Zero Knowledge Proofs

ZKP Applications

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang
ZKP for Machine Learning

Credit Risk Prediction
Criminal Justice
Healthcare

ML inference

fair or not?
Proving ML Inferences using ZKP

Zero-knowledge proof without revealing the ML models
✓ Fairness of ML models
✓ Integrity of ML inferences
Challenges

Efficiency and Scalability of general-purpose SNARKs:
scale to $<2^{30} = 1$ billion gates (64 GB RAM), prover time minutes to hours

VGG 16 on CIFAR-10

- 15 million parameters in the model
- 1.1 billion gates for an inference
Solution: Special-Purpose ZKPs

- Fully connected or convolutional
  - Activation
  - Pooling
  - ...
  - Fully connected or convolutional
    - Softmax
ZKP for Matrix Multiplication [Thaler’13]

Matrix multiplication $C = A \times B$:

$c_{ij} = \sum_k a_{ik} b_{kj}$

$C(x, y) = \sum_z A(x, z) B(z, y)$

$C(i, j) = c_{ij}$  \quad $A(i, k) = a_{ik}$  \quad $B(k, j) = b_{kj}$

- Efficient ZKP with prover time $O(n^2)$, proof size $O(\log n)$
- Faster than computing the result in $O(n^3)$
- Verifying is easier than computing
ZKP for 2-D Convolutions [LXZ’21]

2-D convolution $C = A \ast B$

$O(NK)$ time to compute
Computing Convolution using FFT

- Equivalent to 1-D convolution
  \[ c = a \ast b = \sum_i a_i b_{N-i} \]

- Same as polynomial multiplication
  \[ c(x) = a(x) \cdot b(x) \]

- Can be computed by Fast Fourier Transform (FFT)
ZKP for Fast Fourier Transform

\[ \bar{a} = F \times a \]
\[ \bar{a}(x) = \sum_y F(x, y) \cdot a(y) \]

\[ F = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^1 & \omega^2 \\ 1 & \omega^3 & \omega^2 & \omega^1 \end{pmatrix} \]

\[ \times \text{ Size of } F(x, y) \text{ is } N^2 \]
\[ \checkmark \text{ } F \text{ consists of only } N \text{ distinct values} \]

- An efficient sumcheck protocol with prover time \( O(N) \), proof size \( O(\log N) \), verifier time \( O(\log^2 N) \)
- Sublinear in the computation time \( O(N\log N) \)
# Performance of zkCNN

<table>
<thead>
<tr>
<th></th>
<th>1 inference</th>
<th>Accuracy on 100 images</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prover time</td>
<td>88 seconds</td>
<td>680 seconds</td>
</tr>
<tr>
<td>Proof size</td>
<td>341 KB</td>
<td>673 KB</td>
</tr>
<tr>
<td>Verifier time</td>
<td>59 ms</td>
<td>121 ms</td>
</tr>
</tbody>
</table>

VGG16 on CIFAR10 dataset, **15 million** parameters (**120MB**)
Other Related Works on ZKML

ZKDT[ZFZD20], vCNN [LKKO20], ZEN [FQZ+21], Mystique [WYX+21], pvCNN [WWT+22], [KHSS22], ...
ZKP for Program Analysis
Zero-knowledge Program Analysis

public function: static analysis algorithm

secret program $P$

safety properties of $P$
Zero-knowledge Vulnerability Disclosure

public program

```
#include <stdio.h>
#include <string.h>

void main()
{
    char str1[10];
    char str2[10];

    strcpy(str1, "Meeting");
    printf("length is %zu", strlen(str1));
    strcpy(str2, str1);
    printf("length is %zu", strlen(str2));

    if (strcmp(str1, str2) == 0)
        printf("Both strings are the same\n");

    str1[1] = '-';
    strlen(str2, str1, 3);
    printf("The string is now: %s\n");
    if (strcmp(str1, str2) == 0)
        printf("The strings are still equal\n");
}
```

secret vulnerability

Running the program leads to crash
Challenges

- ZKP schemes support circuits.
- Program analysis is usually RAM computation
Solution: Auxiliary Inputs

Ask the prover to provide additional data as the input of ZKP

- Not trusted
- Not sent to the verifier
- Significantly improves the efficiency of ZKP
Example: worklist algorithm

Program:

```python
1  x1 = source()
2  if (x2 > 5):
3      x3 = x1
4  else
5      x3 = 9
6  sink(x3)
```

Worklist:

<table>
<thead>
<tr>
<th>Line No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x1, x2, x3)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
</tr>
</tbody>
</table>

State:

CFG:
Worklist algorithms: update

Program:

```
1 x1 = source()
2 if (x2 > 5):
3     x3 = x1
4 else
5     x3 = 9
6 sink(x3)
```

Worklist:

- (1, 2)
- (2, 3)
- (2, 4)

State:

<table>
<thead>
<tr>
<th>Line No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x₁, x₂, x₃)</td>
<td>(0, 0, 0)</td>
<td>(1, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
</tr>
</tbody>
</table>

CFG:
Worklist algorithms: update

Program:

```
x1 = source()
if (x2 > 5):
x3 = x1
else
x3 = 9
sink(x3)
```

Worklist:

<table>
<thead>
<tr>
<th>Line No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>State:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(x1, x2, x3)</td>
<td>(0, 0, 0)</td>
<td>(1, 0, 0)</td>
<td>(1, 0, 0)</td>
<td>(1, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(1, 0, 1)</td>
</tr>
</tbody>
</table>

CFG:
Auxiliary inputs

- Prover provides final state of the list
- Prover provides head and tail of each step
- The circuit checks the correctness (offline memory checking [BEGKN’91, Setty’20, ...])
Performance

Program with $T$ steps and $\nu$ variables

Worklist algorithm: $O(T \cdot \nu)$

$\rightarrow$ circuit of size $O(T \cdot \nu + T\log T)$
Related works

- Static analysis: [FDNZ’21, LAHPTW’22, …]
- Vulnerabilities: [GHHKPV’22, CHPPT’23, …]
ZKP for Middlebox
Middleboxes inspect traffic to ensure security policy
Encrypted Traffic
Zero-Knowledge Middleboxes [GAZBW’22]
Challenges

- Work with TLS 1.3
- Legacy cryptographic functions such as AES, SHA
End of Lecture