Tutorial on Secure Multi-Party Computation

Yehuda Lindell

IBM T.J.Watson

Outline

- Part 1:
  - A rigorous approach to security
  - Defining security
    - Network model, adversarial power
  - Feasibility results for secure computation
- Part 2:
  - General constructions

A Rigorous Approach

Heuristic Approach to Security

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

Heuristic Approach – Dangers

- Real adversaries won't tell you that they have broken the protocol
- You can never be really sure that the protocol is secure
- Compare to algorithms:
  - Inputs are not adversarial
  - Hackers will do anything to exploit a weakness – if one exists, it may well be found
  - Security cannot be checked empirically

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
  - Often by reduction to an assumed hard problem, like factoring large composites
- The history of computer security shows that the heuristic approach is likely to fail
  - Security is very tricky and often anti-intuitive

Secure Computation

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.

Protocols and Functions

- Cryptography aims for the following (regarding privacy):
  - A secure protocol must reveal no more information than the output of the function itself
  - That is, the process of protocol computation reveals nothing.
- Cryptography does not deal with the question of whether or not the function reveals much information
  - E.g., mean of two parties' salaries
- Deciding which functions to compute is a different challenge that must also be addressed in the context of privacy preserving data mining.

Defining Security

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**.

Vague Definitions

- If the network and adversarial model are not fully defined, then:
  - Secure protocols can be “broken” because they were proven in an unreasonable setting
  - If the adversary is unknown, then we can only reason about security very informally (this is a very common mistake)
  - It is not clear what is and what is not protected against. (It may be impossible to protect against everything – but we must be up front about it.)

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
  - But, the adversary may also wish to ensure that it always gives the highest bid – to prevent this, require INDEPENDENCE OF INPUTS

Defining Security

- Option 1:
  - Analyze security concerns for each specific problem
    - Auctions: as in previous slide
    - Elections: privacy and correctness only (?)

- Problems:
  - How do we know that all concerns are covered?
  - Definitions are application dependent (need to redefine each time).

Defining Security – Option 2

- The real/ideal model paradigm:
  - Ideal model: parties send inputs to a trusted party, who computes the function and sends the outputs.
  - Real model: parties run a real protocol with no trusted help.

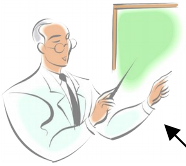
- Informally: a protocol is secure if any attack on a real protocol can be carried out (or simulated) in the ideal model.
- Since essentially **no** attacks can be carried out in the ideal model, security is implied.

The Security Definition:

adversary **A**

≈

adversary **S**



Computational Indistinguishability: **every** probabilistic polynomial-time observer that receives the input/output distribution of the honest parties and the adversary, outputs **1** upon receiving the distribution generated in IDEAL with negligibly close probability to when it is generated in REAL.

REAL

IDEAL

Meaning of the Definition

- Interpretation 1:
  - Security in the ideal model is absolute. Since no attacks are possible in the ideal model, we obtain that the same is also true of the real model.
- Interpretation 2:
  - Anything that an adversary could have learned/done in the real model, it could have also learned/done in the ideal model.
  - Note: real and ideal adversaries have same complexity.

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.
  - Otherwise, the REAL and IDEAL could be easily distinguished.
- Correctness:
  - In the ideal model, the function is always computed correctly.
  - Thus, the same is true in the real-model.
  - Otherwise, the REAL and IDEAL could be easily distinguished.
- Others:
  - For example, 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.

Detailed Definition

Components

- The real model
  - Adversarial power
  - Network model
- The ideal model
  - Adversarial power
  - The trusted party

- We present a definition for the stand-alone model (one protocol execution only)

The Real Model

- The Adversary
  - Probabilistic polynomial-time with auxiliary input (non-uniform model of computation)
  - Malicious – arbitrary actions
  - Static and adaptive variations
    - Static: set of corrupted parties fixed at onset
    - Adaptive: can choose to corrupt parties at any time during computation. Upon corruption receives its view (erasures versus no erasures)
  - Unlimited number of corrupted parties
  - Without loss of generality, it outputs its view in the protocol execution

The Real Model

- The Network
  - Asynchronous: messages can be delayed arbitrarily
    - Adversary controls message delivery
    - Non-issue for two-party case
  - Authenticated channels: can be achieved using a public-key infrastructure of digital signatures
- Honest Parties
  - Work according to protocol and output as instructed

The Ideal Model

- The Trusted Party:
  - Defined by any probabilistic polynomial-time Turing machine – this machine defines the functionality.
  - Trusted party linked to all participants via perfectly private and authenticated channels
  - Upon receiving an input from a party, trusted party runs the machine
  - If there is an output, it sends it to the designated party.
  - Continue as above
- This is more general than secure function evaluation.

The Ideal Model

- The Adversary
  - Probabilistic polynomial-time with auxiliary input (non-uniform model of computation)
  - Malicious – can choose any inputs for corrupted parties, based on initial inputs
  - Static and adaptive variations
    - Static: set of corrupted parties fixed at onset
    - Adaptive: can choose to corrupt parties at any time during computation
  - Unlimited number of corrupted parties
  - Outputs whatever it wishes

The Ideal Model

- Honest Parties
  - Send inputs to the trusted party and output whatever they receive back

Fairness & Guaranteed Output

- The above definition implies guaranteed output delivery and fairness
- But, fairness and guaranteed output delivery cannot be fully achieved when there is no honest majority
  - Our aim is to obtain the maximum security that is possible!
- One solution:
  - Allow the adversary to “instruct” the trusted party which parties should and should not get output

Modified Ideal Model

- Parties send their inputs to the trusted party
- If trusted party should send output to an honest party
  - Trusted party notifies the adversary
  - Adversary authorizes delivery via a special message (note: adversary decides when, if ever, outputs are received by honest parties)

The Security Definition

- A protocol Π securely computes a function **f** if:
  - For every real-model adversary **A**, there exists an ideal-model adversary **S**, such that for every set of inputs
  - the result of a real execution of Π with **A** is computationally indistinguishable from the result of an ideal execution with **S** (where the trusted party computes **f**).
- The result of an execution is defined by the output vector of the honest parties and adversary

Semi-Honest Adversaries

- Parties follow protocol exactly
- For secure function evaluation, ideal/real definition is equivalent to a simpler simulation-based definition
  - For every adversary, there exists a simulator so that the adversary's view in a real computation can be generated just given the output
  - Warning: indistinguishability should hold when comparing the joint distribution over the adversary and honest parties' outputs.

More Definitions

- There are numerous ways to define the real model, regarding both the adversary and network.
- The main thing is to realistically (and conservatively) model the real world scenario and adversarial threats.

Alternative Security Definitions

Alternative Definitions

- Ideal/real paradigm is very powerful and provides strong guarantees
- Sometimes, efficient protocols cannot achieve these strong guarantees
- Furthermore, sometimes these strong guarantees are not all needed
- Conclusion: sometimes we may wish to follow the alternative (first) approach

Alternative Approach

- Define specific requirements for a given task
- Note: this has no effect on other parts of definition (network model, adversarial behavior etc.) – same requirements remain
- Warning: it can take a long time until the “correct” definition is achieved (e.g., history of encryption).
- Nevertheless, it should not be ruled out

An Example: PIR

- PIR = Private Information Retrieval
- Aim: allow a client to query a database without the server learning what the query is.
- Definition:
  - Correctness: when the server is semi-honest, the client always obtains the correct result
  - Privacy: the view of any (even malicious) server when the client's query is i is indistinguishable from its view when the client's query is j.
- Sometimes, data privacy is also required (i.e., client should learn its query and nothing else).

Defining Functionalities

Defining Functionalities

- Given this framework, appropriate functionalities must still be defined for tasks of interest
- The functionality should be simple:
  - E.g., secure function evaluation – trusted party computes function and hands outputs to parties
  - Sometimes, we may wish to "dirty up" the functionality in order to increase efficiency (i.e., trusted party will leak more information).
  - But, we shouldn't go too far

Defining Functionalities

- Distributed data mining:
  - Each party sends its database to the trusted party
  - Trusted party gathers the union of all databases, runs the data mining algorithm and sends the output

- Note: it is clear what information is revealed

Some Feasibility Results

Computational Setting

- Any two-party function can be securely computed in the semi-honest adversarial model [Yao]
- Any multiparty function can be securely computed in the malicious model, for any number of corrupted parties [GMW]
- Remarks:
  - Above results assume the existence of trapdoor permutations
  - With an honest majority, fairness and guaranteed output delivery are achieved. Otherwise, not.
  - Model: static corruptions, authenticated channels, polynomial-time adversary, stand-alone…

Information Theoretic Setting

- Assuming a 2/3 honest majority, any multiparty function can be securely computed in the malicious model [BGW,CCD]
- Assuming a (regular) honest majority and a broadcast channel, any multiparty function can be securely computed in the malicious model [RB]
- Remarks:
  - Above results do not assume any complexity assumptions
  - Model: adaptive corruptions, perfectly private and authenticated channels, unbounded adversary, stand-alone…

Interpretation

- In the models described above, **any distributed task can be securely computed.**
- These are fundamental, and truly amazing, results.

Other Results

- There is a rich literature that presents feasibility results (and efficient protocols) for many different models.
- We will not present a survey of these, however we will talk a little about composition…

Protocol Composition

- Many protocols run concurrently over an Internet-like setting
  - There are actually many types of composition
- Protocols may be designed maliciously to attack existing secure protocols
- Security in the stand-alone setting does not suffice

Feasibility

- Universal composability [Ca]: a security definition that provides strong guarantees under composition
- Feasibility – any multiparty function can be securely computed under universal composability:
  - Honest majority: without setup assumptions [Ca]
  - No honest majority: with a common reference string [CLOS]

Feasibility (cont.)

- If no common reference string (or other setup) is used, then there are broad impossibility results for some settings.
- Many questions of feasibility for the setting of composition are still open.
- Note: composition must be considered. Focus on the stand-alone should be for initial study only. However, it is not realistic.

Outline

- Part 1:
  - A rigorous approach to security
  - Defining security
    - Network model, adversarial power
  - Feasibility results for secure computation
- Part 2:
  - General constructions

The GMW Paradigm

- Construct a protocol for the semi-honest model
- "Compile it" to obtain a protocol that is secure for the malicious model
  - Compilation involves forcing the parties to follow the protocol
- It may be more efficient to work differently

Semi-Honest Construction

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 obtain m_σ 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 σ):
  - Receiver chooses one key-pair (pk,sk) and one public-key pk' (obliviously of secret-key).
  - Receiver sets $pk_{\sigma} = pk$, $pk_{1-\sigma} = pk'$
  - Note: receiver can decrypt for pk_{σ} but not for $pk_{1-\sigma}$
  - Receiver sends pk_0,pk_1 to sender
- Sender (with input m_0,m_1):
  - Sends receiver $c_0=E_{pk0}(m_0)$, $c_1=E_{pk1}(m_1)$
- Receiver:
  - Decrypts c_{σ} using sk and obtains m_{σ}.

Security Proof

- 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 σ.
  - The receiver only knows one secret-key and so can only learn one message
- Formally:
  - Sender's view is independent of receiver's input and so can easily be simulated (just give it 2 keys)
  - Receiver's view can be simulated by obtaining the output m and sending it $E_{pk0}(m), E_{pk1}(m)$.
- 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 (over GF[2]).
- 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: $r=1$

Row	Party 2's shares	Output value
1	$a_2=0, b_2=0$	$(0+0)\cdot(1+0)+1=1$
2	$a_2=0, b_2=1$	$(0+0)\cdot(1+1)+1=1$
3	$a_2=1, b_2=0$	$(0+1)\cdot(1+0)+1=0$
4	$a_2=1, b_2=1$	$(0+1)\cdot(1+1)+1=1$

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).

Conclusion

- Theorem: any functionality f can be securely computed in the semi-honest model.

Remark

- The semi-honest model is often used as a tool for obtaining security against malicious parties.
- In many (most?) settings, security against semi-honest adversaries does not suffice.
- In some settings, it may suffice.
  - One example: hospitals that wish to share data.

Malicious Adversaries

- The above protocol is not secure against malicious adversaries:
  - A malicious adversary may learn more than it should.
  - A malicious adversary can cause the honest party to receive incorrect output.
  - We need to be able to extract a malicious adversary's input and send it to the trusted party.

Obtaining Security

Three goals:

- Force the adversary to use a fixed input
  - Furthermore, make it possible for the ideal-model simulator/adversary to extract this input.
- Force the adversary to use a uniform random tape
- Force the adversary to follow the protocol exactly (consistently with their fixed input and random tape)

Stage 1: Input Commitment

Preliminaries: bit commitment

- Commit Stage:
  - Committer has a bit σ
  - Receiver obtains a commitment string c
- Reveal Stage:
  - Committer sends a decommit message to receiver
  - Receiver uses decommit message and c to obtain σ

Bit Commitment

Security Properties:

- Binding: for every c, there exists only one value σ for which decommitment is accepted
  - Formally: the set of commitment strings to 0 is disjoint from the set of commitment strings to 1.
- Hiding: the receiver cannot distinguish a commitment string that is to 0 from a commitment string that is to 1.

Protocols

- Instructive example: committing to a bit using a telephone book
- Commitment using public-key encryption:
  - Committer chooses a key-pair (pk,sk).
  - Committer sends $(pk, c=E_{pk}(\sigma))$ to the receiver.
- Decommitment:
  - Committer sends the secret-key sk to the receiver
  - Receiver verifies that sk is associated with pk and decrypts, obtaining σ.
- Note: the commitment process is randomized. This is essential for any secure commitment. I.e., function of σ and coins r.

Proving Security

- Assumption: given pk, there is exactly one secret key sk that is associated with pk, and this can be efficiently determined (holds for RSA).
- Binding: encryption must have unique decryption. So given correct sk (above assumption), any c can only decrypt to one of 0 or 1.
- Hiding: without knowledge of sk, cannot distinguish encryptions of 0 from encryption of 1 (by definition of security of encryption).

Commitment Functionality

- Committer sends σ and r to the trusted party
- Trusted party sends the commitment string defined by σ and r to the receiver.
- Note: this defines extraction as well (in the ideal model, must obtain the value σ to send to the trusted party.

Fixing versus Extracting

- Commitment suffices for fixing input (party 1 holds a string that uniquely determines party 2's input, and vice versa).
- This does not suffice for extraction:
  - Need also to use a proof of knowledge (later)…

Coin Tossing

- Aim: fix uniform random tape of each party
- Coin tossing of a bit:
  - Parties 1 and 2 obtain the same uniform (pseudorandom) bit r
- The coin tossing functionality:
  - Trusted party chooses a random bit r and sends it to both parties

Coin Tossing Protocol [Blum]

- Protocol:
  - Party 1 chooses a random bit r_1, computes a commitment c_1 to r_1 and sends c_1 to Party 2
  - Party 2 chooses a random bit r_2 and sends it to Party 1
  - Party 1 decommits, revealing r_1
- Outputs:
  - Both parties output $r=r_1 \oplus r_2$

Proof of Security

- Methodology:
  - Simulator/Ideal adversary will obtain the bit r from the trusted party
  - Simulator will then “interact” with the adversary so that the result of the protocol is r
  - Simulator outputs whatever real adversary outputs

  - This ensures that the result of an ideal execution is indistinguishable from a real protocol.

Augmented Coin Tossing

- Recall: coin tossing is for choosing random tapes of parties.
- But, Party 1 does not know Party 2’s random tape in reality!
- Augmented coin-tossing:
  - Party 1 obtains a random string r
  - Party 2 obtains a commitment to r

Protocol Emulation

- At this stage, each party holds a commitment to the other party's input and random tape.
- A protocol is a deterministic function of a party's input, random tape and series of incoming messages.
- Therefore, the commitments can be used to force the parties to follow the protocol instructions.

Forcing Good Behavior

- AIM: a party should prove that the message it is sending is correct.
  - That is, it is consistent with the protocol instructions, given the input and random-tape that are committed and the incoming messages (that are public).

Tool: Zero Knowledge

- Problem setting: a prover wishes to prove a statement to the verifier so that:
  - Zero knowledge: the verifier will learn nothing beyond the fact that the statement is correct
  - Soundness: the prover will not be able to convince the verifier of a wrong statement

- Zero-knowledge proven using simulation.

Illustrative Example

- Prover has two colored cards that he claims are of different color
- The verifier is color blind and wants a proof that the colors are different.
- Idea 1: use a machine to measure the light waves and color. But, then the verifier will learn what the colors are.

Example (continued)

- Protocol:
  - Verifier writes color1 and color2 on the back of the cards and shows the prover
  - Verifier holds out one card so that the prover only sees the front
  - The prover then says whether or not it is color1 or color2
- Soundness: if they are both the same color, the prover will fail with probability ½. By repeating many times, will obtain good soundness bound.
- Zero knowledge: verifier can simulate by itself by holding out a card and just saying the color that it knows

Zero Knowledge

- Fundamental Theorem [GMR]: zero-knowledge proofs exist for all languages in NP
- Observation: given commitment to input and random tape, and given incoming message series, **correctness** of next message in a protocol is an NP-statement.
- Therefore, it can be proved in zero-knowledge.

Protocol Compilation

- Given any protocol, construct a new protocol as follows:
  - Both parties commit to inputs
  - Both parties generate uniform random tape
  - Parties send messages to each other, each message is proved “correct” with respect to the original protocol, with zero-knowledge proofs.

Resulting Protocol

- Theorem: if the initial protocol was secure against semi-honest adversaries, then the compiled protocol is secure against malicious adversaries.
- Proof:
  - Show that even malicious adversaries are limited to semi-honest behavior.
  - Show that the additional messages from the compilation all reveal nothing.

Summary

- GMW paradigm:
  - First, construct a protocol for semi-honest adv.
  - Then, compile it so that it is secure also against malicious adversaries
- There are many other ways to construct secure protocols – some of them significantly more efficient.
- Efficient protocols against semi-honest adversaries are far easier to obtain than for malicious adversaries.

Useful References

- Oded Goldreich. *Foundations of Cryptography Volume 1 – Basic Tools.* Cambridge University Press.
  - Computational hardness, pseudorandomness, zero knowledge
- Oded Goldreich. *Foundations of Cryptography Volume 2 – Basic Applications.* Cambridge University Press.
  - Chapter on secure computation
- Papers: an endless list (I would rather not go on record here, but am very happy to personally refer people).