Multiparty computations

Stefan Dziembowski

http://www.mimuw.edu.pl/˜std

Instytute of Informatics

Warsaw University

Historical note

Multiparty Computations.

Started by:

Yao *Protocols for secure computations* (1982).

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

Plan

- Examples of problems that we want to solve.

Plan

- Examples of problems that we want to solve.
- Definitions and models.

Plan

- Examples of problems that we want to solve.
- Definitions and models.
- The fundamental results.

Plan

- Examples of problems that we want to solve.
- Definitions and models.
- The fundamental results.
- Some implementations.

Plan

- Examples of problems that we want to solve.
- Definitions and models.
- The fundamental results.
- Some implementations.
- Directions for the future research.

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:

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

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:

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

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:

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

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 1 – more formally

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

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

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 $\{P_1, \ldots, P_n\}$ 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

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

Definitions and models

A protocol

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

$$f : D_1 \times \cdots \times D_n \to C_1 \times \cdots \times C_n$$

A protocol

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

$$f : D_1 \times \cdots \times D_n \to C_1 \times \cdots \times C_n$$

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

A protocol

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

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

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

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

A protocol

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

$$f : D_1 \times \cdots \times D_n \to C_1 \times \cdots \times C_n$$

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

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

Goal: construct a protocol Π_f such that as a result each player P_i learns the value of y_i.

A protocol

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

$$f : D_1 \times \cdots \times D_n \to C_1 \times \cdots \times C_n$$

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

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

Goal: construct a protocol Π_f such that as a result each player P_i learns the value of y_i.

Moreover, the protocol should be secure!.

The adversary

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

The adversary

Some of the players $P_1, \ldots, P_n$ 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 $P_1, \ldots, P_n$ 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 $P_1, \ldots, P_n$ 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 Turing machine.

Active/pasive adversary

Once a player P_i gets corrupted the adversary takes a full control over him. Depending on the model it means one of the following.

Pasive case He gets access to all internal data of P_i and all the messages that come to P_i.

Active/pasive adversary

Once a player P_i gets corrupted the adversary takes a full control over him. Depending on the model it means one of the following.

Pasive case He gets access to all internal data of P_i and all the messages that come to P_i.

Active case He also gets a full control over the actions of P_i.

Active/pasive adversary

Once a player P_i gets corrupted the adversary takes a full control over him. Depending on the model it means one of the following.

Pasive case He gets access to all internal data of P_i and all the messages that come to P_i.

Active case He also gets a full control over the actions of P_i.

The pasive model is less practical

Motivation for pasive case

methodological reasons A pasively-secure protocol can often be upgreaded to an actively secure one.

Motivation for pasive case

methodological reasons A pasively-secure protocol can often be upgreaded to an actively secure one.

practical reasons In some applications pasive security is better than no security.

The power of the adversary

Depending on the settings the adversary my be assumed to have

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

The power of the adversary

Depending on the settings the adversary my be assumed to have

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 settings the adversary my be assumed to have

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).

Un-proven assumptions

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

In the other cases there exist proofs without any un-proven assumptions.

Possible corruptions

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

Possible corruptions

In most of the settings we hae 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 theshold adversary.

Possible corruptions

In most of the settings we hae 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 theshold adversary.

More generally: we can specify an adversary structure, i.e.: a familly $\mathcal{F}$ of subsets of the set $\{P_1, \ldots, P_n\}$ that the adversary can corrupt.

Example of an adversary structure

Suppose we have a group of 10 politicians and 10 scientists.
The adversary can corrupt at most 4 politicians and 1 scientist.

Adaptive adversary

There are two more options to consider.

- The adversary has to decide whom he corrupts before the execution starts.

Adaptive adversary

There are two more options to consider.

- The adversary has to decide whom he corrupts before the execution starts.
- The adversary corrupt the players adaptively one-by-obe qduring the execution of the protocol.

The communication channels

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

computational setting the adversary can evesdrop the communicaiton between the parties.

information-theoretic setting the cannels are secure.

We sometimes also assume an existence of a <u>broadcast channel</u> (available to every player).

Security — informally

Informally speaking security consists of 2 requirements:

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

Security — informally

Informally speaking security consists of 2 requirements:

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

Correctness The adverary cannot influence the outputs of the honest players (in other way than by subsituting the inputs of the corrupted players).

Example

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

Example

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

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

Ideal model

We define an <u>ideal</u> model in which we assume an existance of an uncorruptable <u>trusted party T</u>. In the ideal model the computation is done by T:

Ideal model

We define an ideal model in which we assume an existance of an uncorruptable trusted party T.

In the ideal model the computation is done by T:

1. the players send their inputs $x_1, \ldots, x_n$ to T

Ideal model

We define an ideal model in which we assume an existance of an uncorruptable trusted party T.

In the ideal model the computation is done by T:

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

Ideal model

We define an ideal model in which we assume an existance of an uncorruptable trusted party T. In the ideal model the computation is done by T:

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

Definition of security

We will say that

the protocol Π securely computes f

if (informally):

whatever the adversary can achieve attacking Π he can also achieve in the ideal-model.

For details see:

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 woth 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 woth 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 woth probability 1, or
- allow for imperfect security — i.e. there is a negligible probability of error.

  More precisely, we introduce a security paramter k and require that:

  > for any c the probability of error is smaler 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 phaes.

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 phaes.

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

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

Example of a randomized function

Recall the coin-tossing example. In this case Alice and Bob need to compute a function f that

Example of a randomized function

Recall the coin-tossing example. In this case Alice and Bob need to compute a function f that

- takes no input from Alice and Bob, and takes a random input $r \in \{0, 1\}$.

Example of a randomized function

Recall the coin-tossing example. In this case Alice and Bob need to compute a function f that

- takes no input from Alice and Bob, and takes a random input $r \in \{0, 1\}$.
- returns r.

The fundamental results

The fundamental question

For a arbitrary (efficiently computable)

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

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

The fundamental question

For a arbitrary (efficiently computable)

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

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	pasive	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 interupt the execution at any time :-(

Information-theoretic setting

In a model without a broadcast channel:

number t of corrupted players	pasive	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.

Historical remark

Goldreich, Micali, Widgerson *How to play any mental game — a completenes 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:

$\boxed{\mathcal{Q}_2}$ 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:

$\boxed{\mathcal{Q}_2}$ For every $A, B \in \mathcal{F}$ we have $A \cup B \neq \mathcal{P}$.

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

$\boxed{\mathcal{Q}_3}$ 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:

$\boxed{\mathcal{Q}_2}$ For every $A, B \in \mathcal{F}$ we have $A \cup B \neq \mathcal{P}$.

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

$\boxed{\mathcal{Q}_3}$ For every $A, B, C \in \mathcal{F}$ we have $A \cup B \cup C \neq \mathcal{P}$.

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

Why do we need the majority

We are going to presesnt the proof that in the information-theoretic setting it is not possible to securely compute $f(a, b) = a \wedge b$.

Suppose we have a protocol Π for Alice and Bob that securely computes f. For simplicity, assume it works with zero-error.

The transcript

	Alice		Bob
input	a		b
random input	r_A		r_B
		$m_{A1} \rightarrow$	
		$\leftarrow m_{B1}$	
		$\vdots$	
	y		y

The transcript

	Alice		Bob
input	a		b
random input	r_A		r_B
		$m_{A1} \rightarrow$	
		$\leftarrow m_{B1}$	
		$\vdots$	
	y		y

The transcript

	Alice		Bob
input	a		b
random input	r_A		r_B
		$m_{A1} \rightarrow$	
		$\leftarrow m_{B1}$	
		$\vdots$	
	y		y

The transcript

	Alice		Bob
input	a		b
random input	r_A		r_B
		$m_{A1} \rightarrow$	
		$\leftarrow m_{B1}$	
		$\vdots$	
	y		y

The transcript

	Alice		Bob
input	a		b
random input	r_A		r_B
		$m_{A1} \rightarrow$	
		$\leftarrow m_{B1}$	
		$\vdots$	
output	y		y

The transcript

	Alice		Bob
input	a		b
random input	r_A		r_B
		$m_{A1} \rightarrow$	
		$\leftarrow m_{B1}$	
		$\vdots$	
output	y		y

We define the transcript of the execution of a protocol Π as $T := (m_{A1}, m_{B1}, m_{A2}, \ldots, y)$.

Finalizing the proof

Def. We say that a transcript T is consistent with $a = x$ if it could result from an execution of Π with $a = x$.

Suppose $a = 0$. In this case $y = f(a, b) = 0$. Therefore

- If $b = 0$ then Bob should have no information about b. Thus T has to be consistent both with $a = 0$ and $a = 1$.
- If $b = 1$ then T has to be consistent only with $a = 1$ (otherwise $y = 0$ with $a = b = 1$).

Alice can check which is the case, since she has got an infinite computing power. So, she can learn b.

Some remarks on the proof

The proof works only if we assume that Π works with no error.

However, it can be generalized for the case of imperfect security.

Some remarks on the proof

The proof works only if we assume that Π works with no error.

However, it can be generalized for the case of imperfect security.

It can also be generalized for the multi-player case: it is enogh to assume that there exists two sets A and B in the advesary structure, such that $A \cup B = \mathcal{P}$.

Some remarks on the proof

The proof works only if we assume that Π works with no error.

However, it can be generalized for the case of imperfect security.

It can also be generalized for the multi-player case: it is enogh to assume that there exists two sets A and B in the advesary structure, such that $A \cup B = \mathcal{P}$.

Finally, note that the impossibility proof assumes a passive adversary only.

Implementations

Assumptions

For simplicity we assume that

- the function f is deterministic and
- the output of all the players is the same.

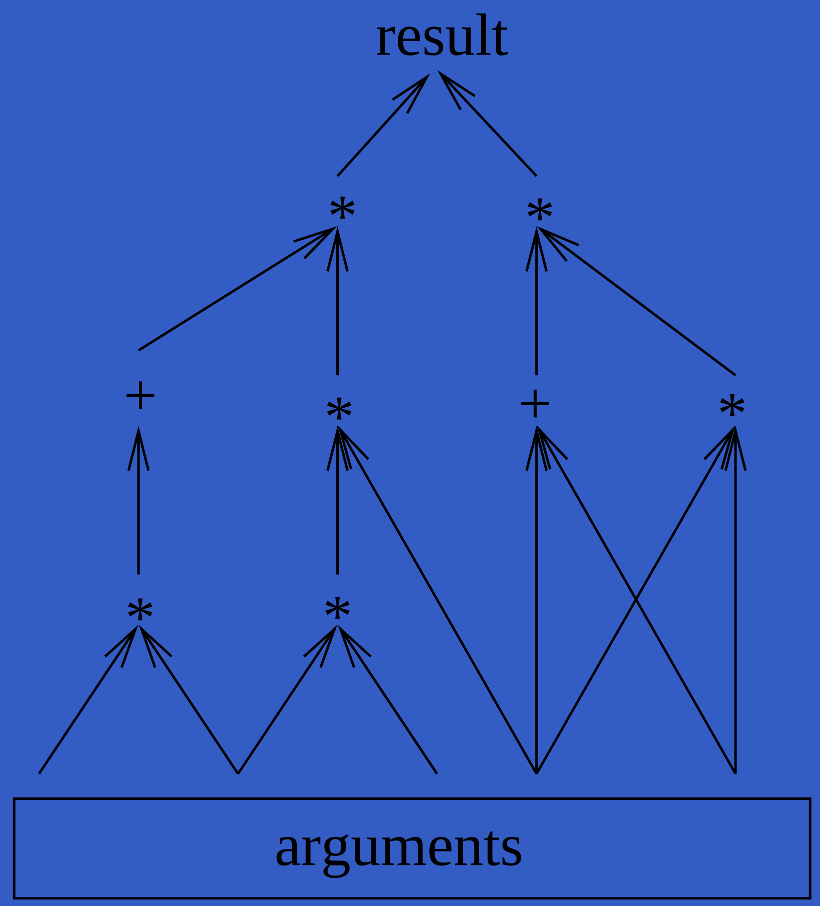
Assumptions

For simplicity we assume that

- the function f is deterministic and
- the output of all the players is the same.

We assume that the inputs of the parties come from some finite field F. The function f is represented as a Boolean circuit over F.

An arithmetic circuit — an example



Sharing a secret

We will use a protocol for secret sharing.

Such a protocol enables each player to share a secret value s among other players.

Sharing a secret

We will use a protocol for secret sharing.

Such a protocol enables each player to share a secret value s among other players.

As a result each player P_j gets some message s_j such that

- A small set of players gets no information about s.
- A big set of players can reconstruct s.

A general paradigm for MPC [1/2]

- Each player P_i secret-shares his input s_i.

A general paradigm for MPC [1/2]

- Each player P_i secret-shares his input s_i.
- The players evaluate the circuit gate-by-gete, maintaining the following invariant:

A general paradigm for MPC [1/2]

- Each player P_i secret-shares his input s_i.
- The players evaluate the circuit gate-by-gete, maintaining the following invariant:

After evaluation of each gate the player hold valid shares of the result

A general paradigm for MPC [2/2]

So, what we need to show is

- how to share and reconstruct a secret
- how to add shared secrets (this is usually easy)
- how to multiply shared secrets (this is harder!).

Computational setting

We first examine the computational setting. The idea is the following. We proceed in two stages:

- We design a pasively secure protocol.
- We upgrade it to active security.

We will examine only pasive two-party case.

Oblivious transfer

The basic tool is a protocol for

1-out-of-k-oblivious transfer

executed between 2 players: sender S and receiver R,

Oblivious transfer

The basic tool is a protocol for

1-out-of-k-oblivious transfer

executed between 2 players: sender S and receiver R, :

- Sender's input is a sequence $x_1, \ldots, x_k$ of elements from some fixed domain.
- Receiver's input is a number $i \in \{1, \ldots, k\}$.

Oblivious transfer

The basic tool is a protocol for

1-out-of-k-oblivious transfer

executed between 2 players: sender S and receiver R, :

- Sender's input is a sequence $x_1, \ldots, x_k$ of elements from some fixed domain.
- Receiver's input is a number $i \in \{1, \ldots, k\}$.
- Sender's output is empty
- Receiver's output is equal to x_i.

Oblivious transfer

The basic tool is a protocol for

1-out-of-k-oblivious transfer

executed between 2 players: sender S and receiver R, :

- Sender's input is a sequence $x_1, \ldots, x_k$ of elements from some fixed domain.
- Receiver's input is a number $i \in \{1, \ldots, k\}$.
- Sender's output is empty
- Receiver's output is equal to x_i.

Let's denote the reciever's output with $\mathrm{OT}_1^k(x_1, \ldots, x_k, i)$.

Example: computing conjunction

Using a protocol for oblivious transfer on can construct a protocol for computing $f(a, b) = a \wedge b$.

Simply:

Example: computing conjunction

Using a protocol for oblivious transfer on can construct a protocol for computing $f(a,b) = a \wedge b$.
Simply:

- Alice acts as the sender with bits $(0, a)$, Bob acts as the receiver with $i = b$.

Example: computing conjunction

Using a protocol for oblivious transfer on can construct a protocol for computing $f(a,b) = a \wedge b$.
Simply:

- Alice acts as the sender with bits $(0, a)$, Bob acts as the receiver with $i = b$.
- Bob sends the bit that he received to Alice. Both anounce it as the result.

Example: computing conjunction

Using a protocol for oblivious transfer on can construct a protocol for computing $f(a, b) = a \wedge b$.
Simply:

- Alice acts as the sender with bits $(0, a)$, Bob acts as the receiver with $i = b$.
- Bob sends the bit that he received to Alice. Both anounce it as the result.

It is easy to see that it works, because

$$\text{OT}((0, a), b) = a \wedge b.$$

Historical note

Oblivious transfer was introduced in:

M.O. Rabin *How to exchange secrets by Oblivious Transfer* (1981)

Historical note

Oblivious transfer was introduced in:

M.O. Rabin *How to exchange secrets by Oblivious Transfer* (1981)

The definition was different, but equivalent.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

- The sender genereates an RSA pair (e, d) and sends e to Bob.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

- The sender genereates an RSA pair (e, d) and sends e to Bob.
- The receiver prepares k ciphertexts $c_1, \ldots, c_k$.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

- The sender genereates an RSA pair (e, d) and sends e to Bob.
- The receiver prepares k ciphertexts $c_1, \ldots, c_k$.
  - the ciphertext for $c_i = E(e, m)$, where m is selected randomly by the receiver.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

- The sender genereates an RSA pair (e, d) and sends e to Bob.
- The receiver prepares k ciphertexts $c_1, \ldots, c_k$.
  - the ciphertext for $c_i = E(e, m)$, where m is selected randomly by the receiver.
  - all the other ciphertexts are completely random.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

- The sender genereates an RSA pair (e, d) and sends e to Bob.
- The receiver prepares k ciphertexts $c_1, \ldots, c_k$.
  - the ciphertext for $c_i = E(e, m)$, where m is selected randomly by the receiver.
  - all the other ciphertexts are completely random.

The receiver sends $c_1, \ldots, c_k$ to the sender.

How to implement OT

$(x_1, \ldots, x_k)$ — input of the receiver, i — input of the sender.

- The sender genereates an RSA pair (e, d) and sends e to Bob.
- The receiver prepares k ciphertexts $c_1, \ldots, c_k$.
  - the ciphertext for $c_i = E(e, m)$, where m is selected randomly by the receiver.
  - all the other ciphertexts are completely random.

  The receiver sends $c_1, \ldots, c_k$ to the sender.
- The sender decrypts all the ciphertexts. Let $m_1, \ldots, m_k$ be the LSB of the plaintexts. He sends $m_1 + x_1, \ldots, m_k + x_k$ to the receiver.
- The receiver decrypts m_i.

Secret sharing

We are now going to implement 2-party computation. Assume field $F = Z_2$. We will use a very simple secret-sharing.

Secret sharing

We are now going to implement 2-party computation. Assume field $F = Z_2$. We will use a very simple secret-sharing.

To share a secret bit s create a random pair (s_A, s_B) such that

$$s_A + s_B = s.$$

Send s_A to Alice and s_B to Bob.

Secret sharing

We are now going to implement 2-party computation. Assume field $F = Z_2$. We will use a very simple secret-sharing.

To share a secret bit s create a random pair (s_A, s_B) such that

$$s_A + s_B = s.$$

Send s_A to Alice and s_B to Bob.

Reconstruciton is trivial.

Addition

Handling addition is also easy.

Addition

Handling addition is also easy.

If:

- x is shared with (x_A, x_B) and
- y is shared with (y_A, y_B)

then: $(x_A + x_B, y_A + y_B)$ will form a sharing of $x + y$.

Multiplication [1/2]

x is shared with (x_A, x_B),
y is shared with (y_A, y_B).

We want to obtain a sharing of $z = xy$.

Multiplication [1/2]

x is shared with (x_A, x_B),
y is shared with (y_A, y_B).

We want to obtain a sharing of $z = xy$.

1. Alice selects a random bit c. This will be her share. Bob needs to calculate $c + z$.

Multiplication [2/2]

2. Alice and Bob engage in 1-out-of-4 oblivious transfer. Alice acts as the sender with the input

$$(\overbrace{c + x_1 y_1}^{1}, \overbrace{c + x_1(y_1+1)}^{2}, \overbrace{c + (x_1+1)y_1}^{3}, \overbrace{c + (x_1+1)(y_1+1)}^{4}).$$

Bob is the receiver, with input

$$(1 + 2x_2 + y_2) = \begin{cases} 1 & \text{if } (x_2, y_2) = (0,0) \\ 2 & \text{if } (x_2, y_2) = (0,1) \\ 3 & \text{if } (x_2, y_2) = (1,0) \\ 4 & \text{if } (x_2, y_2) = (1,1) \end{cases}$$

Multiplication [2/2]

2. Alice and Bob engage in 1-out-of-4 oblivious transfer. Alice acts as the sender with the input

$$(\overbrace{c + x_1 y_1}^{1}, \overbrace{c + x_1(y_1+1)}^{2}, \overbrace{c + (x_1+1)y_1}^{3}, \overbrace{c + (x_1+1)(y_1+1)}^{4}).$$

Bob is the receiver, with input

$$(1 + 2x_2 + y_2) = \begin{cases} 1 & \text{if } (x_2, y_2) = (0,0) \\ 2 & \text{if } (x_2, y_2) = (0,1) \\ 3 & \text{if } (x_2, y_2) = (1,0) \\ 4 & \text{if } (x_2, y_2) = (1,1) \end{cases}$$

Therefore Bob's output is equal to: $c + (x_1 + x_2)(y_1 + y_2) = c + xy$.

Active secuirty

So, how to upgrade a pasively-secure protocol to an adaptively-secure one?

Active secuirty

So, how to upgrade a pasively-secure protocol to an adaptively-secure one?

We will not show the details. In general the idea is as follows.

Active secuirty

So, how to upgrade a pasively-secure protocol to an adaptively-secure one?

We will not show the details. In general the idea is as follows.

- The players commit to their inputs.

Active secuirty

So, how to upgrade a pasively-secure protocol to an adaptively-secure one?

We will not show the details. In general the idea is as follows.

- The players commit to their inputs.
- They prove to each other in zero-knowledge that they execute the protocol correctly.

Commitment schemes — informally

A commitment scheme is a protocol between alice and Bob. Informally speking it works like box with a lock. It consists of two phases. Initially Alice holds a secret s.

commiting Alice puts s into a box. She locks it with the key. She keeps the key and sends the box to Bob.

Commitment schemes — informally

A commitment scheme is a protocol between alice and Bob. Informally speking it works like box with a lock. It consists of two phases. Initially Alice holds a secret s.

- **commiting** Alice puts s into a box. She locks it with the key. She keeps the key and sends the box to Bob.

  Note: Alice cannot modify s (the commitment is binding).

  Bob has no idea about s (hiding)

Commitment schemes — informally

A commitment scheme is a protocol between alice and Bob. Informally speking it works like box with a lock. It consists of two phases. Initially Alice holds a secret s.

- **commiting** Alice puts s into a box. She locks it with the key. She keeps the key and sends the box to Bob.

  Note: Alice cannot modify s (the commitment is binding).

  Bob has no idea about s (hiding)

- **opening** Alice sends the key to Bob. Bob opens the box.

Commitment schemes — more formal

More formally we can define a commitment scheme as an on-line process between Alice and Bob.

commiting Alice has an input s. Bob has no input. The trusted party takes s and stores it. She gives no output.

Commitment schemes — more formal

More formally we can define a commitment scheme as an on-line process between Alice and Bob.

commiting Alice has an input s. Bob has no input. The trusted party takes s and stores it. She gives no output.

opening Alice has an input bit x. Bob has no input.

If $x = 1$ then the trusted party gives s to Bob. Otherwise she gives „?” to Bob.

Implementing a commitment scheme

Alice's secret bit: s

commiting Alice chooses

- an RSA key-pair (e, d),
- a random message m, such that $\mathrm{LSB}(m) = s$.

and sends $(e, E(e, m))$ to Bob.

Implementing a commitment scheme

Alice's secret bit: s

commiting Alice chooses

- an RSA key-pair (e, d),
- a random message m, such that $\text{LSB}(m) = s$.

and sends $(e, E(e, m))$ to Bob.

opening Alice sends m to Bob. Bob computes $\text{LSB}(m)$.

Coin tossing over a phone

To illustrate the use of commitment schemes we show the following protocol for coin-tossing.

1. Alice chooses a random bit $a \in \{0, 1\}$. Bob chooses a random bit $a \in \{0, 1\}$.

Coin tossing over a phone

To illustrate the use of commitment schemes we show the following protocol for coin-tossing.

1. Alice chooses a random bit $a \in \{0,1\}$. Bob chooses a random bit $a \in \{0,1\}$.
2. Alice commits to a.

Coin tossing over a phone

To illustrate the use of commitment schemes we show the following protocol for coin-tossing.

1. Alice chooses a random bit $a \in \{0, 1\}$. Bob chooses a random bit $a \in \{0, 1\}$.
2. Alice commits to a.
3. Bob sends b to Alice.

Coin tossing over a phone

To illustrate the use of commitment schemes we show the following protocol for coin-tossing.

1. Alice chooses a random bit $a \in \{0,1\}$. Bob chooses a random bit $a \in \{0,1\}$.
2. Alice commits to a.
3. Bob sends b to Alice.
4. Alice opens a commitment. The result is $a + b$.

Coin tossing over a phone

To illustrate the use of commitment schemes we show the following protocol for coin-tossing.

1. Alice chooses a random bit $a \in \{0, 1\}$. Bob chooses a random bit $a \in \{0, 1\}$.
2. Alice commits to a.
3. Bob sends b to Alice.
4. Alice opens a commitment. The result is $a + b$.

If Alice refuses to open the commitment then she looses.

IT-setting, pasive adversary

We will now show an MPC protocol that works in the IT-setting, with pasive adversary.

IT-setting, pasive adversary

We will now show an MPC protocol that works in the IT-setting, with pasive adversary.

Suppose the adversary can corrupt at most $t < n/2$ players.

IT-setting, pasive adversary

We will now show an MPC protocol that works in the IT-setting, with pasive adversary.

Suppose the adversary can corrupt at most $t < n/2$ players.

For the secret sharing we will use the idea of:

Shamir *How to Share a Secret* (1979)

Shamir's secret sharing [1/2]

In order to share a secret $s \in F$ among players $P_1, \ldots, P_n$

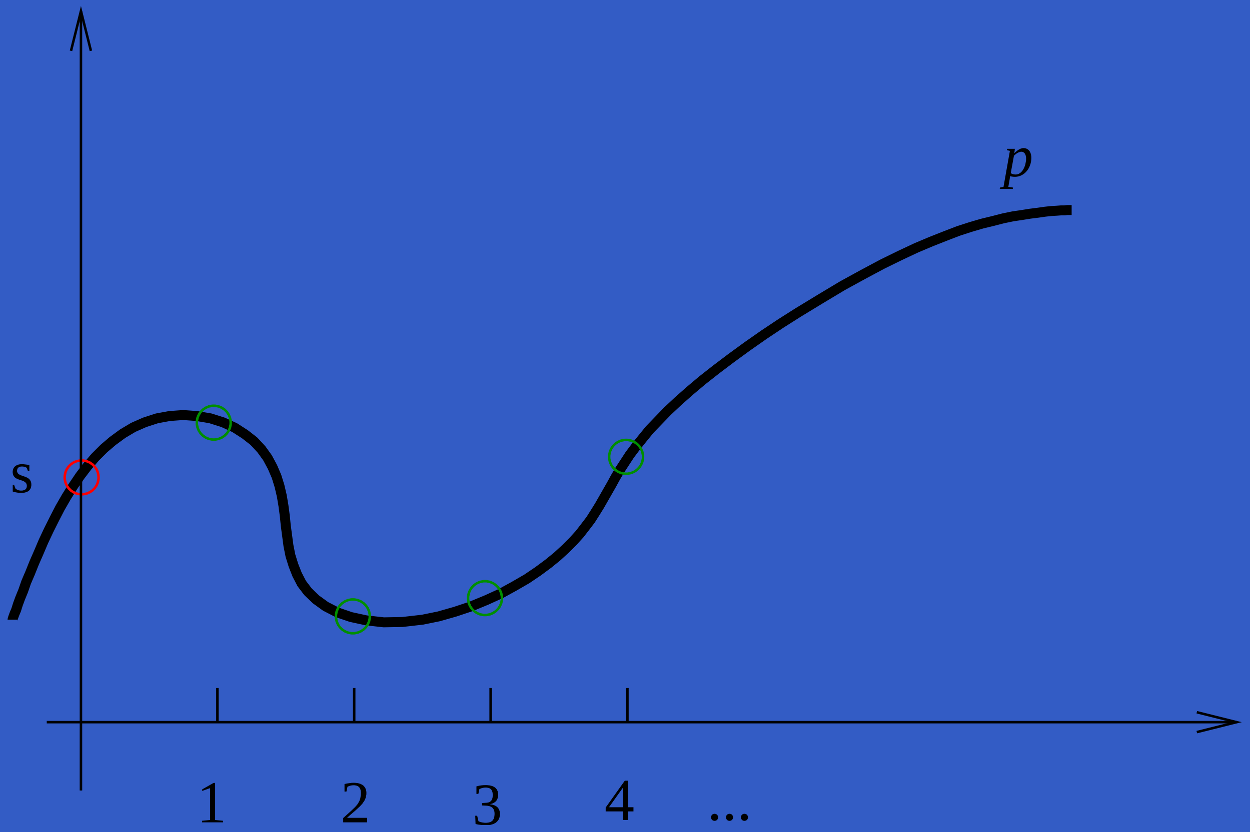
- choose a random polynomial p of degree t, such that $p(0) = s$,

Shamir's secret sharing [1/2]

In order to share a secret $s \in F$ among players $P_1, \ldots, P_n$

- choose a random polynomial p of degree t, such that $p(0) = s$,
- send each $p(i)$ to P_i.

Shamir's secret sharing — intuition



Shamir's secret sharing [2/2]

Now, observe that

- Up to t players have no information about s.
- Any set of $t+1$ player can reconstruct s by interpolating p. In particular if $t = n-1$ then there exist values $r_1, \ldots, r_n$ such that

$$p(0) = \sum_{i=1}^{n} r_i p(i).$$

$(r_1, \ldots, r_n)$ is called a recombination vector.

Addition

Handling addition is trivial. Suppose:

- $x_1, \ldots, x_n$ are shares of x (let p_x be the corresponding polynomial)
- $y_1, \ldots, y_n$ are shares of y (let p_y be the corresponding polynomial)

Addition

Handling addition is trivial. Suppose:

- $x_1, \ldots, x_n$ are shares of x (let p_x be the corresponding polynomial)
- $y_1, \ldots, y_n$ are shares of y (let p_y be the corresponding polynomial)

Ask each P_i to add his shares locally: set $z_i := x_i + y_i$.

Addition

Handling addition is trivial. Suppose:

- $x_1, \ldots, x_n$ are shares of x (let p_x be the corresponding polynomial)
- $y_1, \ldots, y_n$ are shares of y (let p_y be the corresponding polynomial)

Ask each P_i to add his shares locally: set $z_i := x_i + y_i$.

It is easy to see that $z_1, \ldots, z_n$ lye on a polynomial $p_x + p_y$ and thus they form a sharing of $x + y$.

Multiplication [1/3]

A similar idea doesn't work for multiplication.

Multiplication [1/3]

A similar idea doesn't work for multiplication.

This is because the degree of $p_x \cdot p_y$ is to large $2t$.

Also, $p_x \cdot p_y$ is not random ...

Multiplication [2/3]

Let x and y be shared with $x_1, \ldots, y_n$ and $y_1, \ldots, y_n$. We want to obtain a sharing of $z = xy$.

- Each P_i calculates $z_i = x_i \cdot y_i$.
- Each P_i secret-shares z_i among other players. Let $z_{i1}, \ldots, z_{in}$ be the resulting shares.

	P_1	$\cdots$	P_n
	z_1	$\cdots$	z_n
P_1	z_{11}	$\cdots$	z_{n1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
P_n	z_{1n}	$\cdots$	z_{nn}

Multiplication [3/3]

Each z_i is a linear combination of $z_{i1}, \cdots, z_{in}$.

Multiplication [3/3]

Each z_i is a linear combination of $z_{i1}, \cdots, z_{in}$.

Also, recall that $(z_1, \ldots, z_n) \cdot (r_1, \ldots, t_n) = z$ (where $(r_1, \ldots, t_n)$ is a recombination vector).

Multiplication [3/3]

Each z_i is a linear combination of $z_{i1}, \cdots, z_{in}$.
Also, recall that $(z_1, \ldots, z_n) \cdot (r_1, \ldots, t_n) = z$ (where $(r_1, \ldots, t_n)$ is a recombination vector).
Therefore a result of the multiplication

$$\begin{bmatrix} z_{11} & \cdots & z_{n1} \\ \vdots & \vdots & \vdots \\ z_{1n} & \cdots & z_{nn} \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}$$

will form a valid sharing of z (of degree t).

Multiplication [3/3]

Each z_i is a linear combination of $z_{i1}, \cdots, z_{in}$.
Also, recall that $(z_1, \ldots, z_n) \cdot (r_1, \ldots, t_n) = z$ (where $(r_1, \ldots, t_n)$ is a recombination vector).
Therefore a result of the multiplication

$$\begin{bmatrix} z_{11} & \cdots & z_{n1} \\ \vdots & \vdots & \vdots \\ z_{1n} & \cdots & z_{nn} \end{bmatrix} \cdot \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}$$

will form a valid sharing of z (of degree t). This computation can be done locally!

Active security

For active security we need much more work.

We will not show this protocols here.

We will only present a protocol for achieving commitments in the model where

- the adversary can corrupt at most $t < n/2$ players
- the broadcast channel is available.

This is a protocol of

Directions for the future research

The relevant questions

Currently, it seems that we have answers for most of the fundamental questions, i.e. questions of a type:

is . . . possible at all?

The relevant questions

Currently, it seems that we have answers for most of the fundamental questions, i.e. questions of a type:

is . . . possible at all?

There are of course very important fundamental questions that we cannot answer (esp. those that consider the intractability assumptions). But they appear the be very hard. . .

The relevant questions

Currently, it seems that we have answers for most of the fundamental questions, i.e. questions of a type:

is . . . possible at all?

There are of course very important fundamental questions that we cannot answer (esp. those that consider the intractability assumptions). But they appear the be very hard. . .

What is more important are the question of a type

how efficiently can . . . be done?

Large inputs

Current MPC protocols are efficient in a ”theoretical way”.

Large inputs

Current MPC protocols are efficient in a ”theoretical way”.

What we need are protocols that work efficiently on data of a big size.

Large inputs

Current MPC protocols are efficient in a ”theoretical way”.

What we need are protocols that work efficiently on data of a big size.

Probably one would need to design protocols that for specific pourpouses (not generic ones).

Large inputs

Current MPC protocols are efficient in a ”theoretical way”.

What we need are protocols that work efficiently on data of a big size.

Probably one would need to design protocols that for specific pourpouses (not generic ones).

Sometimes we don’t need to now an exact value of f — an approximation is enough.

Large inputs — an example

Consider a group o hospitals.

Each of them has a huge database.

Large inputs — an example

Consider a group o hospitals.

Each of them has a huge database.

Because of the legal and business reasons the hospitals don't want to reveal their databases to each other.

However, the scientists want to compute some statistical data on those databases.

Currently, this is done by a trusted party.

The hospitals

So, can we solve the problem of the hosptals with the cryptographic methods?

In principle: Yes! Just specify the database query as an arithmetic circuit. Represent the data as in some field. Perform a generic protocol for MPC.

The hospitals

So, can we solve the problem of the hosptals with the cryptographic methods?

In principle: Yes! Just specify the database query as an arithmetic circuit. Represent the data as in some field. Perform a generic protocol for MPC.

In practice: No :-(This approach is inefficient. . .

Results so far

Receipt-free voting

The efficiency of MPC

On a more theoretical side one can still try to improve the communication complexity of multiparty protocos (or to show lower bounds).

The most expensive operation is the multiplication. Currently, the best results are: (here n is a number of players).

	threshold	security	nb. of field elems.
[HM01]	$t < n/3$	IT-imperfect	$O(n^2)$
[HMP00]	$t < n/3$	IT-perfect	$O(n^3)$
[CDDHR99, Feh00]	$t < n/2$	IT-imperfect	$O(n^4)$
[CDN01]	$t < n/2$	comput.	$O(n^3)$

The efficiency — references

[HM01] Hirt, Maurer *Robustness for Free in Uncoditional Multi-Party Computation* (2001).

[HMP00] Hirt, Maurer, Przydatek *Efficient Secure Multi-Party Computations* (2000)

[CDDHR00] Cramer, Damgård, Dziembowski, Hirt, Rabin *Effifcient Multiparty Computations Secure Against and Adaptive adversary* (1999)

[Feh00] Serge Fehr *Private Communications*

[CDN01] Cramer, Damgård, Nielsen *Multiparty Computations From Threshold Homomorphic Encryption*.