Mixed-Circuit Computation: The Best of Both Worlds

Marcel Keller

CSIRO's Data61

6 April 2022

Domain Trade-offs

Issue

Need to fix a domain for computation

- Arithmetic (modulo larger integer) is good for integer addition and multiplication
- Binary (modulo 2) is good for comparison etc.

Use both?

Need secure conversion because two different protocols implement two disconnected black boxes.

Base Case: One Bit

Pre:
- $[x]_A$ in arithmetic domain, $x \in \{0, 1\}$
- $[r]_A, [r]_B$ for $r \xleftarrow{\$} \{0, 1\}$ (daBit = doubly authenticated bit)

Post:
- $[x]_B$ in binary domain

---

1. Compute and open $[c]_A = [x \oplus r]_A = [x]_A + [r]_A - 2 \cdot [x]_A \cdot [r]_A$
2. Output $[x]_B = [r]_B \oplus c = [r]_B + c$ (XOR is addition modulo 2)

Correctness

$x \oplus r \oplus r = x$

How To Generate daBits

Want

$([r]_A, [r]_B)$ for random secret bit r

Protocol

1. Party i inputs random bit r_i to $[r_i]_A$ and $[r_i]_B$
2. Parties compute $[r]_A$ and $[r]_B$ by $\bigoplus_i r_i$ in both black boxes
   (Reminder: $x \oplus y = x + y - 2xy \in \mathbb{Z}$)

Security

As with random bits earlier, one uniformly random bit makes the result uniformly random at least for all other parties.

Generalizing to Any Value

Pre:
- $[x]_A$ in arithmetic domain, $x \in \mathbb{Z}_{2^k}$
- $([r_0]_A, [r_0]_B), \ldots, ([r_{k-1}]_A, [r_{k-1}]_B)$ for $r_i \xleftarrow{\$} \{0, 1\}$ (daBits)

Post:
- $[x_0]_B, \ldots, [x_{k-1}]_B$ in binary domain such that $x = \sum_{i=0}^{k-1} x_i \cdot 2^i$.

---

1. Compute and open $[c]_A = [x]_A - \sum_{i=0}^{k-1} [r_i] \cdot 2^i$
2. Use a binary adder to add c and r in $[\cdot]_B$ and output the result

Cost

k daBits and $O(k)$ ANDs

Comparison with Mixed Circuits

Arithmetic-Only Comparison

- Compare difference to zero
- k random bits, $O(k)$ multiplications

Mixed-Circuit Comparison with daBits

- Convert difference to binary circuit to access most-significant bit
- k daBits, $O(k)$ ANDs
- daBits cost at least one multiplication (XOR),
  so the cost is still $O(k)$ multiplications

Better daBits

Problem

XOR in arithmetic circuit is expensive

Idea

Minimize computation in arithmetic circuit at the cost of more (cheaper) computation in the binary circuit

Extended daBits

One value on arithmetic side:

$$([r]_A, [r_0]_B, \ldots, [r_{k-1}]_B)$$

such that

$$r \in \mathbb{Z}_{2^k}, (r_0, \ldots, r_{k-1}) \xleftarrow{\$} \{0, 1\}$$

Extended daBit Generation

Want

$([r]_A, [r_0]_B, \ldots, [r_{k-1}]_B)$ such that $r \in \mathbb{Z}_{2^k}, (r_0, \ldots, r_{k-1}) \xleftarrow{\$} \{0,1\}$

Protocol

1. Party i inputs random bits $r_0^i, \ldots, r_{k-1}^i$ to $[r_0^i]_B, \ldots, [r_{k-1}^i]_B$ and $\sum_{i=0}^{k-1} r_i \cdot 2^i$ to $[r^i]_A$
2. Parties compute $[r]_A = \sum_i [r^i]_A$ and $[r_0]_B, \cdot [r_{k-1}]_B$ from $\{\{[r_j^i]\}_{j=0}^{k-1}\}_{i \in P}$ via binary adder

Cost

- No arithmetic multiplications, just one input per party
- One binary adder per party, $O(k)$ ANDs
- ANDs are typically an order of magnitude cheaper than arithmetic multiplications

Comparison Using Extended daBits

Protocol

1. Extract most significant bit after conversion
2. Convert back using daBit if needed

Cost

For n parties:

- $O(n)$ arithmetic operations
- $O(kn)$ binary operations

Extended daBits of Any Length

Previously

Arithmetic value random in full domain $\mathbb{Z}_{2^k}$
$\Rightarrow$ Wrap-around makes overflow disappear

Want

$r \in [0, 2^l - 1], l \neq k$

Challenge

$r + r' \notin [0, 2^l - 1]$ for $r, r', \in [0, 2^l - 1]$ when computing in modulo 2^k

Solution

Compute carry bits in binary domain and convert to arithmetic for correction

General edaBits Generation

Want

$([r]_A, [r_0]_B, \ldots, [r_{l-1}]_B)$ such that $r \in \mathbb{Z}_{2^k}, (r_0, \ldots, r_{l-1}) \xleftarrow{\$} \{0,1\}$

Protocol*

1. Party i inputs random bits $r_0^i, \ldots, r_{l-1}^i$ to $[r_0^i]_B, \ldots, [r_{l-1}^i]_B$ and $\sum_{i=0}^{l-1} r_i \cdot 2^i$ to $[r^i]_A$
2. Parties compute $[r_0]_B, \cdot [r_{l+\lceil \log_2(n) \rceil - 1}]_B$ from $\{\{[r_j^i]\}_{j=0}^{l-1}\}_{i=0}^{n}$ via binary adder
3. Parties convert $[r_l]_B, \cdots, [r_{l+\lceil \log_2(n) \rceil - 1}]_B$ to $[\cdot]_A$ using daBits
4. Parties compute $[r]_A = \sum_i [r^i]_A - \sum_{i=l}^{l+\lceil \log_2(n) \rceil - 1} [r_i]_A \cdot 2^i$

General edaBit Cost

- As before: $O(n)$ arithmetic inputs, $O(nl)$ binary inputs
- $O(n(l + \log(n))$ ANDs
- $O(\log(n))$ daBits
- Nothing $O(l)$ in arithmetic circuit

Probabilistic Truncation Using edaBits*

Pre:
- $[x]_A$, $x \in [0, 2^{k-1} - 1] \subsetneq \mathbb{Z}_{2^k}$
- $(k - f - 1)$-bit edaBit $[r]$, f-bit edaBit $[r']$
- Random bit $[b]_A$

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

---

1. Parties compute and open $[c]_A = [x] + 2^{k-1} \cdot [b]_A + 2^m \cdot [r]_A + [r']_A$
2. Parties compute $[v]_A = [b]_A \oplus c/2^{k-1}$ (indicating overflow)
3. Output $(c \bmod 2^{k-1})/2^m - [r]_A + 2^{k-1-m} \cdot [v]_A$

- Computation only in arithmetic domain but edaBit generation requires mixed
- No $O(k)$ or $O(f)$ cost in arithmetic domain
- Error the same as earlier

Section 1

Local Conversion

Setup

Previously

Generic methods for any computation over $\mathbb{Z}_{2^k}$

Question

Use secret sharing directly for conversion?

Local Conversion for 2-Party Additive Secret Sharing

Additive Secret Sharing

$$
\begin{aligned}
x &= x^0 + x^1 \mod 2^k \\
&= \sum_{i=0}^{k-1} x_i^0 \cdot 2^i + \sum_{i=0}^{k-1} x_i^1 \cdot 2^i = \sum_{i=0}^{k-1} (x_i^0 + x_i^1) \cdot 2^i
\end{aligned}
$$

Approach

$(x_i^j, 0)$ is a valid secret sharing in binary because $x_i^j \oplus 0 = x_i^j$.

$\Rightarrow$ Compute $[x]_B$ from $[x_i^j]_B$ with a binary adder.

Generate edaBits Using Local Conversion

Protocol for additive secret sharing

1. Parties generate $r^i \xleftarrow{\$} [0, 2^l - 1]$, denote by $[r]$ the secret sharing defined $\{r^i\}$
2. Parties use local share conversion to generate $[r_l]_B, \ldots, [r_{l+\lceil \log(n) \rceil - 1}]_B$, the overflow bits of $\sum_i r^i$
3. Parties use daBits to convert the overflow bits to $[r_l]_A, \ldots, [r_{l+\lceil \log(n) \rceil - 1}]_A$
4. Parties output $[r] - \sum_{i=l}^{l+\lceil \log(n) \rceil - 1} [r_i]_A \cdot 2^i$