Data Confidentiality in Collaborative Computing

Mikhail Atallah

Department of Computer Science
Purdue University
Collaborators

- **Ph.D. students:**
  - Marina Blanton (exp grad ‘07)
  - Keith Frikken (grad ‘05)
  - Jiangtao Li (grad ‘06)

- **Profs:**
  - Chris Clifton (CS)
  - Vinayak Deshpande (Mgmt)
  - Leroy Schwarz (Mgmt)
The most useful data is scattered and hidden

- Data distributed among many parties
- Could be used to compute useful outputs (of benefit to all parties)
- Online collaborative computing looks like a “win-win”, yet ...
- Huge potential benefits go unrealized
- Reason: Reluctance to share information
Reluctance to Share Info

• Proprietary info, could help competition
  – Reveal corporate strategy, performance
• Fear of loss of control
  – Further dissemination, misuse
• Fear of embarrassment, lawsuits
• May be illegal to share
• Trusted counterpart but with poor security
Securely Computing $f(X,Y)$

- **Inputs:**
  - Data X (with Bob), data Y (with Alice)
- **Outputs:**
  - Alice or Bob (or both) learn $f(X,Y)$
Secure Multiparty Computation

- SMC: Protocols for computing with data without learning it
- Computed answers are of same quality as if information had been fully shared
- Nothing is revealed other than the agreed upon computed answers
- No use of trusted third party
SMC (cont’d)

- Yao (1982): \{X \leq Y\}
- Goldwasser, Goldreich, Micali, ...
- General results
  - Deep and elegant, but complex and slow
  - Limited practicality
- Practical solutions for specific problems
- Broaden framework
Potential Benefits ...

- Confidentiality-preserving collaborations
- Use even with trusted counterparts
  - Better security ("defense in depth")
  - Less disastrous if counterpart suffers from break-in, spy-ware, insider misbehavior, ...
  - Lower liability (lower insurance rates)
- May be the only legal way to collaborate
  - Anti-trust, HIPAA, Gramm-Leach-Bliley, ...
... and Difficulties

- Designing practical solutions
  - Specific problems; "moderately untrusted" 3rd party; trade some security; ...
- Quality of inputs
  - ZK proofs of well-formedness (e.g., \{0,1\})
  - Easier to lie with impunity when no one learns the inputs you provide
  - A participant could gain by lying in competitive situations
- Inverse optimization
Quality of Inputs

• The inputs are 3rd-party certified
  – Off-line certification
  – Digital credentials
  – “Usage rules” for credentials

• Participants incentivized to provide truthful inputs
  – Cannot gain by lying
Variant: Outsourcing

- Weak client has all the data
- Powerful server does all the expensive computing
  - Deliberately asymmetric protocols
- Security: Server learns neither input nor output
- Detection of cheating by server
  - E.g., server returns some random values
Models of Participants

• Honest-but-curious
  – Follow protocol
  – Compute all information possible from protocol transcript

• Malicious
  – Can arbitrarily deviate from protocol

• Rational, selfish
  – Deviate if gain (utility function)
Examples of Problems

- Access control, trust negotiations
- Approximate pattern matching & sequence comparisons
- Contract negotiations
- Collaborative benchmarking, forecasting
- Location-dependent query processing
- Credit checking
- Supply chain negotiations
- Data mining (partitioned data)
- Electronic surveillance
- Intrusion detection
- Vulnerability assessment
- Biometric comparisons
- Game theory
Hiding Intermediate Values

• Additive splitting
  - \( x = x' + x'' \), Alice has \( x' \), Bob has \( x'' \)

• Encoder / Evaluator
  - Alice uses randoms to encode the possible values \( x \) can have, Bob learns the random corresponding to \( x \) but cannot tell what it encodes
Hiding Intermediate ... (cont’d)

- Compute with encrypted data, e.g.
- Homomorphic encryption
  - 2-key (distinct encrypt & decrypt keys)
  - $E_A(x) \cdot E_A(y) = E_A(x+y)$
  - Semantically secure: Having $E_A(x)$ and $E_A(y)$ do not reveal whether $x=y$
Example: Blind-and-Permute

• Input: \( c_1, c_2, \ldots, c_n \) additively split between Alice and Bob: \( c_i = a_i + b_i \) where Alice has \( a_i \), Bob has \( b_i \)

• Output: A randomly permuted version of the input (still additively split) s.t. neither side knows the random permutation
Blind-and-Permute Protocol

1. A sends to B: $E_A$ and $E_A(a_1), \ldots, E_A(a_n)$
2. B computes $E_A(a_i) \cdot E_A(r_i) = E_A(a_i + r_i)$
3. B applies $\pi_B$ to $E_A(a_1 + r_1), \ldots, E_A(a_n + r_n)$ and sends the result to A
4. B applies $\pi_B$ to $b_1 - r_1, \ldots, b_n - r_n$
5. Repeat the above with the roles of A and B interchanged
Dynamic Programming for Comparing Bio-Sequences

- $M(i,j)$ is the minimum in cost of transform the prefix of $X$ of length $i$ into the prefix of $Y$ of length $j$

$$M(i,j) = \min\left\{ \begin{array}{l} M(i-1,j-1) + S(\lambda_i, \mu_j) \\ M(i-1,j) + D(\lambda_i) \\ M(i,j-1) + I(\mu_j) \end{array} \right.$$

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Insertion Cost</th>
<th>Deletion Cost</th>
<th>Substitution Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 0 \infty \infty \infty</td>
<td>A 1</td>
<td>A 1</td>
</tr>
<tr>
<td>C \infty 0 \infty \infty</td>
<td>C 1</td>
<td>C 1</td>
</tr>
<tr>
<td>T \infty \infty 0 \infty</td>
<td>T 1</td>
<td>T 1</td>
</tr>
<tr>
<td>G \infty \infty \infty 0</td>
<td>G 1</td>
<td>G 1</td>
</tr>
</tbody>
</table>

http://www.cerias.purdue.edu
Correlated Action Selection

- \((p_1, a_1, b_1), \ldots, (p_n, a_n, b_n)\)
- Prob \(p_j\) of choosing index \(j\)
- A (resp., B) learns only \(a_j\) (\(b_j\))
- Correlated equilibrium
- Implementation with third-party mediator
- Question: Is mediator needed?
Correlated Action Selection (cont’d)

• Protocols without mediator exist
• Dodis et al. (Crypto ’00)
  – Uniform distribution
• Teague (FC ’04)
  – Arbitrary distribution, exponential complexity
• Our result: Arbitrary distribution with polynomial complexity
Correlated Action Selection (cont’d)

- A sends to B: \(E_A(p_j), E_A(a_j), E_A(b_j)\)
- B permutes the n triplets and computes \(E_A(Q_j) = E_A(p_1) \times \ldots \times E_A(p_j) = E_A(p_1 + \ldots + p_j)\)
- B computes \(E_A(Q_j - r_j), E_A(a_j - r'_j), E_A(b_j - r''_j)\), then permutes and sends to A the n triplets so obtained
- A and B select an additively split random \(r (= r_A + r_B)\) and “locate” \(r\) in the additively split list of \(Q_j\)s
Access Control

- Access control decisions are often based on requester *characteristics* rather than identity
  - Access policy stated in terms of attributes

- Digital credentials, e.g.,
  - Citizenship, age, physical condition (disabilities), employment (government, healthcare, FEMA, etc), credit status, group membership (AAA, AARP, ...), security clearance, ...
Access Control (cont’d)

• Treat credentials as sensitive
  – Better individual privacy
  – Better security

• Treat access policies as sensitive
  – Hide business strategy (fewer unwelcome imitators)
  – Less “gaming”
• $M =$ message ; $P =$ Policy ; $C, S =$ credentials
  – Credential sets $C$ and $S$ are issued off-line, and can have their own “use policies”

• Client gets $M$ iff usable $C_j$’s satisfy policy $P$

• Cannot use a trusted third party
Solution Requirements

- Server does not learn whether client got access or not
- Server does not learn anything about client’s credentials, and vice-versa
- Client learns neither server’s policy structure nor which credentials caused her to gain access
- No off-line probing (e.g., by requesting an M once and then trying various subsets of credentials)
Credentials

- Generated by certificate authority (CA), using Identity Based Encryption
- E.g., issuing Alice a student credential:
  - Use Identity Based Encryption with ID = Alice||student
  - Credential = private key corresponding to ID
- Simple example of credential usage:
  - Send Alice M encrypted with public key for ID
  - Alice can decrypt only with a student credential
  - Server does not learn whether Alice is a student or not
Policy

- A Boolean function $p_M(x_1, \ldots, x_n)$
  - $x_i$ corresponds to attribute $attr_i$
- Policy is satisfied iff
  - $p_M(x_1, \ldots, x_n) = 1$ where $x_i$ is 1 iff there is a usable credential in C for attribute $attr_i$
- E.g.,
  - Alice is a senior citizen and has low income
  - Policy=$\text{(disability} \lor \text{senior-citizen}) \land \text{low-income}$
  - $\text{Policy} = (x_1 \lor x_2) \land x_3 = (0 \lor 1) \land 1 = 1$
Ideas in Solution

- **Phase 1: Credential and Attribute Hiding**
  - For each $attr_i$, server generates 2 randoms $r_i[0], r_i[1]$
  - Client learns $n$ values $k_1, k_2, \ldots, k_n$ s.t. $k_i = r_i[1]$ if she has a credential for $attr_i$, otherwise $k_i = r_i[0]$

- **Phase 2: Blinded Policy Evaluation**
  - Client’s inputs are the above $k_1, k_2, \ldots, k_n$
  - Server’s input now includes the $n$ pairs $r_i[0], r_i[1]$
  - Client obtains $M$ if and only if $p_M(x_1, \ldots, x_n) = 1$
Concluding Remarks

• Promising area (both research and potential practical impact)
• Need more implementations and software tools
  – FAIRPLAY (Malkhi et.al.)
• Currently impractical solutions will become practical