# Notebook: Distributed Computation

Henry Blanchette

Spring, 2021

# 1 Security Definitions

## 1.1 Indistinguishable Distributions

**Definition 1.** A function $f : \mathbb{N} \to \mathbb{B}$ is a **negligible function** when

$$\forall c \in \mathbb{Z} : \exists N : \forall n \leqslant N : f(k) \leqslant \frac{1}{k^c}$$

**Definition 2.** A set $X = \{X(k,a)\}_{kin\mathbb{N},a\in\mathbb{B}*}$ is a **distribution ensemble**.

**Definition 3.** Distributions $X$ and $Y$ are **perfectly indistinguishable** when

$$\forall k, a : X(k,a) = Y(k,a)$$

**Definition 4.** Distributions $X$ and $Y$ are **statistically indistinguishable** with $\epsilon$**-closeness** when

$$\forall k, a : \sigma X(k,a), Y(k,a) \leqslant \epsilon(k)$$

where

$$\sigma(X,Y) = \frac{1}{2}\sum_a |\mathsf{Pr}\left[X = a\right] - \mathsf{Pr}\left[Y = a\right]|$$

and $\epsilon$ is a negligible function/

**Definition 5.** Distributions $X$ and $Y$ are **computationally indistinguishable** if for all polytime distinguishers $D$, there exists a negligible function $\epsilon$ such that for all $a, z$:

$$|\mathsf{Pr}\left[D(k,a,z,X(k,a))\right] = 1 - \mathsf{Pr}\left[D(k,a,z,Y(k,a)) = 1\right]| \leqslant \epsilon(k)$$

where $z$ is an auxiliary input.

## 1.2 Secure Computation

Fix a randomized function $f$ from $n$ inputs to $n$ outputs. How should we define security? Important concepts:

- real/ideal paradigm – real-world execution should be "close" to ideal
- define real world
- define ideal world
- defined notion of "closeness"

**Definition 6. Real-world** execution is defined by a protocol $\Pi$ with adversary $A$. A **passive** adversary follows the protocol. An **active** adversary behaves arbitrarily.

$$\mathsf{Real}_{\Pi,A}(k,\vec{x},z)$$

**Definition 7. Ideal-world** execution of $f$ takes into account:

- item substitution
- computation
- output
- aborting?

$$\mathsf{Ideal}_{\Pi,A}(k,\vec{x},z)$$

**Definition 8.** A protocol $\Pi$ is $t$**-secure** if for all probabilistic polytime (PPT) adversaries $A$ corrupting at most $t$ parties, there exists a polytime $S$ corrupting at most $t$ parties such that

$$\left\{(\mathsf{View}_{\Pi,A}(k,\vec{x},z), \mathsf{Out}^H_{\Pi,A}(k,\vec{x},z))\right\} \approx \left\{(\mathsf{Out}^S_{f,S}(k,\vec{x},z), \mathsf{Out}^H_{f,S}(k,\vec{x},z))\right\}$$

are computationally indistinguishable.

## 1.3 Hybrid World

**Definition 9.** A **hybrid-world** protocol $\Pi$ evaluating $f$ is secure if for all PPT $A$, there exists a PPT $S$ such that

$$\mathsf{Hybrid}_{\Pi,A}^{f_1,f_n}(k,\vec{x},z) \approx \mathsf{Ideal}_{f,S}(k,\vec{x},z)$$

**Theorem 1.** If $f_1,\ldots,f_m$ are secure protocol for computing $f_1,\ldots,f_m$, and if $\Pi$ is a secure protocol for computing $f$ in the $(f_1,\ldots,f_m)$-hybrid world, then the composed protocol $\Pi^{f_1,\ldots,f_m}$ is a secure protocol for $f$.

### 1.3.1 Parallel Composition

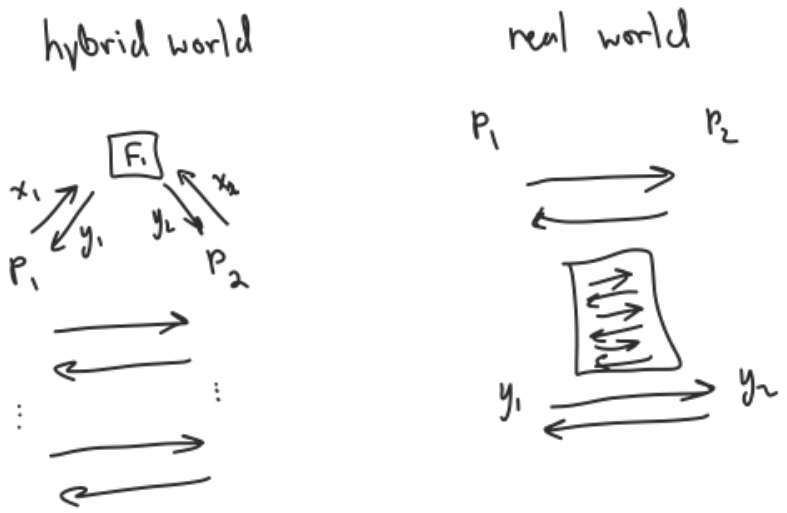


Figure 1: Parallel composition

# 2 Oblivious Transfer

## 2.1 1-out-of-$N$ Oblivious Transfer ($\mathsf{OT}$)

Denote $1-out-of-\mathsf{NOT}$ by $\mathsf{OT}_1^N$.

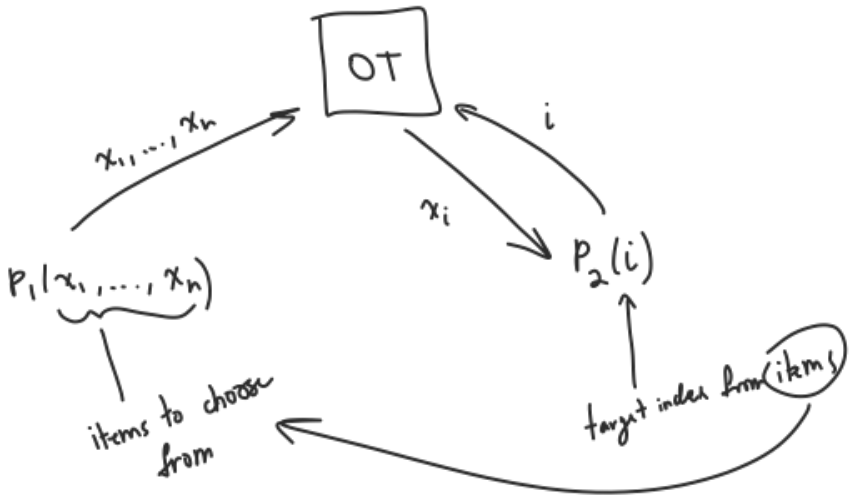


Figure 2: Schematic for $\mathsf{OT}_1^N$

**Protocol 1. Semi-honest** $OT_1^N$. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a CPA-secure public-key encryption scheme. Then the following protocol computes semi-honest $\mathsf{OT}_1^N$.

**Theorem 2.** The above protocol securely computes $\mathsf{OT}_1^N$ in the semi-honest model, given some assumptions specified in the proof

*Proof.* Suppose that $P_1$ is corrupted. Then the only adversary $S_1$ to consider has the exact view as real $P_1$.

Suppose that $P_2$ is corrupted. Then construct the following adversary $S_2$: Then need to prove that $\forall x_1, \ldots, x_N, i$, the real and ideal cases are computationally indistinguishable:

$$\left\{(r_{i*}, r_{i\neq i*}, \mathsf{Enc}_{pk_{i*}}(x_{i*}), \mathsf{Enc}_{pk_{i\neq i*}}(x_{i\neq i*}))\right\} \quad \text{(real)}$$

$$\approx \left\{(r_{i*}, r_{i\neq i*}, \mathsf{Enc}_{pk_{i*}}(x_{i*}), \mathsf{Enc}_{pk_{i\neq i*}}(\vec{0}))\right\} \quad \text{(ideal)}$$

Can prove this by assuming a distinguisher $D$ and showing that $D$ contradicts the $CPA$ security premise.

Additionally need assumption on $\mathsf{SampRand}$, **oblivious key sampling**:

$$(r, \mathsf{SampKey}(1^k; r)) \approx (\mathsf{SampRand}(r); r \leftarrow \mathsf{Gen}(1^k))$$

Altogether, prove in two steps and use transitivity of computational indistinguishability:

$$\mathsf{Ideal} \approx \mathsf{Hybrid} \approx \mathsf{Real}$$

□

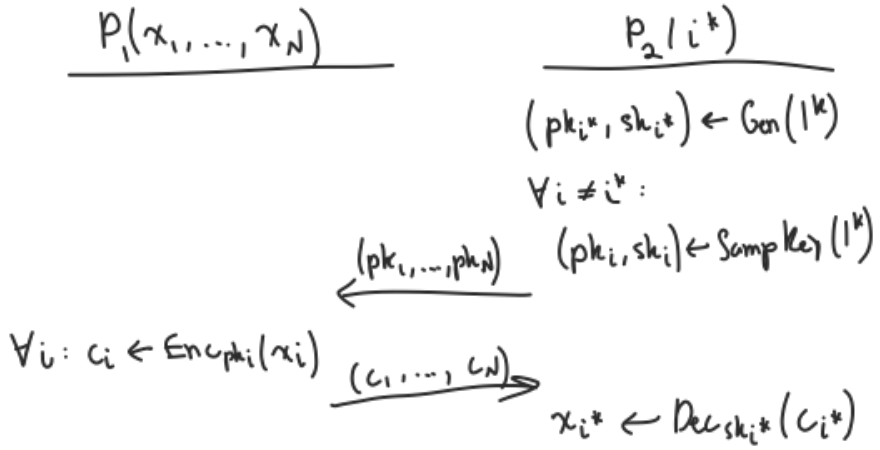


Figure 3: Instance of $\mathsf{OT}_1^N$ with CPA-secure public-key encryption scheme

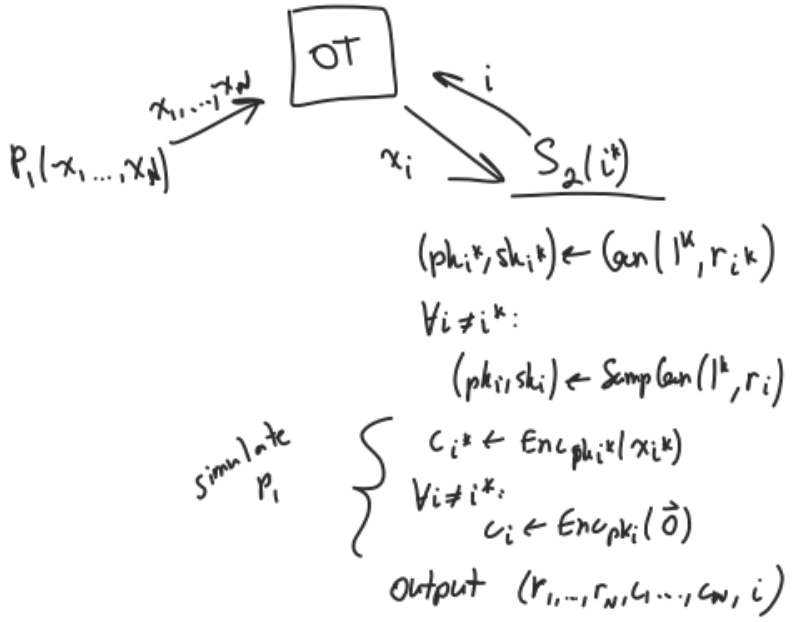


Figure 4: Party 2 corruption case for security proof of theorem 2

# 3 The GMW protocol for MPC

The GMW protocol $(n-1)$-securely computes an $n$-input functionality $F$, for semihonest adversaries, assuming semi-honest $\mathsf{OT}$.

- works in $\mathsf{OT}$-hybrid world; is perfectly secure in $\mathsf{OT}$-hybrid model.
- sufficient to consider deterministic functions, since probabilistic functionalities can be reduced to deterministic functions:

$$f((x_1, r_1), \ldots, (x_n, r_n)) := g(x_1, \ldots, x_n; r_1 \oplus \cdots \oplus r_n)$$

- will consider 2-party case fist, then expand to $n$-party case
- can represent $f$ as a boolean circuit

**Definition 10.** A **secret sharing** of a value $x$ is the choosing of $x_1, \ldots, x_N$ such that $x = x_1 \oplus \cdots x_n$ and then the sending of $x_1, \ldots, x_{N-1}$ to $N-1$ other parties. The $x_N$ is kept for the original sharer, so that if an adversary controls all other parties (such as in the 2-party case), there are still 2 unknowns in the equation $x = x_1 \oplus \cdots x_n$ and so it cannot be solved.

**Protocol 2.** The **GMW** protocol works by taking a functionality $F$, represented by a boolean circuit, and converting each of its logic gates (either AND, OR, or NOT) into secure multi-party computations. These MPCs use a secret sharing of inputs, where each party starts the protocol off by creating a secret sharing of their inputs with each other party. During the MPC of the functionality, only the AND gate case and the end output requires communication between parties, where $\mathsf{OT}$is invoked.

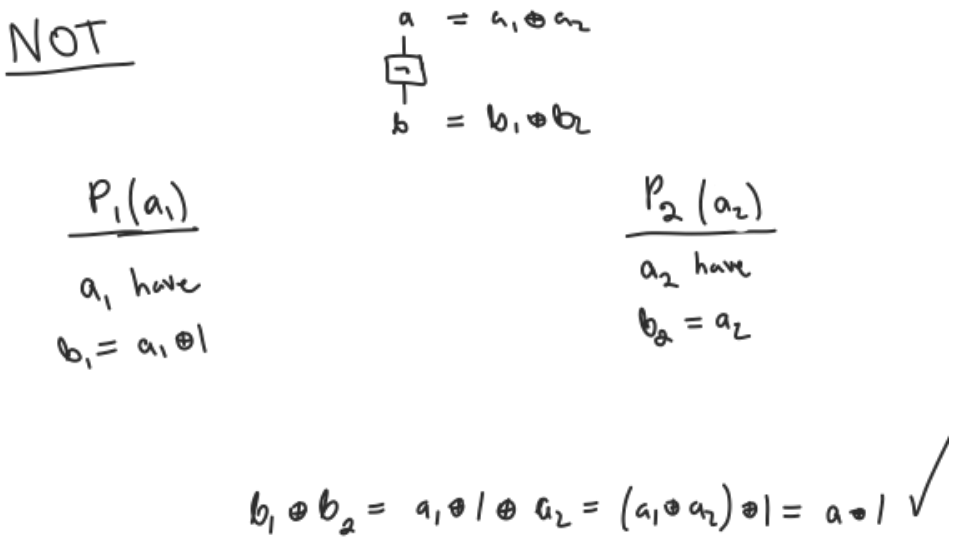


Figure 5: GMW NOT gate

**Output reconstruction.** At the end of computation, each party sends their share of output $i$ to each party that should learn output $i$.

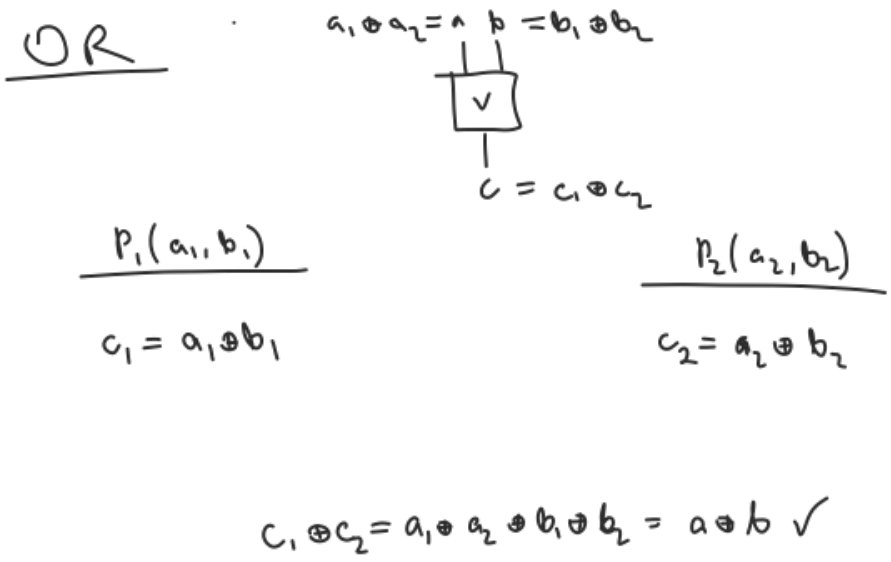


Figure 6: GMW OR gate

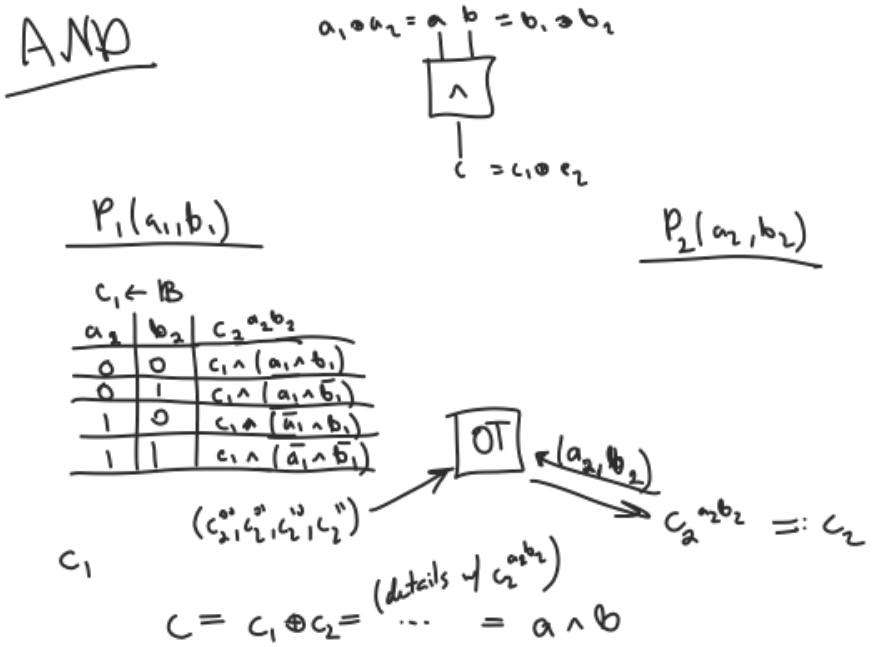


Figure 7: GMW AND gate

### 3.1 Extending BGW

**Semi-honest setting.** BGW has perfect security that tolerates $t < n/2$ corrupt parties. This is optimal since it is impossible to maintain perfect security for $t \geqslant n/2$.

**Malicious setting.** Without a broadcast channel and no prior setup, an extension of BGW can achieve perfect security for $t < n/3$. This is also optimal.

## 3.2 Garbled Circuits

**Definition 11.** A (Yao's) **garbled circuit** can be views as an "encrypted" version of a boolean circuit for some function $f$. One party acts as a garbled-circuit generator, and another party acts as an evaluator of garbled circuits to get the desired result. The garbler associates each wire of the circuit with two cryptographic keys, and ensures that the evaluator learns only one key per wire – in particular, the "correct" key i.e. the key that yields the correct output given the garbler's knowledge of the input wire keys.

For example, suppose the garbler wants to garble an AND gate. Then the garbler chooses keys $a_0, a_1, b_0, b_1, c_0, c_1$, then creates a gate such that if the evaluator only knows keys $a_x$ and $b_y$ then they can only learn key $c_{xy}$, which is the key they need to propogate their evaluated result to the next garbled gate.

**Protocol 3.** A **garbling scheme** for a binary $n$-bit-input circuit $C(x, y) = z$ consists of a pair of functions $(\mathsf{Garble}, \mathsf{Eval})$, where

$$\left(\left\{X_i^0, X_i^1\right\}_{i=1}^n, \left\{Y_i^0, Y_i^1\right\}_{i=1}^n, GC, \left\{Z_i^0, Z_i^1\right\}_{i=1}^n\right) \leftarrow \mathsf{Garble}(C)$$
$$\left\{Z_i\right\}_{i=1}^n \leftarrow \mathsf{Eval}\left(\left\{X_i^{x_i}\right\}, \left\{Y_i^{y_i}\right\}, GC\right)$$

**Correctness.** A garbling scheme is correct when

$$\forall x, y, z : z = C(x, y) \implies \forall i : Z_i = Z_i^{z_i}$$

**Security.** A garbling scheme is secure when there exists a simulator $S$ such that

$$\left(\left\{x_i^0, x_i^1\right\}_{i=1}^n, \left\{y_i^0, y_i^1\right\}_{i=1}^n, GC, \left\{z_i^0, z_i^1\right\}_{i=1}^n\right) \approx \left\{S(y, C(x, y))\right\}_{x,y}$$

where

$$\left\{\left(\left\{x_i^0, x_i^1\right\}_{i=1}^n, \left\{y_i^0, y_i^1\right\}_{i=1}^n, GC, \left\{z_i^0, z_i^1\right\}_{i=1}^n\right) \leftarrow \mathsf{Garble}(C) : \left\{x_i^{x_i}\right\}, \left\{y_i^{y_i}\right\}, GC, \left\{\left(z_i^0, z_i^1\right)\right\}\right\}_{x,y}$$

**Example 1.** Garbling can be achieved using OT.

**Example 2.** Garbling can be achieved using a public-key encryption scheme. Assume that $\mathsf{Dec}_k(\mathsf{Enc}_{k'}(m)) = \perp$ if $k \neq k'$. Given $A \in \left\{A^0, A^1\right\}$ and $B \in \left\{B^0, B^1\right\}$, the evaluator can compare (the correct) $C \in \left\{C^0, C^1\right\}$.

## 3.3 Point-and-Permute

Rather than associating keys with 0 or 1 directly, instead associate a label with each pair of keys. The label is chosen by the garbler and only provided to the evaluator for wires it is supposed to learn.
**Benefits:**

- no need to randomly permute
- evaluator knows which row to decrypt based on labels, which reduced the number of total decryptions by evaluator
- no longer need redundancy in encryption scheme, because
  - cyphertexts can be shorter
  - the $i$th garbled gate can be computed as

$$F_{A^0}(00|i) \oplus F_{B^0}(00|i) \oplus \left[c^{(\lambda_a \wedge \lambda_b) \oplus \lambda_c}, (\lambda_a \wedge \lambda_b) \oplus \lambda_c\right], \text{ etc.}$$

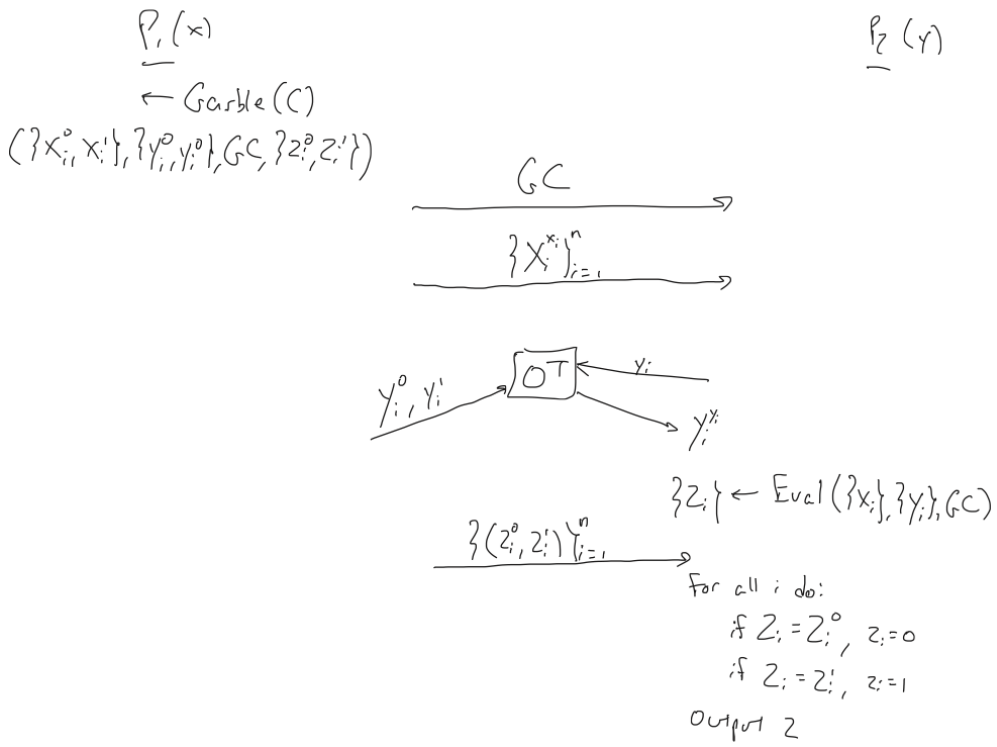


Figure 8: Garbling demonstration using OT

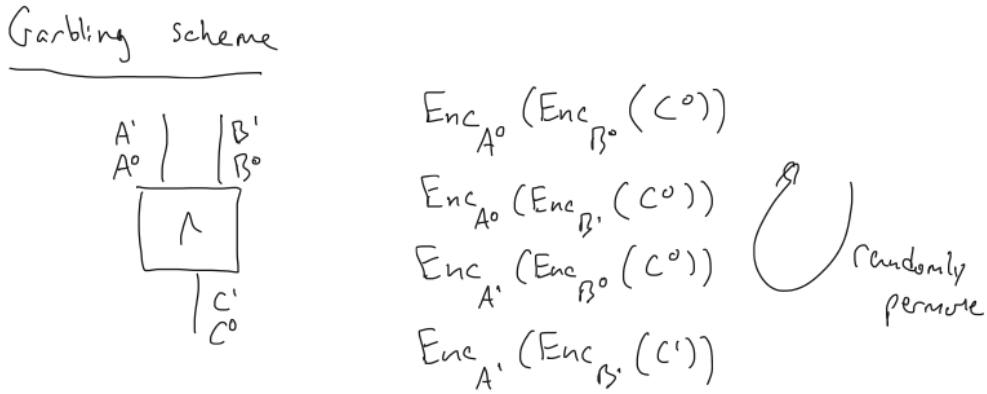


Figure 9: Garbling demonstration using public-key encryption scheme.

### 3.4 Garbled Row Reduction

The following are ways to reduce the number of garbled gates and rows:

- **half-gates**: reduce cyphertexts per garbled gate to 2
- **free XOR**: avoid garbling XOR gates at all
- **global shift** $\Delta$: TODO

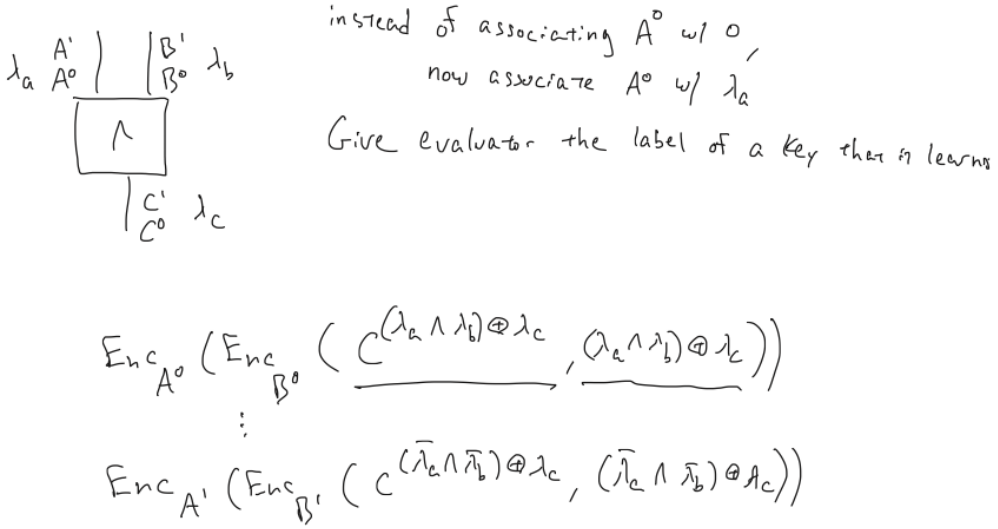


Figure 10: The idea of point-and-permute

# 4 The BMR protocol for MPC

**Protocol 4.** The idea behind **BMR** is to use GMW to compute a garbled circuit. If $F$ can be computed by a constant-depth circuit, then we can securely compute $F$ using the GMW protocol in $O(1)$ rounds.

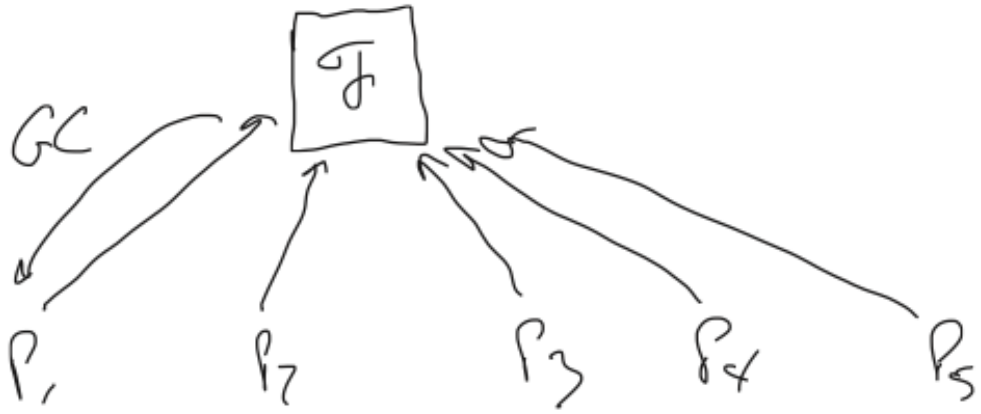


Figure 11: The idea behind the BMR protocol

**Example 3.** A computation of a point-and-permute styled garbled evaluation as BMR

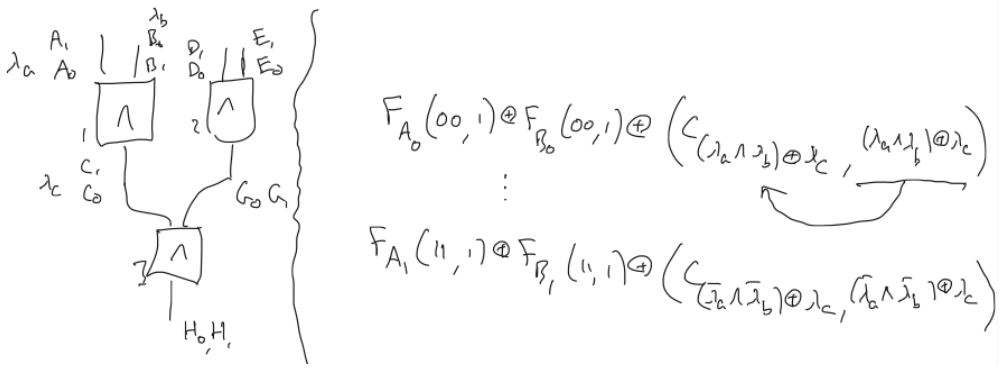

Figure 12: BMR computation example

# 5 The BGW protocol for MPC

**Protocol 5.** The idea behind BGW is to map boolean circuits to arithmetic circuits over a finite field $\mathbb{F}_p$:

$$\begin{aligned} 0 &\mapsto 0 \\ 1 &\mapsto 1 \\ &\mapsto \cdot \\ a \oplus b &\mapsto a + b - 2ab \end{aligned}$$

TODO

**Protocol 6.** The **Shamir secret sharing** protocol is a $t$-out-of-$n$ secrete sharing protocol where

- any $t$ parties can reconstruct the shared value given their shares
- any set of $t - 1$ parties learns no information about shared value

TODO: lecture 7

**Definition 12.** Suppose we have secret sharings $[x], [y]$ and we want to compute $[z]$ where $z = xy$. Assume that parties hold secret sharings $[a], [b], [c]$, which are called a **Beaver triple**. TODO: lecture 7

# 6 Zero Knowledge Proofs

## 6.1 Malicious Security

A malicious adversary may not necessarily follow the protocol i.e. acts arbitrarily.

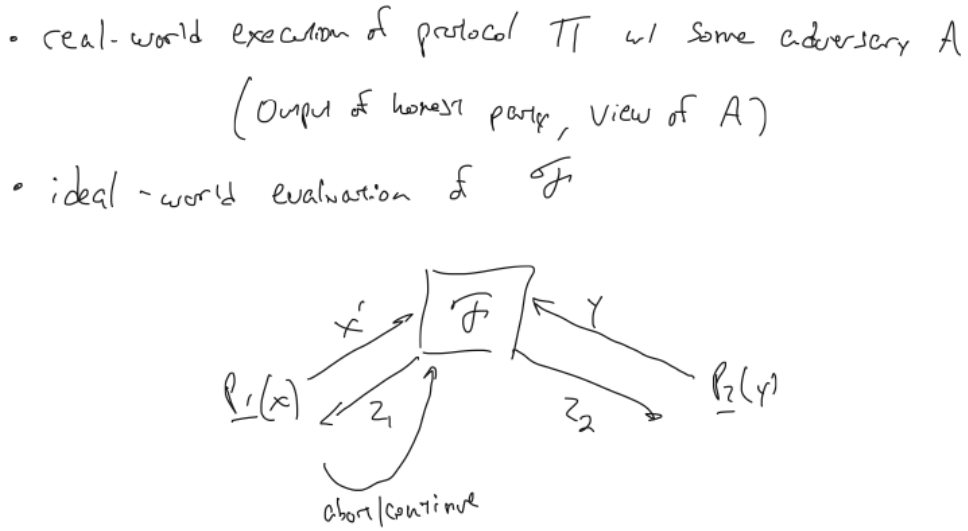


Figure 13: Malicious security

## 6.2 Commitment Schemes

**Definition 13.** A **commitment** scheme satisfies two properties:

- **Binding.** after the commitment phase, there should be at most on $m$ that $S^*$ could validly decommit to
- **Hiding.** after the commitment phase, $R^*$ has no information about $m$

## 6.3 Zero-Knowledge Proofs

Let language $L \in$ NP i.e. there exists an efficient $R_L$ such that $x \in L \iff \exists w : R_L(x, w) = 1$. Examples: SAT, HAM. Note the following setup:

**Definition 14.** A protocol for computing **zero-knowledge proofs** is one that evaluates $\mathcal{F}_{2k}$ against a malicious verifier $V$.

**Definition 15.** A protocol for computing **zero-knowledge proofs of knowledge** (ZKPoK) is one that evaluates $\mathcal{F}_{2k}$ against a malicious prover $P$.

We proceed by giving a ZKPoK protocol for an NP-complete language (HAM), which implies that there is a ZKPoK protocol for each language in NP.

**Theorem 3.** The above protocol is zero knowledge.

*Proof.* TODO: lecture 10 □

**Theorem 4.** The above protocol is a proof of knowledge.

*Proof.* TODO: lecture 10 □

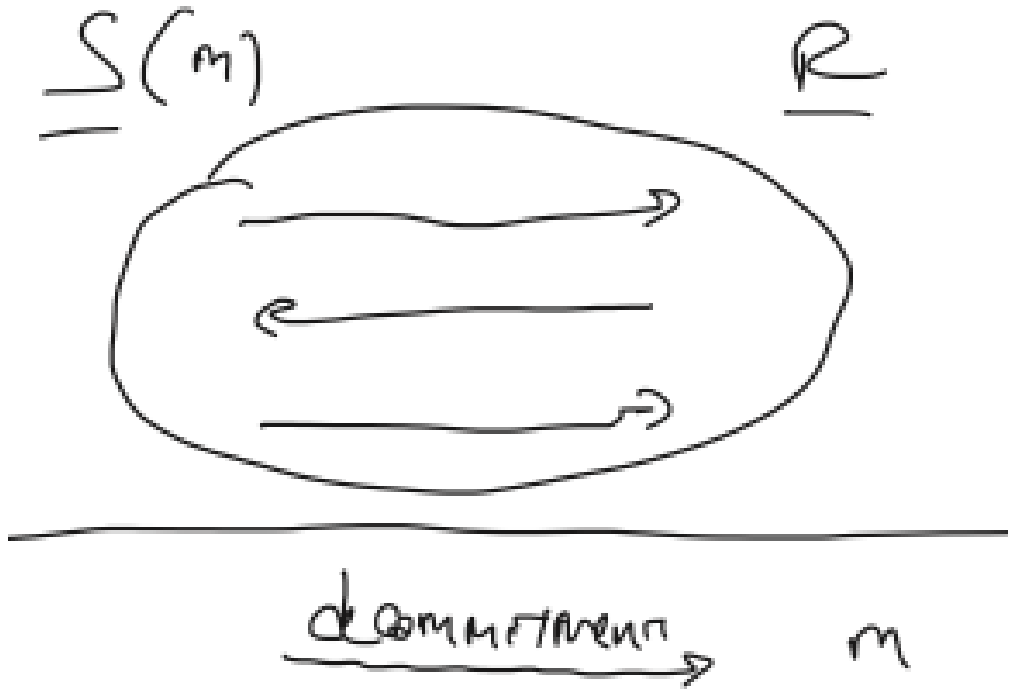


Figure 14: Setup for commitment schemes

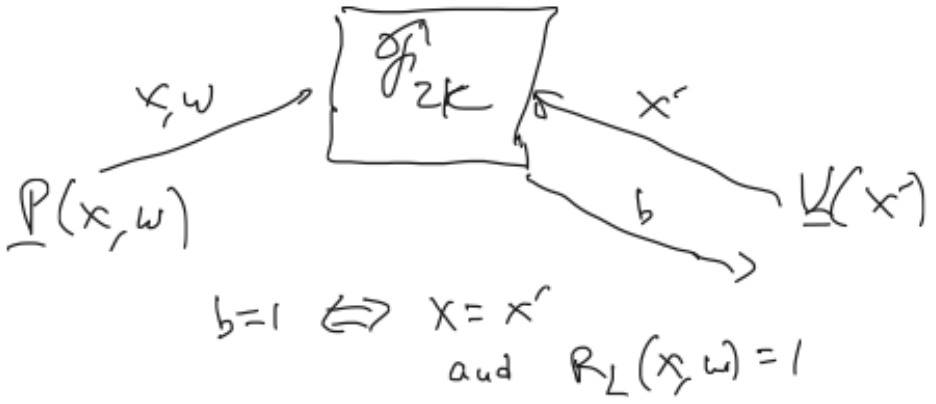


Figure 15: Setup for defining zero-knowledge proofs

Problem: we know how to do a single ZKPoK, but we do not know how to prove that it is ZK in parallel repetition.

**Protocol 7** (KE)**.** The protocol $KE(G)$ runs as follows:

1. run an honest interaction with $P^*$; let challenge be $\vec{b}$
2. if interaction fails, halt

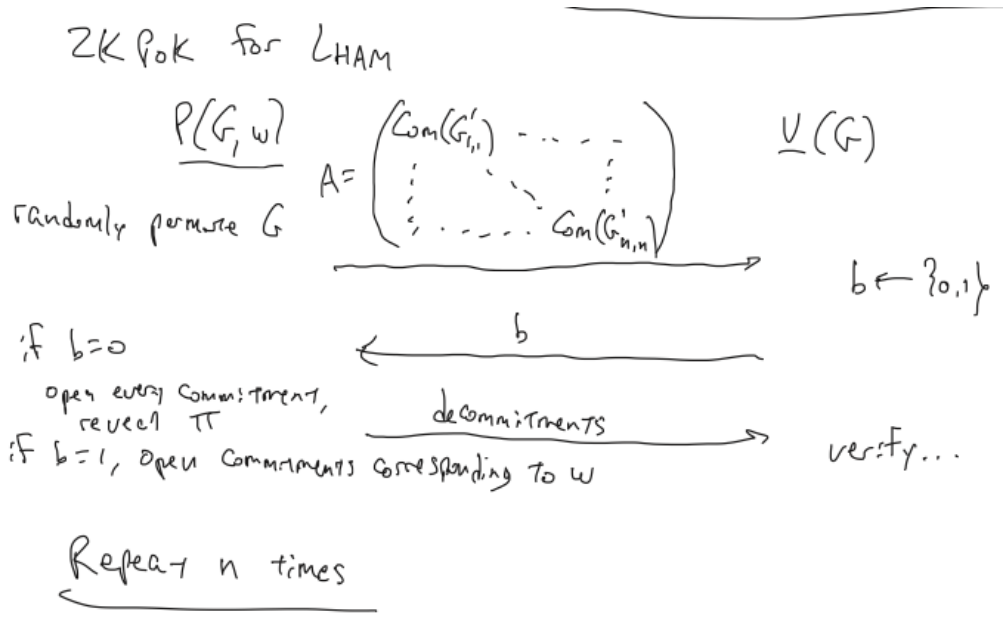


Figure 16: A protocol for ZKPoK for HAM

3. otherwise, set $\vec{b''} = \vec{0}$ and do:
   (a) $\vec{b'} \leftarrow \mathbb{B}^n$
   (b) if $P^*(\vec{b'})$ succeeds then $\vec{b'} \neq \vec{b}$, then break
   (c) if $P^*(\vec{b''})$ succeeds then $\vec{b''} \neq \vec{b}$, then break
   (d) if $\vec{b''} = \vec{1}$, break
   (e) otherwise, increment $\vec{b''}$
4. given two successful executions for distinct challenges, compute a witness $w$

Let $\epsilon$ be the probability that $P^*$ succeeds.

**Theorem 5.** If $\epsilon > 1/2^n$, then KE computes a witness with probability $\epsilon$. Also, KE runs in polynomial expected-time.

*Proof.* TODO: lecture 11 □

**Definition 16.** A PoK has **witness indistinguishability** (WI) if a cheating $V^*$ cannot distinguish which of two possible witnesses $P$ is using.

Note that ZK $\implies$ WI.

**Protocol 8** (Goldeich-Kahan)**.**

**Theorem 6.** The Goldeich-Kahan protocol is ZK.

*Proof.* TODO: lecture 11 □

**Protocol 9** (Feige-Shamir)**.** Let $f$ be a one-way function.

**Theorem 7.** The Feige-Shamir protocol is a ZKPoK.

*Proof.* TODO: lecture 11 □

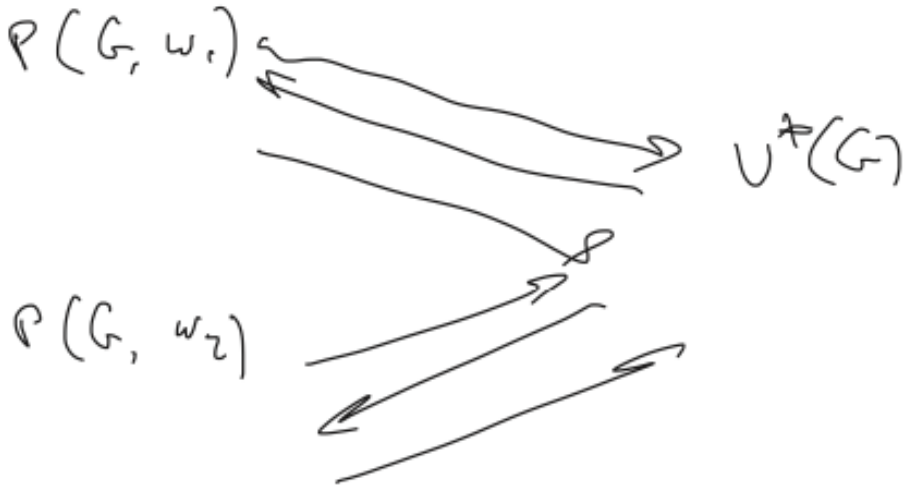


Figure 17: Witness indistinguishability

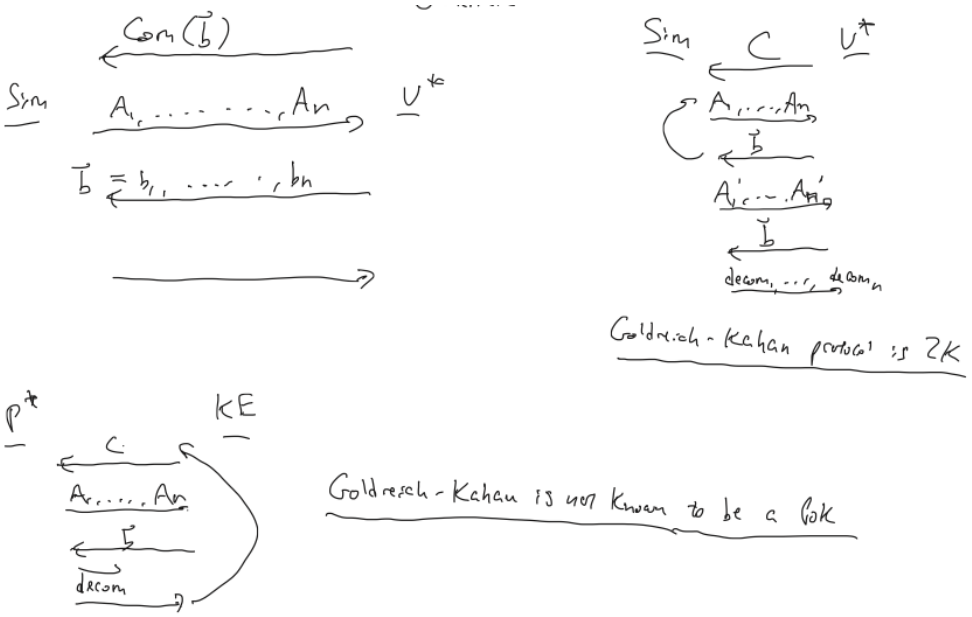


Figure 18: The Goldeich-Kahan protocol

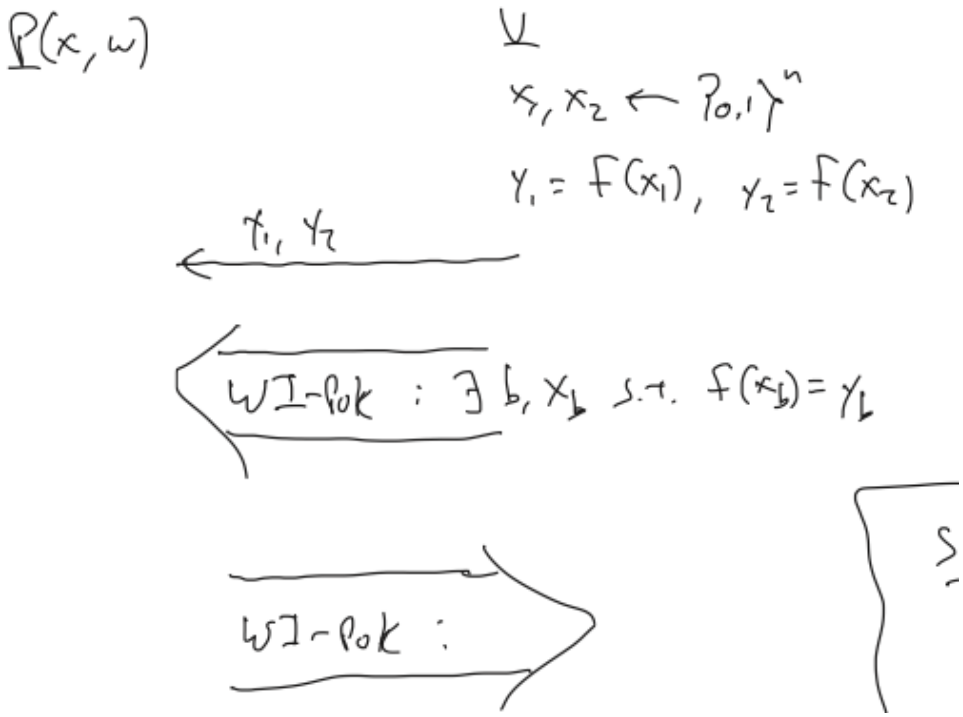


Figure 19: The Feige-Shamir protocol

# 7 GMW Compiler

## 7.1 GMW I Compiler

The goal is to compile any protocol with semi-honest security into a protocol with malicious security, where malicious security is security with abort.

The main idea is that parties run the semi-honest protocol $\Pi$, except that after each step each party gives a ZK proof that they correctly following the protocol. We need to ensure that parties use "good" randomness, and that parties use the same input/randomness throughout.

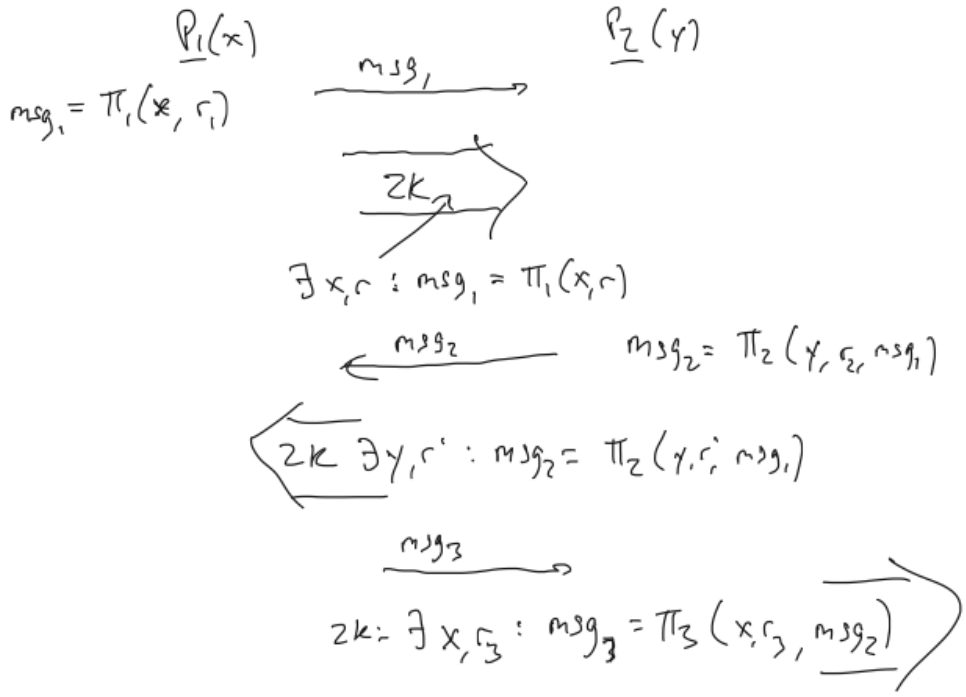


Figure 20: Idea for the GMW I compiler

### 7.1.1 Coin-tossing Protocols

**Protocol 10** (coin-tossing)**.** In this protocol, one party learns a uniformly random value, and the other party gets a commitment to that value.

**Protocol 11** (GMW-I-compiled coin-tossing)**.** This protocol performs the same functionality as the previous coin-tossing protocol, but now with the GMW-I-compiler's assurance that neither party can behave badly.

**Definition 17.** A protocol is **secure with unanimous abort** when it is secure even when

- adversarial parties learn their outputs; then abort or continue
  - if continue, then honest parties get output
  - if halt, honest parties get $\perp$

Security-with-unanimous abort is achievable for $t < n$ given broadcast.

**Definition 18.** A protocol is **secure without abort** *or* **fully secure** when it is secure even when

- all parties send input ot ideal functionality

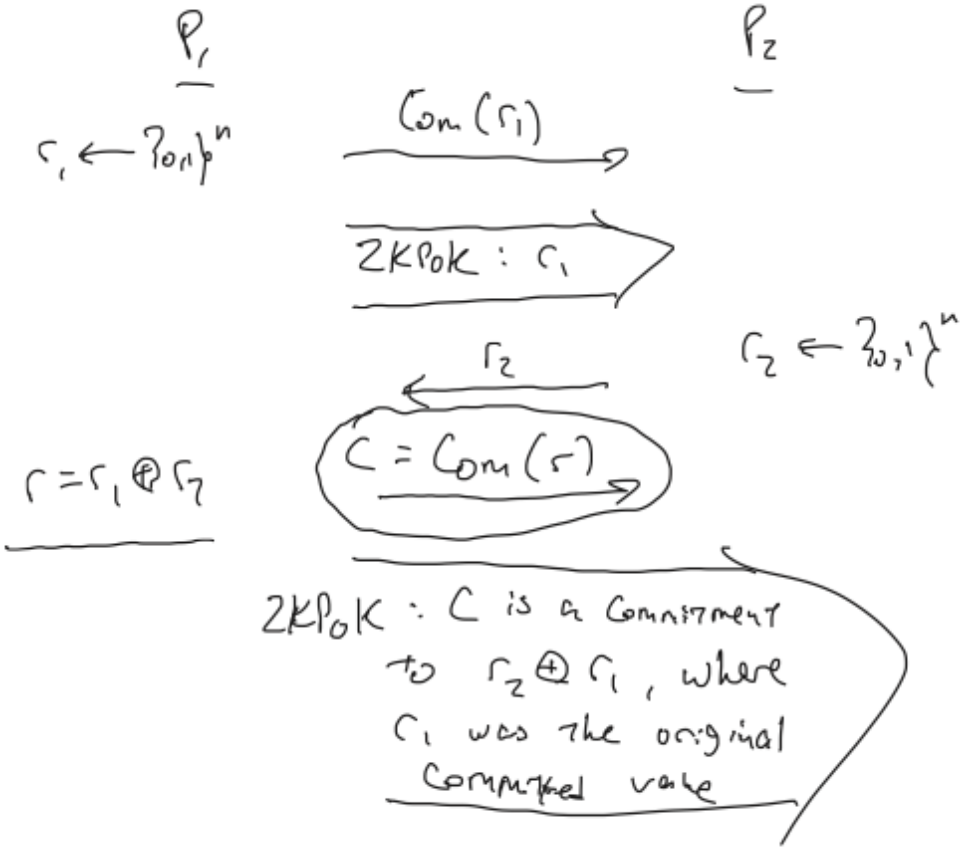


Figure 21: The coin-tossing protocol

- all parties get output

Security-with-abort is achievable for $t < n/2$ given broadcast. But it is not achievable for $t \geqslant n/2$ (in general), even given broadcast.

The GMW I compiler, in the multi-party case, compiles a semi-honest protocol $\Pi$ into a protocol that is secure with unanimous abort.

1. each party commits to its input and gives a ZKPoK of its input over broadcast channel
2. run a multi-party version of coin-tossing
3. run the semi-honest protocol and ZK proof of consistency at every step

## 7.2 GMW II Compiler

**Protocol 12.** Given a semi-honest protocol $\Pi$, the GMW II compiler yields the following protocol:

1. Parties compute $\mathcal{F}_{VSS}$ using a secure-with-abort protocol, once per party. If some $P_i$ misbehaves, kick then out and use a default input in their input's place.
2. Parties do the same for a random version of $\mathcal{F}_{VSS}$, once per party. If some $P_i$ misbehaves, kick them out.
3. Run $\Pi$ using the committed inputs and randomness, giving a ZK proof of correctness after each message. If some $P_i$ fails when giving some ZK proof:

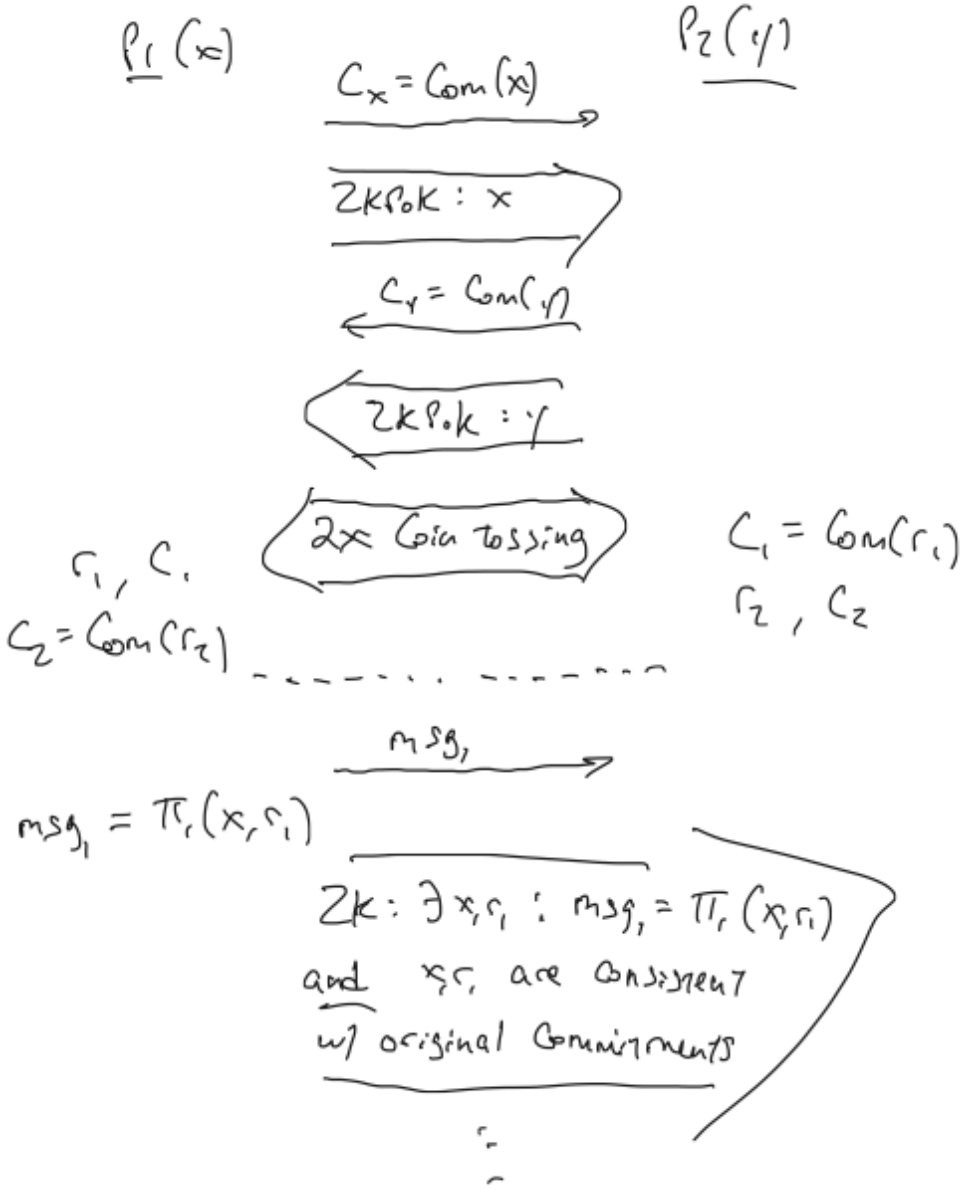


Figure 22: The GMW-I-compiled coin-tossing protocol

(a) each party broadcasts their share of $P_i$'s input/randomness, plus the corresponding $\rho$s

(b) parties reconstruct $P_i$'s input/randomness from $t+1$ correct shares

(c) parties run $\Pi$ on behalf of $P_i$ from then on

### 7.2.1 Broadcast

Parties can realize a broadcast channel using a broadcast protocl.

**Definition 19.** A **broadcast protocol** is a protocol run by parties $P_1, \ldots P_n$ with designated $P^* \in \{P_1, \ldots, P_n\}$ acting as a sender with initial input $m$, and must have two properties:

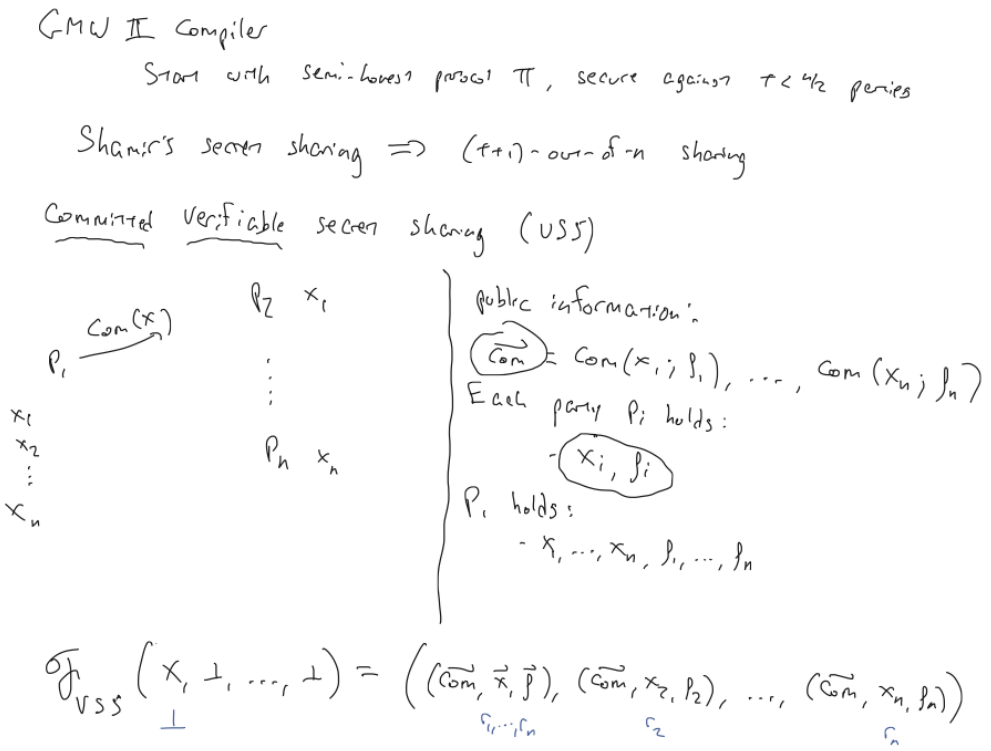


Figure 23: The GMW II compiler

- **Validity.** if $P^*$ is honest, then all honest parties output $m$
- **Consistency.** all honest parties should output the same value


Figure 24: Idea of broadcast protocol

**Definition 20.** A **Byzantine agreement protocol** (BA) is run by parties $P_1, \ldots, P_n$ where each $P_i$ has initial input $m_i$, and must have two properties:

- **Consistency.** all honest parties output the same value
- **Validity.** if all honest parties hold the same input value $m$, then all honest parties output $m$

For $t < n/2$, BA $\implies$ broadcast, and broadcast $\implies$ BA. Additionally, BA only makes sense for $t < n/2$, whereas broadcast makes sense even for $n/2 \leqslant t < n$.

**Lemma 1.** With no prior setup, BA/broadcast are impossible if $t \geqslant n/3$.

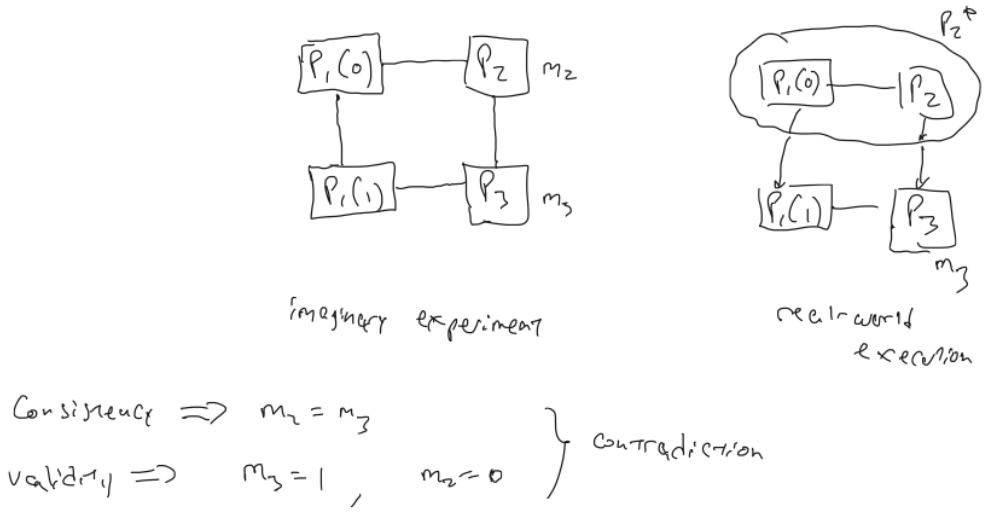


Figure 25: Impossibility of BA/broadcast when $t \geqslant n/3$

*Proof.* □

TODO: lecture 14 has rest of BA/broadcast stuff

# 8 Glossary

**Protocols.**

- **GMW.** multiparty setting, round complexity $O(d)$ where $d$ is the circuit depth (number of non-parallelizable AND gates)
- **Yao.** 2-party setting, round complexity $O(1)$
- **BMR.** multiparty setting, round complexity $O(1)$
- **BGW.** multiparty protocol, round complexity $O(d)$ where $d$ is circuit depth, secure if $t < n/2$, unconditional security, arithmetic circuits of finite field $\mathbb{F}_p$
- **Key-Exchance (KE)**
- **Goldeich-Kahan**
- **Feige-Shamir**
- **GMW I Compiler**
- **GMW II Compiler**
- **Broadcast**
- **Byzantine Agreement**