Secure Multi-Party Computation

Lecture 17

GMW & BGW Protocols

MPC Protocols

MPC Protocols

- Yao's Garbled Circuit : 2-Party SFE secure against passive adversaries

MPC Protocols

- Yao's Garbled Circuit : 2-Party SFE secure against passive adversaries
  - Using OT and PRG

MPC Protocols

- Yao's Garbled Circuit : 2-Party SFE secure against passive adversaries
  - Using OT and PRG
- Today

MPC Protocols

- Yao's Garbled Circuit : 2-Party SFE secure against passive adversaries
  - Using OT and PRG
- Today
  - Passive-secure GMW protocol: Generalizes to any number of parties, uses OT only

MPC Protocols

- Yao's Garbled Circuit : 2-Party SFE secure against passive adversaries
  - Using OT and PRG
- Today
  - Passive-secure GMW protocol: Generalizes to any number of parties, uses OT only
  - Passive-secure BGW protocol: Doesn't even use OT, but relies on honest-majority

MPC Protocols

- Yao's Garbled Circuit : 2-Party SFE secure against passive adversaries
  - Using OT and PRG
- Today
  - Passive-secure GMW protocol: Generalizes to any number of parties, uses OT only
  - Passive-secure BGW protocol: Doesn't even use OT, but relies on honest-majority
  - Going from passive to active security

Basic GMW

Basic GMW

- Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (by Goldreich-Vainish, Haber-Micali,...)

Basic GMW

- Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (by Goldreich-Vainish, Haber-Micali,...)
- Idea: Evaluate a circuit with wire values secured using (linear) **secret-sharing**

Recall Secret-Sharing

Recall Secret-Sharing

- Fix any "secret" s. Let a, b be random conditioned on s = a + b. (All elements from a finite field.)

Recall Secret-Sharing

- Fix any "secret" s. Let a, b be random conditioned on s = a + b. (All elements from a finite field.)
- Each of a, b by itself carries no information about s. (e.g., can pick a at random, set b = s - a.)

Recall Secret-Sharing

- Fix any "secret" s. Let a, b be random conditioned on s = a + b. (All elements from a finite field.)
- Each of a, b by itself carries no information about s. (e.g., can pick a at random, set b = s - a.)
- Will write $[s]_1$ and $[s]_2$ to denote shares of s

Computing on Shares

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

Computing on Shares

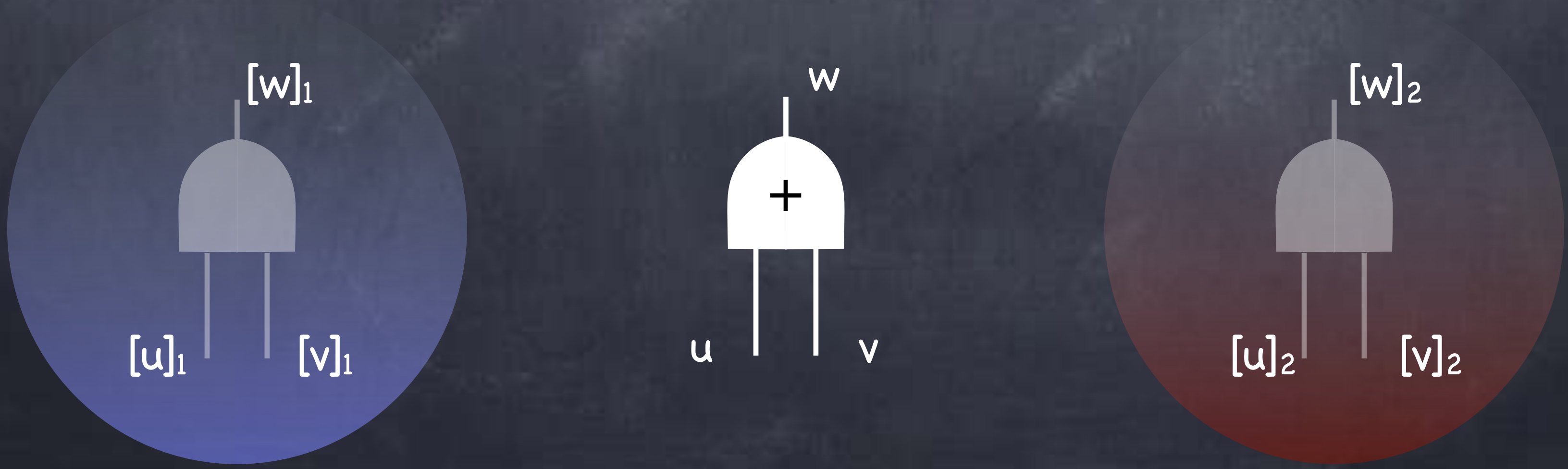
- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

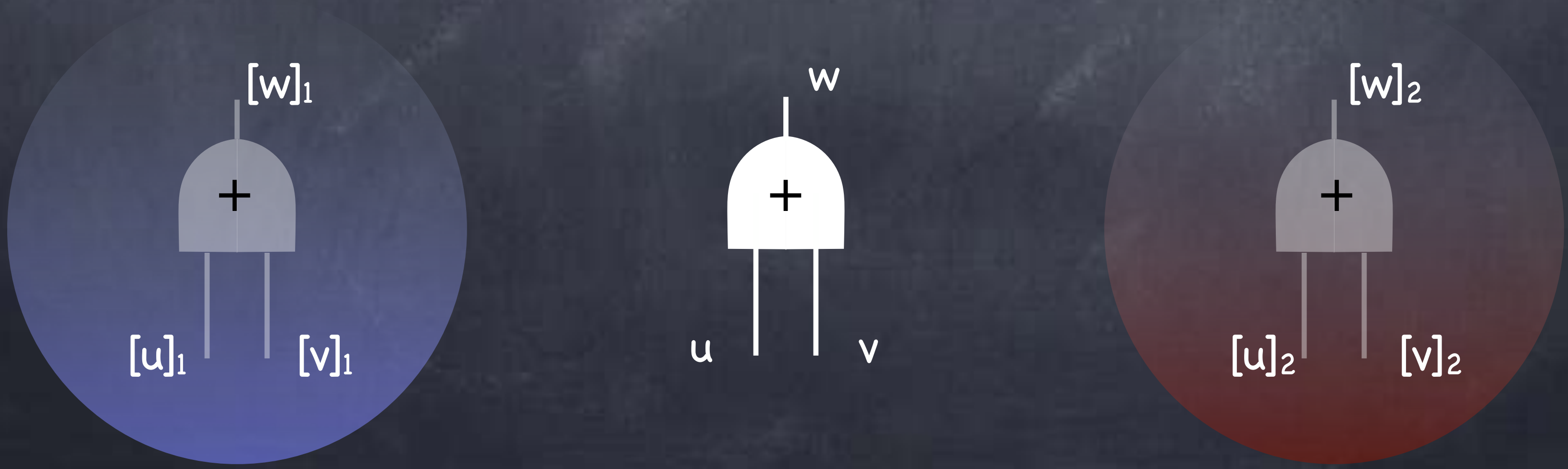
Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.
- $w = u + v$: Each one locally computes $[w]_i = [u]_i + [v]_i$

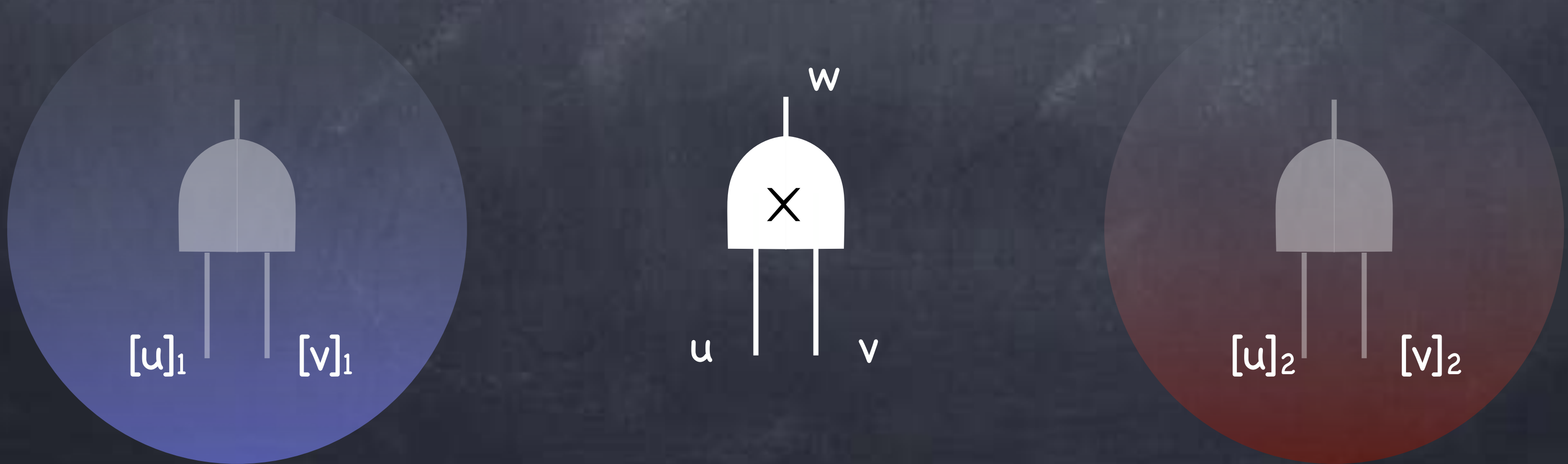


Computing on Shares

- Let gates be + & × (XOR & AND for Boolean circuits)
- Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.
- $w = u + v$: Each one locally computes $[w]_i = [u]_i + [v]_i$

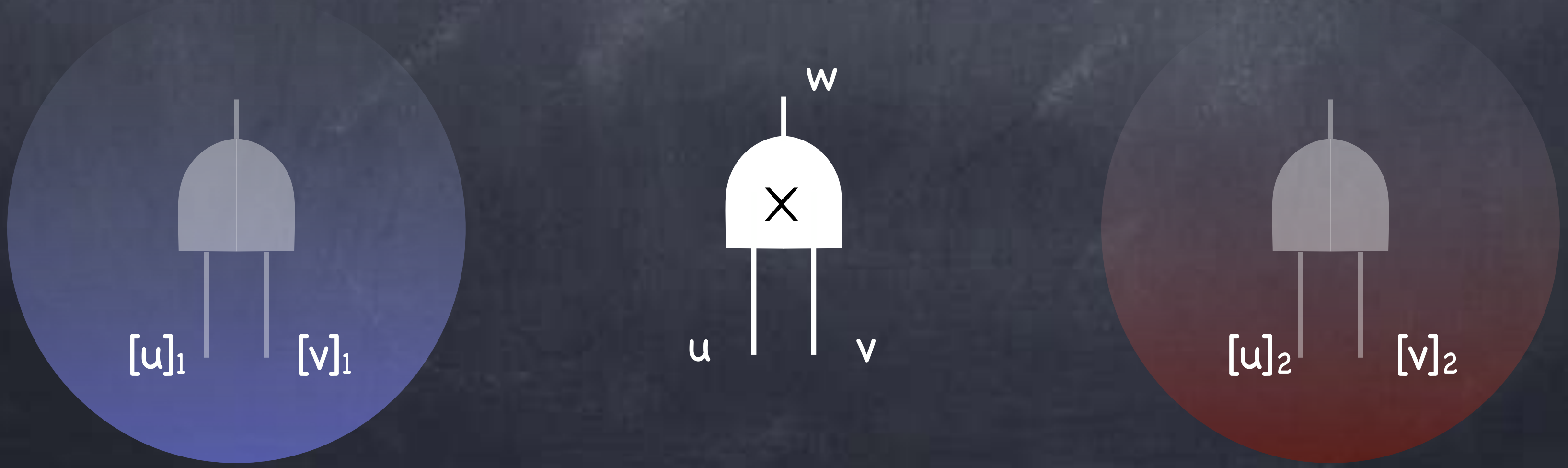


Computing on Shares



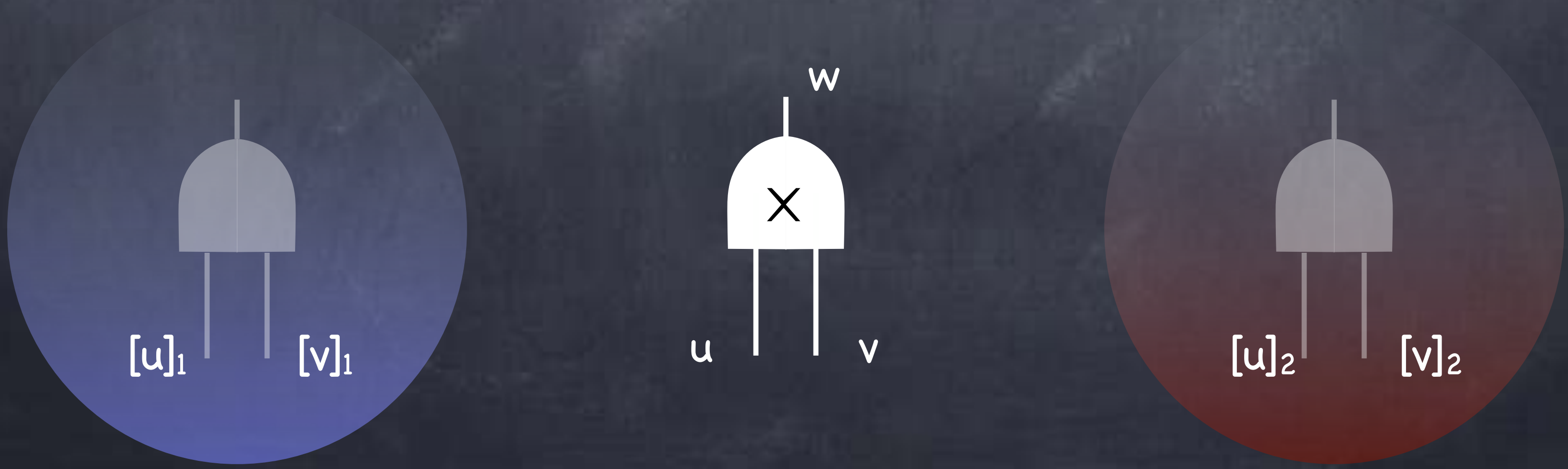
Computing on Shares

- What about $w = u \times v$?



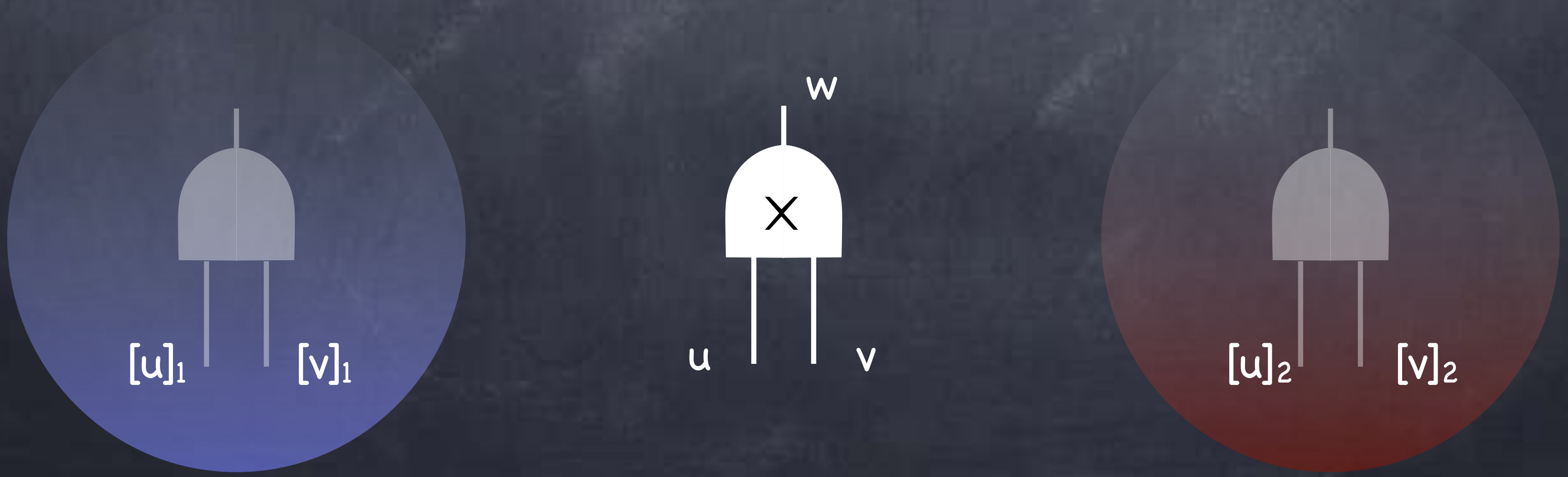
Computing on Shares

- What about $w = u \times v$?
  - Want $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$



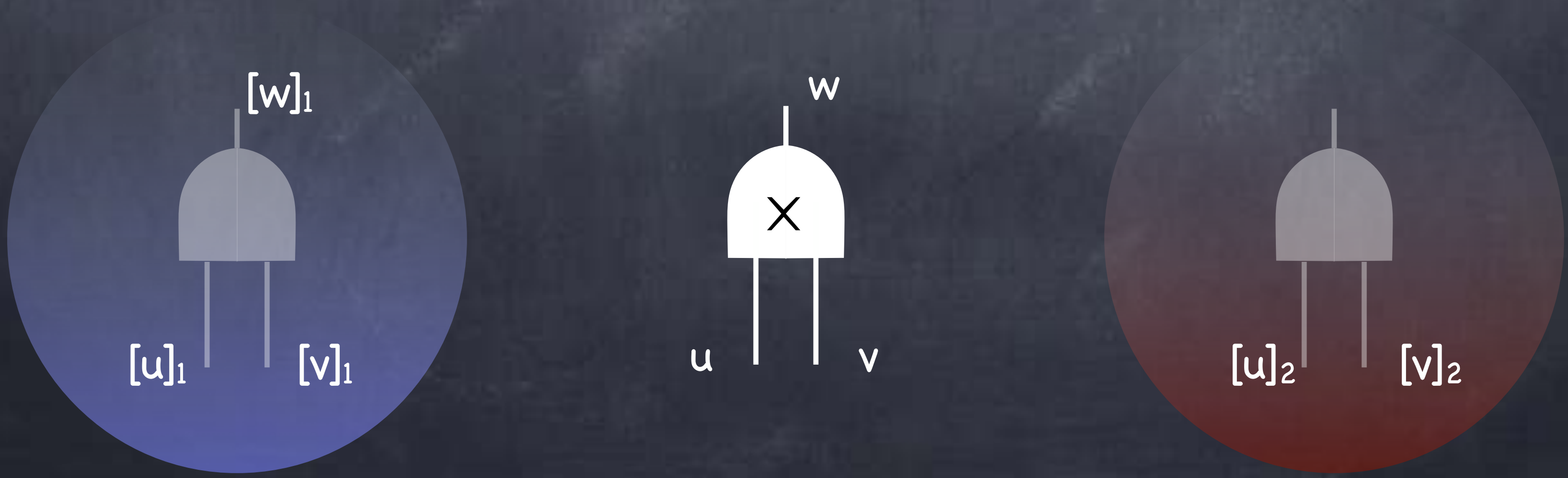
Computing on Shares

- What about $w = u \times v$?
  - Want $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive protocol for small functions



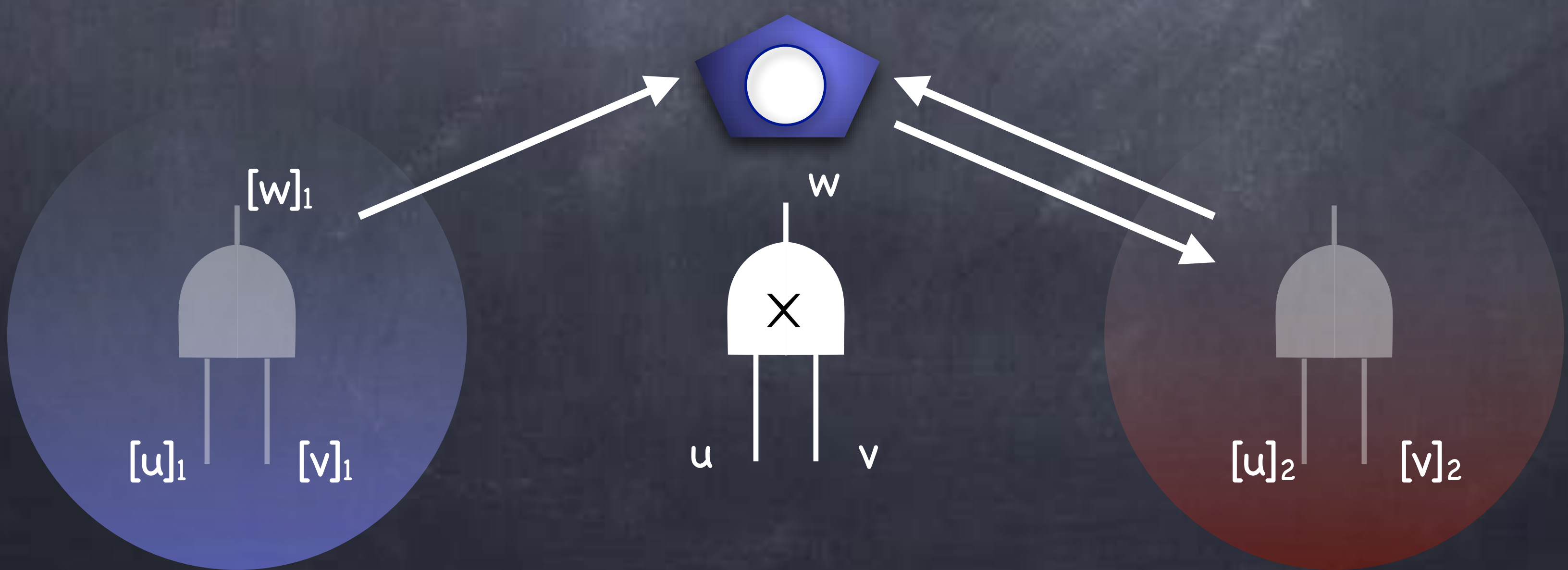
Computing on Shares

- What about $w = u \times v$?
  - Want $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive protocol for small functions



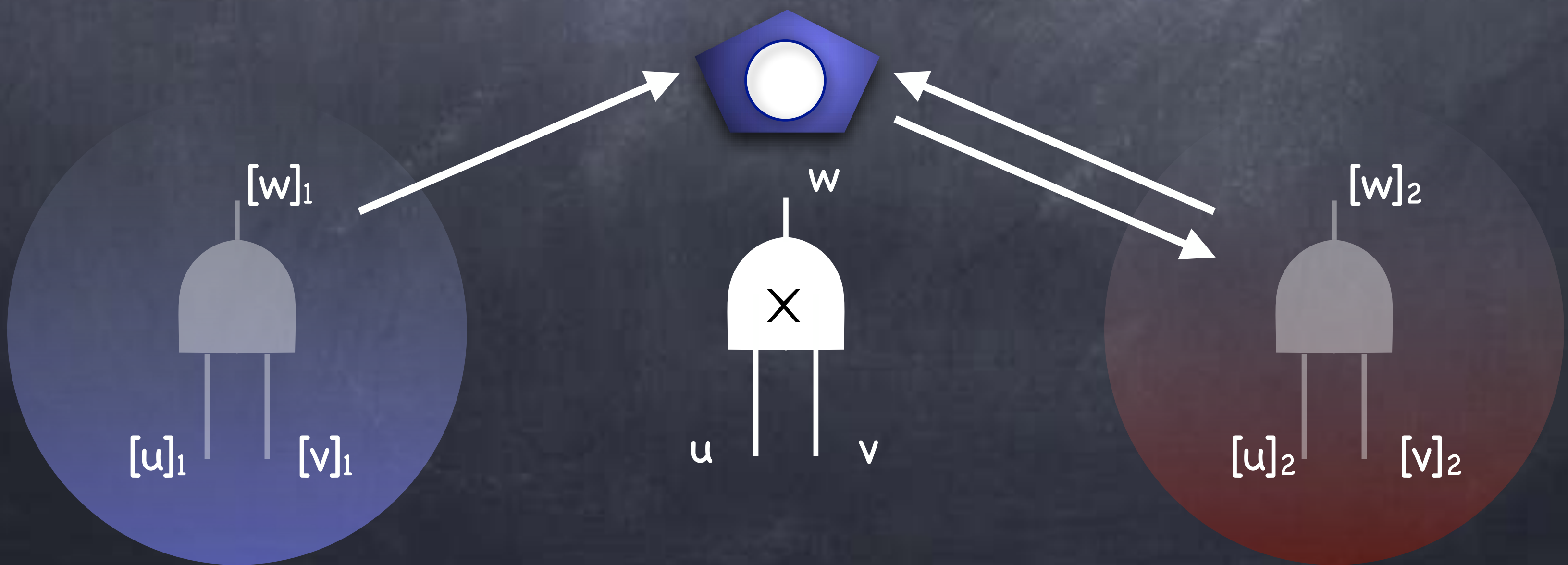
Computing on Shares

- What about $w = u \times v$?
  - Want $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive protocol for small functions



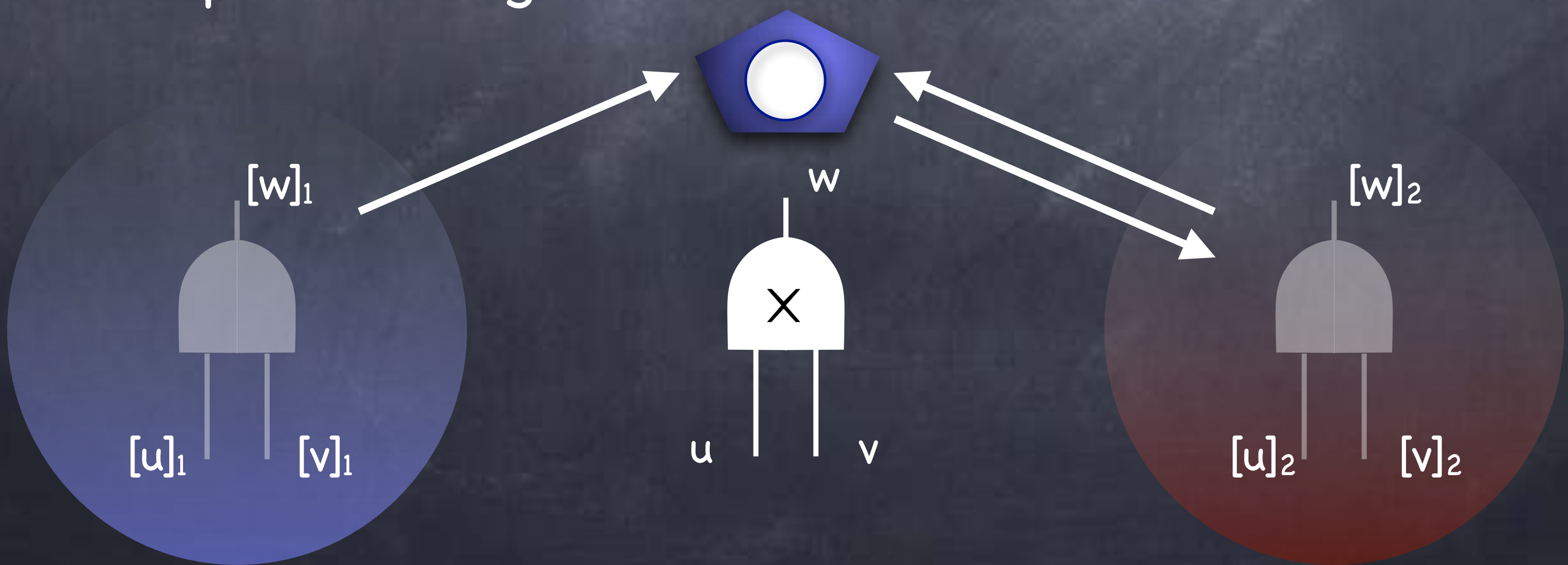
Computing on Shares

- What about $w = u \times v$?
  - Want $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive protocol for small functions



Computing on Shares

- What about $w = u \times v$?
  - Want $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
  - Alice picks $[w]_1$. Can let Bob compute $[w]_2$ using the naive protocol for small functions
    - Bob's input is $([u]_2,[v]_2)$. Over the binary field, this requires a single 1-out-of-4 OT.



GMW: many parties

GMW: many parties

- m-way sharing: $s = [s]_1 + \ldots + [s]_m$

Allows security against arbitrary number of corruptions

GMW: many parties

- m-way sharing: $s = [s]_1 + \ldots + [s]_m$
- Addition, local as before

Allows security against arbitrary number of corruptions

GMW: many parties

- m-way sharing: $s = [s]_1 +...+ [s]_m$

Allows security against arbitrary number of corruptions

- Addition, local as before
- Multiplication: For $w = u \times v$
  $[w]_1 +..+ [w]_m = ([u]_1 +..+ [u]_m) \times ([v]_1 +..+ [v]_m)$
  - Party i computes $[u]_i[v]_i$
  - For every pair (i,j), i≠j, Party i picks random a_{ij} and lets Party j securely compute b_{ij} s.t. $a_{ij} + b_{ij} = [u]_i[v]_j$ using the naive protocol (a single 1-out-of-2 OT)
  - Party i sets $[w]_i = [u]_i[v]_i + \Sigma_j (a_{ij} + b_{ji})$

GMW: with active corruption

- Original GMW approach: Use **Zero Knowledge proofs** (next time) to force the parties to run the protocol honestly
  - Needs (passive-secure) OT to be implemented using a protocol
- Alternate constructions give information-theoretic reduction to OT, starting from passive-secure GMW
  - Recent approach: pre-compile the circuit

Passive-Secure GMW: Closer Look

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Computing shares a_{12}, b_{12} s.t. $a_{12} + b_{12} = [u]_1 \cdot [v]_2$:

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Computing shares a_{12}, b_{12} s.t. $a_{12} + b_{12} = [u]_1 \cdot [v]_2$:
  - Alice picks a_{12} and sends $(-a_{12}, [u]_1 - a_{12})$ to OT. Bob sends $[v]_2$ to OT.

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Computing shares a_{12}, b_{12} s.t. $a_{12} + b_{12} = [u]_1 \cdot [v]_2$:
  - Alice picks a_{12} and sends $(-a_{12}, [u]_1-a_{12})$ to OT. Bob sends $[v]_2$ to OT.
  - What if Alice sends arbitrary (x,y) to OT? Effectively, setting $a_{12} = -x$, $[u]_1' = y-x$.

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Computing shares a_{12}, b_{12} s.t. $a_{12} + b_{12} = [u]_1 \cdot [v]_2$:
  - Alice picks a_{12} and sends $(-a_{12}, [u]_1 - a_{12})$ to OT. Bob sends $[v]_2$ to OT.
  - What if Alice sends arbitrary (x,y) to OT? Effectively, setting $a_{12} = -x$, $[u]_1' = y-x$.
  - What Bob sends to OT is $[v]_2'$

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Computing shares a_{12}, b_{12} s.t. $a_{12} + b_{12} = [u]_1 \cdot [v]_2$:
  - Alice picks a_{12} and sends $(-a_{12}, [u]_1 - a_{12})$ to OT. Bob sends $[v]_2$ to OT.
  - What if Alice sends arbitrary (x,y) to OT? Effectively, setting $a_{12} = -x$, $[u]_1' = y-x$.
  - What Bob sends to OT is $[v]_2'$
- i.e., arbitrary behavior of Alice & Bob while sharing $[u]_1 \cdot [v]_2$ correspond to them locally changing their shares $[u]_1$ and $[v]_2$

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Arbitrary behavior of Alice while sharing $[u]_1 \cdot [v]_2$ and $[u]_2 \cdot [v]_1$ corresponds to her locally changing her shares of u and v

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Arbitrary behavior of Alice while sharing $[u]_1 \cdot [v]_2$ and $[u]_2 \cdot [v]_1$ corresponds to her locally changing her shares of u and v
  - Alice changing her share from $[u]_1$ to $[u]_1'$ is effectively changing u to $u+\Delta_u$, where $\Delta_u = [u]_1' - [u]_1$ depends only on her own view

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Arbitrary behavior of Alice while sharing $[u]_1 \cdot [v]_2$ and $[u]_2 \cdot [v]_1$ corresponds to her locally changing her shares of u and v
  - Alice changing her share from $[u]_1$ to $[u]_1'$ is effectively changing u to $u+\Delta_u$, where $\Delta_u = [u]_1' - [u]_1$ depends only on her own view
- Over all effect: a corrupt party can arbitrarily add Δ_u and Δ_v to wires u and v before multiplication

Passive-Secure GMW: Closer Look

- Multiplication: $[w]_1 + [w]_2 = ([u]_1 + [u]_2) \times ([v]_1 + [v]_2)$
- Arbitrary behavior of Alice while sharing $[u]_1 \cdot [v]_2$ and $[u]_2 \cdot [v]_1$ corresponds to her locally changing her shares of u and v
  - Alice changing her share from $[u]_1$ to $[u]_1'$ is effectively changing u to $u+\Delta_u$, where $\Delta_u = [u]_1' - [u]_1$ depends only on her own view
- Over all effect: a corrupt party can arbitrarily add Δ_u and Δ_v to wires u and v before multiplication
- Also, can add deltas to all input and output wires

Active-Secure Variant of Basic GMW

Active-Secure Variant of Basic GMW

- Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit

Active-Secure Variant of Basic GMW

- Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit
- Idea: "Compile" the circuit such that any additive attack amounts to error (w.h.p.), resulting in random output

Active-Secure Variant of Basic GMW

- Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit
- Idea: "Compile" the circuit such that any additive attack amounts to error (w.h.p.), resulting in random output
- Additive Manipulation Detecting (AMD) circuits

Active-Secure Variant of Basic GMW

- Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit
- Idea: "Compile" the circuit such that any additive attack amounts to error (w.h.p.), resulting in random output
- Additive Manipulation Detecting (AMD) circuits
    - Extension of "AMD codes"

Active-Secure Variant of Basic GMW

- Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit
- Idea: "Compile" the circuit such that any additive attack amounts to error (w.h.p.), resulting in random output
- Additive Manipulation Detecting (AMD) circuits
  - Extension of "AMD codes"
    - e.g. encode x as a vector (x, r, xr) where r is random from a large field. Additive attacks (without knowing r) detected unless $(x+\delta_1)(r+\delta_2) = (xr+\delta_3)$: i.e., $\delta_1 \cdot r + x \cdot \delta_2 + \delta_1 \cdot \delta_2 = \delta_3$. Unlikely unless $\delta_1 = 0$.

Honest Majority

Honest Majority

- So far, arbitrary number of parties can be corrupted (in particular, secure 2-party computation, when one party is corrupt)
  - But needed to rely on OT

Honest Majority

- So far, arbitrary number of parties can be corrupted (in particular, secure 2-party computation, when one party is corrupt)
  - But needed to rely on OT
- Up Next: Adversary can corrupt any set of less than t parties out of m parties (e.g., t = n/2, t=n/3)
  - Then, can get (UC) security just from secure communication channels
  - Bonus (omitted): Can ask for guaranteed output delivery

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)
  - Similar result by Chaum, Crepeau, Damgård (1988)

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)
  - Similar result by Chaum, Crepeau, Damgård (1988)
- Again, gate-by-gate evaluation of shared wire-values

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)
  - Similar result by Chaum, Crepeau, Damgård (1988)
- Again, gate-by-gate evaluation of shared wire-values
- Idea 1: Use a linear secret-sharing scheme that allows local multiplication

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)
  - Similar result by Chaum, Crepeau, Damgård (1988)
- Again, gate-by-gate evaluation of shared wire-values
- Idea 1: Use a linear secret-sharing scheme that allows local multiplication
  - Result can use a different linear secret-sharing scheme

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)
  - Similar result by Chaum, Crepeau, Damgård (1988)
- Again, gate-by-gate evaluation of shared wire-values
- Idea 1: Use a linear secret-sharing scheme that allows local multiplication
  - Result can use a different linear secret-sharing scheme
  - Will rely on $< n/2$ corruption

BGW: Passive Security

- Ben-Or, Goldwasser, Wigderson (1988)
  - Similar result by Chaum, Crepeau, Damgård (1988)
- Again, gate-by-gate evaluation of shared wire-values
- Idea 1: Use a linear secret-sharing scheme that allows local multiplication
  - Result can use a different linear secret-sharing scheme
  - Will rely on $< n/2$ corruption
- Idea 2: Can move from one linear secret-sharing scheme to another securely

BGW

- Idea 1: Use a linear secret-sharing that allows local multiplication, but resulting in shares in a different linear secret-sharing scheme

BGW

- Idea 1: Use a linear secret-sharing that allows local multiplication, but resulting in shares in a different linear secret-sharing scheme
- **Shamir secret-sharing** using degree $\lfloor(n-1)/2\rfloor$ polynomials (privacy against < n/2 (≤ degree+1) corruption)
  - $[s]_i = \sigma(x_i)$ where polynomial σ s.t. $\sigma(0) = s$
  - $\sigma(0)$ = a linear combination of degree+1 shares $\{\sigma(x_i)\}_i$

BGW

- Idea 1: Use a linear secret-sharing that allows local multiplication, but resulting in shares in a different linear secret-sharing scheme

- **Shamir secret-sharing** using degree $\lfloor (n-1)/2 \rfloor$ polynomials (privacy against < n/2 (≤ degree+1) corruption)
  - $[s]_i = \sigma(x_i)$ where polynomial σ s.t. $\sigma(0) = s$
  - $\sigma(0)$ = a linear combination of degree+1 shares $\{\sigma(x_i)\}_i$

- Multiplying two such polynomials for secrets s, t: $\pi = \sigma.\tau$. Then $[s\cdot t]_i = \pi(x_i) = \sigma(x_i)\cdot\tau(x_i)$ and $\pi(0) = s\cdot t$

BGW

- Idea 1: Use a linear secret-sharing that allows local multiplication, but resulting in shares in a different linear secret-sharing scheme

- **Shamir secret-sharing** using degree $\lfloor (n-1)/2 \rfloor$ polynomials (privacy against < n/2 (≤ degree+1) corruption)
  - $[s]_i = \sigma(x_i)$ where polynomial σ s.t. $\sigma(0) = s$
  - $\sigma(0)$ = a linear combination of degree+1 shares $\{\sigma(x_i)\}_i$
- Multiplying two such polynomials for secrets s, t: $\pi = \sigma.\tau$. Then $[s\cdot t]_i = \pi(x_i) = \sigma(x_i)\cdot\tau(x_i)$ and $\pi(0) = s\cdot t$
- Degree of $\pi \leq n-1$: $\pi(0)$ reconstructible from n shares

BGW

- Idea 2: Can move from linear secret-sharing scheme A to linear secret-sharing scheme B securely

BGW

- Idea 2: Can move from linear secret-sharing scheme A to linear secret-sharing scheme B securely
- Given shares $(a_1, \ldots, a_n) \leftarrow \text{Share}_A(s)$
- Share each a_i using scheme B: $(b_{i1},\ldots,b_{in}) \leftarrow \text{Share}_B(a_i)$
- Locally each party j reconstructs using scheme A: $b_j \leftarrow \text{Recon}_A(b_{1j},\ldots,b_{nj})$

BGW

- Idea 2: Can move from linear secret-sharing scheme A to linear secret-sharing scheme B securely
- Given shares $(a_1, \ldots, a_n) \leftarrow \text{Share}_A(s)$
- Share each a_i using scheme B: $(b_{i1},\ldots,b_{in}) \leftarrow \text{Share}_B(a_i)$
- Locally each party j reconstructs using scheme A: $b_j \leftarrow \text{Recon}_A\ (b_{1j},\ldots,b_{nj})$
- Claim: $\text{Recon}_B(b_1,\ldots,b_n) = s$

BGW

- Idea 2: Can move from linear secret-sharing scheme A to linear secret-sharing scheme B securely
- Given shares $(a_1, \ldots, a_n) \leftarrow \text{Share}_A(s)$
- Share each a_i using scheme B: $(b_{i1},\ldots,b_{in}) \leftarrow \text{Share}_B(a_i)$
- Locally each party j reconstructs using scheme A: $b_j \leftarrow \text{Recon}_A(b_{1j},\ldots,b_{nj})$
- Claim: $\text{Recon}_B(b_1,\ldots,b_n) = s$
  - For any linear f, $\text{Recon}_B(\, f(\, \text{Share}_B(\vec{a})\,)\,) = f(\vec{a})$

BGW

- Idea 2: Can move from linear secret-sharing scheme A to linear secret-sharing scheme B securely
- Given shares $(a_1, \ldots, a_n) \leftarrow \text{Share}_A(s)$
- Share each a_i using scheme B: $(b_{i1},\ldots,b_{in}) \leftarrow \text{Share}_B(a_i)$
- Locally each party j reconstructs using scheme A: $b_j \leftarrow \text{Recon}_A(b_{1j},\ldots,b_{nj})$
- Claim: $\text{Recon}_B(b_1,\ldots,b_n) = s$
  - For any linear f, $\text{Recon}_B(\ f(\ \text{Share}_B(\vec{a})\ )\ ) = f(\vec{a})$
  - Recon_A is a linear function

Honest Majority

- Can't tolerate (passive) corruption of n/2 parties unless functionality (passive) trivial for 2-party
- Can't tolerate active corruption of n/3 parties (even for "broadcast") if guaranteed output delivery needed

Honest Majority

- Can't tolerate (passive) corruption of n/2 parties unless functionality (passive) trivial for 2-party
- Can't tolerate active corruption of n/3 parties (even for "broadcast") if guaranteed output delivery needed
- More generally, guaranteed output delivery not possible if:
  - set of parties can be partitioned into three sets, $S_1 \cup S_2 \cup S_3$ such that S_1, S_2 (separately) may be passively corrupt, and S_3 may be actively corrupt

Active Security

- Active security with abort:
  - Run (passive-secure) BGW on an AMD circuit of the function
  - Each party will accept the output only if the output verifies

Active Security

- Active security with abort:
  - Run (passive-secure) BGW on an AMD circuit of the function
  - Each party will accept the output only if the output verifies
  - In IDEAL, adversary can cause selective abort, after seeing its own output

Active Security

- Active security with abort:
  - Run (passive-secure) BGW on an AMD circuit of the function
  - Each party will accept the output only if the output verifies
  - In IDEAL, adversary can cause selective abort, after seeing its own output
- Guaranteed output-delivery possible using alternate methods

Active Security

- Active security with abort:
  - Run (passive-secure) BGW on an AMD circuit of the function
  - Each party will accept the output only if the output verifies
  - In IDEAL, adversary can cause selective abort, after seeing its own output
- Guaranteed output-delivery possible using alternate methods
  - Needs $t < n/3$. (Or $t < n/2$, but using a secure broadcast channel)

Summary

Summary

- Using pair-wise OT (and no computational assumption)

Summary

- Using pair-wise OT (and no computational assumption)
  - Passive security and Active security possible against arbitrarily many corruptions

Summary

- Using pair-wise OT (and no computational assumption)
  - Passive security and Active security possible against arbitrarily many corruptions
- Using Broadcast channel and point-to-point channels

Summary

- Using pair-wise OT (and no computational assumption)
  - Passive security and Active security possible against arbitrarily many corruptions
- Using Broadcast channel and point-to-point channels
  - Active security (with guaranteed output delivery) possible against $t < n/2$ corruptions

Summary

- Using pair-wise OT (and no computational assumption)
  - Passive security and Active security possible against arbitrarily many corruptions
- Using Broadcast channel and point-to-point channels
  - Active security (with guaranteed output delivery) possible against $t < n/2$ corruptions
- Using only point-to-point channels

Summary

- Using pair-wise OT (and no computational assumption)
  - Passive security and Active security possible against arbitrarily many corruptions
- Using Broadcast channel and point-to-point channels
  - Active security (with guaranteed output delivery) possible against $t < n/2$ corruptions
- Using only point-to-point channels
  - Passive security possible against $t < n/2$ corruptions

Summary

- Using pair-wise OT (and no computational assumption)
  - Passive security and Active security possible against arbitrarily many corruptions
- Using Broadcast channel and point-to-point channels
  - Active security (with guaranteed output delivery) possible against $t < n/2$ corruptions
- Using only point-to-point channels
  - Passive security possible against $t < n/2$ corruptions
  - Active security (with guaranteed output delivery) possible against $t < n/3$ corruptions