Lecture Notes: Secure Distributed Computation

Henry Blanchette

Spring, 2021

Contents

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 ϵ**-closeness** when

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

where

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

and ϵ is a negligible function/

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

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

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 Π 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 Π 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 Π 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 Π 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

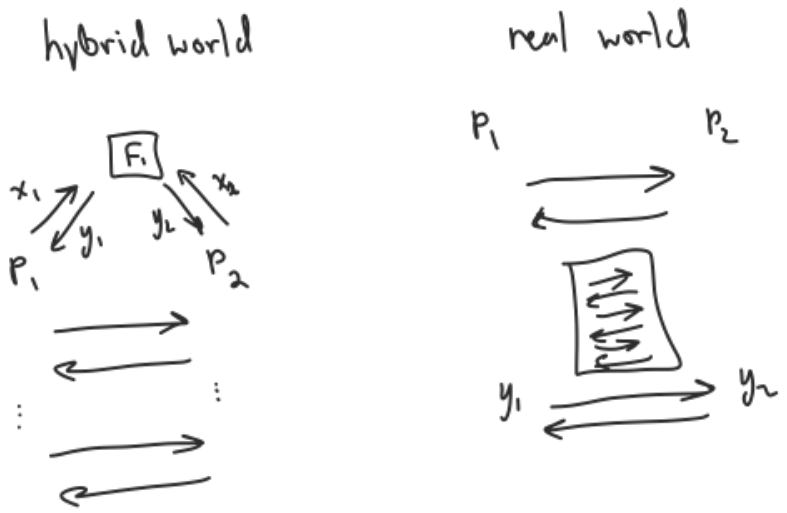


Figure 1: Parallel composition

2 Oblivious Transfer

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

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

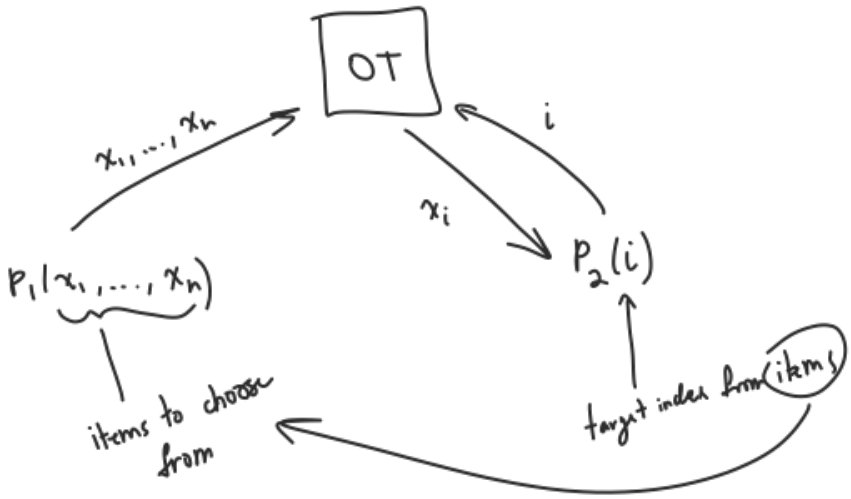


Figure 2: Schematic for OT_1^N

Protocol 1. Semi-honest OT_1^N. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ 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 $\forall x_1, \ldots, x_N, i$, 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:

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

□

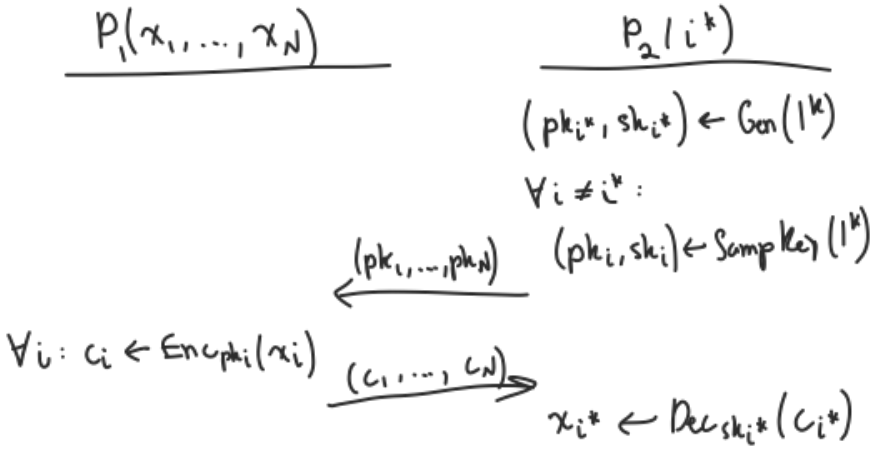


Figure 3: Instance of OT_1^N with CPA-secure public-key encryption scheme

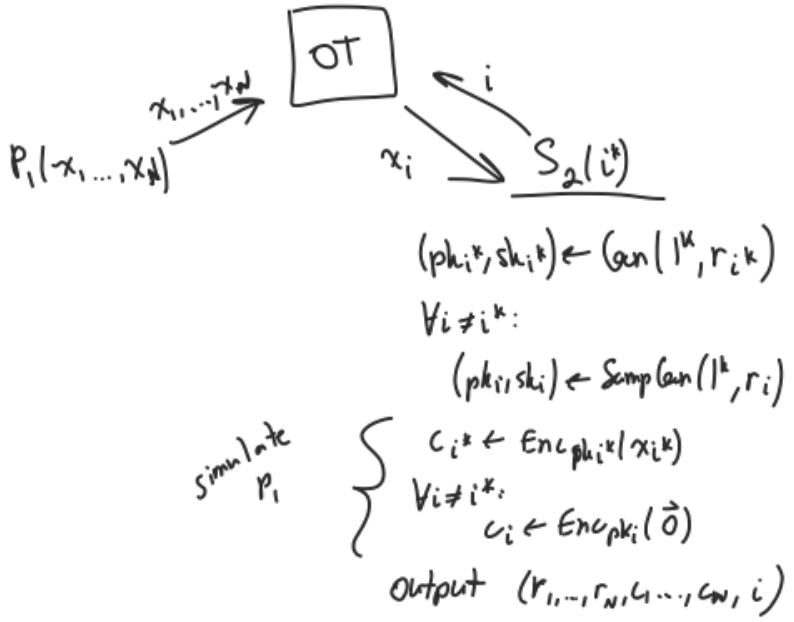


Figure 4: Party 2 corruption case for security proof of theorem 2

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:

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

- 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.

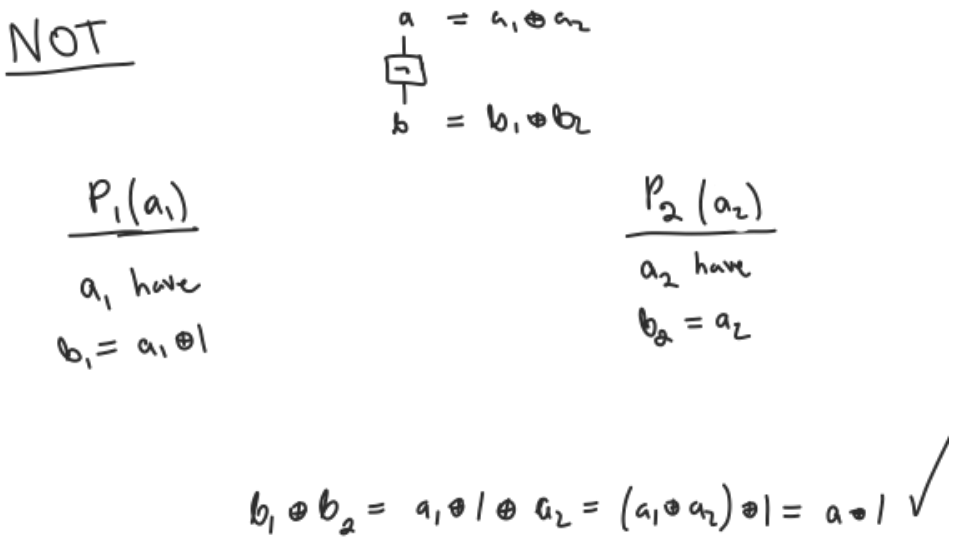


Figure 5: GMW NOT gate

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

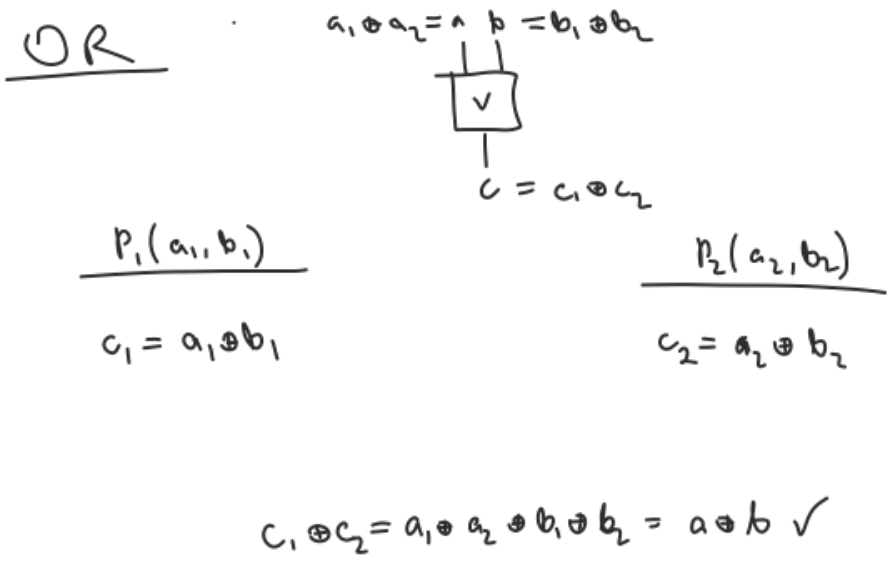


Figure 6: GMW OR gate

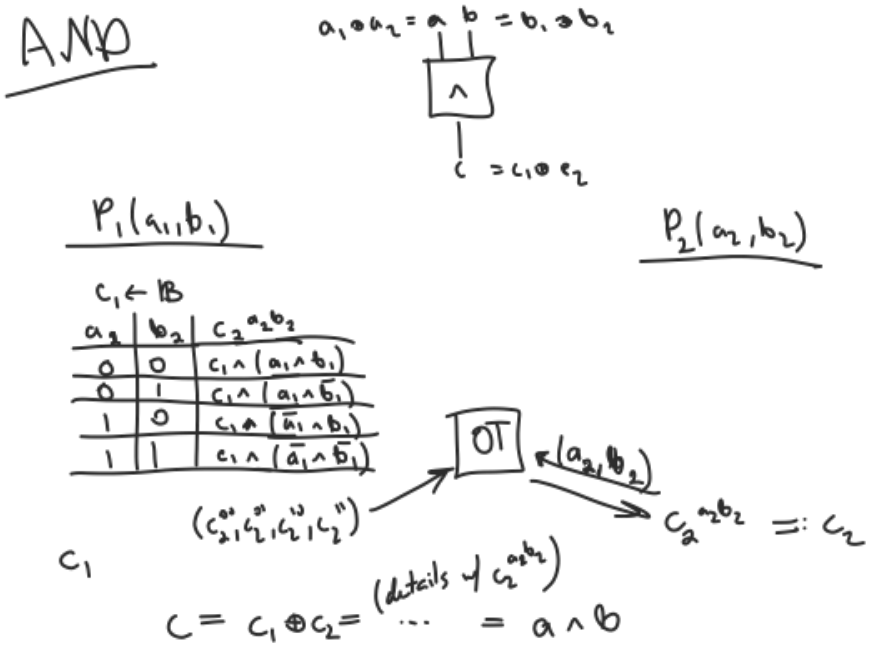


Figure 7: GMW AND gate

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 ith garbled gate can be computed as

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

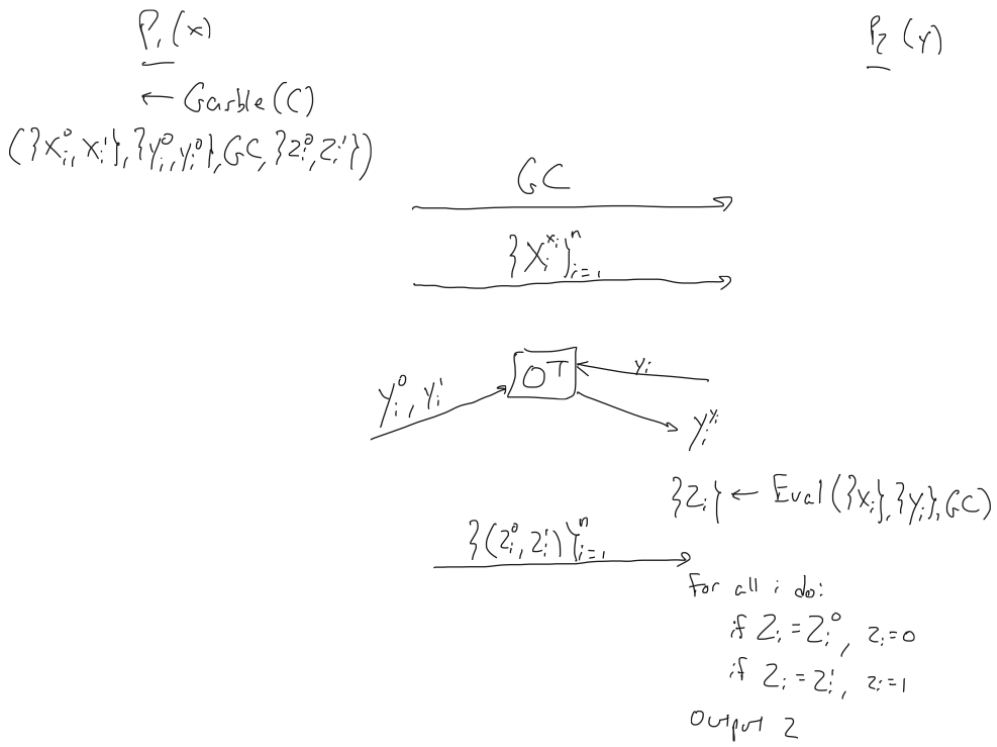


Figure 8: Garbling demonstration using OT

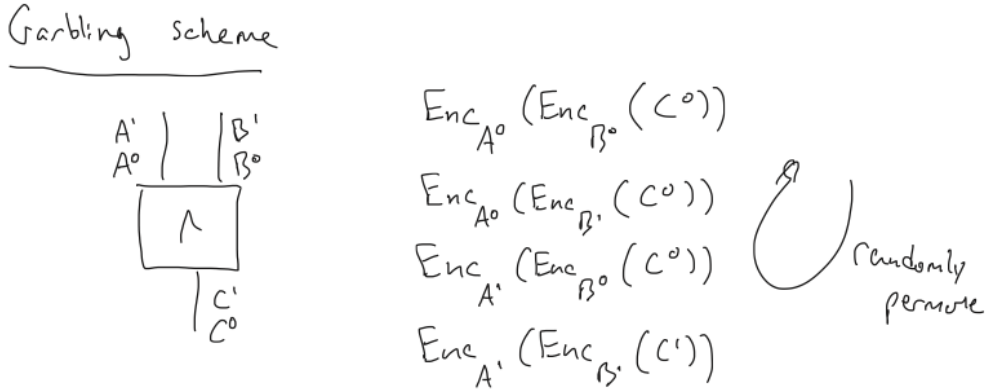


Figure 9: Garbling demonstration using public-key encryption scheme.

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** Δ: TODO

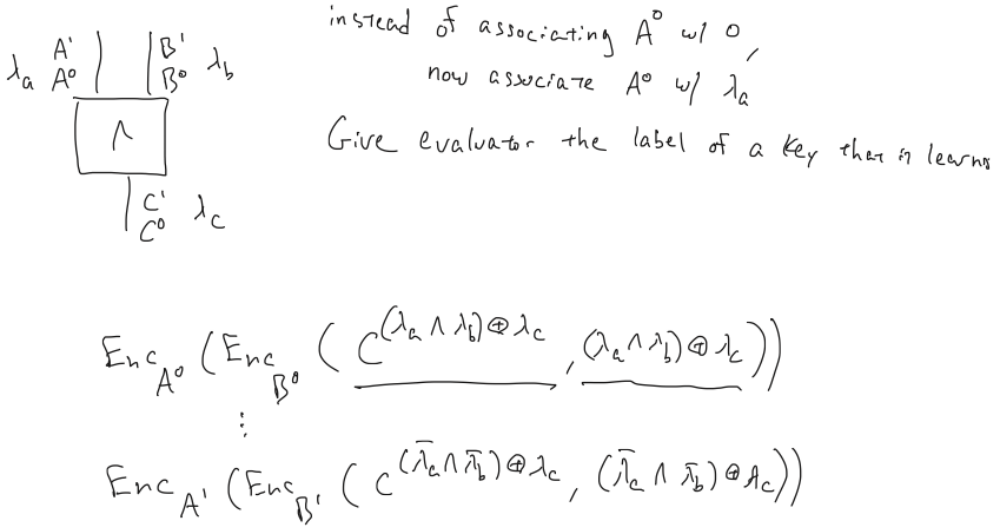


Figure 10: The idea of point-and-permute

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.

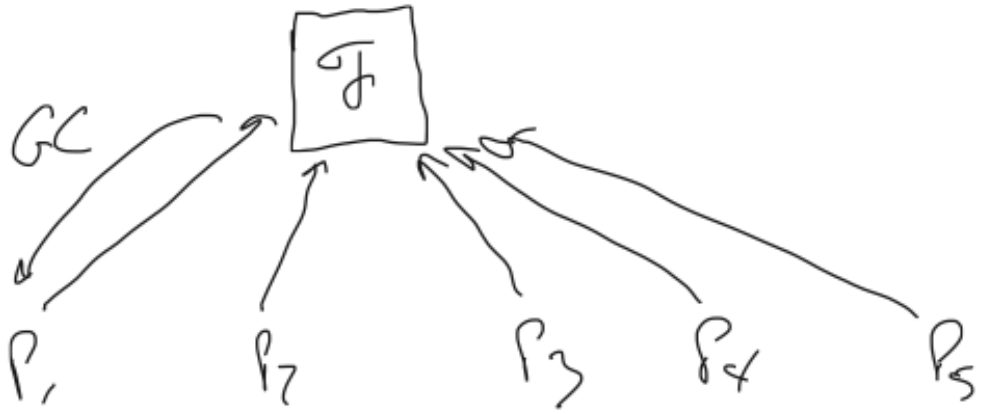


Figure 11: The idea behind the BMR protocol

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

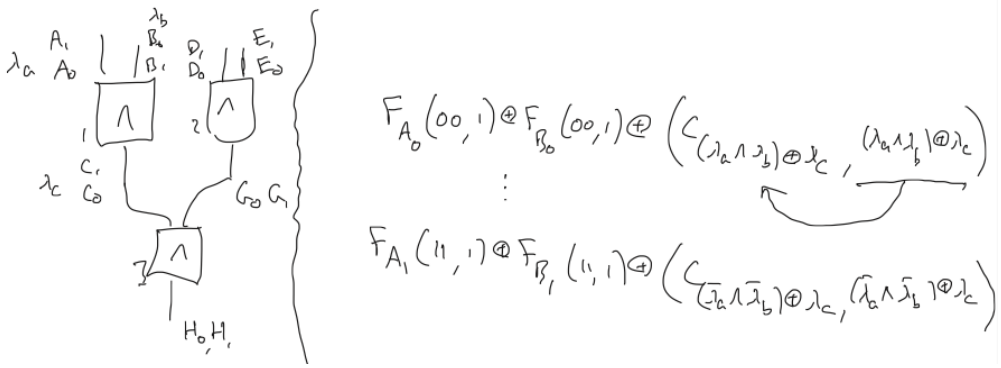


Figure 12: BMR computation example

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 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.

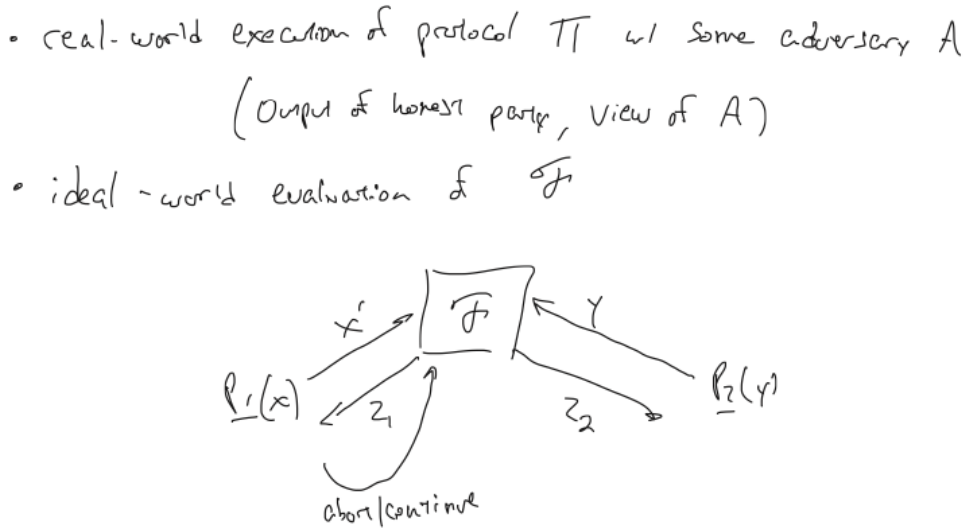


Figure 13: Malicious security

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 □

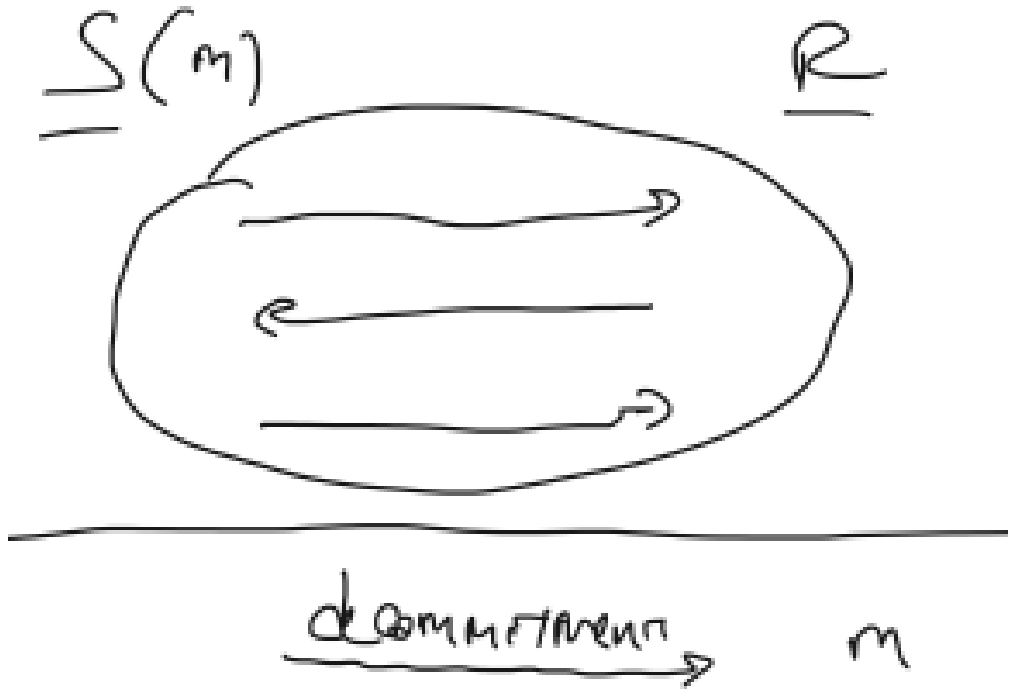


Figure 14: Setup for commitment schemes

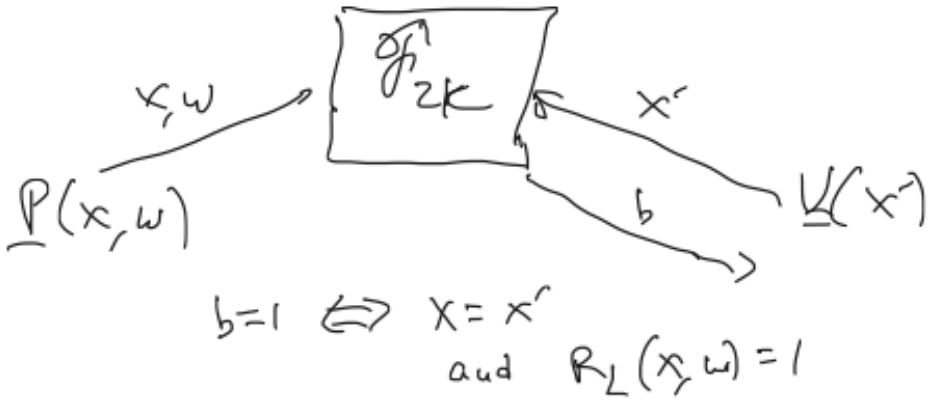


Figure 15: Setup for defining zero-knowledge proofs

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

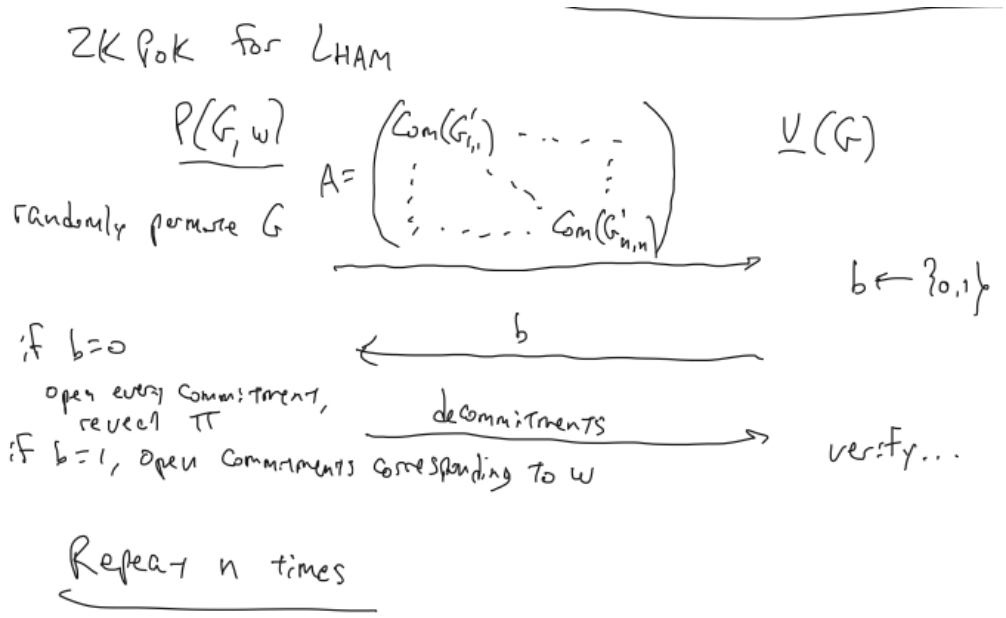


Figure 16: A protocol for ZKPoK for HAM

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 ϵ be the probability that P^* succeeds.

Theorem 5. If $\epsilon > 1/2^n$, then KE computes a witness with probability ϵ. 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 □

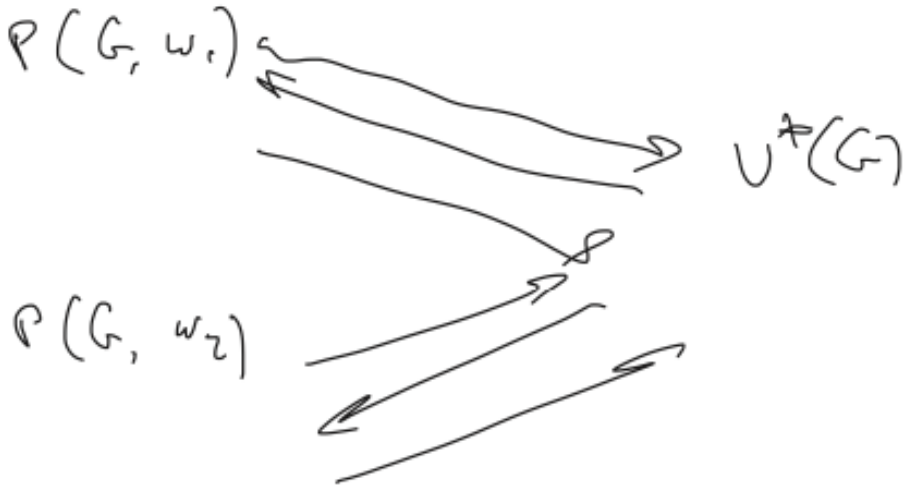


Figure 17: Witness indistinguishability

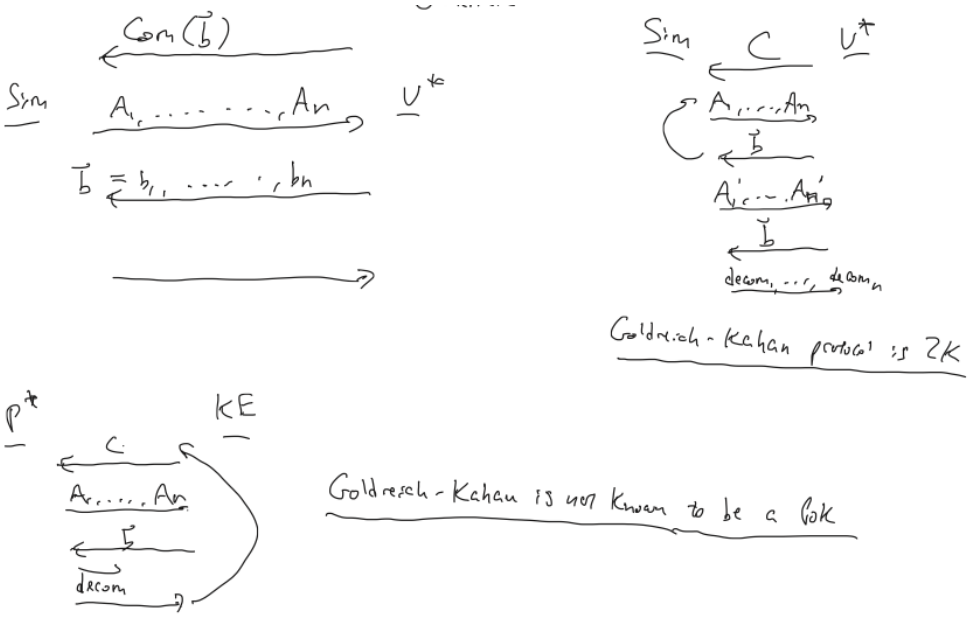


Figure 18: The Goldeich-Kahan protocol

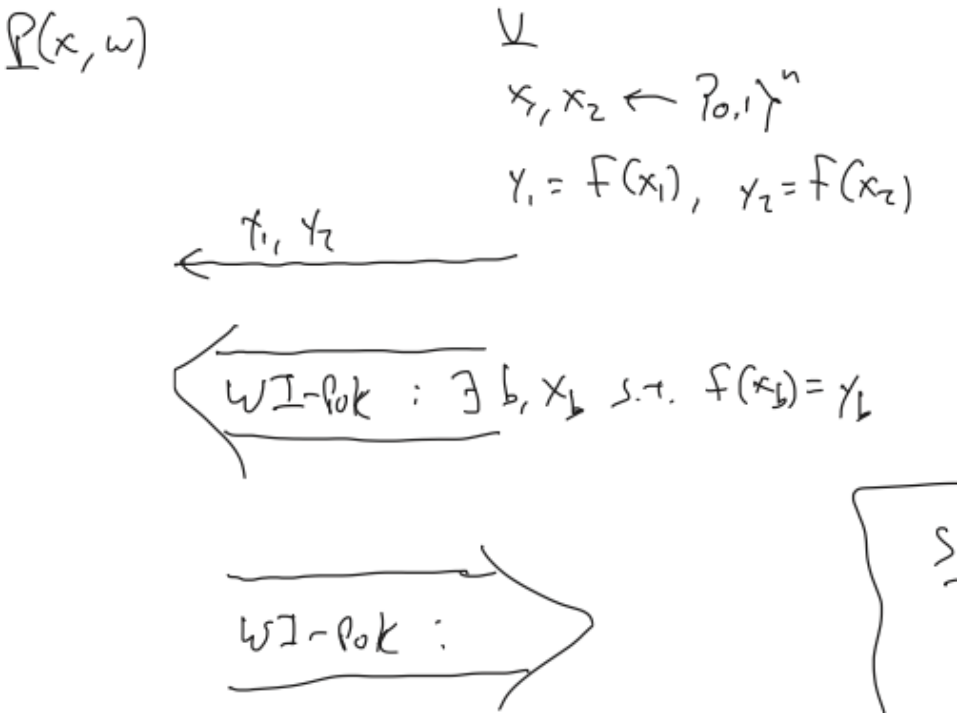


Figure 19: The Feige-Shamir protocol

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 Π, 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.

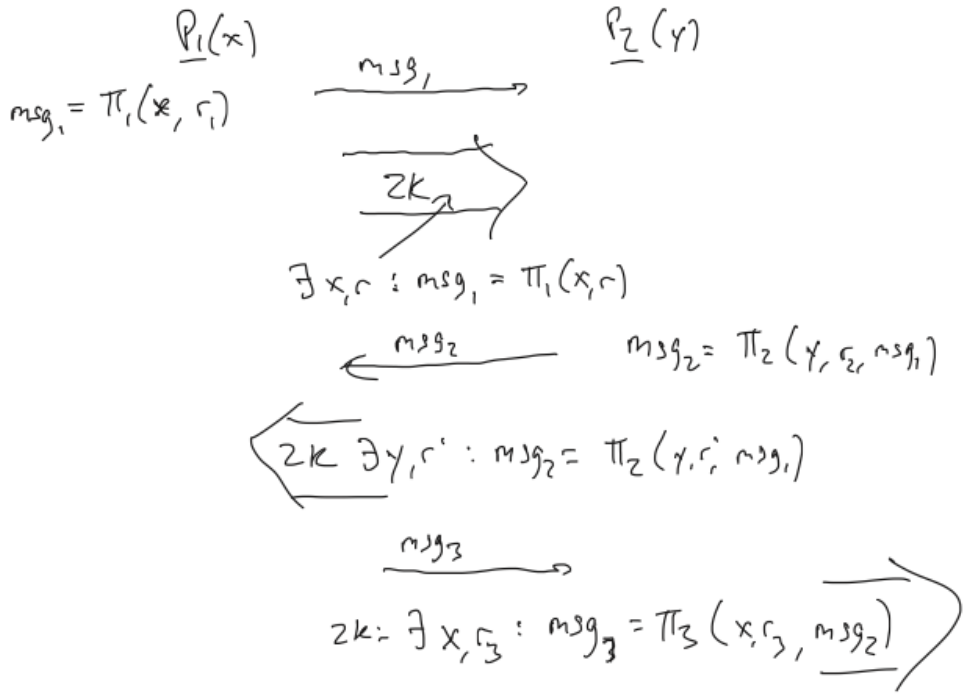


Figure 20: Idea for the GMW I compiler

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

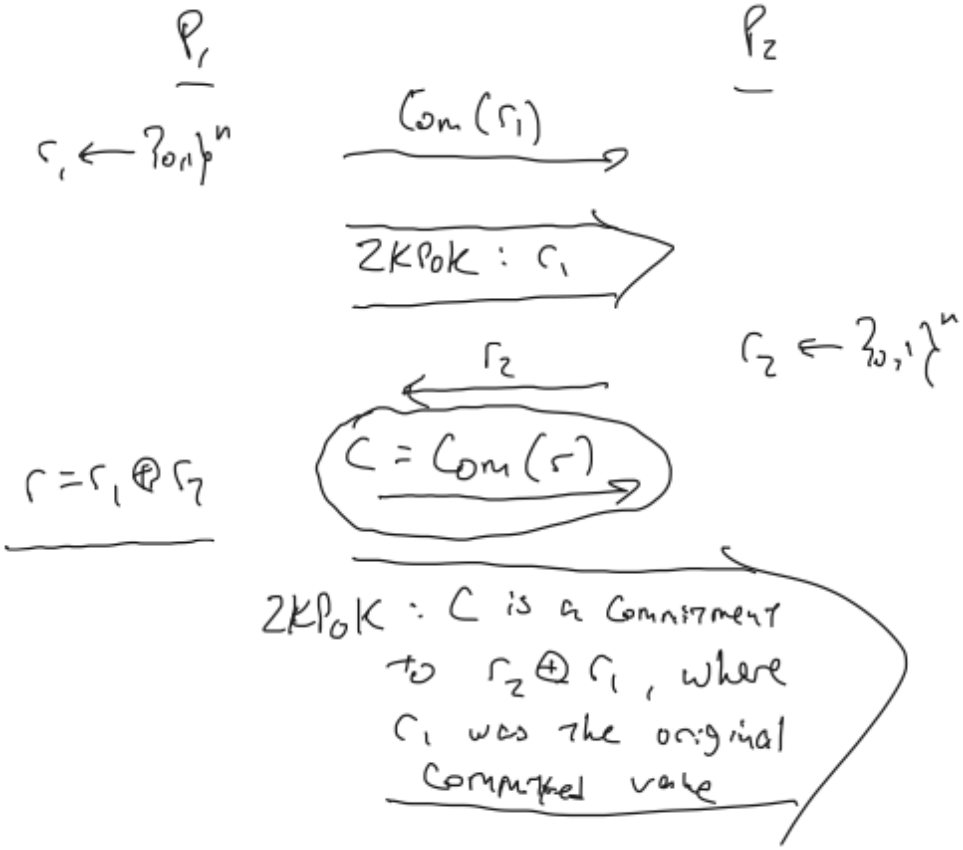


Figure 21: The coin-tossing protocol

- 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 Π 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 Π, 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 Π 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:

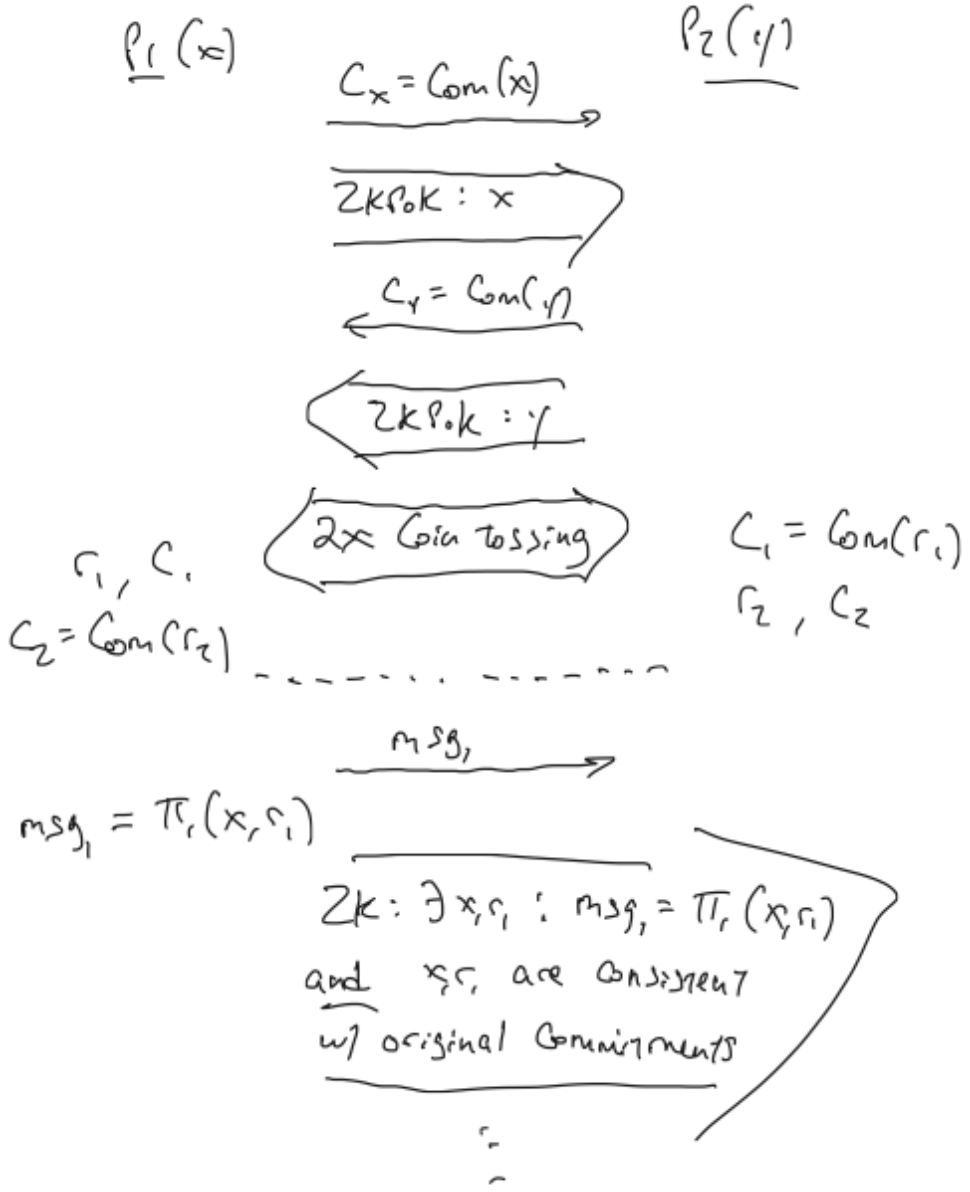


Figure 22: The GMW-I-compiled coin-tossing protocol

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

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

(c) parties run Π on behalf of P_i from then on

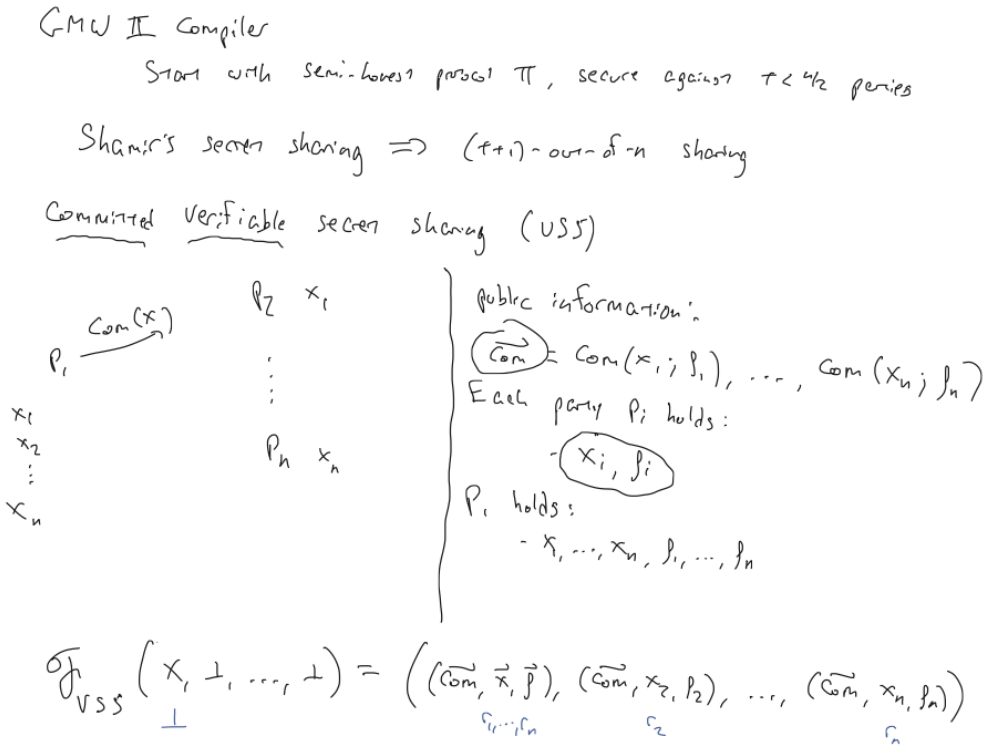


Figure 23: The GMW II compiler

8 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

8.1 Selective-Failure Attack

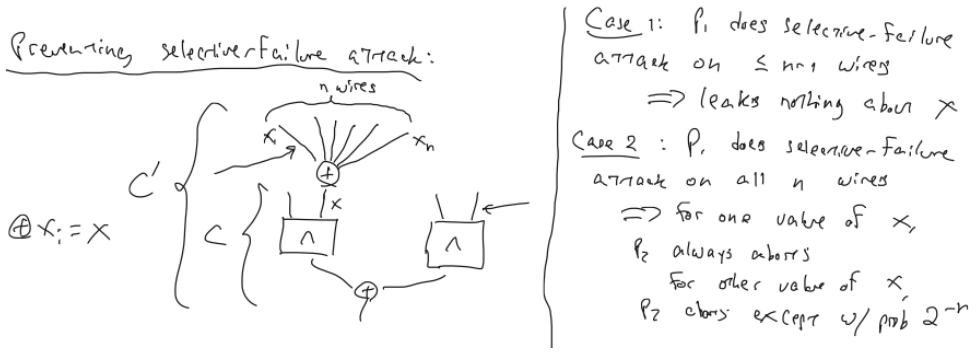


Figure 24: Idea of preventing a selective-failure attack

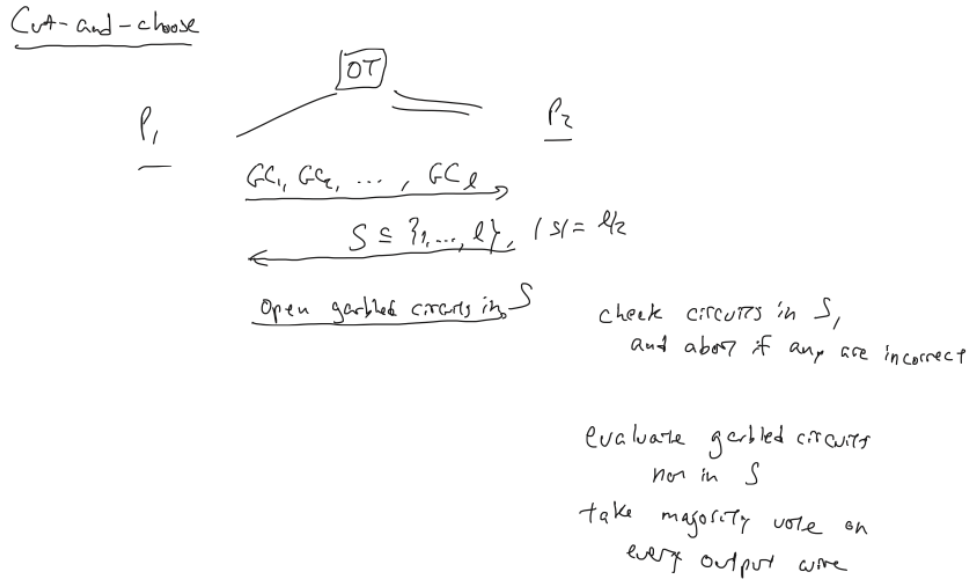


Figure 25: The cut-and-choose protocol to prevent selective-failure attacks

Protocol 13 (Cut-and-Choose)**.** In this protocol, P_1 maximizes its probability of successfully cheating if

$l/4$ circuits are bad. So,

$$\begin{aligned}\mathsf{Prsuccess} &= \frac{\binom{3l/4}{l/2}}{\binom{l}{l/2}} \\ &= \frac{(3l/4)!(l/2)!}{(l/4)!l!} \\ &= \frac{(l/2)\cdots(l/4+1)}{l\cdots(3l/4+1)} \\ &\leqslant 2^{-l/4}.\end{aligned}$$

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 14 (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 $O(|c| \cdot n/\log|c|$

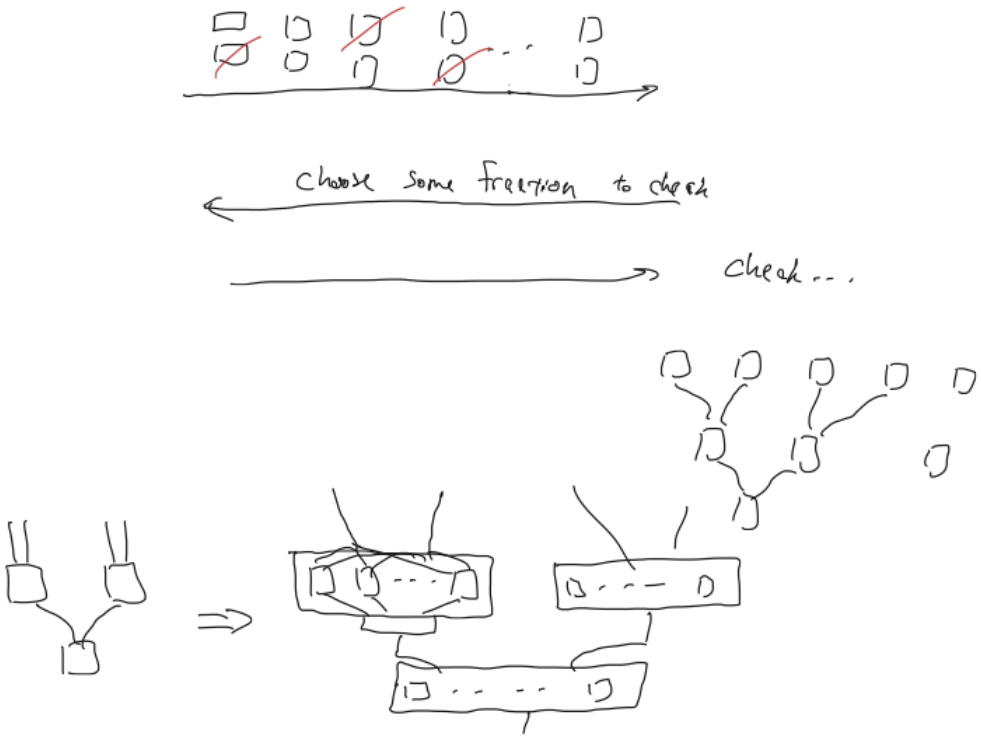


Figure 26: The idea of the LEGO cut-and-choose protocol

TODO: more details at end of lecture 15

9 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 19 (Differential Privacy)**.** Let $D = (x_1, \ldots, x_n) \in X^n$ 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 $M : X^n \to Y$ is **ϵ-differentially private** when for all neighboring D, D' and all $T \subset Y$,

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

M is (ϵ, δ)-differentially private when for D, D', T as above,

$$\mathsf{Pr}[M(D) \in T] \leqslant e^{\epsilon} \cdot \mathsf{Pr}[M(D') \in T] + \delta$$

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

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

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

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

9.1 Laplace Mechanism

Definition 20. For a query $q : X^n \to \mathbb{R}$, 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 $q(D)+$ noise, where noise depends on GS_q and ϵ. Now we need to choose a distribution.

Definition 21. The **Laplace distribution**, $\mathsf{Lap}(\sigma)$, is the distribution such that

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

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

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

Definition 22. The **Laplace mechanism** for differential privacy answers queries q with $q(D) + \mathsf{Lap}(GS_q/\epsilon)$.

Theorem 8. The Laplace mechanism is differentially private.

Proof. TODO: lecture 17 □

Definition 23. The **exponential mechanism** for differential privacy is an abstract mechanism based on a scoring function $\mathsf{score}(D, y)$. Let $GS = \max_y \max_{D \sim D'} |\mathsf{score}(D, y) - \mathsf{score}(D', y)|$. Then the mechanism, on input D, outputs y with probability proportional to $e^{\epsilon \cdot \mathsf{score}(D,y)/2GS}$.

Lemma 1. The exponential mechanism is ϵ-differentially private for any scoring function.

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

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

9.2 Application: Synthetic Data

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

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

using scoring function

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

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

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

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

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

9.3 PAC Learning

Let C be the class of boolean functions $C = c : X \to \mathbb{B}$. For some $c \in C$, a learning algorithm is given $((x_1, c(x_1)), \ldots, (x_n, (c(x_n)))$ 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

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

Use exponential mechanism with scoring function

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

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

$$\mathsf{score}(I, c') \geqslant -\frac{1}{\epsilon}\log(|\mathcal{C}|)$$

which implies that

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

9.4 Centralized versus Local Models

.

Definition 24. Let $R : X \to X'$. Then a querying protocol is **local differentially-private** when

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

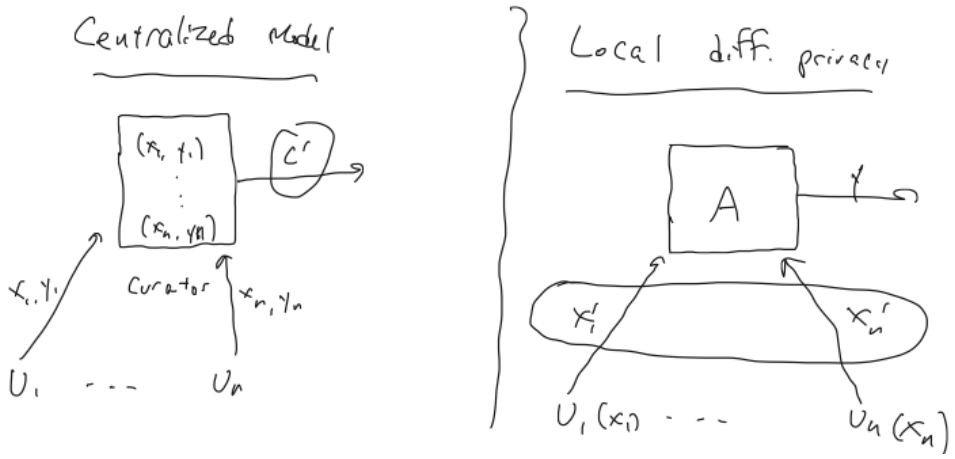


Figure 27: Comparing centralized and local models of differential privacy.

The local differentially-private version of the Laplace mechanism:

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

whereas in the centralized model the last line was

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

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

9.5 Shuffle Model

Idea. To generate a noisy histogram:

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

The expectation is that a γ-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 $\{1, \ldots, k\}$, 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.

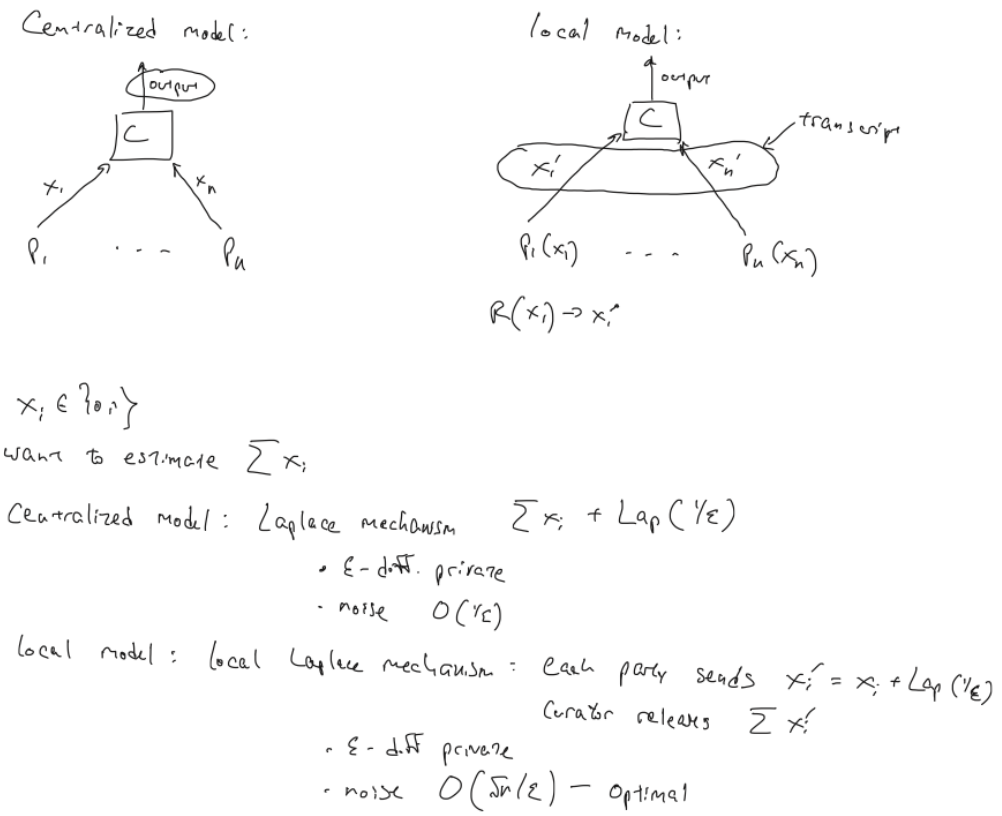


Figure 28: (Again) centralized versus local models of differential privacy

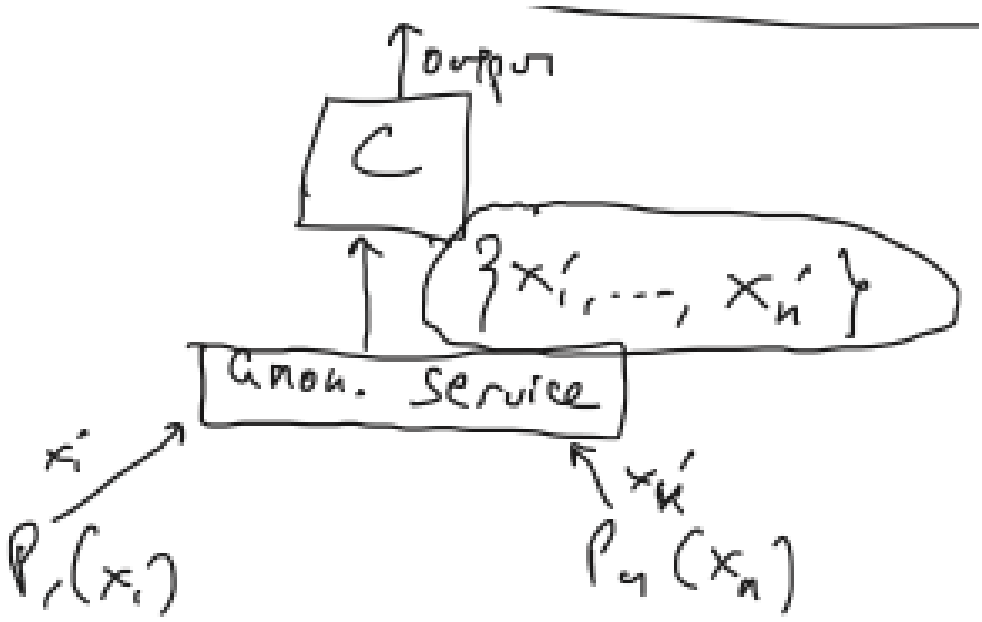


Figure 29: Shuffled model of differential privacy

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

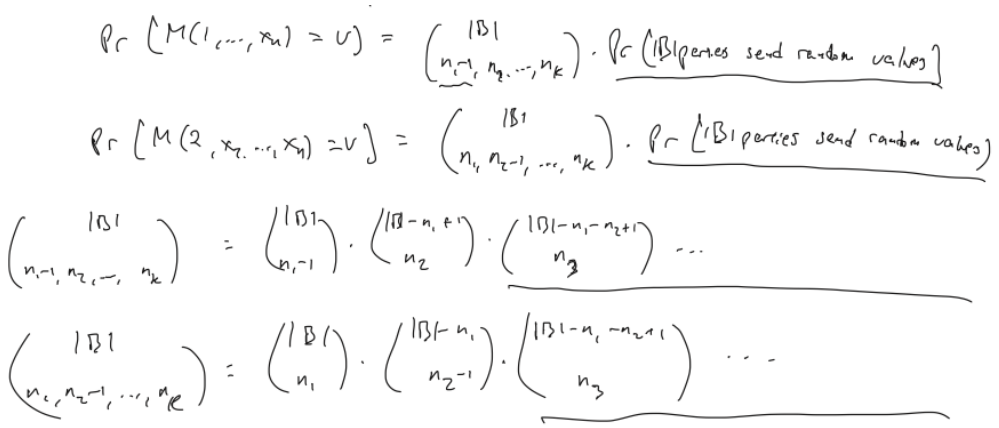


Figure 30: Calculations #1 for shuffled model

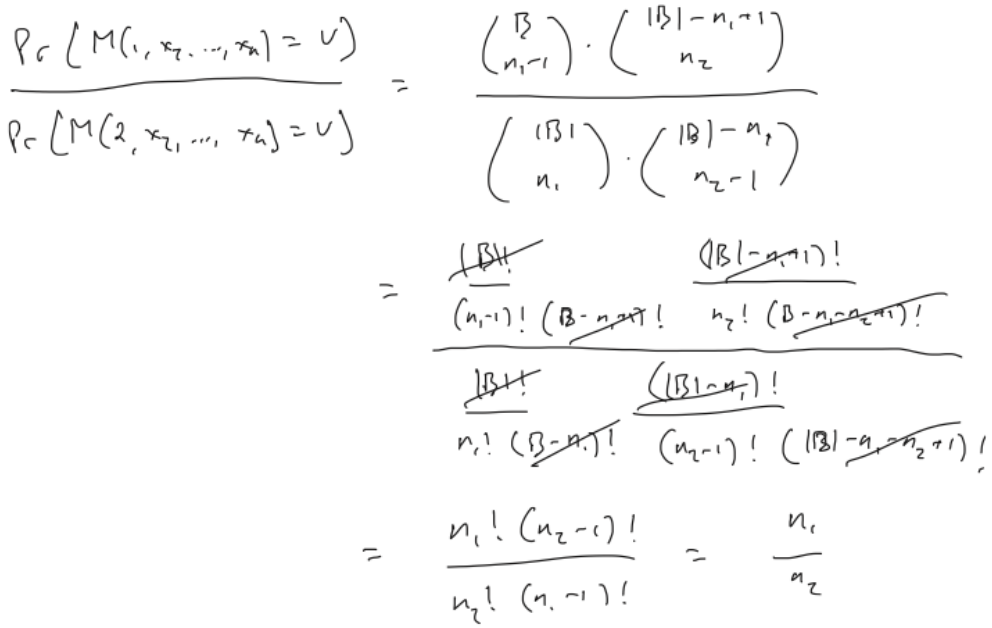


Figure 31: Calculations #2 for shuffled model

Amplification result. Take any R that is ϵ_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.

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

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

Protocol 15 (Shuffled Summation)**.** Each party P_i with input x_i chooses m values $x_i^1, \ldots, x_i^m$ uniformly subject to $\sum_j x_i^j = x_i \mod q$. Then sends $x_i^1, \ldots, x_i^m$ to the anonymization service.

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

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

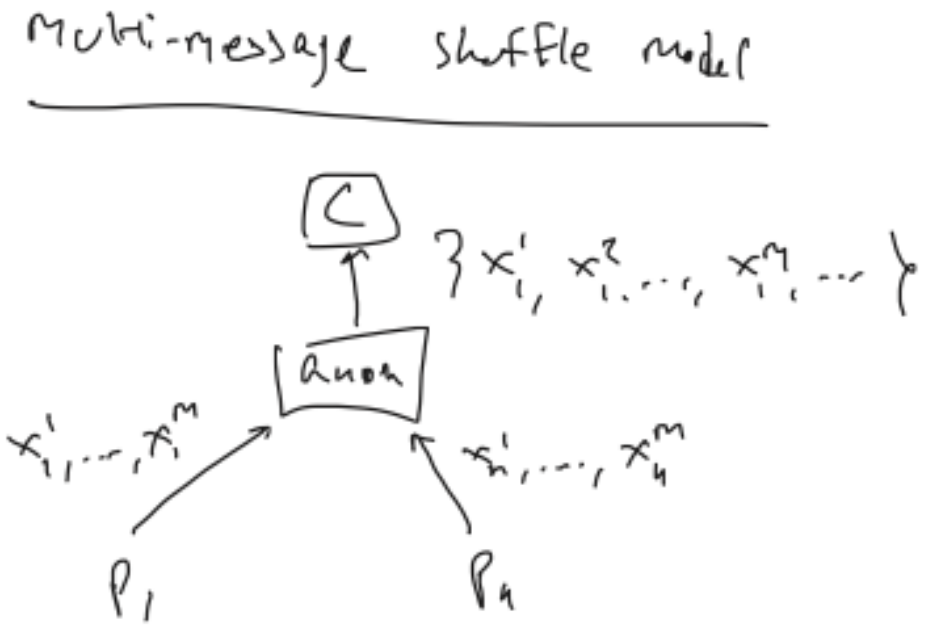


Figure 32: Multi-message shuffled differential privacy model

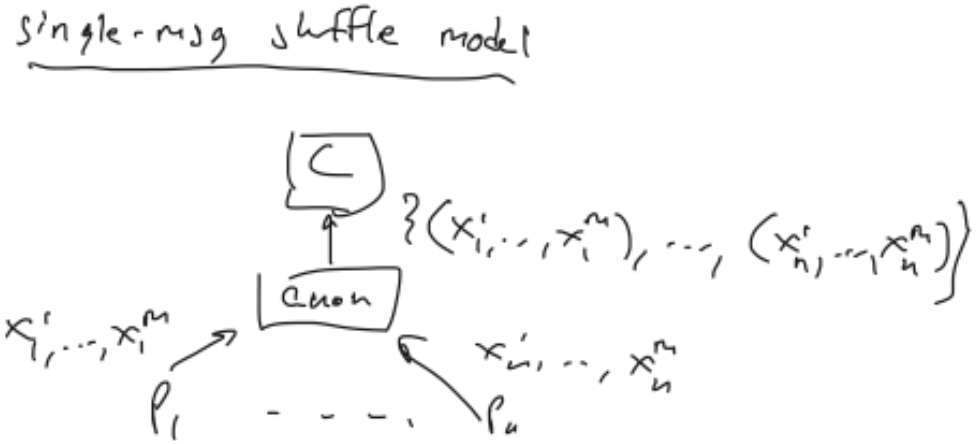


Figure 33: Single-message shuffled differential privacy mdoel

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

Definition 25. A protocol Π 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,

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

Definition 26. A protocol Π is **computationally differentially private** if for all efficient distinguishers D:

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

Example 4. A computationally differentially private protocol.

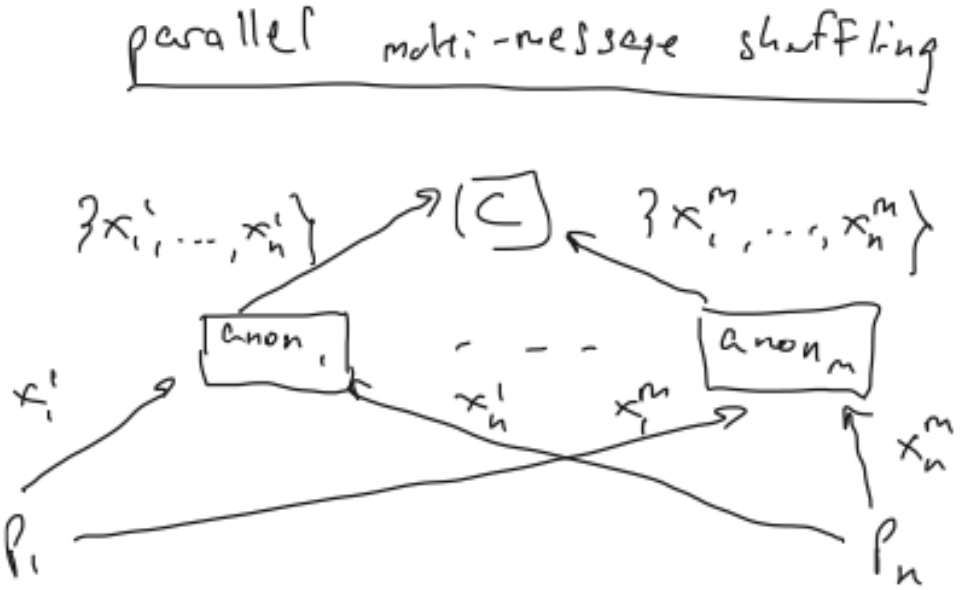


Figure 34: Parallel multi-message differential privacy model

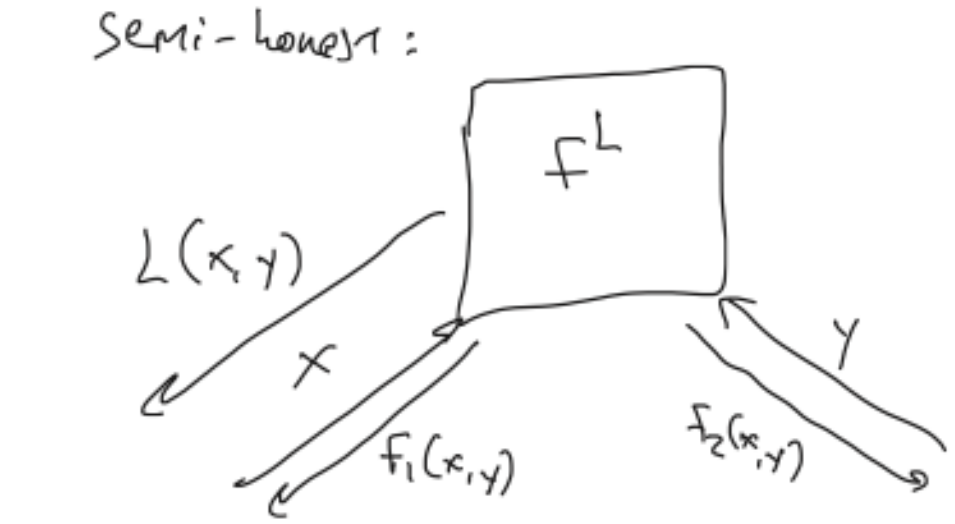
1. Parties set up a threshold homomorphic encryption scheme:
   - have public key pk and $\mathsf{Enc}_{pk}(x_1), \mathsf{Enc}_{pk}(x_2) \implies \mathsf{Enc}_{pk}(x_1 + x_2)$
   - *(threshold)* every party holds a share $sk_i$of secrete key sk
2. Each party sets $\hat{x}_i = x_i +$ noise, then publishes $\mathsf{Enc}_{pk}(\hat{x}_i)$
3. The parties compute $\mathsf{Enc}_{pk}(\sum \hat{x}_i)$
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 (ϵ, δ)-differentially private MPC protocol to compute $\hat{F}$
- use an (ϵ, δ)-differentially private MPC protocol to compute F.

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



Protocol $\mathcal{E}$-diff privately computes F if:

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

Figure 35: TODO: lecture 20

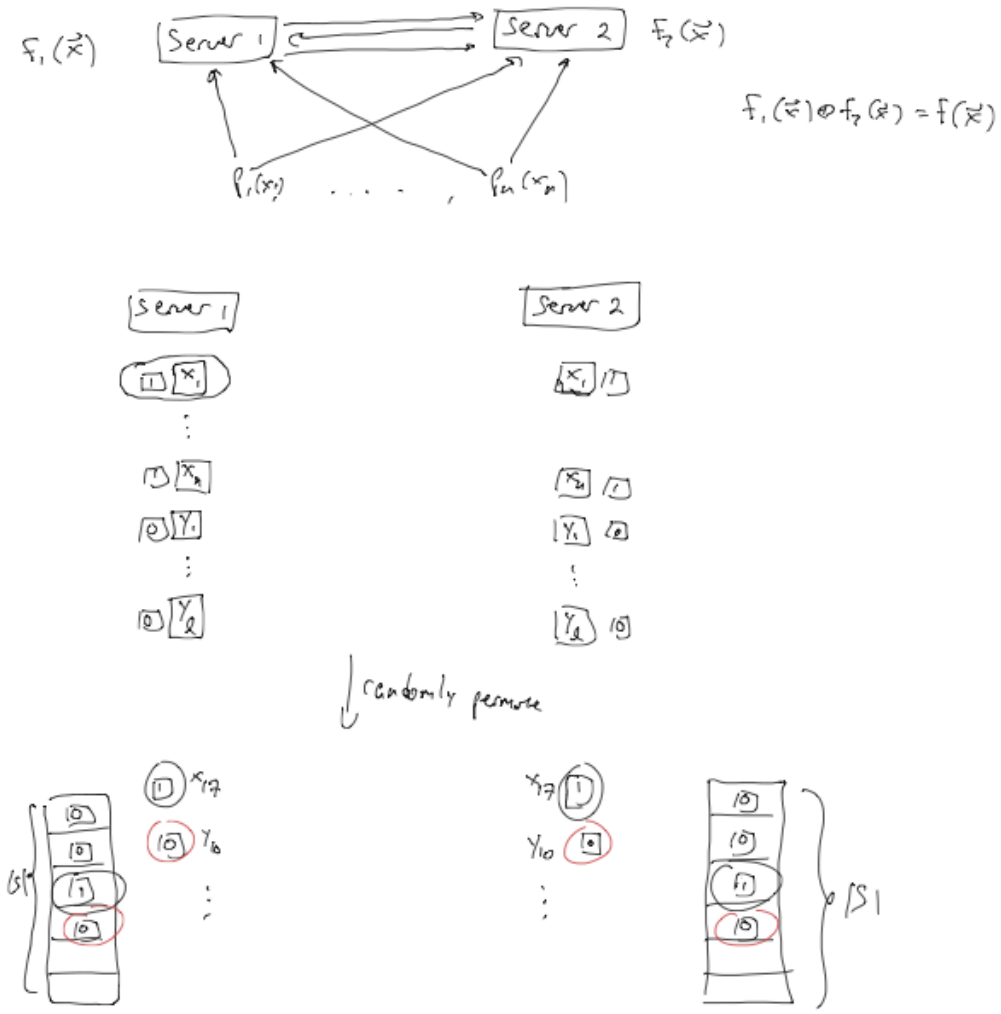


Figure 36: TODO: lecture 20

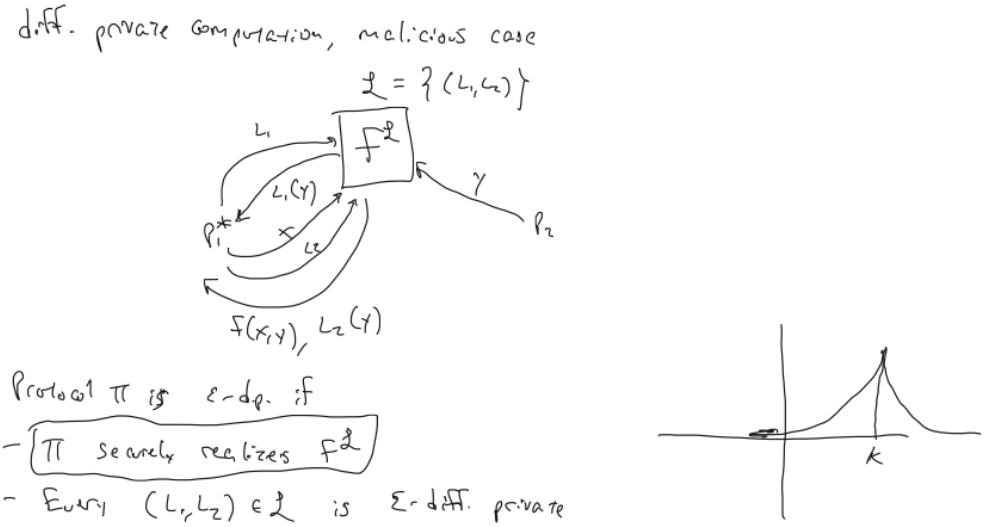


Figure 37: TODO: lecture 20

10 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 16 (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.

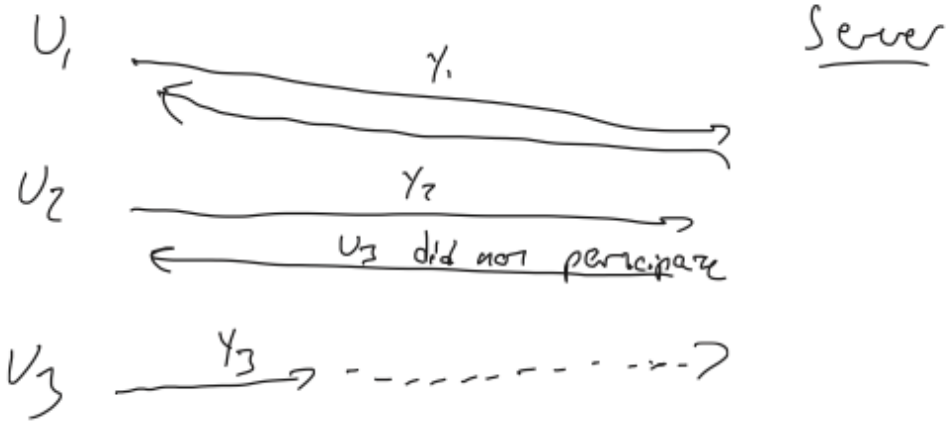


Figure 38: The recovery step modification to the basic private data collection protocol

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

Solution: Individual Masks Users can apply individual mass 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.

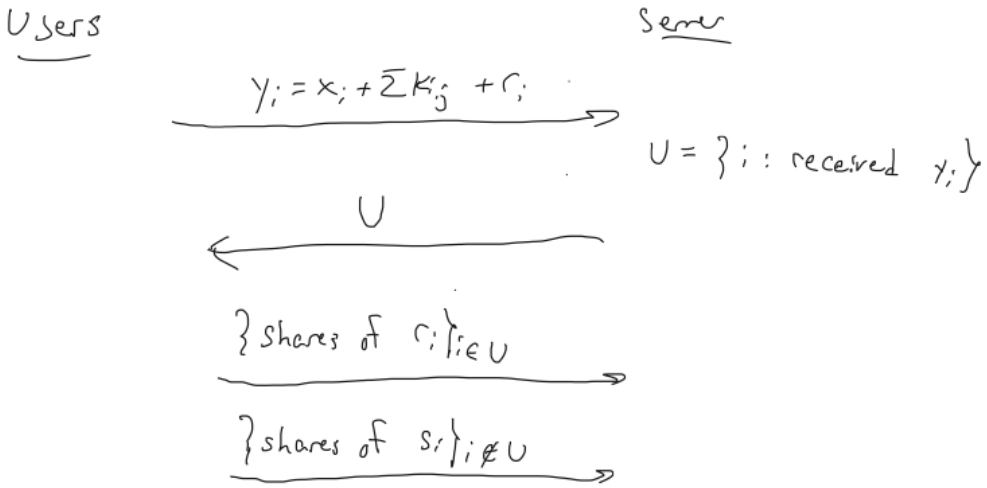


Figure 39: The individual masks modification to the basic private data collection protocol

10.1 Prio

Protocol 17 (Prio)**.** The context is: many clients, $1 < n$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:

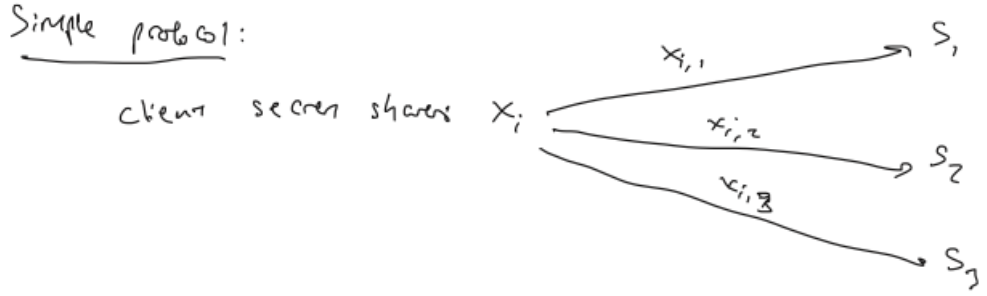


Figure 40: The simple version of the Prio protocol

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.

10.1.1 Prio Protocol Extension 1

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

10.1.2 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 secretely evaluating a $O(1)$-depth circuit.

10.1.3 Prio Protocol Extension 3

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

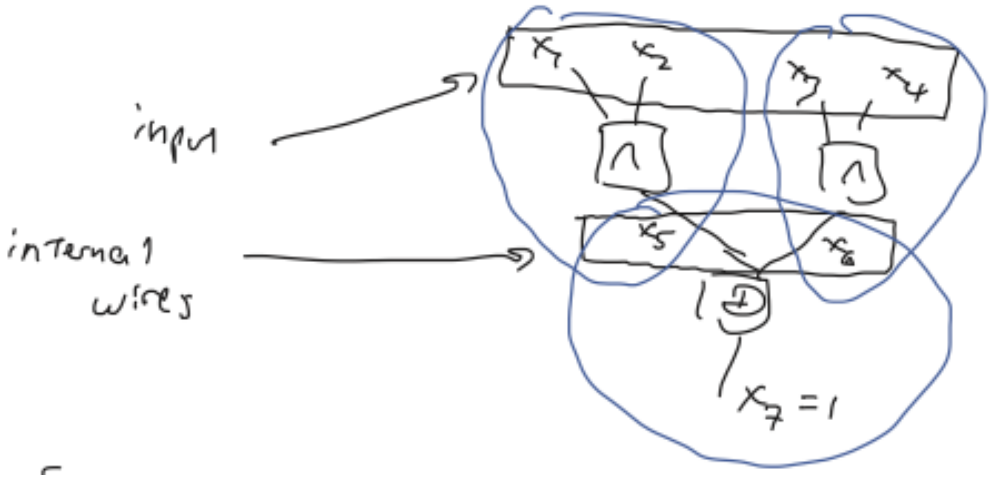


Figure 41: Option 2 for extending the simple Prio protocol

1. Let f be a polynomial such that $f(t) =$ the value on left input wire of gate t, for $1 \leqslant t \leqslant M$.
2. Let g be a polynomial such that $g(t) =$ the value on right input wire of gate t, for $1 \leqslant t \leqslant M$.
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.

$$f_i = \sum_{j=0}^{M} f(j) \cdot \gamma_j$$

$$[f_i] = \sum_{j=0}^{M} [f(j)] \cdot \gamma_j$$

$$h(x) = \sum h_i x^i \qquad [h(x)] = \sum x^i [h_i]$$

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

$$\mathsf{Pr}_r[P(r) = 0] \leqslant \frac{\deg(P)}{\mathbb{F}}$$

Some more details:

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

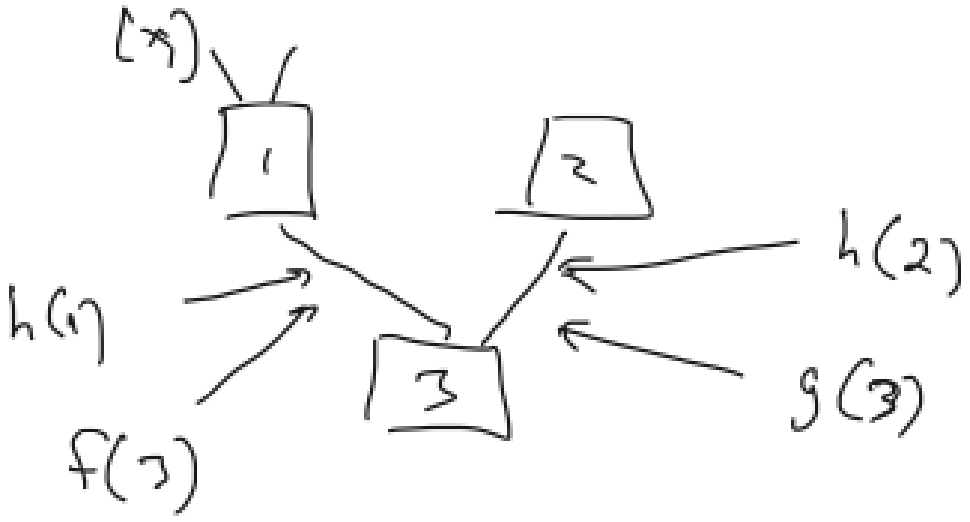


Figure 42: Example of calculations for Prio extension 3 server-side

- 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 $t \cdot \hat{f}(t) \cdot \hat{g}(t) - h(t)) + \delta = 0$.

11 Glossary

11.1 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-Exchance (KE)**
  - **Goldeich-Kahan**
  - **Feige-Shamir**
  - **GMW I Compiler**
  - **GMW II Compiler**