

# Lecture Notes: Secure Distributed Computation

Henry Blanchette

Spring, 2021

## Contents

<b>1</b>	<b>Security Definitions</b>	<b>3</b>
1.1	Indistinguishable Distributions . . . . .	3
1.2	Secure Computation . . . . .	3
1.3	Hybrid World . . . . .	4
1.3.1	Parallel Composition . . . . .	4
<b>2</b>	<b>Oblivious Transfer</b>	<b>5</b>
2.1	1-out-of- $N$ Oblivious Transfer (OT) . . . . .	5
<b>3</b>	<b>GMW protocol for MPC</b>	<b>7</b>
3.1	Extending BGW . . . . .	8
3.2	Garbled Circuits . . . . .	9
3.3	Point-and-Permute . . . . .	9
3.4	Garbled Row Reduction . . . . .	10
<b>4</b>	<b>BMR protocol for MPC</b>	<b>12</b>
<b>5</b>	<b>BGW protocol for MPC</b>	<b>13</b>
<b>6</b>	<b>Zero Knowledge Proofs</b>	<b>14</b>
6.1	Malicious Security . . . . .	14
6.2	Commitment Schemes . . . . .	14
6.3	Zero-Knowledge Proofs . . . . .	14
<b>7</b>	<b>GMW Compiler</b>	<b>19</b>
7.1	GMW I Compiler . . . . .	19
7.1.1	Coin-tossing Protocols . . . . .	19
7.2	GMW II Compiler . . . . .	20
<b>8</b>	<b>Broadcast and Byzantine Agreement</b>	<b>23</b>
8.1	Phase-King Sub-Protocol for BA . . . . .	23
<b>9</b>	<b>Efficient and Maliciously Secure 2PC</b>	<b>25</b>
9.1	Selective-Failure Attack . . . . .	25
<b>10</b>	<b>Differential Privacy</b>	<b>27</b>
10.1	Laplace Mechanism . . . . .	27
10.2	Application: Synthetic Data . . . . .	28
10.3	PAC Learning . . . . .	28
10.4	Centralized versus Local Models . . . . .	28
10.5	Shuffle Model . . . . .	29

<b>11 Private Data Collection</b>	<b>37</b>
11.1 Prio	38
11.1.1 Prio Protocol Extension 1	38
11.1.2 Prio Protocol Extension 2	38
11.1.3 Prio Protocol Extension 3	38
<b>12 Glossary</b>	<b>41</b>
12.1 Protocols	41

# 1 Security Definitions

## 1.1 Indistinguishable Distributions

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

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

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

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

where

$$\sigma(X, Y) = \frac{1}{2} \sum_a |\Pr X = a - \Pr Y = a|$$

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  $a, z$ :

$$|\Pr D(k, a, z, X(k, a)) = 1 - \Pr D(k, a, z, Y(k, a)) = 1| \leq \epsilon(k)$$

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.

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

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

- item substitution
- computation
- output
- aborting?

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

**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

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

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

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

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

#### 1.3.1 Parallel Composition

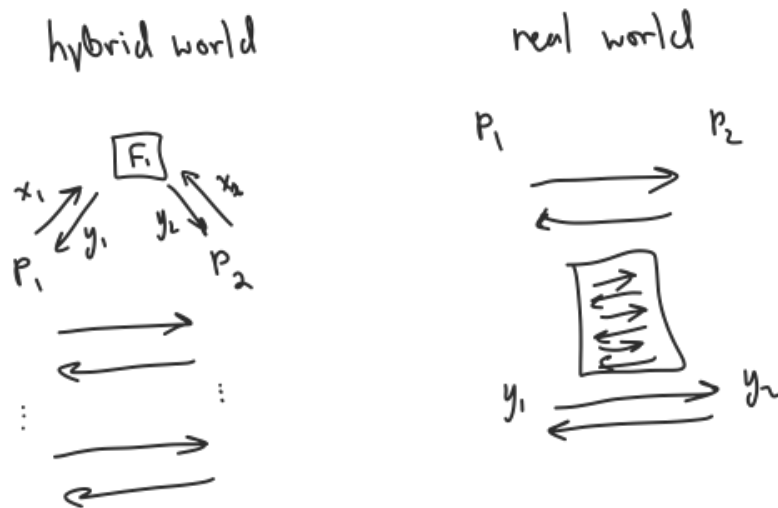


Figure 1: Parallel composition

## 2 Oblivious Transfer

### 2.1 1-out-of- $N$ Oblivious Transfer (OT)

Denote 1-out-of- $N$  OT by  $OT_1^N$ .

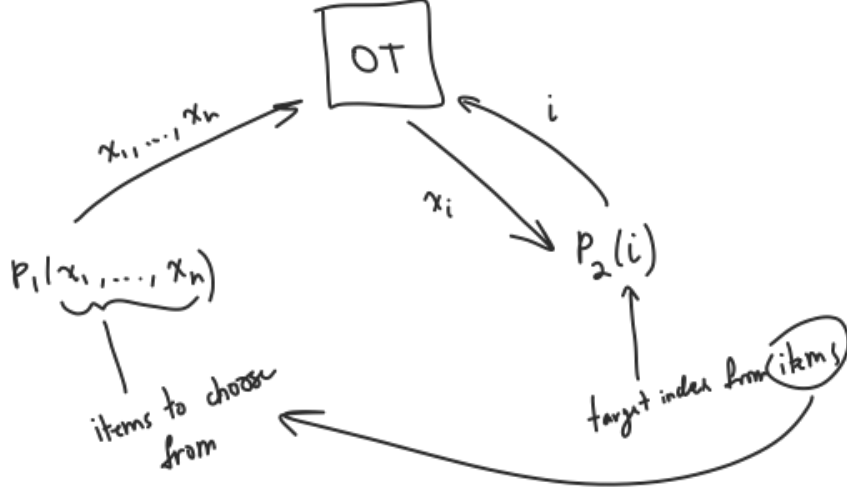


Figure 2: Schematic for  $OT_1^N$

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

**Theorem 2.** The above protocol securely computes  $OT_1^N$  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  $\forall x_1, \dots, x_N, i$ , the real and ideal cases are computationally indistinguishable:

$$\begin{aligned} & \left\{ (r_{i*}, r_{i \neq i*}, \text{Enc}_{pk_{i*}}(x_{i*}), \text{Enc}_{pk_{i \neq i*}}(x_{i \neq i*})) \right\} && \text{(real)} \\ & \approx \left\{ (r_{i*}, r_{i \neq i*}, \text{Enc}_{pk_{i*}}(x_{i*}), \text{Enc}_{pk_{i \neq i*}}(\vec{0})) \right\} && \text{(ideal)} \end{aligned}$$

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

Additionally need assumption on **SampRand**, **oblivious key sampling**:

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

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

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

□

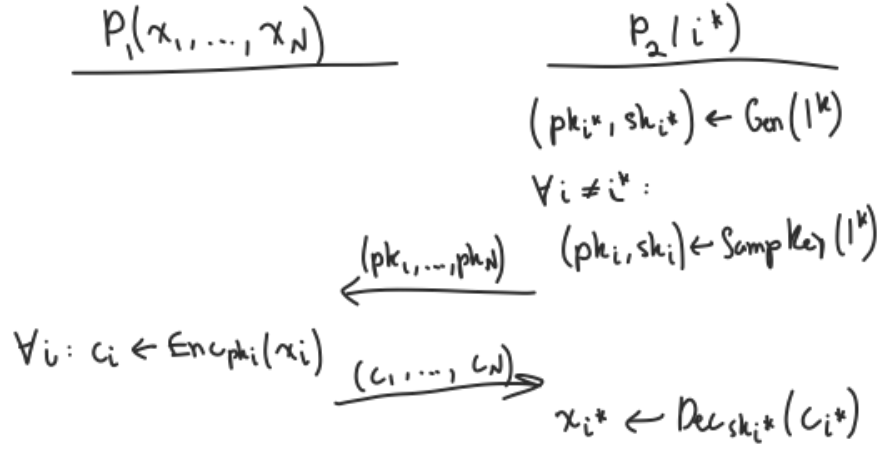


Figure 3: Instance of  $\text{OT}_1^N$  with CPA-secure public-key encryption scheme

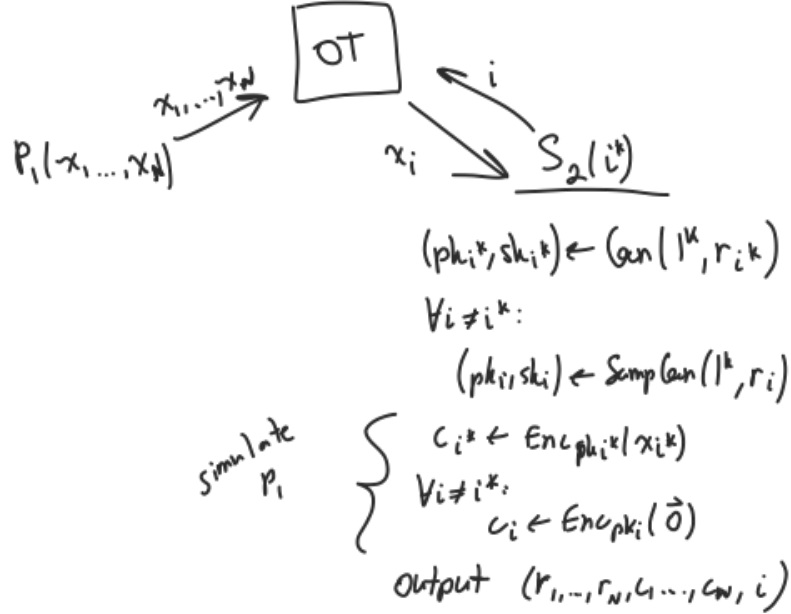


Figure 4: Party 2 corruption case for security proof of theorem 2

### 3 GMW protocol for MPC

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

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

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

- will consider 2-party case first, 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  $x_1, \dots, x_N$  such that  $x = x_1 \oplus \dots \oplus x_N$  and then the sending of  $x_1, \dots, x_{N-1}$  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  $x = x_1 \oplus \dots \oplus x_N$  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 OT is invoked.

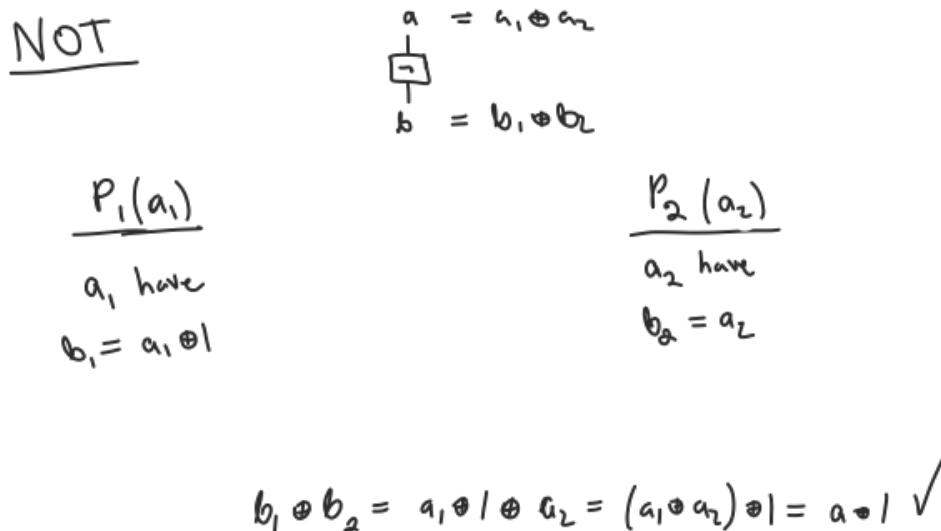


Figure 5: GMW NOT gate

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

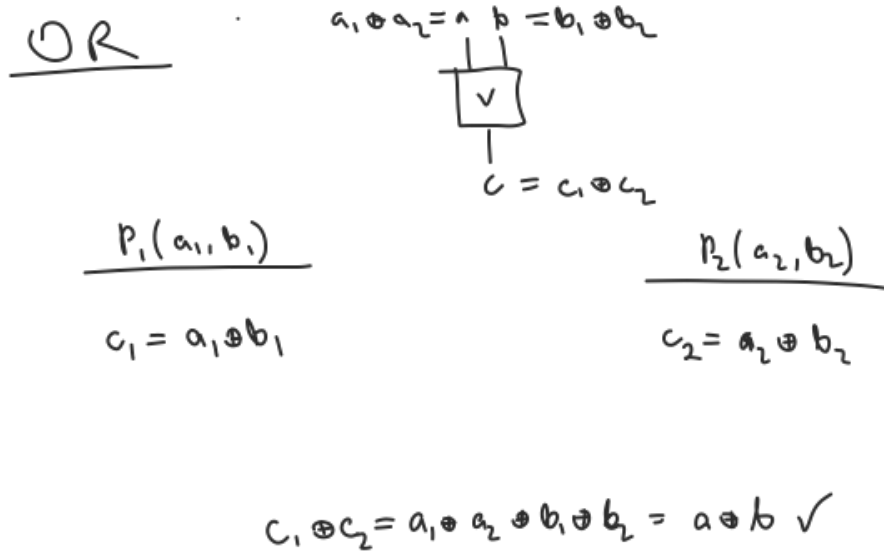


Figure 6: GMW OR gate

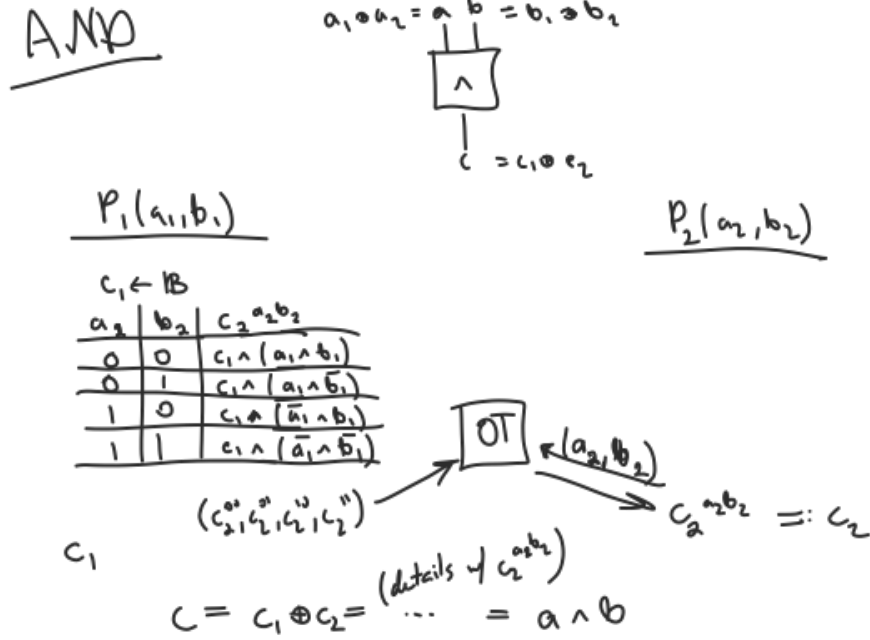


Figure 7: GMW AND gate

### 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  $t \geq n/2$ .



**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 viewed 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 propagate 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 (Garble, Eval), where

$$\begin{aligned} (\{X_i^0, X_i^1\}_{i=1}^n, \{Y_i^0, Y_i^1\}_{i=1}^n, GC, \{Z_i^0, Z_i^1\}_{i=1}^n) &\leftarrow \text{Garble}(C) \\ \{Z_i\}_{i=1}^n &\leftarrow \text{Eval}(\{X_i^{x_i}\}, \{Y_i^{y_i}\}, GC) \end{aligned}$$

**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

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

where

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

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

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

### 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

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

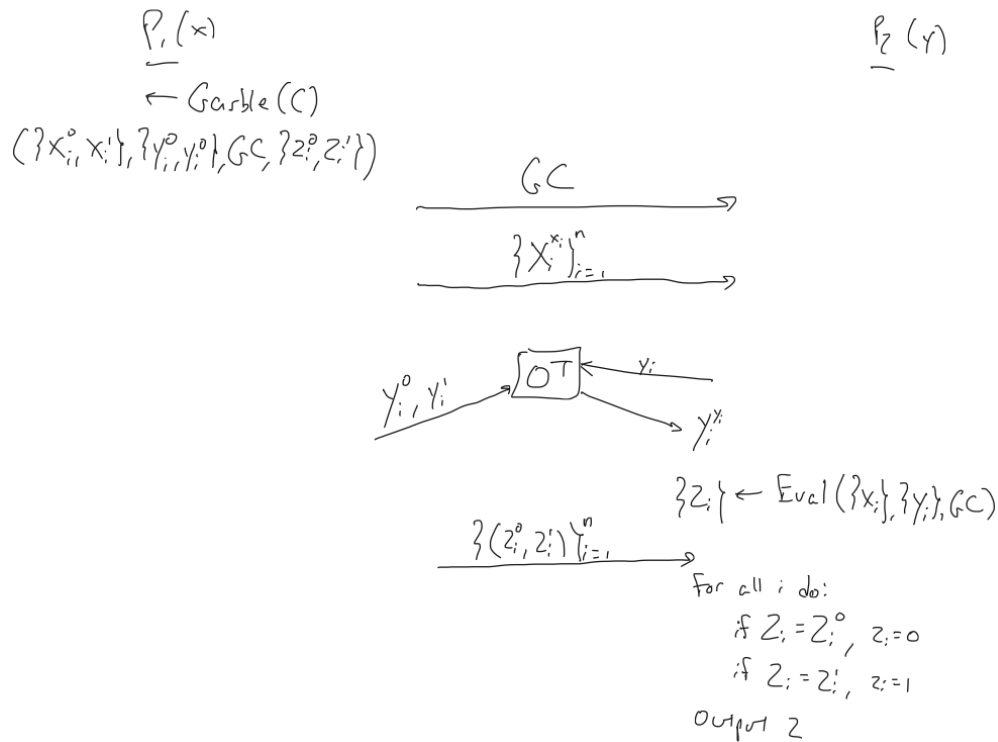


Figure 8: Garbling demonstration using OT

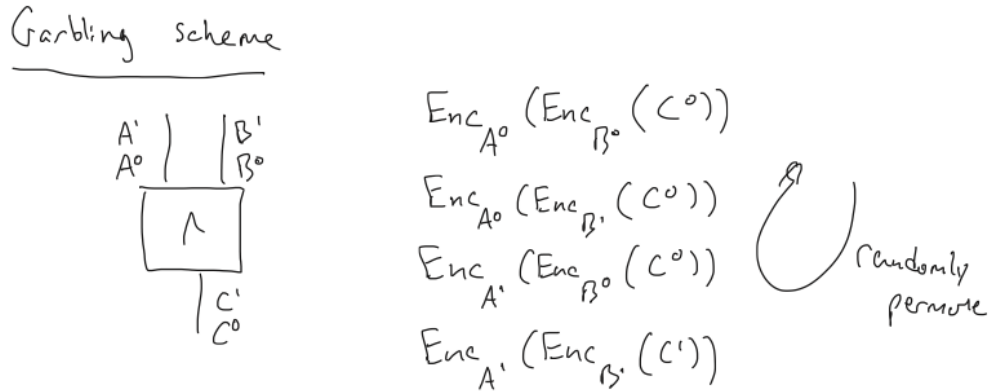
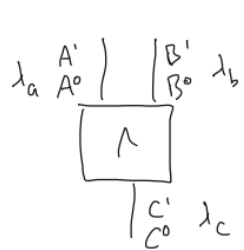


Figure 9: Garbling demonstration using public-key encryption scheme.

### 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



instead of associating  $A^0$  w/ 0,  
now associate  $A^0$  w/  $\lambda_a$

Give evaluator the label of a key that it learns

$$\text{Enc}_{A^0}(\text{Enc}_{B^0}(\underbrace{C^{(\lambda_a \wedge \lambda_b) \oplus \lambda_c}}_{\text{key}}, \underbrace{(\lambda_a \wedge \lambda_b) \oplus \lambda_c}_{\text{data}}))$$

$\vdots$

$$\text{Enc}_{A^1}(\text{Enc}_{B^1}(C^{(\bar{\lambda}_a \wedge \bar{\lambda}_b) \oplus \lambda_c}, (\bar{\lambda}_a \wedge \bar{\lambda}_b) \oplus \lambda_c))$$

Figure 10: The idea of point-and-permute

## 4 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.

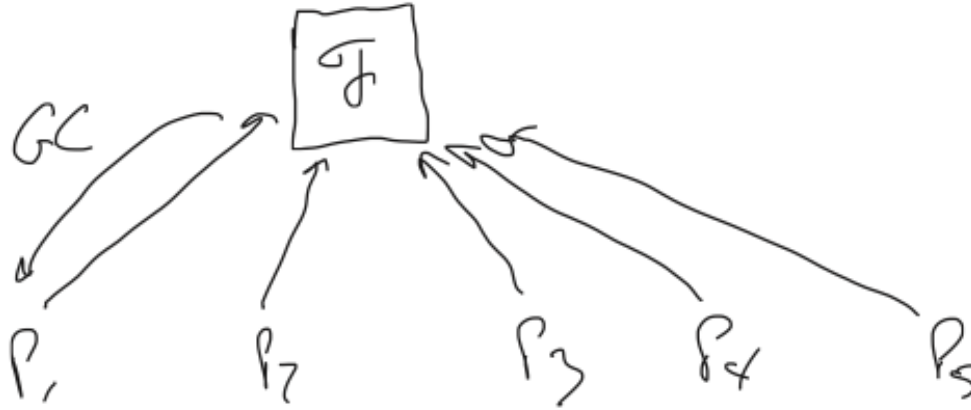


Figure 11: The idea behind the BMR protocol

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

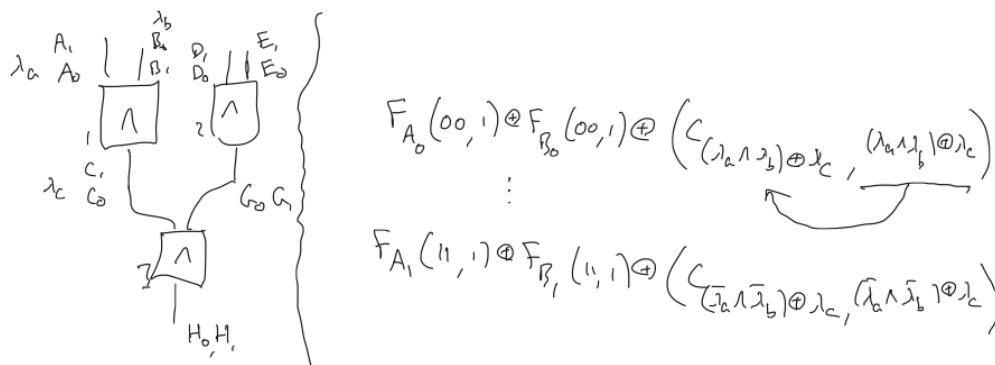


Figure 12: BMR computation example

## 5 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- $n$  secret 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.

- real-world execution of protocol  $\Pi$  w/ some adversary  $A$   
(Output of honest party, view of  $A$ )
- ideal-world evaluation of  $\mathcal{F}$

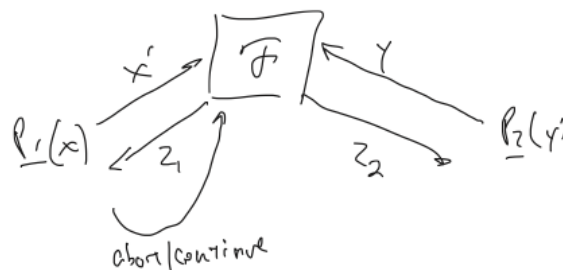


Figure 13: Malicious security

### 6.2 Commitment Schemes

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

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

### 6.3 Zero-Knowledge Proofs

Let language  $L \in \text{NP}$  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

□

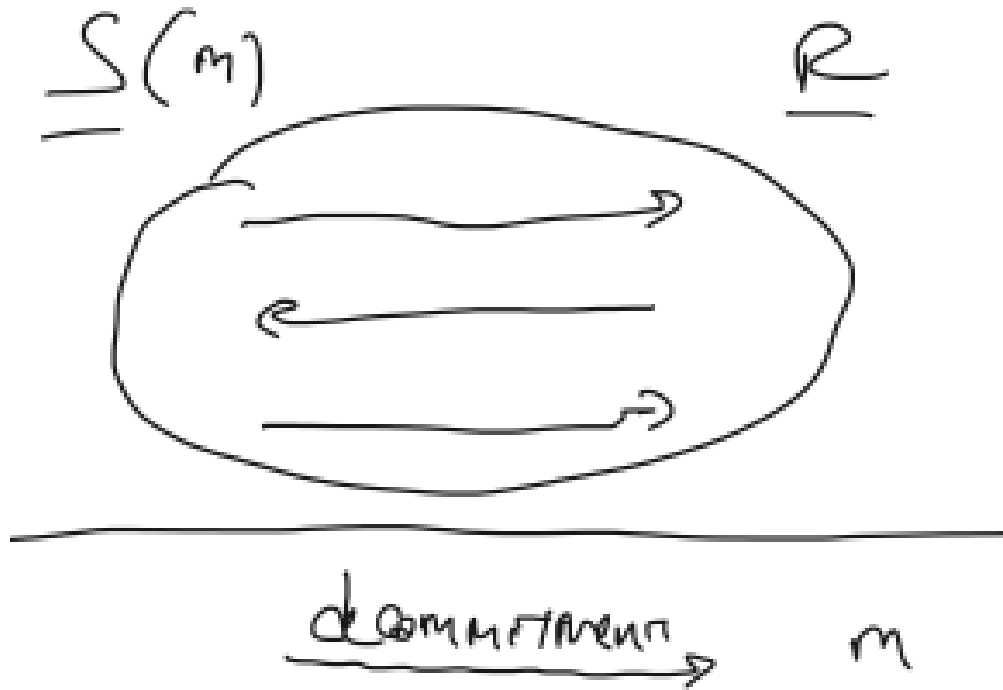


Figure 14: Setup for commitment schemes

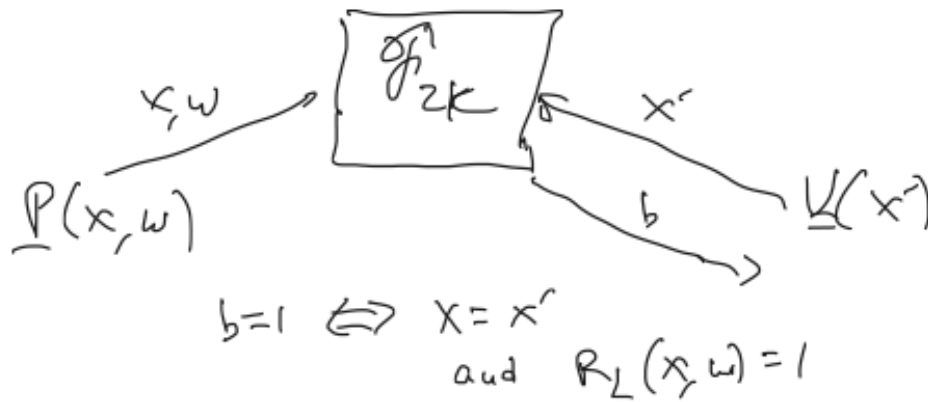


Figure 15: Setup for defining zero-knowledge proofs

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

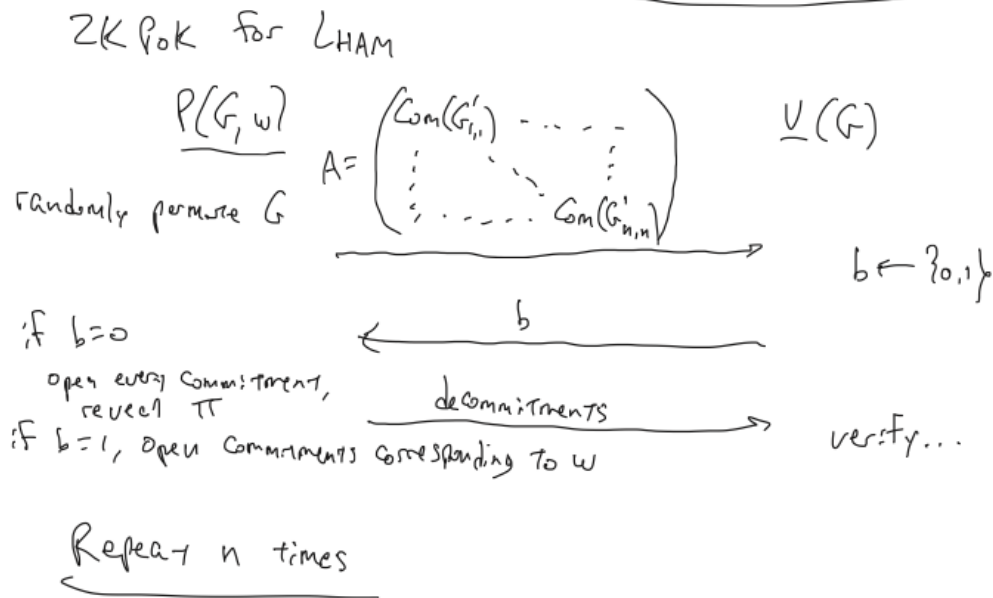


Figure 16: A protocol for ZKPoK for HAM

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 □



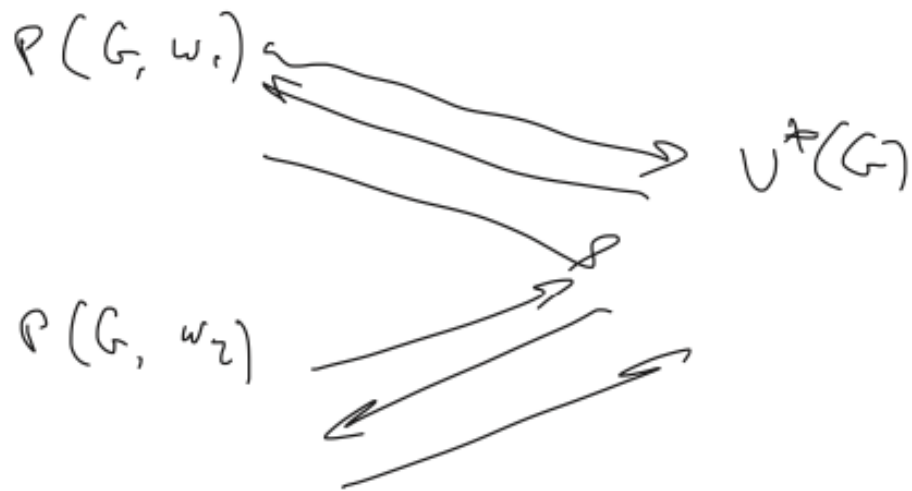


Figure 17: Witness indistinguishability

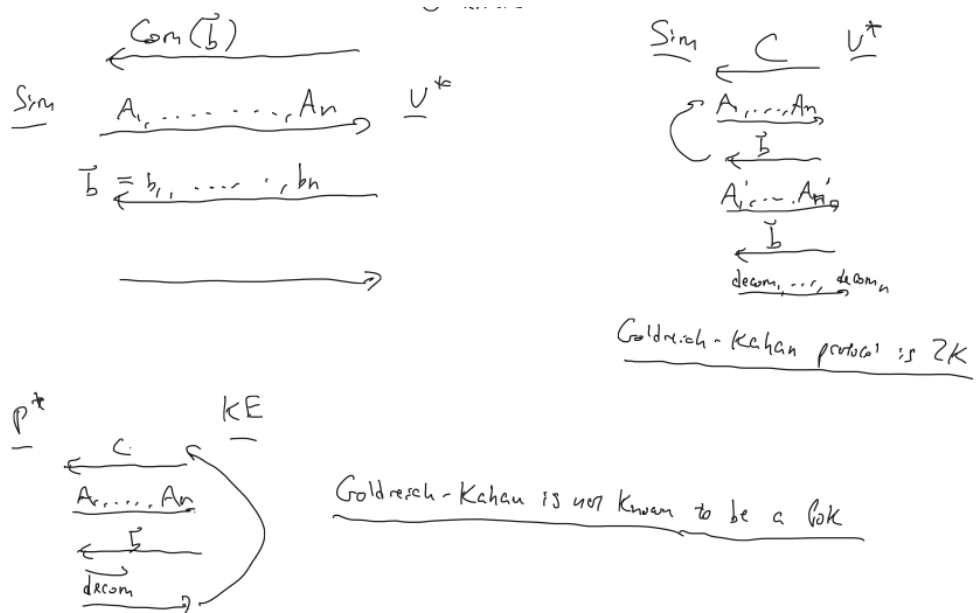


Figure 18: The Goldreich-Kahan protocol

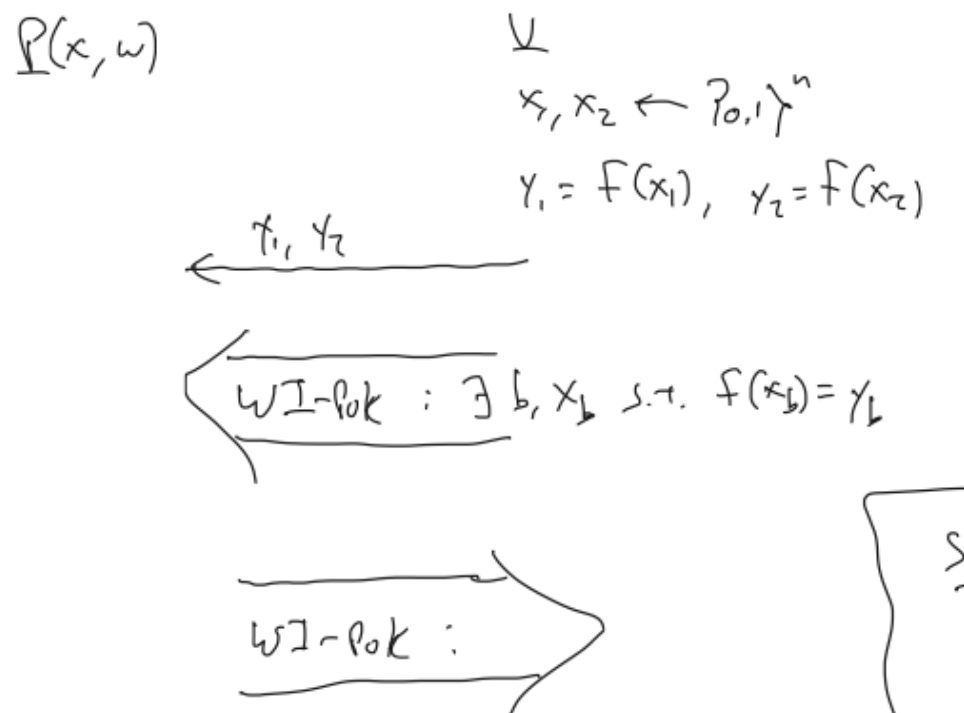


Figure 19: The Feige-Shamir protocol

## 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.

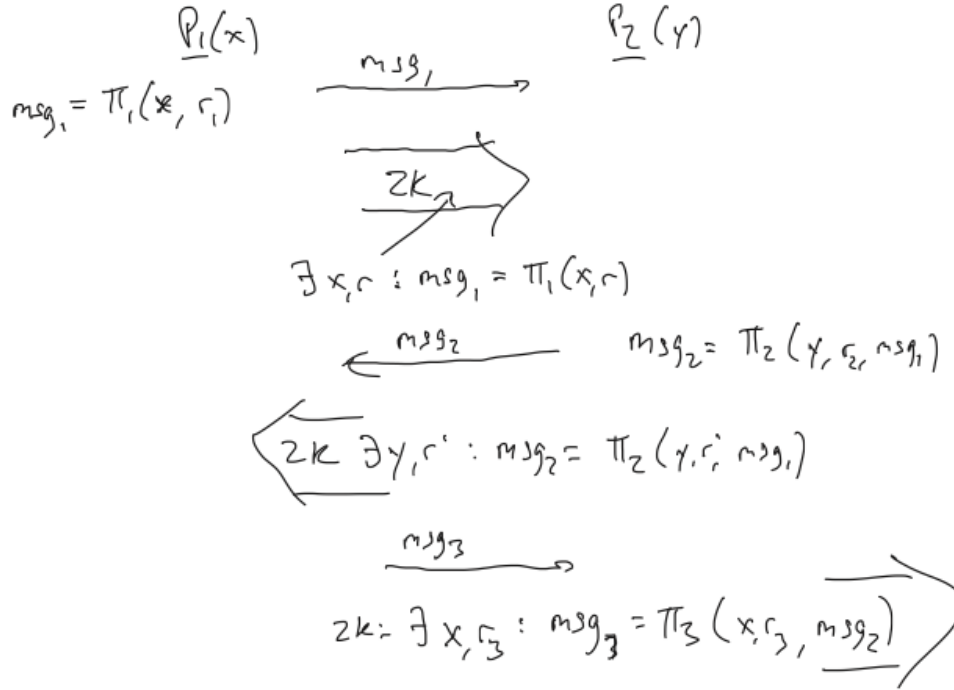


Figure 20: Idea for the GMW I compiler

#### 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 of ideal functionality

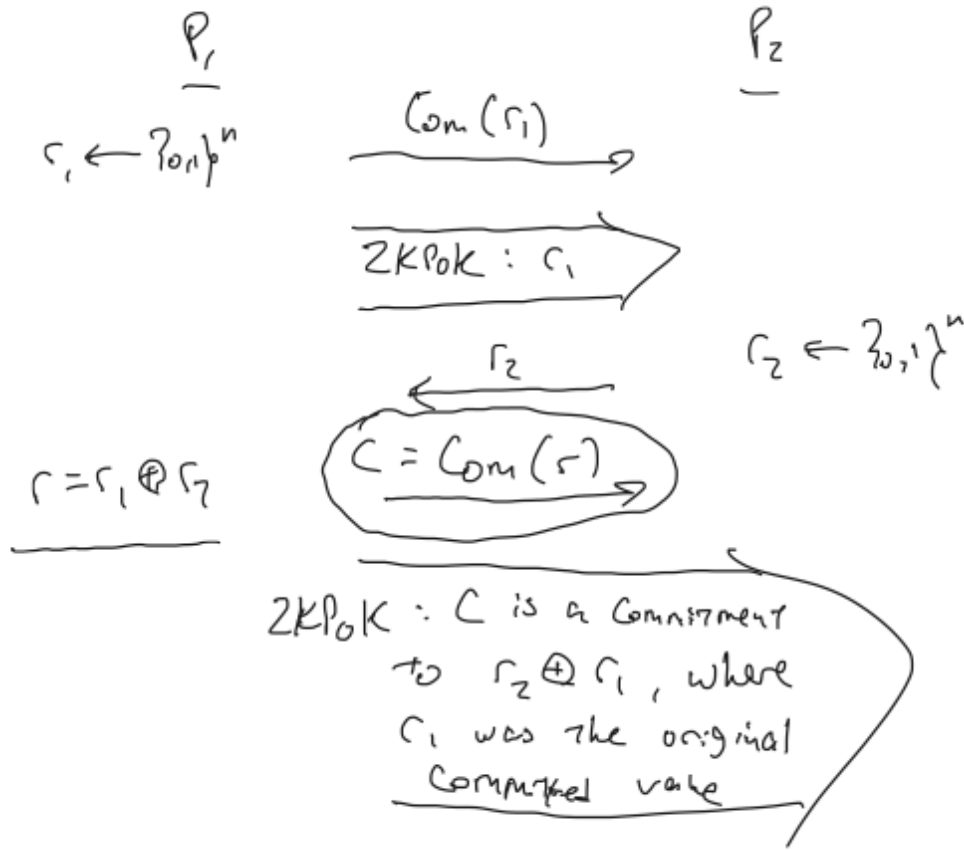


Figure 21: The coin-tossing protocol

- all parties get output

Security-with-abort is achievable for  $t < n/2$  given broadcast. But it is not achievable for  $t \geq n/2$  (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 them 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:

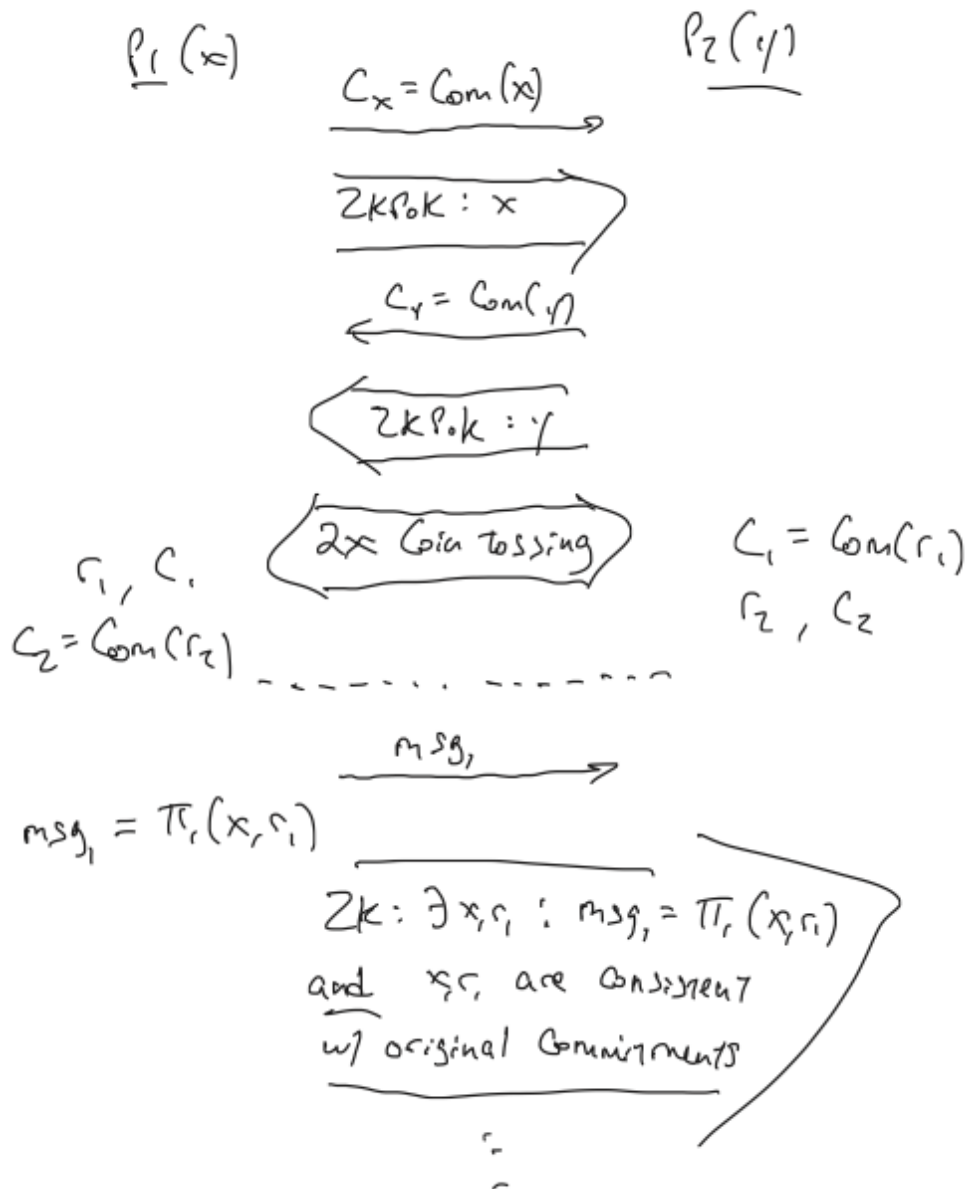


Figure 22: The GMW-I-compiled coin-tossing protocol

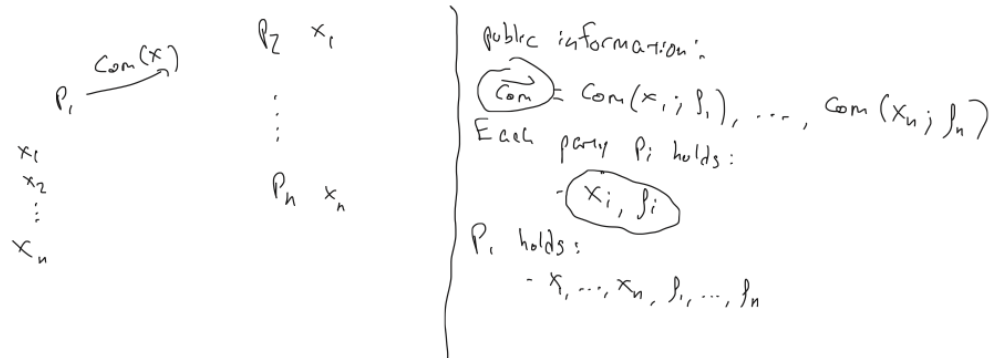
- (a) each party broadcasts their share of  $P_i$ 's input/randomness, plus the corresponding  $\rho$ s
- (b) parties reconstruct  $P_i$ 's input/randomness from  $t + 1$  correct shares
- (c) parties run  $\Pi$  on behalf of  $P_i$  from then on

## GMW II Compiler

Start with semi-honest protocol  $\Pi$ , secure against  $t < n/2$  parties

Shamir's secret sharing  $\Rightarrow$   $(t+1)$ -out-of- $n$  sharing

Committed Verifiable secret sharing (VSS)



$$\sigma_{\text{VSS}} \left( \underset{\perp}{x_1, \perp, \dots, \perp} \right) = \left( \underset{p_1, \dots, p_n}{(\text{Com}, \vec{x}, \vec{p})}, \underset{p_2}{(\text{Com}, x_2, p_2)}, \dots, \underset{p_n}{(\text{Com}, x_n, p_n)} \right)$$

Figure 23: The GMW II compiler

## 8 Broadcast and Byzantine Agreement

Parties can realize a broadcast channel using a broadcast protocol.

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

- **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  $P_1, \dots, P_n$  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  $t < n/2$ ,  $\text{BA} \implies \text{broadcast}$ , and  $\text{broadcast} \implies \text{BA}$ . Additionally, BA only makes sense for  $t < n/2$ , whereas broadcast makes sense even for  $n/2 \leq t < n$ .

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

*Proof.*

□

### 8.1 Phase-King Sub-Protocol for BA

The phase-king sub-protocol will be used in order to construct a BA protocol for  $t < n/3$ :

1. construct a phase-kind sub-protocol with designated party called “King”
2. run phase-king sub-protocol  $t + 1$  times with  $P_1, \dots, P_{t+1}$  as the successive Kings

**Protocol 13** (phase-king). Designate a party  $P_K$  as the “King” of this protocol. The phase-king protocol proceeds in three rounds:

- every  $P_i$  sends its input  $v_i$  to everyone else; each  $P_i$  sees  $C_i^b = 1$  iff there are at least  $n - t$  parties sent to it the bit  $b$

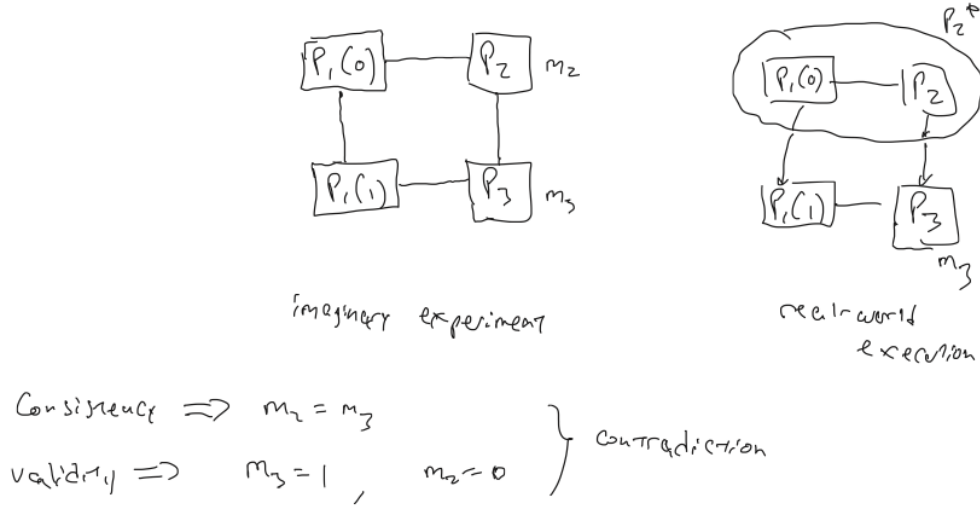


Figure 25: Impossibility of BA/broadcast when  $t \geq n/3$

- each  $P_i$  sends  $C_i^0, C_i^1$  to everyone else; each  $P_i$  sets  $D_i^b := \{\text{the number of parties who sent } C^b = 1\}$ ; if  $D_i^1 > t$  then  $v_i := 1$  else  $v_i := 0$
- $P_K$  sends  $v_K$  to all parties; each  $P_i$  does the following: if  $D_i^{v_i} < n - t$ , then output  $v_K$  else output  $v_i$ .

**Lemma 2.** If  $t < n/2$  and all honest parties begin holding input  $v$ , then they all output  $v$

**Lemma 3.** If  $t < n/3$  and the Kind is honest, then all honest parties agree on their output.

*Proof.* TODO: lecture 14 □

**Definition 21.** A **public-key infrastructure** (PKI) ensures:

- every party  $P_i$  has  $(sk_i, pk_i)$  for a public-key signature scheme
- every party has the same vector  $(pk_1, \dots, pk_n)$  i.e. each party knows all parties' public keys.

**Protocol 14** (Dolev-Strong). This is an “**authenticated broadcast**” protocol i.e. is secure even if all but one of the parties are corrupted. This protocol relies on a PKI. Assume  $P_i$  is the sender. Then an  $(m, i)$ -valid message is learned in round  $i$ , of the form

$$m, \sigma_1, \sigma_2, \dots, \sigma_i,$$

by parties distinct from the receiver. Abbreviation:

$$m\text{-valid message} \iff (m, i)\text{-valid for some } i$$

The protocol proceeds in  $n$  rounds:

1.  $P_1$  signs  $m$  and sends  $m, \sigma_1$  to everyone.
- 2, ...,  $n - 1$  If  $P_i$  received an  $m$ -valid message in the previous round, it appends its signature and sends an  $m$ -valid message in the current round.
2. Let  $S_i$  denote the set of  $m$  for which  $P_i$  received an  $m$ -valid message. If  $|S_i| = 0$  or  $|S_i| > 1$ , then output 1. If  $|S_i| = 1$ , then output the value in  $S_i$ .

**Lemma 4.** The Dolev-Strong protocol is valid and consistent.

*Proof.* Validity is immediate. Consistency amounts to the claim that all honest parties agree on the set of  $m$ -valid messages. TODO: lecture 14. □

The Dolev-Strong protocol as described is not necessarily very efficient, but it can be modified so that it is.



## 9 Efficient and Maliciously Secure 2PC

Ideas:

- use an OT protocol secure against malicious adversaries
- $P_i$  can violate correctness by garbling the wrong circuit, or swapping OT inputs
- the above can lead to violations of privacy
- selective-failure attack on privacy

### 9.1 Selective-Failure Attack

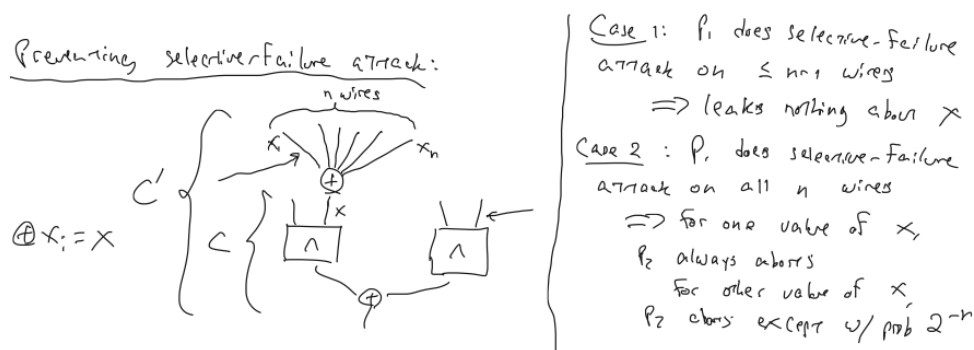


Figure 26: Idea of preventing a selective-failure attack

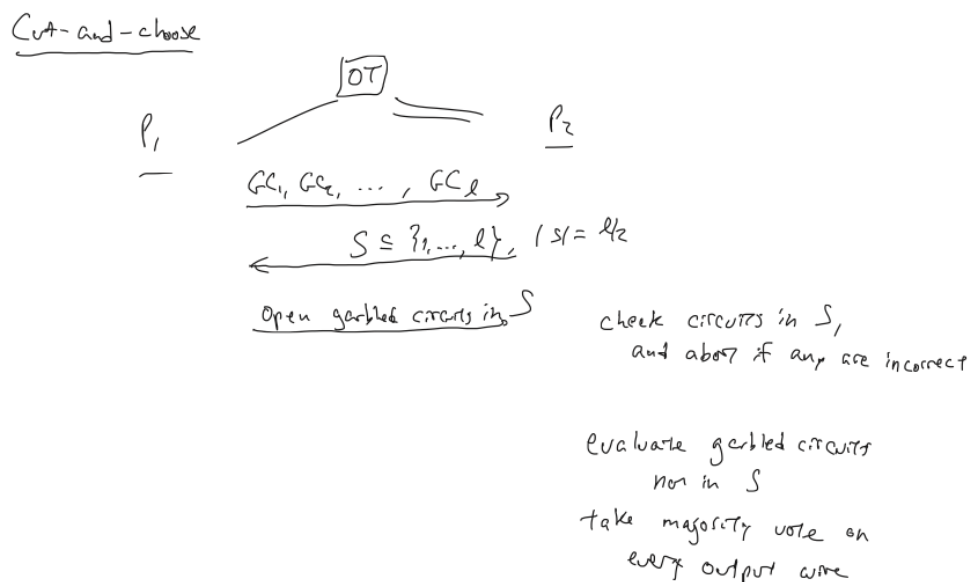


Figure 27: The cut-and-choose protocol to prevent selective-failure attacks

**Protocol 15** (Cut-and-Choose). In this protocol,  $P_1$  maximizes its probability of successfully cheating if

$l/4$  circuits are bad. So,

$$\begin{aligned} \text{Pr}_{\text{success}} &= \frac{\binom{3l/4}{l/2}}{\binom{l}{l/2}} \\ &= \frac{(3l/4)!(l/2)!}{(l/4)!l!} \\ &= \frac{(l/2) \cdots (l/4 + 1)}{l \cdots (3l/4 + 1)} \\ &\leq 2^{-l/4}. \end{aligned}$$

Then the probability for cheating  $2^{-n}$  implies  $l \approx 4n$  (can be improved to  $l \approx 3n$ ).

The previous protocol was the *circuit-level* cut-and-choose protocol. The same idea can be achieved by using garbled gates at the *gate-level*, as demonstrated in the following protocol.

**Protocol 16** (LEGO cut-and-choose). This approach uses garbled circuits and applies the cut-and-choose idea to each gate of the garbled circuit. For security  $2^{-n}$ , the total garbled gates needed is  $O(|c| \cdot n / \log |c|)$

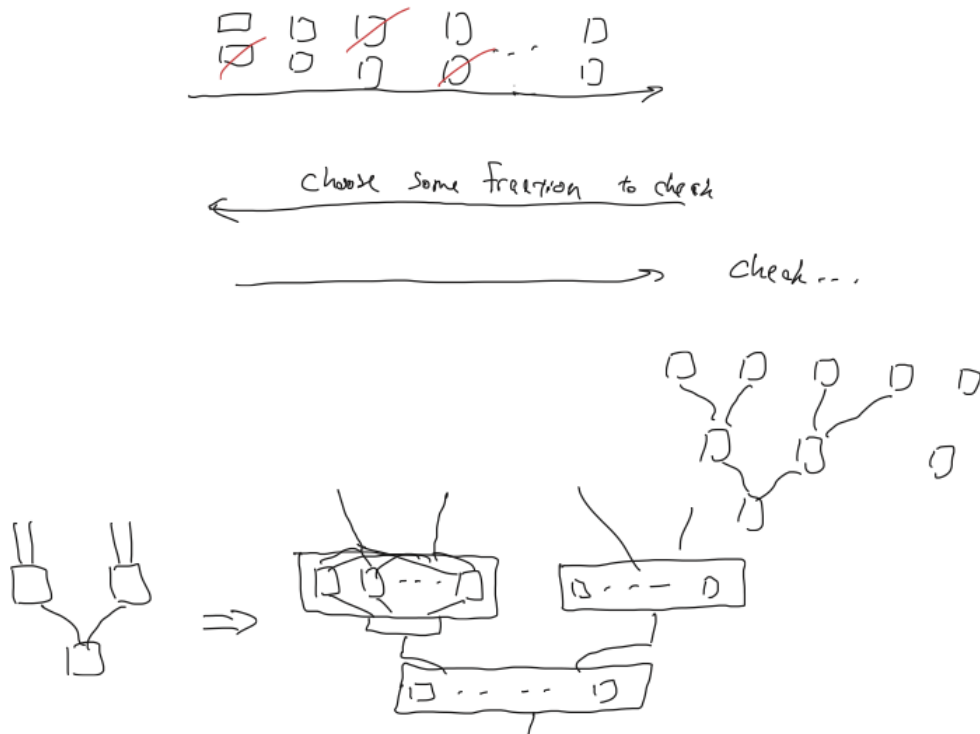


Figure 28: The idea of the LEGO cut-and-choose protocol

TODO: more details at end of lecture 15

## 10 Differential Privacy

**Ideas.** What functionalities  $\mathcal{F}$  are “safe” to compute in the first place? Given a database  $D$ , want to answer query  $q$  on  $D$ , giving an approximate/noisy answer  $q'(D)$ .

*Informally:* releasing  $q'(D)$  is *private* when the answer would be “roughly the same” whether or not a particular user’s data was in  $D$ .

**Definition 22** (Differential Privacy). Let  $D = (x_1, \dots, x_n) \in X^n$  be a database, where  $x_i$  is the data of user  $i$ . Databases  $D, D'$  are **neighboring** if they differ in data of one user. Mechanism  $M : X^n \rightarrow Y$  is  **$\epsilon$ -differentially private** when for all neighboring  $D, D'$  and all  $T \subset Y$ ,

$$\Pr[M(D) \in T] \leq e^\epsilon \cdot \Pr[M(D') \in T]$$

$M$  is  $(\epsilon, \delta)$ -differentially private when for  $D, D', T$  as above,

$$\Pr[M(D) \in T] \leq e^\epsilon \cdot \Pr[M(D') \in T] + \delta$$

*Note.*  $\epsilon = \Omega(1/n)$ ,  $\delta$  can be cryptographically small. So we need to look at privacy/utility trade-off.

**Composition.** If  $M$  is  $\epsilon$ -differentially private and  $D, D'$  differ in data of  $k$  users, then for any  $T \subset Y$ :

$$\Pr[M(D) \in T] \leq e^{k\epsilon} \cdot \Pr[M(D') \in T]$$

If  $M_1, \dots, M_l$  are  $\epsilon$ -differentially private, then  $(M_1 \times \dots \times M_l)$  is  $l\epsilon$ -differentially private. In fact, if  $l \leq 1/\epsilon^2$ , then for any  $\delta > 0$ ,  $(M_1 \times M_l)$  is  $O((l \log(1/\delta))^{1/2} \cdot \epsilon, \delta)$ -differentially private.

### 10.1 Laplace Mechanism

**Definition 23.** For a query  $q : X^n \rightarrow \mathbb{R}$ , defined **global sensitivity** to be

$$GS_q := \max_{D \sim D'} |q(D) - q(D')|.$$

Next, the idea is to answer query  $q$  by returning  $q(D) + \text{noise}$ , where noise depends on  $GS_q$  and  $\epsilon$ . Now we need to choose a distribution.

**Definition 24.** The **Laplace distribution**,  $\text{Lap}(\sigma)$ , is the distribution such that

$$\Pr[z] \propto e^{-|z|/\sigma}$$

with mean 0, standard deviation  $\sigma \cdot \sqrt{2}$ , and

$$\Pr[\text{Lap}(\sigma) > \sigma t] \leq e^{-t}$$

**Definition 25.** The **Laplace mechanism** for differential privacy answers queries  $q$  with  $q(D) + \text{Lap}(GS_q/\epsilon)$ .

**Theorem 8.** The Laplace mechanism is differentially private.

*Proof.* TODO: lecture 17 □

**Definition 26.** The **exponential mechanism** for differential privacy is an abstract mechanism based on a scoring function  $\text{score}(D, y)$ . Let  $GS = \max_y \max_{D \sim D'} |\text{score}(D, y) - \text{score}(D', y)|$ . Then the mechanism, on input  $D$ , outputs  $y$  with probability proportional to  $e^{\epsilon \cdot \text{score}(D, y) / 2GS}$ .

**Lemma 5.** The exponential mechanism is  $\epsilon$ -differentially private for any scoring function.

Utility for the exponential mechanism: with probability  $O(1)$ , the output  $y$  satisfies

$$\text{score}(D, y) \geq \max_{y^*} \text{score}(D, y^*) - O(GS \log |Y| / \epsilon)$$

## 10.2 Application: Synthetic Data

Given a dataset  $D$  and a set of queries  $Q$ . For all  $q \in Q$ ,  $q(D) = \sum_i q(x_i)$ ,  $D = (x_1, \dots, x_n)$ , set

$$\alpha := O\left(\left(\frac{\log |Q| \log |X|}{\epsilon n}\right)^{1/3}\right)$$

We can use the exponential mechanism to output synthetic data  $\hat{D}$  such that

$$\forall q \in Q : |q(\hat{D}) - q(D)| \leq O(\alpha)$$

using scoring function

$$\text{score}(D, \hat{D}) = -\max_{q \in Q} |q(D) - q(\hat{D})|$$

Set  $m = O(\frac{\log |Q|}{\alpha^2})$  to the number of rows in output dataset. This implies

$$\exists \hat{D}^* : \forall q \in Q : |q(\hat{D}^*) - q(D)| \leq O(\alpha)$$

which implies that the output  $\hat{D}$  satisfies

$$\begin{aligned} \text{score}(D, \hat{D}) &\geq \text{score}(D, \hat{D}^*) - O\left(\frac{1}{2} \cdot \log |X|^m\right) \\ &\geq -O(\alpha) - O(\alpha) \\ &\geq -O(\alpha) \\ &\text{with high probability} \end{aligned}$$

## 10.3 PAC Learning

Let  $C$  be the class of boolean functions  $C = c : X \rightarrow \mathbb{B}$ . For some  $c \in C$ , a learning algorithm is given  $((x_1, c(x_1)), \dots, (x_n, c(x_n)))$  where  $x_i \in X$  are sampled from unknown distribution  $D$ . Then  $\mathcal{L}$  should output some  $c' \in C$  such that with high probability

$$\Pr_{x \leftarrow D}[c'(x) = c(x)] \text{ is high}$$

Use exponential mechanism with scoring function

$$\text{score}(\{(x_i, y_i)\}, c') = -|i : c'(x_i) \neq y_i|.$$

If there exists  $c$  such that  $\text{score}(\{(x_i, y_i)\}, c) = 0$ , then output  $c'$  satisfies the following with high probability

$$\text{score}(I, c') \geq -\frac{1}{\epsilon} \log(|C|)$$

which implies that

$$\Pr_{x \leftarrow D}[c'(x) = c(x)] \text{ is high}$$

## 10.4 Centralized versus Local Models

.

**Definition 27.** Let  $R : X \rightarrow X'$ . Then a querying protocol is **local differentially-private** when

$$\forall x, x', T \subset X' : \Pr[R(x) \in T] \leq e^\epsilon \cdot \Pr[R(x') \in T]$$

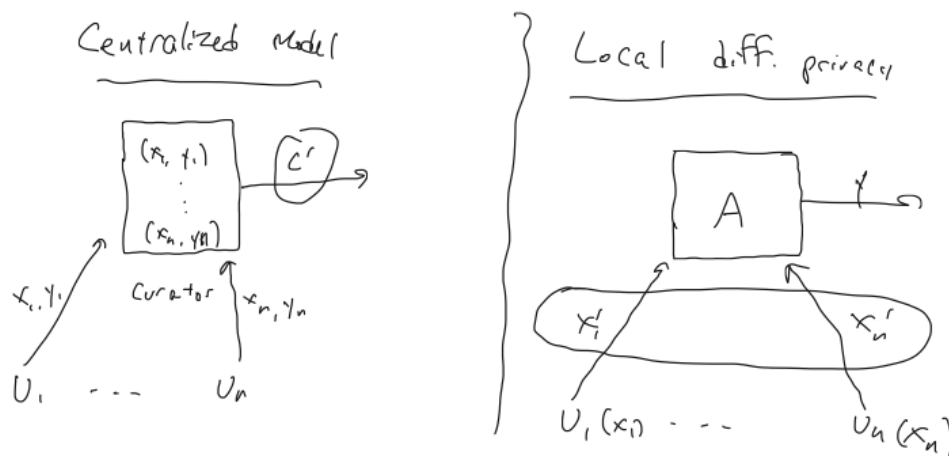


Figure 29: Comparing centralized and local models of differential privacy.

The local differentially-private version of the Laplace mechanism:

$$\begin{aligned}
 R(x) &= x + \text{Lap}(B/\epsilon) = x' \\
 x'_1 &= x_1 + \text{Lap}(B/\epsilon) \\
 &\vdots \\
 x'_n &= x_n + \text{Lap}(B/\epsilon) \\
 \sum x'_i &= \sum x_i + \sum_{i=1}^n \text{Lap}(B/\epsilon)
 \end{aligned}$$

whereas in the centralized model the last line was

$$\sum x'_i = \sum x_i + \text{Lap}(B/\epsilon)$$

TODO: what to make of last slid of lecture 18?

## 10.5 Shuffle Model

**Idea.** To generate a noisy histogram:

1. each party applies randomized response
  - with probability  $1 - \gamma$ :  $x'_i = x_i$
  - with probability  $\gamma$ :  $x'_i \leftarrow \{1, \dots, k\}$
2. each party sends  $x'_i$  to anonymous bulletin board
3. curator gets  $\{x'_1, \dots, x'_n\}$  and generates histogram from that

The expectation is that a  $\gamma$ -fraction of the parties replace their inputs by random values. So, consider the two cases when  $P_1$  changes its inputs:

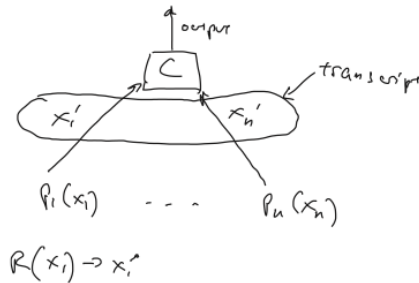
- $P_1$  sends uniform value in  $\{1, \dots, k\}$ , view it as independent of  $P_1$ 's input
- $P_1$  sends its input (either  $x_1$  or  $\hat{x}_1$ ); still a privacy benefit in shuffle model because, with some probability, other parties send  $x_1/\hat{x}_1$  due to randomized response.

Fix any view of the curator — include bit-vector indicating which parties send random values. Let  $v$  denote the values sent by  $P_1$  and the values sent by the parties sending random values.

Centralized model:



Local model:



$$x_i \in \{0,1\}$$

want to estimate  $\sum x_i$

Centralized model: Laplace mechanism  $\sum x_i + \text{Lap}(1/\epsilon)$

- $\epsilon$ -diff. private
- noise  $O(1/\epsilon)$

Local model: local Laplace mechanism: each party sends  $x'_i = x_i + \text{Lap}(1/\epsilon)$   
curator releases  $\sum x'_i$

- $\epsilon$ -diff. private
- noise  $O(\sqrt{n}/\epsilon)$  - optimal

Figure 30: (Again) centralized versus local models of differential privacy

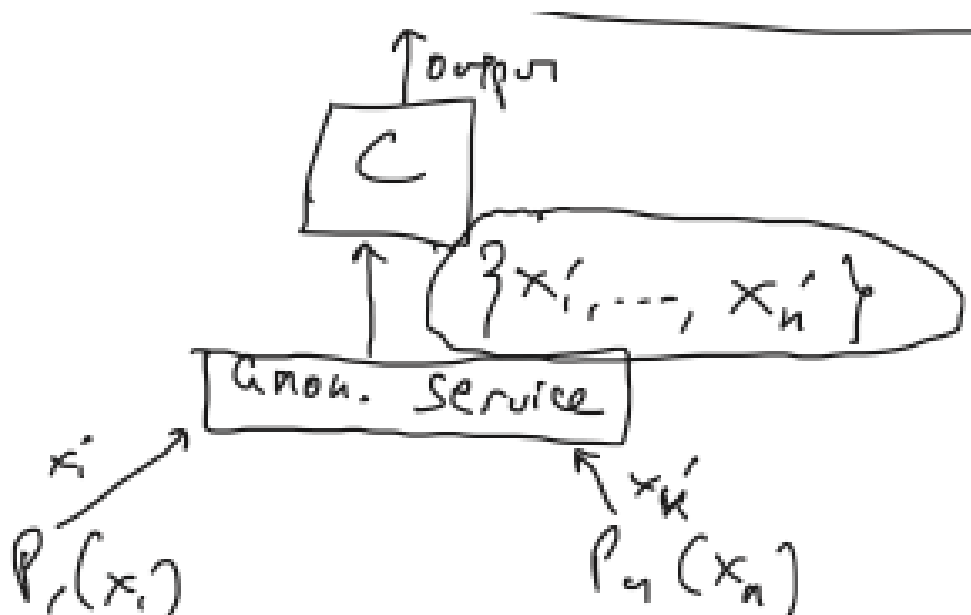


Figure 31: Shuffled model of differential privacy

**Application to Summation.** We can use this protocol to generate a histogram, via computing the sums of the contributed values. This yields noise  $O(n^{1/3})$ .

$$\begin{aligned}
\Pr[M(1, \dots, x_n) = v] &= \binom{|B|}{n_1-1, n_2, \dots, n_K} \cdot \underbrace{\Pr[|B| \text{ parties send random values}]} \\
\Pr[M(2, x_2, \dots, x_n) = v] &= \binom{|B|}{n_1, n_2-1, \dots, n_K} \cdot \underbrace{\Pr[|B| \text{ parties send random values}]} \\
\binom{|B|}{n_1-1, n_2, \dots, n_K} &= \binom{|B|}{n_1-1} \cdot \binom{|B|-n_1+1}{n_2} \cdot \binom{|B|-n_1-n_2+1}{n_3} \dots \\
\binom{|B|}{n_1, n_2-1, \dots, n_K} &= \binom{|B|}{n_1} \cdot \binom{|B|-n_1}{n_2-1} \cdot \binom{|B|-n_1-n_2+1}{n_3} \dots
\end{aligned}$$

Figure 32: Calculations #1 for shuffled model

$$\begin{aligned}
\frac{\Pr[M(1, x_2, \dots, x_n) = v]}{\Pr[M(2, x_2, \dots, x_n) = v]} &= \frac{\binom{|B|}{n_1-1} \cdot \binom{|B|-n_1+1}{n_2}}{\binom{|B|}{n_1} \cdot \binom{|B|-n_1}{n_2-1}} \\
&= \frac{\cancel{|B|!} \cdot \cancel{(|B|-n_1+1)!}}{(n_1-1)! \cdot \cancel{(|B|-n_1)!} \cdot n_2! \cdot \cancel{(|B|-n_1-n_2+1)!}} \\
&= \frac{\cancel{|B|!} \cdot \cancel{(|B|-n_1)!}}{n_1! \cdot \cancel{(|B|-n_1)!} \cdot (n_2-1)! \cdot \cancel{(|B|-n_1-n_2+1)!}} \\
&= \frac{n_1! \cdot (n_2-1)!}{n_2! \cdot (n_1-1)!} = \frac{n_1}{n_2}
\end{aligned}$$

Figure 33: Calculations #2 for shuffled model

**Amplification result.** Take any  $R$  that is  $\epsilon_0$ -local-differentially-private and run it through the shuffle mechanism. Then the result is  $O(\min\{\epsilon_0, 1\} \cdot e^{\epsilon_0} \sqrt{\frac{\log(r/\delta)}{n}}, \delta) - DP$ .

The summation is extracted using an anonymization layer. We want to learn the sum exactly, without revealing parties' inputs i.e.

$$\text{View}(\vec{x}) \approx \text{View}(\vec{x}')$$

for any  $\vec{x}, \vec{x}'$  with the same sum.

**Protocol 17** (Shuffled Summation). Each party  $P_i$  with input  $x_i$  chooses  $m$  values  $x_i^1, \dots, x_i^m$  uniformly subject to  $\sum_j x_i^j = x_i \pmod q$ . Then sends  $x_i^1, \dots, x_i^m$  to the anonymization service.

The curator gets  $\{x_i^j\}$  and outputs the sum. To achieve differential privacy, apply this protocol to  $x_i + \text{noise}$ . The final result is

$$\sum x_i + \sum \text{noise}$$

## Multi-message shuffle model

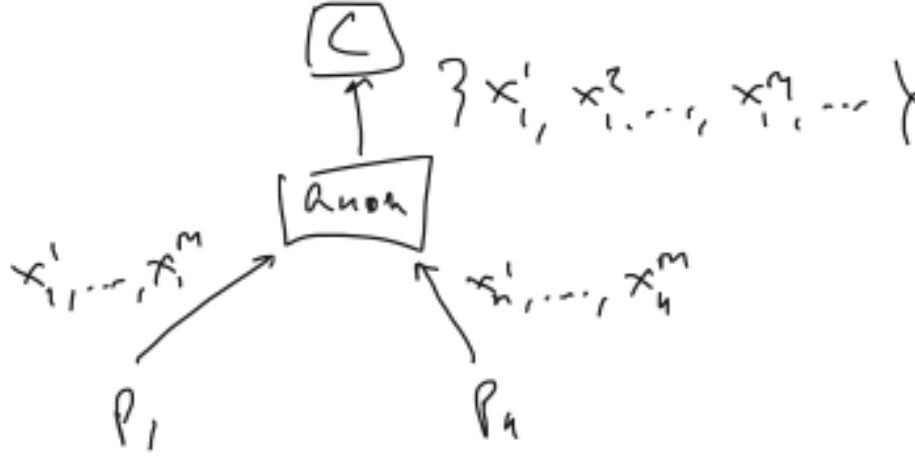


Figure 34: Multi-message shuffled differential privacy model

## Single-msg shuffle model

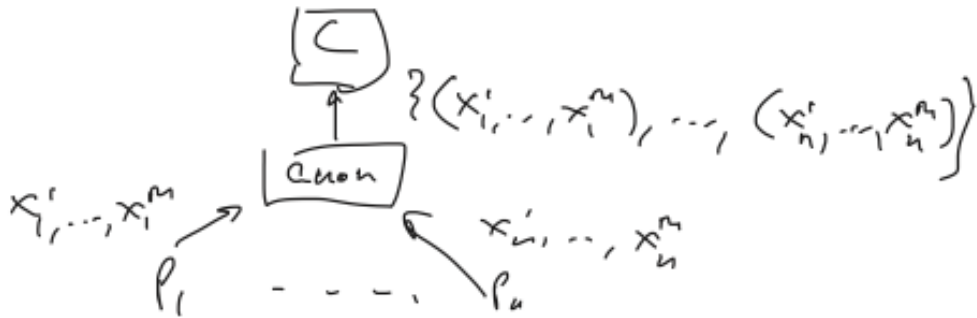


Figure 35: Single-message shuffled differential privacy model

where the noise is  $O(1/\epsilon)$ .

**Definition 28.** A protocol  $\Pi$  is **information-theoretically differentially private** if for any set of  $t$  parties and any neighboring inputs  $\vec{x}, \vec{x}'$  (that are equal for the  $t$  corrupted parties), and any set of views  $V$ ,

$$\Pr[\text{View}_t^\Pi(\vec{x}) \in V] \leq e^\epsilon \Pr[\text{View}_t^\Pi(\vec{x}') \in V]$$

**Definition 29.** A protocol  $\Pi$  is **computationally differentially private** if for all efficient distinguishers  $D$ :

$$\Pr[D(\text{View}_t^\Pi(\vec{x})) = 1] \leq e^\epsilon \Pr[D(\text{View}_t^\Pi(\vec{x}')) = 1] + \delta(n)$$

**Example 4.** A computationally differentially private protocol.



## parallel multi-message shuffling

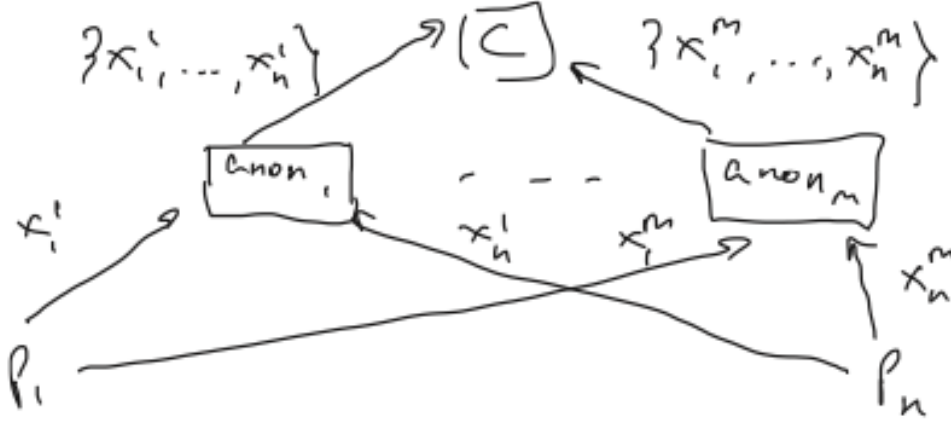


Figure 36: Parallel multi-message differential privacy model

1. Parties set up a threshold homomorphic encryption scheme:
  - have public key  $pk$  and  $\text{Enc}_{pk}(x_1), \text{Enc}_{pk}(x_2) \implies \text{Enc}_{pk}(x_1 + x_2)$
  - (*threshold*) every party holds a share  $sk_i$  of secret key  $sk$
2. Each party sets  $\hat{x}_i = x_i + \text{noise}$ , then publishes  $\text{Enc}_{pk}(\hat{x}_i)$
3. The parties compute  $\text{Enc}_{pk}(\sum \hat{x}_i)$
4. The parties collectively decrypt to get  $\sum \hat{x}_i$

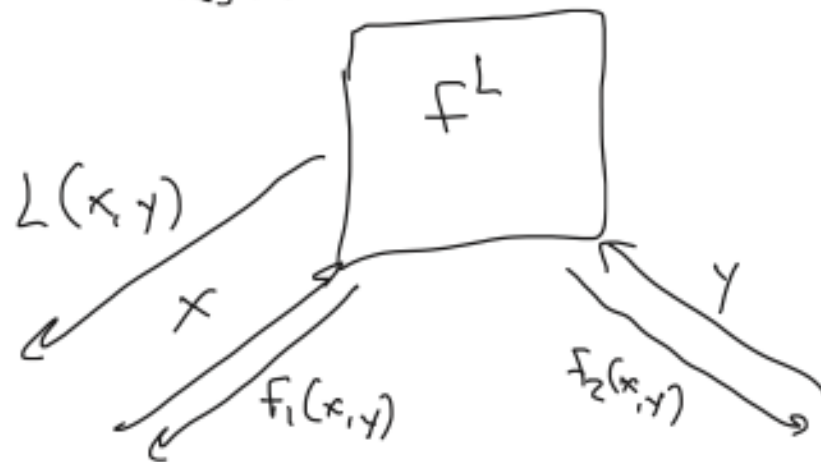
Note that with this method, the noise per party can stand be much lower that in the centralized differential privacy model.

Some possibilities:

- use MPC to compute a differentially private functionality  $\hat{F}$
- use an  $(\epsilon, \delta)$ -differentially private MPC protocol to compute  $\hat{F}$
- use an  $(\epsilon, \delta)$ -differentially private MPC protocol to compute  $F$ .

To show that this protocol is computationally differentially private in the malicious case:

Semi-honest:



Protocol  $\mathcal{E}$ -differentially privately computes  $F, f$ :

- $\pi$  securely computes  $f^L$
- $L$  is  $\mathcal{E}$ -d.p.

Figure 37: TODO: lecture 20

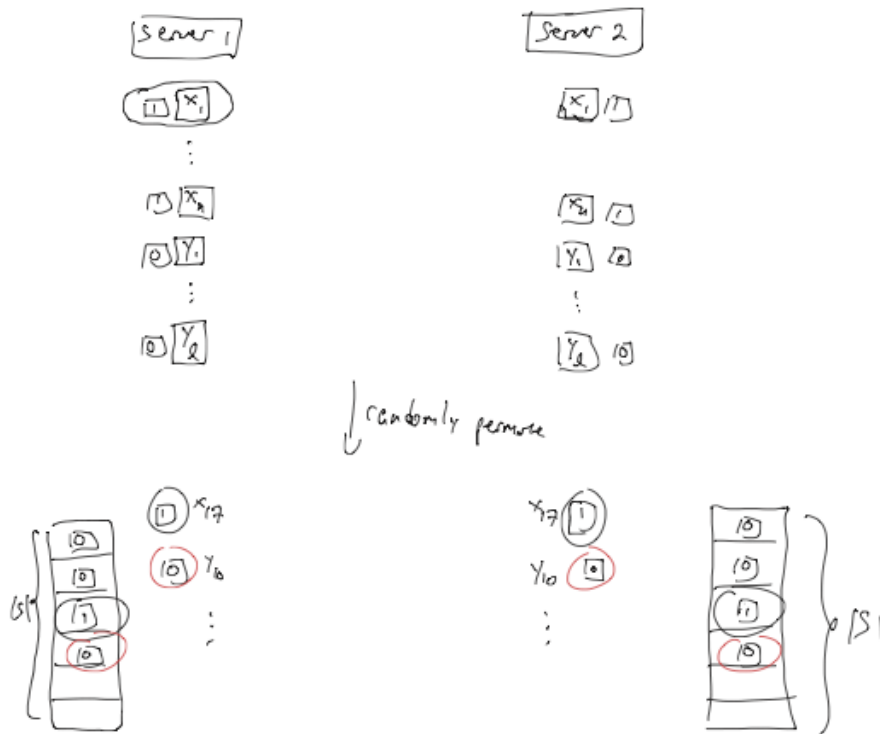
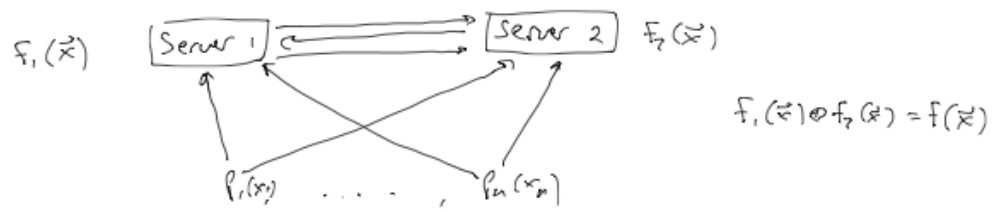
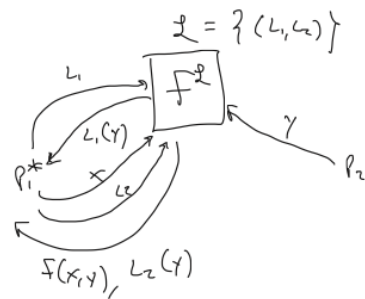


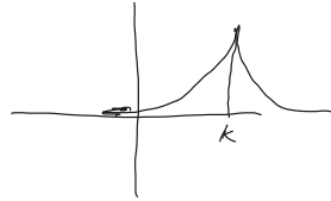
Figure 38: TODO: lecture 20

diff. private computation, malicious case



Protocol  $\pi$  is  $\epsilon$ -dp. if

- $\pi$  securely realizes  $F^L$
- Every  $(L, L_2) \in \mathcal{L}$  is  $\Sigma$ -diff. private



$$\underline{P}_1(x_1, \dots, x_n)$$

$$\underline{P}_2(y_1, \dots, y_n)$$

$$d_1, d_1', x_1, x_{12}, x_{25} \longleftrightarrow y_1, y_2, d_1, d_2.$$

$$x_2, x_7, x_{20}$$

$$y_0, y_{21}, y_{30}$$

$$\vdots$$

$$y_3$$

Figure 39: TODO: lecture 20

## 11 Private Data Collection

**Secure aggregation.** Suppose we have a large number of clients, each holding  $x_i$ . The server wants to compute  $\sum_i x_i$ . Assume all clients have Diffie-Helman (DH) public key  $h_i = g^{s_i}$ , and each pair of clients  $i, j$  can compute shared key  $k_{i,j}$ . The the following is a basic protocol.

**Protocol 18** (Basic Private Data Collection). Each client sends to the server

$$y_i = x_i + \sum_{i < j} k_{i,j} - \sum_{i > j} k_{i,j},$$

where  $\sum_{i < j} k_{i,j} - \sum_{i > j} k_{i,j}$  is  $mask_i$ , such that

$$\sum_i mask_i = 0.$$

This yields that

$$\sum y_i = \sum x_i.$$

**Problem: Fragility.** If the server fails to receive a single  $y_i$  value, then the entire protocol breaks down. It is very fragile!

**Solution: Recovery Step.** Consider this modification to the previous protocol.

1. Each client shares its DH secret  $s_i$  with other clients using a  $t$ -out-of- $n$  secret sharing scheme.
2. The server can request shares of  $s_i$  for any  $y_i$  value it did not receive.

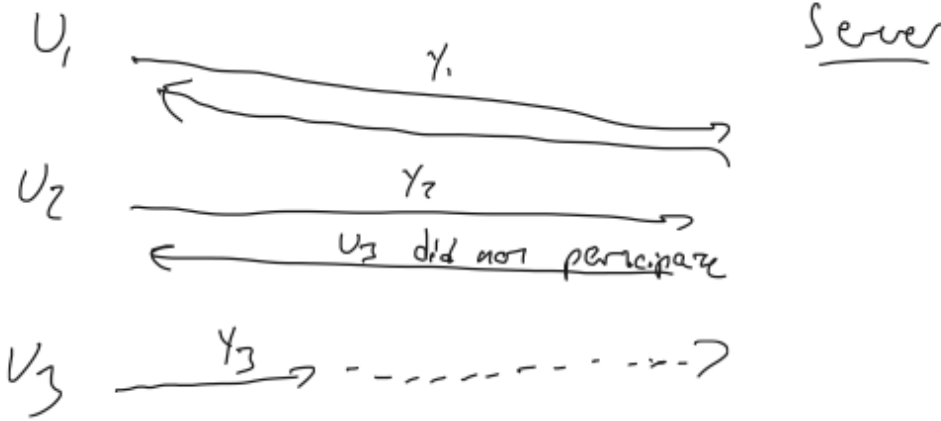


Figure 40: The recovery step modification to the basic private data collection protocol

**Problem: Requires Semihonesty.** Malicious users can break privacy of users, and delayed messages also break privacy.

**Solution: Individual Masks** Users can apply individual mask via

$$y_i = x_i + \sum_{i < j} k_{i,j} - \sum_{i > j} k_{i,j} + r_i$$

where users also secret share  $r_i$  among the other clients.

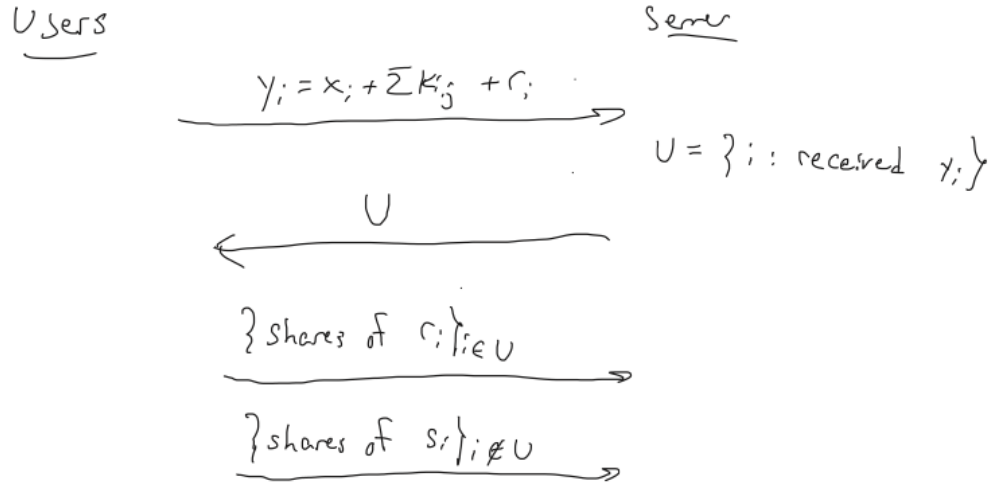


Figure 41: The individual masks modification to the basic private data collection protocol

### 11.1 Prio

**Protocol 19 (Prio).** The context is: many clients,  $1 < n_{\text{servers}}$ , secure against  $n - 1$  semi-honest servers, each client has input  $x_i$ , and we want to compute  $\sum x_i$ . The simple protocol is:

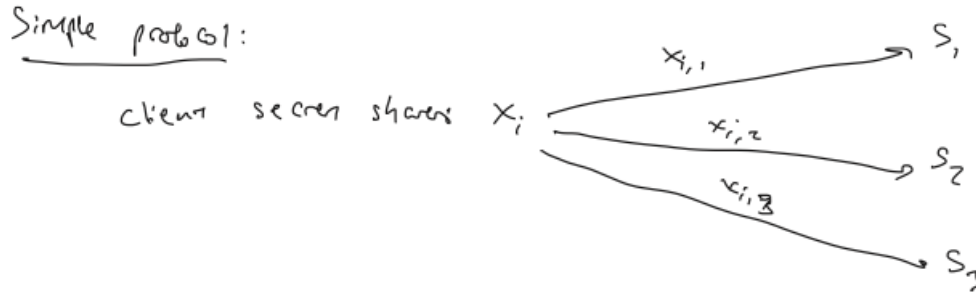


Figure 42: The simple version of the Prio protocol

To extend this simple protocol, the challenge that needs to be addressed is that we need to enforce that the clients' inputs satisfy some predicate i.e.  $C(x) = 1$  for some predicate  $C$ . To accomplish this, there are three options.

#### 11.1.1 Prio Protocol Extension 1

The server can run an MPC protocol evaluating  $C(x)$  and check if the result is 1.

#### 11.1.2 Prio Protocol Extension 2

The client can locally evaluate  $C(x)$ , and determine the value on every wire of the circuit. Then servers can verify correctness by secretly evaluating a  $O(1)$ -depth circuit.

#### 11.1.3 Prio Protocol Extension 3

**Client-side.** The goal is to evaluate  $C(x)$ . Let  $M$  be the number of multiplication gates in  $C$ .

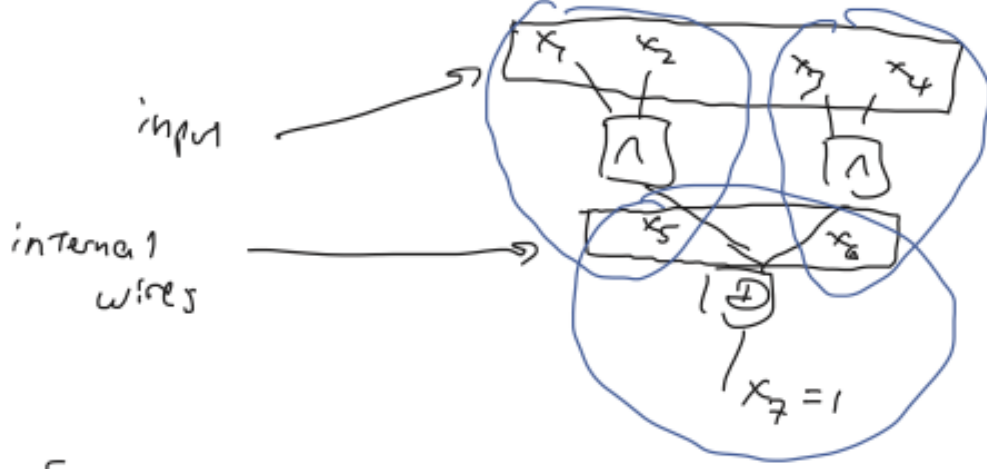


Figure 43: Option 2 for extending the simple Prio protocol

1. Let  $f$  be a polynomial such that  $f(t)$  = the value on left input wire of gate  $t$ , for  $1 \leq t \leq M$ .
2. Let  $g$  be a polynomial such that  $g(t)$  = the value on right input wire of gate  $t$ , for  $1 \leq t \leq M$ .
3. Let  $h = f \cdot g$  i.e.  $h(t)$  = the value of output wire of gate  $t$ .
4. Provide  $[x], [f(0)], [g(0)], [h]$

**Server-side.**

1. Generate  $[\hat{f}]$  and  $[\hat{g}]$  locally.

$$\begin{aligned}
 f_i &= \sum_{j=0}^M f(j) \cdot \gamma_j \\
 [f_i] &= \sum_{j=0}^M [f(j)] \cdot \gamma_j \\
 h(x) &= \sum h_i x^i [h(x)] &= \sum x^i [h_i]
 \end{aligned}$$

The claim two-fold:

1. If a client is honest, then  $\hat{f} = f, \hat{g} = g, \hat{f} \cdot \hat{g} = h$
2. If  $h$  that a client shares is incorrect, then  $\hat{f} \cdot \hat{g} \neq h$

So, the servers check whether

$$P(t) = t \cdot (\hat{f}(t) - \hat{g}(t) - h(t)) \equiv 0$$

i.e. they check if  $P(r) = 0$  at a random  $r$ . If  $P(t) \neq 0$ , then

$$\Pr_r[P(r) = 0] \leq \frac{\deg(P)}{\mathbb{F}}$$

Some more details:

- servers agree on  $r$
- each server locally computes  $[r \cdot \hat{h}(r)], [r \cdot \hat{f}(r)], [r \cdot \hat{g}(r)]$

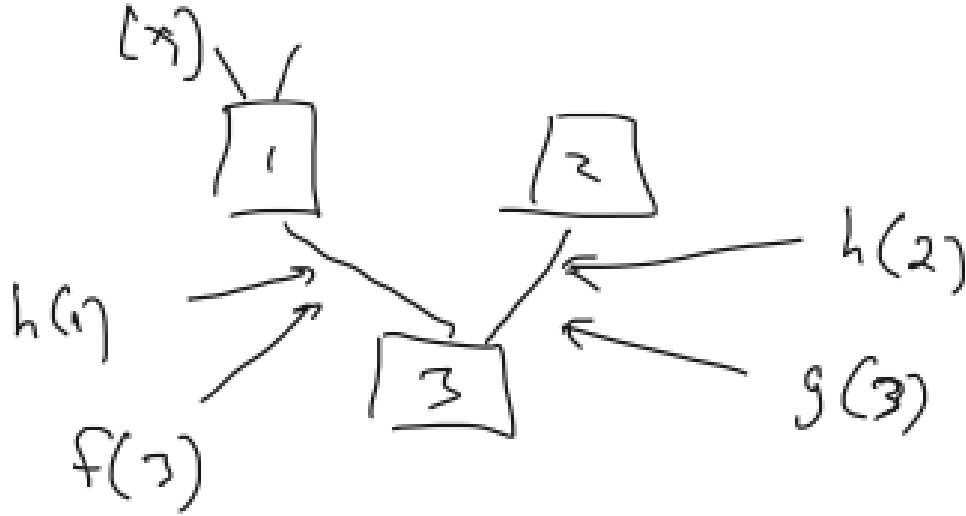


Figure 44: Example of calculations for Prio extension 3 server-side

- each client will also share Beaver triple  $[a], [b], [c], c = ab + \delta$
- servers use Beaver triple to compute  $[r \cdot \hat{f}(r) \cdot \hat{g}(r) + \delta]$
- servers check that  $[r(\hat{f}(r) \cdot \hat{g}(r) - h(r))] = [0]$  and that  $t \cdot \hat{f}(t) \cdot \hat{g}(t) - h(t) + \delta = 0$ .



## 12 Glossary

### 12.1 Protocols

- Multi-Party Computation (MPC):
  - **GMW.** multiparty setting, round complexity  $O(d)$  where  $d$  is the circuit depth (number of non-parallelizable AND gates)
  - **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  $\mathbb{F}_p$
- Circuit Garbling
  - **Yao.** 2-party setting, round complexity  $O(1)$
- Broadcast / Byzantine Agreement (BA)
  - **Broadcast**
  - **Byzantine Agreement**
  - **Dolev-Strong**
- Private Data Collection
  - **Prio**
- To Categorize
  - **Key-Exchange (KE)**
  - **Goldeich-Kahan**
  - **Feige-Shamir**
  - **GMW I Compiler**
  - **GMW II Compiler**