# Intro to Malicious Security

## CS 598 DH

# Today's objectives

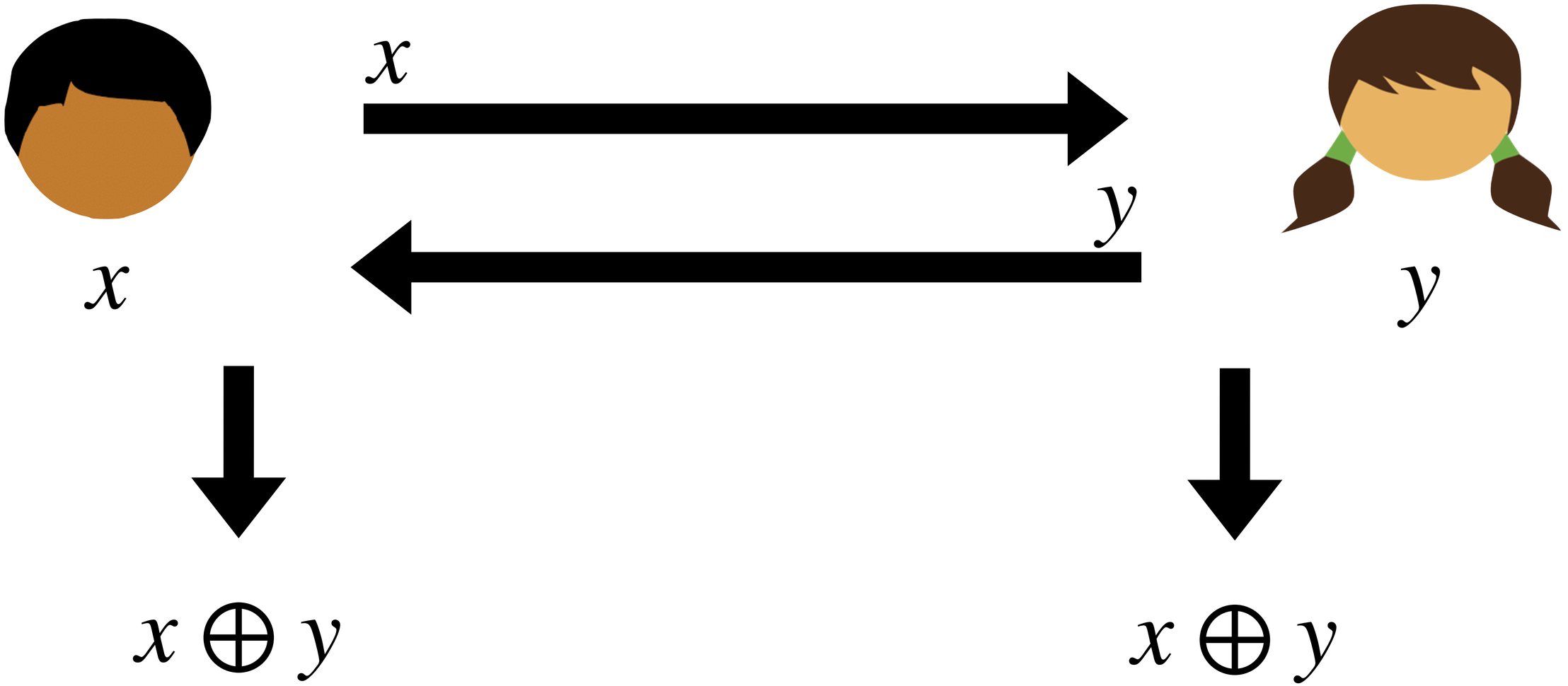
Define malicious security

Introduce notion of fairness

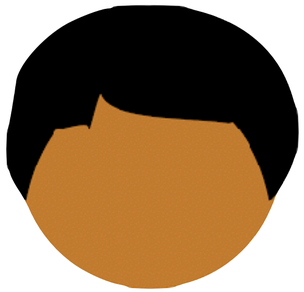
Introduce notion of abort

Prove a contrived protocol is secure in the malicious model

# Why malicious security is harder

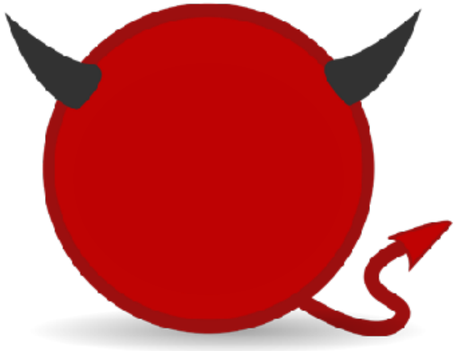


# Why malicious security is harder

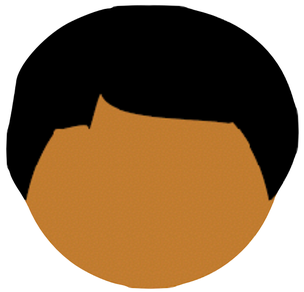


$x$

$x$

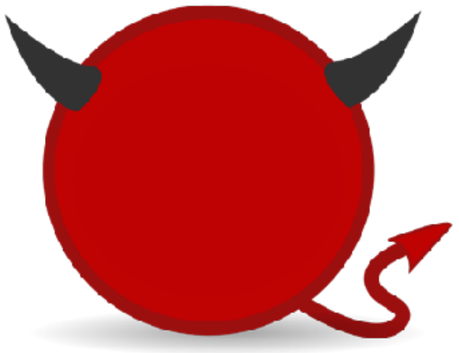


# Why malicious security is harder



$x$

$x$



The adversary is now an *arbitrary program*.
We cannot control its behavior, except by making clever use of cryptography.

# Why malicious security is harder

# Why malicious security is harder


We *cannot* force the adversary to respond

*Fairness: “if one party receives output, then everyone does”*

*Guaranteed Output Delivery: “each party **will** obtain the output”*

*Fairness: "if one party receives output, then everyone does"*

*Guaranteed Output Delivery: "each party **will** obtain the output"*

**Impossible** (in general) without an honest majority

Limits on the Security of Coin Flips
When Half the Processors are Faulty
(Extended Abstract)

Richard Cleve
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

**1. Introduction**

Protocols which allow an asynchronous network of processors to agree on a random (unbiased) bit are proposed in [1] and [4]. It is claimed that (assuming a trapdoor function exists), if less than half of the processors are faulty then the correct processors will still agree on a bit whose bias is negligibly small (when the running time of the processors is $poly(n)$ the bias is smaller than $O(\frac{1}{n^k})$ for all $k$). If half the processors are faulty then these protocols are no longer effective: the bits output by the correct processors may be heavily biased.

We prove that the above protocols are optimal in the sense that no protocol exists which tolerates faults in at least half of the processors. The result is very general because few restrictions are made on the types of communication allowed between correct processors (such as private channels and global channels) and the correct processors only need to agree on a bit in a weak probabilistic sense. Also, the faulty processors do not require very much power. They can privately communicate with each other but they cannot read messages which are exchanged privately between two correct processors.

An interesting instance of the problem arises when the number of processors is fixed at two and one of them may be faulty. This is the so-called coin flipping by telephone problem which is proposed in [3] and cannot be solved with very high security. There are protocols for 2-processor coin tossing which (assuming that a trapdoor function exists) achieve a weaker level of security (these are discussed in section 4). The processors run in $poly(n)$ time and the bias of the bit which a correct processor outputs is less than $O(\frac{1}{n^k})$ for some fixed $k$. More precisely, the bias will be less than $O(\frac{1}{\sqrt{r}})$ where $r$ is the number of rounds of communication in the protocol. For many applications (such as secret exchanging [5]) this weaker level of security is sufficient. In section 2 it is proven that no 2-processor protocol exists for which the bias of the output of a correct processor is less than $O(\frac{1}{r})$.

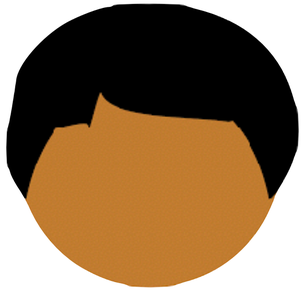
In [4] it is pointed out that multiprocessor coin tossing schemes have their application in the problem of randomly choosing a leader in a network of processors and the problem of fairly allocating resources within a network.

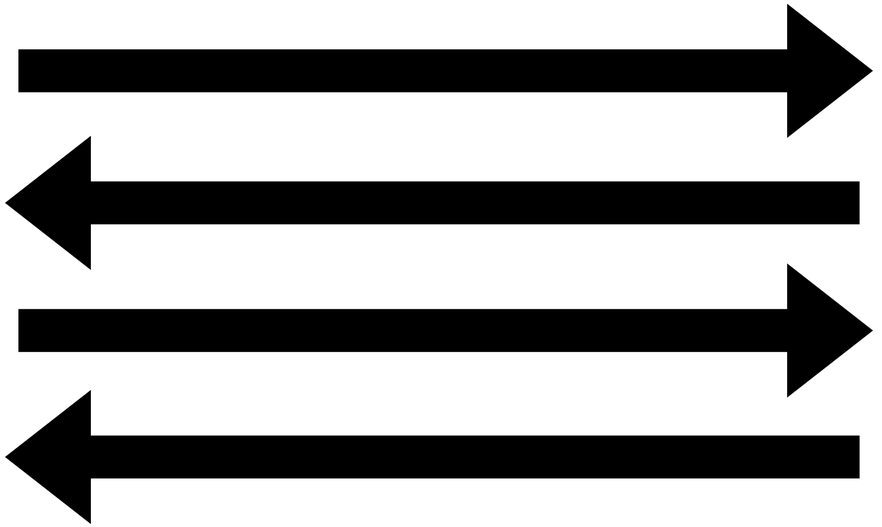
**2. 2-Processor Coin Tossing Schemes**

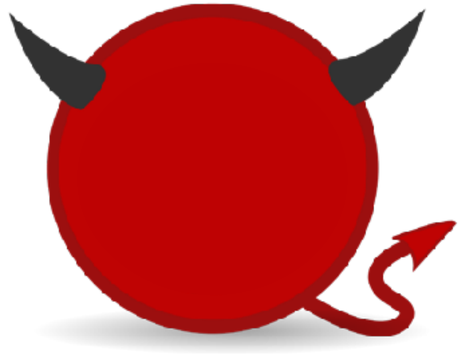
In 2.1, 2-Processor coin tossing schemes are defined precisely and, in 2.2, a lower bound on the security of 2-processor coin tossing schemes is proven.

**2.1 Definitions**

A *2-processor bit selection scheme* is a sequence of pairs of processors $\{(A^n, B^n)\}_{n=1}^{\infty}$ with the following properties. For each $n$, $A^n$ and $B^n$ each have access to a private supply of random bits and they can communicate with each other. If the system is executed then $A^n$, $B^n$ will output bits $a$, $b$ (respectively) within $poly(n)$ time.







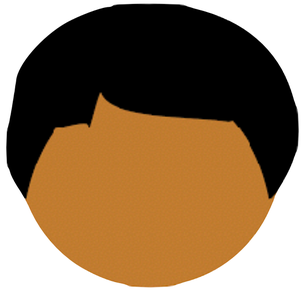
# Limits on the Security of Coin Flips When Half the Processors are Faulty

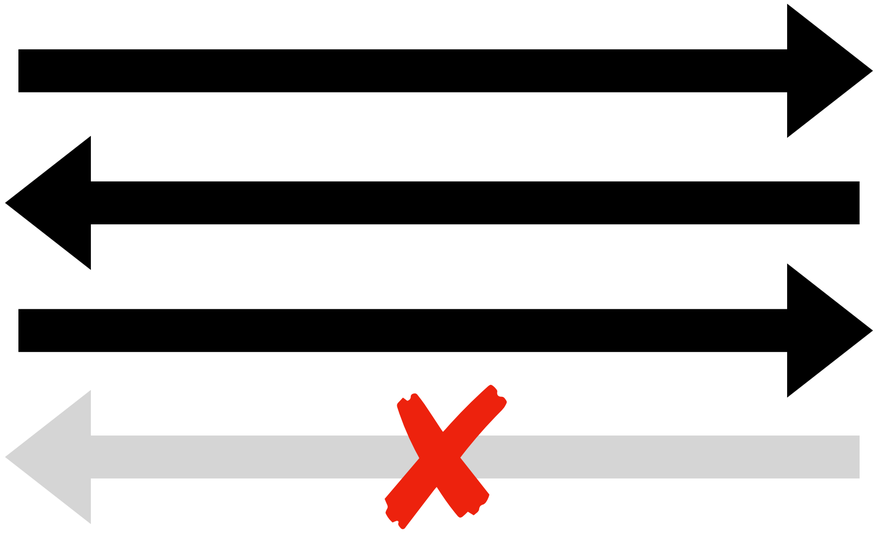
(Extended Abstract)

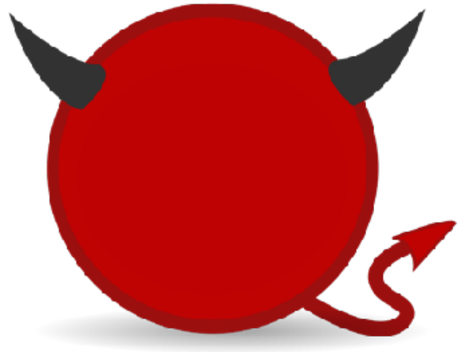
Richard Cleve
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

## 1. Introduction

Protocols which allow an asynchronous network of processors to agree on a random (unbiased) bit are proposed in [1] and [4]. It is claimed that (assuming a trapdoor function exists), if less than half of the processors are faulty then the correct processors will still agree on a bit whose bias is negligibly small (when the running time of the processors is $poly(n)$ the bias is smaller than $O(\frac{1}{n^k})$ for all $k$). If half the processors are faulty then these protocols are no longer effective: the bits output by the may be faulty. This is the so-called coin flipping by telephone problem which is proposed in [3] and cannot be solved with very high security. There are protocols for 2-processor coin tossing which (assuming that a trapdoor function exists) achieve a weaker level of security (these are discussed in section 4). The processors run in $poly(n)$ time and the bias of the bit which a correct processor outputs is less than $O(\frac{1}{n^k})$ for some fixed $k$. More precisely, the bias will be less than $O(\frac{1}{\sqrt{r}})$ where $r$ is the number of rounds of communication in the protocol. For many







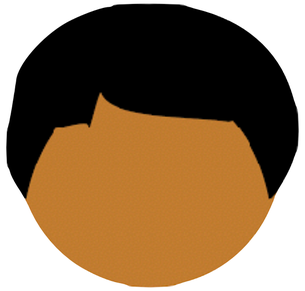
# Limits on the Security of Coin Flips When Half the Processors are Faulty

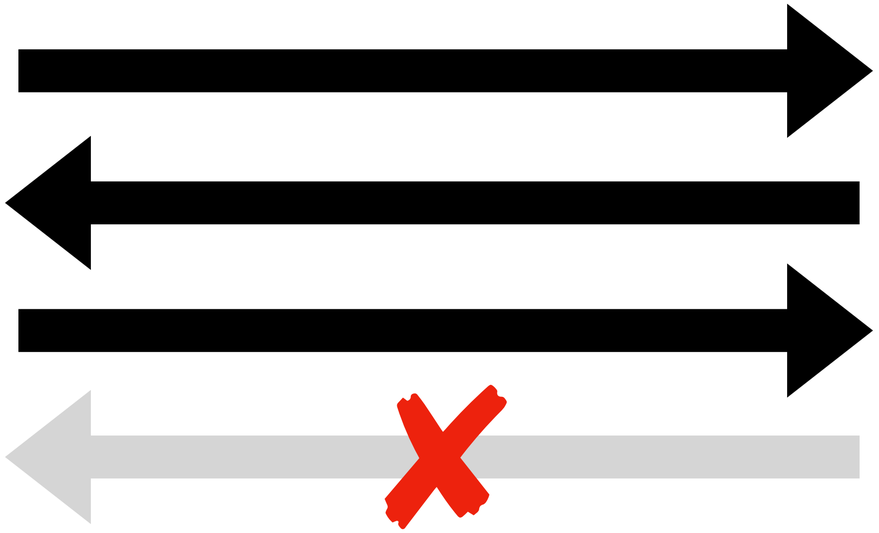
(Extended Abstract)

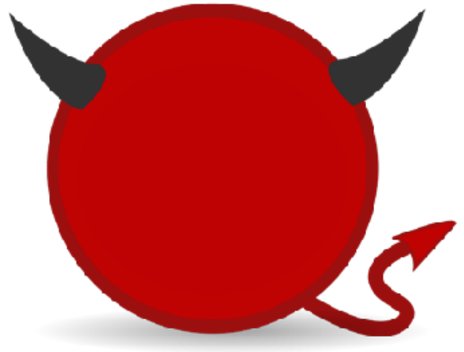
Richard Cleve
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

## 1. Introduction

Protocols which allow an asynchronous network of processors to agree on a random (unbiased) bit are proposed in [1] and [4]. It is claimed that (assuming a trapdoor function exists), if less than half of the processors are faulty then the correct processors will still agree on a bit whose bias is negligibly small (when the running time of the processors is $poly(n)$ the bias is smaller than $O(\frac{1}{n^k})$ for all $k$). If half the processors are faulty then these protocols are no longer effective: the bits output by the may be faulty. This is the so-called coin flipping by telephone problem which is proposed in [3] and cannot be solved with very high security. There are protocols for 2-processor coin tossing which (assuming that a trapdoor function exists) achieve a weaker level of security (these are discussed in section 4). The processors run in $poly(n)$ time and the bias of the bit which a correct processor outputs is less than $O(\frac{1}{n^k})$ for some fixed $k$. More precisely, the bias will be less than $O(\frac{1}{\sqrt{r}})$ where $r$ is the number of rounds of communication in the protocol. For many







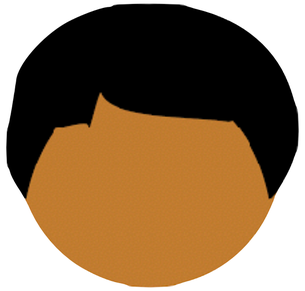
Limits on the Security of Coin Flips
When Half the Processors are Faulty
(Extended Abstract)

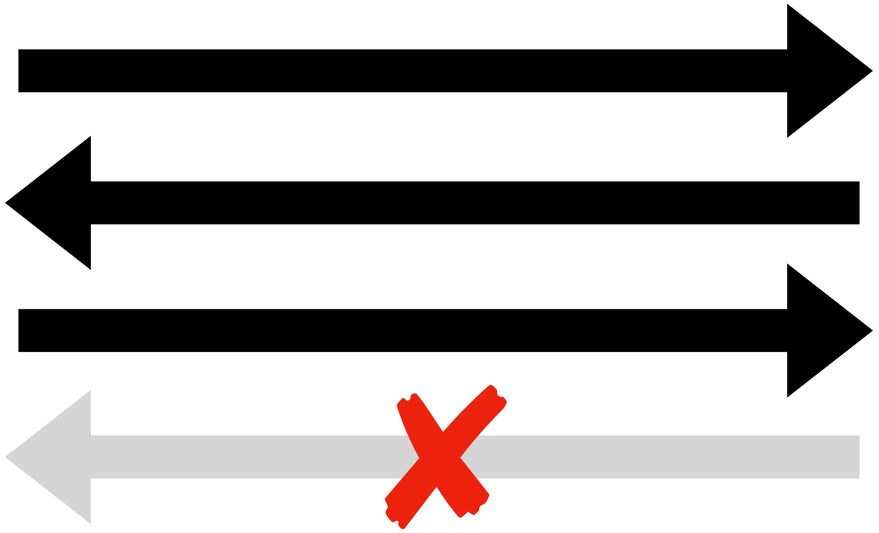
Richard Cleve
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

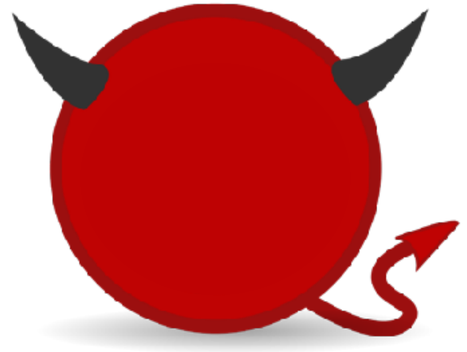
**1. Introduction**

Protocols which allow an asynchronous network of processors to agree on a random (unbiased) bit are proposed in [1] and [4]. It is claimed that (assuming a trapdoor function exists), if less than half of the processors are faulty then the correct processors will still agree on a bit whose bias is negligibly small (when the running time of the processors is $poly(n)$ the bias is smaller than $O(\frac{1}{n^k})$ for all $k$). If half the processors are faulty then these protocols are no longer effective: the bits output by the may be faulty. This is the so-called coin flipping by telephone problem which is proposed in [3] and cannot be solved with very high security. There are protocols for 2-processor coin tossing which (assuming that a trapdoor function exists) achieve a weaker level of security (these are discussed in section 4). The processors run in $poly(n)$ time and the bias of the bit which a correct processor outputs is less than $O(\frac{1}{n^k})$ for some fixed $k$. More precisely, the bias will be less than $O(\frac{1}{\sqrt{r}})$ where $r$ is the number of rounds of communication in the protocol. For many

Idea: One party will learn the output *first.* We cannot force this party to send the last message







Limits on the Security of Coin Flips
When Half the Processors are Faulty
(Extended Abstract)

Richard Cleve
Department of Computer Science
University of Toronto
Toronto, Canada M5S 1A4

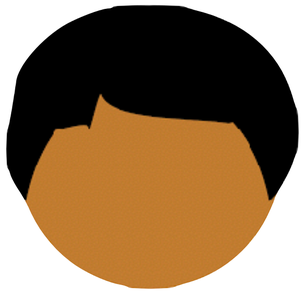
**1. Introduction**

Protocols which allow an asynchronous network of processors to agree on a random (unbiased) bit are proposed in [1] and [4]. It is claimed that (assuming a trapdoor function exists), if less than half of the processors are faulty then the correct processors will still agree on a bit whose bias is negligibly small (when the running time of the processors is $poly(n)$ the bias is smaller than $O(\frac{1}{n^k})$ for all $k$). If half the processors are faulty then these protocols are no longer effective: the bits output by the may be faulty. This is the so-called coin flipping by telephone problem which is proposed in [3] and cannot be solved with very high security. There are protocols for 2-processor coin tossing which (assuming that a trapdoor function exists) achieve a weaker level of security (these are discussed in section 4). The processors run in $poly(n)$ time and the bias of the bit which a correct processor outputs is less than $O(\frac{1}{n^k})$ for some fixed $k$. More precisely, the bias will be less than $O(\frac{1}{\sqrt{r}})$ where $r$ is the number of rounds of communication in the protocol. For many

Idea: One party will learn the output *first.* We cannot force this party to send the last message

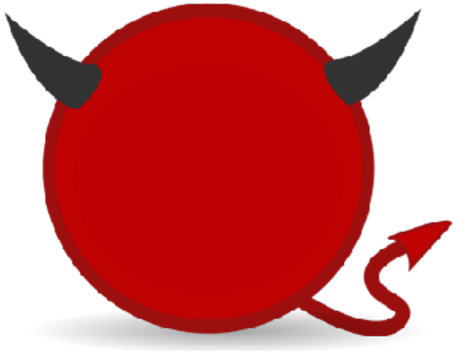
Our definition of malicious security will respect this impossibility

# Why malicious security is harder

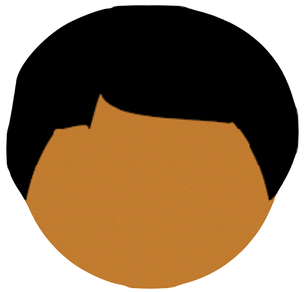


$x$

$x$

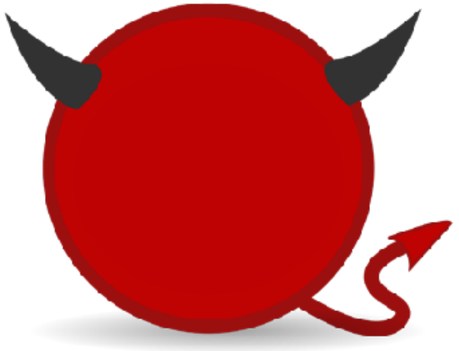


# Why malicious security is harder

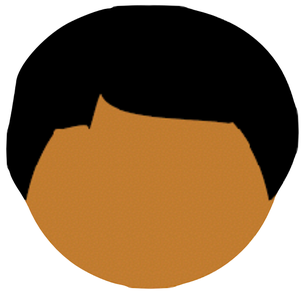


$x$

$x$



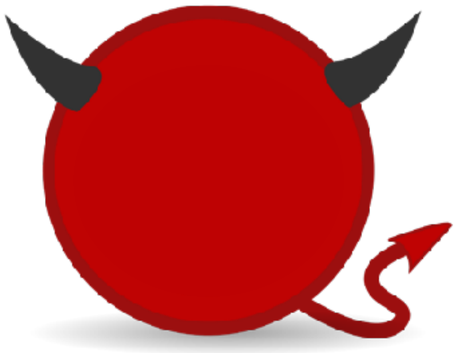
# Why malicious security is harder

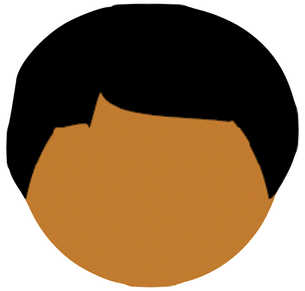


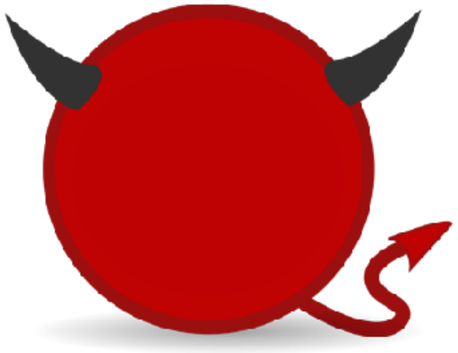
$x$

$x$

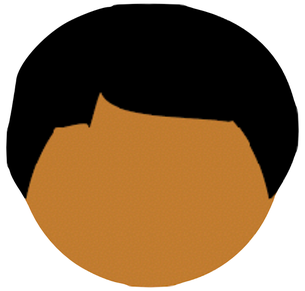
$x$

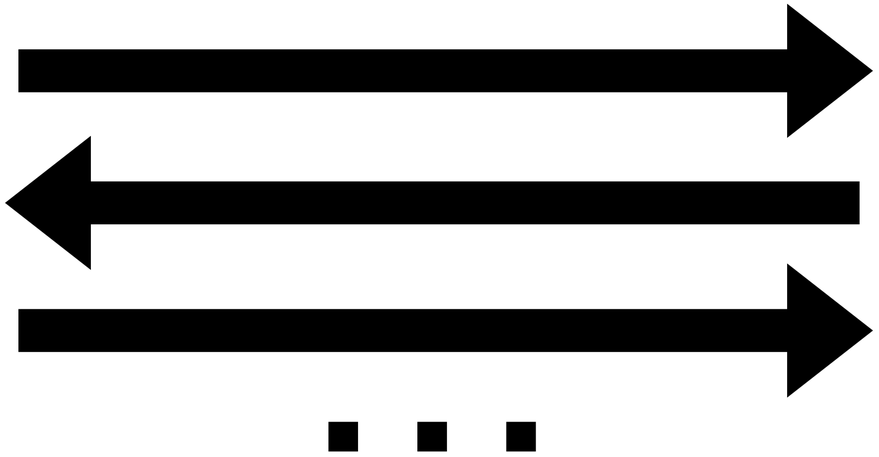


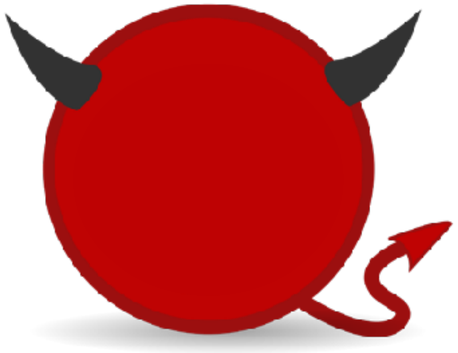




What can go wrong?

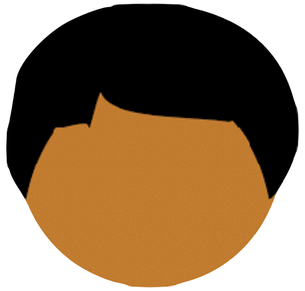


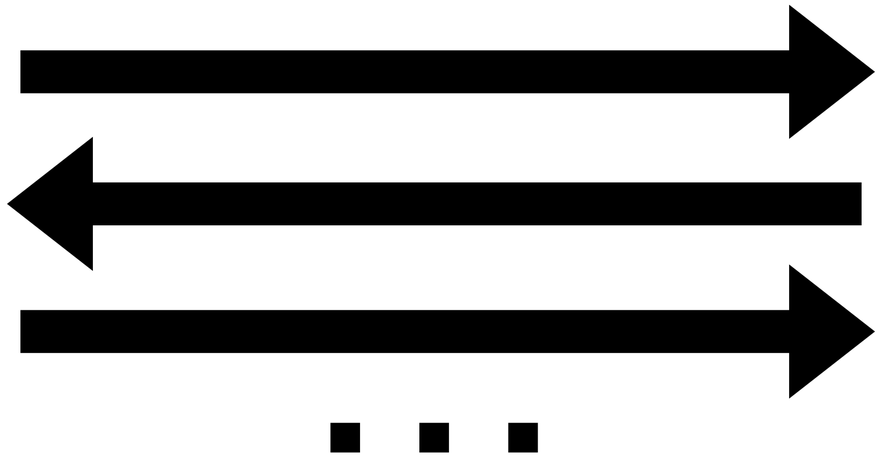


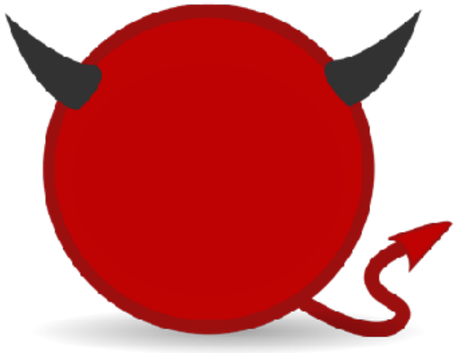


# What can go wrong?

Send the wrong message



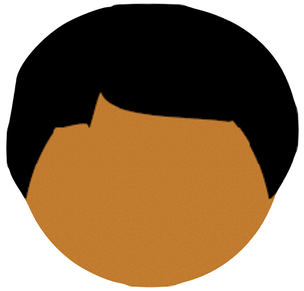


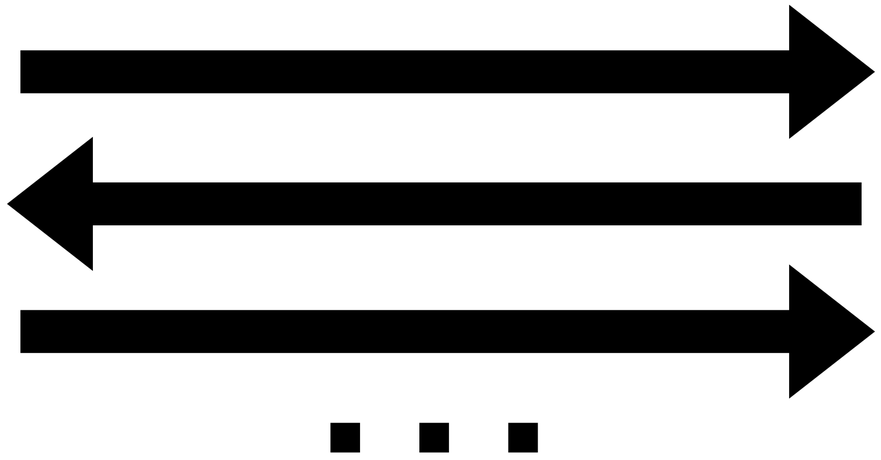


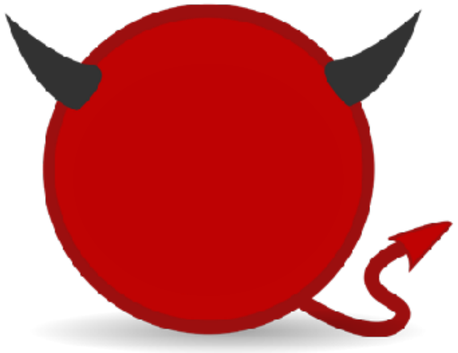
# What can go wrong?

Send the wrong message

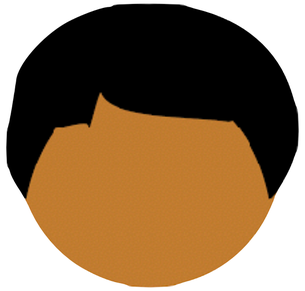
Refuse to send a message

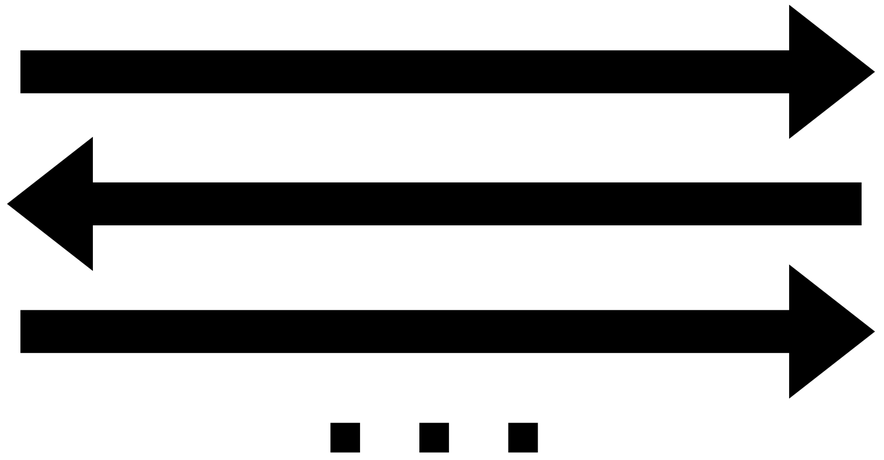


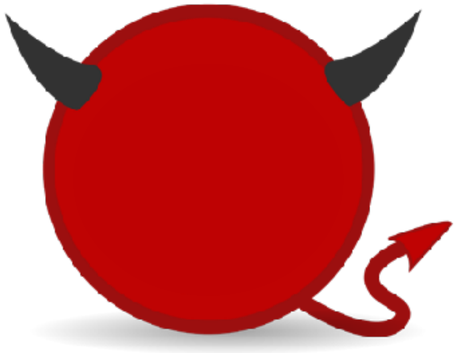




What can go *in terms of outcomes*?





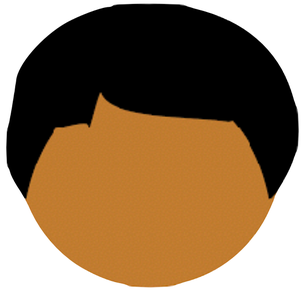


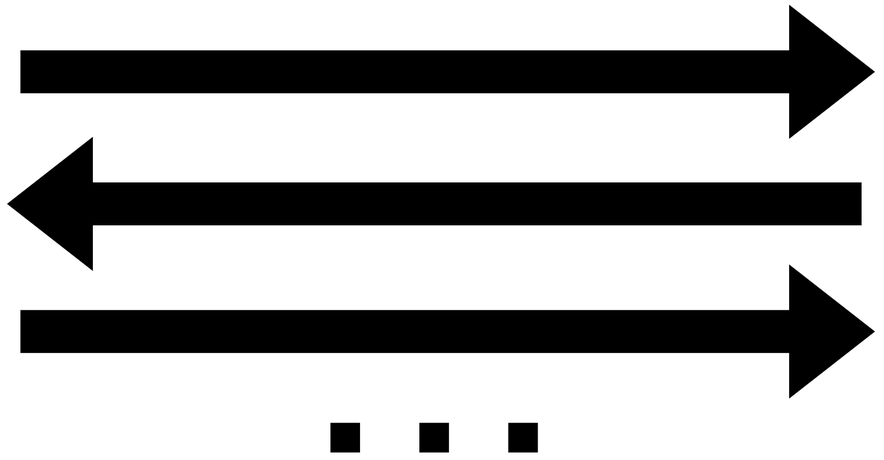
# What can go *in terms of outcomes*?

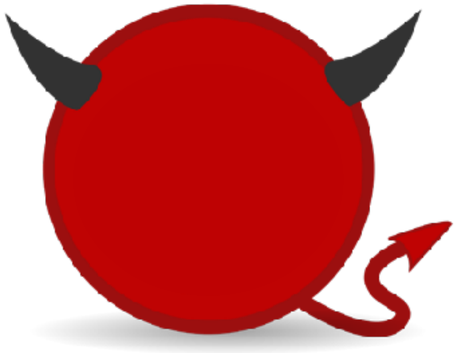
Cause honest party to output wrong answer

Learn too much information about other party's input

Prevent honest party from learning output





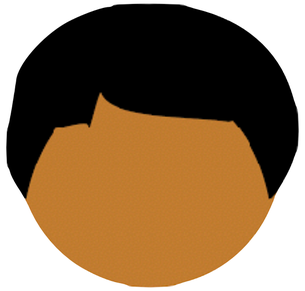


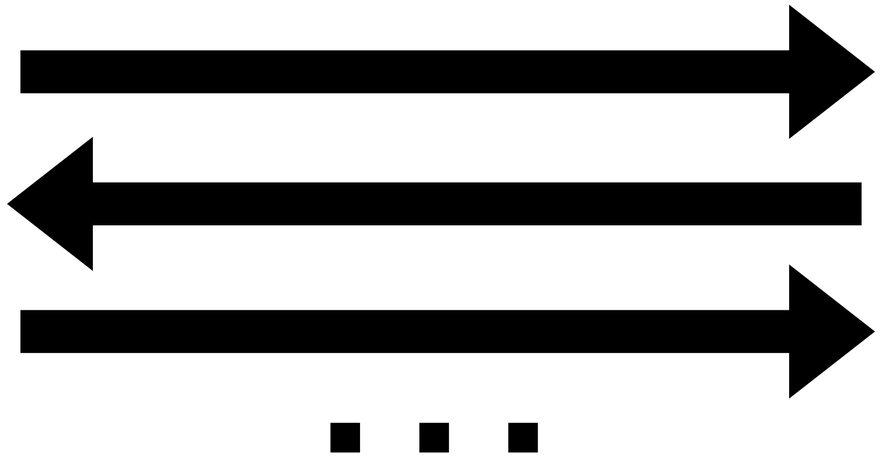
# What can go *in terms of outcomes*?

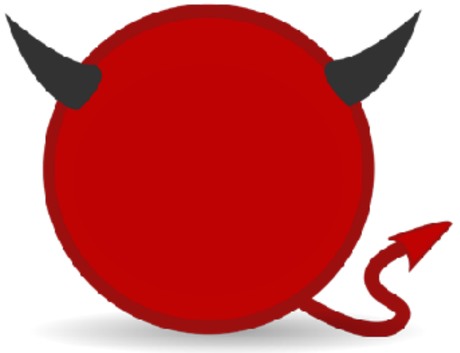
Cause honest party to output wrong answer ✓

Learn too much information about other party's input ✓

Prevent honest party from learning output







# What can go *in terms of outcomes*?

Cause honest party to output wrong answer ✓

Learn too much information about other party's input ✓

Prevent honest party from learning output ✗

# Semi-honest Security

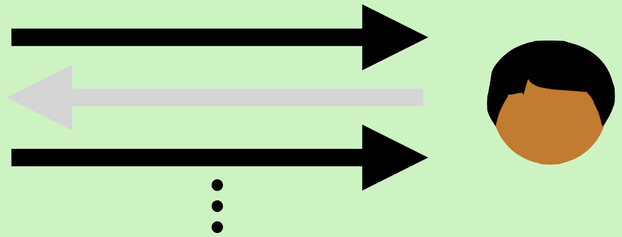

Our definition of semi-honest security compares our real-world protocol to a (very simple) idealized protocol involving a trusted third party.

Malicious security is the same

# Semi-honest Security

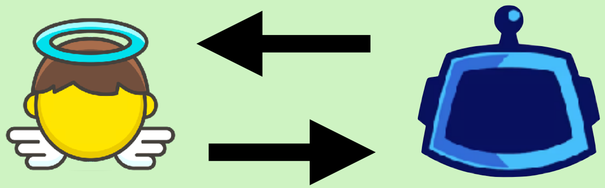
*Simulator*

***Real***



$\text{View}^{\Pi}_{\text{Bob}}(x, y) = \{\ y, m_0, m_1, \ldots\ \}$

***Ideal***



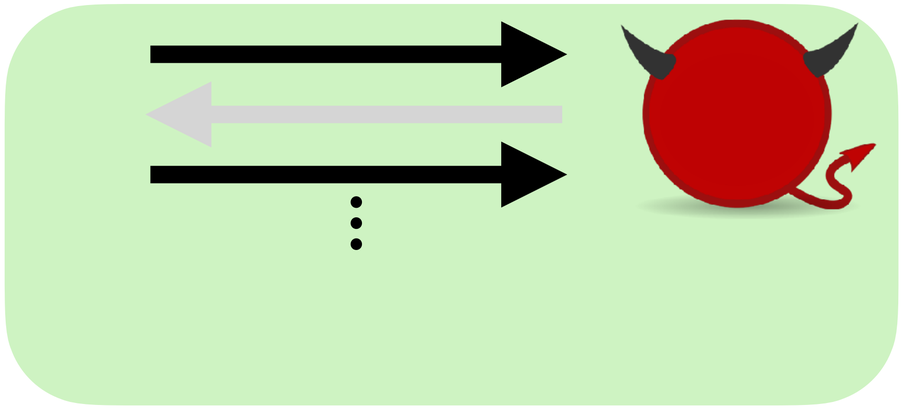
$\text{Output}^{\text{Sim}}_{\text{Bob}}(x, y) = \{\ y, m_0, m_1, \ldots\ \}$

*These should “look the same”*

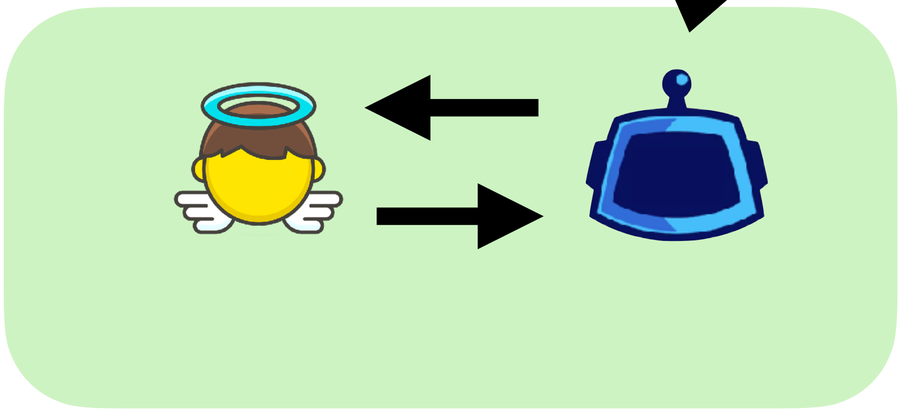
# Malicious Security

*Simulator*

***Real***



***Ideal***

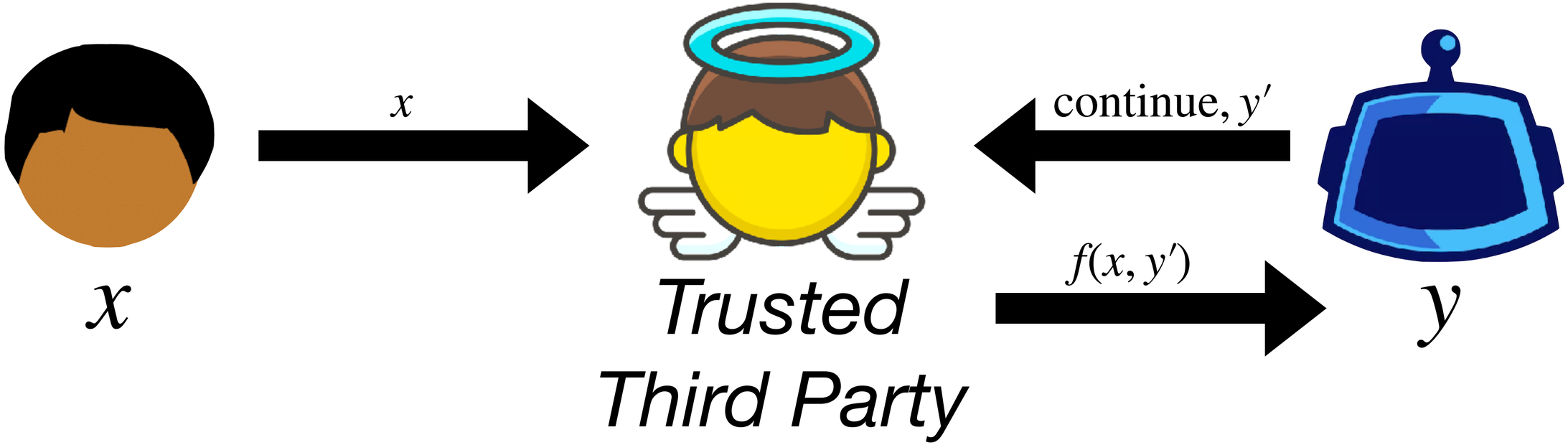


*These **interactions** should “look the same”*

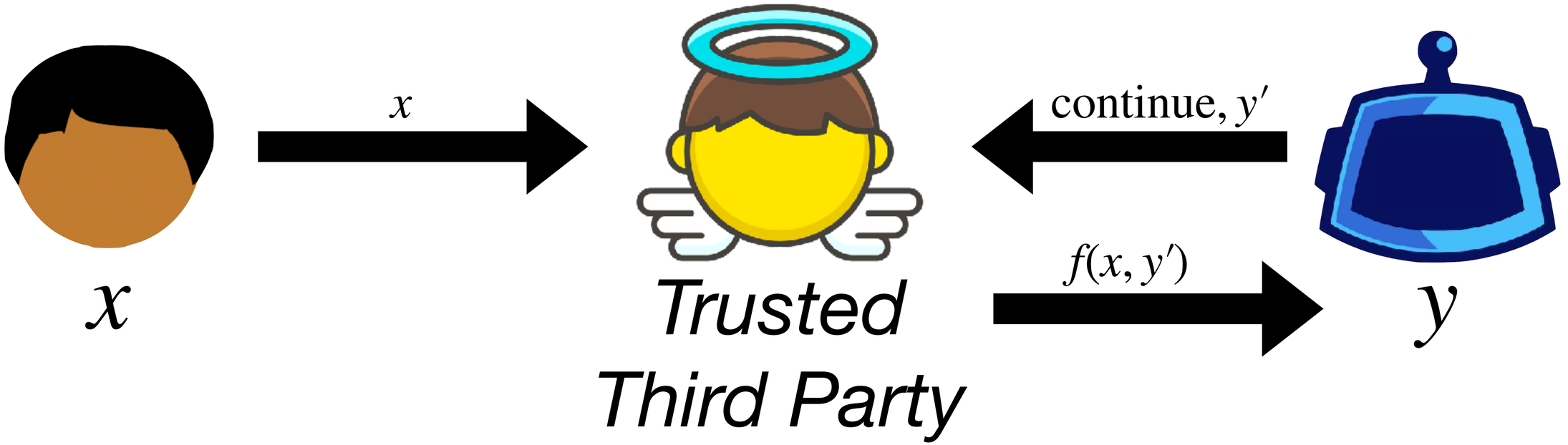
# Malicious security ideal-world execution

# Malicious security ideal-world execution

# Malicious security ideal-world execution

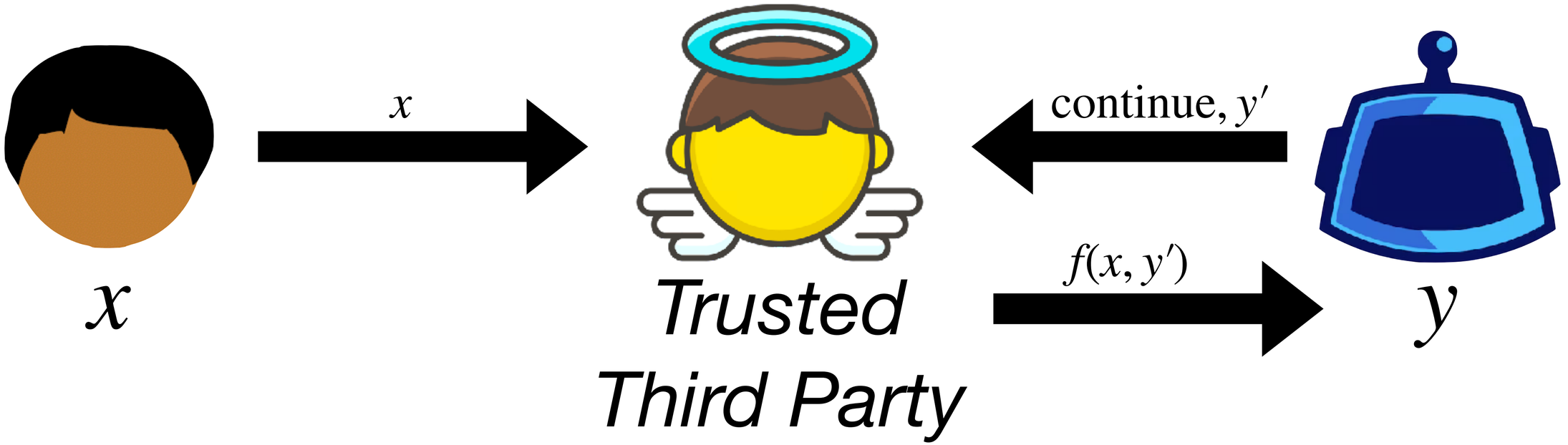


# Malicious security ideal-world execution




Why do we send output to adversary, and not honest party?

# Malicious security ideal-world execution




Why do we send output to adversary, and not honest party?

We need to model that any real-world protocol is necessarily *unfair*

# Malicious security ideal-world execution

# Malicious security ideal-world execution

# Malicious security ideal-world execution


honest party outputs

$f(x, y')$

# Malicious security ideal-world execution


honest party outputs

$f(x, y')$

adversary outputs… ?

# Malicious security ideal-world execution


honest party outputs

$f(x, y')$

adversary outputs… ?

whatever it wants

# How To Simulate It – A Tutorial on the Simulation Proof Technique*

Yehuda Lindell

Dept. of Computer Science
Bar-Ilan University, Israel
lindell@biu.ac.il

April 25, 2021

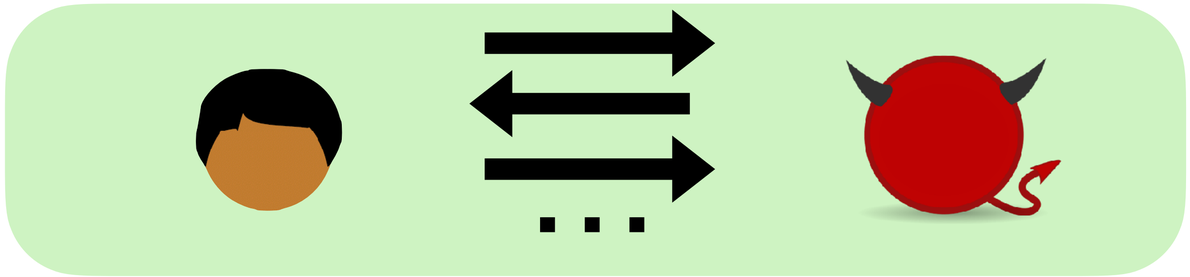
**Abstract**

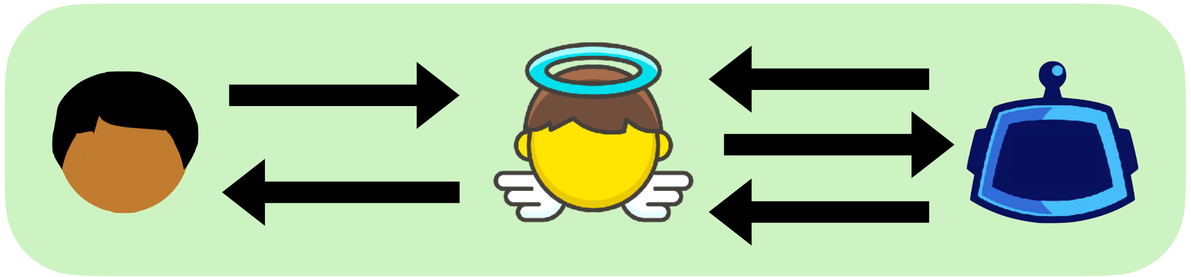
One of the most fundamental notions of cryptography is that of *simulation*. It stands behind the concepts of semantic security, zero knowledge, and security for multiparty computation. However, writing a simulator and proving security via the use of simulation is a non-trivial task, and one that many newcomers to the field often find difficult. In this tutorial, we provide a guide to how to write simulators and prove security via the simulation paradigm. Although we have tried to make this tutorial as stand-alone as possible, we assume some familiarity with the notions of secure encryption, zero-knowledge, and secure computation.

**Keywords:** secure computation, the simulation technique, tutorial

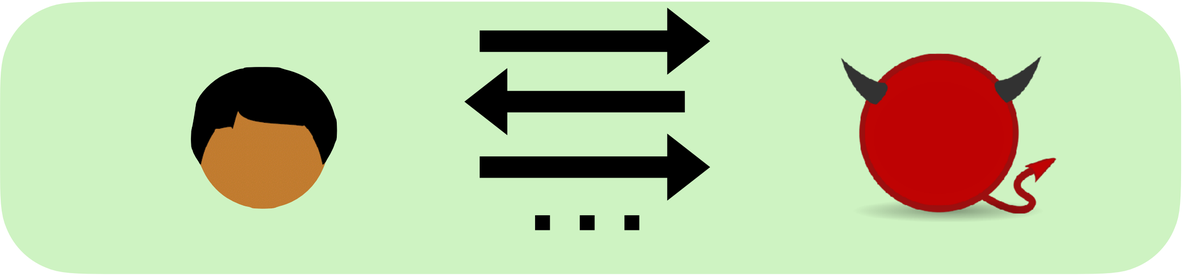
*This tutorial appeared in the book *Tutorials on the Foundations of Cryptography*, published in honor of Oded Goldreich's 60th birthday.

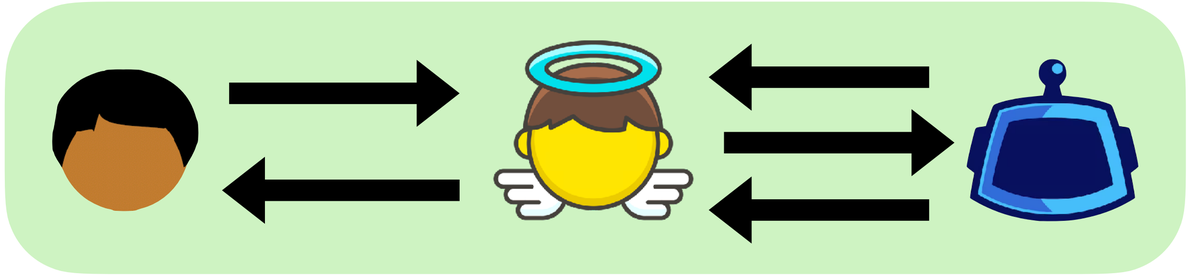
# Malicious Security





# Malicious Security

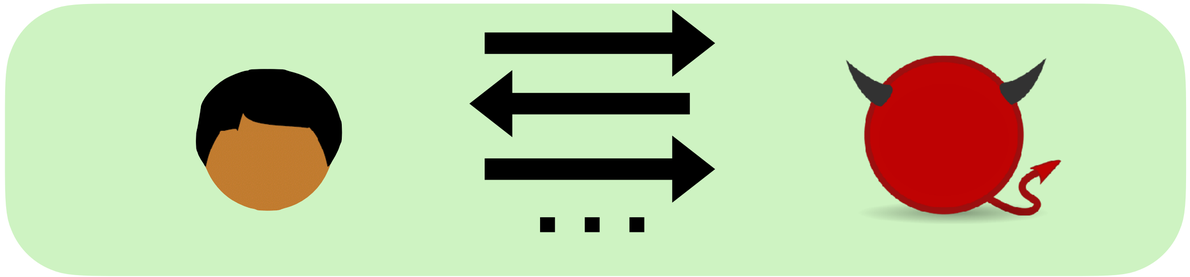


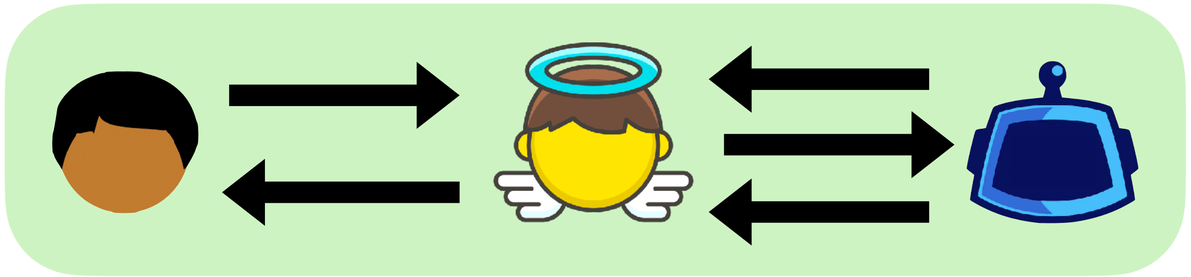


*A protocol $\Pi$ securely realizes a functionality $f$ in the presence of a malicious adversary if for* ***every*** *real-world adversary $\mathcal{A}$ corrupting party $i$,* ***there exists*** *an ideal-world adversary $\mathcal{S}_i$ (a simulator) such that for all inputs $x, y$ the following holds:*

$$\text{Real}^{\Pi}_{\mathcal{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

# Malicious Security





*A protocol $\Pi$ securely realizes a functionality $f$ in the presence of a malicious adversary if for* ***every*** *real-world adversary $\mathcal{A}$ corrupting party $i$,* ***there exists*** *an ideal-world adversary $\mathcal{S}_i$ (a simulator) such that for all inputs $x, y$ the following holds:*

$$\text{Real}^{\Pi}_{\mathcal{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

Ensemble of outputs of ***each*** party

$$\text{Real}^{\Pi}_{\mathscr{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

Adversary $\mathscr{A}$ does whatever it wants and outputs whatever it wants…

$$\text{Real}^{\Pi}_{\mathscr{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

Adversary $\mathscr{A}$ does whatever it wants and outputs whatever it wants…

How do we construct a simulator that outputs something indistinguishable?

$$\text{Real}^{\Pi}_{\mathcal{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

Adversary $\mathcal{A}$ does whatever it wants and outputs whatever it wants…

How do we construct a simulator that outputs something indistinguishable?

*… for **every** real-world adversary $\mathcal{A}$ …,*
***there exists** a simulator $\mathcal{S}$ …*

$$\text{Real}^{\Pi}_{\mathcal{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

Adversary $\mathcal{A}$ does whatever it wants and outputs whatever it wants…

How do we construct a simulator that outputs something indistinguishable?

*… for **every** real-world adversary $\mathcal{A}$ …,*
***there exists** a simulator $\mathcal{S}$ …*

Our simulator only needs to handle a *specific, quantified* $\mathcal{A}$

$$\text{Real}^{\Pi}_{\mathcal{A}}(x, y) \approx \text{Ideal}^{f}_{\mathcal{S}_i}(x, y)$$

Adversary $\mathcal{A}$ does whatever it wants and outputs whatever it wants…

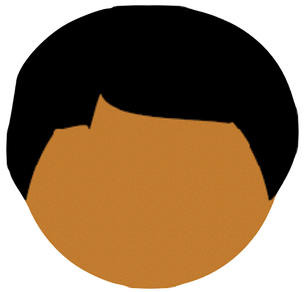
How do we construct a simulator that outputs something indistinguishable?

*… for **every** real-world adversary $\mathcal{A}$ …,*
***there exists** a simulator $\mathcal{S}$ …*

Our simulator only needs to handle a *specific, quantified* $\mathcal{A}$

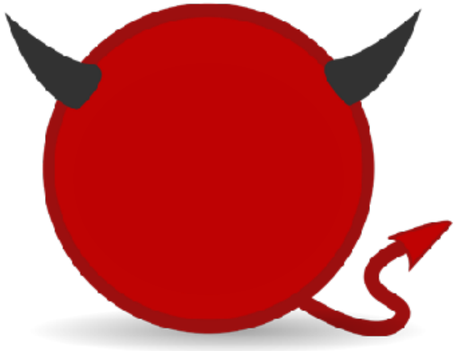
Think of $\mathcal{A}$ as a *program* that $\mathcal{S}$ can call as
many times as it would like

# Remark: Syntactic Convenience

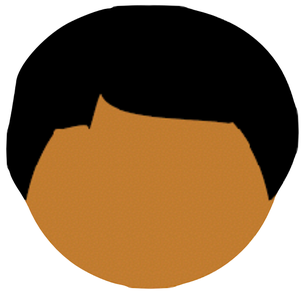


$x$

$x$

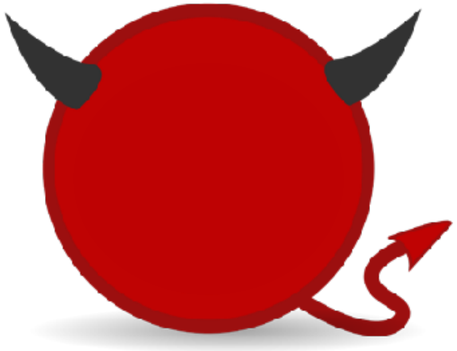


# Remark: Syntactic Convenience

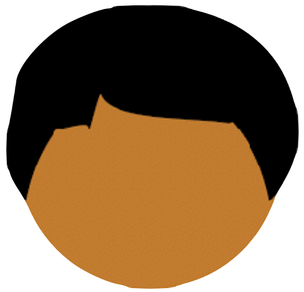


$x$

$x$

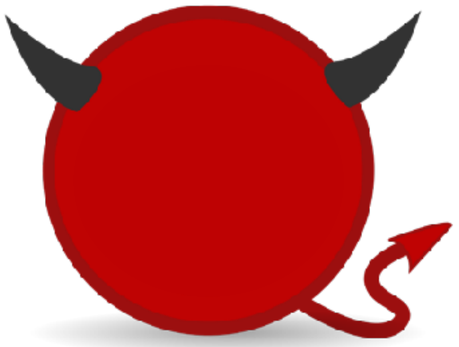



# Remark: Syntactic Convenience



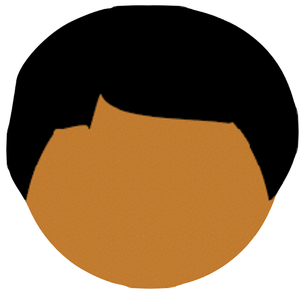
$x$

$x$




If adversary's message does not syntactically match the protocol, we treat that as an abort

# One big difference between semi-honest and malicious



$x$

$x$

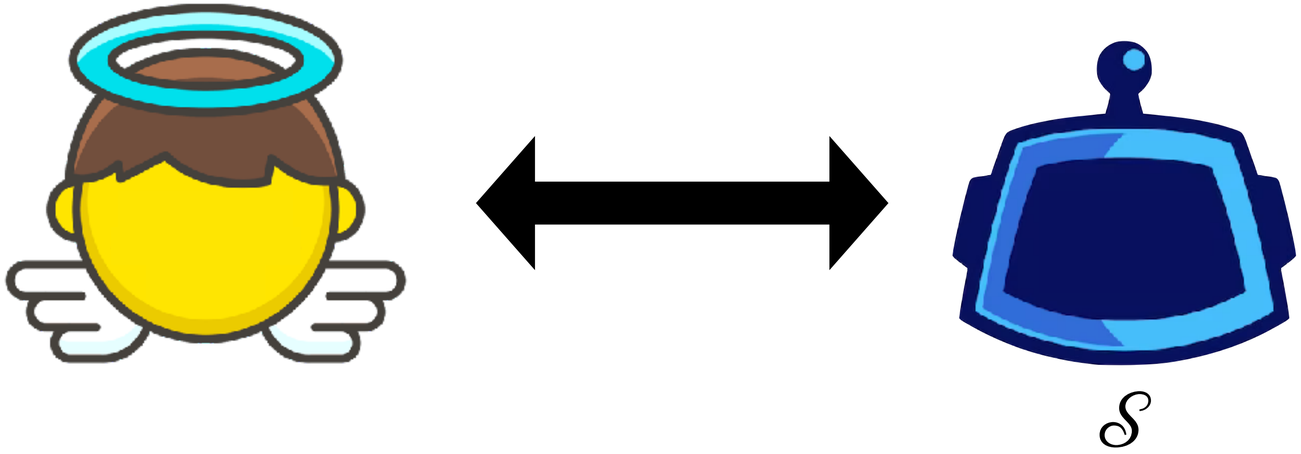
The notion of “input” of the adversary becomes murky

# One big difference between semi-honest and malicious

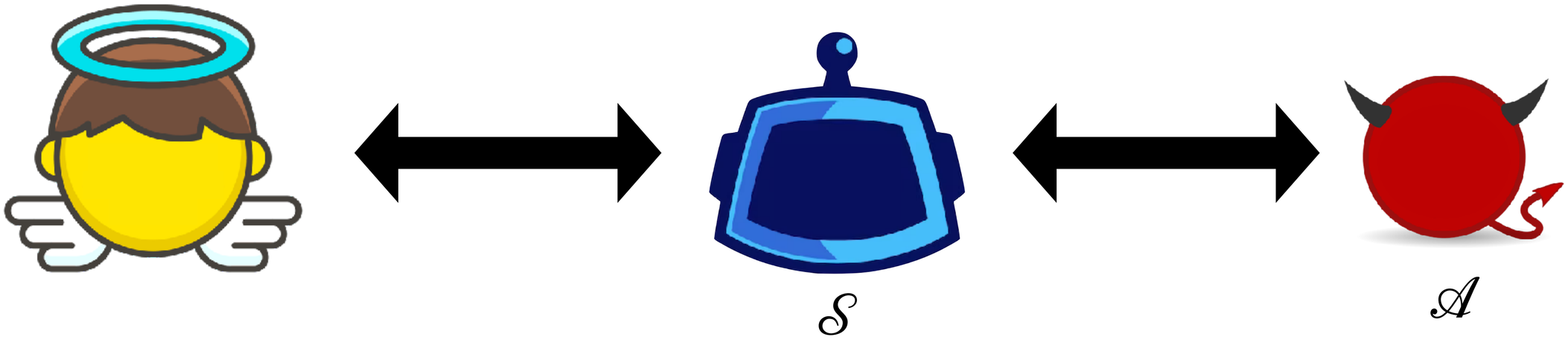

The notion of “input” of the adversary becomes murky

Our malicious simulator will not be given direct access to the adversary’s “input”

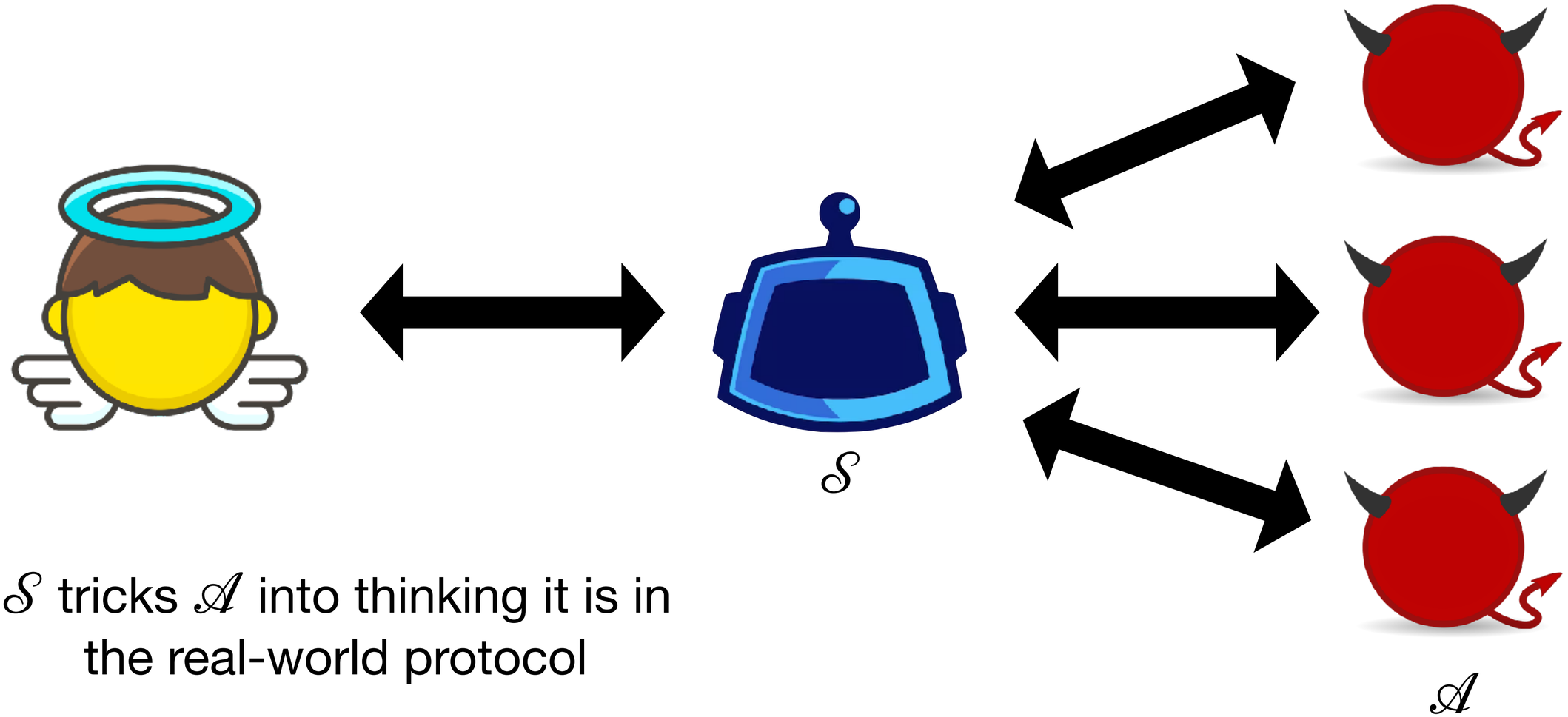
# Typical Proof of Malicious Security



# Typical Proof of Malicious Security



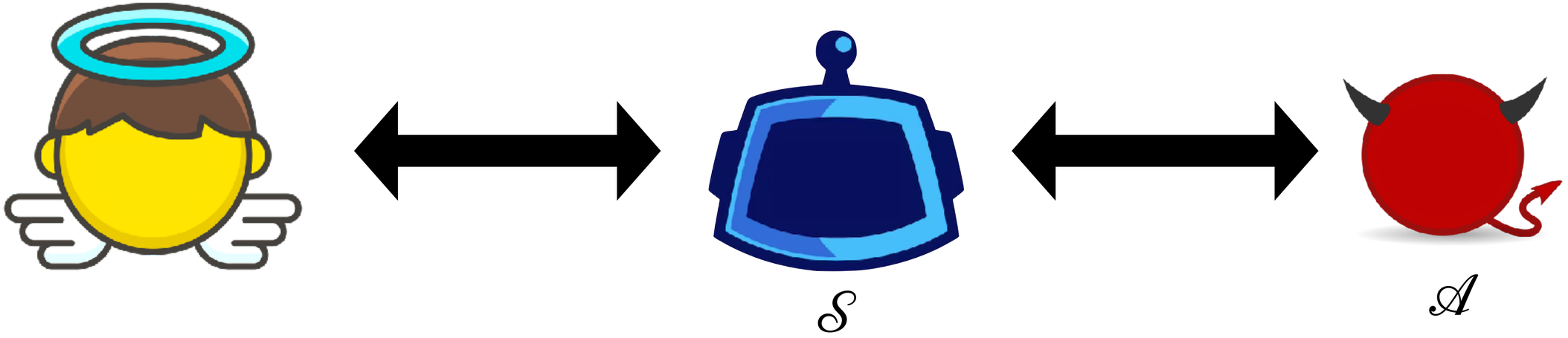
# Typical Proof of Malicious Security




$\mathcal{S}$ tricks $\mathcal{A}$ into thinking it is in the real-world protocol

… and then outputs whatever $\mathcal{A}$ outputs

# Typical Proof of Malicious Security

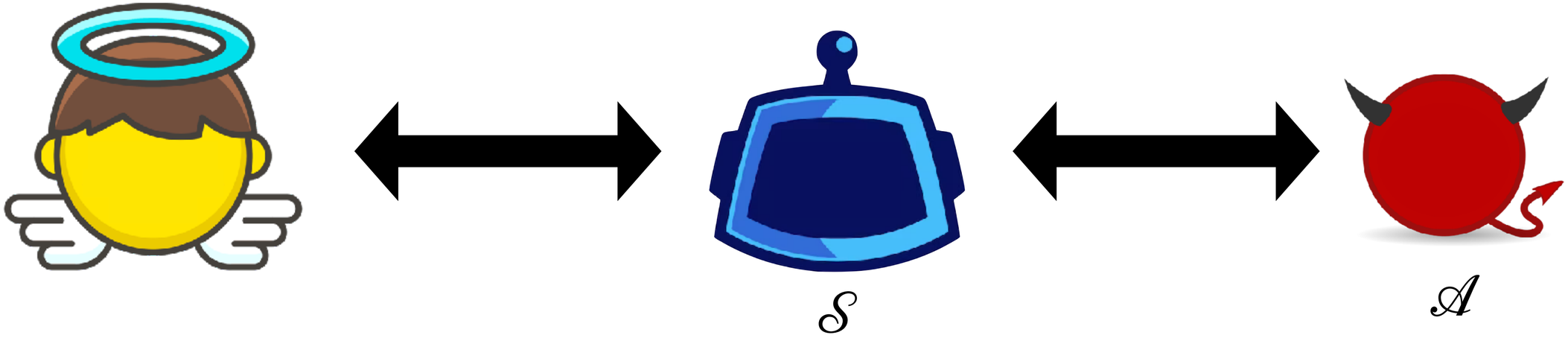



$\mathcal{S}$ tricks $\mathcal{A}$ into thinking it is in the real-world protocol

… and then outputs whatever $\mathcal{A}$ outputs

What does this have to do with preventing attacks?

# Typical Proof of Malicious Security




$\mathcal{S}$ tricks $\mathcal{A}$ into thinking it is in the real-world protocol

… and then outputs whatever $\mathcal{A}$ outputs

What does this have to do with preventing attacks?

Intuition: It is only possible for $\mathcal{S}$ to deceive $\mathcal{A}$ if the protocol restricts $\mathcal{A}$'s behavior in certain ways

# Today's objectives

Define malicious security

Introduce notion of fairness

Introduce notion of abort

Prove a contrived protocol is secure in the malicious model