Ligetron: WASM as an Intermediate Representation and easy tooling for zkSNARKs

Ligero Inc.
Muthu Venkitasubramaniam

Presented on September 27 @ NIST MPTS 2023

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).
The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
Distribution Statement A. Approved for public release. Distribution Unlimited.

Ideal Toolchain to Instrument ZK

Step 1) Write the statement in C/C++

Step 2) Compile it to ZK

Obstacles towards the goal

First, it was, prover time

Today - prover times have come to under 100 ns / gate

Now, representation

Why is this hard? Representation can only involve

1. **Low-level operations - ADD/MUL gates**
2. **Oblivious control flow**

Current approaches

ZK-SNARKs - short proofs that are publicly verifiable

Flatten representation (i.e. circuit) but results in large memory overhead

Use Interaction (i.e. `lose public verifiability)

VOLE-based ZK - Highly efficient (LPZK/EMP/Mac-n-cheese)

- Not publicly verifiable
- “Not” succinct
- “Not” Post-quantum security

Ligetron Performance On a Desktop Browser

- Starting from an **oblivious code written in C**, we achieve
  - Prover: **3.5 us/g**
  - Verifier: **1.5 us/g**
- Starting from any **code written in C with secret memory access** (i.e. RAM) but oblivious otherwise, we achieve
  - Prover: **~10 us/g**
  - Verifier: **~3 us/g**

Can run from a browser on a smartphone!

Our Approach

Key Insights

- WASM is a stack based machine with semantics that has low-level operations yet high-level memory management
- Ligero is an MPC-in-the-head based ZK that can nicely trade space and succinctness

Extent of WASM integration

Where we are:

- uint32, uint64
- Oblivious code
- Secret memory (RAM)

In progress

- Secret branching (more next slide)
- Floating point numbers

```cpp
extern "C" {

  inline int min(int a, int b) {return a <= b ? a : b;}

  inline int oblivious_if(bool cond, int t, int f) {
      int mask = static_cast<int>((1ULL << 33) - cond);
      return (mask & t) | (~mask & f);
  }

  int minDistance(const char* word1, const char* word2, const int m, const int n) {
      int pre;
      int cur[n + 1];
      for (int j = 0; j <= n; j++) {
        cur[j] = j;
      }
      for (int i = 1; i <= m; i++) {
        pre = cur[0];
        cur[0] = i;
        for (int j = 1; j <= n; j++) {
          int temp = cur[j];
          bool cond = word1[i - 1] == word2[j - 1];
          cur[j] = oblivious_if(cond,
            pre,
            min(pre, min(cur[j - 1], cur[j])) + 1);
          pre = temp;
        }
      }
      return cur[n];
  }

  bool statement(const char *word1, const char* word2, const int m, const int n) {
      return minDistance(word1, word2, m, n) < 5;
  }
```

C code for Edit distance

LLVM compiler

```
(module
  (type (;0;) (func (param i32 i32 i32 i32) (result i32)))
  (import "env" "__linear_memory" (memory (;0;) 0))
  (import "env" "__stack_pointer" (global (;0;) (mut i32)))
  (func $minDistance (type 0) (param i32 i32 i32 i32) (result i32)
    (local i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32 i32)
    global.get 0
    local.tee 4
    drop
    i32.const 0
    local.set 5
    local.get 4
    local.get 3
    i32.const 2
    i32.shl
    i32.const 19
    i32.add
    i32.const -16
    i32.and
    i32.sub
    local.tee 6
    drop
    block  ;; label = @1
      local.get 3
      i32.const 0
      i32.lt_s
      br_if 0 (;@1;)
```

WASM code for Edit distance

Distribution Statement A. Approved for public release. Distribution Unlimited.

Demo

Visit ligetron.com

Distribution Statement A. Approved for public release. Distribution Unlimited.