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

Muthu Venkitasubramaniam
Ligero Inc. / Georgetown U.

ZKProof Policy @ DC

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.

Thanks
Stealth Software Tech.

What ZK application do you have for me today?

Muthu Venkitasubramaniam
Ligero Inc. / Georgetown U.

ZKProof Policy @ DC

An application

Alice:
wants to prove to Eve that she earns at least $100,000

Solution: Eve can request tax returns or W2s for the last year

Suppose that
1. Employer commits W2 or
2. IRS commits tax returns
on a public ledger (aka blockchain)

With ZK: Alice proves she has tax return or W2 that:

- Corresponds to the commitment on the blockchain
- Alice's name
- Indicates income > $100,000
- REVEAL NOTHING MORE!

An application

Muthuramakrishnan:
wants to prove to Eve that she earns at least $100,000

Solution: Eve can request tax returns or W2s for the last year

Suppose that
1. Employer commits W2 or
2. IRS commits tax returns
on a public ledger (aka blockchain)

With ZK: Alice proves she has tax return or W2 that:

- Corresponds to the commitment on the blockchain
- Alice's name
- Indicates income > $100,000
- REVEAL NOTHING MORE!

An application

Muthuramakrishnan:
wants to prove to Eve that she earns at least $100,000

Solution: Eve can request tax returns or W2s for the last year

Suppose that
1. Employer commits W2 or
2. IRS commits tax returns
on a public ledger (aka blockchain)

With ZK: Alice proves she has tax return or W2 that:

- Corresponds to the commitment on the blockchain
- **Muthuramakrish's** name
- Indicates income > $100,000
- REVEAL NOTHING MORE!

Ideal Toolchain to Instrument ZK

Step 1) Write the statement in typical high-level language

Step 2) Compile it to ZK

Obstacles towards the goal

Representation!

Why is this hard? Backends need

- Low-level operations - ADD/MUL gates
- Oblivious control flow – unroll loops

Current approaches

Approach 1

ZK-SNARKs need a flattened representation (i.e. circuit) and results in large memory overhead

Approach 2

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

Interactive => Not blockchain friendly

“Not” succinct

“Not” Post-quantum security

Takeaway

1 hour back: “Prover time is the bottleneck” – Justin Thaler

Now – Prover space is the bottleneck

Introducing Ligetron

A Time and Space Efficient ZK-SNARK

Two Ingredients

1. WASM as an intermediate representation
2. A space-efficient variant of the Ligero ZK Proof System [2017]

Our Toolchain

- Step 1: Code application in C/C++/Rust/…
- Step 2: Compile to WASM using existing compiler (eg, emscripten)
- Step 3: Prover and Verifier take WASM code as input

Ligetron Performance

On a Browser!

On a desktop, it is ~100x faster!

- Prover: ~10 us/g
- Verifier: ~3 us/g

```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

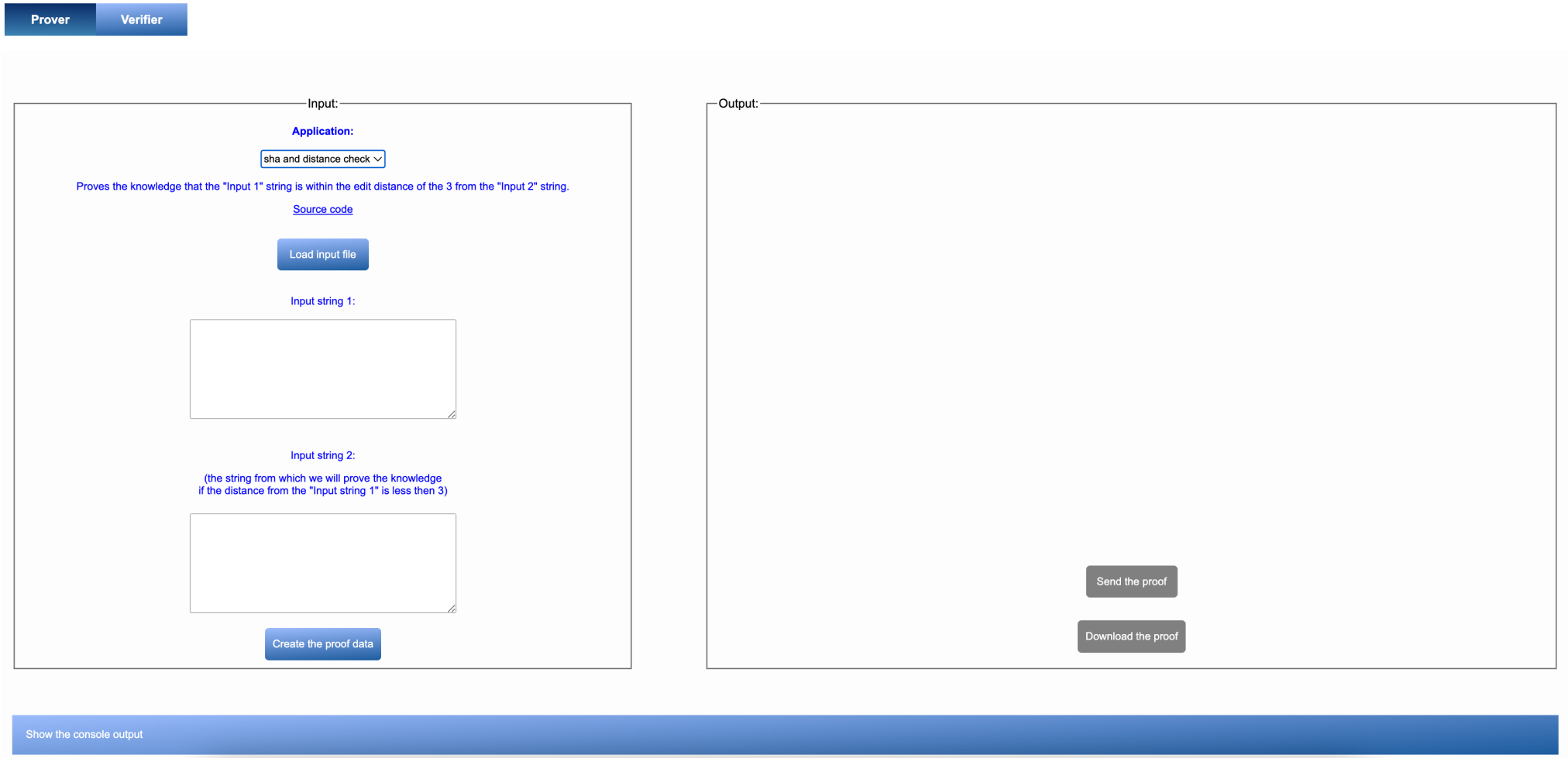
LLVM compiler

```
(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

https://ligetron.com/

ZK Demo



Prover | Verifier

Input:

Application:

sha and distance check

Proves the knowledge that the "Input 1" string is within the edit distance of the 3 from the "Input 2" string.

Source code

Load input file

Input string 1:

Muthuramakrishnan

Input string 2:

(the string from which we will prove the knowledge
if the distance from the "Input string 1" is less then 3)

Muthuramakrishhan

Create the proof data

Output:

Input 1 hash: d39db8914dc6fd02cb35bc9351d6927bc3c2ce1ff27771fcfb75ccbad0c6b091

Result: true

Proof ID: 3avLlx

QR:

Execution time: 13974 ms

Circuit size: 765695

Send the proof

Download the proof

Show the console output

Text redaction

```
#include "sha256.hpp"
#include "convert.hpp"

WASM_MAIN {
    u8 out[32];
    u8 ref[32];

    int argc, buf_size;
    args_sizes_get(&argc, &buf_size);

    char *argv[argc], buf[buf_size];
    args_get(argv, buf);

    i32 len = strlen((const char *)argv[1]);

    // convert the hash from hexadecimal string representation to binary form
    hexToBin((const char *)argv[3], reinterpret_cast<u8*>(ref), 64);

    lonesha256(out, reinterpret_cast<const u8*>(argv[1]), len);

    for (int i = 0; i < 32; i++) {
        assert_constant(argv[3][i]);
        assert_one(out[i] == ref[i]);
    }

    assert_one(strlen((const char *)argv[2]) == len);

    for (int i = 0; i < len; i++) {
        if (argv[2][i] != '#')
        {
            assert_constant(argv[2][i]);
            assert_one(argv[2][i] == argv[1][i]);
        }
    }
}
```

https://ligetron.com/

ZK Demo