Secure Computation on the Web: Computing without Simultaneous Interaction

Shai Halevi, IBM T.J. Watson
Yehuda Lindell, Bar-Ilan University
Benny Pinkas, Bar-Ilan University

My Standard First Slide

Secure Computation

- A set of parties with private inputs
- Parties wish to jointly compute a function of their inputs so that certain security properties (like privacy, correctness and independence of inputs) are preserved
- Properties must be ensured even if some of the parties attack the protocol

- Models any problem:
  - Elections, auctions, private statistical analysis,…

A Question

- **Can elections, auctions, statistical analysis of distributed parties' data really be carried out using secure computation?**
- **Does our model of secure computation really model the needs of these applications?**
  - And I'm **not** talking about efficiency concerns…

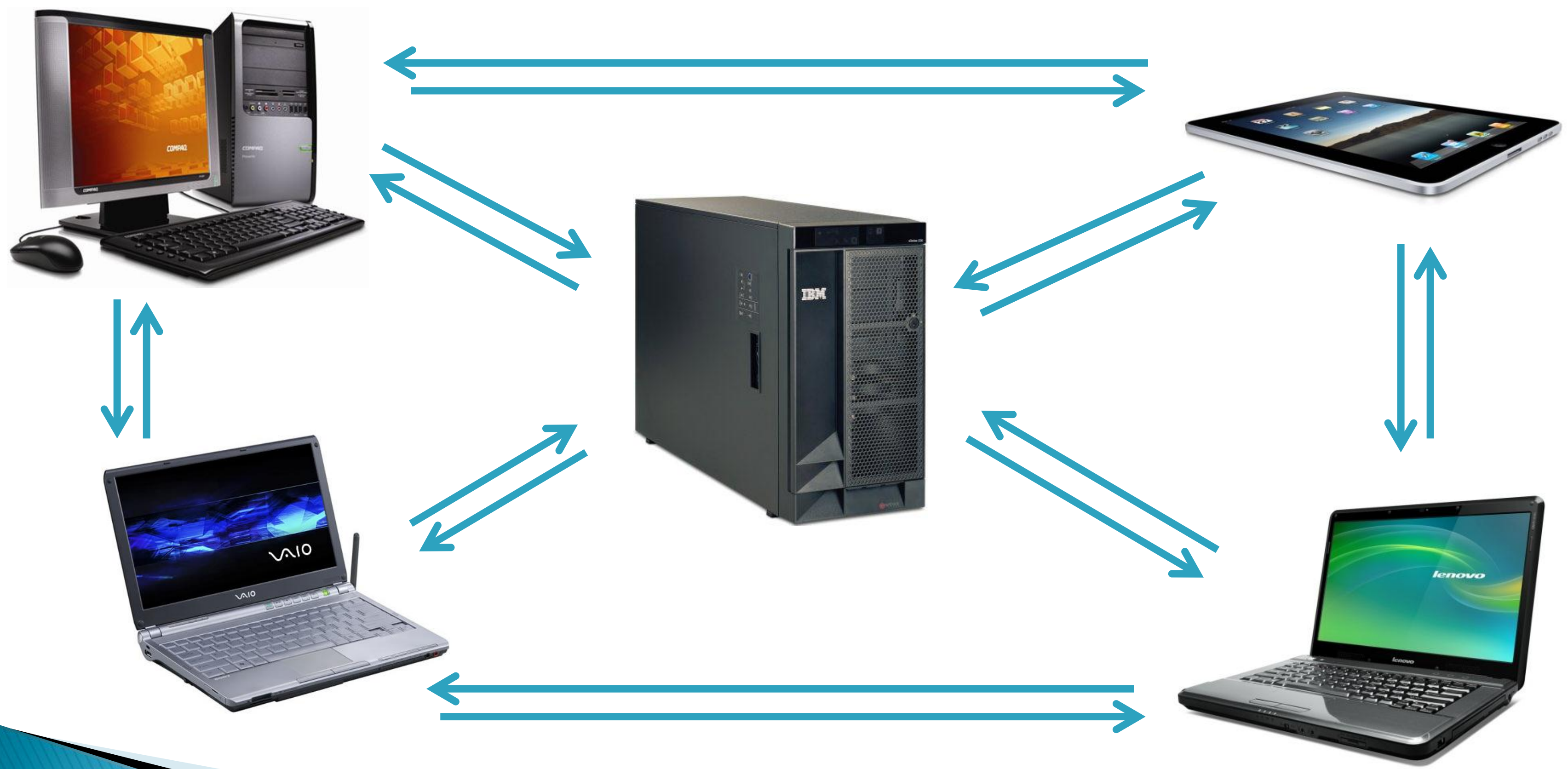
A Big Problem

- **In all known protocols, all parties must interact <u>simultaneously</u>**

- **Arguably, this is a huge obstacle to adoption**
  - A department wants to carry out a faculty tenure vote using a secure protocol
    - When do they run the protocol?
  - A website wishes to securely aggregate statistics about users
    - Each user gives her information only when connected

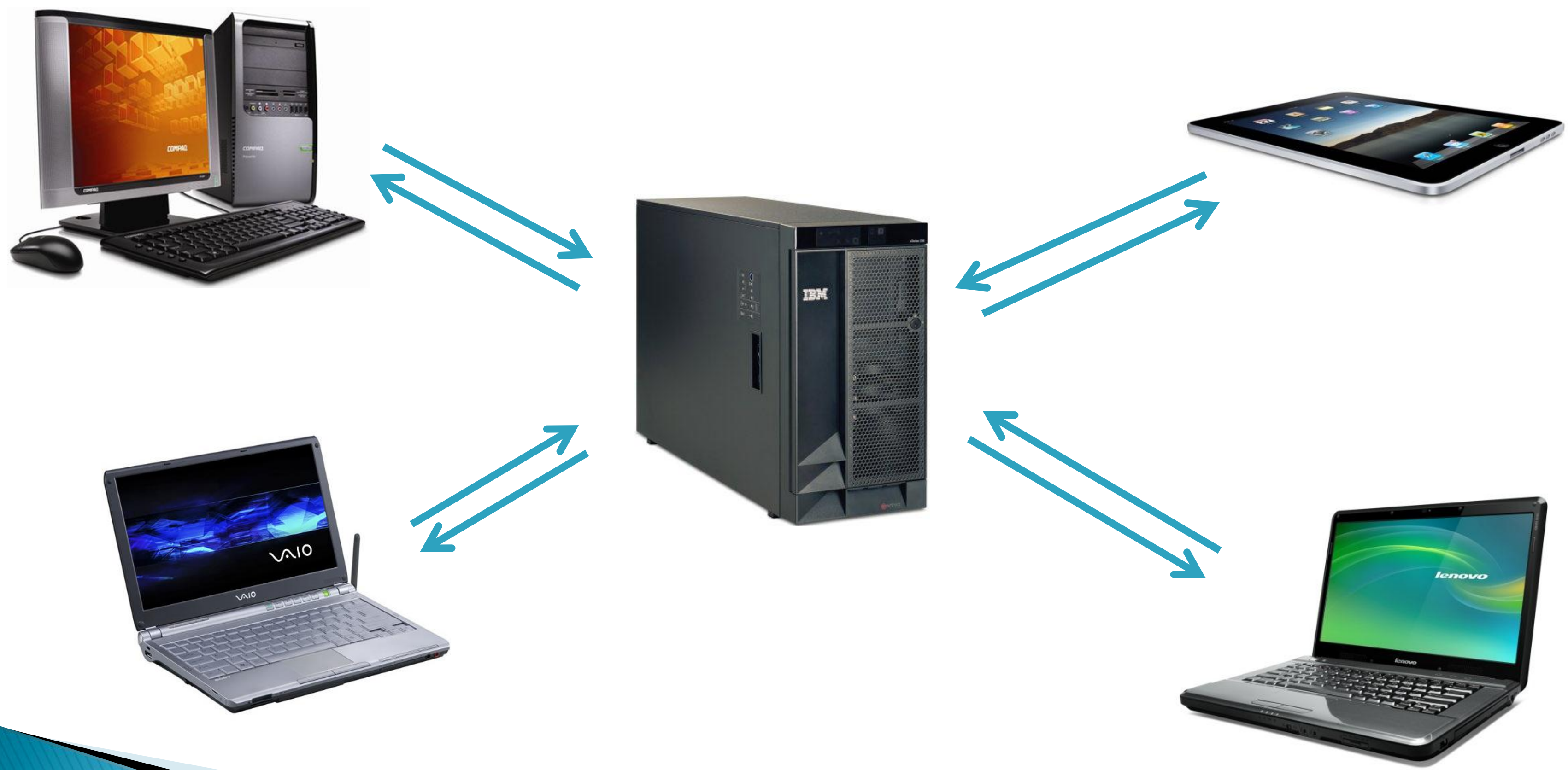
Stated Differently

- The secure computation model:



Stated Differently

- The real-world web model:



An Important Question

- **Can secure computation be made non-simultaneous?**
  - A natural theoretical question
    - Deepens our understanding of the required communication model for secure computation
  - Important ramifications to practice
    - Especially if this can be done efficiently

- Note: fully homomorphic encryption does not solve the problem

Our Model

- **Parties**
  - One server $\boldsymbol{S}$
  - $\boldsymbol{n}$ parties $\boldsymbol{P_1, \ldots, P_n}$
- **Communication model**
  - Each party interacts with the server **exactly once**
    - In all of our protocols, this interaction is a single message from the server to the party and back, but this is not essential to the model
  - At the end, the server obtains the output
- **A protocol for this setting is called one pass**

Residual Function Computation

- Since the protocol is one-pass, the computation carried out by $P_{i+1}, \dots, P_n$ and S is of the residual function

$$g_i(\boldsymbol{x_{i+1}}, \dots, \boldsymbol{x_n}) = f(x_1, \dots, x_i, \boldsymbol{x_{i+1}}, \dots, \boldsymbol{x_n})$$

- If $P_{i+1}, \dots, P_n$ and S are all corrupted and colluding, they can compute $g_i(\boldsymbol{x_{i+1}}, \dots, \boldsymbol{x_n})$ and $g_i(\boldsymbol{x'_{i+1}}, \dots, \boldsymbol{x'_n})$ and so on, on many inputs
    - This is not allowed in classic secure computation but is <u>inherent</u> to the one-pass model

Function Decomposition

- **A decomposition of a function $f(x_1, \ldots, x_n)$ is a series of n two-input functions $f_1, \ldots f_n$ such that $f_n(\cdots f_2(f_1(x_1), x_2) \cdots x_n) = f(x_1, \ldots, x_n)$**
  - In the one-pass setting P_i (and S) compute f_i and pass on the result
  - If $P_{i+1}, \ldots, P_n$ and S are all corrupted and colluding, then they learn the value $f_i(\cdots f_2(f_1(x_1), x_2) \cdots x_i)$

Minimal Disclosure Decomposition

- How much does $f_i(\cdots f_2(f_1(x_1), x_2) \cdots x_i)$ reveal?
- If it reveals nothing more than what can be computed by the residual function
$$g_i(x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_i, x_{i+1}, \ldots, x_n)$$
then it is <u>minimal disclosure</u>

Examples

- **Define $f_1(x_1) = x_1$, $f_2(y_1, x_2) = (y_1, x_2) = (x_1, x_2)$, and so on (all are identity functions), and $f_n = f$**
  - If P_n and S are corrupted, all is revealed

- **Consider the SUM function and define**

$$f_i(y_{i-1}, x_i) = y_{i-1} + x_i$$

  - Given y_i can learn nothing more than sum of first i
  - But this is computable from the residual function
  - This is minimal disclosure

Definition of Security

- **We follow the real/ideal simulation paradigm**
- **Security is formalized as in the standard setting with one exception**
  - If the server is corrupted, then the adversary is given $\boldsymbol{f_i(x_1, \ldots, x_i)}$ where $\boldsymbol{P_i}$ is the last honest party

- **A protocol one-pass securely computes a decomposition if there exists an ideal simulator such that <u>real</u> and <u>ideal</u> are indistinguishable**
  - The protocol is **optimally private** if the decomposition is minimum disclosure

Questions

- **Can this notion be achieved?**
- **If yes,**
  - Under what assumptions?
  - At what cost?

Practical Optimal Protocols

- **Binary symmetric functions**
  - Depend only on Hamming weight of input
  - E.g., AND, OR, PARITY, MAJORITY
- **Concise truth table representation**
  - Example: the MAJORITY function over 5 bits

Hamming Weight	Output
0	0
1	0
2	0
3	1
4	1
5	1

In general, this contains the function output on the relevant weight

Minimum Disclosure Decomposition for Binary Symmetric

- Define $y_1 = f_1(x_1)$ to be the truth table, with the 1st row erased if $x_1 = 1$ and the last row erased if $x_1 = 0$

Hamming Weight	Output	
~~0~~	~~0~~	$x_1 = 1$
1	0	
2	0	
3	1	
4	1	
~~5~~	~~1~~	$x_1 = 0$

Minimum Disclosure Decomposition for Binary Symmetric

- Define $f_2(y_1, x_2)$ to be the truncated truth table, with the last remaining row erased if $x_2 = 0$ and the first row erased if $x_2 = 1$

Hamming Weight	Output	
~~0~~	~~0~~	$x_2 = 1$
1	0	
2	0	
3	1	
4	1	
~~5~~	~~1~~	$x_1 = 0$

Minimum Disclosure Decomposition for Binary Symmetric

- **And so on…**
  - Note, each truth table can be efficiently computed from the previous one

Hamming Weight	Output	
~~0~~	~~0~~	$x_2 = 1$
~~1~~	~~0~~	$x_3 = 1$
2	0	
~~3~~	~~1~~	$x_5 = 0$
~~4~~	~~1~~	$x_4 = 0$
~~5~~	~~1~~	$x_1 = 0$

  - Indeed, the output of $\boldsymbol{MAJ(01100) = 0}$

Minimum Disclosure Decomposition for Binary Symmetric

- **Why is this minimum disclosure?**
  - The truth table reveals nothing more than the output of the function on the remaining inputs

Practical Optimal Protocol for Binary Symmetric Functions

- **Main tool – layer rerandomizable encryption**
  - Denote $E_{pk}(x; r)$ and
    $E_{pk_{1,\dots,pk_{n+1}}}(x; r_1, \dots, r_{n+1}) = E_{pk_1}\left(\cdots E_{pk_{n+1}}(x; r_{n+1}) \cdots ; r_1\right)$
  - This is layer rerandomizable if there exists an efficient procedure that rerandomizes all layers (given public keys)
  - This can be constructed from any rerandomizable encryption, and **highly efficiently** from ElGamal

- **Note: all protocols assume PKI (essential here)**

The Protocol (Semi-Honest)

- **Server S encrypts the truth table under all parties' keys**
  - Using rerandomizable layer encryption
- **For $i = 1, \ldots, n$ (but in any order)**
  - Party P_i retrieves current truth table from the server
  - P_i removes the first or last remaining row, decrypts under its key, rerandomizes every entry of the truth table, and sends to S
- **After all parties conclude, all that remains is a single row, which is the output**

Example

- Majority function with 5 parties

Hamming Weight	Output
0	0
1	0
2	0
3	1
4	1
5	1

Example – MAJORITY

- The server S computes the encrypted concise truth table (pk_6 is the server's public-key)

$E_{pk_1,\ldots,pk_6}(0; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(0; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(0; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(1; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(1; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(1; r_1, \ldots, r_6)$

Example – MAJORITY

- P_1 with input $x_1 = 0$ erases

$E_{pk_1,\ldots,pk_6}(0; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(0; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(0; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(1; r_1, \ldots, r_6)$
$E_{pk_1,\ldots,pk_6}(1; r_1, \ldots, r_6)$

Example – MAJORITY

- P_1 with input $x_1 = 0$ erases, removes its key and rerandomizes

$E_{pk_2,\ldots,pk_6}(0; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(0; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(0; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(1; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(1; r_2, \ldots, r_6)$

Example – MAJORITY

- P_2 with input $x_2 = 1$ erases

$E_{pk_2,\ldots,pk_6}(0; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(0; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(1; r_2, \ldots, r_6)$
$E_{pk_2,\ldots,pk_6}(1; r_2, \ldots, r_6)$

Example – MAJORITY

- P_2 with input $x_2 = 1$ erases, removes its key and rerandomizes

$E_{pk_3,\ldots,pk_6}(0; r_3, \ldots, r_6)$
$E_{pk_3,\ldots,pk_6}(0; r_3, \ldots, r_6)$
$E_{pk_3,\ldots,pk_6}(1; r_3, \ldots, r_6)$
$E_{pk_3,\ldots,pk_6}(1; r_3, \ldots, r_6)$

Example – MAJORITY

- P_3 with input $x_3 = 1$ erases

$E_{pk_3,\dots,pk_6}(0; r_3, \dots, r_6)$
$E_{pk_3,\dots,pk_6}(1; r_3, \dots, r_6)$
$E_{pk_3,\dots,pk_6}(1; r_3, \dots, r_6)$

Example – MAJORITY

- P_3 with input $x_3 = 1$ erases, removes its key and rerandomizes

$E_{pk_4,\dots,pk_6}(0; r_4, \dots, r_6)$
$E_{pk_4,\dots,pk_6}(1; r_4, \dots, r_6)$
$E_{pk_4,\dots,pk_6}(1; r_4, \dots, r_6)$

Example – MAJORITY

- P_4 with input $x_4 = 0$ erases

$E_{pk_4,\ldots,pk_6}(0; r_4, \ldots, r_6)$
$E_{pk_4,\ldots,pk_6}(1; r_4, \ldots, r_6)$

Example – MAJORITY

- P_4 with input $x_4 = 0$ erases, removes its key and rerandomizes

$E_{pk_5,pk_6}(0; r_5, r_6)$
$E_{pk_5,pk_6}(1; r_5, r_6)$

Example

- A corrupted P_5 colluding with a corrupted server know that the first 4 parties were divided evenly, but nothing else

$\boldsymbol{E_{pk_5,pk_6}(0;r_5,r_6)}$
$\boldsymbol{E_{pk_5,pk_6}(1;r_5,r_6)}$

Security

- **If server is honest, no one learns anything**
- **If server is corrupt, it cannot decrypt anything which is still encrypted under an honest party's public-key**
  - Security level achieved when last few parties are corrupted is the same as if they just didn't participate to start with
- **Rerandomization ensures that the row removed is not learned**

Concrete Cost

- **Each party computes on average about $^{3n}/_{2}$ exponentiations**
  - We can do $\mathbf{1000-2000}$ exponentiations per second, making this protocol practical even for thousands of users (unless many come at the same time)
- **For malicious adversaries**
  - Need to add digital signatures and ZK proofs (these are just Diffie-Hellman tuple proofs)
  - The concrete cost is less than $\boldsymbol{8n^2}$ (with Fiat-Shamir)
  - This is still practical for not too many parties
    - About 10 seconds for 40 parties (tenure example)

More Results

- **Highly efficient optimally private protocols for:**
  - Symmetric functions over $\mathbb{Z}_c$
  - Sum function over large domain
  - Selection functions
- **A general feasibility result:**
  - Any decomposition $f_1, \dots, f_n$ can be securely computed, under the DDH assumption (and NIZK for malicious)
  - This can be used **for any decomposition** (minimal or not)
    - The actual security derived depends on the decomposition
    - Minimal is best; if not, then it depends on the application

Summary

- **Fully interactive secure computation is a problem in practice**
  - A **one-pass client/server protocol** is essential for many applications, and is also interesting from a theoretical point of view
- **Our results**
  - Introduced the model and definitions
  - Studied inherent limitations and use function decomposition to model this
  - Constructed highly efficient and practical protocols exist for many natural problems in this setting
  - Proved general feasibility for any decomposition