Secure Multiparty Computation – Basic Cryptographic Methods

Li Xiong

CS573 Data Privacy and Security
Want to know if both parties are interested in each other.
But… Do not want to reveal unrequited love.

Input = 1 :   I love you
Input = 0:   I love you  ... as a friend
Must compute $F(X, Y) = X \land Y$, giving $F(X, Y)$ to both players.

Can we reveal the answer without revealing the inputs?
Pearl wants to know whether she has more toys than Gersh. Doesn’t want to tell Gersh anything.

Gersh is willing for Pearl to find out who has more toys, Doesn’t want Pearl to know how many toys he has.

Can we give Pearl the information she wants, and nothing else, without giving Gersh any information at all?
Secure Multiparty Computation

- A set of parties with private inputs
- Parties wish to jointly compute a function of their inputs so that certain security properties (like privacy and correctness) are preserved
- Properties must be ensured even if some of the parties maliciously attack the protocol
- Examples
  - Secure elections
  - Auctions
  - Privacy preserving data mining
  - ...
Application to Private Data Mining

The setting:
- Data is **distributed** at different sites
- These sites may be third parties (e.g., hospitals, government bodies) or may be the individual him or herself

The aim:
- Compute the data mining algorithm on the data so that **nothing but the output is learned**
- Privacy ≠ Security (why?)
Privacy and Secure Computation

- Privacy ≠ Security
  - Secure computation only deals with the process of computing the function
  - It does not ask whether or not the function should be computed
- A two-stage process:
  - Decide that the function/algorithm should be computed – an issue of privacy
  - Apply secure computation techniques to compute it securely – security
Outline

- Secure multiparty computation
  - Problem and security definitions
  - Feasibility results for secure computation
  - Basic cryptographic tools and general constructions
Heuristic Approach to Security

1. Build a protocol
2. Try to break the protocol
3. Fix the break
4. Return to (2)
Another Heuristic Tactic

- Design a protocol
- Provide a list of attacks that (provably) cannot be carried out on the protocol
- Reason that the list is complete

Problem: often, the list is not complete…
A Rigorous Approach

- Provide an exact problem definition
  - Adversarial power
  - Network model
  - Meaning of security
- Prove that the protocol is secure
Secure Multiparty Computation

- A set of parties with private inputs wish to compute some joint function of their inputs.
- Parties wish to preserve some security properties. e.g., privacy and correctness.
  - Example: secure election protocol
- Security must be preserved in the face of adversarial behavior by some of the participants, or by an external party.
Defining Security

- Components of ANY security definition
  - Adversarial power
  - Network model
    - Type of network
    - Existence of trusted help
    - Stand-alone versus composition
  - Security guarantees

- It is crucial that all the above are explicitly and clearly defined.
Security Requirements

Consider a secure auction (with secret bids):

- An adversary may wish to learn the bids of all parties – to prevent this, require privacy
- An adversary may wish to win with a lower bid than the highest – to prevent this, require correctness
Defining Security

- **Option 1:** analyze security concerns for each specific problem
  - **Auctions:** privacy and correctness
  - **Contract signing:** fairness

- **Problems:**
  - How do we know that all concerns are covered?
  - Definitions are application dependent and need to be redefined from scratch for each task
Defining Security – Option 2

- The real/ideal model paradigm for defining security [GMW,GL,Be,MR,Ca]:
  - Ideal model: parties send inputs to a trusted party, who computes the function for them
  - Real model: parties run a real protocol with no trusted help

- A protocol is secure if any attack on a real protocol can be carried out in the ideal model

- Since no attacks can be carried out in the ideal model, security is implied
The Real Model

\[ x \quad \text{Protocol output} \]

\[ y \quad \text{Protocol output} \]
The Ideal Model

\[ f_1(x,y) \quad f_2(x,y) \]

\[ f_1(x,y) \quad f_2(x,y) \]
The Security Definition:

For every real adversary $A$, there exists an adversary $S$.

Protocol interaction

Trusted party

REAL

IDEAL
Properties of the Definition

- **Privacy:**
  - The ideal-model adversary cannot learn more about the honest party’s input than what is revealed by the function output.
  - Thus, the same is true of the real-model adversary.

- **Correctness:**
  - In the ideal model, the function is always computed correctly.
  - Thus, the same is true in the real-model.

- **Others:**
  - For example, fairness, independence of inputs.
Why This Approach?

- General – it captures all applications
  - The specifics of an application are defined by its functionality, security is defined as above
- The security guarantees achieved are easily understood (because the ideal model is easily understood)
- We can be confident that we did not “miss” any security requirements
Adversary Model

- **Computational power:**
  - Probabilistic polynomial-time versus all-powerful

- **Adversarial behaviour:**
  - Semi-honest: follows protocol instructions
  - Malicious: arbitrary actions

- **Corruption behaviour**
  - Static: set of corrupted parties fixed at onset
  - Adaptive: can choose to corrupt parties at any time during computation

- **Number of corruptions**
  - Honest majority versus unlimited corruptions
Outline

- Secure multiparty computation
  - Defining security
  - Feasibility results for secure computation
  - Basic cryptographic tools and general constructions
Feasibility – A Fundamental Theorem

- Any multiparty functionality can be securely computed
  - For any number of corrupted parties: security with abort is achieved, assuming enhanced trapdoor permutations [Yao,GMW]
  - With an honest majority: full security is achieved, assume private channels only [BGW,CCD]
Outline

- Secure multiparty computation
  - Defining security
  - Feasibility results for secure computation
  - Basic cryptographic tools and general constructions
Public-key encryption

Let \((G,E,D)\) be a public-key encryption scheme

- \(G\) is a key-generation algorithm \((pk,sk) \leftarrow G\)
  - \(Pk\): public key
  - \(Sk\): secret key

- Terms
  - Plaintext: the original text, notated as \(m\)
  - Ciphertext: the encrypted text, notated as \(c\)

- Encryption: \(c = E_{pk}(m)\)
- Decryption: \(m = D_{sk}(c)\)
- Concept of one-way function: knowing \(c\), \(pk\), and the function \(E_{pk}\), it is still computationally intractable to find \(m\).

*Different implementations available, e.g. RSA*
Construction paradigms

- Passively-secure computation for two-parties
  - Use **oblivious transfer** to securely select a value

- Passively-secure computation with shares
  - Use **secret sharing scheme** such that data can be reconstructed from some shares

- From passively-secure protocols to actively-secure protocols
  - Use **zero-knowledge proofs** to force parties to behave in a way consistent with the passively-secure protocol
1-out-of-2 Oblivious Transfer (OT)

- **Inputs**
  - Sender has two messages $m_0$ and $m_1$
  - Receiver has a single bit $\sigma \in \{0, 1\}$

- **Outputs**
  - Sender receives nothing
  - Receiver obtains $m_\sigma$ and learns nothing of $m_{1-\sigma}$
Semi-Honest OT

Let \((G, E, D)\) be a public-key encryption scheme

- \(G\) is a key-generation algorithm \((pk, sk) \leftarrow G\)
- Encryption: \(c = E_{pk}(m)\)
- Decryption: \(m = D_{sk}(c)\)

Assume that a public-key can be sampled without knowledge of its secret key:

- Oblivious key generation: \(pk \leftarrow OG\)
- El-Gamal encryption has this property
Semi-Honest OT

Protocol for Oblivious Transfer

- Receiver (with input $\sigma$):
  - Receiver chooses one key-pair ($pk, sk$) and one public-key $pk'$ (oblivious of secret-key).
  - Receiver sets $pk_\sigma = pk$, $pk_{1-\sigma} = pk'$
  - Note: receiver can decrypt for $pk_\sigma$ but not for $pk_{1-\sigma}$
  - Receiver sends $pk_0, pk_1$ to sender

- Sender (with input $m_0, m_1$):
  - Sends receiver $c_0 = E_{pk_0}(m_0)$, $c_1 = E_{pk_1}(m_1)$

- Receiver:
  - Decrypts $c_\sigma$ using $sk$ and obtains $m_\sigma$. 
Intuition:

- Sender’s view consists only of two public keys $pk_0$ and $pk_1$. Therefore, it doesn’t learn anything about that value of $\sigma$.
- The receiver only knows one secret-key and so can only learn one message.

Note: this assumes semi-honest behavior. A malicious receiver can choose two keys together with their secret keys.
Generalization

- Can define 1-out-of-k oblivious transfer

- Protocol remains the same:
  - Choose \( k-1 \) public keys for which the secret key is unknown
  - Choose 1 public-key and secret-key pair
General GMW Construction

- For simplicity – consider two-party case
- Let $f$ be the function that the parties wish to compute
- Represent $f$ as an arithmetic circuit with addition and multiplication gates
- Aim – compute gate-by-gate, revealing only random shares each time
Random Shares Paradigm

Let $a$ be some value:
- Party 1 holds a random value $a_1$
- Party 2 holds $a + a_1$
- Note that without knowing $a_1$, $a + a_1$ is just a random value revealing nothing of $a$.
- We say that the parties hold random shares of $a$.

The computation will be such that all intermediate values are random shares (and so they reveal nothing).
Circuit Computation

- **Stage 1**: each party randomly shares its input with the other party
- **Stage 2**: compute gates of circuit as follows
  - Given random shares to the input wires, compute random shares of the output wires
- **Stage 3**: combine shares of the output wires in order to obtain actual output
Addition Gates

- Input wires to gate have values $a$ and $b$:
  - Party 1 has shares $a_1$ and $b_1$
  - Party 2 has shares $a_2$ and $b_2$
  - Note: $a_1 + a_2 = a$ and $b_1 + b_2 = b$

- To compute random shares of output $c = a + b$
  - Party 1 locally computes $c_1 = a_1 + b_1$
  - Party 2 locally computes $c_2 = a_2 + b_2$
  - Note: $c_1 + c_2 = a_1 + a_2 + b_1 + b_2 = a + b = c$
Multiplication Gates

- Input wires to gate have values $a$ and $b$:
  - Party 1 has shares $a_1$ and $b_1$
  - Party 2 has shares $a_2$ and $b_2$
  - Wish to compute $c = ab = (a_1+a_2)(b_1+b_2)$

- Party 1 knows its concrete share values.
- Party 2’s values are unknown to Party 1, but there are only 4 possibilities (depending on correspondence to 00,01,10,11)
Multiplication (cont)

- Party 1 prepares a table as follows:
  - Row 1 corresponds to Party 2’s input 00
  - Row 2 corresponds to Party 2’s input 01
  - Row 3 corresponds to Party 2’s input 10
  - Row 4 corresponds to Party 2’s input 11

- Let $r$ be a random bit chosen by Party 1:
  - Row 1 contains the value $a \cdot b + r$ when $a_2=0, b_2=0$
  - Row 2 contains the value $a \cdot b + r$ when $a_2=0, b_2=1$
  - Row 3 contains the value $a \cdot b + r$ when $a_2=1, b_2=0$
  - Row 4 contains the value $a \cdot b + r$ when $a_2=1, b_2=1$
### Concrete Example

- Assume: $a_1=0$, $b_1=1$
- Assume: $r=1$

<table>
<thead>
<tr>
<th>Row</th>
<th>Party 2’s shares</th>
<th>Output value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_2=0, b_2=0$</td>
<td>$(0+0) \cdot (1+0) + 1 = 1$</td>
</tr>
<tr>
<td>2</td>
<td>$a_2=0, b_2=1$</td>
<td>$(0+0) \cdot (1+1) + 1 = 1$</td>
</tr>
<tr>
<td>3</td>
<td>$a_2=1, b_2=0$</td>
<td>$(0+1) \cdot (1+0) + 1 = 0$</td>
</tr>
<tr>
<td>4</td>
<td>$a_2=1, b_2=1$</td>
<td>$(0+1) \cdot (1+1) + 1 = 1$</td>
</tr>
</tbody>
</table>
The Gate Protocol

- The parties run a 1-out-of-4 oblivious transfer protocol.
- Party 1 plays the sender: message $i$ is row $i$ of the table.
- Party 2 plays the receiver: it inputs 1 if $a_2=0$ and $b_2=0$, 2 if $a_2=0$ and $b_2=1$, and so on...
- Output:
  - Party 2 receives $c_2=c+r$ – this is its output
  - Party 1 outputs $c_1=r$,
  - Note: $c_1$ and $c_2$ are random shares of $c$, as required
Summary

- By computing each gate these way, at the end the parties hold shares of the output wires.
- Function output generated by simply sending shares to each other.
Security

- Reduction to the oblivious transfer protocol
- Assuming security of the OT protocol, parties only see random values until the end. Therefore, simulation is straightforward.

- Note: correctness relies heavily on semi-honest behavior (otherwise can modify shares).
Outline

- Secure multiparty computation
  - Defining security
  - Feasibility results for secure computation
  - Basic cryptographic tools and general constructions
- Coming up
  - Applications in privacy preserving distributed data mining
  - Random response protocols
A real-world problem and some simple solutions

- Bob comes to Ron (a manager), with a complaint about a sensitive matter, asking Ron to keep his identity confidential.

- A few months later, Moshe (another manager) tells Ron that someone has complained to him, also with a confidentiality request, about the same matter.

- Ron and Moshe would like to determine whether the same person has complained to each of them without giving information to each other about their identities.

Comparing information without leaking it. Fagin et al, 1996
References

- Secure Multiparty Computation for Privacy-Preserving Data Mining, Pinkas, 2008
- Chapter 7: General Cryptographic Protocols (7.1 Overview), The Foundations of Cryptography, Volume 2, Oded Goldreich
  
  http://www.wisdom.weizmann.ac.il/~Eoded/foc-vol2.html
- Comparing information without leaking it. Fagin et al, 1996
Slides credits

- Tutorial on secure multi-party computation, Lindell
  www.cs.biu.ac.il/~lindell/research-statements/tutorial-secure-computation.ppt

- Introduction to secure multi-party computation, Vitaly Shmatikov, UT Austin
  www.cs.utexas.edu/~shmat/courses/cs380s_fall08/16smc.ppt