Programming MPC: Towards any Function

Marcel Keller

CSIRO's Data61

6 April 2022

What's Missing

Have

Basic operations in the underlying domain:

- Addition
- Multiplication

Want

Any (mathematical) function:

- Comparison
- Bit operations such as shifting
- Exponentiation, square root, trigonometric functions etc.

Section 1

Bit Operations

What's Missing (cont.)

Problem

Black box doesn't allow directly accessing individual bits.
E.g., no relation between bits of Shamir shares and bits of the secret.

Generic solution: bit decomposition

Split secret value to secret bits *within* the black box.

Bit Decomposition

Requirements

- One-time pad
- Binary adder
- Secret random bits

Bit Decomposition Requirement: One-Time Pad

Security

For any x, $x + r \bmod m$ is uniformly random if r is uniformly random.

Application

Encrypt stream of bits by XOR with random stream of bits.

Bit Decomposition Requirement: Binary Adder

Want

Integer addition with only binary operations

Solution: textbook addition*

Steps can be implemented using a simple circuit.

Bit Decomposition Requirement: Secret Random Bits

Want

Secret random bit b inside black box.

Protocol

1. Party i inputs random bit b_i.
2. Parties compute $b = \bigoplus_i b_i$ inside black box.
   $x \oplus y = x + y - 2xy \in \mathbb{Z}$

Result

For every honest party, b is random and unknown to all other parties.

Bit Decomposition

Pre: $[x]$ for $x \in \mathbb{Z}_{2^k}$, $[r_0], \ldots, [r_{k-1}]$ such that $r_i \xleftarrow{\$} \{0, 1\}$

Post: $[x_0], \ldots, [x_{k-1}]$ such that $x = \sum_i x_i \cdot 2^i$ and $x_i \in \{0, 1\}$

---

1. Compute and open $[c] = [x] - \sum_i [r_i] \cdot 2^i$
2. Compute $c_0, \ldots, c_{k-1}$ such that $c = \sum_i c_i \cdot 2^i$ and $c_i \in \{0, 1\}$
3. Compute $[x_0], \ldots [x_{k-1}]$ using a binary adder on $([r_0], \ldots, [r_{k-1}])$ and $(c_0, \ldots, c_{k-1})$

Section 2

Comparison

Generic Computation Using Bit Decomposition

Have

Compute any function with binary circuits by accessing bits

Generic Computation Using Bit Decomposition

Have

Compute any function with binary circuits by accessing bits

Not very efficient

Comparison	Bit decomposition for both inputs then comparison circuit. Can we do better?
Exp/sqrt/...	Algorithms use addition and multiplication. How to minimize use of bit decomposition and recomposition?

Direct Comparison

- $x < y \Leftrightarrow (x - y) < 0$ (subtraction is free)
- $x < 0$ is equivalent to extract most significant bit in two's complement notation

Two's Complement

Approach

- Represent $[-2^{k-1}, 2^{k-1} - 1]$ as $[0, 2^k - 1] \mathrel{\hat{=}} \mathbb{Z}_{2^k}$ (k bits)
- Mapping negative numbers to $[2^{k-1}, 2^k - 1]$ by adding 2^k (most significant bit is 1)

Examples

$$-1 \mathrel{\hat{=}} 2^{k-1} - 1$$

$$-1 + 2 \mathrel{\hat{=}} 2^k - 1 + 2 = 2^k + 1 = 1 \mod 2^k$$

$$-1 - 1 \mathrel{\hat{=}} 2^k - 1 - 1 = 2^k - 2 \mathrel{\hat{=}} -2$$

Comparison with Zero

Want

Extract most significant bit

Approaches

- Plain bit decomposition: already better than 2 BD plus comparison circuits
- Even better: Only extract MSB using tree*

Complexity

Indirect $2k$ random bits $+ \sim 8k$ multiplications (or $\sim 12k$ mult. with fewer rounds)

Direct k random bits $+ \sim 2k$ multiplications (fewer rounds)

Section 3

Fractional Computation

Fractional Numbers

How Processors Do It (Simplified)

Represent $x \approx s \cdot 2^e$ as integers (s, e) such that s is strictly p bits (in $[2^{p-1}, 2^p - 1]$) for some p

It's Complicated

$$
\begin{aligned}
1 + 2^{-p-1} &= 2^{p-1} \cdot 2^{-p+1} + 2^{p-1} \cdot 2^{-2p} \\
&= (2^{p-1} + 2^{-2}) \cdot 2^{-p+1} \\
&\approx 2^{p-1} \cdot 2^{-p+1} \\
&= 1
\end{aligned}
$$

Fixed-Point Representation (Quantization)

Approach

Represent $x \in \mathbb{R}$ as $\lfloor x \cdot 2^f \rceil$ for fixed f.

Computation

- $x + y = \lfloor x \cdot 2^f \rceil + \lfloor y \cdot 2^f \rceil \approx \lfloor (x + y) \cdot 2^f \rceil$
- $x \cdot y = \lfloor x \cdot 2^f \rceil \cdot \lfloor y \cdot 2^f \rceil \cdot {\color{red}2^{-f}} \approx \lfloor (x \cdot y) \cdot 2^f \rceil$

Need

Division by 2^f

Truncation For Fixed-Point Multiplication

Canonical

1. Bit decomposition
2. Recompose higher bits: $\sum_{i=f}^{k} b_i \cdot 2^{i-f}$

Avoid bit decomposition?

Assume $r' \xleftarrow{\$} [0, 2^f - 1], r \xleftarrow{\$} [0, 2^{k-f} - 1]$. Then,

$$(x + r' + r \cdot 2^f)/2^f - r \approx x/2^f$$

if $(x + r' + r \cdot 2^f) < 2^k$ (because of wrap-around modulo 2^k). Would need comparison to make sure, but comparison needs bit composition! Or does it? We'll see later...

Prime Modulus

Computation modulo a prime p is a field

Every non-zero element has an inverse, i.e., x^{-1} such that $x \cdot x^{-1} = 1 \mod p$.

Example

$4 \cdot 10 = 40 = 1 + 39 = 1 \mod 13$

Idea

Use inverse of 2^f for truncation?

Probabilistic Truncation with Prime Modulus

Pre:
- $[x]$ for $x \in [0, 2^k] \subset \mathbb{Z}_p$, p prime such that $p > 2^{k+s}$ for some s
- $[r_0], \ldots, [r_{k+s-1}]$ such that $r_i \overset{\$}{\leftarrow} \{0, 1\}$

Post: $[y]$ such that $y \approx x/2^f$

---

1. Compute and open $[c] = [x] + \sum_{i=0}^{k+s-1} [r_i] \cdot 2^i$
2. Compute $[x'] = (c \bmod 2^f) - \sum_{i=0}^{f-1} [r_i] \cdot 2^i)$
3. Output $(([x] - [x']) \cdot (2^f)^{-1})$

$$x - x' = x - \left(\left(\left(x + \sum_{i=0}^{k+s-1} r_i \cdot 2^i \right) \bmod 2^f \right) - \sum_{i=0}^{f-1} r_i \cdot 2^i \right)$$

$$= x - \left(x \bmod 2^f + \sum_{i=0}^{f} r_i \cdot 2^i - \{0, 2^f\} - \sum_{i=0}^{f} r_i \cdot 2^i \right) = x - x \bmod 2^f + \{0, 2^f\}$$

Probabilistic Truncation with Prime Modulus

Pre:
- $[x]$ for $x \in [0, 2^k] \subset \mathbb{Z}_p$, p prime such that $p > 2^{k+s}$ for some s
- $[r_0], \ldots, [r_{k+s-1}]$ such that $r_i \overset{\$}{\leftarrow} \{0, 1\}$

Post: $[y]$ such that $y \approx x/2^f$

---

1. Compute and open $[c] = [x] + \sum_{i=0}^{k+s-1} [r_i] \cdot 2^i$
2. Compute $[x'] = (c \bmod 2^f) - \sum_{i=0}^{f-1} [r_i] \cdot 2^i)$
3. Output $(([x] - [x']) \cdot (2^f)^{-1})$

$$
\begin{aligned}
(x - x') \cdot (2^f)^{-1} &= (x - x \bmod 2^f + \{0, 2^f\}) \cdot (2^f)^{-1} \\
&= x/2^f + \{0, 1\}
\end{aligned}
$$

Truncation Error

$$\{0, 2^f\} = \left(\left(x + \sum_{i=0}^{k+s-1} r_i \cdot 2^i\right) \bmod 2^f\right) - \left(x \bmod 2^f + \sum_{i=0}^{f} r_i \cdot 2^i\right)$$

$$= \left(\left(x \bmod 2^f + \sum_{i=0}^{f} r_i \cdot 2^i\right) \stackrel{?}{>} 2^f\right) \cdot 2^f$$

Probabilistic truncation

The larger x, the more likely the inequality is true, which means the more likely the result is rounded up rather than down. This is actually a nice property because some machine learning algorithms love a bit of random noise.

Truncation Security

Concern

The protocol reveals $x + \sum_{i=0}^{k+s-1} r_i \cdot 2^i$

Security

- $x \in [0, 2^k - 1]$, $r \in [0, 2^{s+k} - 1]$
- $x + r$ looks a random value in $[0, 2^{s+k} - 1]$ except with probability 2^{-s}
- Good enough for large s
- What is large? With $s = 40$, a billion repetitions mean an overall failure probability of $\sim 1/1000$.

Truncation Cost

Bit decomposition $k + s$ random bits, $\sim 6k$ multiplications

Probabilistic $k + s$ random bits

Section 4

Exponentation

Integer Exponentation

Pre: $[x]$

Post: $[b^x]$ for positive integer b

1. Bit decomposition into $[x_i]$, $i = 0, \ldots, k-1$
2. Output $\prod_{i=0}^{k-1}(1 + [x_i] \cdot (b^{2^i} - 1))$

$$(1 + x_i \cdot (b^{2^i} - 1)) = \begin{cases} b^{2^i} & x_i = 1 \\ 1 & x_i = 0 \end{cases}$$

$$= b^{x_i \cdot 2^i}$$

$$\Rightarrow \prod_{i=0}^{k-1}(1 + [x_i] \cdot (b^{2^i} - 1)) = \prod_{i=0}^{k-1} b^{x_i \cdot 2^i} = b^{\sum_{i=0}^{k-1} x_i \cdot 2^i}$$

Fractional Exponentation

Approach

1. Reduce to integer base:

$$b^x = 2^{\log_2(b)\cdot x}$$

2. Reduce to integer exponent:

$$2^{\lfloor x \rfloor + (x - \lfloor x \rfloor)} = 2^{\lfloor x \rfloor} \cdot 2^{x - \lfloor x \rfloor}$$

3. Use polynomial approximation (Taylor series) for $x - \lfloor x \rfloor \in [0, 1)$:

$$2^x = \sum_{i=0}^{\infty} \frac{\log(2)^i}{i!} x^i$$