

---

# Adaptiveness in the MPC

## *Introduction*

Stefan Dziembowski

`www.dziembowski.net`

Institute of Informatics

Warsaw University



# Historical note

---

## Multiparty Computations. (MPC)

Started by:

Yao, Protocols for secure computations, 1997

Goldreich, Micali, Wigderson How to play any mental game — a completeness theorem for protocols with honest majority, 1986

# Plan

---

1. Introduction to the MPC ←
2. Introduction to Adaptiveness in the MPC
3. Formal security definitions
4. Some equivalence and non-equivalence proofs

# Example 1: love problem

---

Alice

# Example 1: love problem

---

Alice and Bob

# Example 1: love problem

---

Alice and Bob

- want to check if they love each other
- keeping their feelings as secret as possible.

# Example 1 – more formally

---

Alice has a private input  $a$  and Bob has a private input  $b$ :

$$a := \begin{cases} 1 & \text{if Alice loves Bob} \\ 0 & \text{otherwise} \end{cases} \quad b := \begin{cases} 1 & \text{if Bob loves Alice} \\ 0 & \text{otherwise} \end{cases}$$

The goal of Alice and Bob is to compute  $f(a, b) := a \wedge b$  while keeping their inputs **as private as possible**.

# Example 1 – more formally

---

Alice has a private input  $a$  and Bob has a private input  $b$ :

$$a := \begin{cases} 1 & \text{if Alice loves Bob} \\ 0 & \text{otherwise} \end{cases} \quad b := \begin{cases} 1 & \text{if Bob loves Alice} \\ 0 & \text{otherwise} \end{cases}$$

The goal of Alice and Bob is to compute  $f(a, b) := a \wedge b$  while keeping their inputs **as private as possible**. I.e.:

- If  $f(a, b) = 1$  then  $a = b = 1$ , so no privacy is possible.

# Example 1 – more formally

---

Alice has a private input  $a$  and Bob has a private input  $b$ :

$$a := \begin{cases} 1 & \text{if Alice loves Bob} \\ 0 & \text{otherwise} \end{cases} \quad b := \begin{cases} 1 & \text{if Bob loves Alice} \\ 0 & \text{otherwise} \end{cases}$$

The goal of Alice and Bob is to compute  $f(a, b) := a \wedge b$  while keeping their inputs **as private as possible**. I.e.:

- If  $f(a, b) = 1$  then  $a = b = 1$ , so no privacy is possible.
- But: if  $a = 0$  then  $f(a, 0) = f(a, 1) = 0$ , so Alice doesn't know whether Bob loves her or not!

# Example 2: coin tossing

---

Alice and Bob want to toss a coin over a phone line (or Internet).

More precisely:

- Alice and Bob start with no inputs.
- They want to obtain a bit

$$r := \begin{cases} 0 & \text{with probability } \frac{1}{2} \\ 1 & \text{with probability } \frac{1}{2} \end{cases}$$

# Example: e-voting

---

Suppose we have a group  $\{P_1, \dots, P_n\}$  of people. Each  $P_i$  has a private input

$$x_i := \begin{cases} 1 & \text{if he votes „yes”} \\ 0 & \text{otherwise.} \end{cases}$$

They want to **privately** compute a value of a function

$$f(x_1, \dots, x_n) := x_1 + \dots + x_n.$$

# The goal

---

Consider a group  $\{P_1, \dots, P_n\}$  of people and a fixed (publicly known) function

$$f : D_1 \times \dots \times D_n \rightarrow C_1 \times \dots \times C_n$$

# The goal

---

Consider a group  $\{P_1, \dots, P_n\}$  of people and a fixed (publicly known) function

$$f : D_1 \times \dots \times D_n \rightarrow C_1 \times \dots \times C_n$$

Each  $P_i$  holds his **private input**  $x_i \in D_i$ .

# The goal

---

Consider a group  $\{P_1, \dots, P_n\}$  of people and a fixed (publicly known) function

$$f : D_1 \times \dots \times D_n \rightarrow C_1 \times \dots \times C_n$$

Each  $P_i$  holds his **private input**  $x_i \in D_i$ .

Set  $(y_1, \dots, y_n) := f(x_1, \dots, x_n)$

# The goal

---

Consider a group  $\{P_1, \dots, P_n\}$  of people and a fixed (publicly known) function

$$f : D_1 \times \dots \times D_n \rightarrow C_1 \times \dots \times C_n$$

Each  $P_i$  holds his **private input**  $x_i \in D_i$ .

Set  $(y_1, \dots, y_n) := f(x_1, \dots, x_n)$

**Goal:** construct a protocol  $\Pi_f$  such that as a result each player  $P_i$  learns the value of  $y_i$ .

# The goal

---

Consider a group  $\{P_1, \dots, P_n\}$  of people and a fixed (publicly known) function

$$f : D_1 \times \dots \times D_n \rightarrow C_1 \times \dots \times C_n$$

Each  $P_i$  holds his **private input**  $x_i \in D_i$ .

Set  $(y_1, \dots, y_n) := f(x_1, \dots, x_n)$

**Goal:** construct a protocol  $\Pi_f$  such that as a result each player  $P_i$  learns the value of  $y_i$ .

Moreover, the protocol should be **secure!**.

# What is security?

---

On the next few slides we are going to discuss the notion of **security**.

# What is security?

---

On the next few slides we are going to discuss the notion of **security**.

As it turns out this notion can have different meaning.

Everything depends on the model. . .

# What is security?

---

On the next few slides we are going to discuss the notion of **security**.

As it turns out this notion can have different meaning.

Everything depends on the model. . .

In general we will assume that

a **protocol**

is attacked by an

**adversary.**

# The adversary

---

Some of the players  $P_1, \dots, P_n$  may be cheating, i.e. they may try to abuse the protocol. Such players are called **corrupted**.

# The adversary

---

Some of the players  $P_1, \dots, P_n$  may be cheating, i.e. they may try to abuse the protocol. Such players are called **corrupted**.

In order to make the assumptions as pessimistic as we can we will assume that **the corrupted players may act in coalitions**.

# The adversary

---

Some of the players  $P_1, \dots, P_n$  may be cheating, i.e. they may try to abuse the protocol. Such players are called **corrupted**.

In order to make the assumptions as pessimistic as we can we will assume that **the corrupted players may act in coalitions**.

To model it we assume that there exists a single

adversary  $\mathcal{A}$

who can corrupt certain players.

# The adversary

---

Some of the players  $P_1, \dots, P_n$  may be cheating, i.e. they may try to abuse the protocol. Such players are called **corrupted**.

In order to make the assumptions as pessimistic as we can we will assume that **the corrupted players may act in coalitions**.

To model it we assume that there exists a single

adversary  $\mathcal{A}$

who can corrupt certain players.

The adversary  $\mathcal{A}$  is modelled as a probabilistic Turing machine.

We will now have a look on what the adversary is allowed to do.

# Active/passive adversary

---

Once a player  $P_i$  gets corrupted the adversary takes a control over him.

**Passive adversary** He gets access to all internal data of  $P_i$  and all the messages that come to  $P_i$ . (This is also called „honest but curious“.)

# Active/passive adversary

---

Once a player  $P_i$  gets corrupted the adversary takes a control over him.

**Passive adversary** He gets access to all internal data of  $P_i$  and all the messages that come to  $P_i$ . (This is also called „honest but curious“.)

**Active adversary** He also gets a full control over the actions of  $P_i$ .

# Active/passive adversary

---

Once a player  $P_i$  gets corrupted the adversary takes a control over him.

**Passive adversary** He gets access to all internal data of  $P_i$  and all the messages that come to  $P_i$ . (This is also called „honest but curious“.)

**Active adversary** He also gets a full control over the actions of  $P_i$ .

The passive model is considered in the literature because of:

**methodological reasons** A passively-secure protocol can often be upgraded to an actively secure one.

# Active/passive adversary

---

Once a player  $P_i$  gets corrupted the adversary takes a control over him.

**Passive adversary** He gets access to all internal data of  $P_i$  and all the messages that come to  $P_i$ . (This is also called „honest but curious“.)

**Active adversary** He also gets a full control over the actions of  $P_i$ .

The passive model is considered in the literature because of:

**methodological reasons** A passively-secure protocol can often be upgraded to an actively secure one.

**practical reasons** In some applications passive security is better than no security.

# The power of the adversary

---

Depending on the setting the adversary has

**computational setting:** limited computing power (usually it is randomized poly-time),

# The power of the adversary

---

Depending on the setting the adversary has

**computational setting:** limited computing power (usually it is randomized poly-time),

**information-theoretic setting:** unlimited computing power, or

# The power of the adversary

---

Depending on the setting the adversary has

**computational setting:** limited computing power (usually it is randomized poly-time),

**information-theoretic setting:** unlimited computing power, or

**non-standard setting:** unlimited computing power but is limited in some other way (noisy channels, memory bounded cryptography).

# The power of the adversary

---

Depending on the setting the adversary has

**computational setting:** limited computing power (usually it is randomized poly-time),

**information-theoretic setting:** unlimited computing power, or

**non-standard setting:** unlimited computing power but is limited in some other way (noisy channels, memory bounded cryptography).

In case of the computational setting we need to base the security on some unproven assumptions (e.g. existence of one-way trap door permutations).

In the other cases there exist proofs without any unproven assumptions.

---

# Possible corruptions

---

In most of the settings we have to assume that the adversary cannot corrupt as many players as he wants.

# Possible corruptions

---

In most of the settings we have to assume that the adversary cannot corrupt as many players as he wants.

Often, in order for a protocol to be secure we assume that the adversary can corrupt at most  $t < n$  players. This is called a **threshold adversary**.

# Possible corruptions

---

In most of the settings we have to assume that the adversary cannot corrupt as many players as he wants.

Often, in order for a protocol to be secure we assume that the adversary can corrupt at most  $t < n$  players. This is called a **threshold adversary**.

More generally: we can specify an

**adversary structure**, i.e.: a (subset-closed) family  $\mathcal{F}$  of the subsets of the set  $\{P_1, \dots, P_n\}$ .

The adversary can corrupt a set  $X$  only if  $X \in \mathcal{F}$ .

# Example of an adversary structure

$\mathcal{P}$  — a set of 5 politicians,  $\mathcal{S}$  — a set of 5 scientists  $\mathcal{S}$ .



The adversary can corrupt at most 4 politicians and 1 scientist.

$$\mathcal{F} := \{X : |X \cap \mathcal{S}| \leq 1 \wedge |X \cap \mathcal{P}| \leq 4\}$$

# Adaptive adversary

---

There are two more options to consider.

**adaptive security** The adversary has to decide whom he corrupts before the execution starts.

# Adaptive adversary

---

There are two more options to consider.

**adaptive security** The adversary has to decide whom he corrupts before the execution starts.

**non-adaptive security** The adversary corrupt the players **adaptively** one-by-one during the execution of the protocol.

# Adaptive adversary

---

There are two more options to consider.

**adaptive security** The adversary has to decide whom he corrupts before the execution starts.

**non-adaptive security** The adversary corrupt the players **adaptively** one-by-one during the execution of the protocol.

The difference between these two notions is the main topic of this talk!

# The communication channels

---

Usually we assume that there exists a connection between every pair of players. Depending on the setting we assume that

**computational setting** the adversary can eavesdrop the communication between the parties.

**information-theoretic setting** the channels are secure.

# The communication channels

---

Usually we assume that there exists a connection between every pair of players. Depending on the setting we assume that

**computational setting** the adversary can eavesdrop the communication between the parties.

**information-theoretic setting** the channels are secure.

We sometimes also assume an existence of a **broadcast channel** (available to every player).

# The communication channels

---

Usually we assume that there exists a connection between every pair of players. Depending on the setting we assume that

**computational setting** the adversary can eavesdrop the communication between the parties.

**information-theoretic setting** the channels are secure.

We sometimes also assume an existence of a **broadcast channel** (available to every player).

Moreover the network may be **synchronous**, or not (usually we assume it is).

# Security — informally

---

Informally speaking security consists of 2 requirements:

**Privacy** The adversary doesn't learn anything about the inputs of the honest players (that is not implied by the inputs and outputs of the corrupted players)

# Security — informally

---

Informally speaking security consists of 2 requirements:

**Privacy** The adversary doesn't learn anything about the inputs of the honest players (that is not implied by the inputs and outputs of the corrupted players)

**Correctness** The adversary cannot influence the outputs of the honest players (in other way than by substituting the inputs of the corrupted players).

# Security — informally

---

Informally speaking security consists of 2 requirements:

**Privacy** The adversary doesn't learn anything about the inputs of the honest players (that is not implied by the inputs and outputs of the corrupted players)

**Correctness** The adversary cannot influence the outputs of the honest players (in other way than by substituting the inputs of the corrupted players).

**Note:** When computing  $f(a, b) = a \wedge b$  if  $a = 1$  then Alice gets full information about  $b$ .

# Security — informally

---

Informally speaking security consists of 2 requirements:

**Privacy** The adversary doesn't learn anything about the inputs of the honest players (that is not implied by the inputs and outputs of the corrupted players)

**Correctness** The adversary cannot influence the outputs of the honest players (in other way than by substituting the inputs of the corrupted players).

**Note:** When computing  $f(a, b) = a \wedge b$  if  $a = 1$  then Alice gets full information about  $b$ .

A corrupted Alice can always lie (and say that she loves Bob although she doesn't).

# Ideal model

---

To capture the phenomenon we introduce an  
**ideal model for  $f$** .

In the ideal model in which we assume an existence of an incorruptible **trusted party  $T_f$** . The computation is done by  $T_f$ :

# Ideal model

---

To capture the phenomenon we introduce an  
**ideal model for  $f$ .**

In the ideal model in which we assume an existence of an incorruptible **trusted party  $T_f$** . The computation is done by  $T_f$ :

1. the players send their inputs  $x_1, \dots, x_n$  to  $T_f$

# Ideal model

---

To capture the phenomenon we introduce an  
**ideal model for  $f$** .

In the ideal model in which we assume an existence of an incorruptible **trusted party  $T_f$** . The computation is done by  $T_f$ :

1. the players send their inputs  $x_1, \dots, x_n$  to  $T_f$
2.  $T$  computes  $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$

# Ideal model

---

To capture the phenomenon we introduce an  
**ideal model for  $f$ .**

In the ideal model we assume the existence of an incorruptible **trusted party  $T_f$** . The computation is done by  $T_f$ :

1. the players send their inputs  $x_1, \dots, x_n$  to  $T_f$
2.  $T$  computes  $(y_1, \dots, y_n) = f(x_1, \dots, x_n)$
3.  $T$  sends each  $y_i$  to  $P_i$ . Each  $P_i$  outputs  $y_i$ .

# Definition of security

---

We will say that

the protocol  $\Pi$  securely computes  $f$

if (informally):

whatever the adversary can achieve attacking  $\Pi$  he can also achieve in the ideal-model.

# Definition of security

---

We will say that

the protocol  $\Pi$  securely computes  $f$

if (informally):

whatever the adversary can achieve attacking  $\Pi$  he can also achieve in the ideal-model.

To be more formal we would need to define the notions of a **simulator** and **environment**.

# Definition of security

---

We will say that

the protocol  $\Pi$  securely computes  $f$

if (informally):

whatever the adversary can achieve attacking  $\Pi$  he can also achieve in the ideal-model.

To be more formal we would need to define the notions of a **simulator** and **environment**.

We will discuss it later. Our definitions we be based on:

**Canetti** Security and Composition of Multi-party  
Cryptographic protocols, 2000

# Perfect/imperfect security

---

If we are working in the information-theoretic model we can either

- require perfect security — i.e. the security holds with probability 1, or

# Perfect/imperfect security

---

If we are working in the information-theoretic model we can either

- require perfect security — i.e. the security holds with probability 1, or
- allow for imperfect security — i.e. there is a **negligible** probability of error.

# Perfect/imperfect security

---

If we are working in the information-theoretic model we can either

- require perfect security — i.e. the security holds with probability 1, or
- allow for imperfect security — i.e. there is a negligible probability of error.

More precisely, the parties input a security parameter  $k$ , and:

for any  $c$  the probability of error is smaller than  $k^{-c}$ , for  $k$  sufficiently large.

# Reactive systems

---

Recall that the trusted party  $T$  simply computes a function.

We can consider a more general setting in which  $T$  is performing some on-line process. The computation is done in phases.

# Reactive systems

---

Recall that the trusted party  $T$  simply computes a function.

We can consider a more general setting in which  $T$  is performing some on-line process. The computation is done in phases.

Example:

**Phase 1** one of the players deposits a secret

**Phase 2** the secret is revealed to the players.

# Randomized functions

---

The function  $f$  can be

randomized

# Randomized functions

---

The function  $f$  can be

randomized

To model it we assume that the function  $f$  takes an extra input  $R$  selected uniformly at random from some domain  $\mathcal{R}$ .  
I.e. the function  $f$  has a type:

$$f : D_1 \times \cdots \times D_n \times \mathcal{R} \rightarrow C_1 \times \cdots \times C_n$$

# Randomized functions

---

The function  $f$  can be

randomized

To model it we assume that the function  $f$  takes an extra input  $R$  selected uniformly at random from some domain  $\mathcal{R}$ .  
I.e. the function  $f$  has a type:

$$f : D_1 \times \cdots \times D_n \times \mathcal{R} \rightarrow C_1 \times \cdots \times C_n$$

**Example.** The coin-tossing protocol:

$$f : \emptyset \times \emptyset \times \{0, 1\} \rightarrow \{0, 1\}$$

with  $f(\cdot, \cdot, r) := r$ .

# The fundamental question

---

For an arbitrary (efficiently computable) function

$$f : D_1 \times \cdots \times D_n \times R \rightarrow C_1 \times \cdots \times C_n$$

does there exist an efficient secure multi-party protocol computing  $f$ ?

# The fundamental question

---

For an arbitrary (efficiently computable) function

$$f : D_1 \times \cdots \times D_n \times R \rightarrow C_1 \times \cdots \times C_n$$

does there exist an efficient secure multi-party protocol computing  $f$ ?

**Answer.** depends on the setting ...

# Computational setting

---

Can one construct a protocol for any  $f$ ?

number $t$ of corrupted players	passive	active
$t < n/2$	YES	YES
$t \geq n/2$	YES	YES*

\*If  $t \geq n/2$  and the adversary is active then the adversary can interrupt the execution at any time :-)

# Information-theoretic setting

---

In a model without a broadcast channel:

number $t$ of corrupted players	passive	active
$t < n/3$	YES	YES
$n/3 \leq t < n/2$	YES	NO*
$n/2 \leq t$	NO	NO

\*If  $n/3 \leq t < n/2$  and the adversary is active then one can construct a secure protocol assuming an existence of a broadcast channel.

# References

---

Goldreich, Micali, Widgerson How to play any mental game — a completeness theorem for protocols with honest majority., 1986

Chaum, Crepau, Damgård Multi-Party Unconditionally Secure Protocols. 1988

Ben-Or, Goldwasser, Widgerson Completeness theorems for Non-cryptographic, Fault Tolerant Distributed Computations. 1988

T. Rabin, Ben-Or Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. 1989

# Generalization

---

The above results can be generalized to arbitrary adversary structures as follows. Let  $\mathcal{P}$  be the entire set of players. Let  $\mathcal{F}$  denote an adversary structure.

# Generalization

---

The above results can be generalized to arbitrary adversary structures as follows. Let  $\mathcal{P}$  be the entire set of players. Let  $\mathcal{F}$  denote an adversary structure.

The „ $t < n/2$ ” requirement can be translated to:

$Q_2$

For every  $A, B \in \mathcal{F}$  we have  $A \cup B \neq \mathcal{P}$ .

# Generalization

---

The above results can be generalized to arbitrary adversary structures as follows. Let  $\mathcal{P}$  be the entire set of players. Let  $\mathcal{F}$  denote an adversary structure.

The „ $t < n/2$ ” requirement can be translated to:

$Q_2$

For every  $A, B \in \mathcal{F}$  we have  $A \cup B \neq \mathcal{P}$ .

The „ $t < n/3$ ” requirement can be translated to:

$Q_3$

For every  $A, B, C \in \mathcal{F}$  we have  $A \cup B \cup C \neq \mathcal{P}$ .

# Generalization

---

The above results can be generalized to arbitrary adversary structures as follows. Let  $\mathcal{P}$  be the entire set of players. Let  $\mathcal{F}$  denote an adversary structure.

The „ $t < n/2$ ” requirement can be translated to:

$$\boxed{Q_2} \quad \text{For every } A, B \in \mathcal{F} \text{ we have } A \cup B \neq \mathcal{P}.$$

The „ $t < n/3$ ” requirement can be translated to:

$$\boxed{Q_3} \quad \text{For every } A, B, C \in \mathcal{F} \text{ we have } A \cup B \cup C \neq \mathcal{P}.$$

**Hirt, Maurer** Complete characterization of adversaries tolerable in secure multi-party computations, 1997

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ←
3. Formal security definitions
4. Some equivalence and non-equivalence proofs

# Adaptive vs. nonadaptive security

---

We will be interested in examining the relationship between adaptive and non-adaptive security

The presentation is based on mostly on

Ran Canetti, Ivan Damgard, Stefan Dziembowski, Yuval Ishai, and Tal Malkin On Adaptive vs. Non-adaptive Security of Multiparty Protocols. EUROCRYPT 2001.

# Adaptive $\not\equiv$ non-adaptive

---

Before we go to the formal definitions we will first have a look at the example of a protocol that is **secure non-adaptively** but **not secure adaptively**.

# Adaptive $\not\equiv$ non-adaptive

---

Before we go to the formal definitions we will first have a look at the example of a protocol that is **secure non-adaptively** but **not secure adaptively**.

This protocol was first published in

**T. Rabin, Ben-Or** Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. 1989

(it was claimed there to be secure **adaptively**)

# Adaptive $\not\equiv$ non-adaptive

---

Before we go to the formal definitions we will first have a look at the example of a protocol that is **secure non-adaptively** but **not secure adaptively**.

This protocol was first published in

**T. Rabin, Ben-Or** Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. 1989

(it was claimed there to be secure **adaptively**)

The non-adaptive insecurity was observed in

**Cramer, Damgård, Dziembowski, Hirt and Rabin** Efficient Multiparty Computations Secure Against an Adaptive Adversary, EUROCRYPT 1999

# Weak Secret Sharing (WSS)

---

The protocol of [RabinBenOr89] implements **Weak Secret Sharing (WSS)**.  
It is a „distributed analogue of commitment schemes.  
It is an example of a reactive system. It consist of 2 phases:

# Weak Secret Sharing (WSS)

---

The protocol of [RabinBenOr89] implements **Weak Secret Sharing (WSS)**.

It is a „distributed analogue of commitment schemes.

It is an example of a reactive system. It consist of 2 phases:

**Committing** A player called a *Dealer* inputs a secret value  $s$ . Basing on it he distributes some information to the other players.

# Weak Secret Sharing (WSS)

---

The protocol of [RabinBenOr89] implements **Weak Secret Sharing (WSS)**.

It is a „distributed analogue of commitment schemes.

It is an example of a reactive system. It consist of 2 phases:

**Committing** A player called a *Dealer* inputs a secret value  $s$ . Basing on it he distributes some information to the other players.

An adversary should get essentially no information about  $s$ .

(This is called: **hiding**).

# Weak Secret Sharing (WSS)

---

The protocol of [RabinBenOr89] implements **Weak Secret Sharing (WSS)**.

It is a „distributed analogue of commitment schemes.

It is an example of a reactive system. It consist of 2 phases:

**Committing** A player called a *Dealer* inputs a secret value  $s$ . Basing on it he distributes some information to the other players.

An adversary should get essentially no information about  $s$ .

(This is called: **hiding**).

**Opening** If the *Dealer* is honest then all the players output  $s$ .

# Weak Secret Sharing (WSS)

---

The protocol of [RabinBenOr89] implements **Weak Secret Sharing (WSS)**. It is a „distributed analogue of commitment schemes.

It is an example of a reactive system. It consist of 2 phases:

**Committing** A player called a *Dealer* inputs a secret value  $s$ . Basing on it he distributes some information to the other players.

An adversary should get essentially no information about  $s$ .

(This is called: **hiding**).

**Opening** If the *Dealer* is honest then all the players output  $s$ .

An adversary cannot force the players to output any other value (even when he corrupts the *Dealer*). But he can make them output no value at all.

(This is called **binding**)

# Weak Secret Sharing (WSS)

---

The protocol of [RabinBenOr89] implements **Weak Secret Sharing (WSS)**. It is a „distributed analogue of commitment schemes.

It is an example of a reactive system. It consist of 2 phases:

**Committing** A player called a *Dealer* inputs a secret value  $s$ . Basing on it he distributes some information to the other players.

An adversary should get essentially no information about  $s$ .

(This is called: **hiding**).

**Opening** If the *Dealer* is honest then all the players output  $s$ .

An adversary cannot force the players to output any other value (even when he corrupts the *Dealer*). But he can make them output no value at all.

(This is called **binding**)

We will assume that adversary can corrupt at most  $t < \frac{n}{2}$  players and that a **broadcast channel** is available for the players.

# Shamir's Secret Sharing — specification

---

To construct a WSS protocol we use the **Shamir's Secret Sharing** protocol.

# Shamir's Secret Sharing — specification

---

To construct a WSS protocol we use the **Shamir's Secret Sharing** protocol.

Let  $t$  be some threshold number. Secret Sharing is a protocol allowing the **Dealer** to share a secret value  $s$  among  $n$  players in such a way that

- every group of players of a size at most  $t$  gets no information about  $s$ , and

# Shamir's Secret Sharing — specification

---

To construct a WSS protocol we use the **Shamir's Secret Sharing** protocol. Let  $t$  be some threshold number. Secret Sharing is a protocol allowing the **Dealer** to share a secret value  $s$  among  $n$  players in such a way that

- every group of players of a size at most  $t$  gets no information about  $s$ , and
- every group of players of a size greater than  $t$  can reconstruct  $s$ .

# Shamir's Secret Sharing — specification

---

To construct a WSS protocol we use the **Shamir's Secret Sharing** protocol. Let  $t$  be some threshold number. Secret Sharing is a protocol allowing the **Dealer** to share a secret value  $s$  among  $n$  players in such a way that

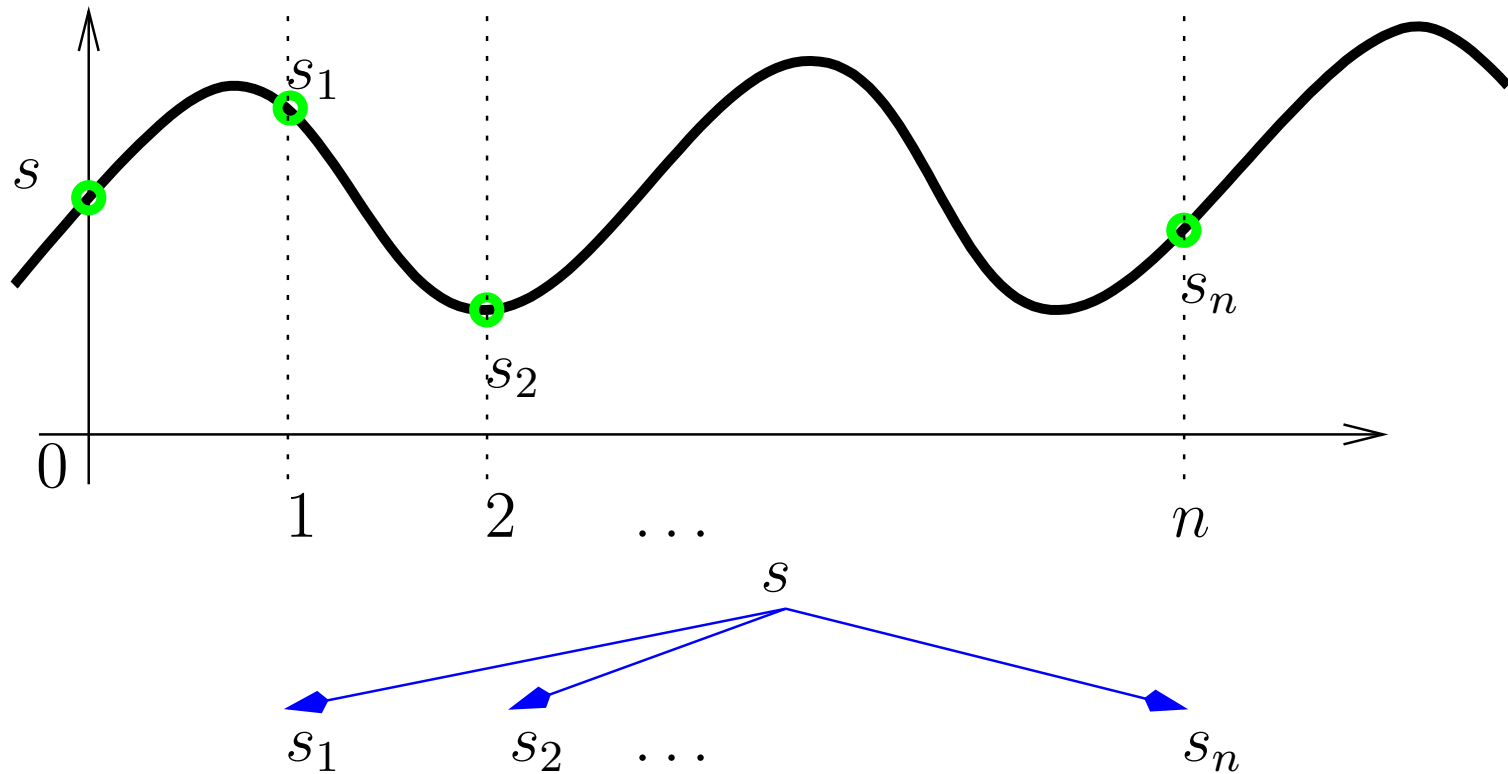
- every group of players of a size at most  $t$  gets no information about  $s$ , and
- every group of players of a size greater than  $t$  can reconstruct  $s$ .

The protocol should be secure against a passive (“honest but curious”) adversary.

We assume that  $s$  is an element of some finite field (of a reasonable size).

# Shamir's Secret Sharing — implementation

The *Dealer* chooses a random polynomial  $p$  of a degree  $t$  such that  $p(0) = s$ .



The *Dealer* sends each  $s_i$  to each  $P_i$

# WSS — implementation (1/2)

---

The main idea: use Information Checking (IC) protocol [RabinBenOr89].

**Intuition:** IC = distributed unconditionally secure signature scheme.

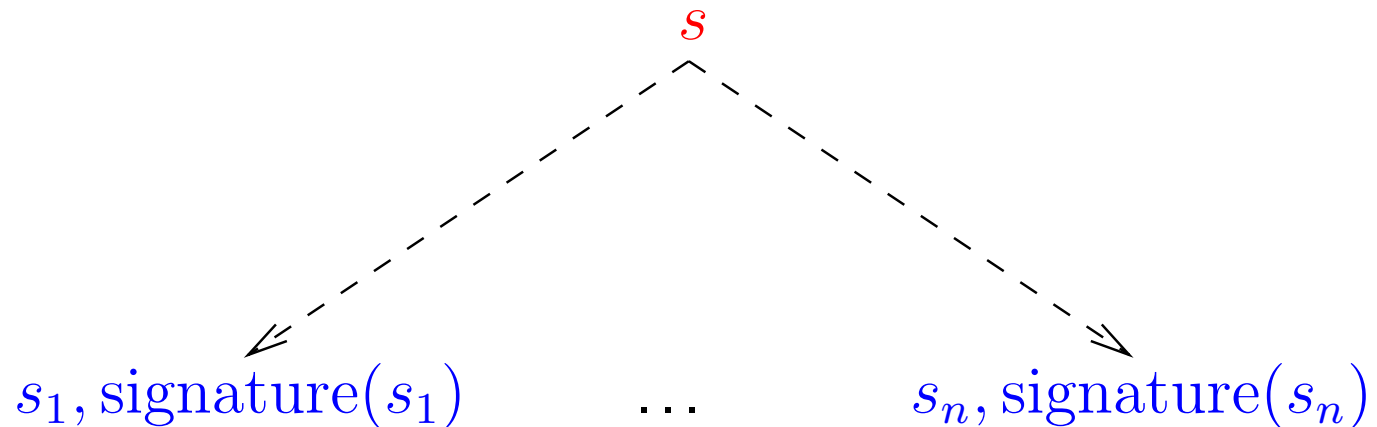
# WSS — implementation (1/2)

---

The main idea: use Information Checking (IC) protocol [RabinBenOr89].

**Intuition:** IC = distributed unconditionally secure signature scheme.

To commit to  $s$  the *Dealer* shares  $s$  with Shamir's Secret Sharing (with the same  $t$ ). To each share he attaches has to „signature”.



# WSS — implementation (2/2)

---

To open the commitment the *Dealer* broadcasts  $s$  together with all the shares  $s_1, \dots, s_n$ .

# WSS — implementation (2/2)

---

To open the commitment the *Dealer* broadcasts  $s$  together with all the shares  $s_1, \dots, s_n$ .

1. If  $s_1, \dots, s_n$  do not interpolate  $s$  then the players halt.

# WSS — implementation (2/2)

---

To open the commitment the *Dealer* broadcasts  $s$  together with all the shares  $s_1, \dots, s_n$ .

1. If  $s_1, \dots, s_n$  do not interpolate  $s$  then the players halt.
2. If a player  $P_i$  discovers that the *Dealer* broadcasted a wrong  $s_i$  then he broadcasts  $s_i, \text{signature}(s_i)$ . If the players see such a broadcast then they halt.

# WSS — implementation (2/2)

---

To open the commitment the *Dealer* broadcasts  $s$  together with all the shares  $s_1, \dots, s_n$ .

1. If  $s_1, \dots, s_n$  do not interpolate  $s$  then the players halt.
2. If a player  $P_i$  discovers that the *Dealer* broadcasted a wrong  $s_i$  then he broadcasts  $s_i, \text{signature}(s_i)$ . If the players see such a broadcast then they halt.
3. Otherwise the players output  $s$  (the opening is successful).

# WSS — security argument

---

Why is this secure? The following argument was used in [RabinBenOr89]:

# WSS — security argument

---

Why is this secure? The following argument was used in [RabinBenOr89]:

The only way for the *Dealer* to cheat is to broadcast (during the opening phase) some „false” shares  $s'_i$  of the corrupted players.

# WSS — security argument

---

Why is this secure? The following argument was used in [RabinBenOr89]:

The only way for the *Dealer* to cheat is to broadcast (during the opening phase) some „false” shares  $s'_i$  of the corrupted players.

There need to be at least  $t + 1$  honest players (because  $t < \frac{n}{2}$ ). For simplicity assume that the honest players are

$$\mathcal{H} = \{P_1, \dots, P_{t+1}\}$$

Their shares  $s_1, \dots, s_{t+1}$  interpolate a unique polynomial of degree  $t$ .

# WSS — security argument

---

Why is this secure? The following argument was used in [RabinBenOr89]:

The only way for the *Dealer* to cheat is to broadcast (during the opening phase) some „false” shares  $s'_i$  of the corrupted players.

There need to be at least  $t + 1$  honest players (because  $t < \frac{n}{2}$ ). For simplicity assume that the honest players are

$$\mathcal{H} = \{P_1, \dots, P_{t+1}\}$$

Their shares  $s_1, \dots, s_{t+1}$  interpolate a unique polynomial of degree  $t$ .

This cannot be changed by adding „false” shares...

# A problem

---

What is wrong with this reasoning?

# A problem

---

What is wrong with this reasoning?

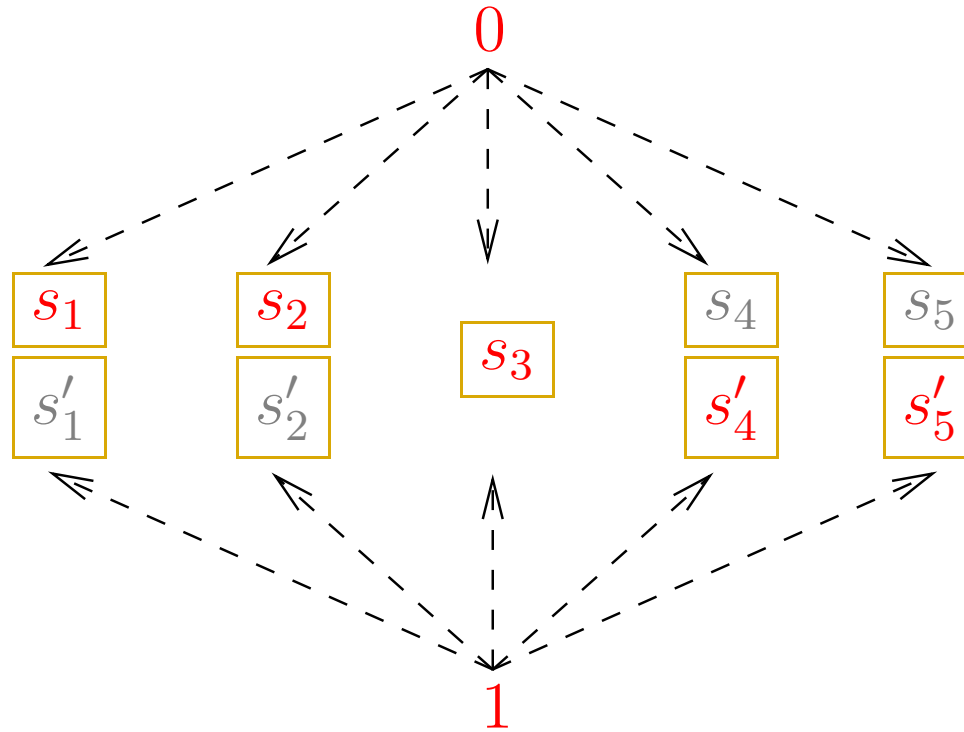
In the **non-adaptive** case everything is OK.

But in the **adaptive** case the argument brakes. This is because the set  $\mathcal{H}$  of the honest players can change over the time!

# An example

---

Suppose there are 5 players, at most 2 can be corrupted.



# Conclusion

---

There exist protocols that are secure **only non-adaptively**.

We will now systematically analyze the situations in which problems like this can occur.

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ✓
3. Formal security definitions ←
4. Some equivalence and non-equivalence proofs

# Security definitions – introduction

---

We follow the formulation of

Canetti Security and Composition of Multi-party Cryptographic protocols,  
2000

# Security definitions – introduction

---

We follow the formulation of

Canetti Security and Composition of Multi-party Cryptographic protocols,  
2000

**Main idea:** Suppose that we have proven that a protocol  $\Pi$  „emulates”  
a trusted party  $T_f$ .

# Security definitions – introduction

---

We follow the formulation of

Canetti Security and Composition of Multi-party Cryptographic protocols,  
2000

**Main idea:** Suppose that we have proven that a protocol  $\Pi$  „emulates”  
a trusted party  $T_f$ .

then we should be able to construct protocols that access  $T_f$  and then  
securely exchange  $T_f$  it  $\Pi$ .

We are going to formalize the definition of the security in the adaptive  
case.

# Security definitions – introduction

---

We follow the formulation of

**Canetti** Security and Composition of Multi-party Cryptographic protocols,  
2000

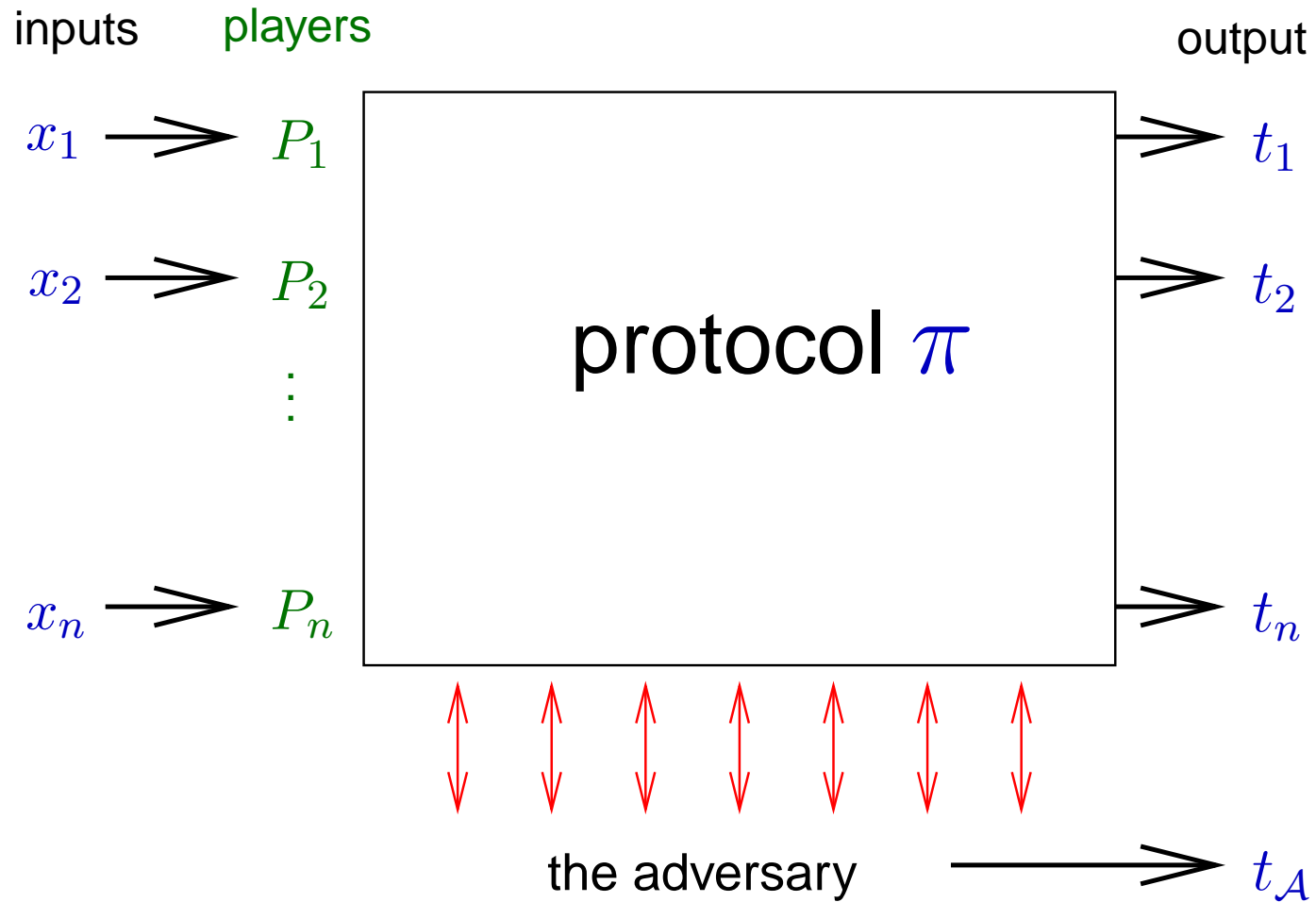
**Main idea:** Suppose that we have proven that a protocol  $\Pi$  „emulates”  
a trusted party  $T_f$ .

then we should be able to construct protocols that access  $T_f$  and then  
**securely exchange**  $T_f$  it  $\Pi$ .

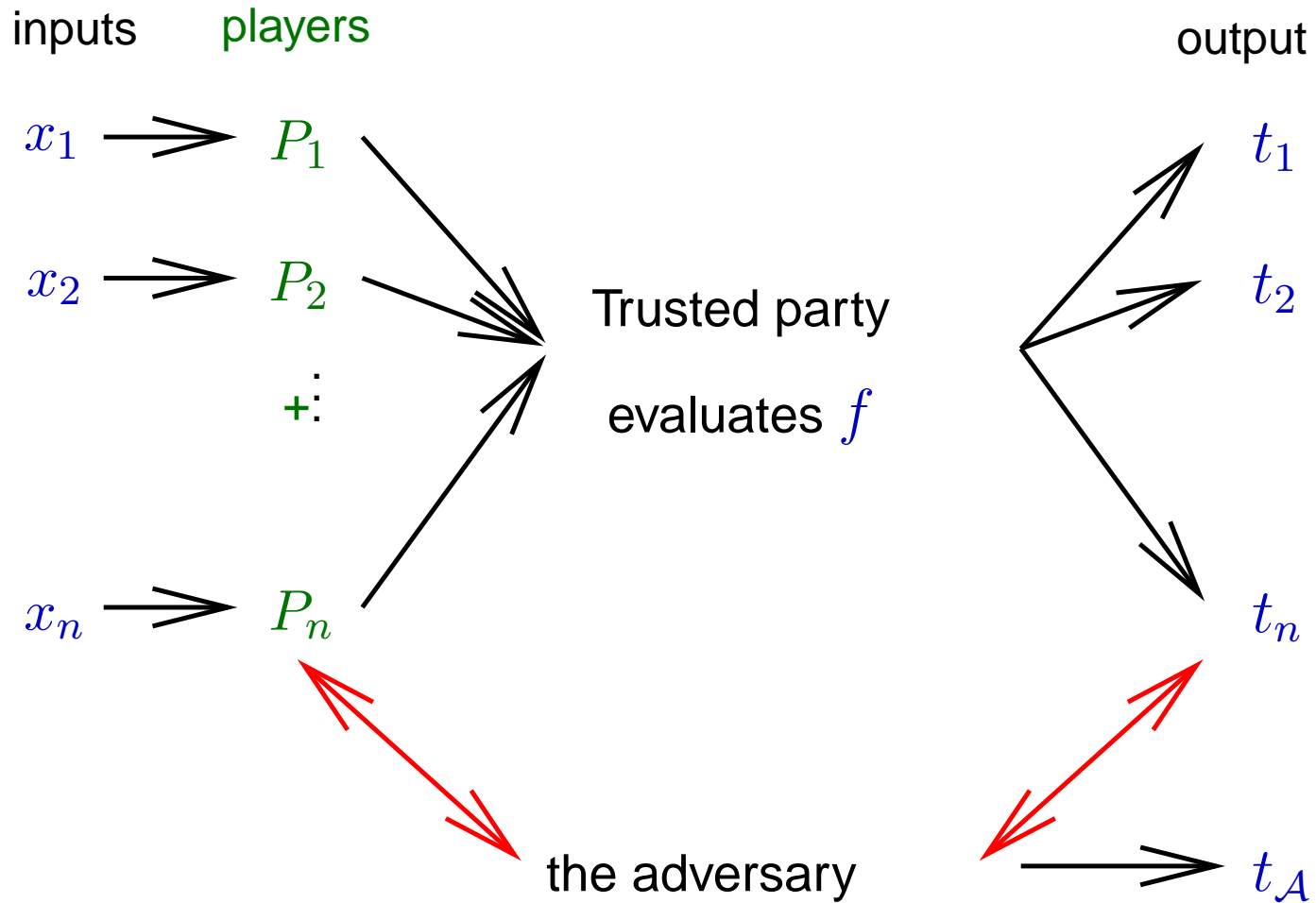
We are going to formalize the definition of the security in the adaptive  
case.

Let us first look again at the **real-life** and **ideal** executions.

# The real life execution



# The ideal model



In this case the adversary will usually be called a **simulator**.

# Formal definition of the real-life execution

---

We model the adversary  $\mathcal{A}$  and the players as a set  $\Pi$  of interactive and randomized **Turing Machines**  $P_1, \dots, P_n$

# Formal definition of the real-life execution

---

We model the adversary  $\mathcal{A}$  and the players as a set  $\Pi$  of interactive and randomized **Turing Machines**  $P_1, \dots, P_n$

Adversary  $\mathcal{A}$  receives a **random input**  $r_0$  and a security parameter  $k$ .

# Formal definition of the real-life execution

---

We model the adversary  $\mathcal{A}$  and the players as a set  $\Pi$  of interactive and randomized Turing Machines  $P_1, \dots, P_n$

Adversary  $\mathcal{A}$  receives a random input  $r_0$  and a security parameter  $k$ .

Each  $P_i$  receives an input  $x_i$  and a random input  $r_i$  and a security parameter  $k$ .

# Formal definition of the real-life execution

---

We model the adversary  $\mathcal{A}$  and the players as a set  $\Pi$  of interactive and randomized **Turing Machines**  $P_1, \dots, P_n$

Adversary  $\mathcal{A}$  receives a **random input**  $r_0$  and a security parameter  $k$ .

Each  $P_i$  receives an **input**  $x_i$  and a **random input**  $r_i$  and a security parameter  $k$ .

The computation proceeds in **rounds**. In each round  $\mathcal{A}$  can **adaptively corrupt** the players. Once a player  $P_i$  is corrupted  $\mathcal{A}$  receives **entire internal data** of a  $P_i$  which in particular includes:

- the input and the random input
- the history of the messages received by the player.

# Formal definition of the real-life execution

---

We model the adversary  $\mathcal{A}$  and the players as a set  $\Pi$  of interactive and randomized **Turing Machines**  $P_1, \dots, P_n$

Adversary  $\mathcal{A}$  receives a **random input**  $r_0$  and a security parameter  $k$ .

Each  $P_i$  receives an **input**  $x_i$  and a **random input**  $r_i$  and a security parameter  $k$ .

The computation proceeds in **rounds**. In each round  $\mathcal{A}$  can **adaptively corrupt** the players. Once a player  $P_i$  is corrupted  $\mathcal{A}$  receives **entire internal data** of a  $P_i$  which in particular includes:

- the input and the random input
- the history of the messages received by the player.

If  $\mathcal{A}$  is **active** then he gets a full control over  $P_i$ . In the **passive** case he can only read the messages sent to  $P_i$ .

# The output of the real-life execution

---

Every  $P_i$  outputs some value  $w_i$ . (The corrupted parties output  $\perp$ .)  
Adversary outputs  $w_{\mathcal{A}}$ . W.l.o.g we assume that it contains all the information that  $\mathcal{A}$  saw (plus maybe some function of it).

# The output of the real-life execution

---

Every  $P_i$  outputs some value  $w_i$ . (The corrupted parties output  $\perp$ .)

Adversary outputs  $w_{\mathcal{A}}$ . W.l.o.g we assume that it contains all the information that  $\mathcal{A}$  saw (plus maybe some function of it).

Fixed  $\vec{r} = r_0, \dots, r_n$ ,  $\vec{x} = x_1, \dots, x_t$  and  $k$  determine uniquely the execution of an adversary  $\mathcal{A}$  against a protocol  $\Pi$ .

# The output of the real-life execution

---

Every  $P_i$  outputs some value  $w_i$ . (The corrupted parties output  $\perp$ .)

Adversary outputs  $w_{\mathcal{A}}$ . W.l.o.g we assume that it contains all the information that  $\mathcal{A}$  saw (plus maybe some function of it).

Fixed  $\vec{r} = r_0, \dots, r_n$ ,  $\vec{x} = x_1, \dots, x_t$  and  $k$  determine uniquely the execution of an adversary  $\mathcal{A}$  against a protocol  $\Pi$ .

Set:

$$\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x}, \vec{r}) := (w_{\mathcal{A}}, w_1, \dots, w_n)$$

# The output of the real-life execution

---

Every  $P_i$  outputs some value  $w_i$ . (The corrupted parties output  $\perp$ .)

Adversary outputs  $w_{\mathcal{A}}$ . W.l.o.g we assume that it contains all the information that  $\mathcal{A}$  saw (plus maybe some function of it).

Fixed  $\vec{r} = r_0, \dots, r_n$ ,  $\vec{x} = x_1, \dots, x_t$  and  $k$  determine uniquely the execution of an adversary  $\mathcal{A}$  against a protocol  $\Pi$ .

Set:

$$\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x}, \vec{r}) := (w_{\mathcal{A}}, w_1, \dots, w_n)$$

If we choose the random inputs  $\vec{r}$  uniformly at random then

$$\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})$$

is a probability distribution.

# Formal model of the ideal process (1/3)

---

Ideal model is parameterized by an  $n$ -party function  $f$  whose domain is:

$$\underbrace{\mathbb{N}}_{\text{sec. param.}} \times \underbrace{\{0, 1\}}_{\text{input of } \mathcal{A}} \times \underbrace{(\{0, 1\}^*)^n}_{\text{players' inputs}} \times \underbrace{\{0, 1\}^*}_{\text{random input}}$$

# Formal model of the ideal process (1/3)

---

Ideal model is parameterized by an  $n$ -party function  $f$  whose domain is:

$$\underbrace{\mathbb{N}}_{\text{sec. param.}} \times \underbrace{\{0, 1\}}_{\text{input of } \mathcal{A}} \times \underbrace{(\{0, 1\}^*)^n}_{\text{players' inputs}} \times \underbrace{\{0, 1\}^*}_{\text{random input}}$$



and range is:

$$\underbrace{\{0, 1\}}_{\text{output of } \mathcal{A}} \times \underbrace{(\{0, 1\}^*)^n}_{\text{players' outputs}}$$

# Formal model of the ideal process (2/3)

---

Recall that the „adversary” in the ideal process is an interactive, randomized Turing Machine called a **simulator** ( $\mathcal{S}$ )

$\vec{x} = (x_1, \dots, x_n)$  — inputs of the players,

$r_0$  — random input of  $\mathcal{S}$ ,  $r_f$  — random input of the trusted party.

The ideal process proceeds in **stages**:

**First corruption stage**  $\mathcal{S}$  receives  $r_0$  and adaptively „corrupts some players”. Once a player  $P_i$  gets corrupted  $\mathcal{S}$  learns  $x_i$ .

**Computation stage** The players send their input to the trusted party  $T$ . In the „active” case  $\mathcal{S}$  has a right to choose the inputs of the corrupted players.  $\mathcal{S}$  also chooses the input  $y_0$ . Let  $\vec{y} = y_1, \dots, y_n$  be the value received by  $T$ .

$T$  computes  $(v_0, \dots, v_n) = f(k, \vec{y}, r_f)$ .

$T$  sends each  $v_i$  to  $P_i$  and  $v_0$  to  $\mathcal{S}$ .

# Formal model of the ideal process (3/3)

---

**Second corruption stage  $\mathcal{S}$**  learns the values received by the corrupted players. He can now adaptively corrupt come more players (and learn their inputs and outputs).

# Formal model of the ideal process (3/3)

---

**Second corruption stage**  $\mathcal{S}$  learns the values received by the corrupted players. He can now adaptively corrupt some more players (and learn their inputs and outputs).

**Output** Each honest  $P_i$  outputs  $v_i$ . Each corrupted player outputs  $\perp$ . The simulator outputs some arbitrary value  $w_{\mathcal{S}}$ .

# Formal model of the ideal process (3/3)

---

**Second corruption stage**  $\mathcal{S}$  learns the values received by the corrupted players. He can now adaptively corrupt some more players (and learn their inputs and outputs).

**Output** Each honest  $P_i$  outputs  $v_i$ . Each corrupted player outputs  $\perp$ . The simulator outputs some arbitrary value  $w_{\mathcal{S}}$ .

Let

$$\text{ideal}_{f,\mathcal{S}}(k, \vec{x}, (r_0, r_f)) := (w_{\mathcal{S}}, w_1, \dots, w_n)$$

# Formal model of the ideal process (3/3)

---

**Second corruption stage**  $\mathcal{S}$  learns the values received by the corrupted players. He can now adaptively corrupt some more players (and learn their inputs and outputs).

**Output** Each honest  $P_i$  outputs  $v_i$ . Each corrupted player outputs  $\perp$ . The simulator outputs some arbitrary value  $w_{\mathcal{S}}$ .

Let

$$\text{ideal}_{f,\mathcal{S}}(k, \vec{x}, (r_0, r_f)) := (w_{\mathcal{S}}, w_1, \dots, w_n)$$

If we choose the random inputs  $(r_0, r_f)$  uniformly at random then

$$\text{ideal}_{f,\mathcal{S}}(k, \vec{x})$$

is a probability distribution.

# Comparing real with ideal model

---

If we fix a protocol  $\Pi$  and an adversary  $\mathcal{A}$  we get a sequence of distributions

$$\text{exec}_{\Pi, \mathcal{A}} := \{\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^*} \cdot$$

# Comparing real with ideal model

---

If we fix a protocol  $\Pi$  and an adversary  $\mathcal{A}$  we get a sequence of distributions

$$\text{exec}_{\Pi, \mathcal{A}} := \{\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^*} \cdot$$

If we fix a function  $f$  and a simulator  $\mathcal{S}$  then we get a sequence of distributions

$$\text{ideal} := \{\text{ideal}_{f, \mathcal{S}}(k, \vec{x}, (r_0, r_f))\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^*} \cdot$$

# Comparing real with ideal model

---

If we fix a protocol  $\Pi$  and an adversary  $\mathcal{A}$  we get a sequence of distributions

$$\text{exec}_{\Pi, \mathcal{A}} := \{\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^* \cdot}$$

If we fix a function  $f$  and a simulator  $\mathcal{S}$  then we get a sequence of distributions

$$\text{ideal} := \{\text{ideal}_{f, \mathcal{S}}(k, \vec{x}, (r_0, r_f))\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^* \cdot}$$

The intuitive statement:

„whatever the adversary can achieve attacking  $\Pi$  in the real-life he can be achieve in the ideal-process (computing  $f$ )”

can be now translated to the following:

# Comparing real with ideal model

---

If we fix a protocol  $\Pi$  and an adversary  $\mathcal{A}$  we get a sequence of distributions

$$\text{exec}_{\Pi, \mathcal{A}} := \{\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^* \cdot}$$

If we fix a function  $f$  and a simulator  $\mathcal{S}$  then we get a sequence of distributions

$$\text{ideal} := \{\text{ideal}_{f, \mathcal{S}}(k, \vec{x}, (r_0, r_f))\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^* \cdot}$$

The intuitive statement:

„whatever the adversary can achieve attacking  $\Pi$  in the real-life he can be achieve in the ideal-process (computing  $f$ )”

can be now translated to the following:

$\forall \mathcal{A}$

$\text{exec}_{\Pi, \mathcal{A}}$  „is close to”  $\text{ideal}_{f, \mathcal{S}}$

# Comparing real with ideal model

---

If we fix a protocol  $\Pi$  and an adversary  $\mathcal{A}$  we get a sequence of distributions

$$\text{exec}_{\Pi, \mathcal{A}} := \{\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^* \cdot}$$

If we fix a function  $f$  and a simulator  $\mathcal{S}$  then we get a sequence of distributions

$$\text{ideal} := \{\text{ideal}_{f, \mathcal{S}}(k, \vec{x}, (r_0, r_f))\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^* \cdot}$$

The intuitive statement:

„whatever the adversary can achieve attacking  $\Pi$  in the real-life he can be achieve in the ideal-process (computing  $f$ )”

can be now translated to the following:

$\forall \mathcal{A} \exists \mathcal{S}$  with a „similar” computing power such that  
 $\text{exec}_{\Pi, \mathcal{A}}$  „is close to”  $\text{ideal}_{f, \mathcal{S}}$

# Comparing real with ideal model

---

If we fix a protocol  $\Pi$  and an adversary  $\mathcal{A}$  we get a sequence of distributions

$$\text{exec}_{\Pi, \mathcal{A}} := \{\text{exec}_{\Pi, \mathcal{A}}(k, \vec{x})\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^*}.$$

If we fix a function  $f$  and a simulator  $\mathcal{S}$  then we get a sequence of distributions

$$\text{ideal} := \{\text{ideal}_{f, \mathcal{S}}(k, \vec{x}, (r_0, r_f))\}_{k \in \mathbb{N}, \vec{x} \in (\{0,1\}^*)^n \in \{0,1\}^*}.$$

The intuitive statement:

„whatever the adversary can achieve attacking  $\Pi$  in the real-life he can be achieve in the ideal-process (computing  $f$ )”

can be now translated to the following:

$\forall \mathcal{A} \exists \mathcal{S}$  with a „similar” computing power such that  
 $\text{exec}_{\Pi, \mathcal{A}}$  „is close to”  $\text{ideal}_{f, \mathcal{S}}$

# Probability ensembles

---

**Def.** An infinite sequence

$$\{X(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$$

(where each  $X(k, a)$  is a probability distribution) is called a **distribution ensemble**.

# What does „close to” mean?

---

Let  $X = \{X(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  and  $Y = \{Y(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  be probability ensembles.

# What does „close to” mean?

---

Let  $X = \{X(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  and  $Y = \{Y(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  be probability ensembles.

- If for  $k$  sufficiently large and every  $a$  the distributions  $X(k, a)$  and  $Y(k, a)$  are identical then we say that  $X$  and  $Y$  are **identically distributed** and write  $X \stackrel{d}{=} Y$ .

# What does „close to” mean?

---

Let  $X = \{X(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  and  $Y = \{Y(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  be probability ensembles.

- If for  $k$  sufficiently large and every  $a$  the distributions  $X(k, a)$  and  $Y(k, a)$  are identical then we say that  $X$  and  $Y$  are **identically distributed** and write  $X \stackrel{d}{=} Y$ .
- If  $\max_a (\delta(X(k, a); Y(k, a)))$  is negligible(in  $k$ ) then we say that  $X$  and  $Y$  are **statistically indistinguishable** and write  $X \stackrel{s}{\approx} Y$ .

# What does „close to” mean?

---

Let  $X = \{X(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  and  $Y = \{Y(k, a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$  be probability ensembles.

- If for  $k$  sufficiently large and every  $a$  the distributions  $X(k, a)$  and  $Y(k, a)$  are identical then we say that  $X$  and  $Y$  are **identically distributed** and write  $X \stackrel{d}{=} Y$ .
- If  $\max_a (\delta(X(k, a); Y(k, a)))$  is negligible(in  $k$ ) then we say that  $X$  and  $Y$  are **statistically indistinguishable** and write  $X \stackrel{s}{\approx} Y$ .
- If any polynomial-time algorithm cannot distinguish between  $X(k, a)$  and  $Y(k, a)$  with a probability non-negligibly (in  $k$ ) better than  $\frac{1}{2}$  then we say that  $X$  and  $Y$  are **computationally indistinguishable** and write  $X \stackrel{c}{\approx} Y$ .

# The power of the adversary

---

Another problem is to specify the computational power of the adversary and the simulator.

# The power of the adversary

---

Another problem is to specify the computational power of the adversary and the simulator. We will consider the following options

**IT -security** The adversary and the simulator have infinite computing power.

# The power of the adversary

---

Another problem is to specify the computational power of the adversary and the simulator. We will consider the following options

**IT -security** The adversary and the simulator have infinite computing power.

**computational security** The expected running time adversary and the simulator is polynomial (in the security parameter  $k$ )

# The security definition

---

Let  $\mathcal{F}$  be an adversary structure.

**Def.** We will say that a protocol  $\Pi$  **IT-securely** evaluates  $f$  (with **perfect emulation**) against any adversary who can **adaptively** corrupt sets of players from  $\mathcal{F}$  if

# The security definition

---

Let  $\mathcal{F}$  be an adversary structure.

**Def.** We will say that a protocol  $\Pi$  **IT-securely** evaluates  $f$  (with **perfect emulation**) against any adversary who can **adaptively** corrupt sets of players from  $\mathcal{F}$  if for any such adversary  $\mathcal{A}$

# The security definition

---

Let  $\mathcal{F}$  be an adversary structure.

**Def.** We will say that a protocol  $\Pi$  **IT-securely** evaluates  $f$  (with **perfect emulation**) against any adversary who can **adaptively** corrupt sets of players from  $\mathcal{F}$  if for any such adversary  $\mathcal{A}$  there exists a simulator  $\mathcal{S}$  such that  $\text{exec}_{\Pi, \mathcal{A}} \stackrel{d}{=} \text{ideal}_{f, \mathcal{S}}$ .

# The security definition

---

Let  $\mathcal{F}$  be an adversary structure.

**Def.** We will say that a protocol  $\Pi$  **IT-securely** evaluates  $f$  (with **perfect emulation**) against any adversary who can **adaptively** corrupt sets of players from  $\mathcal{F}$  if for any such adversary  $\mathcal{A}$  there exists a simulator  $\mathcal{S}$  such that  $\text{exec}_{\Pi, \mathcal{A}} \stackrel{d}{=} \text{ideal}_{f, \mathcal{S}}$ .

„ $\stackrel{\mathcal{S}}{\approx}$ ”  $\rightsquigarrow$  „**statistical emulation**”

„ $\stackrel{\mathcal{C}}{\approx}$ ”  $\rightsquigarrow$  „**computational emulation**”

# The security definition

---

Let  $\mathcal{F}$  be an adversary structure.

**Def.** We will say that a protocol  $\Pi$  **IT-securely** evaluates  $f$  (with **perfect emulation**) against any adversary who can **adaptively** corrupt sets of players from  $\mathcal{F}$  if for any such adversary  $\mathcal{A}$  there exists a simulator  $\mathcal{S}$  such that  $\text{exec}_{\Pi, \mathcal{A}} \stackrel{d}{=} \text{ideal}_{f, \mathcal{S}}$ .

„ $\stackrel{\mathcal{S}}{\approx}$ ”  $\rightsquigarrow$  „**statistical emulation**”

„ $\stackrel{\mathcal{C}}{\approx}$ ”  $\rightsquigarrow$  „**computational emulation**”

If we require that  $\mathcal{S}$  and  $\mathcal{A}$  are poly-time then „IT-securely” is replaced with „computationally-securely”

# The non-adaptive adversary

---

An adversary is **non-adaptive** if he has to choose the set of the corrupted players before the protocol starts.

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ✓
3. Formal security definitions ✓
4. Some equivalence and non-equivalence proofs ←

# The picture

---

**Question:** When is the **non**-adaptive security equivalent to the adaptive security?

# The picture

---

**Question:** When is the **non**-adaptive security equivalent to the adaptive security?

# The picture

---

**Question:** When is the **non**-adaptive security equivalent to the adaptive security?

- In the **passive** case:

# The picture

---

**Question:** When is the **non**-adaptive security equivalent to the adaptive security?

- In the **passive** case:

# of parties		remark
large	<b>NOT EQUIVALENT</b>	except of perfect emul. + IT-sec.
small	<b>EQUIVALENT</b>	

# The picture

---

**Question:** When is the **non**-adaptive security equivalent to the adaptive security?

- In the **passive** case:

# of parties		remark
large	<b>NOT EQUIVALENT</b>	except of perfect emul. + IT-sec.
small	<b>EQUIVALENT</b>	

- In the **active** case:

# The picture

**Question:** When is the **non**-adaptive security equivalent to the adaptive security?

- In the **passive** case:

# of parties		remark
large	<b>NOT EQUIVALENT</b>	except of perfect emul. + IT-sec.
small	<b>EQUIVALENT</b>	

- In the **active** case:

# of parties	
$\geq 3$	<b>NOT EQUIVALENT</b>
$< 3$	<b>IT DEPENDS</b>

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ✓
3. Formal security definitions ✓
4. Some equivalence and non-equivalence proofs
  - (a) passive case ←
  - (b) active case

# The non-equivalence in the passive case

---

We are going to examine the **passive case**.

# The non-equivalence in the passive case

---

We are going to examine the **passive case**.

First we show that **if the number of parties is large** then there is **no equivalence** in the case of **any type of security** and computational and statistical emulations.

(here „large” means at least linear in  $k$ )

# The non-equivalence in the passive case

---

We are going to examine the **passive case**.

First we show that **if the number of parties is large** then there is **no equivalence** in the case of **any type of security** and computational and statistical emulations.

(here „large” means at least linear in  $k$ )

For this we show an example of a function  $f$  and a protocol  $\Pi$  that (in the passive case)

IT-securely evaluates  $f$  against a **non**-adaptive adversary (statistical or computational emulation).

# The non-equivalence in the passive case

---

We are going to examine the **passive case**.

First we show that **if the number of parties is large** then there is **no equivalence** in the case of **any type of security** and computational and statistical emulations.

(here „large” means at least linear in  $k$ )

For this we show an example of a function  $f$  and a protocol  $\Pi$  that (in the passive case)

IT-securely evaluates  $f$  against a **non**-adaptive adversary (statistical or computational emulation).

but

does **not** securely evaluate  $f$  against an **adaptive** adversary even according to the weakest definition possible (computational security and emulation)

# Example 1 — specification

---

There are  $2n + 2$  players: the dealer  $D$  and

# Example 1 — specification

---

There are  $2n + 2$  players: the dealer  $D$  and  $P_0, \dots, P_{2n}$ .

# Example 1 — specification

---

There are  $2n + 2$  players: the dealer  $D$  and  $P_0, \dots, P_{2n}$ .

The functions  $f$  is specified as follows:

- The dealer  $D$  takes some bit value  $s \in \{0, 1\}$ . Other players take no input.
- Nobody outputs anything.

# Example 1 — specification

---

There are  $2n + 2$  players: the dealer  $D$  and  $P_0, \dots, P_{2n}$ .

The functions  $f$  is specified as follows:

- The dealer  $D$  takes some bit value  $s \in \{0, 1\}$ . Other players take no input.
- Nobody outputs anything.

Suppose the adversary cannot corrupt  $D$  but can corrupt  $P_0$  and at most  $n$  players from the set  $P_1, \dots, P_{2n}$ ,

# Example 1 — specification

---

There are  $2n + 2$  players: the dealer  $D$  and  $P_0, \dots, P_{2n}$ .

The functions  $f$  is specified as follows:

- The dealer  $D$  takes some bit value  $s \in \{0, 1\}$ . Other players take no input.
- Nobody outputs anything.

Suppose the adversary cannot corrupt  $D$  but can corrupt  $P_0$  and at most  $n$  players from the set  $P_1, \dots, P_{2n}$ , i.e. the adversary structure is a family

$$\mathcal{F} := \{X : D \notin X \wedge X \cap \{P_1, \dots, P_{2n}\} \leq n\}$$

# Example 1 — the implementation

---

Consider the following (stupid) implementation:

1. The dealer  $D$  chooses a random subset

$$R = \{R_1, \dots, R_n\} \subset \{1, \dots, 2n\}.$$

# Example 1 — the implementation

---

Consider the following (stupid) implementation:

1. The dealer  $D$  chooses a random subset

$$R = \{R_1, \dots, R_n\} \subset \{1, \dots, 2n\}.$$

He also chooses  $n$  random bits  $x_1, \dots, x_n$  such that

$$x_1 \oplus \dots \oplus x_n = s$$

# Example 1 — the implementation

---

Consider the following (stupid) implementation:

1. The dealer  $D$  chooses a random subset

$$R = \{R_1, \dots, R_n\} \subset \{1, \dots, 2n\}.$$

He also chooses  $n$  random bits  $x_1, \dots, x_n$  such that

$$x_1 \oplus \dots \oplus x_n = s$$

2. The dealer sends the description of  $R$  to  $P_0$ .

# Example 1 — the implementation

---

Consider the following (stupid) implementation:

1. The dealer  $D$  chooses a random subset

$$R = \{R_1, \dots, R_n\} \subset \{1, \dots, 2n\}.$$

He also chooses  $n$  random bits  $x_1, \dots, x_n$  such that

$$x_1 \oplus \dots \oplus x_n = s$$

2. The dealer sends the description of  $R$  to  $P_0$ .

He also sends each  $x_i$  to  $P_i$ .

# Non adaptive security of Example 1

---

We are going to argue that this implementation securely evaluates  $f$  against a **non**-adaptive adversary.

# Non adaptive security of Example 1

---

We are going to argue that this implementation securely evaluates  $f$  against a **non**-adaptive adversary.

Let  $\mathcal{A}$  be a (non-adaptive) adversary. We are going to construct a simulator  $\mathcal{S}$  for it.

# Non adaptive security of Example 1

---

We are going to argue that this implementation securely evaluates  $f$  against a **non**-adaptive adversary.

Let  $\mathcal{A}$  be a (non-adaptive) adversary. We are going to construct a simulator  $\mathcal{S}$  for it.

Simulator  $\mathcal{S}$  works as follows:

1. He starts  $\mathcal{A}$ . He starts simulating it (with some random input).

# Non adaptive security of Example 1

---

We are going to argue that this implementation securely evaluates  $f$  against a **non**-adaptive adversary.

Let  $\mathcal{A}$  be a (non-adaptive) adversary. We are going to construct a simulator  $\mathcal{S}$  for it.

Simulator  $\mathcal{S}$  works as follows:

1. He starts  $\mathcal{A}$ . He starts simulating it (with some random input). He also starts all the parties of the protocol (with some random inputs). Clearly he doesn't know Dealer's input  $s \in \{0, 1\}$ , so he chooses it randomly.

# Non adaptive security of Example 1

---

We are going to argue that this implementation securely evaluates  $f$  against a **non**-adaptive adversary.

Let  $\mathcal{A}$  be a (non-adaptive) adversary. We are going to construct a simulator  $\mathcal{S}$  for it.

Simulator  $\mathcal{S}$  works as follows:

1. He starts  $\mathcal{A}$ . He starts simulating it (with some random input). He also starts all the parties of the protocol (with some random inputs). Clearly he doesn't know Dealer's input  $s \in \{0, 1\}$ , so he chooses it randomly.
2. He simulates the execution of  $\mathcal{A}$  against the protocol.

# Non adaptive security of Example 1

---

We are going to argue that this implementation securely evaluates  $f$  against a **non**-adaptive adversary.

Let  $\mathcal{A}$  be a (non-adaptive) adversary. We are going to construct a simulator  $\mathcal{S}$  for it.

Simulator  $\mathcal{S}$  works as follows:

1. He starts  $\mathcal{A}$ . He starts simulating it (with some random input). He also starts all the parties of the protocol (with some random inputs). Clearly he doesn't know Dealer's input  $s \in \{0, 1\}$ , so he chooses it randomly.
2. He simulates the execution of  $\mathcal{A}$  against the protocol.
3. He outputs the output of  $\mathcal{A}$

# The correctness of the simulation

---

Let  $\text{Error}_{\text{exec}}$  denote the event that in the real-life execution the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

# The correctness of the simulation

---

Let  $\text{Error}_{\text{exec}}$  denote the event that in the real-life execution the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

Let  $\text{Error}_{\text{ideal}}$  denote the event that in the ideal process the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

# The correctness of the simulation

---

Let  $\text{Error}_{\text{exec}}$  denote the event that in the real-life execution the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

Let  $\text{Error}_{\text{ideal}}$  denote the event that in the ideal process the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

It is easy to see that the conditional distribution of

$$P_{\text{exec}_{f,S}}(k,s) | \text{Error}_{\text{exec}}$$

is identical to

$$P_{\text{ideal}_{f,S}}(k,s) | \text{Error}_{\text{ideal}}$$

# The correctness of the simulation

---

Let  $\text{Error}_{\text{exec}}$  denote the event that in the real-life execution the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

Let  $\text{Error}_{\text{ideal}}$  denote the event that in the ideal process the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

It is easy to see that the conditional distribution of

$$P_{\text{exec}_{f,S}(k,s) | \text{Error}_{\text{exec}}}$$

is identical to

$$P_{\text{ideal}_{f,S}(k,s) | \text{Error}_{\text{ideal}}}$$

Observe that  $P[\text{Error}_{\text{exec}}] = P[\text{Error}_{\text{ideal}}] = \binom{2n}{n}^{-1} \leq 2^{-n}$ .

# The correctness of the simulation

---

Let  $\text{Error}_{\text{exec}}$  denote the event that in the real-life execution the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

Let  $\text{Error}_{\text{ideal}}$  denote the event that in the ideal process the adversary corrupted exactly the set  $P_{R_1}, \dots, P_{R_n}$  chosen by  $D$ .

It is easy to see that the conditional distribution of

$$P_{\text{exec}_{f,S}}(k,s) | \text{Error}_{\text{exec}}$$

is identical to

$$P_{\text{ideal}_{f,S}}(k,s) | \text{Error}_{\text{ideal}}$$

Observe that  $P[\text{Error}_{\text{exec}}] = P[\text{Error}_{\text{ideal}}] = \binom{2n}{n}^{-1} \leq 2^{-n}$ .

This value is negligible since we assumed that  $n$  is at least linear in  $k$ .

Therefore  $\text{exec}_{f,S} \stackrel{s}{\approx} \text{ideal}_{f,S}$  and hence  $\text{exec}_{f,S} \stackrel{c}{\approx} \text{ideal}_{f,S}$

# Example 1 — adaptive insecurity

---

Adaptive insecurity of this protocol is straightforward.

# Example 1 — adaptive insecurity

---

Adaptive insecurity of this protocol is straightforward.

An **adaptive** adversary  $\mathcal{A}$  can perform the following attack:

1. He corrupts  $P_0$ .

This way he learns  $R = \{R_1, \dots, R_n\}$ .

# Example 1 — adaptive insecurity

---

Adaptive insecurity of this protocol is straightforward.

An **adaptive** adversary  $\mathcal{A}$  can perform the following attack:

1. He corrupts  $P_0$ .

This way he learns  $R = \{R_1, \dots, R_n\}$ .

2. He corrupts the players  $P_{R_1}, \dots, P_{R_n}$ .

This way he learns  $x_1, \dots, x_n$ . He can now compute  $s = x_1 \oplus \dots \oplus x_n$ .

# Example 1 — adaptive insecurity

---

Adaptive insecurity of this protocol is straightforward.

An **adaptive** adversary  $\mathcal{A}$  can perform the following attack:

1. He corrupts  $P_0$ .

This way he learns  $R = \{R_1, \dots, R_n\}$ .

2. He corrupts the players  $P_{R_1}, \dots, P_{R_n}$ .

This way he learns  $x_1, \dots, x_n$ . He can now compute  $s = x_1 \oplus \dots \oplus x_n$ .

So, with probability 1 the adversary can learn  $s$ .

# Example 1 — adaptive insecurity

---

Adaptive insecurity of this protocol is straightforward.

An **adaptive** adversary  $\mathcal{A}$  can perform the following attack:

1. He corrupts  $P_0$ .

This way he learns  $R = \{R_1, \dots, R_n\}$ .

2. He corrupts the players  $P_{R_1}, \dots, P_{R_n}$ .

This way he learns  $x_1, \dots, x_n$ . He can now compute  $s = x_1 \oplus \dots \oplus x_n$ .

So, with probability 1 the adversary can learn  $s$ .

Any simulator cannot learn  $s$  with probability better than guessing. So for any simulator the distributions  $\text{exec}_{f,S}$  and  $\text{ideal}_{f,S}$  are far apart!

# Example 1 — adaptive insecurity

---

Adaptive insecurity of this protocol is straightforward.

An **adaptive** adversary  $\mathcal{A}$  can perform the following attack:

1. He corrupts  $P_0$ .

This way he learns  $R = \{R_1, \dots, R_n\}$ .

2. He corrupts the players  $P_{R_1}, \dots, P_{R_n}$ .

This way he learns  $x_1, \dots, x_n$ . He can now compute  $s = x_1 \oplus \dots \oplus x_n$ .

So, with probability 1 the adversary can learn  $s$ .

Any simulator cannot learn  $s$  with probability better than guessing. So for any simulator the distributions  $\text{exec}_{f,S}$  and  $\text{ideal}_{f,S}$  are far apart!

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ✓
3. Formal security definitions ✓
4. Some equivalence and non-equivalence proofs
  - (a) passive case
    - i. large number of parties: imperfect emulation ✓
    - ii. large number of parties: perfect emulation ←
    - iii. small number of parties
  - (b) active case

# The perfect emulation

---

What remains is the case of the perfect emulation.

# The perfect emulation

---

What remains is the case of the **perfect emulation**.

We will now show that in the case of **computational** security we also have a separation of the non-adaptive and the adaptive case.

# The perfect emulation

---

What remains is the case of the **perfect emulation**.

We will now show that in the case of **computational** security we also have a separation of the non-adaptive and the adaptive case.

To construct the example we will need a **perfectly hiding bit commitment scheme**, i.e. a commitment scheme in which (until the opening phase) the secret is protected unconditionally.

# A perfectly hiding commitment scheme

---

An example of such commitment scheme is as follows:

$p$  — prime,  $g$  — a generator of  $Z_p^*$ ,

$pk$  — an element of  $Z_p^*$  such that  $\log_g pk$  is unknown (we will call it a **public key**)

# A perfectly hiding commitment scheme

---

An example of such commitment scheme is as follows:

$p$  — prime,  $g$  — a generator of  $Z_p^*$ ,

$pk$  — an element of  $Z_p^*$  such that  $\log_g pk$  is unknown (we will call it a **public key**)

$$\text{commit}(b, pk, r) := \begin{cases} g^x & \text{if } b = 0 \\ pk \cdot g^x & \text{if } b = 1 \end{cases}$$

A commitment to  $b$  is  $\text{commit}(b, pk, r)$  where  $r \in Z_{p-1}$  is random).

# A perfectly hiding commitment scheme

---

An example of such commitment scheme is as follows:

$p$  — prime,  $g$  — a generator of  $Z_p^*$ ,

$pk$  — an element of  $Z_p^*$  such that  $\log_g pk$  is unknown (we will call it a **public key**)

$$\text{commit}(b, pk, r) := \begin{cases} g^x & \text{if } b = 0 \\ pk \cdot g^x & \text{if } b = 1 \end{cases}$$

A commitment to  $b$  is  $\text{commit}(b, pk, r)$  where  $r \in Z_{p-1}$  is random).

To **open** the commitment the committer publishes  $r$ .

# A perfectly hiding commitment scheme

---

An example of such commitment scheme is as follows:

$p$  — prime,  $g$  — a generator of  $Z_p^*$ ,

$pk$  — an element of  $Z_p^*$  such that  $\log_g pk$  is unknown (we will call it a **public key**)

$$\text{commit}(b, pk, r) := \begin{cases} g^x & \text{if } b = 0 \\ pk \cdot g^x & \text{if } b = 1 \end{cases}$$

A commitment to  $b$  is  $\text{commit}(b, pk, r)$  where  $r \in Z_{p-1}$  is random).

To **open** the commitment the committer publishes  $r$ .

To be able to open both  $0$  and  $1$  the cheating committer would need to know  $r_0$  and  $r_1$  such that  $g^{r_0} = pk g^{r_1}$ . So, he would be able to compute  $\log_g pk = r_0 - r_1$ .

# A perfectly hiding commitment scheme

---

An example of such commitment scheme is as follows:

$p$  — prime,  $g$  — a generator of  $Z_p^*$ ,

$pk$  — an element of  $Z_p^*$  such that  $\log_g pk$  is unknown (we will call it a **public key**)

$$\text{commit}(b, pk, r) := \begin{cases} g^x & \text{if } b = 0 \\ pk \cdot g^x & \text{if } b = 1 \end{cases}$$

A commitment to  $b$  is  $\text{commit}(b, pk, r)$  where  $r \in Z_{p-1}$  is random).

To **open** the commitment the committer publishes  $r$ .

To be able to open both  $0$  and  $1$  the cheating committer would need to know  $r_0$  and  $r_1$  such that  $g^{r_0} = pk g^{r_1}$ . So, he would be able to compute  $\log_g pk = r_0 - r_1$ .

On the other hand: the distribution of  $\text{commit}(0)$  is identical to  $\text{commit}(1)$

# Example 2 — specification

---

There are  $2n + 1$  players:

- dealer  $D$  — whose input is

# Example 2 — specification

---

There are  $2n + 1$  players:

- dealer  $D$  — whose input is
  - a function  $h$  from a family of collision-resistant hash functions

# Example 2 — specification

---

There are  $2n + 1$  players:

- dealer  $D$  — whose input is
  - a function  $h$  from a family of collision-resistant hash functions
  - a public key  $pk$  for a perfectly-hiding bit commitment scheme

# Example 2 — specification

---

There are  $2n + 1$  players:

- dealer  $D$  — whose input is
  - a function  $h$  from a family of collision-resistant hash functions
  - a public key  $pk$  for a perfectly-hiding bit commitment scheme
- players  $P_1, \dots, P_{2n}$  — each  $P_i$  taking an input bit  $b_i$

# Example 2 — specification

---

There are  $2n + 1$  players:

- dealer  $D$  — whose input is
  - a function  $h$  from a family of collision-resistant hash functions
  - a public key  $pk$  for a perfectly-hiding bit commitment scheme
- players  $P_1, \dots, P_{2n}$  — each  $P_i$  taking an input bit  $b_i$

The output of each party is  $(h, pk)$ .

# Example 2 — specification

---

There are  $2n + 1$  players:

- dealer  $D$  — whose input is
  - a function  $h$  from a family of collision-resistant hash functions
  - a public key  $pk$  for a perfectly-hiding bit commitment scheme
- players  $P_1, \dots, P_{2n}$  — each  $P_i$  taking an input bit  $b_i$

The output of each party is  $(h, pk)$ .

The adversary is allowed to corrupt the dealer and  $n$  players out of the set  $\{P_1, \dots, P_{2n}\}$

# Example 2 — implementation

---

Consider the following implementation:

1. The dealer  $D$  sends  $(h, pk)$  to all parties

# Example 2 — implementation

---

Consider the following implementation:

1. The dealer  $D$  sends  $(h, pk)$  to all parties
2. Each  $P_i$  selects a random  $r_i$  and sends to all parties  $c_i := \text{commit}(pk, b_i, r_i)$ .

# Example 2 — implementation

---

Consider the following implementation:

1. The dealer  $D$  sends  $(h, pk)$  to all parties
2. Each  $P_i$  selects a random  $r_i$  and sends to all parties  $c_i := \text{commit}(pk, b_i, r_i)$ .
3. All parties output  $(h, pk)$

# Example 2 — implementation

---

Consider the following implementation:

1. The dealer  $D$  sends  $(h, pk)$  to all parties
2. Each  $P_i$  selects a random  $r_i$  and sends to all parties  $c_i := \text{commit}(pk, b_i, r_i)$ .
3. All parties output  $(h, pk)$

It easy to show that this is **non**-adaptively computationally secure (with perfect emulation).

# Example 2 — implementation

---

Consider the following implementation:

1. The dealer  $D$  sends  $(h, pk)$  to all parties
2. Each  $P_i$  selects a random  $r_i$  and sends to all parties  $c_i := \text{commit}(pk, b_i, r_i)$ .
3. All parties output  $(h, pk)$

It easy to show that this is **non**-adaptively computationally secure (with perfect emulation).

But is it also **adaptively** secure?

# Example 2 — implementation

---

Consider the following implementation:

1. The dealer  $D$  sends  $(h, pk)$  to all parties
2. Each  $P_i$  selects a random  $r_i$  and sends to all parties  $c_i := \text{commit}(pk, b_i, r_i)$ .
3. All parties output  $(h, pk)$

It easy to show that this is **non**-adaptively computationally secure (with perfect emulation).

But is it also **adaptively** secure?

Answer: No!

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning
2. Listen to all  $c_i$ 's.

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning
2. Listen to all  $c_i$ 's.
3. Compute  $h(c_1, \dots, c_{2n})$ . Interpret the result as a description of a set  $R := \{R_1, \dots, R_n\} \in \{1, \dots, 2n\}$ .

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning
2. Listen to all  $c_i$ 's.
3. Compute  $h(c_1, \dots, c_{2n})$ . Interpret the result as a description of a set  $R := \{R_1, \dots, R_n\} \in \{1, \dots, 2n\}$ .
4. Corrupt the players in the set  $\mathcal{C} := \{P_{R_1}, \dots, P_{R_n}\}$ .
5. Output everything that you saw (including  $c_1, \dots, c_{2n}$ )

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning
2. Listen to all  $c_i$ 's.
3. Compute  $h(c_1, \dots, c_{2n})$ . Interpret the result as a description of a set  $R := \{R_1, \dots, R_n\} \in \{1, \dots, 2n\}$ .
4. Corrupt the players in the set  $\mathcal{C} := \{P_{R_1}, \dots, P_{R_n}\}$ .
5. Output everything that you saw (including  $c_1, \dots, c_{2n}$ )

Suppose we've got a simulator  $\mathcal{S}$  for  $\mathcal{A}$ . We construct an algorithm that takes as input  $(h, pk)$  and either

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning
2. Listen to all  $c_i$ 's.
3. Compute  $h(c_1, \dots, c_{2n})$ . Interpret the result as a description of a set  $R := \{R_1, \dots, R_n\} \in \{1, \dots, 2n\}$ .
4. Corrupt the players in the set  $\mathcal{C} := \{P_{R_1}, \dots, P_{R_n}\}$ .
5. Output everything that you saw (including  $c_1, \dots, c_{2n}$ )

Suppose we've got a simulator  $\mathcal{S}$  for  $\mathcal{A}$ . We construct an algorithm that takes as input  $(h, pk)$  and either

- finds a collision in  $h$ , or

# An attack on Example 2

---

Consider the following adversary  $\mathcal{A}$ :

1. Corrupt  $D$  at the beginning
2. Listen to all  $c_i$ 's.
3. Compute  $h(c_1, \dots, c_{2n})$ . Interpret the result as a description of a set  $R := \{R_1, \dots, R_n\} \in \{1, \dots, 2n\}$ .
4. Corrupt the players in the set  $\mathcal{C} := \{P_{R_1}, \dots, P_{R_n}\}$ .
5. Output everything that you saw (including  $c_1, \dots, c_{2n}$ )

Suppose we've got a simulator  $\mathcal{S}$  for  $\mathcal{A}$ . We construct an algorithm that takes as input  $(h, pk)$  and either

- finds a collision in  $h$ , or
- breaks the commitment scheme (with the public key  $pk$ ).

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .
3. When  $\mathcal{S}$  corrupts a some  $P_i$  give it  $0$  as the input of  $P_i$ .

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .
3. When  $\mathcal{S}$  corrupts a some  $P_i$  give it 0 as the input of  $P_i$ .
4. Let  $\mathcal{C}_0$  be the set of the corrupted players (Observe  $|\mathcal{C}_0| = n$ ). Let  $P_j$  be the last player corrupted. Let  $(c_1, \dots, c_{2n})$  be the output of  $\mathcal{S}$ .

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .
3. When  $\mathcal{S}$  corrupts a some  $P_i$  give it 0 as the input of  $P_i$ .
4. Let  $\mathcal{C}_0$  be the set of the corrupted players (Observe  $|\mathcal{C}_0| = n$ ). Let  $P_j$  be the last player corrupted. Let  $(c_1, \dots, c_{2n})$  be the output of  $\mathcal{S}$ .
5. Rewind  $\mathcal{S}$  to its state just after Step 1. Run it again. The only difference is that now give as the input of  $P_j$  value 1. Let  $(c'_1, \dots, c'_{2n})$  be the output of  $\mathcal{S}$ .

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .
3. When  $\mathcal{S}$  corrupts a some  $P_i$  give it 0 as the input of  $P_i$ .
4. Let  $\mathcal{C}_0$  be the set of the corrupted players (Observe  $|\mathcal{C}_0| = n$ ). Let  $P_j$  be the last player corrupted. Let  $(c_1, \dots, c_{2n})$  be the output of  $\mathcal{S}$ .
5. Rewind  $\mathcal{S}$  to its state just after Step 1. Run it again. The only difference is that now give as the input of  $P_j$  value 1. Let  $(c'_1, \dots, c'_{2n})$  be the output of  $\mathcal{S}$ .

**Observe** The set of corrupted players in both runs is the same. But the inputs to the player  $P_j$  are different. Thus either:

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .
3. When  $\mathcal{S}$  corrupts a some  $P_i$  give it 0 as the input of  $P_i$ .
4. Let  $\mathcal{C}_0$  be the set of the corrupted players (Observe  $|\mathcal{C}_0| = n$ ). Let  $P_j$  be the last player corrupted. Let  $(c_1, \dots, c_{2n})$  be the output of  $\mathcal{S}$ .
5. Rewind  $\mathcal{S}$  to its state just after Step 1. Run it again. The only difference is that now give as the input of  $P_j$  value 1. Let  $(c'_1, \dots, c'_{2n})$  be the output of  $\mathcal{S}$ .

**Observe** The set of corrupted players in both runs is the same. But the inputs to the player  $P_j$  are different. Thus either:

- $c_j = c'_j$  — in this case we have broken the commitment scheme.

# The algorithm

---

1. Run  $\mathcal{S}$  with some random input.
2. When  $\mathcal{S}$  corrupts  $D$  give it  $(h, pk)$  as the input of  $D$ .
3. When  $\mathcal{S}$  corrupts a some  $P_i$  give it 0 as the input of  $P_i$ .
4. Let  $\mathcal{C}_0$  be the set of the corrupted players (Observe  $|\mathcal{C}_0| = n$ ). Let  $P_j$  be the last player corrupted. Let  $(c_1, \dots, c_{2n})$  be the output of  $\mathcal{S}$ .
5. Rewind  $\mathcal{S}$  to its state just after Step 1. Run it again. The only difference is that now give as the input of  $P_j$  value 1. Let  $(c'_1, \dots, c'_{2n})$  be the output of  $\mathcal{S}$ .

**Observe** The set of corrupted players in both runs is the same. But the inputs to the player  $P_j$  are different. Thus either:

- $c_j = c'_j$  — in this case we have broken the commitment scheme.
- $c_j \neq c'_j$  — in this case we have found a collision in  $h$ .

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ✓
3. Formal security definitions ✓
4. Some equivalence and non-equivalence proofs
  - (a) passive case
    - i. large number of parties: imperfect emulation ✓
    - ii. large number of parties: perfect emulation ✓
    - iii. **small number of parties** ←
  - (b) active case

# Small number of parties – equivalence

---

We are going to show that the non-adaptive security is equivalent to the adaptive security when the number of the parties is logarithmic in the security parameter  $k$ .

# Small number of parties – equivalence

---

We are going to show that the non-adaptive security is equivalent to the adaptive security when the number of the parties is logarithmic in the security parameter  $k$ .

We are first going to examine the case of the **perfect emulation**.

# Small number of parties – equivalence

---

We are going to show that the non-adaptive security is equivalent to the adaptive security when the number of the parties is logarithmic in the security parameter  $k$ .

We are first going to examine the case of the **perfect emulation**.

So, take some protocol  $\Pi$  and a function  $f$  and assume that:

For every **non-adaptive** adversary  $\mathcal{A}^{\text{non-adap}}$  there exists a simulator  $\mathcal{S}^{\text{non-adap}}$  such that

$$\text{exec}_{\Pi, \mathcal{A}^{\text{non-adap}}} \stackrel{d}{=} \text{ideal}_{f, \mathcal{S}^{\text{non-adap}}} \quad (\star)$$

# Small number of parties – equivalence

---

We are going to show that the non-adaptive security is equivalent to the adaptive security when the number of the parties is logarithmic in the security parameter  $k$ .

We are first going to examine the case of the **perfect emulation**.

So, take some protocol  $\Pi$  and a function  $f$  and assume that:

For every **non-adaptive** adversary  $\mathcal{A}^{\text{non-adap}}$  there exists a simulator  $\mathcal{S}^{\text{non-adap}}$  such that

$$\text{exec}_{\Pi, \mathcal{A}^{\text{non-adap}}} \stackrel{d}{=} \text{ideal}_{f, \mathcal{S}^{\text{non-adap}}} \quad (\star)$$

We have to show that the same holds for every **adaptive** adversary (i.e. we will show that  $\forall \mathcal{A}^{\text{adapt}} \exists \mathcal{S}^{\text{adapt}}$  such that  $(\star)$  holds).

# Small number of parties – equivalence

---

We are going to show that the non-adaptive security is equivalent to the adaptive security when the number of the parties is logarithmic in the security parameter  $k$ .

We are first going to examine the case of the **perfect emulation**.

So, take some protocol  $\Pi$  and a function  $f$  and assume that:

For every **non-adaptive** adversary  $\mathcal{A}^{\text{non-adap}}$  there exists a simulator  $\mathcal{S}^{\text{non-adap}}$  such that

$$\text{exec}_{\Pi, \mathcal{A}^{\text{non-adap}}} \stackrel{d}{=} \text{ideal}_{f, \mathcal{S}^{\text{non-adap}}} \quad (\star)$$

We have to show that the same holds for every **adaptive** adversary (i.e. we will show that  $\forall \mathcal{A}^{\text{adapt}} \exists \mathcal{S}^{\text{adapt}}$  such that  $(\star)$  holds).

**Moreover:**  $\mathcal{S}^{\text{non-adap}}$  is poly-time in the running time of  $\mathcal{A}^{\text{non-adap}}$  then  $\mathcal{S}^{\text{adapt}}$  will be poly-time in the running time of  $\mathcal{A}^{\text{adapt}}$ .

# Assumptions

---

We will assume that the in the adversary corrupts the players **after** the execution of the protocol.

# Assumptions

---

We will assume that the adversary corrupts the players **after** the execution of the protocol.

We will assume that the simulator corrupts the players in the **second corruption stage**.

# Assumptions

---

We will assume that the adversary corrupts the players **after** the execution of the protocol.

We will assume that the simulator corrupts the players in the **second corruption stage**.

In the passive case this can be done without loss of generality.

# Assumptions

---

We will assume that the adversary corrupts the players **after** the execution of the protocol.

We will assume that the simulator corrupts the players in the **second corruption stage**.

In the passive case this can be done without loss of generality.

We can also assume that the adversary is deterministic.

Let  $\mathcal{B}$  be an adversary structure.

# Analogy: the card game

---

Imagine a set of  $n$  cards and an „adversary structure”  $\mathcal{B} \subseteq 2^{\{1, \dots, n\}}$ .

# Analogy: the card game

---

Imagine a set of  $n$  cards and an „adversary structure”  $\mathcal{B} \subseteq 2^{\{1, \dots, n\}}$ .

Consider two rooms.

# Analogy: the card game

---

Imagine a set of  $n$  cards and an „adversary structure”  $\mathcal{B} \subseteq 2^{\{1, \dots, n\}}$ .  
Consider two rooms.

In the **first room** the **adversary**  $\mathcal{A}$  is playing.

# Analogy: the card game

---

Imagine a set of  $n$  cards and an „adversary structure”  $\mathcal{B} \subseteq 2^{\{1, \dots, n\}}$ .  
Consider two rooms.

In the **first room** the **adversary**  $\mathcal{A}$  is playing.

The faces of the cards are chosen according to some distribution  
 $V = (V_1, \dots, V_n)$ .

# Analogy: the card game

---

Imagine a set of  $n$  cards and an „adversary structure”  $\mathcal{B} \subseteq 2^{\{1, \dots, n\}}$ .  
Consider two rooms.

In the **first room** the **adversary**  $\mathcal{A}$  is playing.

The faces of the cards are chosen according to some distribution  $V = (V_1, \dots, V_n)$ .

The cards are initially covered. The adversary can sequentially uncover the cards according to some **fixed, deterministic strategy**. The set of uncovered cards has to remain in  $\mathcal{B}$ .

# Analogy: the card game

---

Imagine a set of  $n$  cards and an „adversary structure”  $\mathcal{B} \subseteq 2^{\{1, \dots, n\}}$ .  
Consider two rooms.

In the **first room** the **adversary**  $\mathcal{A}$  is playing.

The faces of the cards are chosen according to some distribution  $V = (V_1, \dots, V_n)$ .

The cards are initially covered. The adversary can sequentially uncover the cards according to some **fixed, deterministic strategy**. The set of uncovered cards has to remain in  $\mathcal{B}$ .

At the end  $\mathcal{A}$  outputs the **identity and the contents of the uncovered cards**.

# The second room

---

In the **second room** the **simulator  $\mathcal{S}$**  is playing.

# The second room

---

In the **second room** the **simulator  $\mathcal{S}$**  is playing.

He is given  $n$  covered **blank** cards. He can sequentially uncover them and **fill them with the contents**.

# The second room

---

In the **second room** the **simulator  $\mathcal{S}$**  is playing.

He is given  $n$  covered **blank** cards. He can sequentially uncover them and **fill them with the contents**.

At any moment when the set of uncovered cards is  $b$  he can sample  $\tilde{V}_b$  (which on the set  $b$  is distributed identically to  $V$ ).

# The second room

---

In the **second room** the **simulator**  $\mathcal{S}$  is playing.

He is given  $n$  covered **blank** cards. He can sequentially uncover them and **fill them with the contents**.

At any moment when the set of uncovered cards is  $b$  he can sample  $\tilde{V}_b$  (which on the set  $b$  is distributed identically to  $V$ ).

His goal is to output a string distributed identically to the output of  $\mathcal{A}$ .

# The second room

---

In the **second room** the **simulator**  $\mathcal{S}$  is playing.

He is given  $n$  covered **blank** cards. He can sequentially uncover them and **fill them with the contents**.

At any moment when the set of uncovered cards is  $b$  he can sample  $\tilde{V}_b$  (which on the set  $b$  is distributed identically to  $V$ ).

His goal is to output a string distributed identically to the output of  $\mathcal{A}$ .

This is similar to what we have to prove...

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

We write „ $v \xrightarrow{\mathcal{A}} b$ ” if view  $v$  leads the adversary to corrupt  $b$  **at some point**.

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

We write „ $v \xrightarrow{\mathcal{A}} b$ ” if view  $v$  leads the adversary to corrupt  $b$  **at some point**.

(For example  $v \xrightarrow{\mathcal{A}} \emptyset$  always).

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

We write „ $v \xrightarrow{\mathcal{A}} b$ ” if view  $v$  leads the adversary to corrupt  $b$  **at some point**.  
(For example  $v \xrightarrow{\mathcal{A}} \emptyset$  always).

If  $b' \subseteq b$  then it also makes sense to ask if  $v_{b'} \xrightarrow{\mathcal{A}} b$ . Note that this can be answered efficiently.

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

We write „ $v \xrightarrow{\mathcal{A}} b$ ” if view  $v$  leads the adversary to corrupt  $b$  **at some point**.  
(For example  $v \xrightarrow{\mathcal{A}} \emptyset$  always).

If  $b' \subseteq b$  then it also makes sense to ask if  $v_{b'} \xrightarrow{\mathcal{A}} b$ . Note that this can be answered efficiently.

For a set  $b$  of players let  $\mathcal{A}_b$  be a **non-adaptive** adversary that corrupts set  $b$ .

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

We write „ $v \xrightarrow{\mathcal{A}} b$ ” if view  $v$  leads the adversary to corrupt  $b$  **at some point**.  
(For example  $v \xrightarrow{\mathcal{A}} \emptyset$  always).

If  $b' \subseteq b$  then it also makes sense to ask if  $v_{b'} \xrightarrow{\mathcal{A}} b$ . Note that this can be answered efficiently.

For a set  $b$  of players let  $\mathcal{A}_b$  be a **non-adaptive** adversary that corrupts set  $b$ .

From non-adaptive security we know for every such  $\mathcal{A}_b$  there exists a simulator  $\mathcal{S}_b$ .

# Some notation

---

Let  $\mathcal{A}$  be an adaptive adversary. We will construct a simulator for it.

A **view**  $v = (v_1, \dots, v_n)$  **of an execution** is an  $n$ -tuple such that each  $v_i$  represents a state of a player  $P_i$  after the execution.

For  $b \in \{P_1, \dots, P_n\}$  a **partial view**  $v_b$  is equal to  $v$  restricted to  $b$ .

We write „ $v \xrightarrow{\mathcal{A}} b$ ” if view  $v$  leads the adversary to corrupt  $b$  **at some point**.  
(For example  $v \xrightarrow{\mathcal{A}} \emptyset$  always).

If  $b' \subseteq b$  then it also makes sense to ask if  $v_{b'} \xrightarrow{\mathcal{A}} b$ . Note that this can be answered efficiently.

For a set  $b$  of players let  $\mathcal{A}_b$  be a **non-adaptive** adversary that corrupts set  $b$ .

From non-adaptive security we know for every such  $\mathcal{A}_b$  there exists a simulator  $\mathcal{S}_b$ .

If we know the inputs of the players in  $b$  then we can execute  $\mathcal{S}_b$  and **sample**  $V_b$ .

# The simulator

---

We will use the non-adaptive simulator in a **black-box** manner.

# The simulator

---

We will use the non-adaptive simulator in a **black-box** manner.

Simulator  $\mathcal{S}$  for an adversary  $\mathcal{A}$ :

# The simulator

---

We will use the non-adaptive simulator in a **black-box** manner.

Simulator  $\mathcal{S}$  for an adversary  $\mathcal{A}$ :

1. Set  $b_0 := \emptyset$ .

# The simulator

---

We will use the non-adaptive simulator in a **black-box** manner.

Simulator  $\mathcal{S}$  for an adversary  $\mathcal{A}$ :

1. Set  $b_0 := \emptyset$ .
2. For  $i = 0, 1, 2, \dots$  do

# The security proof

---

We now have to argue that the output of the simulator  $\mathcal{S}$  is distributed identically with the output of the adversary  $\mathcal{A}$ .

# The security proof

---

We now have to argue that the output of the simulator  $\mathcal{S}$  is distributed identically with the output of the adversary  $\mathcal{A}$ . Consider two experiments.

# The security proof

---

We now have to argue that the output of the simulator  $\mathcal{S}$  is distributed identically with the output of the adversary  $\mathcal{A}$ . Consider two experiments.

**The execution of simulator  $\mathcal{S}$ .** Let  $\tilde{B}_i$  be the random variable denoting the set of the players corrupted in the  $i$ th iteration. Let  $\tilde{V}_i$  be the view  $v_i$  sampled in this iteration.

If the simulation terminated before the  $i$ th iteration we let  $\tilde{B}_i = \tilde{V}_i = \perp$ .

# The security proof

---

We now have to argue that the output of the simulator  $\mathcal{S}$  is distributed identically with the output of the adversary  $\mathcal{A}$ . Consider two experiments.

**The execution of simulator  $\mathcal{S}$ .** Let  $\tilde{B}_i$  be the random variable denoting the set of the players corrupted in the  $i$ th iteration. Let  $\tilde{V}_i$  be the view  $v_i$  sampled in this iteration.

If the simulation terminated before the  $i$ th iteration we let  $\tilde{B}_i = \tilde{V}_i = \perp$ .

**The execution of  $\mathcal{A}$  against  $\Pi$ .** Let  $B_i$  and  $V_i$  be the corresponding random variables in this experiment.

# The security proof

---

**Lemma.** For any  $i$  and any set  $b_i \in \mathcal{B}$  the following conditional distributions are identical:

- the distribution of  $\tilde{V}_i$  conditioned on  $\tilde{B}_i = b_i$ .
- the distribution of  $V_i$  conditioned on  $B_i = b_i$ .

# The security proof

---

**Lemma.** For any  $i$  and any set  $b_i \in \mathcal{B}$  the following conditional distributions are identical:

- the distribution of  $\tilde{V}_i$  conditioned on  $\tilde{B}_i = b_i$ .
- the distribution of  $V_i$  conditioned on  $B_i = b_i$ .

**Lemma.** For every  $i$  we have  $\tilde{V}_i \stackrel{d}{=} V_i$

# The security proof

---

**Lemma.** For any  $i$  and any set  $b_i \in \mathcal{B}$  the following conditional distributions are identical:

- the distribution of  $\tilde{V}_i$  conditioned on  $\tilde{B}_i = b_i$ .
- the distribution of  $V_i$  conditioned on  $B_i = b_i$ .

**Lemma.** For every  $i$  we have  $\tilde{V}_i \stackrel{d}{=} V_i$

**Lemma.**  $\mathcal{S}$  perfectly emulates  $\mathcal{A}$ .

# The complexity of the simulator

---

Let  $\tilde{T}_i$  be the number samples in the  $i$ th iteration.

Let  $\text{sup}(\tilde{B}_i)$  denote the support of the random variable  $\tilde{B}_i$  (excluding  $\perp$ ).

For every  $b_i$  we have:

$$P \left[ \tilde{V}_{B_i \rightarrow b_i} \right] = P \left[ \tilde{B}_i = b_i \right]$$

# The complexity of the simulator

---

Let  $\tilde{T}_i$  be the number samples in the  $i$ th iteration.

Let  $\text{sup}(\tilde{B}_i)$  denote the support of the random variable  $\tilde{B}_i$  (excluding  $\perp$ ).

For every  $b_i$  we have:

$$P \left[ \tilde{V}_{B_i \rightarrow b_i} \right] = P \left[ \tilde{B}_i = b_i \right]$$

Therefore

$$E(\tilde{T}_i) = \sum_{b_i \in \text{sup}(\tilde{B}_i)} E(\tilde{T}_i | \tilde{B}_i = b_i) \cdot P \left[ \tilde{B}_i = b_i \right].$$

which is equal to

$$\sum_{b_i \in \text{sup}(\tilde{B}_i)} (1/P \left[ \tilde{B}_i = b_i \right]) \cdot P \left[ \tilde{B}_i = b_i \right] = \left| \text{sup}(\tilde{B}_i) \right|.$$

So  $\sum_i E(T_i) \leq |\mathcal{B}|$ .

# The complexity of the simulator

---

Let  $\tilde{T}_i$  be the number samples in the  $i$ th iteration.

Let  $\text{sup}(\tilde{B}_i)$  denote the support of the random variable  $\tilde{B}_i$  (excluding  $\perp$ ).

For every  $b_i$  we have:

$$P \left[ \tilde{V}_{B_i \rightarrow b_i} \right] = P \left[ \tilde{B}_i = b_i \right]$$

Therefore

$$E(\tilde{T}_i) = \sum_{b_i \in \text{sup}(\tilde{B}_i)} E(\tilde{T}_i | \tilde{B}_i = b_i) \cdot P \left[ \tilde{B}_i = b_i \right].$$

which is equal to

$$\sum_{b_i \in \text{sup}(\tilde{B}_i)} (1/P \left[ \tilde{B}_i = b_i \right]) \cdot P \left[ \tilde{B}_i = b_i \right] = \left| \text{sup}(\tilde{B}_i) \right|.$$

So  $\sum_i E(T_i) \leq |\mathcal{B}|$ .

# Plan

---

1. Introduction to the MPC ✓
2. Introduction to Adaptiveness in the MPC ✓
3. Formal security definitions ✓
4. Some equivalence and non-equivalence proofs
  - (a) passive case ✓
  - (b) active case ←

# Example 1 – introduction

---

We are going to show an example of a function  $f$  and a protocol  $\Pi$  that

securely evaluates  $f$  against any **non**-adaptive adversary according to the strongest definition possible (IT-security, perfect emulation).

but

does **not** securely evaluate  $f$  against an **adaptive** even according to the weakest definition possible (computational security and emulation)

# Example 1 — specification

---

3 players (connected by secure channels):

- a dealer  $D$ ,
- two receivers  $R_1, R_2$ .

# Example 1 — specification

---

3 players (connected by secure channels):

- a dealer  $D$ ,
- two receivers  $R_1, R_2$ .

The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

# Example 1 — specification

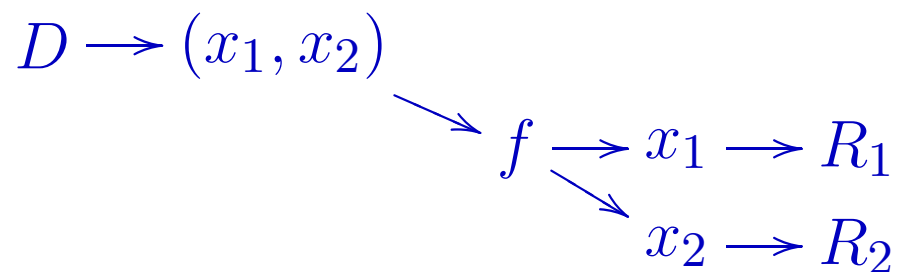
---

3 players (connected by secure channels):

- a dealer  $D$ ,
- two receivers  $R_1, R_2$ .

The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

Function  $f$ :



$$x_1, x_2 \in \{0, 1\}$$

# Example 1 — implementation

---

0.  $D$  inputs  $x_1, x_2$ .

# Example 1 — implementation

---

0.  $D$  inputs  $x_1, x_2$ .
1.  $D$  sends  $x_1$  to  $R_1$ .

# Example 1 — implementation

---

0.  $D$  inputs  $x_1, x_2$ .
1.  $D$  sends  $x_1$  to  $R_1$ .
2.  $D$  sends  $x_2$  to  $R_2$ .

# Example 1 — implementation

---

0.  $D$  inputs  $x_1, x_2$ .
1.  $D$  sends  $x_1$  to  $R_1$ .
2.  $D$  sends  $x_2$  to  $R_2$ .
3.  $R_1$  outputs  $x_1$  and  $R_2$  outputs  $x_2$ .

# Example 1 — implementation

---

0.  $D$  inputs  $x_1, x_2$ .
1.  $D$  sends  $x_1$  to  $R_1$ .
2.  $D$  sends  $x_2$  to  $R_2$ .
3.  $R_1$  outputs  $x_1$  and  $R_2$  outputs  $x_2$ .

This protocol **is** non-adaptively secure: for every adversary  $\mathcal{A}$  we can construct a simulator  $\mathcal{S}$ . The actions of the simulator depend on the set of players corrupted by  $\mathcal{A}$ . Two cases to consider here are:

1. Player  $D$  is corrupted.
2. Player  $D$  is honest, but  $R_1$  is corrupted.

# Example 1 — implementation

---

0.  $D$  inputs  $x_1, x_2$ .
1.  $D$  sends  $x_1$  to  $R_1$ .
2.  $D$  sends  $x_2$  to  $R_2$ .
3.  $R_1$  outputs  $x_1$  and  $R_2$  outputs  $x_2$ .

This protocol **is** non-adaptively secure: for every adversary  $\mathcal{A}$  we can construct a simulator  $\mathcal{S}$ . The actions of the simulator depend on the set of players corrupted by  $\mathcal{A}$ . Two cases to consider here are:

1. Player  $D$  is corrupted.
2. Player  $D$  is honest, but  $R_1$  is corrupted.

This simulation does not work if the set of corrupted players is not fixed in advance!

# The adaptive insecurity of Example 1 (1/2)

---

We show a scenario in which the adversary can achieve more in the real life execution of  $\Pi$  than in the ideal evaluation of  $f$ .

# The adaptive insecurity of Example 1 (1/2)

---

We show a scenario in which the adversary can achieve more in the real life execution of  $\Pi$  than in the ideal evaluation of  $f$ .

The adversary knows that  $x_1 = x_2$  and the distribution of the dealer's inputs  $(x_1, x_2) \in \{(0, 0), (1, 1)\}$  is uniform.

# The adaptive insecurity of Example 1 (1/2)

---

We show a scenario in which the adversary can achieve more in the real life execution of  $\Pi$  than in the ideal evaluation of  $f$ .

The adversary knows that  $x_1 = x_2$  and the distribution of the dealer's inputs  $(x_1, x_2) \in \{(0, 0), (1, 1)\}$  is uniform.

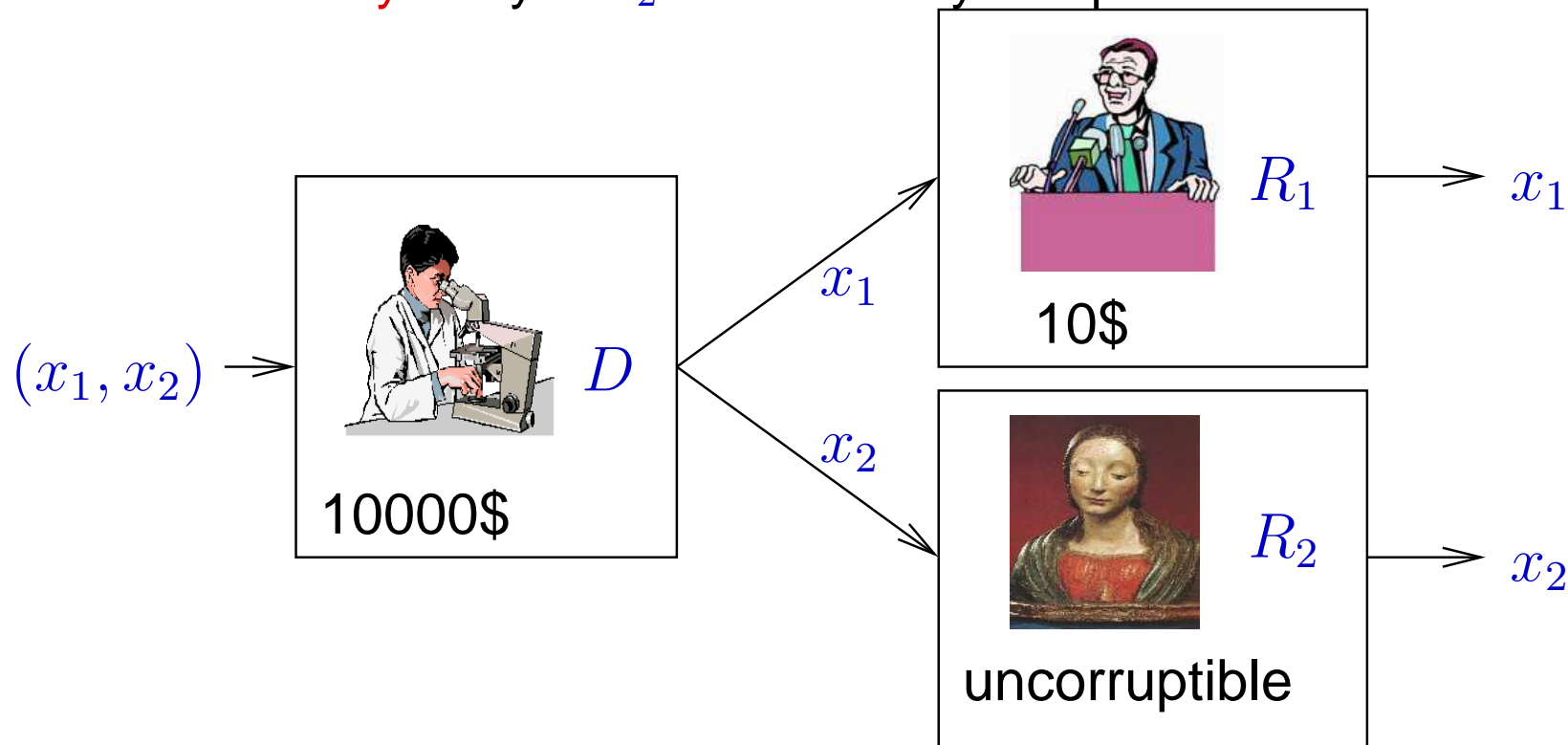
**Goal of the adversary:** Player  $R_2$  should always output 0.

# The adaptive insecurity of Example 1 (1/2)

We show a scenario in which the adversary can achieve more in the real life execution of  $\Pi$  than in the ideal evaluation of  $f$ .

The adversary knows that  $x_1 = x_2$  and the distribution of the dealer's inputs  $(x_1, x_2) \in \{(0, 0), (1, 1)\}$  is uniform.

**Goal of the adversary:** Player  $R_2$  should always output 0.



# The adaptive insecurity of Example 1 (2/2)

---

There exists an adaptive strategy for the adversary to achieve his goal in real-life execution of  $\Pi$  by

- corrupting always  $R_1$ , and
- corrupting  $D$  only when necessary.

# The adaptive insecurity of Example 1 (2/2)

---

There exists an adaptive strategy for the adversary to achieve his goal in real-life execution of  $\Pi$  by

- corrupting always  $R_1$ , and
- corrupting  $D$  only when necessary.

In this way he pays on average  $10\$ + \frac{1}{2}10000\$ = 5010\$$ .

# The adaptive insecurity of Example 1 (2/2)

---

There exists an adaptive strategy for the adversary to achieve his goal in real-life execution of  $\Pi$  by

- corrupting always  $R_1$ , and
- corrupting  $D$  only when necessary.

In this way he pays on average  $10\$ + \frac{1}{2}10000\$ = 5010\$$ .

It is easy to see that in the ideal model the adversary (to achieve the same goal) needs to corrupt  $D$  **always** (and to pay  $10000\$$ ). This is because:

- If the adversary does not corrupt  $D$  before the computation point then he has no information on  $x_1$ .
- After the computation point it is too late to change the output of  $R_2$ .

## Example 2 — introduction

---

The setting as in Example 1.: players  $D$ ,  $R_1$ , and  $R_2$ . The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

## Example 2 — introduction

---

The setting as in Example 1.: players  $D$ ,  $R_1$ , and  $R_2$ . The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

The players are given access to a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing the dealer  $D$  to commit to one bit.

## Example 2 — introduction

---

The setting as in Example 1.: players  $D$ ,  $R_1$ , and  $R_2$ . The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

The players are given access to a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing the dealer  $D$  to commit to one bit.

We want to construct a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing him to commit to a pair of bits.

## Example 2 — introduction

---

The setting as in Example 1.: players  $D$ ,  $R_1$ , and  $R_2$ . The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

The players are given access to a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing the dealer  $D$  to commit to one bit.

We want to construct a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing him to commit to a pair of bits.

Immediate solution: compose!

## Example 2 — introduction

---

The setting as in Example 1.: players  $D$ ,  $R_1$ , and  $R_2$ . The adversary can **actively** corrupt  $D$  or  $R_1$  (or both). Player  $R_2$  is incorruptible.

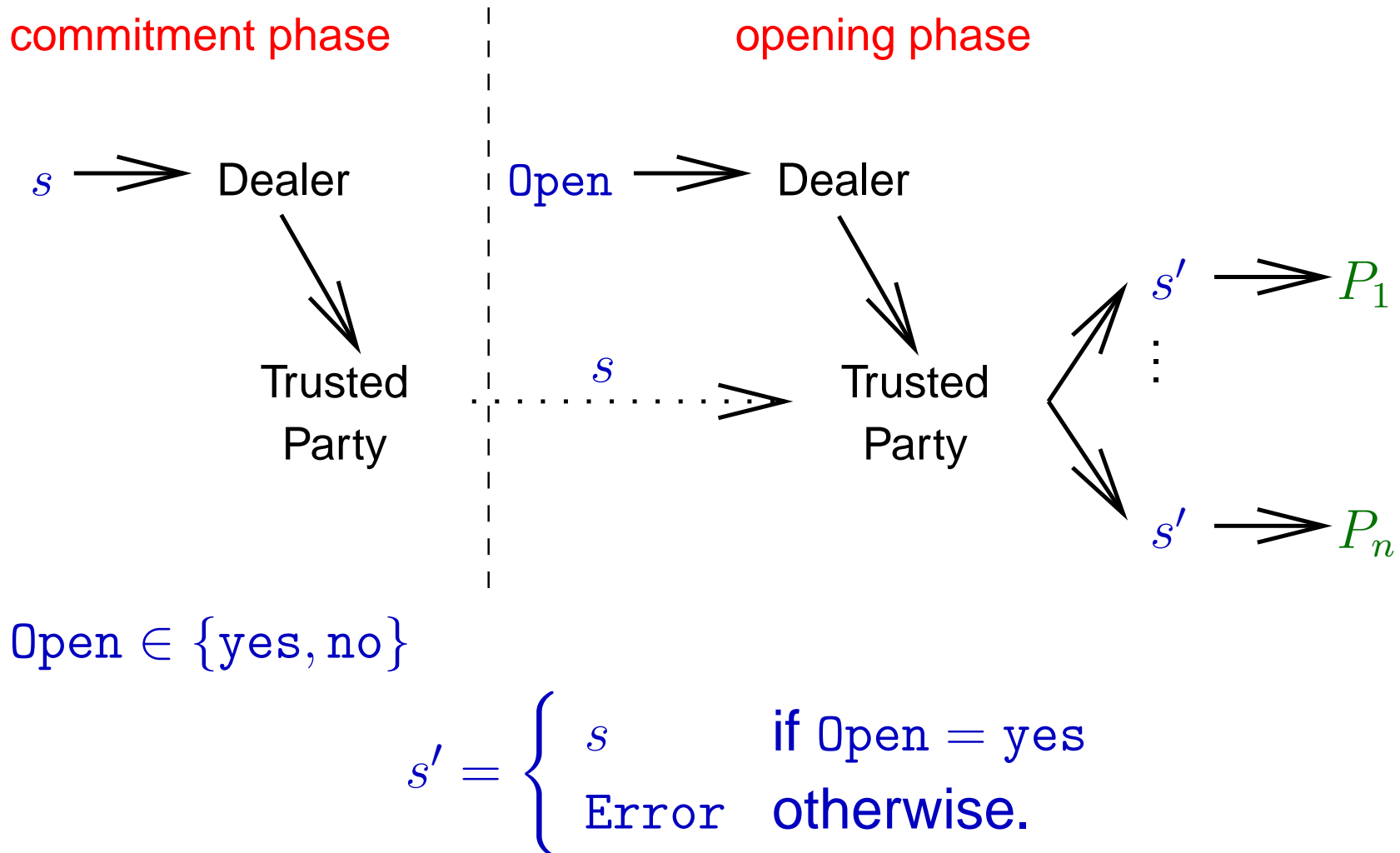
The players are given access to a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing the dealer  $D$  to commit to one bit.

We want to construct a commitment scheme  $\text{Commit}_1, \text{Open}_1$  allowing him to commit to a pair of bits.

Immediate solution: compose!

**Warning:** do it with care.

# The formal definition of a commitment scheme



# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

$\text{Commit}_2$

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

$\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

$\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .
2. Using  $\text{Commit}_1$  the dealer commits to  $x_1$ .

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

$\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .
2. Using  $\text{Commit}_1$  the dealer commits to  $x_1$ .
3. Using  $\text{Commit}_1$  the dealer commits to  $x_2$ .

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

$\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .
2. Using  $\text{Commit}_1$  the dealer commits to  $x_1$ .
3. Using  $\text{Commit}_1$  the dealer commits to  $x_2$ .

$\text{Open}_2$

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

$\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .
2. Using  $\text{Commit}_1$  the dealer commits to  $x_1$ .
3. Using  $\text{Commit}_1$  the dealer commits to  $x_2$ .

$\text{Open}_2$

1. Using  $\text{Open}_1$  the dealer opens  $x_1$ .

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

## $\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .
2. Using  $\text{Commit}_1$  the dealer commits to  $x_1$ .
3. Using  $\text{Commit}_1$  the dealer commits to  $x_2$ .

## $\text{Open}_2$

1. Using  $\text{Open}_1$  the dealer opens  $x_1$ .
2. Using  $\text{Open}_1$  the dealer opens  $x_2$ .

# The protocol

---

We construct a 2-bit commitment scheme  $\text{Commit}_2, \text{Open}_2$  by composing 1-bit commitment scheme  $\text{Commit}_1, \text{Open}_1$  **sequentially**.

## $\text{Commit}_2$

1. The dealer inputs  $(x_1, x_2)$ .
2. Using  $\text{Commit}_1$  the dealer commits to  $x_1$ .
3. Using  $\text{Commit}_1$  the dealer commits to  $x_2$ .

## $\text{Open}_2$

1. Using  $\text{Open}_1$  the dealer opens  $x_1$ .
2. Using  $\text{Open}_1$  the dealer opens  $x_2$ .
3. If both openings were successful then the players accept the opening and output  $(x_1, x_2)$ . Otherwise they output **Error**.

# The security of $\text{Commit}_2, \text{Open}_2$

---

The protocol from the previous slide is non-adaptively secure.

It is also secure if there is only one receiver.

But it is not secure in our settings (two receivers, adaptive adversary).

# The adaptive insecurity of Example 2 (1/2)

---

Suppose the adversary knows that the input values of  $D$  are distributed uniformly.

# The adaptive insecurity of Example 2 (1/2)

---

Suppose the adversary knows that the input values of  $D$  are distributed uniformly.

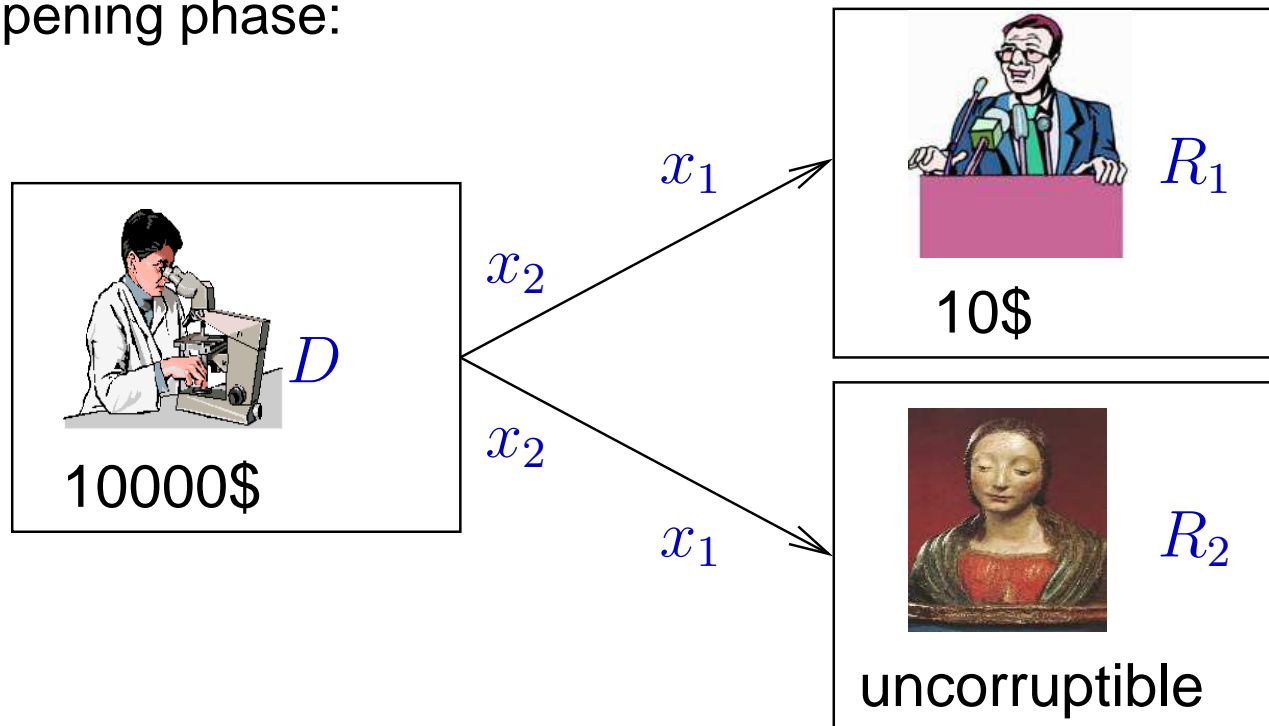
**Goal of the adversary:** prevent  $R_2$  from outputting (in the opening phase) any pair with first component equal to 0.

# The adaptive insecurity of Example 2 (1/2)

Suppose the adversary knows that the input values of  $D$  are distributed uniformly.

**Goal of the adversary:** prevent  $R_2$  from outputting (in the opening phase) any pair with first component equal to 0.

The opening phase:



# The adaptive insecurity of Example 2 (2/2)

---

As in example one: in the real-life the adversary can

# The adaptive insecurity of Example 2 (2/2)

---

As in example one: in the real-life the adversary can

1. Corrupt  $R_1$ .

# The adaptive insecurity of Example 2 (2/2)

---

As in example one: in the real-life the adversary can

1. Corrupt  $R_1$ .
2. Learn  $x_0$ .

# The adaptive insecurity of Example 2 (2/2)

---

As in example one: in the real-life the adversary can

1. Corrupt  $R_1$ .
2. Learn  $x_0$ .
3. Depending on the value of  $x_0$  decide to whether to corrupt  $D$ .

# The adaptive insecurity of Example 2 (2/2)

---

As in example one: in the real-life the adversary can

1. Corrupt  $R_1$ .
2. Learn  $x_0$ .
3. Depending on the value of  $x_0$  decide to whether to corrupt  $D$ .

This costs on average  $10\$ + \frac{1}{2}10000\$ = 5010\$$ .

# The adaptive insecurity of Example 2 (2/2)

---

As in example one: in the real-life the adversary can

1. Corrupt  $R_1$ .
2. Learn  $x_0$ .
3. Depending on the value of  $x_0$  decide to whether to corrupt  $D$ .

This costs on average  $10\$ + \frac{1}{2}10000\$ = 5010\$$ .

In the ideal-model he needs to corrupt  $D$  **always** (what costs  $\$10000$ ).

# Moral

---

When composing commitment schemes do it parallelly!

# Other issues

---

In [CDDIM] we also consider the following issues:

1. The **active** 2-party case.
  - (a) with a secure channel

# Other issues

---

In [CDDIM] we also consider the following issues:

1. The **active** 2-party case.
  - (a) with a secure channel
  - (b) without a secure channel

# Other issues

---

In [CDDIM] we also consider the following issues:

1. The **active** 2-party case.
  - (a) with a secure channel
  - (b) without a secure channel
2. The setting with **the environment**