Introduction to Applied ZK

Code: https://github.com/hyunsooda/zk-practice

02/08/2024

Why Zero-Knowledge Proof (ZKP)

- Invented from Cryptography area
- A mathematical tool that proves a given statement is true, while not revealing knowledge of the statement
- Identity
  - Prover
  - Verifier
- Use cases
  - Ideally any, but mostly hard to apply it

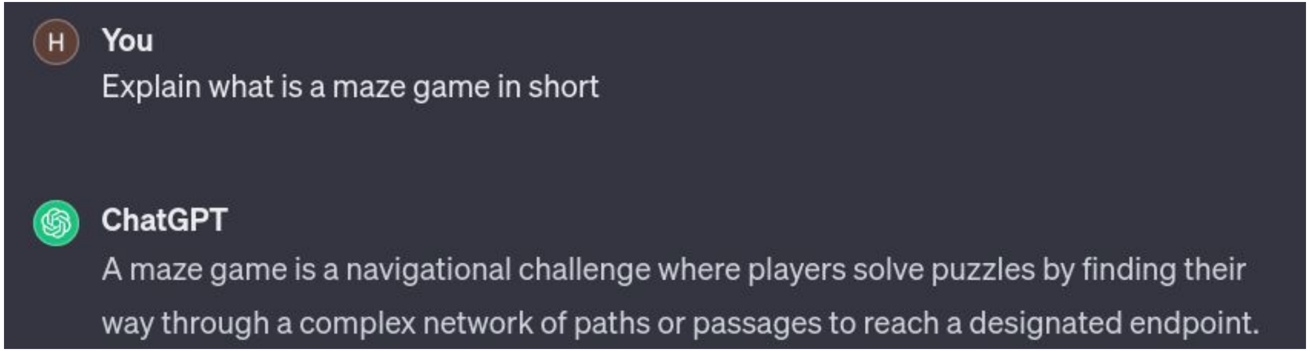
Circom (Circuit compiler)

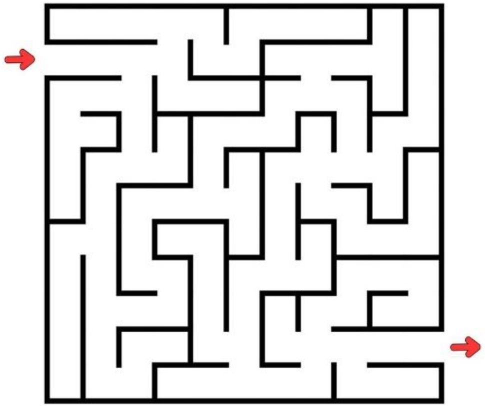
- A strong tool that generates a verifiable circuit (proof)
- Better intuitive than Zokrates
- Getting started: https://docs.circom.io/

Agenda

- Demonstration 1: Maze game ZK
- Demonstration 2: Computer vision task with ZK

Demonstration 1: Maze Game





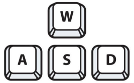
- Teacher raise a quiz to students
  - “Anybody knows an correct path of this maze?”
- A young student found the answer, but he says
  - I know it, but I’m not going to share it
- Teacher says,
  - How can I give you a score without the answer?

Demonstration 1: Maze Game (Definition)

- This simple storyline can be perfectly tailored with ZKP
- Let's transform the problem with ZKP style
  - ~~Legacy problem: Submit your maze path~~
  - Refined problem: Submit your witness on the circuit
  - ~~Legacy solver: Take the path step by step~~
  - Refined solver: Verify the circuit with the known maze map
  - Contribution: Can verify without revealing the knowledge (maze path)

Demonstration 1: Maze Game (Approach)

- **Input1**: maze map (can be public)
- **Input2**: maze goal (can be public)
- **Input3**: maze path (private)
- **Output**: Boolean (*True* if its path is valid, *False* otherwise)
- **Method**: Implement step-by-step path mover (Very simple)
  - e.g., a given path: “ssswd”

    - Move bottom three times, move up, and move right
    - Now, the current position is the goal?
      - Return 1, if yes
      - Return 0, otherwise

Demonstration 1: Maze Game (Code) (1/4)

```
// 1: goal
const maze = [
  0, 0, 0, 0, 1,
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0
];
```

- How to implement “step-by-step path mover”?
- It’s very simple in high-level language

```
if (input[i] == ‘a’) {
  pos += -1;
}
if (input[i] == ‘d’) {
  pos += 1;
}
if (input[i] == ‘s’) {
  pos += col;
}
if (input[i] == ‘w’) {
  pos += -col;
}
```

Circuit Rule1: control-flow is not available if it is associated with ***“signal”***

Circuit Rule2:
string is not the primitive type. Only the integer is provided. We have to encode the string with the set of integers

Demonstration 1: Maze Game (Code) (2/4)

Phase1. Type encoding
(straightforward)

```
[Type encoding]
F("a") => F(0)
F("d") => F(1)
F("s") => F(2)
F("w") => F(3)
```

Phase2. Define the function ***f***

```
[Def. function]
F(0) => -1
F(1) => 1
F(2) => col
F(3) => -col
```

Demonstration 1: Maze Game (Code) (3/4)

```
[Type encoding]
F(0) => -1
F(1) => 1
F(2) => col
F(3) => -col
```

Define a function *f* such that satisfies the desired output for the corresponding inputs

1st try

```
F(x) = 2 * x -1
F(0) = -1  (O)
F(1) = 1   (O)
F(2) = 3   (X)
F(3) = 5   (X)
```

nth try

```
g(0) = g(1) = 2 * x - 1
g(2) = col * H2 - g({0,1})   //cancel-out g({0,1})
g(3) = col * -2 * H3         //cancel-out g(2)
g(x) = g({0,1}) + g(2) + g(3)
H2 = 1 if x > 1, 0 otherwise
H3 = 1 if x > 2, 0 otherwise
F(x) = g(x)
```

Demonstration 1: Maze Game (Code) (4/4)

```
We should have paid the implementation cost
to transform the control-flow to circuit
```

```c
if (input[i] == 'a') {
  pos += -1;
}
if (input[i] == 'd') {
  pos += 1;
}
if (input[i] == 's') {
  pos += col;
}
if (input[i] == 'w') {
  pos += -col;
}
```

C code

```
g(0) = g(1) = 2 * x - 1
g(2) = col * H2 - g({0,1})  //cancel-out g({0,1})
g(3) = col * -2 * H3        //cancel-out g(2)
g(x) = g({0,1}) + g(2) + g(3)
H2 = 1 if x > 1, 0 otherwise
H3 = 1 if x > 2, 0 otherwise
F(x) = g(x)
```

Circuit

Demonstration 1: Maze Game (Retrospection)

- Now the student can prove that he knows the correct path without revealing the maze path
- Teacher can verify that he really knows the answer without knowing the exact answer
- Lessons learned:
  - This kind of problems can be extended to the ZKP
    - encoding/circuit transform are another challenge
- Code

Demonstration 2: Computer vision task (AI) (Intro 1/2)

- In recent years, AI has undoubtedly dominated
  - Generative AI
  - Large Language Model (LLM)
- Computer vision
  - AI-aided vision tasks have been widely used in real-world
    - Automated driving, Embedded devices (Galaxy S24), etc
  - Takes an input image, processing with trained model, outputs a valuable information
    - Classification
    - Segmentation
    - Recognition

Demonstration 2: Computer vision task (AI) (Intro 2/2)

- How to train model?

```
1. Feed the test images and labels(answer) to the model
2. Do forward propagation (calc. difference) & backward propagation (feedback)
3. Repeat 1 and 2 until a given threshold is satisfied
```

- Model serving
    - Provide a service based on the trained model (e.g., image classification)
- **What is private?**
    - **User's input** (e.g., image, health information, etc)
    - **Model's internal state** (weight and gradient)

Demonstration 2: Computer vision task (AI) (Definition)

- In academia, numerous studies are delving into the realm of securing AI, which are focused on
  - **Model's output integrity**: *Trusted Execution Environment (TEE)*
  - **Hiding model's input**: *Homomorphic Encryption (HE)*
  - **Hiding model's input:** *ZKP (most recently raised)*
- In this demonstration, we will not only explore how to verify the model's execution integrity, but also not reveal the model's input(image) through *ZKP*

Demonstration 2: Computer vision task (AI) (Approach)

- **Input1**: Fully-Connected(FC) Layers (matrix values)
- **Input2**: Input image (28x28)
- **Input3**: Salt (to be used for hashing)
- **Input4**: Expected model's hash
- **Input5**: Expected model's output
- **Output**: Model's output
- **Method**: Implement forward propagation in circuit
  - This is a naive implementation. Recent papers' approaches will be briefly introduced

Model Architecture
FC1 (784, 200)
ReLU1 (200, 200)
FC2 (200, 10)

Model arch. used for this demo.

Demonstration 2: Computer vision task (AI) (Code) (1/6)

```
// 1. calculate fc1 layer
component mmv1 = MatMulVec(weightRow, weightCol);
for (var i=0; i<weightRow; i++) {
    for (var j=0; j<weightCol; j++) {
        fc1In[i][j] ==> mmv1.A[i][j];
    }
}
for (var i=0; i<weightCol; i++) {
    image[i] ==> mmv1.x[i];
}
signal fc1Out[weightRow] <== mmv1.out;
```

Model Architecture
FC1 (784, 200)
ReLU1 (200, 200)
FC2 (200, 10)

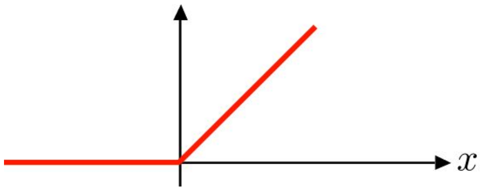
1. MatVecMul
(FC1 weight * image) = FC2

Demonstration 2: Computer vision task (AI) (Code) (2/6)

```
// 2. calculate relu layer
component relu[weightRow];
for (var i=0; i<weightRow; i++) {
    relu[i] = ReLU();
    fc1Out[i] ==> relu[i].in;
}
```

2. ReLU(FC1) = ReLU1
(FC1 weight * image) = FC2

$$\mathrm{ReLU}(x) \triangleq \max(0, x)$$



Def. ReLU(x)

Model Architecture
FC1 (784, 200)
ReLU1 (200, 200)
FC2 (200, 10)

Demonstration 2: Computer vision task (AI) (Code) (3/6)

```
// 3. calculate fc2 layer
component mmv2 = MatMulVec(outputSize, weightRow);
for (var i=0; i<outputSize; i++) {
    for (var j=0; j<weightRow; j++) {
        fc2In[i][j] + NEGATIVE_COMPLEMENT ==> mmv2.A[i][j];
    }
}
for (var i=0; i<weightRow; i++) {
    relu[i].out ==> mmv2.x[i];
}
```

```
component argMax = ArgMax(outputSize);
mmv2.out ==> argMax.in;
signal output out <== argMax.out;
```

Model Architecture
FC1 (784, 200)
ReLU1 (200, 200)
FC2 (200, 10)

3. MatVecMul
(ReLU1 * image) = FC2

4. ArgMax
ArgMax(FC2) = 0 | 1 | …

Demonstration 2: Computer vision task (AI) (Code) (4/6)

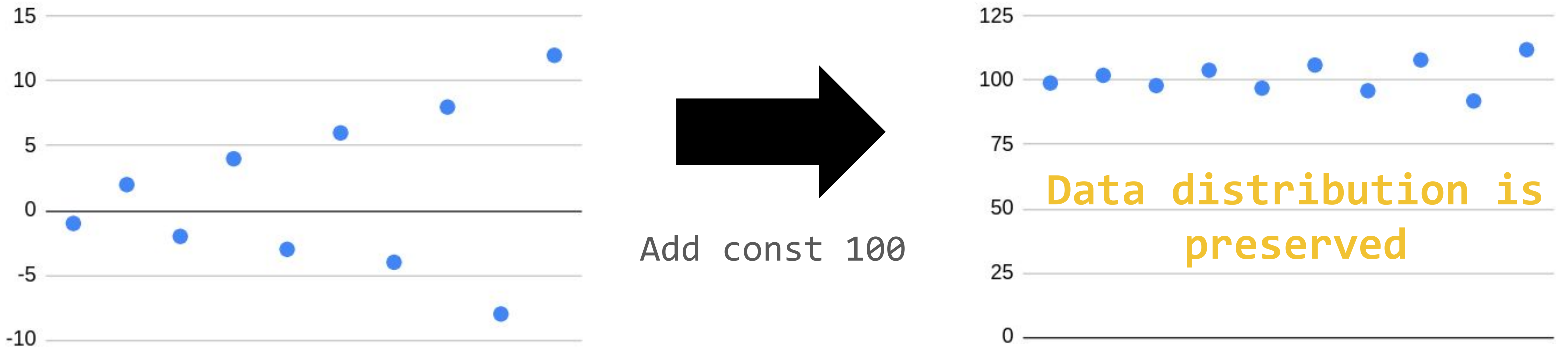
5. Model Checksum

(Actually, this is the first step, but prepared at the end for natural flow)

```
// 0. model integrity check
component getModelHash = GetModelHash(weightRow, weightCol, outputSize);
fc1In            ==> getModelHash.fc1In;
fc2In            ==> getModelHash.fc2In;
modelHashSalt    ==> getModelHash.modelHashSalt;
expectedModelHash === getModelHash.out;
```

Demonstration 2: Computer vision task (AI) (Code) (5/6)

- Consideration1. Negative values
  - Circom supports only one primitive type, ***integer***
  - **Solution**. Add big constant values so that all matrix values should be larger than zero
    - e.g., Given `arr[0] = -13`, we add a big constant
      - `arr[0] = arr[0] + 100000000 = 99999987`



Demonstration 2: Computer vision task (AI) (Code) (6/6)

- Consideration2. Floating values
  - Circom supports only one primitive type, ***integer***
  - **Solution**. Ceiling
    - e.g., `Given arr[0] = 18.251`, ceiling the real number
      - `arr[0] = ceil(arr[0])`
      - In this demo, I exported ceiled-weight from trained model and fed to circuit singal
  - ref: [See Model Quantilization]()
- Consideration3. Model Checksum
  - Hashing all model's weight is too much costly; not compiled also
  - **Solution**. Element sum
  - I iterated all matrix element and summed-up and hashes it instead
    - If you find this seems vulnerable. Please share your thought or solution 😊

Demonstration 2: Computer vision task (AI) (Retrospection)

- A good security textbook and paper said,
  - Solution finding should be hard (i.e., ~= NP-Hard)
  - Verification should be easy (e.g., ~= O(1))
- The verification implementation of this demonstration is not efficient
  - We calculated all the matrix multiplication O(N^3)
  - There's a good alternative (Freivalds' algorithm) which we may consider (O(n^2.373))

Input [edit]

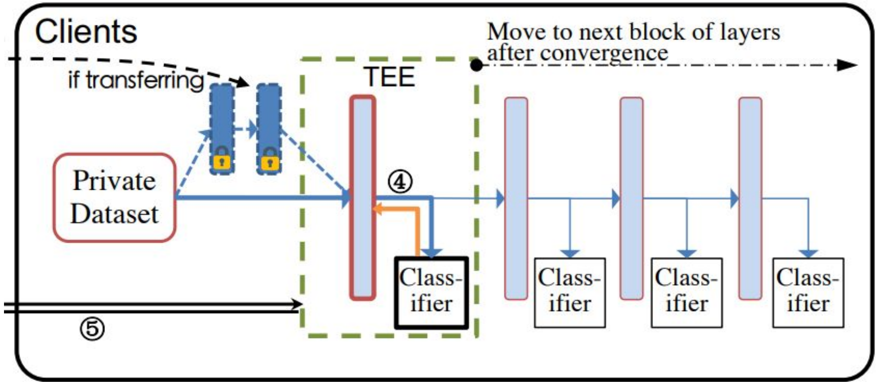
Three *n* × *n* matrices A, B, and C.

Output [edit]

Yes, if $A \times B = C$; No, otherwise.

Demonstration 2: Computer vision task (AI) (Retrospection)

- If your verification step is too big to transform circuit, divide the verification phase into two steps, ***front-end layer*** and ***backend layer***
- This idea was inspired from another research area
  - PPFL'MobiSys21 partially executed the inference steps in TEE, another partial steps in host environment



Demonstration 2: Computer vision task (AI) (Retrospection)

- If you're interested in more detail on this area, refer two papers
  - "Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning", Security'21
  - "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy", CCS'21

Conclusion

- We explored toy example of applied ZK
    - How to transform(encoding/decoding) to circuit
- Pros
    - Verifiability without revealing knowledge is appealing
- Cons
    - Expressiveness is too limited to utilize it practically