ejemplos de uso con valores concretos.**
    a. Ejemplo 1:
        - Entradas: a = 12, b = 3 (definidas en input.json).
        - Operación: $(12^2 + 3^2) \% p = (144 + 9) \% p = 153 \% p$.
        - Salida: c = 153 (asumiendo que p > 153).
    b. Ejemplo 2:
        - Entradas: a = 5, b = 7.
        - Operación: $(25 + 49) \% p = 74 \% p$.
        - Salida: c = 74 (si p > 74).

Si a o b son mayores o iguales que p, la prueba falla debido a las restricciones del circuito.