# The Past, Present and Future of Multiparty Computation

**IACR Distingushed lecture, 2006**

Ivan Damgård

BRICS, Århus University

## The MPC problem (as we usually describe it)

n players P1, P2, …, Pn

Player Pi holds input xi

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

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

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

**Generality**:
Virtually any protcol problem can *in principle* be solved using general MPC

# 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

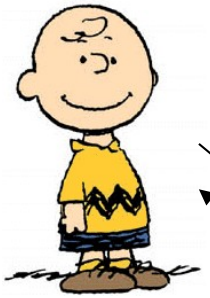
Communication Network

x3, y3

x4,y4

# Goal of MPC – a bit more precisely

Adv



Exchange inputs/results with players

T

Corrupt

Want protocol to be *equivalent* to a trusted (uncorruptible) party T, who gets inputs from players, computes results and returns them to the players.

## Goal of MPC, con't

If this equivalence is for real, then protocol must guarantee any security property that T guarantees, e.g.:

Adv may decide inputs that corrupted players should contribute, but honest players always get correct results based on the inputs contributed.

Adv only learns inputs/outputs of corrupted players.

Many technically different ways to fomalize the meaning of "equivalent"

- Canetti's UC framework
- Pfitzmann/Waidner/Baches BRS framework

But basic intuition remains the same.

## Known Results on general MPC

General flavor of known results: as long as not too much corruption happens, any function can be securely computed. If there is too much, some functions become impossible to handle (usually includes the most interesting ones).

For example, assuming secure point-to-point channels:

- Active, unbounded adversary:
  any function can be securely computed with perfect security iff < n/3 of the players are corrupted.

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

- active, unbounded adversary:
  any function can be securely computed with small error prob. iff < n/2 of the players are corrupted.

Results of [Chaum,Crepeau,Damgaard,'88], [Ben-Or,Goldwasser,Wigderson,'88] [Rabin,Ben-Or,'89], [Hirt,Maurer,'99], [Cramer,Damgaard,Dziembowskie,Hirt,Rabin,'00]

## Known Results, computational security

- Passive, polynomial time adversary:

Assuming one-way trapdoor permutations exist, any function can be securely computed with computational security if number of corrupted players is < n.

- Active, polynomial time adversary:

Assuming one-way trapdoor permutations exist, any function can be securely computed with computational security iff < n/2 corruptions.

Results of [Yao'86], [Chaum, Damgård, Van de Graaf 87] [Goldreich,Micali,Wigderson,'87],[Canetti,Feige,Goldreich,Naor,'96]

## Any big questions left?

Many, if not most major theory questions on MPC answered

(at least) One exception, though:

which functions can be computed securely, with unconditional security and in a constant number of rounds?

## Applications in practice of MPC?

With all these nice general tools around, why are they not used more in practice?

Obvious answer: general MPC protocols are not efficient enough

- But is it that simple?

## My view on applications of MPC

- of course, efficiency is important

But other factors play a role too – perhaps we have the wrong attitude:

”Great theoretical solutions looking for applications”

Maybe time for a different point of view:

”Take the application as point of departure – shape the solution accordingly”

More precisely:

Study a (range of) applications scenarios. Find out exactly what sort of secure computation is needed. What would motivate the players to participate? What behavior are they likely to show?

And design the protocol based on that.

## In the following:

- Some example application scenarios for illustration
- Examples of recent designs and tools that might be useful in the applications.

# The domain we look at:

Applications in ”Information Economy”, more precisely:

- Auctions
- Benchmarking
- Database Privacy

More details to follow..

# Simple first price Auction

Some participants are bidding for some item. Each bidder has 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

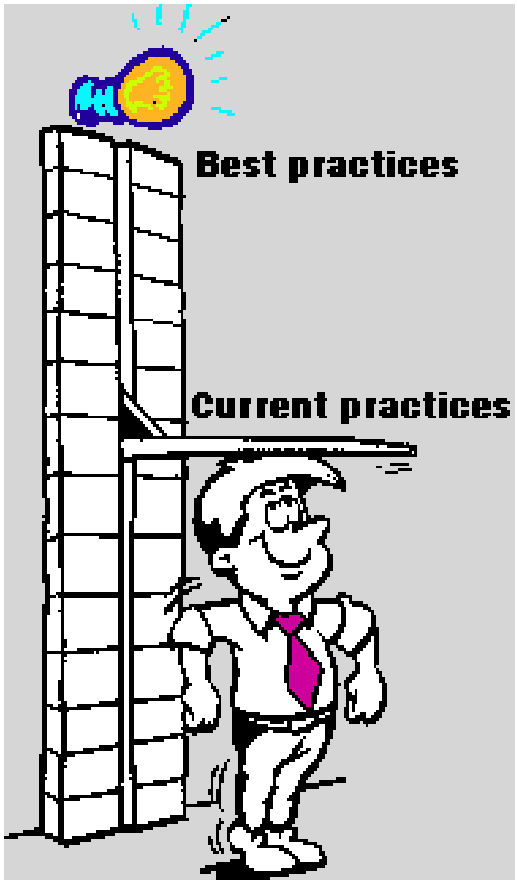
Market clearing price: price per unit where total supply in market = total demand.

“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.



# Database privacy

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

Goal: to extract statictics drawing on all databases simultaneously.

Clear that all applications require correctness and some amount of privacy, and that in principle this can be handled with MPC.

## But exactly what computation is required?

What we found:

Need to do to *integer* arithmetic on *relatively small* numbers. Typically 32 bits enough.

Often addition, multiplication and comparison is sufficient.

Division sometimes needed – but often we can work around.

A concrete example..

## On-line Auctions with submitted maximum bids

Many on-line auctions offer to participants that they can submit a maximum bid. Then the system bids for you, so you don’t have to be on-line all the time.

The maximum is confidential!

If known ahead of time, consequences for other bidders and for auction house.

⇒ Auction house not the best candidate for a trusted party..

## Demands to a protocol solution?

- Same functionality for client as in conventional solutions: just submit your maximum bid, and you don’t have to be involved later.

- Speed is important: the auction runs on-line, cannot wait for a slow protocol to finish.

- Good if solution mimics conventional way to establish trust: ”we are subject to external review and accounting” – so protocol should involve the auction house and an external ”reviewer”.

## Implies Design Goals for Protocol:

- Protocol is built for A (auction house), B( A's accounting company) and C(client).

- C submits securely a number m to A and B, preferably no interaction.

- Later, when public number x is given (the current price), A and B can work together to compute securely whether m >= x

- A or B should not be able to learn information on m by themselves.

- Do not necessarily need security against active cheating, as long as incorrect behavior by A,B can be detected, this may be enough incentive to behave correctly: Cheating is bad for business (cf. Rational crypto).

- Speed more important than compact representation of data (we don't have much secret data around anyway).

## A solution

[Damgård, Krøigaard and Geisler 06, see Eprint soon]

Notation $m = m_0 + 2m_1 + \ldots + 2^{l}m_l$  $x = x_0 + 2x_1 + \ldots + 2^{l}x_l$

C secret-shares each bit $m_i$ in m additively modulo a small prime v,

i.e., $m_i = m_i^A + m_i^B \bmod v$. $[m_i]$ denotes the set of shares in $m_i$

C sends all shares of A to A encrypted under A's public key, same for B.

Now A, B have public number x and $[m_0],\ldots,,[m_l]$

Note that $[x_0 \oplus m_0] = [m_0] + x_0 - 2x_0[m_0]$ - can be computed locally

Define $e_i = 1 + x_i - m_i + (x_{i+1} \oplus m_{i+1}) + \ldots + (x_l \oplus m_l)$  Note that:

$m > x$ iff there exists an i such that $e_i = 0$.

$[e_0], \ldots, [e_l]$ can be computed locally by A and B

Variant of technique by Kolesnikov – our variant uses smaller numbers, just need v > l to avoid overflow

$\Rightarrow$ better efficiency

## The Final Problem

Given $[e_0]$, … ,$[e_l]$, want to detect if we have a zero somewhere.

Assume A has a key pair (pk, sk) for a cryptosystem that is homomorphic modulo v.

Each $e_i = e_i^A + e_i^B \bmod v$. A sends $E_{pk}(e_i^A)$ to B.

B computes $E_{pk}(e_i)$ from $E_{pk}(e_i^A)$ and $e_i^B$.

B randomizes the encryptions and sends them to A in random rotated order.

A decrypts and checks for occurrence of a 0.

**Security**

Passive attacks: straightforward.

Active attacks:

C can submit shares of non-binary values, but behavior of A,B still consistent with *some* legal input

A or B can cause incorrect results, but if encrypted inputs are logged, C can later prove that he was cheated.

## A Suitable Cryptosystem

Choose modulus n= pq such that p-1, q-1 both divisible by primes v and u, where u is about 160 bits.

Generate g,h in Zn* such that ord(g) = uv, ord(h) =u

pk: n,g,h

sk: u

$E_{pk}(m,r) = g^m h^r \bmod n$

To decrypt, raise to power u, get $(g^u)^m \bmod n$

- we just need to check if m=0, so can check if this is 1. General decryption possible also since v is small.

## Performance

We did an implementation in Java with 1024 bit modulus.

Comparison of 16 bit numbers takes 0.28 seconds including all computation and communication. Can save a factor of about 10 using preprocessing.

## Applications with large amounts of data

- such as database applications or some auctions.

Previous representation no good: each l-bit number gets expanded to l full-scale encryptions (= 1000 l bits).

So need more compact ways to handle numbers…

## Shamir Secret Sharing

A Dealer holds a secret value s in Zp*, p > n is a prime, p large enough so that we have room for our numbers (typical cases p about $2^{65}$).

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

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

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

Properties:

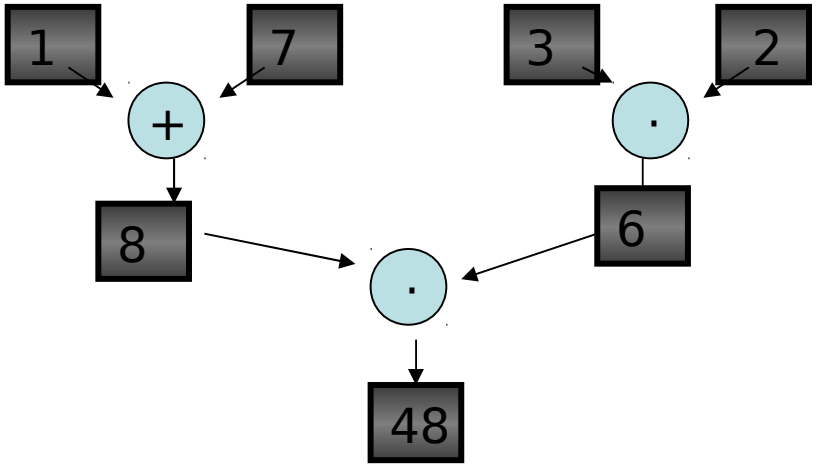
- Any subset of at most t players has no information on s
- Any subset of at least t+1 players can easily compute s – can be done by taking a linear combination of the shares they know.

$\Rightarrow$ If Adversary corrupts at most t players, secure to share secrets this way.

Notation: [s] means s, secret shared

# Computation involving addition and multiplication

- threshold adversary, may corrupt up to t players, t< n/2 (so cannot have n=2, nor t >= n/2 in the following..)




Circuit and inputs given

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

Computing phase: compute new sharings.

Open outputs

Compute new sharings:

[a], [b] -> [a+b] easy, just add shares locally, no communication
[a] public constant v -> [av] easy, multiply shares by v locally.
[a], [b] -> [ab] only a bit harder, requires constant-round communication: multiply shares locally, share results, and take linear combination locally.

More work in multiplication for active security, but still constant round and OK efficiency.

## A Caveat?

We wanted integer addition and multiplication, whereas the protocol comes with add/mult modulo p?

No problem if sizes of numbers can be controlled: just choose p large enough so that no overflow occurs. We found that 65-bit p was OK in the cases we looked at (where inputs were 32 bits or less).

## But what about comparison?

Given [a], [b] want to compute [v] where v is a bit such that v=0 if a< b, and 1 otherwise. - Not trivial!

Could do add/multiply efficiently because all numbers are shared ”in one piece.”

But this also means we have no direct access to the binary representation of a and b $\Rightarrow$ can’t directly use the ideas from earlier in the talk. No small arithmetic circuit over p exists for comparison.

# A solution

[Damgård et al., TCC 06, plus recent work..]

Problem: hard to go from [c] to $[c_0]$, $[c_1]$,..., $[c_n]$ where $[c_i]$ is i'th bit of c.

Observation: much easier go from $[c_0]$, $[c_1]$,..., $[c_n]$ to [c] since Shamir secret sharing is linear - just multiply by 2-powers and add.

Algorithm

1. Generate random shared 0/1 values $[r_0]$,...., $[r_n]$, compute [r],

where $r = r_0 + 2\, r_1 + \ldots + 2^n r_n$ n chosen so r >> a,b

2. Compute and open T= r + a – b (statistically secure since r >> a,b).

3. Now we just need to compare public number T to bit-wise shared number r – same problem as before!

# Improvements?

So far, best protocols for comparison have constant round and use a O(n) secure multiplications to compare n –bit numbers.

Is O(n) multiplications really necessary? It seems we should be able to do better since we have arithmetic available that can handle many bits ”in one go”.

But it’s open!

Another open problem: the compact representation using Shamir-sharing only allows efficient multiplication if t< n/2.

If t >= n/2 we need some form of public-key crypto to do multiplications. How do we do this (really) efficiently?

# Implementation..

Secure, Computing Economy and Trust (SCET), see [Damgård et al. FC 06].

A research project aimed at developing protocols and software to do MPC solutions for auctions and related problems.

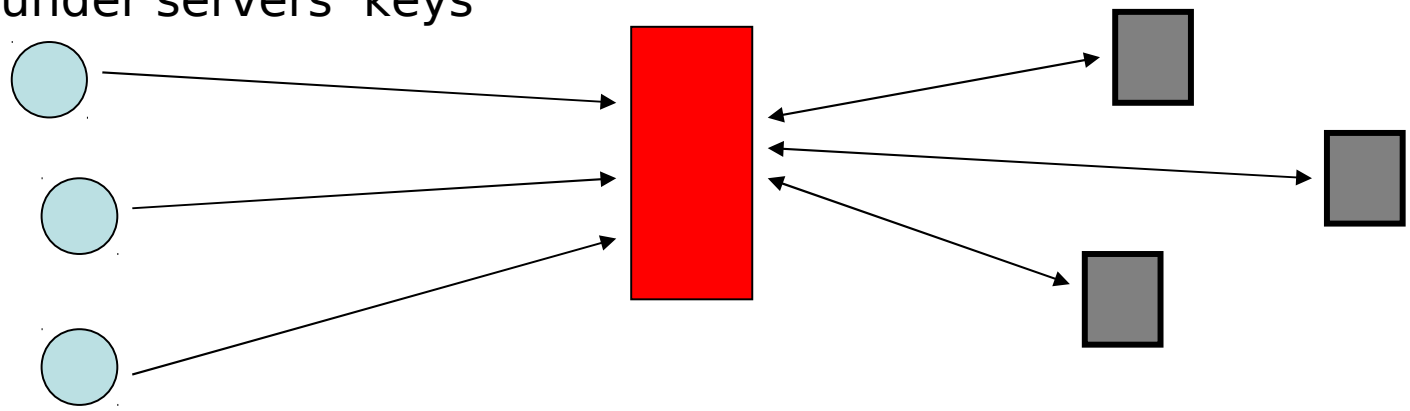
Implementation in C# on .net platform. Current implementation, only passive security, working on the active case..

Set-up:

**Input Clients**
- provide inputs: shares encrypted under servers' keys

**Coordinator**
- handles communication

**Servers**
– perform the actual computation



## Application Scenario

A double auction with 500-1000 input clients, 3-5 servers ($\Rightarrow$ secure if at most 1 resp. 2 corrupt)

Need only additions and 10-12 comparisons to run double auction.

Additions trivial, time for comparisons (seconds):

| (n,t) | Preproc. | | On-line | Total |
|---|---|---|---|---|
| (3,1) | 0.42 | 0.35 | 0.77 | |
| (5,2) | 0.68 | 0.40 | 1.08 | |
| (7,3) | 1.78 | 0.62 | 2.40 | |

Measurements on standard Internet connections, coordinator inside University firewall, 1 server outside using VPN.

More info and downloadable demo:

http://www.sikkerhed.alexandra.dk/uk/projects/scet.htm

## Going a step further: Research project SIMAP (Secure Information Managing and Processing)

Goal: develop domain specific programming language for applications of MPC:  SMCL

Easily describe computation you want, who participates, what should be known by whom and when, how many corruptions do we want to tolerate.

Compiler produces code to be run by participants, will execute the computation securely.

Language contains so far secret and public Integers and Booleans, simple arithmetic and control structures. Client/Server model as in SCET.

Partners: Crypto and Programming language groups from Aarhus University, economists from Copenhagen, IBM, Danisco, Lauritz.com.

So far: First version of compiler is ready, based on runtime system written in Java.

# Example Program in SMCL

For a variant of the millionaries problem by Yao: a number of billionaries. Each wants to know if he is the richest or not.
Clients: deliver personal net worth as a secret integer and recieve the result.
Servers: The servers must decide which millionary is the richest and send a boolean value of true to him and only him. The rest should get a false.

```
Declare Client Billionaire:
  Tunnel of int netWorth;          //tunnels provide secure transport to server side

  function void main(int[] args) { ask(); }

  function void ask() {
    netWorth.put(readInt());       //client starts by putting input in tunnel, then waits
  }

  function void tell(bool b) {     //called by server side when result is ready
    if (b)
      display("You are the richest!");
    else
      display("You are not rich enough!");
  }
```

# The server side

**Declare Server server:**

```
function void main(int[] args) {
  Group of Billionaire billionaires;

  sClient richest;  //secret pointer to richest client
  sint max = 0;     //eventually holds size of largest fortune

  foreach (Client c in billionaires) {
    sint netWorth = c.netWorth.get();  // get input from client
    sbool b = netWorth > max;          //richer then current record?
    max = b ? netWorth : max;          //set max according to b
    richest = b ? c : richest;         //set pointer to richest client
  }

  foreach (Client c in billionaires) {
    sbool b = (c == richest);
    c.tell(b);
  }
}
```

## In Conclusion..

Many applications of MPC can be realized with the efficiency and functionality that is required in real life.

Soon, you can write and maintain your own secure computation in a high-level language.

Will all this be used?

I think yes, but give it time!

Compare to Digital signatures: invented in 1977, first nation-wide system in Denmark started 2 years ago!

More info on research projects, see

http://www.sikkerhed.alexandra.dk/uk/projects/simap.htm

New collaborators welcome!