Documentación Técnica del Circuito zk-SNARK

Maria Jose Duarte Kowalewski (5224801) - Kevin Mathias Galeano Saldivar (5985474)

1. Estructura del Circuito

1.1 Descripción General

El circuito implementa una operación aritmética que combina dos entradas (`a` y `b`), calcula sus cuadrados, suma los resultados y aplica una operación de módulo. Este tipo de circuito es común en aplicaciones de criptografía y pruebas de conocimiento cero (zk-SNARK), donde se requiere demostrar el conocimiento de ciertos valores sin revelarlos.

1.2 Componentes del Circuito

El circuito está compuesto por los siguientes módulos:

1. **Square:** Calcula el cuadrado de una entrada.
    - **Entrada:** `in` (valor numérico).
    - **Salida:** `out` (cuadrado del valor de entrada).
2. **Mod:** Aplica una operación de módulo a una entrada.
    - **Entrada:** `in` (valor numérico).
    - **Salida:** `out` (resultado de `in % p`).
3. **Suma:** Suma los resultados de los cuadrados de `a` y `b`.
    - **Entrada:** Dos valores numéricos.
    - **Salida:** Un valor numérico.

1.3 Flujo de Datos

1. Las entradas `a` y `b` se elevan al cuadrado utilizando el módulo Square.
2. Los resultados de los cuadrados se suman.
3. La suma se pasa al módulo Mod, que calcula el módulo.
4. El resultado final se asigna a la salida c.

1.4 Consideraciones de Diseño

- **Restricciones Aritméticas:** El circuito está diseñado para operar sobre campos finitos, lo que es esencial para la compatibilidad con zk-SNARK.
- **Eficiencia:** El uso de operaciones modulares y cuadrados está optimizado para minimizar el número de restricciones y reducir el costo computacional.

2. Proceso de Generación de Pruebas

2.1 Objetivo de las Pruebas

El objetivo de las pruebas es demostrar que el circuito funciona correctamente para un conjunto de entradas y que la salida es válida según las restricciones definidas. Esto se logra mediante la generación de pruebas zk-SNARK, que permiten verificar la corrección del cálculo sin revelar las entradas originales.

2.2 Herramientas Utilizadas

- **Circom:** Compilador de circuitos aritméticos que transforma el código en una representación intermedia (R1CS).
- **SnarkJS:** Biblioteca que implementa el protocolo Groth16 para generar y verificar pruebas zk-SNARK.
- **Groth16:** Protocolo de pruebas de conocimiento cero que utiliza curvas elípticas y pruebas sucintas para garantizar la privacidad y eficiencia.

2.3 Metodología de Generación de Pruebas

1. Compilación del Circuito:

El circuito se compila en un formato intermedio llamado R1CS (Rank-1 Constraint System), que representa las restricciones aritméticas del circuito. También se generan archivos adicionales, como el binario WASM, que permite la ejecución del circuito en un entorno web.

2. Generación del Testigo (Witness):

El testigo es una asignación de valores que satisface todas las restricciones del circuito. Utilizando el binario WASM, se genera el testigo (`witness.wtns`), que contiene los valores intermedios y la salida del circuito.

3. Trusted Setup:

El protocolo Groth16 requiere una fase de configuración inicial llamada *Trusted Setup*.

- En esta fase, se generan claves de prueba y verificación utilizando parámetros criptográficos seguros.
- El proceso incluye la generación de un archivo `.ptau` (Powers of Tau) y la creación de claves específicas para el circuito.

4. Generación de la Prueba:

Utilizando las claves generadas en el Trusted Setup y el testigo, se genera una prueba zk-SNARK, que es un conjunto de valores criptográficos que demuestran la validez del cálculo sin revelar las entradas originales.

2.4 Ejemplo de Generación de Pruebas

Para las entradas `a = 2` y `b = 3`:

1. Se calcula $a^2 = 4$ y $b^2 = 9$.
2. Se suma $4 + 9 = 13$.
3. Se aplica el módulo 7: $13 \% 7 = 6$.
4. La prueba zk-SNARK demuestra que este cálculo es correcto sin revelar los valores de a y b.

3. Proceso de Verificación

3.1 Objetivo de la Verificación

La verificación tiene como objetivo asegurar que la prueba generada es válida y que el circuito cumple con las restricciones definidas. Esto garantiza la integridad y corrección del cálculo.

3.2 Herramientas Utilizadas

- **SnarkJS:** Para verificar la prueba zk-SNARK.
- **Groth16:** Protocolo utilizado para la verificación, que garantiza la eficiencia y seguridad de la prueba.

3.3 Metodología de Verificación

1. **Carga de la Clave de Verificación:**
   La clave de verificación (`verification_key.json`) se carga para validar la prueba.
2. **Carga de las Señales Públicas:**
   Las señales públicas (`public.json`) contienen los valores que se hacen visibles durante la verificación (en este caso, la salida `c`).
3. **Carga de la Prueba:**
   La prueba (`prueba.json`) se carga para su validación.
4. **Verificación:**
   Utilizando Groth16, se verifica que la prueba es válida y que las señales públicas coinciden con las restricciones del circuito.

3.4 Ejemplo de Verificación

Para el ejemplo anterior (c = 6):

1. La clave de verificación confirma que la prueba es válida.
2. Las señales públicas demuestran que la salida c es correcta.
3. La verificación devuelve true, indicando que la prueba es válida.

4. Ejemplos de Uso con Valores Concretos

4.1 Ejemplo 1

Entradas:

- a = 2
- b = 3

Cálculo:

- $a^2 = 4$
- $b^2 = 9$
- sum = 4 + 9 = 13
- c = 13 % 7 = 6

Resultado:

La prueba zk-SNARK demuestra que c = 6 es correcto.

4.2 Ejemplo 2

Entradas:

- a = 5
- b = 4

Cálculo:

- $a^2 = 25$
- $b^2 = 16$
- sum = 25 + 16 = 41

- c = 41 % 7 = 6

Resultado:

La prueba zk-SNARK demuestra que c = 6 es correcto.

5. Conclusiones

El circuito implementado es un ejemplo práctico de cómo se pueden utilizar zk-SNARKs para demostrar la validez de cálculos sin revelar las entradas originales. El uso de Groth16 garantiza la eficiencia y seguridad de las pruebas, mientras que herramientas como Circom y SnarkJS facilitan la implementación y verificación del circuito. Este enfoque es especialmente útil en aplicaciones de privacidad y criptografía, donde la confidencialidad de los datos es crítica.