RETHINKING ALGORITHMS FOR SECURE COMPUTATION

A Greedy Approach

Muthu Venkitasubramaniam
(joint work with abhi shelat)
Secure Computation

• **[STEP 1]** Compile $f$ to
  – Boolean Circuits, Arithmetic Circuits, ORAM

• **[STEP 2]** Generic Approaches
  – Yao based, GMW based, Information Theoretic based

• How to determine which approach
  – Depends on size, latency, bandwidth, etc

• Sometimes specific approaches are better
  – PSI (great primitive, several applications)
THIS TALK

A New Algorithmic Approach for Designing Secure Computation Protocols
SECURE COMPUTATION OF MEDIAN
Aggarwal, Mishra, Pinkas (Eurocrypt `04, JOC `10)

Secure comparison (e.g. a small circuit)

A finds median of $S_A$, call it $m_A$
B finds median of $S_B$, call it $m_B$

$m_A < m_B$

A deletes $x \in S_A$ s.t. $x < m_A$.
B deletes $x \in S_B$ s.t. $x \geq m_B$.

YES

A deletes $x \in S_A$ s.t. $x \geq m_A$.
B deletes $x \in S_B$ s.t. $x < m_B$.

NO

Slides borrowed from Benny Pinkas
WALK THROUGH

Median found!!

Slides borrowed from Benny Pinkas
PROVING Semi-honest SECURITY

What functionality do we need to compute securely?

MILLIONAIRE

Slides borrowed from Benny Pinkas
WHAT ELSE CAN WE COMPUTE USING MILLIONAIRE?

- Convex Hull
- Minimum Spanning Tree [BS05]
- Unit Job Scheduling
- Single Source All Destination Shortest Paths [BS05]
- Set Cover / Vertex Cover / Max Cover*
## RESULTS
(communication complexity)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Our Work (O)</th>
<th>Circuit (Ω)</th>
<th>ORAM (Ω)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convex Hull</td>
<td>$O\ell$</td>
<td>$I \log(I)\ell$</td>
<td>$I \log^3(I)\ell$</td>
</tr>
<tr>
<td>MST</td>
<td>$V\ell$</td>
<td>$(V\alpha(V))^2\ell$</td>
<td>$V\alpha(V) \log^3(V)\ell$</td>
</tr>
<tr>
<td>Unit Job Scheduling</td>
<td>$O\ell$</td>
<td>$I^2\ell$</td>
<td>$I \log^3(I)\ell$</td>
</tr>
<tr>
<td>Single Src ADSP</td>
<td>$V\ell$</td>
<td>$E^2\ell$</td>
<td>$E \log^3(E)\ell$</td>
</tr>
<tr>
<td>Cover Problems</td>
<td>$O\ell$</td>
<td>$I_s^2\ell$</td>
<td>$I_s \log^3(I_s)\ell$</td>
</tr>
</tbody>
</table>

$I$ - input size  
$O$ - output size  
$\ell$ - integer representation  
$\alpha()$ - Inverse Ackerman fn.  
$V$ - #Vertices  
$E$ - #Edges
WHAT PARADIGM ABSTRACTS THESE ALGORITHMS?

- Convex Hull
- Minimum Spanning Tree
- Unit Job Scheduling
- Single Source All Destination Shortest Paths
- Set Cover / Vertex Cover / Max Cover*
Greedy Algorithms

• Iteratively find the (local) optimal choice and hope for the best

• Leads to optimal in many problems
  – Convex Hull: Jarvis March
  – MST: Kruskal, Prim’s algorithm
  – Job Scheduling (many variants)
  – Shortest Path: Dijkstra
  – Set Cover: Submodular Function Approximation
Our Greedy-Millionaire Framework

A function $f$ is *secure greedy compatible* if there exists a function $F$ such that:

1. **UNIQUE SOLUTION**
   Given inputs $U$ and $V$ of Alice and Bob $f(U, V)$ is unique

2. **UNIQUE ORDER** – If $f(U, V) = (c_1, \ldots, c_l)$, then
   
   $F(\text{ }, U \quad V) = c_1$ and $F(c_i, U \quad V) = c_{i+1}$

3. **LOCAL UPDATABILITY**

   $F(c_i, U \quad V) = LT(F(c_i, U), F(c_i, V))$
Secure Greedy-Millionaire Algorithm

**Generic Iterative Secure Computation**

Alice Input: Distinct elements $U = \{u_1, \ldots, u_n\}$

Bob Input: Distinct elements $V = \{v_1, \ldots, v_n\}$

Output:

1. Alice initializes $(u_a, k_a) \leftarrow F(\bot, U)$ and Bob initializes $(v_b, k_b) \leftarrow F(\bot, V)$.
2. Repeat for $\ell(|U|, |V|)$ times:
   a. Alice and Bob execute the secure protocol $c_j \leftarrow LT_f((u_a, k_a), (v_b, k_b))$.
   b. Alice updates $(u_a, k_a) \leftarrow F(c_{\leq j}, U)$ and Bob updates $(v_b, k_b) \leftarrow F(c_{\leq j}, V)$.

**Generalized Compare**

Alice Input: Tuple $(u, x)$ with $k$-bit integer key $x$

Bob Input: Tuple $(v, y)$ $k$-bit integer key $y$

LT$_f$ Output: Return $u$ if $x > y$ and $v$ otherwise
Secure Greedy-Millionaire Algorithm

**Generic Iterative Secure Computation**

Alice Input: Distinct elements $U = \{u_1, \ldots, u_n\}$
Bob Input: Distinct elements $V = \{v_1, \ldots, v_n\}$
Output:

1. Alice initializes $(u_a, k_a) \leftarrow F(\bot, U)$ and Bob initializes $(v_b, k_b) \leftarrow F(\bot, V)$.
2. Repeat for $\ell(|U|, |V|)$ times:
   a. Alice and Bob execute the secure protocol $c_j \leftarrow LT_f((u_a, k_a), (v_b, k_b))$.
   b. Alice updates $(u_a, k_a) \leftarrow F(c_{\leq j}, U)$ and Bob updates $(v_b, k_b) \leftarrow F(c_{\leq j}, V)$.

**Correctness:**

$f(U, V) = (c_1, \ldots, c_l)$

$F(\bot, U \cup V) = c_1$ and $F(c_i, U \cup V) = c_{i+1}$

$F(c_i, U \cup V) = LT(F(c_i, U), F(c_i, V))$
**Secure Greedy-Millionaire Algorithm**

**Generic Iterative Secure Computation**

Alice Input: Distinct elements $U = \{u_1, \ldots, u_n\}$

Bob Input: Distinct elements $V = \{v_1, \ldots, v_n\}$

Output:

1. Alice initializes $(u_a, k_a) \leftarrow F(\bot, U)$ and Bob initializes $(v_b, k_b) \leftarrow F(\bot, V)$.
2. Repeat for $\ell(|U|, |V|)$ times:
   a. Alice and Bob execute the secure protocol $c_j \leftarrow LT_f((u_a, k_a), (v_b, k_b))$.
   b. Alice updates $(u_a, k_a) \leftarrow F(c_{\leq j}, U)$ and Bob updates $(v_b, k_b) \leftarrow F(c_{\leq j}, V)$.

**Simulation:**

Input $U$ and Output $(c_1, \ldots, c_l)$

Unique Solution and Unique Order

- Output of iteration $i$ is $c_i$
A set system \((S,I)\) where \(S\) is a finite set, and \(I\) a nonempty family of subsets of \(S\) is a matroid if

**Hereditary Property:**
If \(B \in I\) and \(A \subseteq B\), then \(A \in I\).

**Exchange Property:**
If \(A,B \in I\) and \(|A| < |B|\), then there exists \(x\) in \(B \setminus A\) such that \(A \cup \{x\}\) is in \(I\).

**Weighted Matroid:** a weight function \(w : S \to \mathbb{R}^+\)

**THEOREM:** The greedy algorithm finds maximal independent set with minimum cost.
Examples of Matroids

Example 1: Let $M$ be a matrix. Let $S$ be the set of rows of $M$ and $I = \{ A \mid A \subseteq S, A \text{ is linearly independent} \}$

Example 2: Let $G = (V,E)$ be an undirected graph. Choose $S = E$ and $I = \{ A \mid H = (V,A) \text{ is an induced subgraph of } G \text{ such that } H \text{ is a forest} \}$
Greedy Algorithm for Matroids

Greedy ALGORITHM \(((S, I), w)\)

1. Set A to be empty
2. For each \(x\) in \(S\) taken in monotonically decreasing order do
   - If \(A \cup \{x\}\) in \(I\) then set \(A = A \cup \{x\}\)
3. Return \(A\)

Matroids are secure-greedy-compatible if

- **UNIQUE SOLUTION and UNIQUE ORDER**: Assume weights are distinct
- **LOCAL UPDATABILITY**: If membership in \(I\) can be done locally
CAN WE ACHIEVE MALICIOUS SECURITY?

• Unfortunately NOT because we iteratively reveal answer
  – Adversary can adaptively abort in the middle of the computation
SECURE MEDIAN COMPUTATION

Slides borrowed from Benny Pinkas
PROVING MALICIOUS SECURITY
CAN WE ACHIEVE MALICIOUS SECURITY?

• Unfortunately NOT because we iteratively reveal answer
  – Adversary can adaptively abort in the middle of the computation

NEXT BEST THING: Covert Security
Covert Security

Definition (Informal): [Aumann-Lindell\textsuperscript{\textasciitilde10}] A protocol \( \pi \) is said to compute \( f \) in the presence of covert adversaries with \( \varepsilon \)-deterrence if for every PPT Bob and distinguisher D there exists negligible function \( \mu() \) such that

\[
\Pr[\text{Alice outputs "Bob is corrupt"}] \geq \varepsilon \text{ (Distinguishing gap)} - \mu(k)
\]

IDEA: After output is revealed, prove that in each step, the greedy update was correctly done
Achieving Covert Security

• Adaptively select inputs
  – Use commitments

• Failure to follow greedy update
  – Use inputs output of order
  – Missing inputs, i.e. use only a subset of inputs committed

• IDEA: Use signatures and consistency checks
Secure Greedy Covert Protocol – High-Level

• **Input Commitment Phase:** Using an extractable commitment Alice and Bob commit to their inputs.
  – Alice and Bob additional share verification keys for a signature scheme

• **Secure Computation Phase:** As before iteratively reveal answers. Additionally outputs are signed by both parties.

• **Consistency Check Phase:** A short protocol that shows each input committed in the first phase used correctly
Consistency Checks

For every input commitment prove that the value contained is either

– In the output, or
– Not part of the optimal solution

**Convex Hull:** Show that a particular point is not on the hull.
Consistency Checks - Matroids

Let \((S,I)\) be a weighted matroid set system.

**Question:** How do you show that particular element is not part of minimum cost maximal independent set?

**MST:** Show that a particular edge does not decrease cost of tree
Show that in the cycle this edge is of maximum cost
Consistency Checks - Matroids

Let \((S,I)\) be a weighted matroid set system.

**Question:** How do you show that particular element is not part of minimum cost maximal independent set?

**Matroid:** Show that a particular element does not decrease cost of independent set.

Show that in the *fundamental cycle* this element is of maximum cost

Proof Length: \(O(|B|)\) per input
Efficient Consistency Check - MST

- Naïve approach: Cost $O(|V|)$ proof length per edge
- Improve to $O(\log n)$ per edge
- **IDEA:** UNION-FIND data structure
  - Using the pointer data structure: FIND operations cost $O(\log n)$ and Union operations cost $O(1)$
  - Use signatures to get union and find operations attested
- If we use Tarjan’s Union-Find, we can improve to $O(\alpha(n))$ where $\alpha$ is the inverse ackerman function.
## RESULTS FOR COVERT SECURITY

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Our Work (O)</th>
<th>Circuit (Ω)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COVERT</td>
<td>O\lplus I\l</td>
<td>I log(I)\l</td>
</tr>
<tr>
<td>Convex Hull</td>
<td>V log(V)\l</td>
<td>(Vα(V))\l^2</td>
</tr>
<tr>
<td>MST</td>
<td>O\lplus I\l</td>
<td>I^2\l</td>
</tr>
<tr>
<td>Unit Job Scheduling</td>
<td>V\lplus E\l</td>
<td>E^2\l</td>
</tr>
<tr>
<td>Single Src ADSP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| O - output size               | α() - Inverse Ackerman fn. |
| I - input size                | V - #Vertices |
| \ell - integer representation | E - #Edges |
CONCLUSION

• Leverage techniques from algorithms to improve secure computation
• Secure computation using only comparison operations
• OPEN PROBLEM 1: What about other primitives?
• OPEN PROBLEM 2: What about other paradigms?
  – Dynamic Programming
  – Randomized Algorithms