Advanced Topics in Cryptography

Lecture 8
Secure two-Party Computation

Benny Pinkas

Related papers

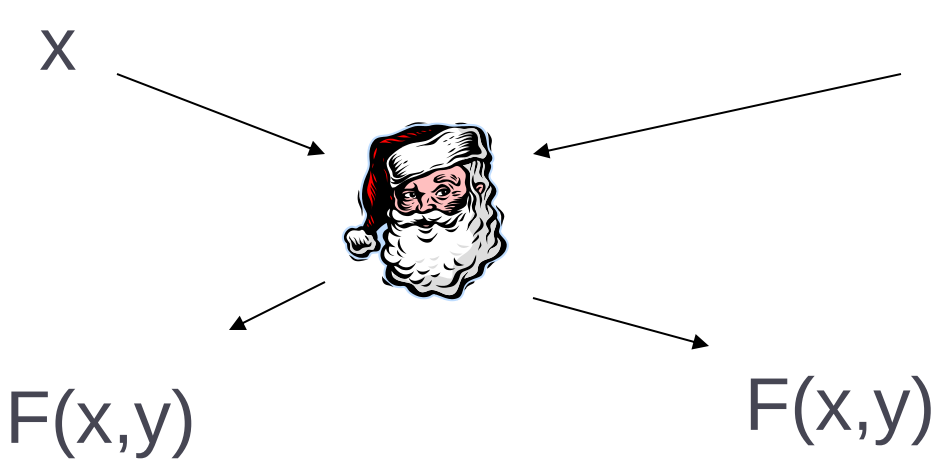
- Related papers:
  - A. Yao
    How to Generate and Exchange Secrets.
    In 27*th FOCS*, pages 162–167, 1986.
    (the first paper on secure computation)
  - Y. Lindell and B. Pinkas
    A Proof of Yao's Protocol for Secure Two-Party Computation,
    http://eprint.iacr.org/2004/175.
    (full proof of security)

Secure two-party computation - definition

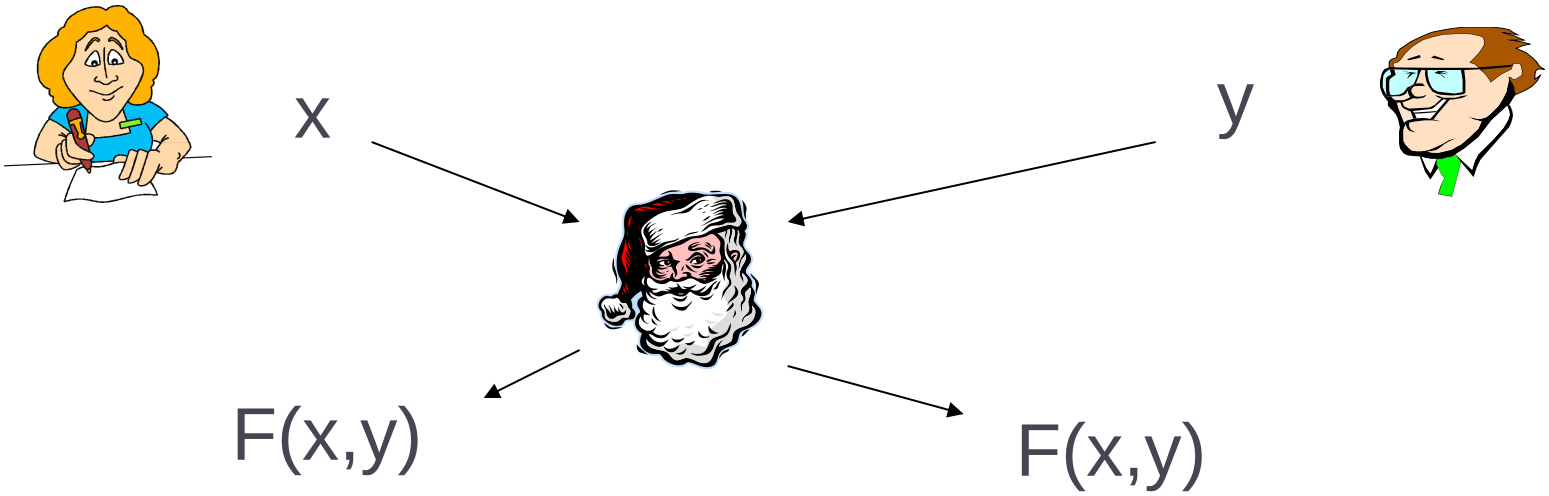
Input: x y

Output: F(x,y) *and nothing else*

As if…



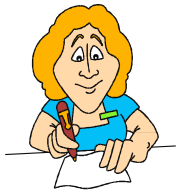
Does the trusted party scenario make sense?



- We cannot hope for more privacy
- Does the trusted party scenario make sense?
  - Are the parties motivated to submit their true inputs?
  - Can they tolerate the disclosure of F(x,y)?
- If so, we can implement the scenario without a trusted party.

Secure two-party computation - definition

Real world

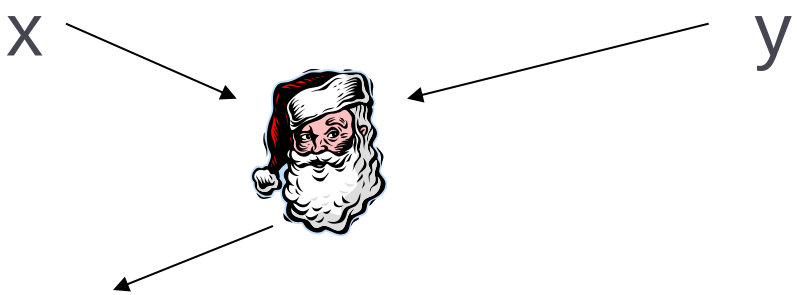


Input: x y

Output: F(x,y) *and nothing else*

As if…

Ideal world



Definition

- For every A in the real world, there is an A' in the ideal world, s.t. whatever A can do in the real world A' can do in the ideal world

- The same for the other party. Need not worry about the case that both are corrupt.

- Semi-honest case: (A behaves according to the protocol.)
  - It is sufficient to require that A is able to simulate the interaction from its input and output alone.

Simulation based definition of security, for Deterministic Functionalities in the Semi-honest case

- In the case of deterministic functionalities, the outputs are fully determined by the inputs
- It suffices to separately prove
  - Correctness
  - Simulation: show that can generate view of semi-honest adversary (corrupted parties' view), given inputs and outputs only
    - In other words…

Deterministic Functionalities

- Separately prove the following two statements
  - The output of the protocol is indistinguishable from the output of the functionality
  - There exists a simulator S_1 such that for any adversary A controlling P1, the output of **A**, and the output of $\mathbf{S_1}$ given $\mathbf{x_1}$ and $\mathbf{f_1(x,y)}$, are indistinguishable.
  - Namely, $\{S_1(x, f_1(x, y))\}_{x,y\in\{0,1\}^*} \equiv \{view^{\pi}_1(x, y)\}_{x,y\in\{0,1\}^*}$

    (If the view of the adversary controlling P_1 in the protocol is indistinguishable from that generated by the simulator, so is also the output generated by the adversary.)

Deterministic Functionalities

- Similarly
  - Prove that there exists a simulator S_2 such that for any adversary A controlling P2, the output of **A**, and the output of **S2** given $\mathbf{x_2}$ and $\mathbf{f_2(x,y)}$, are indistinguishable.

  - Namely, $\{S_2(y, f_2(x, y))\}_{x,y\in\{0,1\}^*} \equiv \{view^{\pi}_2(x, y)\}_{x,y\in\{0,1\}^*}$

Functionalities with Output to a Single Party

- In the standard definition of secure computation, both parties receive (possibly different) outputs.
  - It is often simpler to assume that only party P_2 receives output.
  - This suffices for the general case:
  - Any protocol that can be used to securely compute any ppt functionality f(x,y) where only P_2 receives output, can be used to securely compute any efficient functionality $f=(f_1,f_2)$ where P_1 receives $f_1(x,y)$ and P_2 receives $f_2(x,y)$.
  - Given $f(x,y)=(f_1,f_2)$, we define $f'((x,k),y) = E_k(f_1(x,y)), f_2(x,y)$. I.e., P_1's input to f' includes a key k, and the output contains an encryption of f_1 with k, and also f_2. P2 can learn this output and send its first part to P_1.

Secure two-party computation of general functions [Yao]

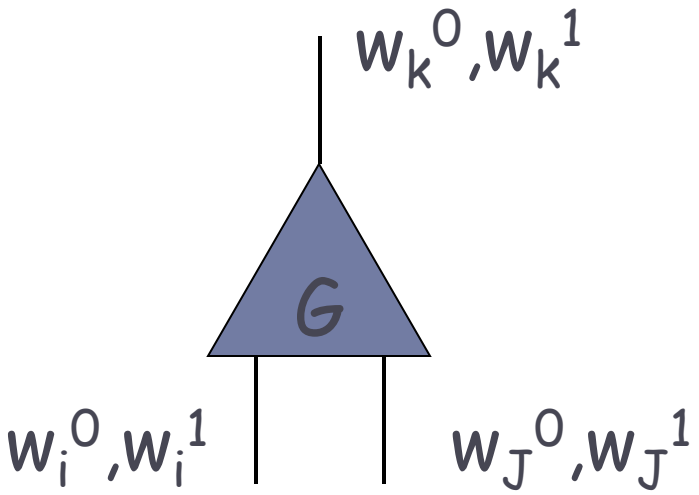
- First, represent the function F as a Boolean circuit C
- This is always possible
- Sometimes it is easy (additions, comparisons)
- Sometimes the result is inefficient (e.g. for indirect addressing)

Basic ideas

- A simple circuit is evaluated by
  - setting values to its input gates
  - For each gate, computing the value of the outgoing wire as a function of the wires going into the gate.
- Secure computation:
  - No party should learn the values of any wires, except for the output wires of the circuit
- Yao's protocol
  - A compiler which takes a circuit and transforms it to a circuit which hides all information but the final output.

Garbling the circuit

- Bob (aka P_1, or "the constructor") constructs the circuit, and then garbles it.



$W_k^0 \equiv 0$ on wire k
$W_k^1 \equiv 1$ on wire k

(Alice, P_2, will learn one string per wire, but not which bit it corresponds to.)

Gate tables

- For every gate, every combination of input values is used as a key for encrypting the corresponding output
- Assume G=AND. Bob constructs a table:
  - Encryption of w_k^0 using keys w_i^0, w_j^0
  - Encryption of w_k^0 using keys w_i^0, w_j^1
  - Encryption of w_k^0 using keys w_i^1, w_j^0
  - Encryption of w_k^1 using keys w_i^1, w_j^1
  - …and permutes the order of the entries

- Result: given w_i^x, w_j^y, can compute $w_k^{G(x,y)}$
  - (encryption can be done using a prf)

The encryption scheme being used (I)

- The encryption must be secure in the sense that
  - for every two (known) messages x and y, no adversary can distinguish an encryption of x from an encryption of y.
  - This must hold even if many messages are encrypted with the same key. Therefore, a one-time pad is not a good choice.
  - Motivation: a wire might be used in many gates, and the corresponding garbled value is used as an encryption key in each of them.

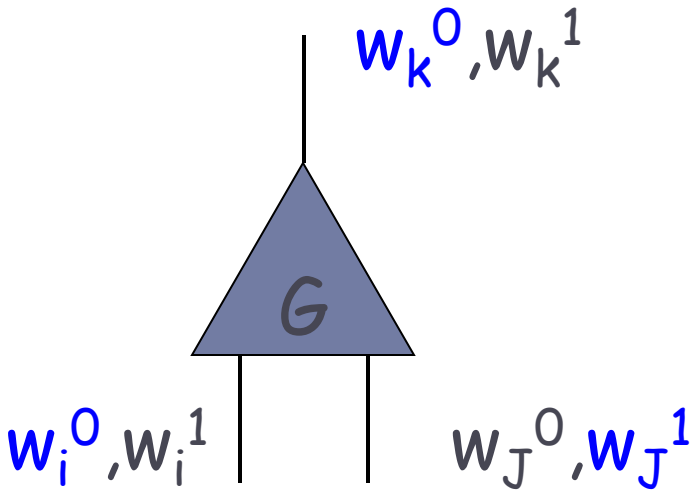
The encryption scheme being used (II)

- It must hold that there will be negligible probability that an encryption with one key will fall in the range of encryptions with another key.
  - So that when Alice tries to decrypt the entries in the table, she will only be successful for a single entry.
- In addition, given a key k, it is must be possible to verify if a given ciphertext is in the range of k.

- These properties are satisfied by taking a semantically secure encryption E, and using it to encrypt x by encrypting $x|0^n$.
  - Namely, compute $E_k(x)=(r,\ f_k(r) \oplus x0^n)$, where f is a prf.

Secure computation

- Bob sends the table of gate G to Alice
- Given, e.g., w_i^0, w_J^1, Alice computes w_k^0, but doesn't know the actual values of the wires.
- Alice cannot decrypt the entries of input pairs different from (0,1)
- For the wires of circuit output:
  - Bob does not define "garbled" values for the output wires, but rather encrypts instead a 0/1 value.



Secure computation

- Bob sends to Alice
  - Tables encoding each circuit gate.
  - Garbled values (w's) of his input values.

- If Alice gets garbled values (w's) of her input values, she can compute the output of the circuit, and nothing else.
  - Why can't Bob provide Alice with the keys corresponding to both 0 and 1 for her input wires?

Alice's input

- For every wire i of Alice's input:
  - The parties run an OT protocol
  - Alice's input is her input bit (s).
  - Bob's input is w_i^0, w_i^1
  - Alice learns w_i^s

- The OTs for all input wires can be run in parallel.
- Afterwards Alice can compute the circuit by herself.
  - She decrypts the entries in each gate until finding the entry which ends with 0^n. Then continues to the next layer of the circuit.

Secure computation – the big picture (simplified)

- Represent the function as a circuit C
- Bob sends to Alice 4|C| encryptions (e.g., 64|C| Bytes)*.
- Alice performs an OT for every input bit. (Can do, e.g. 1000 OTs per sec.)

- Relatively low overhead:
  - Constant number of (~1) rounds of communication.
  - Public key overhead depends on the size of Alice's input
  - Communication depends on the size of the circuit
  - Efficient for medium size circuits!
  - (*) Note that using the encryption system we describe earlier requires longer ciphertexts, but it is possible to use other security assumptions that result in shorter ciphertexts.

Secure computation – correctness

- Holds since the encryption scheme has the property that there is negligible probability that an encryption with one key will fall in the range of encryptions with another key.
- Therefore Alice can always identify the table entry which corresponds to the actual value computed in the circuit.

- Removing the small error probability:
  - When generating the circuit, Bob verifies that all tables always decrypt to a single value.
  - There is a different technique that uses a single additional bit for signaling.

Secure computation: security (semi-honest case)

- A simulation based proof of security:
- In the protocol:
  - Bob sends tables and his own garbled values to Alice
  - The parties run OTs where Alice learns garbled values
  - Alice computes the output of the circuit and sends it to Bob

- A corrupt Bob: its view in the protocol contains the execution of the OTs and a single message containing f(x,y) received from Alice.

Secure computation: security (semi-honest case)

- A corrupt Bob: its view in the protocol contains the execution of the OTs and a single message containing f(x,y) received from Alice.

- Since the OTs are secure, there is a simulator which simulates Bob's view in the OT given its input to them alone.

- The simulator of Bob's view in Yao's protocol has inputs x,f(x,y). It operates in the following way:
  - First simulates the messages that Bob sends to Alice. √
  - Then simulates Bob's view in the OT protocols. √
  - Then simulates Bob receiving f(x,y) from Alice. √

Secure computation: security (semi-honest case)

- A corrupt Alice, intuition:
  - Since OTs are secure, learns one garbled value per input wire.
  - In every gate, if she knows only one garbled value of every input wire, she cannot decrypt more than a single value of output wire.

- A simulation argument appears at "*A Proof of Yao's Protocol for Secure Two-Party Computation*"
  - The simulator knows y and f(x,y).
  - It must send a garbled circuit to Alice. It cannot construct it according to the protocol since it does not know x.

Secure computation: security (semi-honest case)

- The simulation
  - The simulator knows y and f(x,y).
  - Instead of generating a correct circuit, the simulator sends Alice a "fake" circuit that always computes f(x,y), regardless of its inputs.
  - This is done by constructing gate tables that encrypt the same garbled value in all 4 entries.
    - Therefore regardless of the actual input to the circuit, its output and all internal values will always be the same.
  - The detailed proof shows that the security of the encryptions ensure that Alice cannot distinguish this circuit from the correct circuit.

Secure computation: security (semi-honest case)

- More details about the proof
  - Show that Alice cannot distinguish the circuit it receives from the correct circuit.
  - First, show that Alice's view in a real execution is indistinguishable from a hybrid distribution $H_{ot}(x, y)$ in which the real oblivious transfers are replaced with simulated ones.
  - Then consider a series of **hybrids** $H_i(x,y)$ in which one gate at a time is replaced in the real garbled circuit.
  - $H_0(x,y)$ is equal to $H_{ot}(x,y)$ and contains a real garbled circuit
  - $H_{|C|}(x,y)$ contains the fake circuit constructed by S.
  - The difference between $H_i(x,y)$ and $H_{i+1}(x,y)$ is that one more real table is replaced with a fake one.

Secure computation: security (semi-honest case)

- More details about the proof
  - Denote by p_i the probability with which Alice outputs 1 when she is given $H_i(x,y)$ as input.
  - Suppose that it is possible to distinguish with probability p between $H_0(x,y)$ and $H_{|C|}(x,y)$. Namely, $|p_{|C|} - p_0| > p$.
  - It holds that $p_{|C|} - p_0 = (p_{|C|} - p_{|C|-1})+(p_{|C|-1} - p_{|C|-2})+\ldots+(p_1 - p_0)$
  - Therefore $p < |p_{|C|} - p_0| \leq |p_{|C|} - p_{|C|-1}|+|p_{|C|-1} - p_{|C|-2}|+\ldots+|p_1 - p_0|$
  - Therefore there is an $1 \leq i < |C|$ such that $|p_{i+1} - p_i| > p/|C|$. Namely, it is possible to distinguish with this probability between $H_i(x,y)$ and $H_{i+1}(x,y)$.

  - But then it is possible to use the distinguisher between $H_i(x,y)$ and $H_{i+1}(x,y)$ in order to break the security of the encryption scheme (by showing a reduction from breaking the encryption to the distinguisher).

Secure computation: security (semi-honest case)

- More details about the proof
  - If it is possible to distinguish with probability p between $H_0(x,y)$ and $H_{|C|}(x,y)$, then there must be an $1 \leq I < |C|$ such that it is possible to distinguish with probability at least $p/|C|$ between $H_i(x,y)$ and $H_{i+1}(x,y)$.
  - But then it is possible to use the distinguisher between $H_i(x,y)$ and $H_{i+1}(x,y)$ in order to break the security of the encryption scheme (by showing a reduction from breaking the encryption to the distinguisher).