Advanced Topics in Cryptography

Lecture 2: oblivious transfer, two-party secure computation

Benny Pinkas

Related papers

- 1-out-of-N oblivious transfer
  - M. Naor and B. Pinkas
    Computationally Secure Oblivious Transfer
    Journal of Cryptology, Vol. 18, No. 1, 2005.
- Secure Computation
  - A. Yao
    How to Generate and Exchange Secrets.
    In 27*th FOCS*, pages 162–167, 1986.
    (the first paper on secure computation)
  - D. Malkhi, N. Nisan, B. Pinkas and Y. Sella,
    Fairplay - A Secure Two-Party Computation System, Proceedings of Usenix Security '2004.
    (efficient implementation of two-party 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)

1-out-of-N OT

- A generalization of 1-out-of-2 OT:
  - Sender has N inputs, $x_0, \ldots x_N$.
  - Receiver has an input $j \in \{1,2,\ldots,N\}$.
- Output:
  - Receiver learns x_j and nothing else.
  - Sender learns nothing about j.

- We would like to construct 1-out-of-N OT, or reductions from 1-out-of-N OT to 1-out-of-2 OT.
  - It was shown that any such reduction which provides unconditional security requires at least N-1 OTs.
  - Since OT has a high computational overhead, we would like to do better than that.

Construction 1: A recursive protocol for 1-out-of-N OT

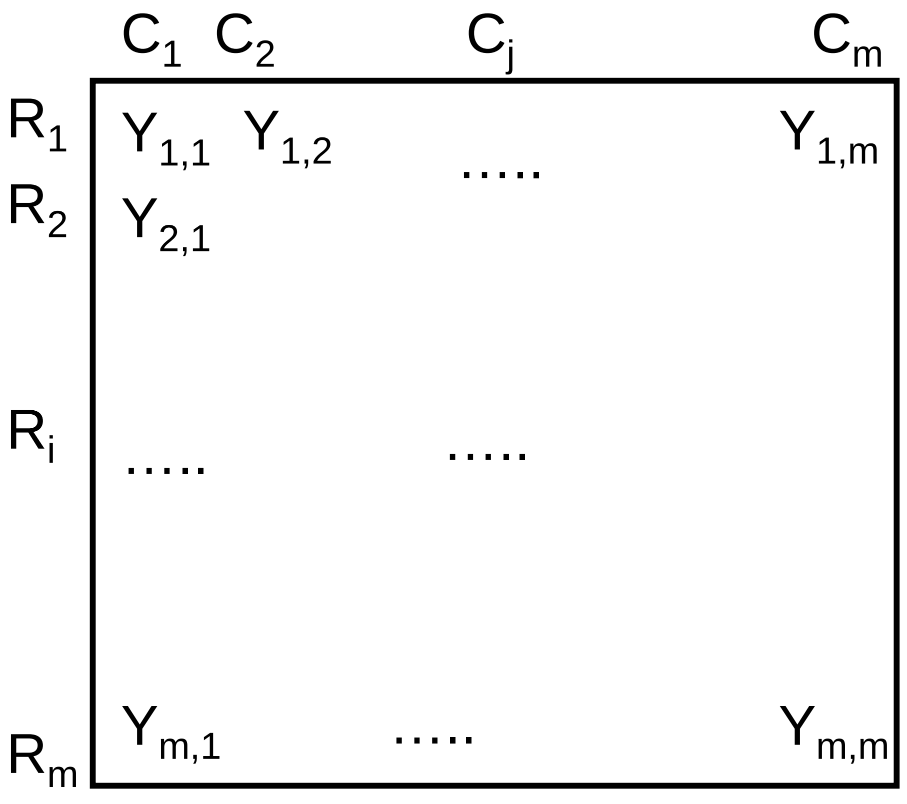
- The reduction uses a pseudo-random function $F_k()$.
  - It holds that if *k* is chosen at random and kept secret, no adversary can distinguish between $(x,F_k(x))$ and a random value, for every x.
- The protocol reduces 1-out-of-m OT to 1-out-of-$\sqrt{m}$ OT. This can done recursively.

A recursive protocol for 1-out-of-N OT

Sender's original input:

$$\begin{matrix} X_{1,1} & X_{1,2} & \ldots\ldots & X_{1,m} \\ X_{2,1} & & & \\ \ldots\ldots & & \ldots\ldots & \\ X_{m,1} & & \ldots\ldots & X_{m,m} \end{matrix}$$

A recursive protocol for 1-out-of-N OT

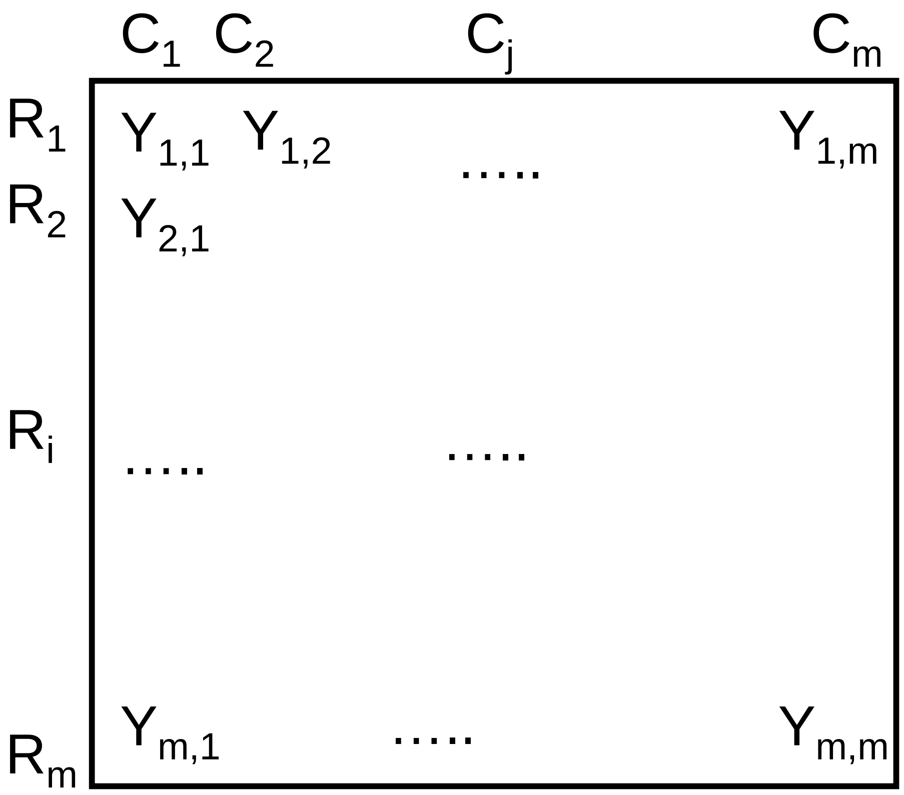


Sender replaces each $X_{i,j}$ with its encryption using the keys R_i and C_j

$$Y_{i,j}=X_{i,j} \oplus F_{Ri}(j) \oplus F_{Cj}(i).$$

no value of F() is used more than once

A recursive protocol for 1-out-of-N OT



- Receiver uses two invocations of 1-out-of-m OT to learn R_i and C_j.
- Sender sends all Y values
- Receiver decrypts $Y_{i,j}$ and learns $X_{i,j}$
- Every other Y value is encrypted with at least one key unknown to the receiver

Construction 2: a reduction to 1-out-of-2 OT

- Assume $N=2^n$. The receiver's input is $j=j_n,\ldots,j_1$.
- Preprocessing: the sender prepares 2n keys
  - $(k_{1,0},k_{1,1}), (k_{2,0},k_{2,1}),\ldots, (k_{n,0},k_{n,1})$.
  - and encryptions $Y_i = X_i \oplus F_{K\_\{1,i1\}}(i) \oplus \ldots \oplus F_{K\_\{1,in\}}(i)$
    - (namely, X_i is encrypted using the keys corresponding to the bits of i).
- For each 1 ≤ s ≤ n, the parties run a 1-out-of-2 OT:
  - The sender's input is $(k_{s,0},k_{s,1})$.
  - The receiver's input is j_s.
- The sender sends $Y_1,\ldots,Y_n$ to the receiver.
- The receiver reconstructs x_j.

- Why can't we use $Y_i = X_i \oplus K_{1,i1}(i) \oplus \ldots \oplus K_{1,in}(i)$?

Analysis

- Overhead:
  - N=logN invocations of 1-out-of-2 OT (this is the bulk of the overhead).
  - The preprocessing stage requires NlogN invocations of the pseudo-random function *F()*.
- Receiver privacy (hand-waving):
  - Since the 1-out-of-2 OTs do not leak information about the receiver’s input.
- Sender privacy:
  - It can be shown that if the receiver learns about more than a single item, then either the 1-out-of-2 OT is not secure, or F() is not pseudo-random.

Applications

- Database queries
- Checking the size of a search engine index??

Secure two-party computation - definition

Input: x y

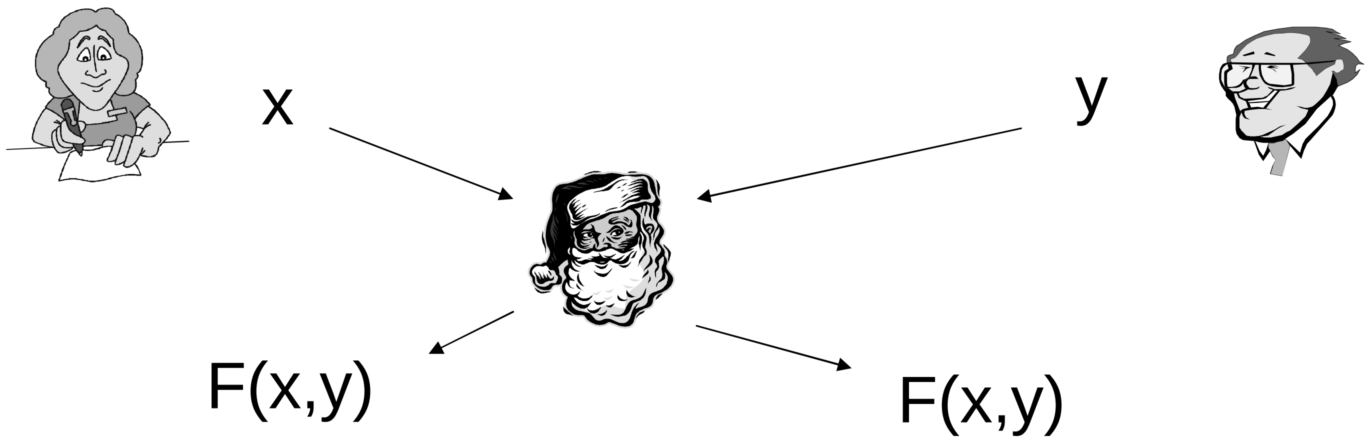
Output: F(x,y) *and nothing else*

As if… x y

F(x,y) F(x,y)

Examples…

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.

Fairness, aka early termination

- Suppose both parties (A and B) need to learn the output
- Assume that the last message in the protocol goes from A to B
- A malicious A does not send that message
  - $\Rightarrow$ B does not learn output

- There is no perfect solution to this problem. However, this corrupt behavior is detectable.

Secure two-party computation - definition

Real world

Input: x y

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

As if…

x y

Ideal world

F(x,y)

Definition

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

- The same for B. Need not worry about the case the 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 the output alone.

Examples of Simple Privacy Preserving Primitives

- Reasonably efficient solutions satisfying the definition above.
  - Is X > Y? Is X = Y?
  - *What is $X \cap Y$? What is median of $X \cup Y$?*
  - Auctions (negotiations). Many parties, private bids. Compute the winning bidder and the sale price, but nothing else.
  - Add privacy to existing data mining algs.

Secure two-party computation of general functions [Yao]

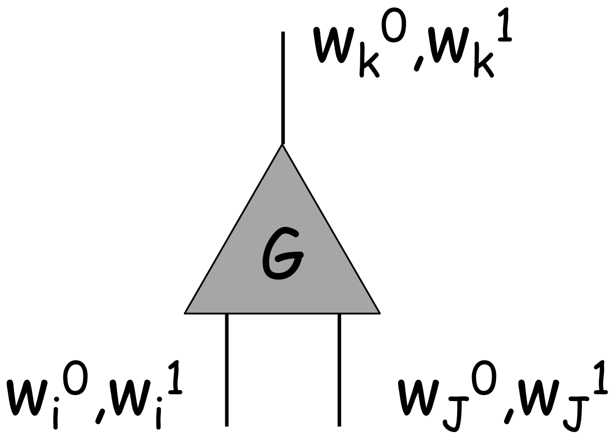
- First, represent the function F as a Boolean circuit C
- It's always possible
- Sometimes it's easy (additions, comparisons)
- Sometimes the result is inefficient (e.g. for indirect addressing)

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 "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 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)

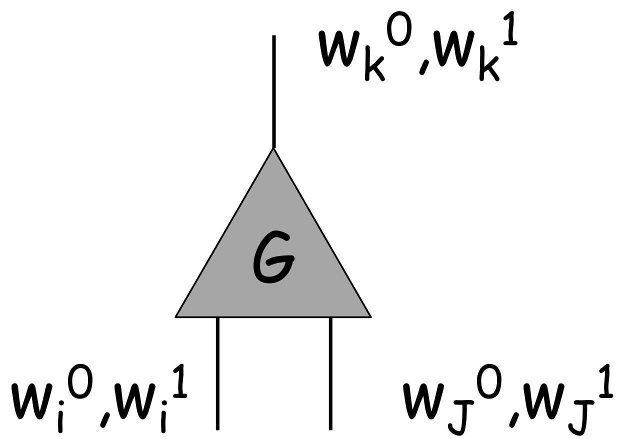
The encryption scheme being used

- The encryption scheme must be secure 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.

- It must hold that a random string happens to be a correct ciphertext only with negligible probability.
  - So that when Alice tries to decrypt the entries in the table, she will only be successful for on entry.

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 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 the Bon 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.

Secure computation – the big picture (simplified)

- Represent the function as a circuit C
- Bob sends to Alice 4|C| encryptions (e.g., 50|C| Bytes).
- Alice performs an OT for every input bit. (Can do, e.g. 100 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!

Secure computation: security (semi-honest case)

- In the protocol:
  - Bob sends tables to Alice
  - The parties run OTs where Alice learns garbled values
  - Alice computes the output of the circuit

- A corrupt Bob: sees the execution of the OTs. If OTs are secure learns nothing about Alice's input.

- A corrupt Alice:
  - Since OTs are secure, learns one garbled value per inptu 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*"

Example

- Comparing two N bit numbers

- What’s the overhead?

Applications

- Two parties. Two large data sets.
- Max?
- Mean?
- Median?
- Intersection?

Conclusions

- If the circuit is not too large:
  - Efficient secure two-party computation.
  - Efficient multi-party computation with two semi-trusted parties.
  - An “open” question: >2 semi-trusted parties.
- If the circuit is large: we currently need ad-hoc solutions.