Session 3: The GMW and BMR Multi-Party Protocols

Benny Pinkas

Bar-Ilan University

Overview

- The **GMW** (Goldreich-Micali-Wigderson) protocol
  – In this lecture we only cover security against semi-honest adversaries
  – # rounds depends on circuit depth
  – O. Goldreich, Foundations of Cryptography, Vol. II, Chapter 7.
- Oblivious Transfer (OT) is extensively used in the GMW protocol
  – OT extension is a method that greatly reduces the overhead of OT

The setting (for GMW protocol)

- Parties $P_1,\ldots,P_n$
- Inputs $x_1,\ldots,x_n$ (bits, but can be easily generalized)
- Outputs $y_1,\ldots,y_n$

- The functionality is described as a Boolean circuit.
  - Wlog, uses only XOR (+) and AND gates
  - These gates correspond to +, * modulo 2.
  - Wires are **ordered** so that if wire k is a function of wires i and j, then i<k and j<k.

The setting

- The adversary controls a subset of the parties
  - This subset is defined before the protocol begins (is “non-adaptive”)
  - We will not cover the adaptive case

- Communication
  - Synchronous
  - Private channels between any pair of parties (can be easily implemented using encryption)

Adversarial models

- We will cover the semi-honest case

- If adversaries can be malicious but do not abort
  - GMW: A protocol secure against any number of malicious parties
- If adversaries can be malicious and can also abort
  - GMW: A protocol secure against a minority of malicious parties with abort (will not be discussed here)

Protocol for semi-honest setting

- **The protocol in a nutshell:**
  – Each party shares its input bit
  – Scan the circuit gate by gate
    - Input values of gate are shared by the parties
    - Run a protocol computing a sharing of the output value of the gate
    - Repeat
  – Publish outputs

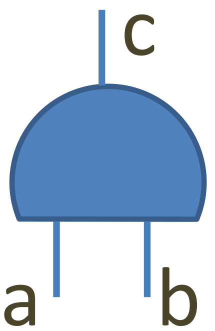
Protocol for semi-honest setting

- **The protocol:**
  - Each party shares its input bit
  - The sharing procedure:
    - P_i has input bit x_i
    - It chooses random bits $r_{i,j}$ for all $i \neq j$.
    - Sends bit $r_{i,j}$ to P_j.
    - Sets its own share to be $r_{i,i} = x_i + (\Sigma_{j\neq i}\, r_{i,j})$ mod 2
    - Therefore $\Sigma_{j=1\ldots n}\, r_{i,j} = x_i$ mod 2.

  - Now every P_j has n shares, one for each input x_i of each P_i.

Evaluating the circuit

- Scan circuit by the order of wires
- Wire c is a function of wires a,b
  - P_i has shares a_i, b_i. Must get share c_i of c .
  - Addition (xor) gate:
    - P_i computes $c_i=a_i+b_i$.
  - Indeed, $c = a+b \pmod 2 = (a_1+\ldots+a_n) + (b_1+\ldots+b_n) = (a_1+b_1)+\ldots+(a_n+b_n) = c_1+\ldots+c_n$



Evaluating multiplication (AND) gates

- $c = a \cdot b = (a_1 + \ldots + a_n) \cdot (b_1 + \ldots + b_n) = \Sigma_{i=1\ldots n}\, a_i b_i + \Sigma_{i \neq j} a_i b_j = \Sigma_{i=1\ldots n}\, a_i b_i + \Sigma_{1 \leq i < j \leq n}\, (a_i b_j + a_j b_i) \mod 2$

- P_i will obtain a share of $a_i b_i + \Sigma_{i \neq j}\, (a_i b_j + a_j b_i)$

- Computing $a_i b_i$ by P_i is easy
- What about $a_i b_j + a_j b_i$?
- P_i and P_j run the following protocol for every (i,j)

Evaluating multiplication gates

- Input: P_i has a_i,b_i, P_j has a_j,b_j.
- P_i outputs $a_ib_j+a_jb_i+s_{i,j}$. P_j outputs $s_{i,j}$.
- P_j:
  - Chooses a random $s_{i,j}$
  - Computes the four possible outcomes of $a_ib_j+a_jb_i+s_{i,j}$, depending on the four options for P_i's inputs.
  - Sets these values to be its input to a 1-out-of-4 OT
- P_i is the receiver, with input $2a_i+b_i$.

Recovering the output bits

- The protocol computes shares of the output wires
- Each party sends its share of an output wire to the party P_i that should learn that output
- P_i can then sum the shares, obtain the value and output it

Proof of Security

- **Recall definition of security for semi-honest setting:**
  - Simulation - Given input and output, can generate the adversary's view of a protocol execution.

- Suppose that an adversary controls the set J of all parties but P_i.
- The simulator is given (x_j,y_j) for all $P_j \in J$.

The simulator

- Shares of input wires: $\forall j \in J$ choose
  - a random share $r_{j,i}$ to be sent from P_j to P_i,
  - and a random share $r_{i,j}$ to be sent from P_i to P_j.
- Shares of multiplication gate wires:
  - $\forall j<i$, choose a random bit as the value learned in the 1-out-of-4 OT.
  - $\forall j>i$, choose a random $s_{i,j}$, and set the four inputs of the OT accordingly.
- Output wire y_j of $j \in J$: set the message received from P_i as the XOR of y_j and the shares of that wire held by $P_j \in J$.

Security proof

- **The output of the simulation is distributed identically to the view in the real protocol**
  - Certainly true for the random shares $r_{i,j}$, $r_{j,i}$ sent from and to P_i.
  - OT for j<i: output is random, as in the real protocol.
  - OT for j<i: input to the OT defined as in the real protocol.
  - Output wires: message from P_i distributed as in the real protocol.

- **QED**

Performance

- **Must run an OT for every multiplication gate**
  - Namely, public key operations per multiplication gate
  - Need a communication round between all parties per every multiplication gate

  - Can process together a set of multiplication gates if all their input wires are already shared
  - Therefore number of rounds is O(d), where d is the depth of the circuit (counting only multiplication gates).

Oblivious Transfer Extension

Oblivious Transfer

- Oblivious Transfer (OT)
  - Sender (P_1) has two inputs x_0,x_1
  - Receiver (P_2) has an input bit s
  - Receiver learns x_s
- Variant: random OT
  - Sender (P_1) has two inputs x_0,x_1
  - For a randomly chosen bit s, receiver learns (x_s,s)

Efficiency of Oblivious Transfer

- OT is very efficient, but still requires exponentiations per transfer
  - When doing thousands (or millions) of OTs, this will become very costly
- Protocols for secure computation typically use OTs per gate or per input bit

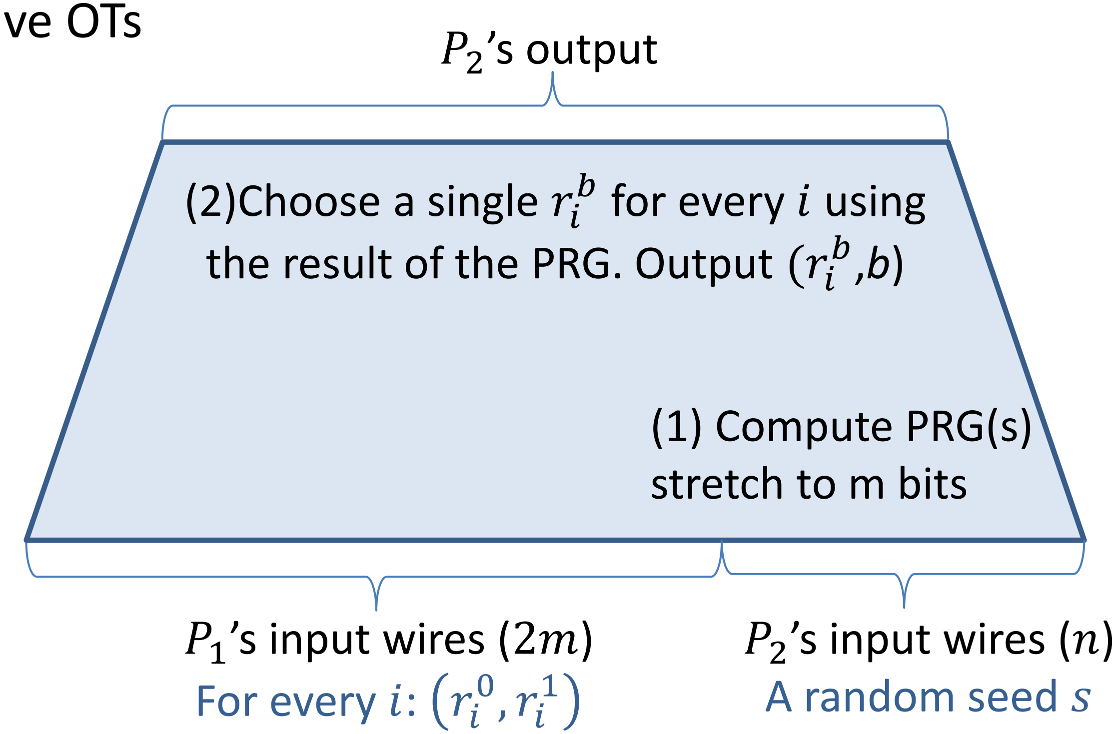
- Impagliazzo and Rudich 1989: there is no blackbox construction of OT from OWF ☹

Oblivious Transfer Extensions

- An OT extension is a protocol that:
  - Uses a “small” number of base OTs (e.g., 128)
  - Uses cheap symmetric crypto to achieve many OTs (e.g., millions)
  - This is like hybrid encryption
- Note that it’s not clear that this is even possible!

Beaver's OT Extension

- **A theoretical construction**
  - The number of OTs in Yao's protocol depends only on evaluator's input
  - Computing the circuit requires only n OTs but provides $m \gg n$ effective OTs



Random vs Regular OT

- **Beaver's protocol computes a random OT**
  - P_2 is the receiver. Its input bit s is randomly chosen.
  - P_1 is the sender. It has a pair of input bits (r_0,r_1).
  - P_2 learns the bit r_s.

Random vs Regular OT

- **We can construct regular OT from random OT (where both parties inputs are random)**
  - P_1's input: (x_0, x_1) P_2's input: σ
  - Parties run random OT on bits (r_0, r_1) and s
    - P_2 receives s, r_s
  - P_2 sends $t = s \oplus \sigma$ to P_1 (essentially tells P_1 the order in which P_1 should mask its inputs).
  - P_1 sends $y_0 = x_0 \oplus r_t$ and $y_1 = x_1 \oplus r_{1-t}$
  - P_2 outputs $y_\sigma \oplus r_s$

Random vs Regular OT

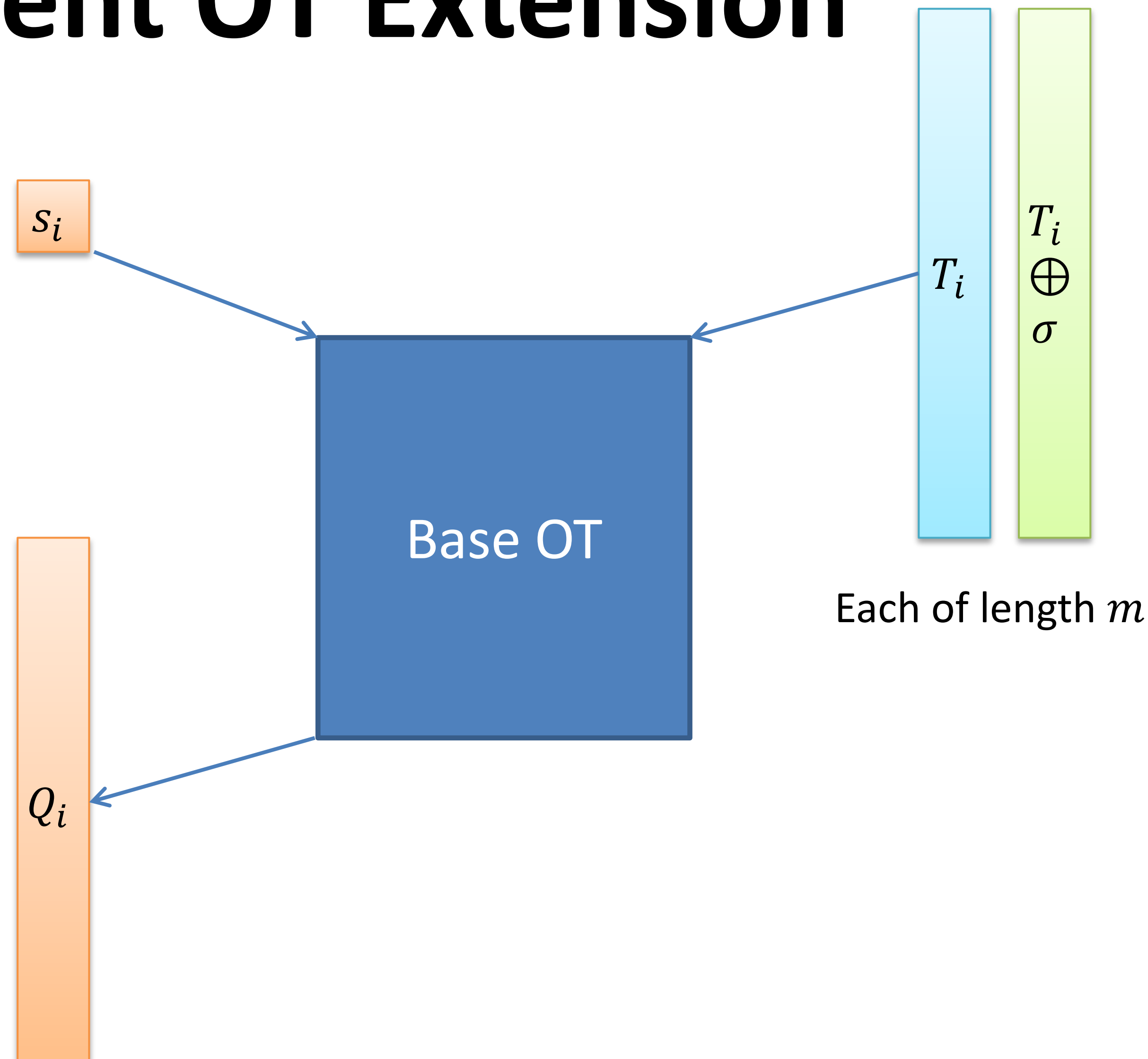
- **Correctness:**
  - If $s = \sigma$ then $t = 0$ and so $y_0 = x_0 \oplus r_0$ and $y_1 = x_1 \oplus r_1$
    - In this case $y_\sigma \oplus r_s = x_\sigma$
  - If $s \neq \sigma$ then $t = 1$ and so $y_0 = x_0 \oplus r_1$ and $y_1 = x_1 \oplus r_0$
    - In this case , too, $y_\sigma \oplus r_s = x_\sigma$
- **Privacy:**
  - P_1 sees only a random bit t and so learns nothing about σ
  - P_2 can learn one of (r_0, r_1) and so only one of (x_0, x_1)

Efficient OT Extension

- **A protocol for extending n =128 OTs to m =10^6 OTs**
  - By Ishai, Kilian, Nissim and Petrank
- **Sender's input: $\left(\boldsymbol{x_1^0}, \boldsymbol{x_1^1}\right), \dots, \left(\boldsymbol{x_m^0}, \boldsymbol{x_m^1}\right)$**
- **Receiver's input: $\boldsymbol{\sigma = \sigma_1, \dots, \sigma_m}$**
- **First phase:**
  - Receiver samples random strings $T_1, \dots, T_n$ each of length m
  - Receiver prepares pairs $(T_i, T_i \oplus \sigma)$ and plays sender in OT
  - Sender chooses random $s = s_1, \dots, s_n$
  - Sender plays receiver with input s_i

Note: roles in these *n* OTs are reversed!

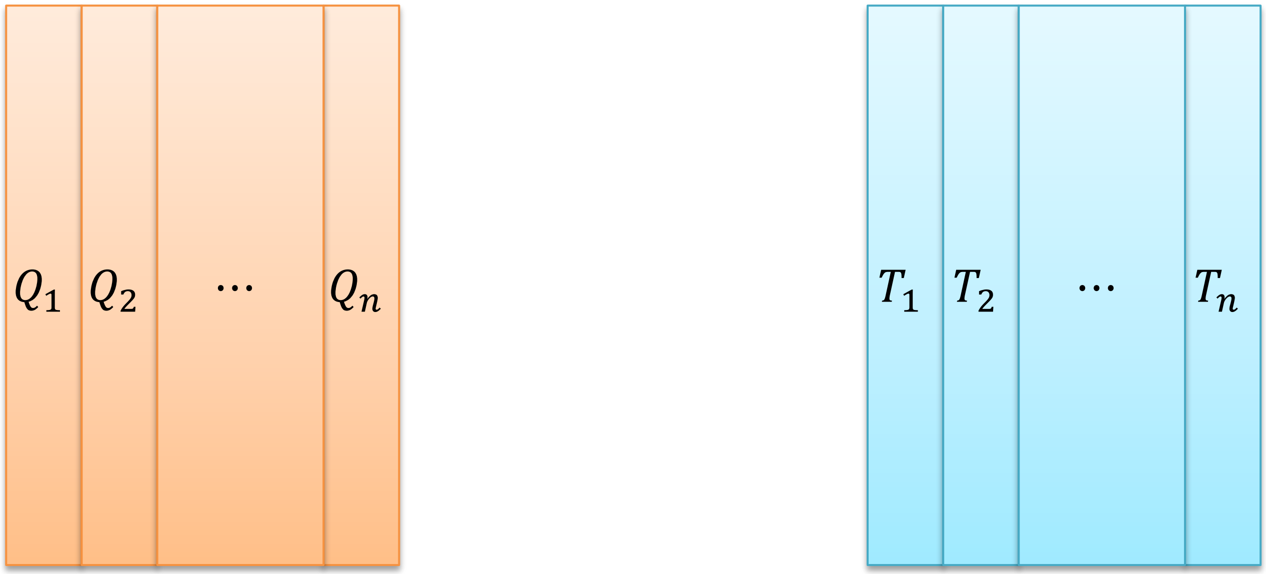
Efficient OT Extension



$$Q_i = \begin{cases} T_i & \text{if } s_i = 0 \\ T_i \oplus \sigma & \text{if } s_i = 1 \end{cases}$$

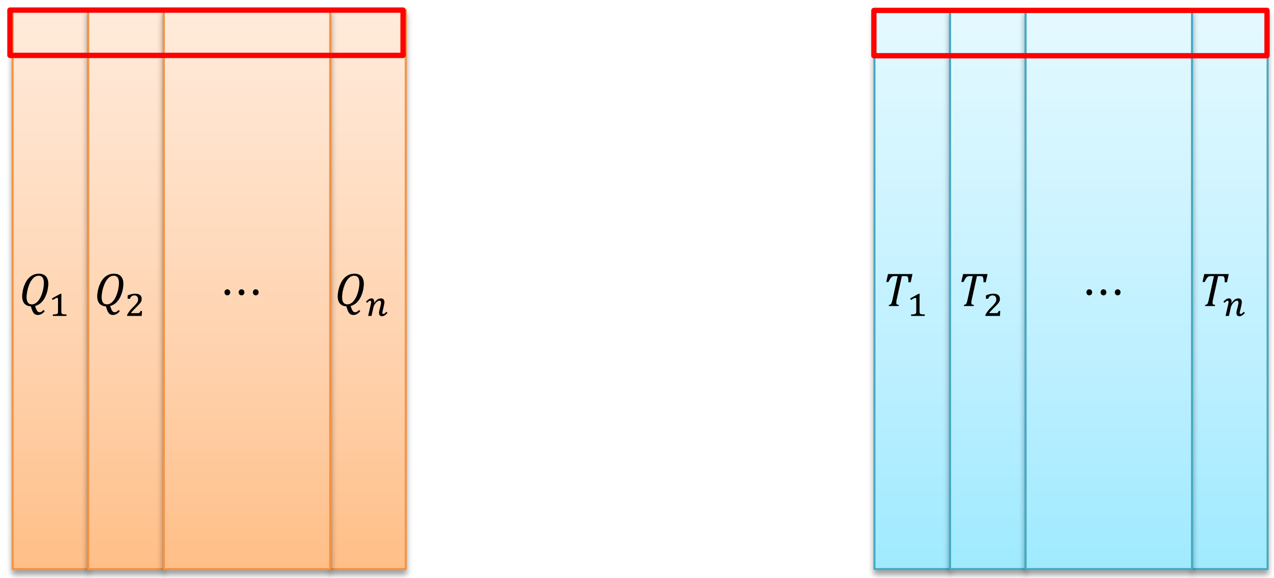
Efficient OT Extension

$$Q_i = \begin{cases} T_i & \text{if } s_i = 0 \\ T_i \oplus \sigma & \text{if } s_i = 1 \end{cases}$$



Efficient OT Extension

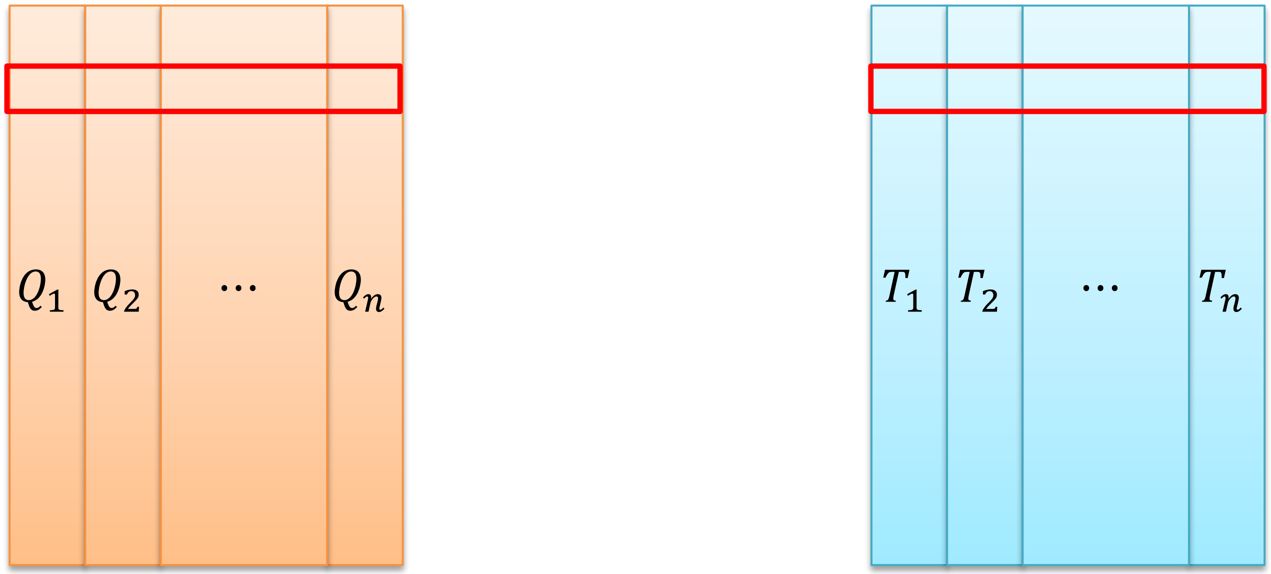
$$Q_i = \begin{cases} T_i & \text{if } s_i = 0 \\ T_i \oplus \sigma & \text{if } s_i = 1 \end{cases}$$



- If $\sigma_1 = 0$ then the **first** row of Q equals the **first** row of T (whatever s equals)
- If $\sigma_1 = 1$ then the **first** row of Q equals the **first** row of T XORed with s:
  - If $s_i = 0$, then equals the first entry in T_i
  - If $s_i = 1$, then equals the first entry in $T_i \oplus 1$ (since XORed with σ_1)
  - In both cases, obtain XOR with s

Efficient OT Extension

$$Q_i = \begin{cases} T_i & \text{if } s_i = 0 \\ T_i \oplus \sigma & \text{if } s_i = 1 \end{cases}$$



- If $\sigma_2 = 0$ then the **second** row of Q equals the **second** row of T (whatever s equals)
- If $\sigma_2 = 1$ then the **second** row of Q equals the **second** row of T XORed with s:
  - If $s_i = 0$, then equals the first entry in T_i
  - If $s_i = 1$, then equals the first entry in $T_i \oplus 1$ (since XORed with σ_1)
- In both cases, obtain XOR with s

Efficient OT Extension

- **Using n base OTs, the matrix is transferred**
- **Look at each row separately (there are m rows)**
  - For the ith row; denote $Q(i)$ and $T(i)$
    - If $\sigma_i = 0$ then $T(i) = Q(i)$
    - If $\sigma_i = 1$ then $T(i) = Q(i) \oplus s$
- **To carry out the ith transfer (phase 2 of the protocol)**
  - Sender sends $y_i^0 = H\big(i, Q(i)\big) \oplus x_i^0$ and $y_i^1 = H(i, Q(i) \oplus s) \oplus x_i^1$
  - Receiver computes $x_i^{\sigma_i} = H\big(i, T(i)\big) \oplus y_i^{\sigma}$
- **Correctness**
  - If $\sigma_i = 0$ then $T(i) = Q(i)$ and so result is correct
  - If $\sigma_i = 1$ then $T(i) = Q(i) \oplus s$ and so result is correct

Efficient OT Extension – Security

- **Corrupted sender**
  - The sender receives either T_i or $T_i \oplus \sigma$
  - Since T_i is random, this reveals nothing about σ

Efficient OT Extension – Security

- **Corrupted receiver**
  - The sender's values are masked by $H\big(i, Q(i)\big)$ and $H(i, Q(i) \oplus s)$
  - The receiver has $H(i, T(i))$ which equals one of them but does not know anything about s (sender's queries in base Ots)
    - In the ROM, without knowing s cannot query the value
    - Can also prove assuming that $r_1, \dots, r_m, H(s \oplus r_1), \dots, H(s \oplus r_m)$ is pseudorandom
    - Note that the receiver knows $r_1, \dots, r_m$ but not s, and $H(s \oplus r_i)$ masks the ith value that the receiver should **not** receive

Complexity of OT extension

- Run n oblivious transfers (costing a few exponentiations each)
- Each actual OT costs a few hash operations
- This is very efficient and can be used to carry out millions of OTs per second
  - [Asharov,Lindell,Schneider,Zohner ACM CCS 2013]
- **Malicious adversaries**: more later in the winter school