How to Compute Securely

- even if you trust only yourself

Ivan Damgård,

Computer Science dept., Aarhus University

Private Information

PIN codes, the customer database of a company, patient records at the hospital,..

- it's often valuable, must keep it confidential

- but sometimes, combining private information from different sources can provide large added value.

A few examples...

Simple first price Auction

Some participants are bidding for some item. Each bidder has private information: some maximum price he is willing to pay
– but wants to pay as small a price as he can get away with.

Goal: find a winner and a price in a fair way.

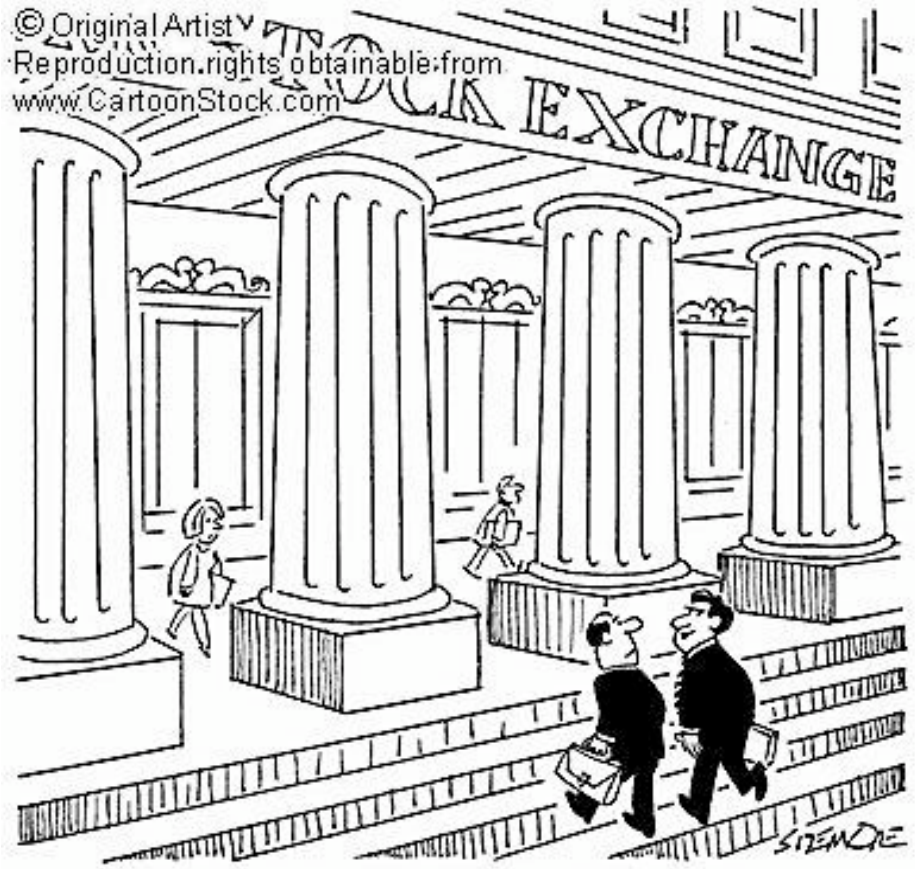
Double Auction – the "Stock Exchange"

Several potential buyers and sellers want to exchange a commodity.

Each seller is willing to sell various quantities, depending on the price he can get. Each buyer will buy various quantities, again depending on the price.

Goal: find a fair market price, given the existing supply and demand

Data like the max price you are willing to pay must be private..



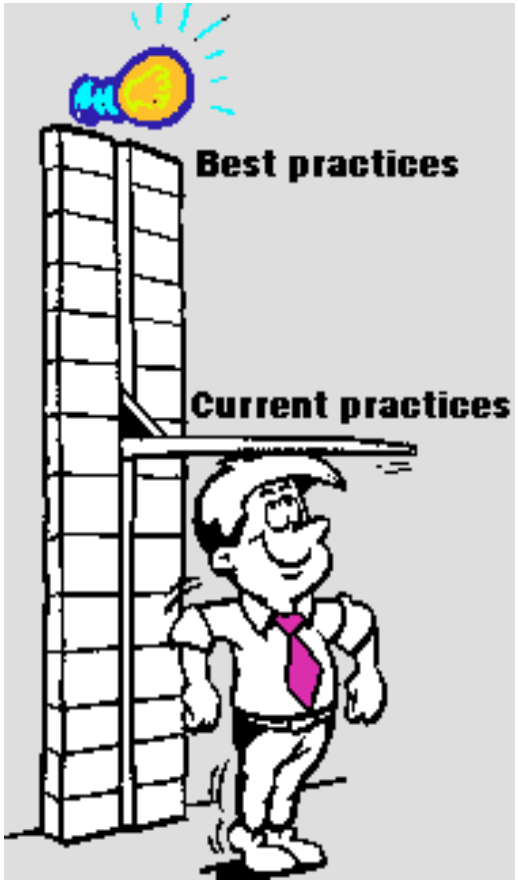
"Ready for another roll of the dice?"

Benchmarking

A number of companies work in the same sector. Each company has data on how their business is running – productions costs, turnover, etc.

Goal: each company wants to find out how well it is doing compared to others

- but without disclosing information to the competitors..



Database privacy

Several different institutions possess different databases with information on individual persons.

Goal: to extract statistics drawing on all databases simultaneously.

- but without breaking the privacy laws by giving someone full acces to all the databases.

A Fundamental Problem

How do we compute the desired results in these scenarios, without giving up on privacy?

One idea: find someone we can all trust, give him all the data and let him compute the result we want.

But finding such a trusted party can be impossible or very expensive.

Can we do without a trusted third party?

This is the *Multiparty Computation Problem*..

The MPC problem

n players P1, P2, …, Pn

Player Pi holds input xi

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

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

We want this to hold, even when some of the players behave adversarially - are corrupted by an *Adversary.*

Potentially Very Useful Tool:
For instance,to implement a trusted mediator in a mechanism.
But beware of differences between crypto and game theory way to model behavior..

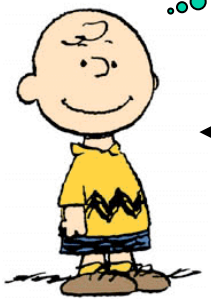
The scenario

Corruption can be *passive*: just observe computation and mess.
Or *active*: take full control

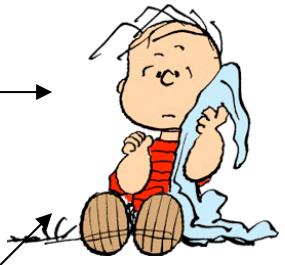
Adv

Corrupt

The players



x1, y1



x2, y2

Inputs, Desired outputs

Communication Network

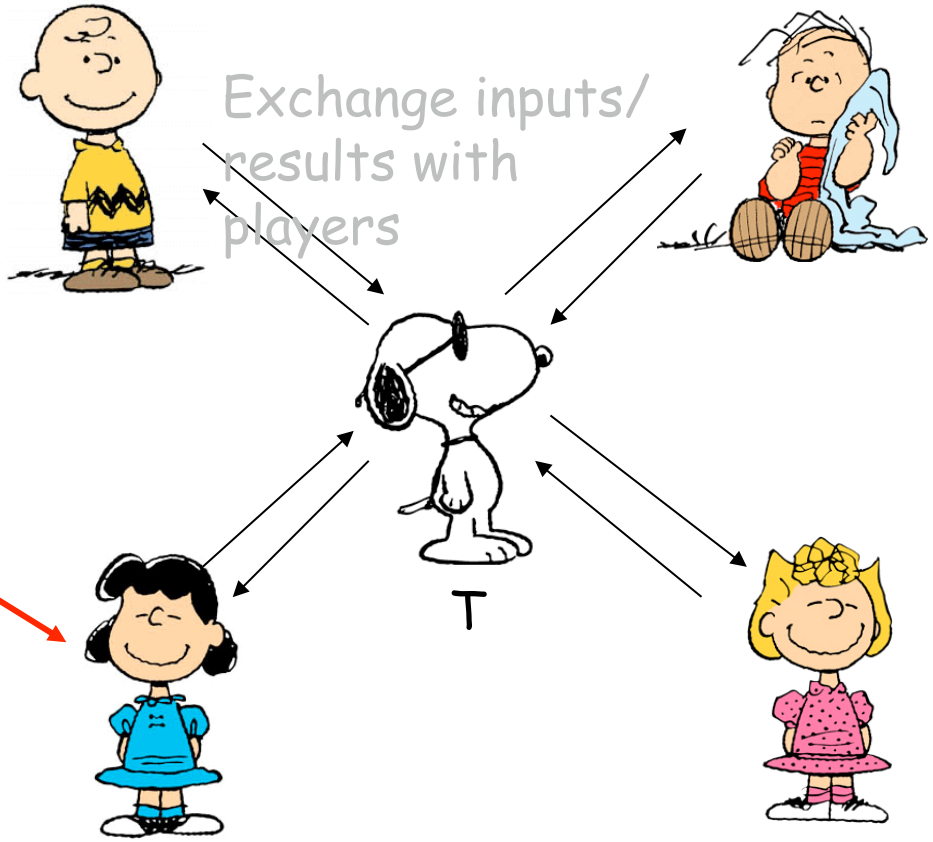
x3,y3

x4,y4

Goal of MPC – a bit more precisely

Adv

Running protocol is *equivalent* to using a trusted (uncorruptible) party T, who gets inputs from players, computes f and returns outputs to the players.



Basic Fact about MPC (Simplified):

Any function that can be computed efficiently, can also be computed securely – long series of results starting in the late 80-ties
[Yao 86], [Goldreich-Micali-Wigderson 87],
[Chaum Damgård Crepeau 88], [Ben-Or Goldwasser Wigderson 88]

- but not always with the same efficiency..

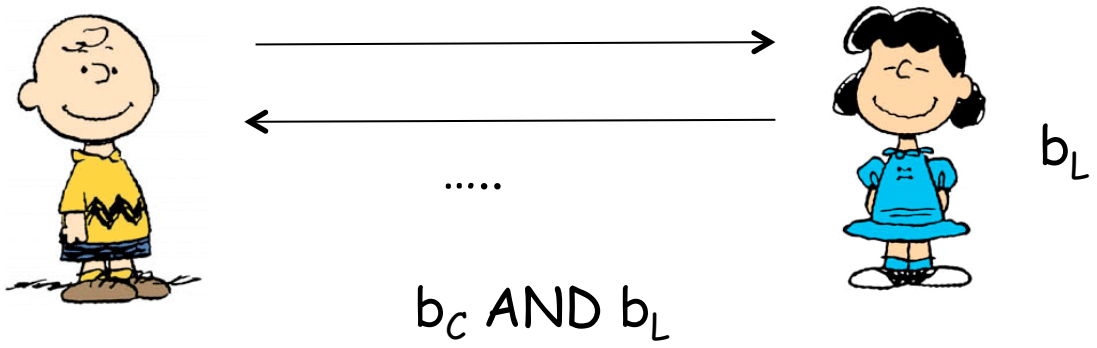
The Hardest Case: all but one player may be corrupt
- Only need to trust yourself.

Often a natural model of trust for real applications – if we have just two parties, this is the only possibility..

But there is a limit to the type of solution we can get…

Computing the AND – mutual interest or not?

Maybe Charlie is interested in Lucy and she in him ….
Or maybe not.
Can they find out without risking embarrassment?
Equivalent to computing the AND of two bits securely



If b_L=0, Lucy already knows the answer is 0, so she must not learn anything new.

➔ If Charlie is supposed to send first message, cannot send anything that gives information on b_C.

➔ Lucy in same situation.

➔ No solution possible?! ☹

Not Really Impossible…

..if we assume bounded computing power – means we can use cryptography.

Say Charlie has a pair of keys
- Public key pkC for encryption - known by all
- Secret key skC for decryption - known only to Charlie

For any message m, have $D_{skC}(E_{pkC}(m)) = m$
➔ Charlie can send his input to Lucy, encrypted under pkC.

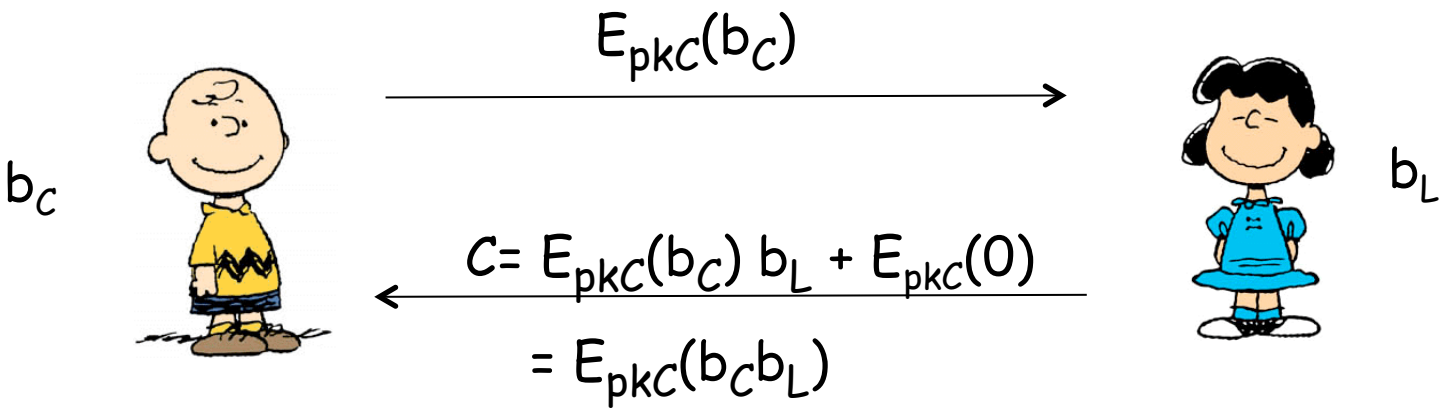
Want more: **homomorphic** encryption:
Ciphertexts are elements in some abelian group determined by the public key, and we have

$$x \, E_{pkC}(a) + E_{pkC}(b) = E_{pkC}(xa+b)$$

For integers x, a and b.

A Solution Using Homomorphic Encryption

Remark: encryption function must be randomized to be secure.
Otherwise, if b is a bit, trivial to find b from $E_{pkC}(b)$



Decrypt to
get b_C AND b_L

Semi-Homomorphic Encryption mod p and Multiparty Computation

(Bendlin, Damgård, Orlandi, Zakarias 2010)

- A weaker notion of homomorphic encryption, several known cryptosystems can be modified to be semi-homomorphic mod a prime p:

[Okamoto-Uchyama98],[Paillier 99],[Damgård-Jurik 01], [Regev 05], [Damgård, Krøigaard, Geisler 09], [Lubashevsky, Palacio, Segev 10], [van Dijk, Gentry, Halevi, Vaikunthanathan 10].

Suppose computing f can be done using T arithmetic operations mod p. Then f can be computed securely using $O(T\, n^2)$ encryption/decryption operations, n the number of players.

Protocol tolerates that up to n-1 of the n players are actively corrupted.

Futhermore..

Can split protocol in off-line and an on-line phase

In off-line phase inputs need not be known, just produce "raw material" for the on-line phase. Computationally heavy stuff, like public-key crypto used only here.

In on-line phase actual computation is done. Uses no crypto, only basic arithmetic mod p. Hence much more efficient.

Preliminary implementation results: for a 65 bit prime p, about 6 msec per secure multiplication, additions negligible.

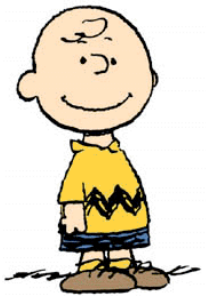
The On-line Phase

A shared representation of numbers mod p

$x = x_C + x_L \bmod p$

Additional data to prevent Lucy from lying about x_L

Similar data set up to prevent Charlie from lying..



Verification key
(a, b_x)

Authentication code
$mL_x = ax_L + b_x \bmod p$

Authentication code
$mC_x = cx_L + d_x \bmod p$

Verification key
(c, d_x)

Data of correct form can be set up in off-line phase

Computing on Shared Numbers

Two numbers x, y with authenticated shares

To add x and y, just add corresponding numbers locally..

Charlie	Lucy
x_C (a, b_x) mC_x	x_L (c, d_x) mL_x
y_C (a, b_y) mC_y	y_L (c, d_y) mL_y
x_C+y_C (a, b_x+b_y) mC_x+mC_y	x_L+y_L (c, d_x+d_y) mL_x+mL_y

Now have a representation of the same form of x+y (all additions mod p).

Multiplication more complicated, requires extra data prepared in off-line phase.

Applications of MPC.

From the SIMAP project, predates CFEM.

A secure double auction for trading contracts for sugar beet production. A nation-wide market for such contracts.

No single trusted auctioneer, multiparty computation is used to compute the result of the auction, based on encrypted bids submitted by the farmers

The protocol executed by 3 parties: the farmers' organization, the company processing the sugar beets (Danisco) and SIMAP.

Application Cont'd

- Shows multiparty computation is practical: 2400 bids, each containing several hundred numbers, results ready in about 30 minutes.
- more later, in Jakob Pagter's talk.
- First implementation used simple protocol assuming at most one corrupted player. Would be more natural to do a 2-party solution.
- By recent results: we are close to being able to do this in practice.

Conclusion

MPC is a very powerful tool for implementing auctions, benchmarking, procurement etc. with improved confidentiality

In many cases, a 2-party solution, or a solution where players only need to trust themselves is the most natural approach.

Even for this most difficult case, MPC is becoming practical.

But we do not know the exact complexities yet. At this point it seems to be much more expensive to compute securely than to just compute. But is this true?