Multiparty Computation

CPT 2006

Ivan Damgård and Jesper Buus Nielsen

BRICS, Århus University

The MPC problem

n players P1, P2, …, Pn

Player Pi holds input xi

Goal: for some given function f with n inputs and n outputs, compute f(x1,…,xn)= (y1,…,yn) *securely*, i.e., we want a protocol such that:

- Pi learns the correct value of yi
- No information on inputs in leaked to Pi, other than what follows from xi and yi.

We want this to hold, even when some of the players behave *adversarially.*

Examples:
Match-making, Electronic Voting,

Generality

MPC is extremely general: a solution implies *in principle* a solution to *any* cryptographic protocol problem.

But note: not all problems can be modelled by computing a single function securely.
Example: Secure electronic payments – is more like secure computation of several functions, keeping track of some state information in between.

Not a problem, however: the definition we will see is fully general, and the protocols we describe are actually fully general as well, although they are phrased as solutions for computing a single function, for simplicity.

Modelling Adversarial Behavior

Assume one central *adversary* Adv.
Adv may *corrupt* some of the players and use this to learn information he should not know, or mess up the results.

When Pi is corrupted, Adv learns complete history of Pi.

An adversary may be

- **Passive or Active**: just *monitor* corrupted players or take full *control*.
- **Static or Adaptive**: all corruptions take place before protocol starts, or happen dynamically during protocol (but once you're corrupt, you stay bad).
- **Unbounded or probabilistic polynomial time**

Goal of MPC, a bit more precisely:

Want protocol to work *as if* we had a trusted party T, who gets inputs from players, computes results and returns them to the players, hence:

Adv may decide inputs for corrupted players, but honest players get

Bounds on corruption

If Adv can corrupt an *arbitrary* subset of players, most problems cannot be solved – for instance, what does security mean if everyone is corrupted?

So need to define some bound on which subsets can be corrupt.

Adversary Structure Γ: family of subsets of **P**= {P1,…,Pn}

Adv is a Γ*-adversary*: set of corrupted players is in Γ at all times

To make sense, Γ must be monotone: $B \in \Gamma$ and $A \subseteq B$ implies $A \in \Gamma$ i.e. if Adv can corrupt set B, he can choose to corrupt any subset of B.

Threshold-t structure: contains all subsets of size at most t.

Γ is Q3: for any $A1,A2,A3 \in \Gamma$, $A1 \cup A2 \cup A3$ is *smaller* than **P**

Γ is Q2: for any $A1,A2 \in \Gamma$, $A1 \cup A2$ is *smaller* than **P**

Threshold-t structure for $t < n/3$ is Q3

Threshold-t structure for $t < n/2$ is Q2

Why General Access Structures?

-And not just a bound on the number of players that can be corrupt?

Threshold adversaries (where we just bound the number of corruptions) make sense in a network where all nodes are equally hard to break into. This is often not the case in practice.

With general access structures, we can express things such as:
the adversary can break into a small number of the more secure nodes and a larger number of less secure ones.

Modelling Communication

Asynchronous network: adversary sees all messages, can delay them indefinitely.

Some forms of the MPC problem are harder or impossible on such a network, in any case additional complications. So in these lectures:

Synchronous network: communication proceeds in rounds – in each round each player may send a message to each other player, all messages received in same round.

Two main variants:

- *Information Theoretic scenario*: Adv does not see communication between honest (uncorrupted) players ⇒ can get security for unbounded Adv.

- *Cryptographic scenario*: Adv sees all messages sent, but cannot change messages exchanged between honest players ⇒ can only get security for (poly-time) bounded Adv.

Summary

Corruption can be *passive*: just observe computation and mess.

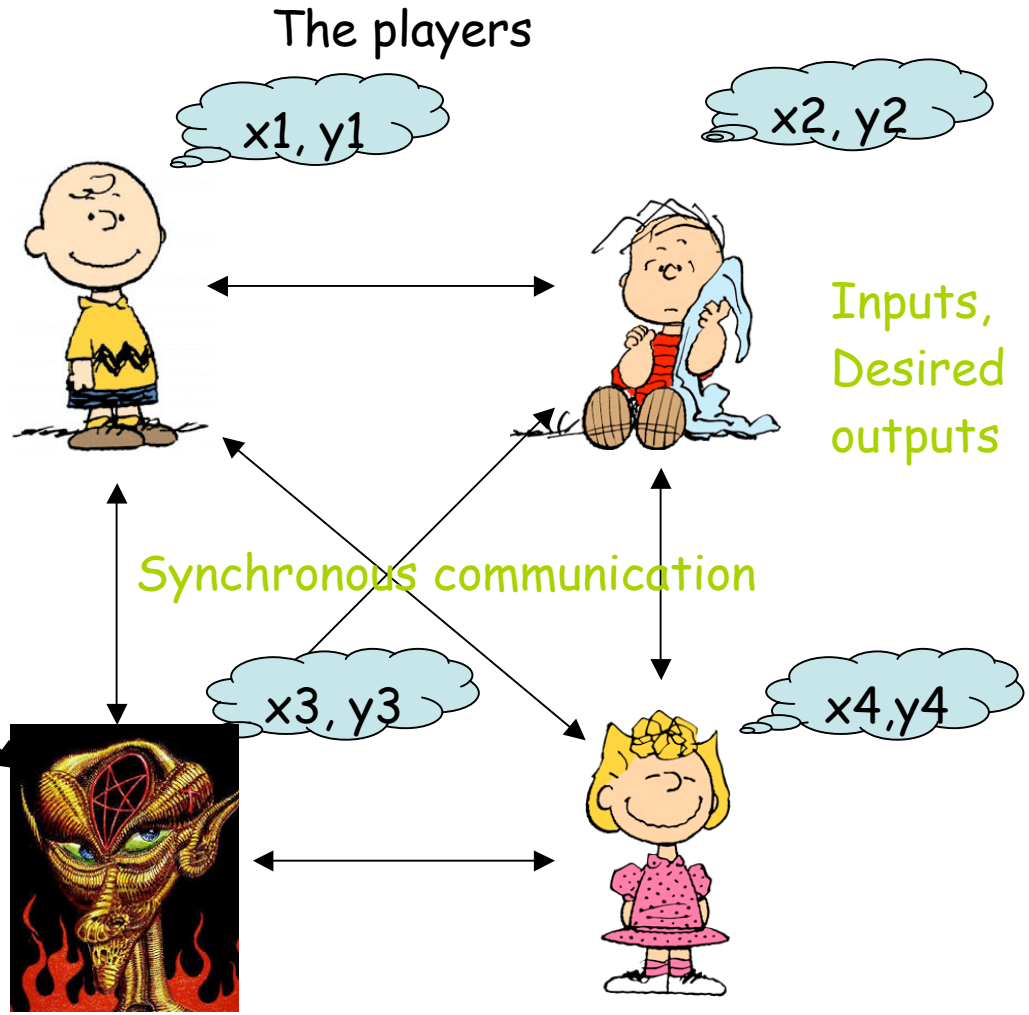
Or *active*: take full control

Adv

Corrupt

Crypto scenario: Adv sees all messages; or

I.T. scenario: no info on honest-to-honest mess.



Adv can choose which players to corrupt *statically* or *adaptively* – but set of corrupted players must be "not too large", i.e., it must be in the given adversary structure _

Known Results, Information theoretic scenario

• Passive, adaptive, unbounded _-adversary:
any function can be securely computed with perfect security iff _ is Q2
(in threshold-t case, if and only if t< n/2)

Meaning of "only if": there exists a function that cannot be computed securely, if condition on _ (t) not satisfied.

• Active, adaptive, unbounded _-adversary:
any function can be securely computed with perfect security iff _ is Q3
in threshold-t case, iff t< n/3

If we assume that a broadcast channel is given for free, and we accept a non-zero error probability, more is possible:

• i.t. scenario with broadcast and active, adaptive, unbounded _-adversary:
any function can be securely computed with small error prob. iff _ is Q2
in threshold-t case, iff t< n/2

Results of [Chaum,Crepeau,Damgaard,'88], [Ben-

Known Results, Cryptographic Scenario

• Passive, adaptive, polynomial time adversary:
Assuming one-way trapdoor permutations exist, any function can be securely computed with computational security if number of corrupted players is < n.

• Active, adaptive, polynomial time _-adversary:
Assuming one-way trapdoor permutations exist, any function can be securely computed with computational security iff _ is Q2
(in threshold-t case, iff t< n/2.)

Results of [Yao'86],
[Goldreich,Micali,Wigderson,'87],[Canetti,Feige,Goldreich,Naor,'96]

Defining Security of MPC (or Protocols in General)

Classical approach: write list of desired properties

- Not good, hard to make sure the list is complete. Ex. Secure voting.

New approach: define an ideal version of the functionality we want. Say that protocol is good if it is "equivalent" to the ideal thing.

-Advantage: from equivalence, we know that the protocol has all good properties of the ideal version – also those we did not think of yet!

We will give a variant of the definition of [Canetti'01] for Universally Composable (UC) Security.

(Variant is due to [Nielsen'03]. Main difference: explicitly considers synchronous networks, original UC was for the asynchronous case).

General Adversarial Activity

Important note before we look at the definition: in general the adversary may be able to do more than corrupt players and control their behavior:

- Most protocols run as a part of a bigger system, fx think of key exchange protocols or electronic payment schemes.
- A real-life adversary will attack the whole system, not just the protocol
- Hence the adversary may have some knowledge on the inputs that the *honest* players will contribute, maybe he can even control them.
- Also, the adversary might get some information on the results that the honest players obtain from the protocol – perhaps even full information.

⇒ In our definition, the adversary should be allowed to choose inputs for honest players and see their results.

General Adversarial Activity, Cont'd

⇒ In our definition, the adversary should be allowed to choose inputs for honest players and see their results.

Question: this is very strange – didn't we say that the adversary should not get information about honest players' results??

Answer: what we said was that ***the protocol*** should not release such information. The protocol cannot help it, if the adversary learns something from other sources, such as the surrounding system.

So we are going to demand that the protocols works as it should, *regardless of what the adversary knows or can control by means that are* ***external*** *to the protocol.*

Concepts in the definition.

All entities (players, adversary, trusted parties) formally speaking are interactive, probabilistic Turing machines. Everyone gets as input security the parameter $k \in \mathbf{N}$, Adv also gets aux. input $z \in \{0,1\}^*$.

The Ideal Functionality (or Trusted Party) T:

- Models the functionality we would like the protocol to implement.
- Cannot be corrupted.
- Can communicate with all players and with the adversary.
- Receives inputs from all players, does computation according to its program and returns results to players.
- Has memory and can be invoked several times - can therefore be used to model any reasonable primitive we could imagine.

Of course, no such T exists in real life, only used as specification of what we would like to have. Definition basically says that a protocol is good (w.r.t.

An example ideal functionality.

An ideal functionality for yes/no elections might look as follows:

- The ideal functionality $T_{election}$ is connected to n parties P1, …, Pn
- In each round it interprets the input from Pi as a bit $bi \in \{0,1\}$
- Then it computes $b = b1+\ldots+bn$
- Then it outputs b to all parties

Imagine that n parties was connected to such a trusted party $T_{election}$ in the real world using completely secure channels.

This would allow them to hold ultimately secure yes/no elections!

The idea behind formulating an ideal functionality T for a given task is that

Plan for Definition.

Define two processes:

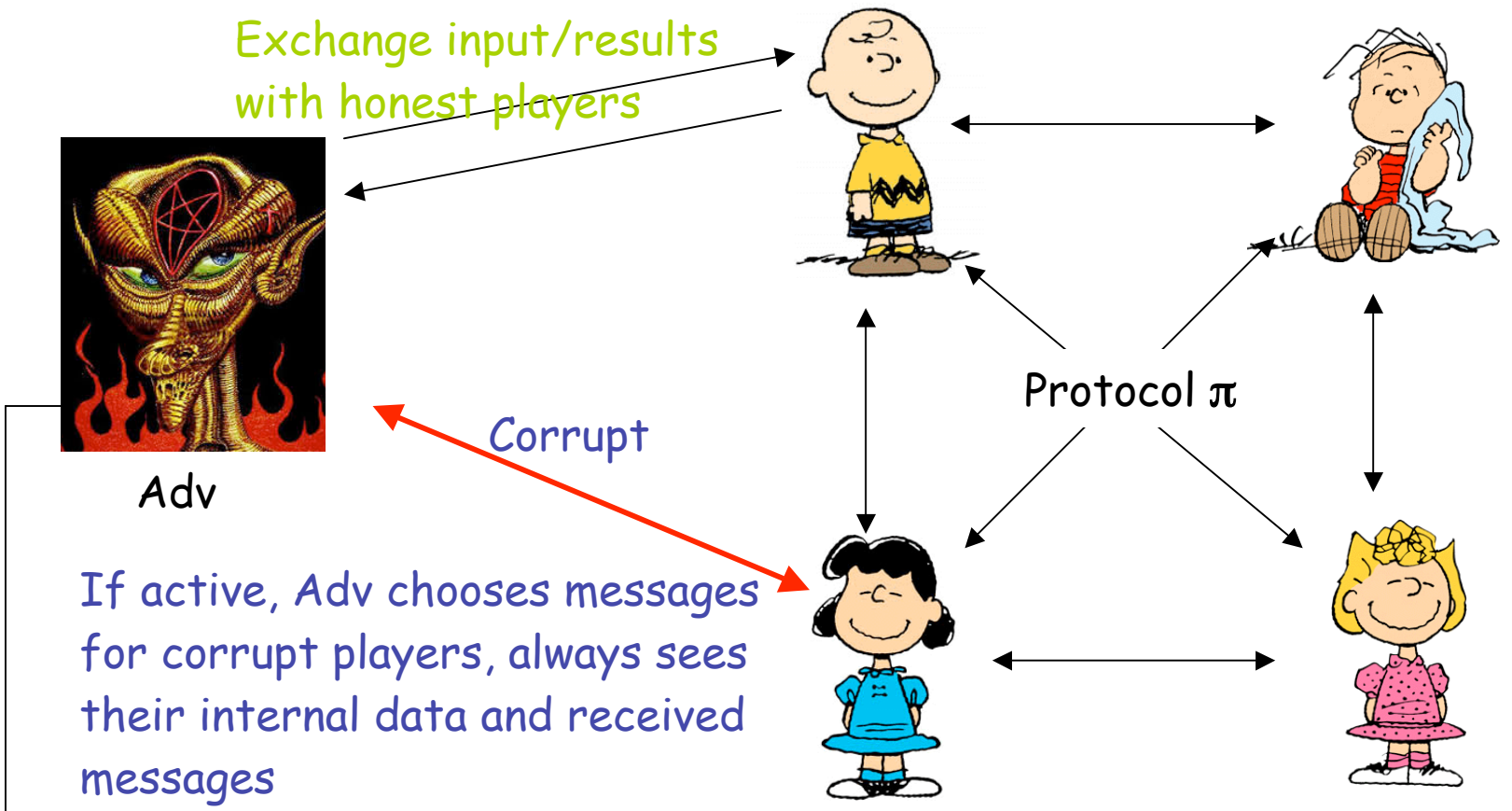
- The Real Process
- The Ideal Process

In the Real Process, we have the adversary Adv and the players executing the protocol π (no trusted party).

In the Ideal Process, we still have the Adv, but π is replaced by the trusted party T (plus a simulator S to be explained later).

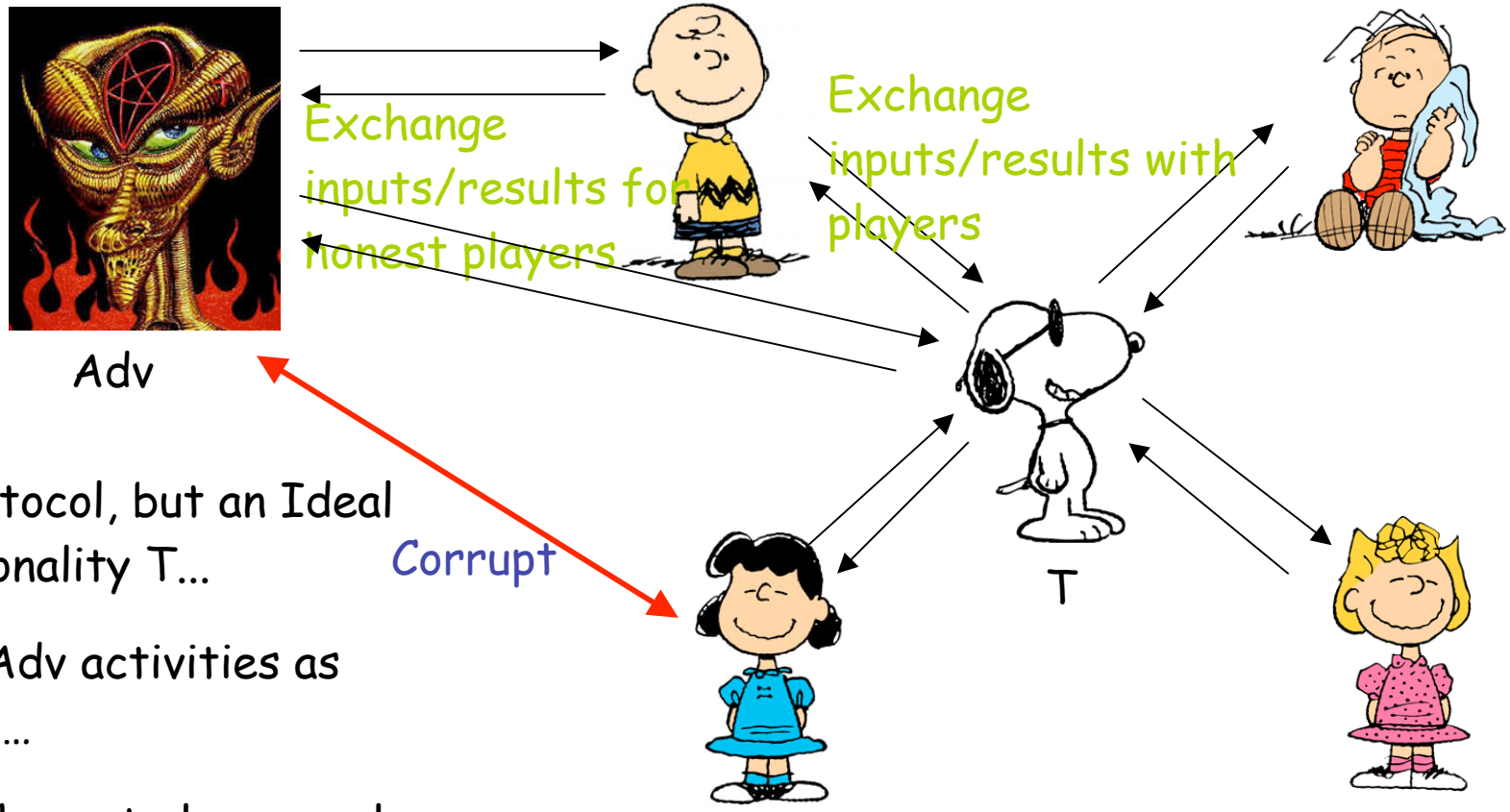
We will say that π *securely realizes* T if Adv cannot tell whether he is in the real or in the ideal process.

The Real Process



When protocol over, Adv outputs a bit. Given Adv, π ,k and z, this is a random variable called $REAL_{Adv, \pi}(k,z)$

The Ideal Process (First Attempt)



No Protocol, but an Ideal Functionality T...

Same Adv activities as before...

So, as honest players only forward info between Adv and T, we may as well abstract them away...

But Adv will expect to see internal data and protocol messages received by corrupted players. No protocol here – so what to do?

The Ideal Process

To make things seem the same to Adv in the real as in the ideal process, we introduce the Simulator..

Who will "magically" produce Adv's view of attacking the protocol, based on the exchange with T.

Adv

Exchange inputs/results for honest players

Corrupt

T

S is told which messages to 'send' on behalf of corrupt parties

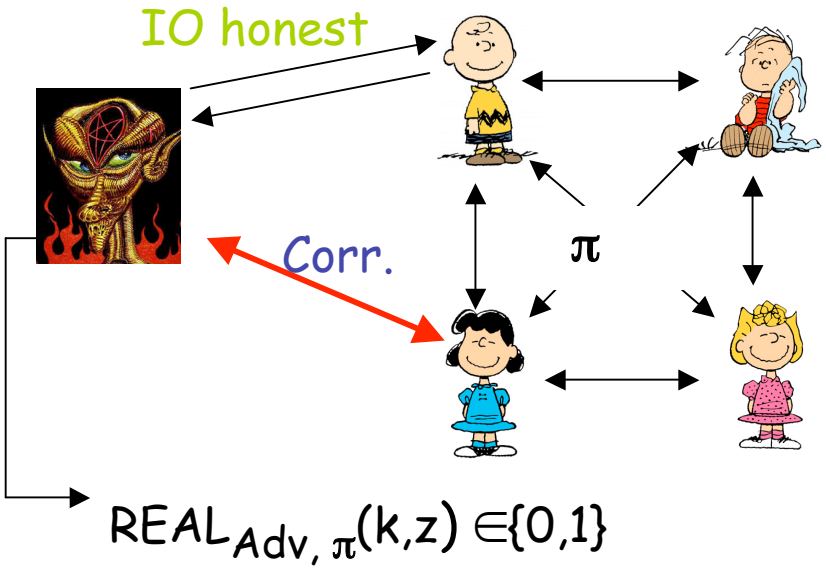
S

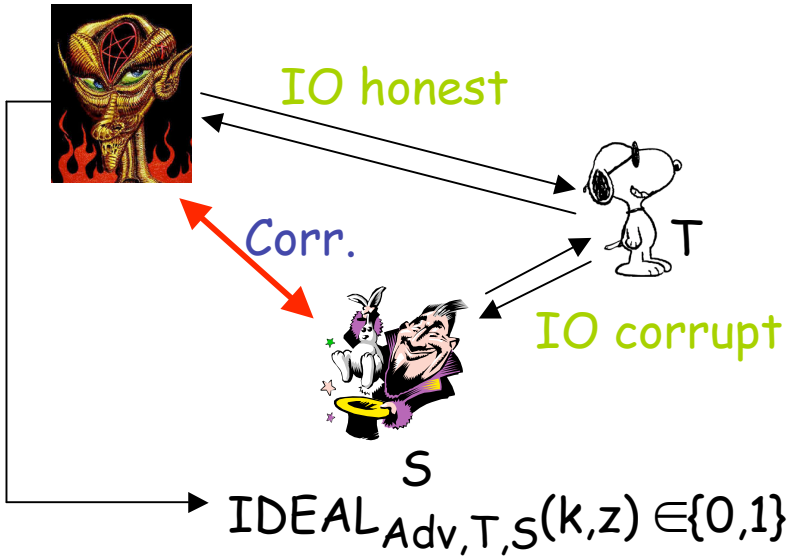
Exchange inputs/results for corrupt players

S must 'simulate' messages received by corrupt parties

At end of protocol, Adv outputs a bit, a random variable called $IDEAL_{Adv,T,S}(k,z)$

The Real Process vs. the Ideal Process





Security: there exists a simulator S such that no Adv can guess which process it is in!

The Definition

We say that π _-*securely realizes* T *perfectly* if there exists a simulator S such that for all adversaries Adv corrupting only sets in _ it holds that

$$\text{Prob}(\text{IDEAL}_{\text{Adv},T,S}(k,z) = 0) = \text{Prob}(\text{REAL}_{\text{Adv},\pi}(k,z) = 0)$$

For all k and z.

Intuition: we think of Adv's output bit as its guess at whether it is in the real or ideal world. The two probabilities equal means: it has no idea. Note we say for all Adv, so no bound here on Adv's computing power.

We talk about realizing T *statistically* if

$$|\text{Prob}(\text{IDEAL}_{\text{Adv},T,S}(k,z) = 0) - \text{Prob}(\text{REAL}_{\text{Adv},\pi}(k,z) = 0)|$$

is negligible in k (for arbitrary choice of z).

Realizing T *computationally*: the same, but but only for all polynomial time bounded Adv.

Intuition on Definition

It ensures that the real-life process inherits natural security properties from the ideal process. For instance..

The protocol forces Adv to be aware of which inputs corrupt players contribute: S must figure out which input values to send to T in ideal process. These inputs must follow from the protocol messages sent by corrupt players (and produced by Adv).

The protocol ensures that honest players compute correct results: is always true in ideal process by def. of T, and if protocol produces inconsistent results, Adv could distinguish easily.

The protocol does not release information it shouldn't: S is able to simulate convincingly Adv's view of attacking the protocol, based only on what T sends to corrupt players.

Often called: *Independence of inputs, Correctness, Privacy*

Coming Up

We are going to prove the following:

For a passive, adaptive, unbounded t-adversary: Any function f can be securely computed with perfect security if t< n/2

A first observation:

The function can always be written as an arithmetic circuit modulo some prime p

Well known that Boolean circuits with AND and NOT is general enough.

Represent inputs as 0/1 values mod p.

a AND b -> ab NOT a -> 1-a

Secret Sharing

A Dealer holds a secret value s in Zp*, p > n is a prime.

Dealer chooses a random polynomial f() over Zp* of degree at most t, such that f(0)=s:

$f(x) = s + a_1 x + a_2 x^2 + \ldots + a_t x^t$

Dealer sends $s_i = f(i)$ privately to Pi.

Properties:

- Any subset of at most t players has no information on s
- Any subset of at least t+1 players can easily compute s – can be done by taking a linear combination of the shares they know.

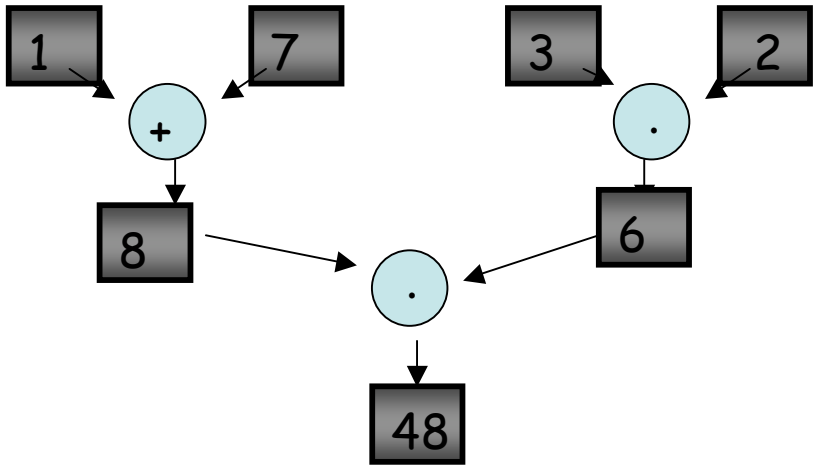
A consequence – the reconstruction vector:

There exists a *reconstruction vector* $(r_1,\ldots,r_n)$ such that for any polynomial h() of degree less than n:

$h(0) = r_1 h(1) + \ldots + r_n h(n)$

A Protocol for the Passive Corruption Case, I.T. scenario

- threshold adversary, may corrupt up to t players, t< n/2.



Circuit and inputs given

Create "objects" representing inputs, jointly held by players, value not accessible to adversary.

Computing phase: compute new objects.

Open outputs

Create Objects (Sharing Phase):

Each Pi shares each of his input value using a random polynomial of degree at most t, sends a share of each input to each player.

Notation: $a \xrightarrow{f()} a_1, a_2, \ldots, a_n$
means: value a has been shared using polynomial f(), resulting in shares $a_1,\ldots,a_n$, where player Pi knows a_i.

Computation Phase

Addition Gates

Input: $a \xrightarrow{f_a()} a_1,\ldots,a_n$ and $b \xrightarrow{f_b()} b_1,\ldots,b_n$

Desired Output: $c = a+b \xrightarrow{f_c()} c_1,\ldots,c_n$

Each player sets $c_i := a_i+b_i$.

Then we have what we want: $a+b \xrightarrow{f_c()} c_1,\ldots,c_n$, with $f_c() = f_a()+f_b()$ - works, since adding two polynomials of degree $\leq t$ produces a polynomial of degree $\leq t$. Clearly *private*, as no information is exchanged.

Multiplication Gates

Input: $a \xrightarrow{f_a()} a_1,\ldots,a_n$ and $b \xrightarrow{f_b()} b_1,\ldots,b_n$

Desired Output: $c = ab \xrightarrow{f_c()} c_1,\ldots,c_n$.

Each player sets $d_i := a_i\, b_i$.

If we set $h() = f_a()\, f_b()$, then $d_i = f_a(i)\, f_b(i) = h(i)$. Also $h(0) = ab = c$. Unfortunately, $h()$ may have degree up to $2t$. If the degree grows larger

Multiplication Gates, con't

We have public reconstruction vector $(r_1,\ldots,r_n)$ – know that

$c = h(0) = r_1 h(1) + \ldots + r_n h(n) = r_1 d_1 + \ldots + r_n d_n$ - since $\deg(h) \le 2t < n$

Each player Pi creates $d_i \xrightarrow{h_i()} c_{i1}, c_{i2},\ldots,c_{in}$. So we have:

Known by:	P1	P2		Pn
r_1	r_1	r_1		r_1
$d_1 \xrightarrow{h_1()}$	$+c_{11}$	$+c_{12}$	…	$+c_{1n}$
r_2	r_2	r_2		r_2
$d_2 \xrightarrow{h_2()}$	$+c_{21}$	$+c_{22}$	…	$+c_{2n}$
	…	…		…
r_n …	r_n	r_n		r_n
$= d_n \xrightarrow{h_n()} f_c()$	$= c_{n1}$	$= c_{n2}$	…	$= c_{nn}$
c	c_1	c_2	…	c_n

c is now shared using polynomial $f_c()$, where

$f_c() = \sum r_i h_i()$

Since $\deg(h_i) \le t$ we have that $\deg(f_c) \le t$

Output Opening Phase

Having made our way through the circuit, we have for each output value y:

$y \text{—} f_y() \rightarrow y_1, \ldots, y_n$

If y is to be received by player Pi, each Pj sends y_j to Pi.

Pi reconstructs in the normal way: $y = r_1 y_1 + \ldots + r_n y_n$.

Security, intuitively:

Correctness: Outputs trivially correct, since all players follow protocol

Privacy: For every input from an honest player, intermediate result and outputs of honest players, Adv sees at most t shares. These are always t random field elements, so reveal no information.

Independence of inputs: For every input from corrupt player Pi, the simulator sees the t+1 shares sent to honest players by Adv (on behalf of Pi), which makes S aware which input Pi 'used'

Coming Up

1. More details on secret sharing
2. More details on the formal proof of security in the UC framework

Secret Sharing

- A prime $p > n$ defining a field Z_p
- Secret sharing of $y \in Z_p$ among n parties with threshold t:
  - Pick $a1, \ldots, at \in Z_p$ uniformly at random
  - Let $fy(x) = y + a1\, x + a2\, x^2 + \ldots + at\, x^t$
  - For $i=1, \ldots, n$ let $yi = fy(i)$
  - Send yi to party Pi
  - We write $y \longrightarrow fy() \rightarrow y1, \ldots, yn$
- Privacy: Given any t of the elements y1,…, yn one learns nothing about y
- Reconstruction: Given any t+1 of the elements y1,…, yn one can efficiently compute y

A Fundamental Theorem of Algebra

- Let $f(x) = a0 + a1\ x + a2\ x^2 + \ldots + at\ x^t$ over a field
    - I.e. $\deg(f) \leq t$
- f(x) is either 0 in all points or has at most t distinct roots
- I other words:
    - If f(x) has more than t roots, then f(x) = 0

Lagrange Interpolation

- Used to prove both Privacy and Reconstruction
- Let I be a subset of Z_p of size t+1
- For $i \in I$ define $\delta_i(x) = \Pi_{j \in I \setminus \{i\}} (x-j) \left(\Pi_{j \in I \setminus \{i\}} (i-j) \right)^{-1}$
  - $\deg(\delta_i) \leq t$
  - $\delta_i(i) = 1$
  - $\delta_i(j) = 0$ for $j \in I \setminus \{i\}$
- Given a degree t polynomial a(x) let $fa(x) = \Sigma_{i \in I} a(i) \delta_i(x)$
  - For $i \in I$ it holds that $fa(i) = a(i)$
  - $\deg(fa-a) \leq t$ and fa-a has t+1 roots
  - So, $(fa-a)(x) = 0$ and thus $fa = a$
  - Therefore a(x) can be computed from a(i) for $i \in I$

Reconstruction

-Secret sharing of $y \in Z_p$ among n parties with threshold t:

- Pick $a1, \ldots, at \in Z_p$ uniformly at random
- Let $a(x) = y + a1\, x + a2\, x^2 + \ldots + at\, x^t$
- For $i=1, \ldots, n$ let $yi = a(i)$ and send yi to party Pi

- Given $yi=a(i)$ for $i \in I$ where I is of size t+1
  - Compute $\delta_i(x) = \Pi_{j \in I \setminus \{i\}} (j-x) \; (\Pi_{j \in I \setminus \{i\}} (j-i))^{-1}$
  - Compute $ri = \delta_i(0)$ for $i \in I$
  - Compute $\Sigma_{i \in I}\, yi\, ri = \Sigma_{i \in I}\, a(i)\, \delta_i(0) = fa(0) = a(0) = y$

- ri depends only on I and i

- So, Y is a known linear combination of the t+1 known yi

Privacy

-Secret sharing of $y \in Z_p$ among n parties with threshold t:

- Pick $a1, \ldots, at \in Z_p$ uniformly at random
- Let $a(x) = y + a1\, x + a2\, x^2 + \ldots + at\, x^t$
- For $i=1, \ldots, n$ let $yi = a(i)$ and send yi to party Pi

- Given $yi=a(i)$ for $i \in I$ where I is of size t
- The shared secret is $y = a(0)$
- For any $y0 \in Z_p$ could it be that $a(0) = y0$?
  - Let $I' = I \cup \{0\}$ so that I' has size t+1
  - $\exists$ one deg(t) polynomial $f_{y0}(x)$ s.t. $f_{y0}(i) = yi$ for $i \in I'$
  - If $a(x) = f_{y0}(x)$ then $a(0) = y0$
  - So, all secrets $y=a(0)$ are equally likely!

Privacy, The 'Patching' View

-Secret sharing of $y \in Z_p$ among n parties with threshold t:

- Pick $a(x) = y + a1\ x + a2\ x^2 + \ldots + at\ x^t$ at random

- For i=1, …, n let yi = a(i) and send yi to party Pi

- Assume that an adversary controls t parties Pi for $i \in I$

- Let Pj be an honest party (i.e. $j \notin I$) and assume that Pj shared some value

- As a result the adversary sees yi for $i \in I$

-By "Patching" the sharing to be a sharing of y we mean that we compute the unique deg(t) polynomial f(x) s.t. f(0) = y and f(i) = yi for $i \in I$

- Having used f(x) to share would not change the view of the adversary !

Proving the Protocol Secure

-The goal of the protocol was to compute a function

$(y1, \ldots, yn) = F(x1, \ldots, xn)$

specified as an arithmetic circuit over Z_p

-The ideal functionality $F_{MPC}(F)$ for the task looks as follows:

- Interpret the input from Pi in the first round as an element $xi \in Z_p$
- (Wait *ComputeDelay* rounds)
- Compute $(y1, \ldots, yn) = F(x1, \ldots, xn)$
- Deliver yi to Pi

- We say that a protocol Π t-securely computes F (using *ComputeDelay* rounds) if it realizes $F_{MPC}(F)$ against adversaries corrupting at most t parties

Proving the Protocol Secure(2)

-The ideal functionality $F_{MPC}(F)$ for the task looks as follows:

- Interpret the input from Pi in the first round as an element $xi \in Z_p$
- Compute (y1, …, yn) = F(x1, …, xn)
- Deliver yi to Pi

- In the simulation:

- The simulator specifies the inputs xi for corrupted Pi
- The simulator does not see xi for honest Pi !!
- Then (y1, …, yn) = F(x1, …, xn) is computed by $F_{MPC}(F)$
- The simulator gets the outputs yi for corrupted Pi
- The simulator must simulate the view of all corrupted parties in the protocol given only the above powers

Proving the Protocol Secure(3)

- In the protocol $t < n/2$ such that $n-t \geq t+1$ parties are honest
- The Protocol works as follows
  1. Party Pi shares its input xi with threshold t
  2. The parties do some secret sharings of shares and adds some shares locally
  3. Parties reconstruct the outputs yi to Pi

Proving the Protocol Secure(4)

1. Party Pi shares its input xi
2. Secret sharings of shares and adding shares locally
3. Parties reconstruct the outputs yi to Pi

- The simulator works as follows:

1. - For honest Pi do a sharing of 0 instead of xi
   - For corrupt Pi use the n-t shares yj sent to honest parties to compute the input xi of corrupt Pi
   - Input xi for each corrupt Pi to $F_{MPC}(F)$
2. Run as in the protocol
3. - Get yi for each corrupt Pi from $F_{MPC}(F)$
   - "Patch" the sharing in the simulation to be a sharing of yi
   - Reconstruct yi to Pi as in the protocol
4. If Pi is corrupted during the simulation, get xi from $F_{MPC}(F)$

Protocol for Active Adversaries

General idea: use protocol for passive case, but make players *prove* that they send correct information.

Main tool for this: *commitment scheme*

Intuition: committer Pi puts secret value s∈K "in a locked box" and puts it on the table. Later, Pi can choose to open the box, by releasing the key.

- Hiding – no one else can learn s from the commitment
- Binding – having given away the box, Pi cannot change what is inside.

We can model a commitment scheme by an ideal functionality Fcom, that offers the following commands:

- (Commit, s, sid) – if player Pi sends this, and if honest players acknowledge by sending (Commit, sid), Fcom records the fact that Pi has now committed to a value and stores s in a variable called sid.
- (Open, sid) – if Pi sends this and all honest players acknowledge by sending (Open, sid), Fcom recovers s and sends s to all players.

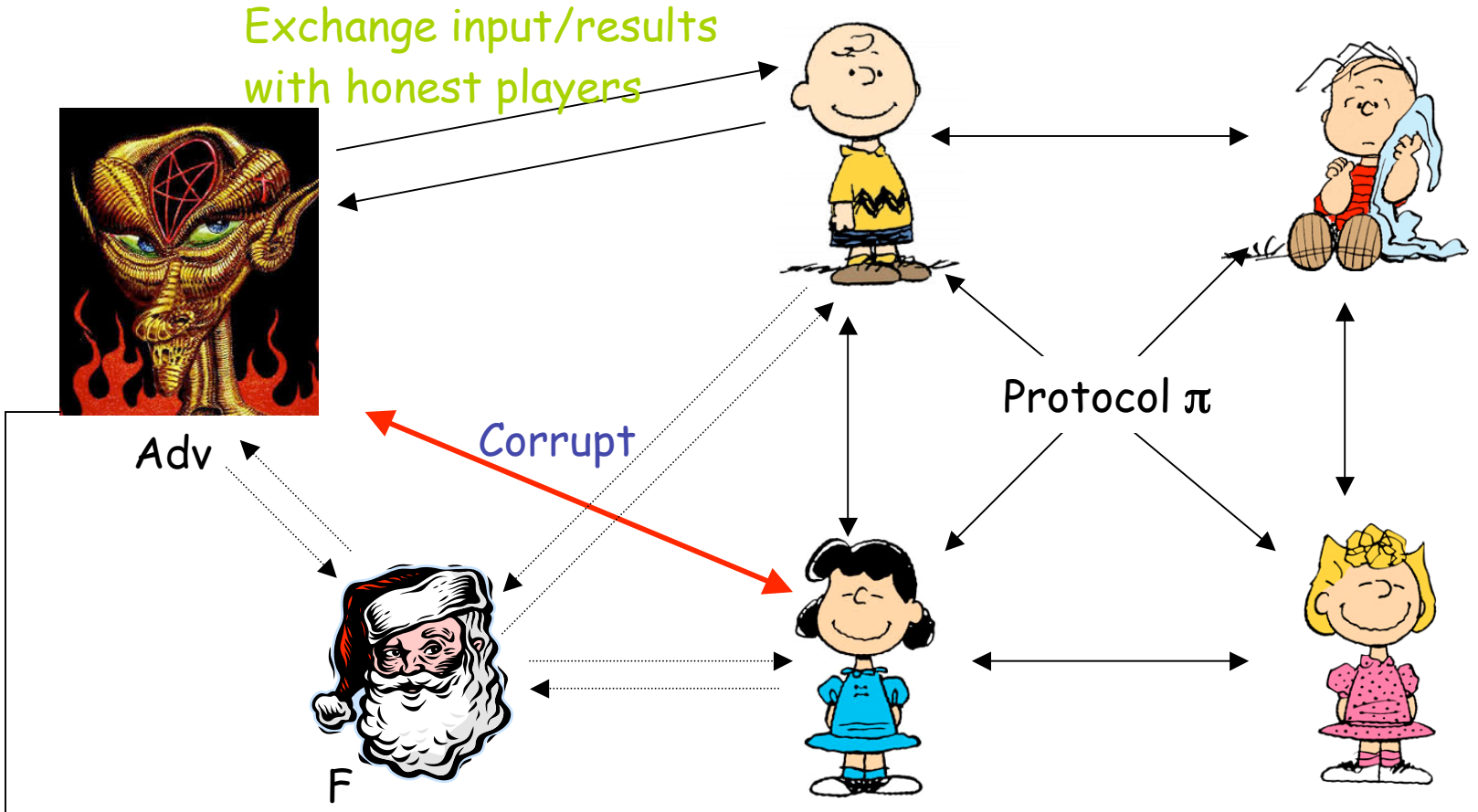
Acknowledgements are part of specification since implementation is a

Using Functionalities in Protocols

The plan is to use a commitment scheme, i.e., (an extention of) Fcom as a "subrutine" to build a realization of F_{MPC} , secure against active cheating. So we need:

- A model specifying what it means to use an ideal functionality in a protocol. As a result, we can formally specify what it means that protocol π "implements F_{MPC} when given access to Fcom".
- A theorem saying that if protocol θ realizes for instance Fcom securely, then it is OK to replace Fcom by θ.
- Doing this in π would result in the desired real-life protocol for F_{MPC}

The F-Hybrid model: "Realizing T given F"



When protocol over, Adv outputs a bit. Given Adv, π ,k and z, this is a random variable called $\mathrm{HYBRID}_{F,\,Adv,\,\pi}(k,z)$

The Ideal Process

Same as before, except: Adv now expects to act in a world where F is present, so S must also emulate Adv's and corrupt players' interaction with F

Adv

Exchange inputs/results for honest players

Corrupt

T

S

Exchange inputs/results for corrupt players

At end of protocol, Adv outputs a bit, a random variable called $IDEAL_{Adv,T,S}(k,z)$

Definition

We say that π $\_$-*securely realizes* T *perfectly* in the F-hybrid model if there exists a simulator S such that for all adversaries Adv corrupting only sets in $\_$ it holds that

$$\text{Prob}(\text{IDEAL}_{Adv,T,S}(k,z)=0) = \text{Prob}(\text{HYBRID}_{F,Adv,\pi}(k,z)=0)$$

For all k and z.

If π is a protocol in the F-Hybrid model, and θ is a protocol realizing F, we let π^{θ} be the protocol obtained by replacing all calls to F by calls to θ.

Composition Theorem

If π $\_$-securely realizes T in the F-hybrid model and θ securely realizes F (in the real model), then π^{θ} $\_$-securely realizes T in the real model.

Intuition: to implement T, enough to first show how to do it, assuming we are (magically) given access to F. Then show how to implement F, then a (real) implementation for T automatically follows.

Definition of Fcom functionality.

Notation for commitments: $[s]_i$

-means: Pi has successfully committed to s, and Fcom has stored s in its internal memory.

Commit command

Goal: create $[s]_i$

• Executed if all honest players send a "commit i" command and Pi sends a value s. If Pi is corrupt, he may send "refuse" instead.

• If Pi refused, send "fail" to all players, otherwise store s in a new variable and send "success" to everyone.

Open command

Goal: open $[s]_i$

• Executed if all honest players send a "open" command referring to $[s]_i$. If Pi is corrupt, he may send "refuse" instead.

• If Pi refused, send "fail" to all players, otherwise send s to everyone.

We need Fcom to offer more functionality, it needs to implement Homomorphic Commitments, i.e. the following two commands

CommitAdd command

Goal: from $[a]_i$ and $[b]_i$ create new commitment $[a]_i + [b]_i = [a+b]_i$

- Executed if all honest players send a "CommitAdd" command referring to $[a]_i$ and $[b]_i$.

- Fcom will compute a+b and store it in a new variable, as if committed to by Pi (in particular, Pi can open this new commitment, as if he had committed to a+b in the normal way.)

ConstantMult command

Goal: from $[a]_i$ and public u, create new commitment $u \bullet [s]_i = [ua]_i$

- Executed if all honest players send a "ConstantMult u" command referring to $[a]_i$.

- Fcom will compute ua and store it in a new variable, as if committed to by Pi

Advanced commands

From the basic Commit, Open, CommitAdd and ConstantMult commands, anything else we need can be built, but for simplicity, we define some extra commands..

CTP command (CTP: Commitment Transfer Protocol)

Goal: from $[s]_i$, produce $[s]_j$ for $i \neq j$

- Executed if all honest players send a CTP command referring to $[s]_i$, i and j. If Pi is corrupt, he may send "refuse" instead.
- If Pi refused, send "fail" to all players, otherwise store s in a new variable as if committed by Pj, send "success" to everyone and send s to Pj.

CMP command (CMP: Commitment Multiplication Protocol)

Goal: Given $[a]_i$ $[b]_i$ $[c]_i$ Pi can convince all players that c=ab (if true)

• executed if all honest players send a CMP command referring to $[a]_i$ $[b]_i$ $[c]_i$. If Pi is corrupt, he may send "refuse".

• if Pi refused or if c≠ab, send "fail" to all players. Otherwise, send "success" to everyone.

CSP command (CSP: Commitment Sharing Protocol)

Goal: Given $[a]_i$, create $[a_1]_1, [a_2]_2, \ldots, [a_n]_n$ where $f(i)=a_i$ and f is a polynomial of degree at most t.

• executed if all honest players send a CSP command referring to $[a]_i$. Pi should send a polynomial f() of degree at most t. If Pi is corrupt, he may send "refuse".

• if Pi refused, send "fail" to all players. Otherwise, for i=1..n, compute a_i = f(i) store it in a variable as if committed by Pi and send "success" to everyone.

Implementation of CSP from basic Fcom commands.

Pi chooses random polynomial

$f_a(x) = a + c_1 x + c_2 x^2 + \ldots + c_t x^t$ and make commitments:

$[c_1]_i, [c_2]_i, \ldots, [c_t]_i$.

We define $a_j = f_a(j)$.

By calling the CommitAdd and ConstantMult commands, we can create commitments:

$[a_j]_i = [a]_i + j \bullet [c_1]_i + j^2 \bullet [c_2]_i + \ldots + j^t \bullet [c_t]_i$.

Finally, we use CTP to create $[a_j]_j$ from $[a_j]_i$.

During creation and manipulation of the commitments, Pi can refuse if he is corrupt (and he's the only one who can do so). This counts as Pi refusing the entire CSP operation.

We return to implementation of other commands later.

Protocol for Active Adversary

- Adv is adaptive, unbounded and corrupts up to t players, t< n/3.

- We assume Fcom is available, with the Commit, Open, CommitAdd, ConstantMult, CTP, CMP and CSP commands.

Same phases as in passively secure protocol, but now we want to maintain that all players are committed to their shares of all values.

For simplicity, assume first that no one behaves such that Fcom will return fail.

Input Sharing Phase

Pi commits to his input value a: creates $[a]_i$, then we call the CSP command.

So we have…

Result of Input Sharing Phase

- Each input value a has been shared by some player Pi using a polynomial $f_a()$, where $f_a()$ is of degree $\leq t$.
- If Pi is honest, $f_a()$ is random of degree $\leq t$.
- Each player Pi is committed to his share of a.

Notation:

$a \xrightarrow{f_a()} [a_1]_1, [a_2]_2, \ldots, [a_n]_n$

Computation Phase

Addition Gates

Input: $a \xrightarrow{f_a()} [a_1]_1, [a_2]_2, \ldots, [a_n]_n$ and $b \xrightarrow{f_b()} [b_1]_1, [b_2]_2, \ldots, [b_n]_n$

Desired Output: $c = a+b \xrightarrow{f_c()} [c_1]_1, [c_2]_2, \ldots, [c_n]_n$

Each player Pi sets $c_i := a_i + b_i$ and all players compute $[c_i]_i = [a_i]_i + [b_i]_i$.
Produces desired result with $f_c() = f_a() + f_b()$.

Multiplication Gates

Input: $a \xrightarrow{f_a()} [a_1]_1, [a_2]_2, \ldots, [a_n]_n$ and $b \xrightarrow{f_b()} [b_1]_1, [b_2]_2, \ldots, [b_n]_n$

Desired Output: $c = ab \xrightarrow{f_c()} [c_1]_1, [c_2]_2, \ldots, [c_n]_n$

Each player Pi sets $d_i := a_i\, b_i$, makes commitment $[d_i]_i$ and uses CMP on commitments $[a_i]_i$, $[b_i]_i$, $[d_i]_i$ to show that d_i is correct

If we set $h() = f_a()\, f_b()$, then $d_i = f_a(i)\, f_b(i) = h(i)$. Also $h(0) = ab = c$

So we can use essentially same method as in passive case to get to a sharing of c using a random polynomial of degree $\leq t$.

Multiplication Gates, con't

Public reconstruction vector is still $(r_1,\ldots,r_n)$

Using same method as in input sharing phase,

each player Pi creates $d_i \xrightarrow{h_i()} [c_{i1}]_1, [c_{i2}]_2,\ldots,[c_{in}]_n$. So we have:

Committed by: P1 P2 Pn

$r_1 \bullet$ $r_1 \bullet$ $r_1 \bullet$

$d_1 \xrightarrow{h_1()} + [c_{11}]_1 + [c_{12}]_2 \ldots + [c_{1n}]_n$

$r_2 \bullet$ $r_2 \bullet$ $r_2 \bullet$

$d_2 \xrightarrow{h_2()} + [c_{21}]_1 + [c_{22}]_2 \ldots + [c_{2n}]_n$

... ...

... $r_n \bullet$ $r_n \bullet$ $r_n \bullet$

$d_n \xrightarrow{h_n()} = [c_{n1}]_1 = [c_{n2}]_2 \ldots = [c_{nn}]_n$

$c \xrightarrow{f_c()} [c_1]_1 \quad [c_2]_2 \quad \ldots \quad [c_n]_n$

Computed using calls to Fcom

c is now shared using polynomial $f_c()$, where

$f_c() = \sum r_i h_i()$

Output Opening Phase

Having made our way through the circuit, we have for each output value y:

$y \xrightarrow{f_y()} [y_1]_1, \ldots, [y_n]_n$

If y is to be received by player Pi, each commitment is opened such that all information is sent privately to Pi.

Pi recontructs in the normal way.

How to handle Failures

If a player Pi sends refuse in some command, causing Fcom to return "fail":

- In input sharing phase: ignore or use default value of input

- In computation phase: can always go back to start, open Pi's inputs and recompute, simulating Pi openly. Also more efficient solution: since $t < n/3$ at least $n-t > 2t$ players do multiplication step correctly. So can still do multiplication step using reconstruction vector tailored to the set that behaves well.

- In output opening phase: the receiver of an output just ignores incorrectly opened commitments – there is enough info to reconstruct, since $n-t > t$.

Implementing Fcom commands

- CommitAdd, ConstantMult
- CPT protocol
- CMP protocol

Idea for commitments: implement using secret sharing. To commit to s, a dealer D just creates

s—f()→ s1,…,sn

To open, D broadcasts f(), each player Pi says if his share was f(i). Opening accepted, if at least n-t players agree.

The good news: CommitAdd can be implemented by just locally adding shares, ConstantMult by multiplyinjg all shares by constant. Furthermore, if D remains honest, Adv learns no information at commitment time.

The bad news: who says D distributes correctly computed shares? If not, s not uniquely determined, D may open different values later.

Some Wishful Thinking..

Suppose for a moment we could magically force D to distribute shares consistent with a polynomial f() of degree t < n/3.

Then it works!

- Easy to see that things are fine if D remains honest

- If D corrupt, want to show that D cannot convincingly open any $s' \neq s$.
Assume he could. When opening s', D broadcasts polynomial f'().
At least n-t> 2t players agree ⇒ at least t+1 honest players agree ⇒
f'() agrees with f() in t+1 points ⇒ f'()=f() ⇒ s=s', contradiction.

Therefore sufficient to force D to be consistent

How to force consistency

Main tool:

$f(X,Y) = \sum f_{ij} X^i y^j$

- a bivariate polynomial of degree at most t in both variables. Will assume f() is symmetric, i.e. $f_{ij} = f_{ji}$

Define, for $0 < i,j \leq n$:

$f_0(X) = f(X,0)$, and set $s = f_0(0)$, $si = f_0(i)$

$fi(X) = f(X,i)$, $fi(j) = sij$

How to think of this:
s is the "real" secret to be shared, using polynomial $f_0()$. Hence $f_0(i) = si$ will be player Pi's share in s. The rest of the machinery is just for checking.

Observations, by symmetry:

$si = f_0(i) = f(i,0) = f(0,i) = fi(0)$ *(fi(X) shares si)*

$sij = fi(j) = f(j,i) = fj(i) = sji$ *(i'th share of sj = j'th share of si)*

Commit Protocol

1. Dealer D chooses random bivariate polynomial f() as above, such that f(0,0)= s, the value he wants to commit to. Sends privately fi() to player Pi how sets si = fi(0) and sets si1 = fi(1), …, sin = fi(n)
2. Pi sends sij to Pj, who compares to sji – broadcast "complaint" if conflict.
3. D must broadcast correct value of all sij's complained about
4. If some Pi finds disagreement between broadcasted values and what he received privately from D, he broadcasts "accuse D" (in which case we know that either Pi or D is corrupted)
5. In response to accusation from Pi, D must broadcast the polynomial fi(X) he sent to Pi. This may cause further players to find disagreement as in Step 4, they then accuse D. Now:
   - If D has been accused by more than t players, then D is disqualified.
   - Otherwise commitment is accepted. Players accusing D in Step 4 use the fi() broadcast as their polynomial. All others stick to the polynomial received in Step 1. Each player Pi stores fi(0) as his part of the commitment.

Commitments, more concretely

In our implementation, a commitment

$[a]_i$ is a set of shares: a1 a2 ... an

held by P1 P2 Pn

where Pi knows the polynomial fa() that was used to create the shares – and where fa(0) =a.

- Checking using bivariate polynomial forces Pi to create shares correctly
- Opening means Pi broadcasts fa(), each Pj checks if fa(j) = aj, complains if not, opening accepted iff at most t complaints.
- $[a]_i + [b]_i$ means: each Pj has aj and bj, now computes cj:= aj + bj. Pi computes fc() = fa() + fb(). We now have new commitment $[a+b]_i$, defined by shares c1,…,cn, and polynomial fc().
- u• $[a]_i$ means: each Pj has aj, now computes dj:= u aj. Pi computes fd():= u fa(). We now have new commitment $[ua]_i$, defined by shares d1,…,dn and polynomial fd().

Implementing CTP (Commitment Transfer) Command:

Purpose: from $[a]_i$, produce $[a]_j$

• Given $[a]_i$, Pi sends privately to Pj the polynomial fa() defining the commitment.

• Each player Pk sends his share of the commitment privately to Pj.

• Pj checks that what he gets from Pk is fa(k). If more than t disagreements, complain, and Pi must open in public. Note: if there's a complaint, either Pi or Pj is bad, so Adv knows a already ⇒ no harm in making it public.

Reminder:

Input Sharing Phase

Pi commits to input value a : $[a]_i$, then does *Commitment Share Protocol* (CSP):

Choose random polynomial

$fa(x) = a + c1\ x + c2\ x^2 + \ldots + ct\ x^t$ and make commitments:

$[c1]_i,\ [c2]_i,\ldots,\ [ct]_i$.

The j'th share of a is aj= fa(j).

Players can now immediately compute commitments to the shares:

$[aj]_i = [a]_i + j \bullet [c1]_i + j^2 \bullet [c2]_i + \ldots + j^t \bullet [ct]_i$.

Finally, we use CTP to create $[aj]_j$ from $[aj]_i$.

Implementing CMP (Commitment Multiplication) Command

Given $[a]_i$, $[b]_i$, $[c]_i$, Pi wants to convince us that c= ab.

Pi uses CSP command to create:

a —fa()→ $[a1]_1$, $[a2]_2$,…, $[an]_n$

b —fb()→ $[b1]_1$, $[b2]_2$,…, $[bn]_n$

c —fc()→ $[c1]_1$, $[c2]_2$,…, $[cn]_n$

Where fc() = fa() fb()

Even if Pi corrupt, this guarantees that all committed shares are consistent with polynomials of degree at most t, t, and 2t, and that fa(0)=a, fb(0)=b, fc(0)=c.

Hence sufficient to verify that indeed fc() = fa() fb():

Each Pi checks that ci= ai bi. If not he complains and proves his case by opening the commitments. Honest players will do this correctly, so we know that fc() agrees with fa() fb() in at least n-t > 2t points ⇒ fc() = fa() fb().

Coming Up

- A few words on how to prove active security
- Details on the security of the commitment scheme

Proving security of active case

First show that high-level protocol where Fcom is assumed, securely realizes F_{MPC}.

Almost same proof as for passive case: same private data held by players as in passive case, honest players handle them using same algorithms. Perfect simulation results

Then show that Fcom implementation is secure

Basic Ideas:

When corrupt player commits, simulator can reconstruct value committed to from the messages sent, because consistency is enforced. So the simulator knows what to send to Fcom.

When honest player commits to value s, s is not known to simulator. So we show the adversary a secret sharing of 0 in place of the shares in s that the honest player would send.

At opening time, simulator gets s from Fcom, then 'patches' the set of shares of honest parties to be consistent with s, and claim these new

The Commitment Scheme

- In the protocol we needed a commitment scheme secure against t corruptions for $t < n/3$
- We saw the protocol but still have to argue that it is:
  - **Correct**: If the dealer D is honest then its commitment is always accepted
  - **Binding**: Even a cheating D cannot open an accepted commitment to more than one value
  - **Hiding**: The t corrupted parties' view of the commitment protocol is independent of the value committed to

Commitment

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \leq t$
2. D: Let $f_i(X) = f(X,i)$ and send $f_i(X)$ to P_i
3. P_i: Compute $s_i=f_i(0)$ and send $f_i(j)$ to P_j
4. P_j: If $f_j(i) \neq f_i(j)$ then broadcast "P_i and P_j has conflict"
   a) D: Broadcast $s_{ij} = f(i,j)$
   b) P_j: If $f_j(i) \neq s_{ij}$ then broadcast "D and P_j has conflict"
   c) D: Broadcast $f_j(X)$ with $f_j(i) = s_{ij}$
   d) P_j: Use the broadcast $f_j(X)$ instead of that from Step 2
5. P_i: If $f_i(j) \neq f_j(i)$ for any $f_j(X)$ broadcast above, then broadcast "D and P_i has conflict"
6. P: If number of parties in conflict with D is $\leq t$ then accept

CLAIM: If accepted, then there exists a degree t polynomial $g(X)$ such that $s_i=g(i)$ for all honest parties (and $g=f_0$ if D is honest)

Commitment

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \leq t$
2. D: Let $f_i(X) = f(X,i)$ and send $f_i(X)$ to P_i
3. P_i: Compute $s_i=f_i(0)$ and send $f_i(j)$ to P_j
4. - 6. Make sure that CLAIM is true

CLAIM: If accepted, then there exists a degree t polynomial ***g*** such that $s_i=g(i)$ for all honest parties (and $g=f_0$ if D is honest)

Opening (to some receiver R)

1. D: Send $g(X) = f_0(X)$ to R
2. P_i: Send s_i to R
3. If $\deg(g) \leq t$ and $g(i) = s_i$ for i=1, …, n (except for t points), then accept as opened to $s=g(0)$

If all are to see the opening, then broadcast all values instead

Correct & Binding

CLAIM: There exists a degree t polynomial ***g*** such that $s_i = g(i)$ for all honest parties (and $g = f_0$ if D is honest)

1. D: Send $g(X) = f_0(X)$ to R
2. P_i: Send s_i to R
3. If $\deg(g) \leq t$ and $g(i) = s_i$ for $i=1, \ldots, n$ (except for t points), then accept as opened to $s=g(0)$

- **Correct**: Follows from last part of CLAIM as $g(0)=f_0(0)=f(0,0)=s$
- **Binding**: Follows from CLAIM as follows:
  - Assume that R receives an accepting opening
  - By definition, $g(i) = s_i$ for at least n-t parties P_i
  - So, $g(i) = s_i$ for at least $n-t-t=n-2t>t$ honest parties P_i
  - So, $g(i) = g(i)$ for more than t points i
  - So, $g(X) = g(X)$ by Lagrange interpolation as $\deg(g), \deg(g) < t+1$

Commitment, Hiding

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \leq t$
2. D: Let $f_i(X) = f(X,i)$ and send $f_i(X)$ to P_i
3. P_i: Compute $s_i=f_i(0)$ and send $f_i(j)$ to P_j
4. P_j: If $f_j(i) \neq f_i(j)$ then broadcast "P_i and P_j has conflict"
   a) D: Broadcast $s_{ij} = f(i,j)$
   b) P_j: If $f_j(i) \neq s_{ij}$ then broadcast "D and P_j has conflict"
   c) D: Broadcast $f_j(X)$ with $f_j(i) = s_{ij}$
   d) P_j: Use the broadcast $f_j(X)$ instead of that from Step 3
5. P_i: If $f_i(j) \neq f_j(i)$ for any $f_j(X)$ broadcast above, then broadcast "D and P_i has conflict"
6. P_i: If number of parties in conflict with D is $> t$ then accept

Green: Already known by adversary or receiver, so enough to argue that Step 2 leaks no information

Commitment, Hiding(2)

- Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $deg(f) \leq t$. Let $f_i(X) = f(X,i)$ and send $f_i(X)$ to P_i
- Let A be the indices of the t corrupted parties
- Enough to argue: $\forall s' \exists$same number of $g(X,Y)$ s.t.
  $g(X,Y)=g(Y,X)$, $deg(g)\leq t$, $g(0,0) = s'$ and $g(X,i) = f_i(X)$ for $i\in A$
- Define $h(X) = \Pi_{i\in A}(1 - i^{-1}X)$ s.t. $deg(h)\leq t$, $h(i\in A)=0$ and $h(0)=1$
- Let $Z(X,Y) = h(X)h(Y)$ s.t.
  $Z(X,Y)=Z(Y,X)$, $deg(Z)\leq t$, $Z(X,i\in A)=0$, $Z(0,0)=1$
- For any s' let $g(X,Y) = f(X,Y) + (s'-s)Z(X,Y)$ s.t.
  - $g(X,Y)=g(Y,Z)$, $deg(g(X,Y))\leq t$
  - For $i\in A$: $g(X,i) = f(X,i) + (s'-s)Z(X,i) = f_i(X)$
  - $g(0,0) = f(0,0) + (s'-s) = s + (s' - s) = s'$

Proving CLAIM

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \le t$
2. D: $f_i(X) = f(X,i)$, send $f_i(X)$ to P_i
3. P_i: $s_i=f_i(0)$, send $f_i(j)$ to P_j

CLAIM: If accepted, then there exists a degree $\le t$ polynomial $g(X)$ such that $s_i=g(i)$ for all honest parties (and $g=f_0$ if D is honest)

4. P_j: If $f_j(i) \neq f_i(j)$, then broadcast "P_i and P_j has conflict"
   a) D: Broadcast $s_{ij} = f(i,j)$
   b) P_j: If $f_j(i) \neq s_{ij}$ then broadcast "D and P_j has conflict"
   c) D: Broadcast $f_j(X)$ with $f_j(i) = s_{ij}$
   d) P_j: Use the broadcast $f_j(X)$ instead of that from Step 2
5. P_i: If $f_i(j) \neq f_j(i)$ for any $f_j(X)$ broadcast above, then broadcast "D and P_i has conflict"

Proving CLAIM

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \leq t$
2. D: $f_i(X) = f(X,i)$, send $f_i(X)$ to P_i
3. P_i: $s_i=f_i(0)$, send $f_i(j)$ to P_j

- Let B denote the honest parties
- Let C denote the honest parties which did not conflict with D
- See that $|C| > t$ when accept

4. P_j: If $f_j(i) \neq f_i(j)$, then broadcast "P_i and P_j has conflict"
   a) D: Broadcast $s_{ij} = f(i,j)$
   b) P_j: If $f_j(i) \neq s_{ij}$ then broadcast "D and P_j has conflict"
   c) D: Broadcast $f_j(X)$ with $f_j(i) = s_{ij}$
   d) P_j: Use the broadcast $f_j(X)$ instead of that from Step 2
5. P_i: If $f_i(j) \neq f_j(i)$ for any $f_j(X)$ broadcast above, then broadcast "D and P_i has conflict"

B: honest; C: honest, no conflict; $|C|>t$

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \le t$
2. D: $f_i(X) = f(X,i)$, send $f_i(X)$ to P_i
3. P_i: $s_i=f_i(0)$, send $f_i(j)$ to P_j

- See that $i \in C$ and $j \in B$ implies that $f_i(j)=f_j(i)$
- Let $g(X) = \Sigma_{i \in C} r_i f_i(X)$ where r_i is such that $h(0) = \Sigma_{i \in C} r_i h(i)$ when $\deg(h) \le t$

4. P_j: If $f_j(i) \neq f_i(j)$, then broadcast "P_i and P_j has conflict"
   a) D: Broadcast $s_{ij} = f(i,j)$
   b) P_j: If $f_j(i) \neq s_{ij}$ then broadcast "D and P_j has conflict"
   c) D: Broadcast $f_j(X)$ with $f_j(i) = s_{ij}$
   d) P_j: Use the broadcast $f_j(X)$ instead of that from Step 2
5. P_i: If $f_i(j) \neq f_j(i)$ for any $f_j(X)$ broadcast above, then broadcast "D and P_i has conflict"

B: honest; C honest, no conflict; $h(0) = \Sigma_{i \in C} r_i h(i)$ when $\deg(h) \le t$

$i \in C$ and $j \in B$ implies that $f_i(j) = f_j(i)$; $g(X) = \Sigma_{i \in C} r_i f_i(X)$

1. Dealer D: Pick random $f(X,Y) = \sum f_{ij} X^i Y^j$ with $f(X,Y)=f(Y,X)$ and $f(0,0) = s$ and $\deg(f(X,Y)) \le t$
2. D: $f_i(X) = f(X,i)$, send $f_i(X)$ to P_i
3. P_i: $s_i = f_i(0)$, send $f_i(j)$ to P_j

- For $j \in B$: $g(j) = \Sigma_{i \in C} r_i f_i(j)$
- $= \Sigma_{i \in C} r_i f_j(i) = f_j(0) = s_j$
- Also: $\deg(g) \le t$
- QED

4. P_j: If $f_j(i) \neq f_i(j)$, then broadcast "P_i and P_j has conflict"
   a) D: Broadcast $s_{ij} = f(i,j)$
   b) P_j: If $f_j(i) \neq s_{ij}$ then broadcast "D and P_j has conflict"
   c) D: Broadcast $f_j(X)$ with $f_j(i) = s_{ij}$
   d) P_j: Use the broadcast $f_j(X)$ instead of that from Step 2
5. P_i: If $f_i(j) \neq f_j(i)$ for any $f_j(X)$ broadcast above, then broadcast "D and P_i has conflict"

Recap

- We have seen that any function can be computed perfectly securely with threshold $t<n/3$ against an active adversary in the information theoretic model
- We have seen that any function can be computed perfectly securely with threshold $t<n/2$ against a passive adversary in the information theoretic model

Coming Up

- We will look at why the bounds $t<n/3$ respectively $t<n/2$ are optimal
- We see that any function can be computed *statistically* securely with threshold $t<n/2$ against an active adversary in the information theoretic model
- We will look at security in the cryptographic model, where no secure channels are assumed
- We take a brief look at security against general (non-threshold)

Passive Security vs. n/2

- We have seen that any function can be computed perfectly securely with threshold t<n/2 against a passive adversary in the information theoretic model
- This is optimal in the sense that there exist functions which can***not*** be computed securely with threshold t<n/2 against a passive adversary in the information theoretic model
- E.g. the AND function: *A*'s and B' inputs are bits a, b the output is a bit c, where c=1 if and only if a=1 and b=1 (here n=2 and t=1)
- If a=0, then c=0 and *A* is to learn nothing.
- But, using infinite computing power, *A* can determine if there exists randomness r' such that if *A* ran with input a=1 using r', then the same conversation she just had with B would result (and B can do the same thing)
  - If *yes*: Then *A* learns that b=0, as a=1 and c=0

Active Security vs. n/3

- We have seen that any function can be computed perfectly securely with threshold t<n/3 against an active adversary in the information theoretic model
- This is optimal in the sense that there exist functions which can***not*** be computed securely with threshold t<n/3 against a passive adversary in the information theoretic model
- E.g. the function for n=3 (and thus t=1) where *A* has input *a* and B and *C* have outputs *b=a* and *c=a* (this is the BROADCAST function)
- Informally the proof goes as follows:
  - *A* might e.g. send a to B and *C*
  - But if *A* is cheating he might send different values to B and *C*
  - So, B and *C* must make sure to output the same value
  - So, B might e.g. ask *C* for its value

Active Security vs. n/2

- So, active security is impossible for $t \geq n/3$, because already broadcast is impossible for $t \geq n/3$
- So, what if we *assume* that the parties has a broadcast channel at their disposal. Then the previous result does not apply anymore. Is active secure MPC then still impossible for $t \geq n/3$?
- No, any function can be computed *statistically* securely with threshold $t<n/2$ against an active adversary in the information theoretic model with broadcast
- The protocol that we saw used that $t \geq n/3$ in two places:
  - To ensure that broadcast could be done
  - To make sure that commitments can only be opened to one value
- The binding requirement can also be guaranteed for $t<n/2$ if the dealer is required to *sign* the shares s_i of the secret s
- The signature functionality can be implemented statistically secure

Active Security vs. n/2

- To see that signatures help, consider the following simplified protocol (more detail in the note):
- COMMIT(s):
  - The dealer shares s with threshold $t<n/2$ with $f(X)$
  - Thus a share s_i is sent to P_i
  - The dealer also sends a signature S_i on s_i
  - If an honest party does not get a correct signature, it broadcasts a complaint and the dealer makes s_i and S_i public
- OPEN:
  - The dealer broadcasts $f(X)$
  - P_i broadcasts s_i and S_i
  - The opening, to $s=f(0)$, is accepted if $f(i)=s_i$ for all i where S_i is a correct signature

The Cryptographic Model

- All the possibility results until now assume that there exists perfectly secure channels
- If we restrict the adversary to polynomial time we can however implement the channels:
  - First model the secure channels using some ideal functionality Fsmt for *secure message transmission*
    - S has input m and Fsmt outputs m to R
  - Then notice that our protocols are secure when Fsmt is used for communication
  - Then implement Fsmt in the cryptographic model
  - Then use the composition theorem to get a protocol for the cryptographic model

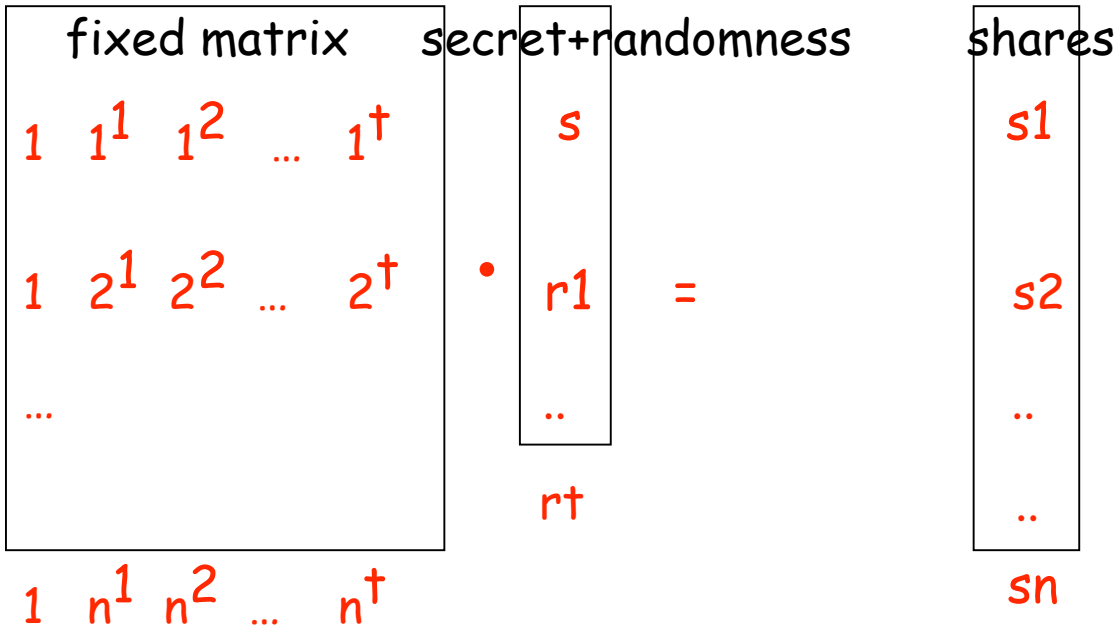
Implementing Fsmt

- Not surprisingly the ideal functionality Fsmt can be implemented using encryption. There are however a few technicalities to be observed:
- To guarantee *privacy* the encryption scheme must be semantically secure
  - I.e. it is randomized and leaks no information to a poly-time adversary
- To guarantee *independence of inputs* the encryption scheme must be secure against chosen ciphertext attack and the sender must include its identity in the plaintext
  - Avoids that you can send the same message as another party by copying the ciphertext
- If the encryption scheme should be secure against an adaptive adversary, it must additionally be what is known as *non-committing*

How to go from threshold to general adversaries.

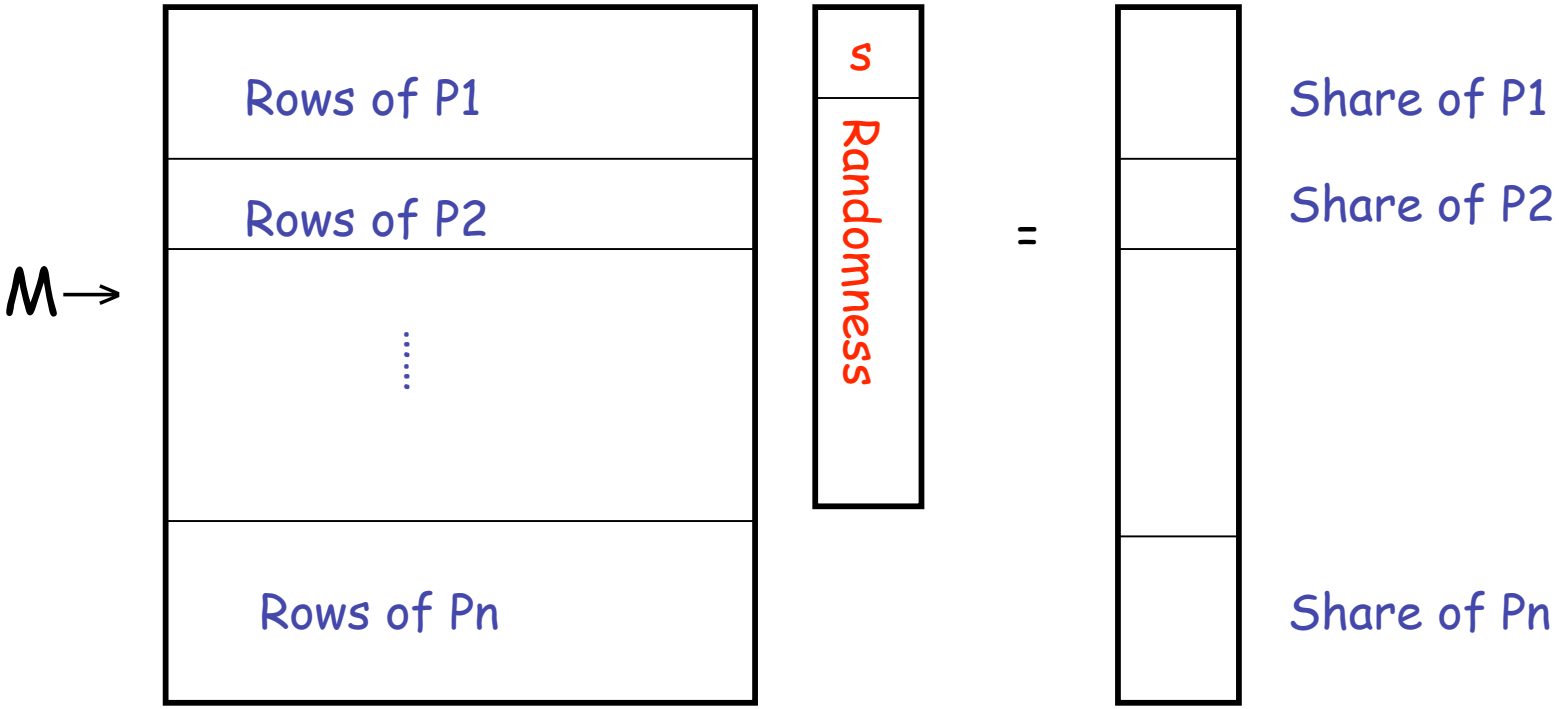
Use same ideas, but more general form of secret sharing…

Shamir's scheme can be written as



Each player "owns" a row of the matrix and is assigned the share corresponding to his row. Can be generalized to other matrices than Van

Linear Secret Sharing Schemes (LSSS).



Subset A can reconstruct s if their rows span (1, 0, 0,…,0), otherwise they have no information.

LSSS is most powerful general SS method known, can handle *any* adversary structure (see exercises). Shamir, Benaloh-Leichter, Van Dijk, Brickell are special cases.

Reminder

Adversary Structure _: family of subsets of **P**= {P1,…,Pn}

List of subsets the adversary can corrupt.

Threshold-t structure: contains all subsets of size at most t.

_ is Q3: for any A1,A2,A3 ∈ _, A1∪A2 ∪A3 is *smaller* than **P**

_ is Q2: for any A1,A2 ∈ _, A1∪A2 is *smaller* than **P**

To make our protocol work for general Q2/Q3 adversaries, plug in an LSSS M for _ instead of Shamir's scheme.

The bad news: only works if M has special property, must be *multiplicative*, M is not multiplicative in general.

The good news: from M, can always construct multiplicative M' of size at most twice that of M.

[CDM 00], see full version on web page

Could we use any secret sharing scheme? – probably not! [CDD 01].