Efficient MPC

Correlated Randomness and Arithmetic Circuits

Claudio Orlandi, Aarhus University
We’re hiring!

• PhD students, postdocs, assistant professors (tenure track), associate professors

• **Topics**: blockchain, differential privacy, zero-knowledge proofs, secure multiparty computation, formal verification, language design and semantics for smart contracts, ...

• More info at [https://iacr.org/jobs/](https://iacr.org/jobs/)
Online Poker

2♠, 5♠, 2♥, 5♥, J♦
Q♠, Q♣, 7♠, 3♥, 2♦
10♠, 9♣, 8♠, 7♦, 6♦
3♠, 4♠, 7♥, Q♦, 10♦
Poker with Pirates

2♠, 5♠, 2♥, 5♥, J♦

Q♠, Q♣, 7♠, 3♥, 2♦,

10♦, 9♣, 8♠, 7♦, 6♦

A♠, A♣, A♥, A♦, K♦
Secure Computation
Hospitals and Insurances

- **Problem:** Sick people forget to claim compensations from insurance

- **Solution:** Insurances and hospitals could periodically compare their data to find and help these people

- **Privacy Issue:** insurance and medical records are sensitive data! No other information than what is strictly necessary must be disclosed!
MPC Goes Live (2008)

Bogetoft et al.  
“Multiparty Computation Goes Live”

- January 2008
- **Problem**: determine market price of sugar beets contracts
- 1200 farmers
- Computation: 30 minutes
Last decade: commercial interest and social value of MPC

- Estonian study on student dropout
- Boston women workforce council, study on wage gap

Figure 1: Illustration of a deployment of the protocol implementation for two participants.
Secure Computation

- Privacy
- Correctness
- Input independence
- ...
Part 1: Correlated Randomness and Arithmetic Circuits

- Warmup: One-Time Truth Tables

- Arithmetic Black Box and Evaluating Circuits with Beaver’s trick

- Simple Unconditionally Secure Protocols
Trusted Party

\[ x \rightarrow z \rightarrow y \]

Trusted Dealer

\( (r_A, r_B) \leftarrow D \)

\[ r_A \leftarrow r_B \]

\[ f(x, y) \]
“The simplest 2PC protocol ever”

\[(r_A, r_B) \leftarrow D\]
“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

1) Write the truth table of the function F you want to compute

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x</strong></td>
<td><strong>y</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
“The simplest 2PC protocol ever” OTTT
(Preprocessing phase)

2) Pick random \((r, s)\), rotate rows and columns

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
0 & 1 & 4 & 4 & 1 \\
1 & 2 & 2 & 2 & 3 \\
2 & 0 & 0 & 4 & 3 \\
3 & 0 & 0 & 4 & 1 \\
\end{array}
\]
“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

3) Secret share the truth table i.e.,

Pick $T_1$ at random, and let

\[
\begin{array}{cccc}
1 & 4 & 4 & 1 \\
2 & 2 & 2 & 3 \\
0 & 0 & 4 & 3 \\
0 & 0 & 4 & 1 \\
\end{array}
\]

$-$

$T_1$
“The simplest 2PC protocol ever”

(Online phase)

\[ u = x + r \]

\[ v = y + s \]

\[ z_2 = T_2[u, v] \]

output \( f(x, y) = T_1[u, v] + z_2 \)

“Privacy”: inputs masked w/uniform random values

Correctness: by construction
"The simplest 2PC protocol ever" OTTT

\[ u = x + r \]

Simulated view, given \( x \) and \( f(x,y) \) (but not \( y \))

\[ z_2 = f(x,y) - T1[u,v] \]

output \( f(x,y) = T1[u,v] + z_2 \)
What about active security?

\[ u = x + r \]
\[ v = y + s \]
\[ z_2 = T2[u,v] \]

output \( f(x,y) = T1[u,v] + z_2 \)
What about active security?

\[ u = x + r \]
\[ v = y + s + e_1 \]
\[ T_2[u,v] + e_2 \]
Is this cheating?

- \( v = y + s + e1 = (y + e1) + s = y' + s \)
  - Input substitution, **not cheating** according to the definition!

- \( M2[u,v] + e2 \)
  - Changes output to \( z' = f(x,y) + e2 \)
  - Example: \( f(x,y)=1 \iff x=y \) (e.g. pwd check)
  - \( e2=1 \) the output is 1 whp  (login without pwd!)
    - *Clearly breach of security!*
**Force Bob to send the right value**

- **Problem:** Bob can send the wrong shares
- **Solution:** use MACs
  - e.g. \( m=ax+b \) with \( (a,b) \leftarrow F \) (e.g., \( F=\mathbb{Z}_p \) with \( p \geq 2^k \) prime)

Abort if \( m' \neq ax'+b \)
OTTT+MAC

\[ u = x + r \]
\[ v = y + s \]

\[ \text{output } f(x,y) = T1[u,v] + T2[u,v] \]
else
\[ \text{abort} \]

Statistical security vs. malicious Bob w.p. \(1 - 2^{-k}\)
“The simplest 2PC protocol ever” OTTT

- Optimal communication complexity 😊

- Storage exponential in input size 😞

→ Represent function using circuit instead of truth table!
Part 1: Correlated Randomness and Arithmetic Circuits

• Warmup: One-Time Truth Tables

• Arithmetic Black Box and Evaluating Circuits with Beaver’s trick

• Simple Unconditionally Secure Protocols
Circuit based computation
What kind of circuit?

• **Boolean**
  – Addition & Multiplication modulo 2 (XOR, AND)

• **Arithmetic: which modulo?**
  – In a field \((\mathbb{Z}_p, \text{GF}(2^k))\)?
  – Determined by Public Key (e.g., Paillier, LWE, ...)
  – Arbitrary? (e.g., modulo \(2^{32}\))
The Arithmetic Black Box (ABB)

• A reactive functionality which allows to manipulate secret values

• Often a good abstraction:
  – if you want to implement some algorithm in MPC, you might not care too much about how operation are implemented, just what the ”interface” is.
ABB: Basic Commands

• \([x] \leftarrow \text{Input}(P_i, x)\)
  – Party \(P_i\) inputs a secret value \(x\), all other parties get a “handle/pointer” to \([x]\)

• \(x \leftarrow \text{Open}(P_j, [x])\)
  – If all parties agree, party \(P_j\) learns the secret value contained in \([x]\)

• \([z] \leftarrow \text{Add}([x],[y])\) // or \([z] = [x] + [y]\)
  – If all parties agree, a new handle \([z]\) is created such that \(z=x+y\)
    – \([z] \leftarrow \text{Add}(c,[x]), [z] \leftarrow \text{Mul}(c,[x])\) easy from Add

• \([z] \leftarrow \text{Mul}([x],[y])\) // or \([z] = [x] \cdot [y]\)
  • If all parties agree, a new handle is created such that \(z=x \cdot y\)
ABB: Advanced (Efficient) Commands

• \([r] \leftarrow \text{Rand}()\)
  – Generate a random handle for \(r\)
  – Could have been implemented by \([r_i] \leftarrow \text{Input}(P_i, r_i)\) and \([r] \leftarrow [r_1] + \ldots + [r_n]\)

• \(b \leftarrow \text{IsZero}([x])\)
  – Could be implemented by \([z] = [x] \cdot [r]\) for random \(r\), then open \(z\) and check if \(z = 0\).

• \([x_1], \ldots, [x_n] \leftarrow \text{BitsOf}([x])\)
  – Useful and typically expensive

• Exercise: how would you implement these?
  – \([b] \leftarrow \text{IsZero}([x])\)  // \(b = 1\) iff \(x = 0\)
  – \(b \leftarrow \text{Equality}([x],[y])\)  // \(b = 1\) iff \(x = y\)
  – \(b \leftarrow \text{IsBit}([x])\)  // \(b = 1\) iff \(x \in \{0,1\}\)
Beaver’s random triples trick

\[[z] \leftarrow \text{Mul}([x],[y]):\]

1. \(([a],[b],[c]) \leftarrow \text{RandMul}()\)
   
   Creates random tuple such that \(c = a \times b\)

2. \(e = \text{Open}([a] + [x])\)

3. \(d = \text{Open}([b] + [y])\)

4. Compute \([z] = [c] + e[y] + d[x] - ed\)
   
   \(ab + (ay + xy) + (bx + xy) - (ab + ay + bx + xy)\)

\(Is this secure?\)

\(e,d\) are “one-time-pad” encryptions of \(x\) and \(y\) using \(a\) and \(b\)
Beaver and Preprocessing

- Independent of $x,y$
- Typically only depends on size of $f$
- Uses public key cryptography (slower)

- Uses only information theoretic tools (order of magn. faster)
Implementing the Arithmetic Black Box

• How to implement the basic commands?
  – Input, Add, Mul/RandMul

• In the remaining time:
  – Additive Secret Sharing
    • Passive Security
    • Active Security
  – Replicated Secret Sharing
  – Shamir Secret Sharing
Invariant

• For each *wire x* in the circuit we have
  – \([x] := (x_1, x_2)\) \(\text{// read “x in a box”}\)
  – Where Alice holds \(x_1\)
  – Bob holds \(x_2\)
  – Such that \(x_1 + x_2 = x\)

• Notation overload:
  – \(x\) is both the r-value and the l-value of \(x\)
  – use \(n(x)\) for name of \(x\) and \(v(x)\) for value of \(x\) when in doubt.
  – Then \([n(x)] = (x_1, x_2)\) such that \(x_1 + x_2 = v(x)\)
Circuit Evaluation

1) \([x] \leftarrow \text{Input}(A,x)\):
   - chooses random \(x_2\) and send it to Bob
   - set \(x_1 = (x + x_2) \mod M\) \hspace{1cm} // symmetric for Bob
     \hspace{1cm} // mod omitted from now on

   Alice only sends a random value! “Clearly” secure

2) \(z \leftarrow \text{Open}(A,[z])\):
   - Bob sends \(z_2\)
   - Alice outputs \(z = (z_1 + z_2)\) \hspace{1cm} // symmetric for Bob

   Alice should learn \(z\) anyway! “Clearly” secure
2) \( z \leftarrow \text{Add}([x],[y]) \quad // \text{at the end } z=x+y \)

- Alice computes \( z_1 = x_1 + y_1 \)
- Bob computes \( z_2 = x_2 + y_2 \)

No interaction! “Clearly” secure

“for free” : only a local addition!
Circuit Evaluation

2a) \([z] \leftarrow \text{Mul}(c,[x])\)  
   // at the end \(z = c \times x\)
   - Alice computes \(z_1 = c \times x_1\)
   - Bob computes \(z_2 = c \times x_2\)

2c) \([z] \leftarrow \text{Add}(c,[x])\)  
   // at the end \(z = c + x\)
   - Alice computes \(z_1 = c + x_1\)
   - Bob computes \(z_2 = x_2\)
3) Multiplication?

How to compute $[z]=[xy]$?

Alice, Bob should compute

$$z_1 + z_2 = (x_1 + x_2)(y_1 + y_2)$$

$$= x_1 y_1 + \overbrace{x_2 y_1 + x_1 y_2} + \overbrace{x_2 y_2}$$

Alice can compute this

Bob can compute this

How do we compute this?
RandMul() with Trusted Dealer

Pick random $a_1, a_2, b_1, b_2, c_1$ and

$c_2 = (a_1 + a_2)(b_1 + b_2) - c_1$
Implementing the Arithmetic Black Box

• How to implement the basic commands?
  – Input, Add, Mul/RandMul

• In the remaining time:
  – **Additive Secret Sharing**
    • Passive Security
    • **Active Security**
  – Replicated Secret Sharing
  – Shamir Secret Sharing
Secure Computation

\[ z^* = (x_1 + e) \]

\[ w \]

\[ w + e \]
Active Security?

• “Privacy?”
  – even a malicious Bob does not learn anything 😊

• “Correctness?”
  – a corrupted Bob can change his share during any “Open” (both final result or during multiplication) leading the final output to be incorrect 😞
Problem

2) \( z \leftarrow \text{Open}(A,[z]) \):

- Bob sends \( z_2 + e \)
- Alice outputs \( z = z_1 + z_2 + e \) // error change output distribution in way that cannot be simulated by input substitution
Authenticated Shares

• **Passive share**: [x] means
  – Alice has $x_1$, Bob has $x_2$,
    $$x_1 + x_2 = x$$

• **MAC on Share** $\langle x \rangle$ (BeDOZa, TinyOT, ...):
  – [x] plus:
    – Bob has a MAC key $(\Delta_2, K_2)$, Alice has a MAC $M_1$:
      $$M_1 = \Delta_2 x_1 + K_2$$
    – (Symmetric for Bob)
Authenticated Shares

• Is the representation $\lbrack x \rbrack$ still linear?
  Yes, if $\Delta_1, \Delta_2$ are “global” keys

$\lbrack x \rbrack = ([x], (\Delta_1, K_1(x), M_1(x)), (\Delta_2, K_2(x), M_2(x)))$

$\lbrack y \rbrack = ([y], (\Delta_1, K_1(y), M_1(y)), (\Delta_2, K_2(y), M_2(y)))$

$\lbrack z \rbrack = ([x+y],$

$(\Delta_1, K_1(x)+K_1(y), M_1(x) + M_1(y)),$

$(\Delta_2, K_2(x)+K_2(y), M_2(x) + M_2(y)))$
Better MACs for MPC

• **SPDZ:**
  – **Problem:** with MAC on Share you need to store a MAC for every other party!
  – **Solution:** MAC value directly instead
    – $\llbracket x \rrbracket = ([x], [M(x)], [\Delta])$ with $M(x) = \Delta x$ ($\Delta$ is global)

• **MiniMAC:**
  – **Problem:** MAC must be large for unpredictability. If working in small field, need to have multiple MACs per value.
  – **Solution:** Compute MAC on vector of bits instead

• **SPDZ2K:**
  – **Problem:** MACs don’t work modulo power of 2’s (not a field).
  – **Solution:** compute MAC modulo $2^{k+s}$

• ...
Implementing the Arithmetic Black Box

• How to implement the basic commands?
  – Input, Add, Mul/RandMul

• In the remaining time:
  – Additive Secret Sharing
    • Passive Security
    • Active Security
  – Replicated Secret Sharing
  – Shamir Secret Sharing
Implementing the Arithmetic Black Box

• How to implement the basic commands?
  – Input, Add, Mul/RandMul

• In the remaining time:
  – Additive Secret Sharing
    • Passive Security
    • Active Security
  – Replicated Secret Sharing
  – Shamir Secret Sharing
Replicated Secret Sharing

• **[x]** means:
  - \( x = x_1 + x_2 + x_3 \) where
  - \( P_1 \) knows \((x_1, x_2)\)
  - \( P_2 \) knows \((x_2, x_3)\)
  - \( P_3 \) knows \((x_3, x_1)\)

• **[x]** \( \leftarrow \) **Input**\((P_i, x)\)
  - \( P_i \) picks random shares and distributes them.

• **x** \( \leftarrow \) **Open**\((P_i, [x])\)
  - Everyone sends their shares to \( P_i \) who reconstructs.

• **[x]** \( \leftarrow \) **Add**\(([x], [y])\)
  - Everyone locally adds their shares.

\( n=3 \) parties \( t \leq 1 \) corruptions

No party alone can reconstruct the secret
Replicated Secret Sharing

- $[z] = \text{Mul}([x],[y])$

Goal, compute random such that

$$z = (x_1 + x_2 + x_3)(y_1 + y_2 + x_3) = x_1y_1 + x_2y_1 + x_3y_1 + x_1y_2 + x_2y_2 + x_3y_2 + x_1y_3 + x_2y_3 + x_3y_3$$

$n=3$ parties $t \leq 1$ corruptions
• \([z] = \text{Mul}([x],[y])\)
  
  – \(P_1\) computes \(z_1 = x_1y_1 + x_2y_1 + x_1y_2\)
    • Symmetric for \(P_2, P_3, \ldots\)
  
  – \([z_1] \leftarrow \text{Input}(P_1,z_1)\)  
    // Why resharing?
    • Symmetric for \(P_2, P_3, \ldots\)

  – \([z] = [z_1] + [z_2] + [z_3]\)
Implementing the Arithmetic Black Box

• How to implement the basic commands?
  – Input, Add, Mul/RandMul

• In the remaining time:
  – Additive Secret Sharing
    • Passive Security
    • Active Security
  – Replicated Secret Sharing
  – Shamir Secret Sharing
Shamir vs. Replicated Secret Sharing

• **Share size:**
  – Shamir is optimal (size of share = size of secret)
  – RSS scales horribly with the number of parties

• **Generality:**
  – Shamir works only in fields
  – RSS works in any ring
• \([x]\) means:
  
  - \(x = p(0)\) where
  - \(p(\alpha) = x_0 + x_1\alpha\)
  - \(P_1\) knows \(p(1)\)
  - \(P_2\) knows \(p(2)\)
  - \(P_3\) knows \(p(3)\)
Shamir Secret Sharing

• \([x]\) means:
  - \(x = p(0)\) where
  - \(p(\alpha) = x_0 + x_1\alpha\)
  - \(P_1\) knows \(p(1)\)
  - \(P_2\) knows \(p(2)\)
  - \(P_3\) knows \(p(3)\)

No party alone can reconstruct the secret
Shamir Secret Sharing

• [x] means:
  – $x = p(0)$ where
  – $p(\alpha) = x_0 + x_1 \alpha$
  – $P_1$ knows $p(1)$
  – $P_2$ knows $p(2)$
  – $P_3$ knows $p(3)$

Any two parties can reconstruct $x$
Reconstruction - Details

• Given \( p(1), p(2) \) one can reconstruct \( p(x) \) as

\[
p(\alpha) = \delta_1(\alpha)p(1) + \delta_2(\alpha)p(2)
\]

• \( \delta_i(\alpha) \) is a poly s.t.

\[
\begin{align*}
\delta_i(i) &= 1 \\
\delta_i(j) &= 0 \text{ for all } j \text{ in the reconstruction set (except } i) 
\end{align*}
\]

• In our case

\[
\begin{align*}
\delta_1(\alpha) &= (\alpha -2)(1-2)^{-1} \\
\delta_2(\alpha) &= (\alpha -1)(2-1)^{-1}
\end{align*}
\]

• To reconstruct secret enough to compute

\[
p(0) = \delta_1(0)p(1) + \delta_2(0)p(2)
\]

• (Generalizes to any other degree)
n=3 parties
t≤1 corruptions
Computations in field

Shamir Secret Sharing

• \([z]=\text{Add}([x],[y])\) means:
  - \(x=p(0), y=q(0)\)
  - \(p(\alpha) = x_0 + x_1\alpha\)
  - \(q(\alpha) = y_0 + y_1\alpha\)

- \(P_1\) computes \(p(1)+q(1)\)
- \(P_2\) computes \(p(2)+q(2)\)
- \(P_3\) computes \(p(3)+q(3)\)
• \([z]=\text{Mul}([x],[y])\) (part 1):
  – \(x=p(0), y=q(0)\)
  – \(p(\alpha) = x_0 + x_1\alpha\)
  – \(q(\alpha) = y_0 + y_1\alpha\)

  – \(P_1\) computes \(t(1)=p(1)\times q(1)\)
  – \(P_2\) computes \(t(2)=p(2)\times q(2)\)
  – \(P_3\) computes \(t(3)=p(3)\times q(3)\)

• \(t(0)=xy\) (as desired)
• But \(t\) has the wrong degree!
• \(t(\alpha) = t_0 + t_1\alpha + t_2\alpha^2\)
Shamir Secret Sharing

- \([z]=\text{Mul}([x],[y])\) (part 2):
  - \([z_1] \leftarrow \text{Input}(P_1,t(1))\)
  - Symmetric for \(P_2, P_3\)
  - Then reconstruct i.e.

\[
[t(0)]=\delta_1[t(1)]+\delta_2[t(2)]+\delta_2[t(3)]
\]
  - But \(t(0)=z\), so we’re done!

- Exercise: find the the values \(\delta_1,\delta_2,\delta_3\)
  (Hint, the degree is different this time!)
Recap

• Simple protocols with trusted dealer
  – OTTT
  – Circuit evaluation with random triples
  – Active security via information theoretic MACs

• Simple protocols for 3 parties, 1 corruption
  – Replicated Secret Sharing
  – Shamir Secret Sharing

Coming up next:
• How to get rid of the trusted dealer?
  – Protocols for secure multiplication
  – OT and OT extension

• Efficiency of 2PC based on garbled circuits
  – Garbling techniques
  – Techniques for Active Security

• If time (and patience) allows
  – Anonymity in Cryptocurrencies
Primary References

• Cryptographic Computing, lecture notes, [http://orlandi.dk/crycom](http://orlandi.dk/crycom) (with theory and programming exercises)
• On the Power of Correlated Randomness in Secure Computation (Ishai et al.)
• Semi-homomorphic Encryption and Multiparty Computation (Bendlin et al.)
• Secure multi-party computation made simple (Maurer)
• A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation (Asharov et al.)
• A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority (Lindell et al.)
Other References

• A New Approach to Practical Active-Secure Two-Party Computation (Nielsen et al.)
• Web-based Multi-Party Computation with Application to Anonymous Aggregate Compensation Analytics (Lapets et al.)
• Multiparty Computation Goes Live (Bogetoft et al.)
• Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation (Bogdanov et al.)
• Efficient Multiparty Protocols Using Circuit Randomization (Beaver)
• How to Share a Secret (Shamir)
• Chaum et al. (Multiparty Unconditionally Secure Protocols)
• SPD\(Z_{2k}\): Efficient MPC mod 2k for Dishonest Majority (Cramer et al.)
• Constant-Overhead Secure Computation of Boolean Circuits using Preprocessing (Damgård et al.)
• Multiparty Computation from Somewhat Homomorphic Encryption (Damgård et al.)
• Primitives and applications for multi-party computation (Toft)
• Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Ben-Or et al.)