# Recent Results in Secure Computation

**Benny Pinkas**, Bar-Ilan University

# A canonical example: The millionaires' problem

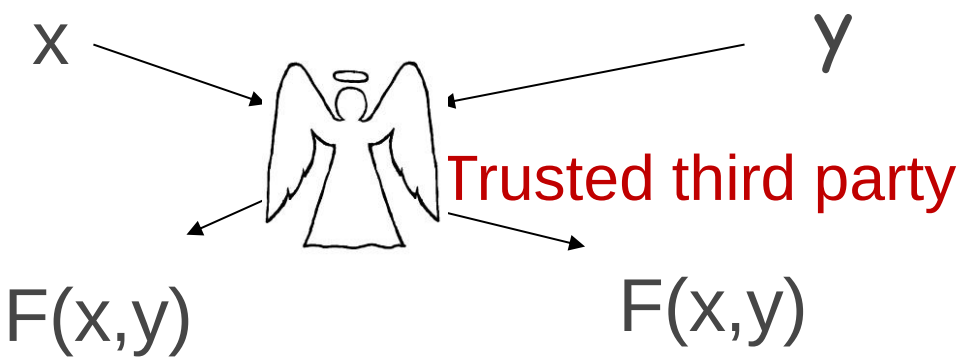

- Want to find out if X > Y
- But leak no other information! (even to each other)
- Standard crypto tools (encryption) do not help in this case!

# Secure two-party computation - definition

Input: x y

Output: F(x,y) *and nothing more*

As if…




Similar definitions exist for the multi-party case

# Example application: The millionaires' problem


Comparing numbers is useful for auctions, bidding, and negotiations.

# Example application: Auctions and bidding

- Run an auction while hiding the bids even from the auctioneer itself
  - The auctioneer learns who won the auction and how much the winner has to pay. Everything else is kept secret.
  - It is possible to support any auction rule (e.g., second-price auctions)
  - Efficient for even for thousands of bids

# Example application: Private Set Intersection (PSI)

Client

Server

| | | |
|---|---|---|
| Input: | $X = x_1 \ldots x_n$ | $Y = y_1 \ldots y_n$ |
| Output: | $X \cap Y$ only | nothing |

Other variants exist (e.g., both parties learn output; client learns size of intersection; compute some other function of the intersection, etc.)

# Example application: AES

| | | |
|---|---|---|
| Input: | X, K1 | secret key K2 |
| Output: | $E_{(K1 \text{ xor } K2)}(x)$ | nothing |

Instead of hiding the key using DRM, store it remotely.
Can encrypt without revealing the data.

# Example application: Cloud computing

user cloud

Input: X, Program →

Output: Program(X) nothing

The holy grail of secure computation.

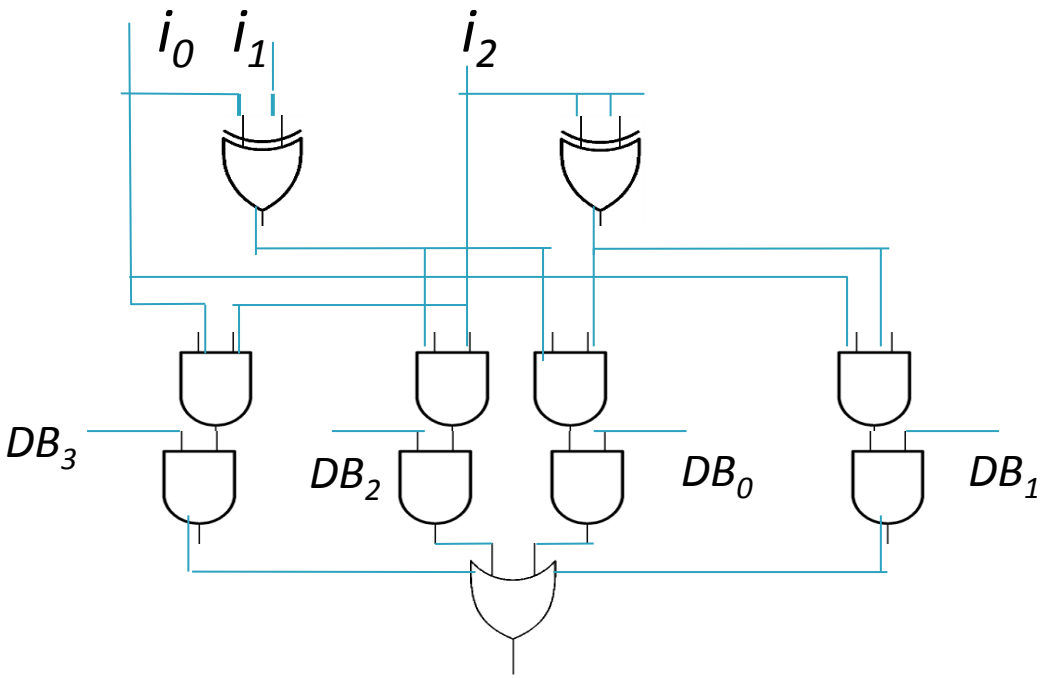
Possible using fully homomorphic encryption.

Far from being practical.

# Generic secure computation

- Can be used to securely compute any function

- Based on representing the function as a **Boolean circuit**



# Generic secure computation

- Can be used to securely compute any function

- Based on representing the function as a **Boolean circuit**
  - A Turing machine running in memory M and time T can be replaced by a circuit of size O(TM)
  - For many tasks, the circuit is linear in the input length
    - Adding or comparing two numbers
    - An AES circuit has about 30,000 gates
  - There exist compilers from programs to circuits
  - We can handle circuits with $10^6$ - $10^9$ gates.

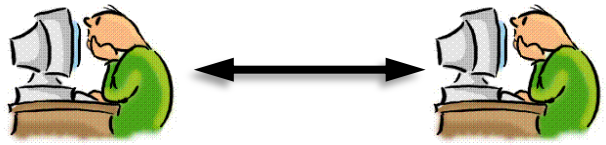
# Generic secure computation

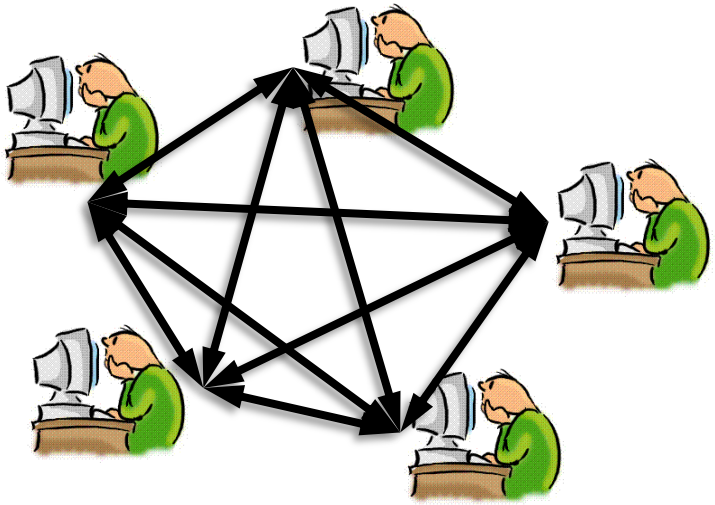
- Performance
  - Depends on security, preprocessing, and engineering
  - Secure computation of AES: from 3ms to 3sec per block

# Settings

# “Classical” MPC settings

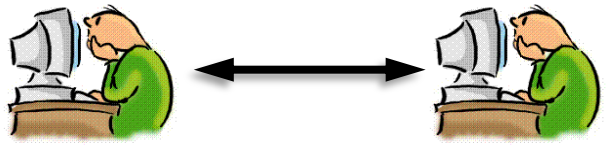
- Two or more parties with symmetric roles
- Each with its own input
- Exchanging messages with each other

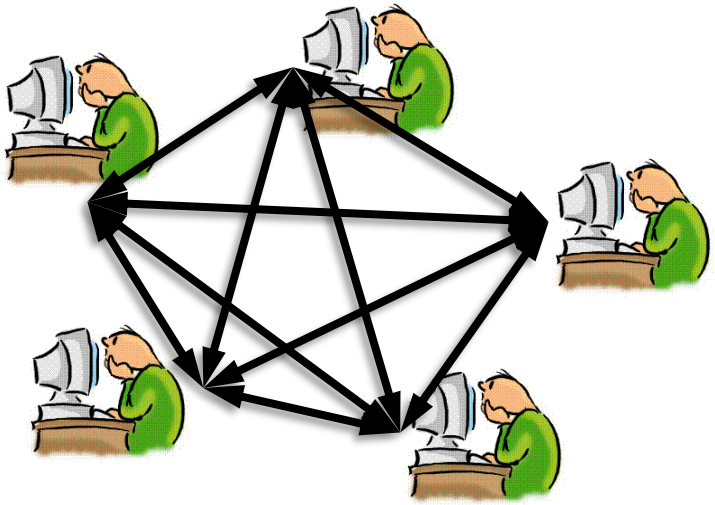




# “Classical” MPC settings

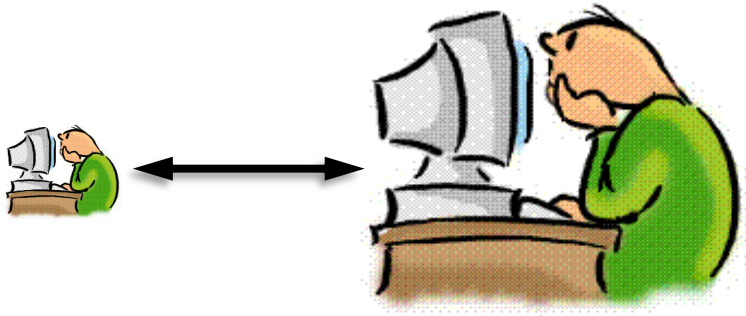
- This model might not be realistic
  - Asymmetric resources / tasks
  - Synchronization problems
- E.g.,
  - auctions, data sharing
  - outsourced computation





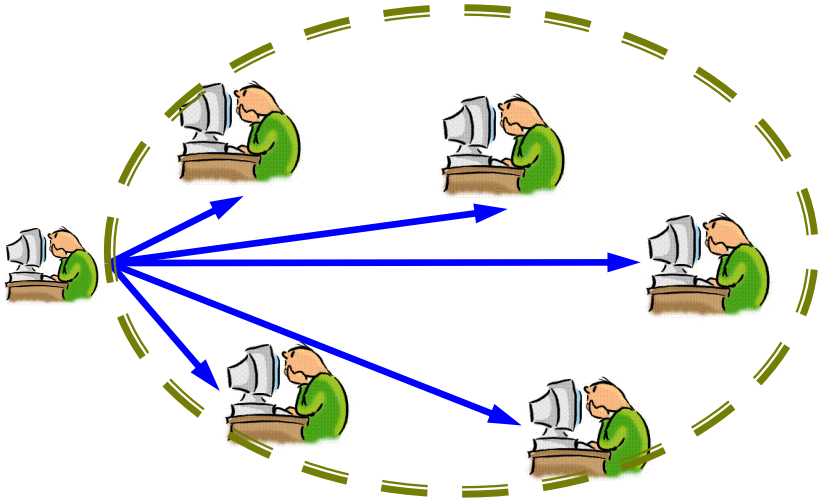
# Potential settings: “The Cloud”

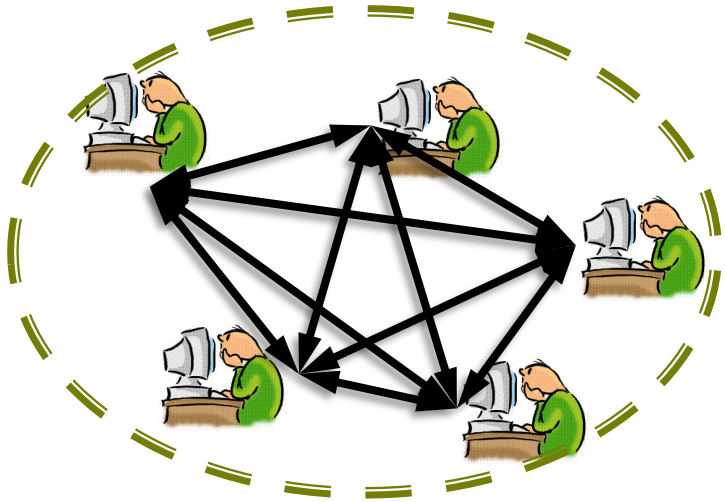
- “The Cloud”
- A weak client outsources its data and computation to the cloud
- Can be implemented at great costs using FHE



# Potential settings: different roles

- Many parties provide inputs, in a single interaction
- Later, computation servers run the computation (potentially in several rounds)
- Secure as not too many servers collude
- Relevant for auctions, etc.





# Potential “business models”

- Software as a service.
  - Software runs in the cloud. Input should be kept hidden from software provider.

- Outsourcing data
  - Data is stored in the cloud. Client wants to run analysis tools on its data.

# Potential “business models”

- Data sharing
  - Multiple parties with private inputs wish to run an algorithm over their combined data.

- Distributing trust
  - Storing shares of sensitive data/keys in multiple servers (and being able to use them), so that breaking into any one server does not leak any useful data.

# Potential “business models”

- Two-party transactions
  - A pair of parties with sensitive data and a specific algorithm (e.g., intersection and its variants)
  - Many pairs of parties run simple algs on their data (comparisons, trading,…)

# Recent research at BIU

- We have done a lot of work on improving the overhead of secure computation protocols

- In particular
  - Minimizing the interaction in secure computation protocols (namely, achieving non-interactive secure comptation)
  - Moving most work to a preprocessing stage
  - Security against malicious adversaries

# Recent research at BIU: SCAPI

- SCPAI: Secure Computation API
- An open-source Java library for secure computation

- Three layers:
  - Low-level cryptographic functions *(AES, hash, public key)*
  - Non-interactive mid-level cryptographic functions *(encryption, signature)*
  - Interactive cryptographic protocols *(secure computation, zero-knowledge proofs)*

# SCAPI: Flexibility

- Three layers:
  - Low-level cryptographic functions *(AES, hash, public key)*
  - Non-interactive mid-level cryptographic functions
  - Interactive cryptographic protocols
- Can easily use different libraries
  - Native Java vs. very efficient C libraries
- Can use different primitives
  - Public key operations modulo p vs. in an elliptic curve group

SCAPI is constantly updated to use the state of the art in secure computation

# Example application: Private Set Intersection (PSI)

Client

Server

| | | |
|---|---|---|
| Input: | $X = x_1 \ldots x_n$ | $Y = y_1 \ldots y_n$ |
| Output: | $X \cap Y$ only | nothing |

# PSI Applications

- **User-to-user Matching**
  - Two mobile users compute the intersection of their contact lists
  - Two mobile users check how good they match to each other (a dating app?)

# PSI Applications

- **User-to-service matching**
  - Mobile device has web history of user. A service wants to check if some item is in the history (content targeting).
  - Mobile device has list of ads shown to a user.
    The user now shops at a site.
    A service wants to check if the user was shown an ad for that specific site (checking ad conversion rate).

# A naïve PSI protocol

- A naïve solution:
  - A and B agree on a "cryptographic hash function" H()
  - B *(with input $y_1,\ldots,y_n$)* sends to A: $H(y_1),\ldots, H(y_n)$
  - A *(with input $x_1,\ldots,x_n$)* compares this to $H(x_1),\ldots, H(x_n)$ and finds the intersection

- Does not protect B's privacy if inputs do not have considerable entropy

# PSI protocols

- Several secure protocols for PSI exist

- A straightforward generic protocol based on a circuit is inefficient
  - For input sets of length n it requires $n^2$ comparisons
  - More efficient circuits exist
    - Of size O(nlogn)
    - E.g., sort the union of the two sets; compare adjacent items; shuffle the results.

# PSI protocols

- We recently compared the most promising PSI protocols, as well as
  - Optimized the protocols using new techniques (OT extension and advanced hashing)
  - Designed a new protocol tailored for the new techniques

# PSI based on Diffie-Hellman

- The protocol [M86, HFH99, AES03]:

$\alpha$

$x_1,\ldots,x_n$

$\beta$

$y_1,\ldots,y_n$

in parallel

$(H(x_1))^\alpha,\ldots, (H(x_n))^\alpha$ →

← $(H(y_1))^\beta,\ldots, (H(y_n))^\beta$

← $((H(x_1))^\alpha)^\beta,\ldots, ((H(x_n))^\alpha)^\beta$

$((H(y_1))^\beta)^\alpha,\ldots, ((H(y_n))^\beta)^\alpha$

in parallel

Compares the two lists

(H is modeled as a random oracle. Security based on DDH)

Implementation: very simple; can be based on elliptic-curve crypto; minimal communication.

What else could we want?

# Results (2014): run time (256K items)

| Protocol | 80-bit security (sec) | 128-bit security (sec) | Comm. Mbit |
|---|---|---|---|
| DH FFC | 99 | 1224 | 192 |
| DH ECC | 178 | 416 | 26 |
| Blind RSA | 125 | 1982 | 72 |
| Circuit + GMW | 807 | 1304 | 23400 |
| Optimized circuit | 462 | 762 | 14040 |
| Garbled Bloom | 72 | 154 | 1393 |
| Optimized G. Bloom | 34 | 68 | 740 |
| OT + hashing | 13 | 14 | 78 |

(Single core desktop)

New ideas can probably improve by a factor of 5-10.

# Conclusions so far

- Set intersection can be efficiently applied to very large input sets

- Different settings require different protocols
  - Communication
  - Generality

# Conclusions

- Many tasks have efficient secure computation solutions

- If you wish that you had a trusted party for computing a task
  - And you're OK with disclosing the final output of the computation
  - Then it might be possible to implement the computation without any trusted party