Secure Computation on Floating Point Numbers

Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele
University of Notre Dame

February 27, 2013
Outline

1 Introduction
   Secure Multiparty Computation
   Framework
   Building Blocks

2 New tools
   New Building Blocks
   Basic FL Operations
   Complex FL Operations
   Type Conversion

3 Security Analysis and Experiments
Outline

1. Introduction
   Secure Multiparty Computation
   Framework
   Building Blocks

2. New tools
   New Building Blocks
   Basic FL Operations
   Complex FL Operations
   Type Conversion

3. Security Analysis and Experiments
Outline

1. **Introduction**
   - Secure Multiparty Computation
     - Framework
     - Building Blocks

2. **New tools**
   - New Building Blocks
   - Basic FL Operations
   - Complex FL Operations
   - Type Conversion

3. **Security Analysis and Experiments**
SMC

A number of parties \( (n > 2) \) wish to jointly and securely compute a known function \( (F) \) on their private inputs.

- Privacy-Preserving Computation
- Secure Outsourcing / Cloud Computation
- Secure Collaborative Computation
Secure Multiparty Computation

SMC-Cont.

Recent progress has made it fast.

- Generally, any computable function can be evaluated securely (e.g., as a Boolean or arithmetic circuit)
- Optimization of existing techniques
SMC-Cont.

Recent progress has made it fast.

• Generally, any computable function can be evaluated securely (e.g., as a Boolean or arithmetic circuit)
• Optimization of existing techniques
• Mainly integer domain
• Little attempt on real numbers
SMC-Cont.

Recent progress has made it fast.

- Generally, any computable function can be evaluated securely (e.g., as a Boolean or arithmetic circuit)
- Optimization of existing techniques
- Mainly integer domain
- Little attempt on real numbers
- NO Floating point support in SMC
Outline

1. **Introduction**
   - Secure Multiparty Computation
   - Framework
   - Building Blocks

2. **New tools**
   - New Building Blocks
   - Basic FL Operations
   - Complex FL Operations
   - Type Conversion

3. **Security Analysis and Experiments**
Secret Sharing

Linear Secret Sharing scheme (such as Shamir secret sharing scheme \(^1\))

- \(P_1 \cdots P_n\) parties engage in a \((n, t)\)-secret sharing scheme \((t < n/2)\)
- \([\times]\)
- Linear combination of secrets can be computed locally
- Multiplication of two secrets requires one round of an interactive operation
- Performance Metric: \# of interactive operations along with \# of sequential interactions (rounds)

---

\(^1\) A. Shamir. How to share a secret. Communications of the ACM, 1979
Outline

1 Introduction
- Secure Multiparty Computation Framework
- Building Blocks

2 New tools
- New Building Blocks
- Basic FL Operations
- Complex FL Operations
- Type Conversion

3 Security Analysis and Experiments
• $[b] \leftarrow LT([x], [y], \ell)$ Catrina and de Hoogh’s takes 4 rounds and $4\ell - 2$ interactive operations.

• $[y] \leftarrow Trunc([x], \ell, m)$ 4 rounds and $4m + 1$ interactions

• $[x_{m-1}] \cdots [x_0] \leftarrow BitDec([x], \ell, m)$ log $m$ rounds and $m \log(m)$ interactions
Outline

1. Introduction
   Secure Multiparty Computation
   Framework
   Building Blocks

2. New tools
   New Building Blocks
   Basic FL Operations
   Complex FL Operations
   Type Conversion

3. Security Analysis and Experiments
FL Representation

\[ u = (1 - z)(1 - 2s)v2^p \]

- Normalized Value \( v \in [2^{\ell-1}, 2^\ell) \)
- Power \( p \in (-2^{k-1}, 2^{k-1}) \)
- Sign indicator \( s = \{0, 1\} \)
- Zero indicator \( z = \{0, 1\} \)
  - \( u = 0 \iff z = 1, v = p = 0 \)
FL Representation
\[ u = (1 - z)(1 - 2s)v2^p \]
- Normalized Value \( v \in [2^{\ell-1}, 2^\ell) \)
- Power \( p \in (-2^{k-1}, 2^{k-1}) \)
- Sign indicator \( s = \{0, 1\} \)
- Zero indicator \( z = \{0, 1\} \)
  - \( u = 0 \iff z = 1, v = p = 0 \)

Error Detection:
- Invalid operation
- Division by zero
- Overflow and Underflow
FL Representation
\[ u = (1 - z)(1 - 2s)v2^p \]
- Normalized Value \( v \in [2^{\ell-1}, 2^\ell) \)
- Power \( p \in (-2^{k-1}, 2^{k-1}) \)
- Sign indicator \( s = \{0, 1\} \)
- Zero indicator \( z = \{0, 1\} \)
  \[ u = 0 \iff z = 1, v = p = 0 \]

Error Detection:
- Invalid operation
- Division by zero
- Overflow and Underflow
- Inexact
Outline

1. Introduction
   Secure Multiparty Computation Framework
   Building Blocks

2. New tools
   New Building Blocks
   Basic FL Operations
   Complex FL Operations
   Type Conversion

3. Security Analysis and Experiments
New Building Blocks

- \([y] \leftarrow \text{Trunc}([a], \ell, [m])\)
  - \(O(\ell)\) invocations and \(O(\log \log \ell)\) rounds

- \([a_0], \ldots, [a_{\ell-1}] \leftarrow \text{B2U}([a], \ell)\)
  - \(O(\ell)\) invocations and \(O(\log \log \ell)\) rounds

- \([2^a] \leftarrow \text{Pow2}([a], \ell)\)
  - \(O((\log \ell)(\log \log \ell))\) invocations and \(O(\log \log \ell)\) rounds
Outline

1. **Introduction**
   - Secure Multiparty Computation Framework
   - Building Blocks

2. **New tools**
   - New Building Blocks
   - Basic FL Operations
   - Complex FL Operations
   - Type Conversion

3. **Security Analysis and Experiments**
Basic FL Operations

Basic FL-1

• $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLMul}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$
  • $O(\ell)$ invocations and $O(1)$ rounds
Introduction

New tools

Security Analysis and Experiments

Basic FL Operations

Basic FL-1

\[ \langle [v], [p], [z], [s] \rangle \leftarrow \text{FLMul}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle) \]

- \( O(\ell) \) invocations and \( O(1) \) rounds

\[ \langle [v], [p], [z], [s] \rangle \leftarrow \text{FLDiv}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle) \]

- \( O(\ell \log \ell) \) invocations and \( O(\log \ell) \) rounds
Basic FL Operations

Basic FL-1

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLMul}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$
  - $O(\ell)$ invocations and $O(1)$ rounds

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLDiv}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$
  - $O(\ell \log \ell)$ invocations and $O(\log \ell)$ rounds

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLAdd}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \langle [v_2], [p_2], [z_2], [s_2] \rangle)$
  - $O(\ell \log \ell + k)$ invocations and $O(\log \ell)$ rounds
Basic FL Operations

Basic FL-2

- \([b] \leftarrow \text{FLLT}(⟨[v_1], [p_1], [z_1], [s_1]⟩, ⟨[v_2], [p_2], [z_2], [s_2]⟩)\)
  - \(O(\ell + k)\) invocations and \(O(1)\) rounds

- \(⟨[v], [p], [z], [s]⟩ \leftarrow \text{FLRound}(⟨[v_1], [p_1], [z_1], [s_1]⟩, \text{mode})\)
  - \(\text{mode} = 0 \rightarrow \text{floor and mode} = 1 \rightarrow \text{ceiling}\)
  - \(O(\ell + k)\) invocations and \(O(\log \log \ell)\) rounds
FLRound

\[
\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLRound}(\langle [v_1], [p_1], [z_1], [s_1] \rangle, \text{mode})
\]

- \([a] \leftarrow \text{LTZ}([p_1], k)\);
- \([b] \leftarrow \text{LT}([p_1], -\ell + 1, k)\);
- \(\langle [v_2], [2^{-p_1}] \rangle \leftarrow \text{Mod2m}([v_1], \ell, -[a](1 - [b])[p_1])\);
- \([c] \leftarrow \text{EQZ}([v_2], \ell)\);
- \([v] \leftarrow [v_1] - [v_2] + (1 - [c])[2^{-p_1}](\text{XOR}(\text{mode}, [s_1]))\);
- \([d] \leftarrow \text{EQ}([v], 2^\ell, \ell + 1)\);
- \([v] \leftarrow [d]2^{\ell-1} + (1 - [d])[v]\);
- \([v] \leftarrow [a][(1 - [b])[v] + [b](\text{mode} - [s_1])) + (1 - [a])[v_1]\);
- \([s] \leftarrow (1 - [b]\text{mode})[s_1]\);
- \([z] \leftarrow \text{OR}(\text{EQZ}([v], \ell), [z_1])\);
- \([v] \leftarrow [v](1 - [z])\);
- \([p] \leftarrow ([p_1] + [d][a](1 - [b]))(1 - [z])\);
Outline

1. **Introduction**
   - Secure Multiparty Computation Framework
   - Building Blocks

2. **New tools**
   - New Building Blocks
   - Basic FL Operations
   - Complex FL Operations
   - Type Conversion

3. **Security Analysis and Experiments**
Complex FL Operations

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLSqrt}(\langle [v_1], [p_1], [z_1], [s_1] \rangle)$

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLExp2}(\langle [v_1], [p_1], [z_1], [s_1] \rangle)$

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FLLog2}(\langle [v_1], [p_1], [z_1], [s_1] \rangle)$
Outline

1. Introduction
   Secure Multiparty Computation Framework
   Building Blocks

2. New tools
   New Building Blocks
   Basic FL Operations
   Complex FL Operations
   Type Conversion

3. Security Analysis and Experiments
• Integer: $\gamma$ bits

• Fixed point: $u = \overline{u}2^{-f}$
  • $\overline{u}$: signed $\gamma$-bit integer

• Floating Point: $u = (1 - 2s)(1 - z)v2^p$
  • $v$: normalized $\ell$-bit value and
  • $p$: signed $k$-bit exponent
  • $k > \max(\lceil \log(\ell + f) \rceil, \lceil \log(\gamma) \rceil)$
Conversion-cont.

\[ \langle [v], [p], [z], [s] \rangle \leftarrow \text{Int2FL}([a], \gamma, \ell) \]
Conversion-cont.

• $\langle[v], [p], [z], [s]\rangle \leftarrow \text{Int2FL}([a], \gamma, \ell)$

• $[g] \leftarrow \text{FL2Int}(\langle[v], [p], [z], [s]\rangle, \ell, k, \gamma)$
Conversion-cont.

- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{Int2FL}([a], \gamma, \ell)$
- $[g] \leftarrow \text{FL2Int}(\langle [v], [p], [z], [s] \rangle, \ell, k, \gamma)$
- $\langle [v], [p], [z], [s] \rangle \leftarrow \text{FP2FL}([g], \gamma, f, \ell, k)$
- $[g] \leftarrow \text{FL2FP}(\langle [v], [p], [z], [s] \rangle, \ell, k, \gamma, f)$
Outline

1 Introduction
   Secure Multiparty Computation
   Framework
   Building Blocks

2 New tools
   New Building Blocks
   Basic FL Operations
   Complex FL Operations
   Type Conversion

3 Security Analysis and Experiments
Security

- Cannetti’s composition theorem

- Secure in the malicious adversaries model
Experiments

- **Integer**: $\gamma = 64$
  - $|q| > 2\gamma + \kappa + 1 = 177$
- **Fixed point**: $\gamma = 64$ and $f = 32$ (precision: $2^{-32}$)
  - $|q| > \gamma + 3f + \kappa = 208$
- **Floating point**: $\ell = 32$ and $k = 9$ (precision: $2^{-256}$)
  - $|q| > 2\ell + \kappa + 1 = 113$
Experiments Cont.

- C/C++ using the GMP library
- (3, 1)-Shamir secret sharing
- Boost libraries for communication and OpenSSL for securing the communication
- 2.2 GHz Linux machines on a 1Gbps LAN
Addition

![Graph showing time per operation vs. operations for Integer, Fixed Point, and Floating Point arithmetic.](image-url)

- **Time / Operation (ms)**
  - Integer
  - Fixed Point
  - Floating Point

**Operations**
- 10^1
- 10^2
- 10^3
- 10^4
- 10^5

**Graph Legend**
- Red line with circles: Integer
- Blue dashed line with squares: Fixed Point
- Black dashed line with diamonds: Floating Point
Multiplication

![Graph showing time per operation for different types of arithmetic operations (integer, fixed point, floating point) plotted on a logarithmic scale.](image-url)

- **Legend:**
  - Red solid line: Integer
  - Blue dashed line: Fixed Point
  - Blue dotted line: Floating Point
Division

![Graph showing time per operation for Integer, Fixed Point, and Floating Point divisions.](image-url)

- **Time / Operation (ms)**
  - **10**
  - **20**
  - **40**
  - **60**
  - **80**

- **Operations**
  - **10^1**
  - **10^3**
  - **10^5**

Legend:
- **Red circles** (Integer)
- **Blue dashed line** (Fixed Point)
- **Black dotted line** (Floating Point)
Comparison

![Graph comparing time per operation for different types of numbers: Integer, Fixed Point, and Floating Point. The x-axis represents the number of operations, ranging from $10^1$ to $10^5$, and the y-axis represents time per operation in milliseconds. The graph shows a decrease in time as the number of operations increases for all types of numbers. Integer operations have the least time, followed by Fixed Point, and then Floating Point.](image-url)
Exp & Log

![Graph showing the time / operation (ms) for Logarithm and Exponentiation operations over different numbers of operations.](figure)

*Figure: Test NDSS'13 Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele University of Notre Dame*