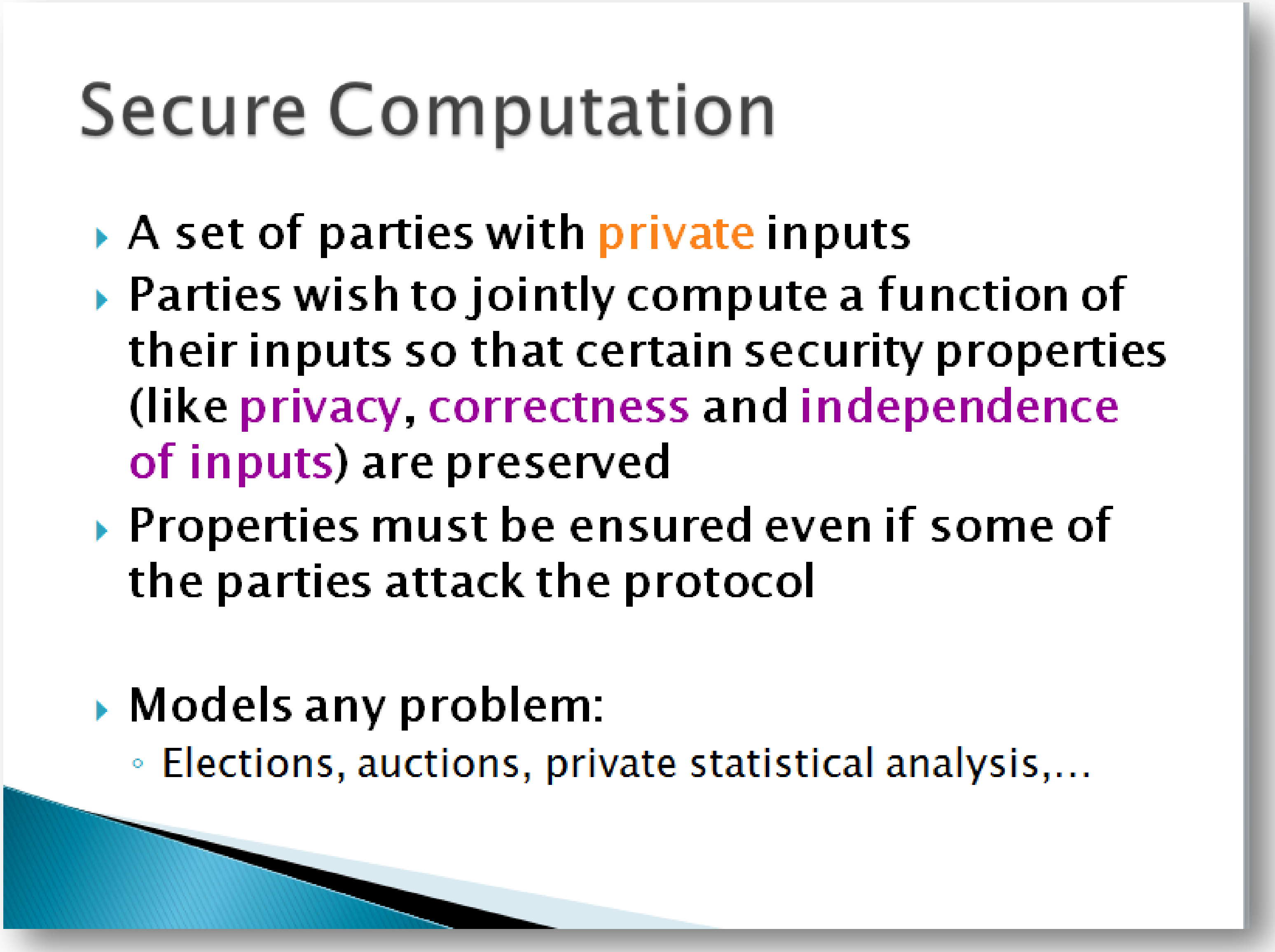
Secure Computation on the Web: Computing without Simultaneous Interaction

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

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 we’re **not** talking about efficiency concerns…

A Big Problem

- In all known protocols, all parties must interact simultaneously

- Arguably, this is a huge obstacle to adoption
  - A program committee wants to vote on the best paper using a secure protocol
    - When do they run the protocol?
  - A website wishes to securely aggregate statistics about users
    - Each user provides 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

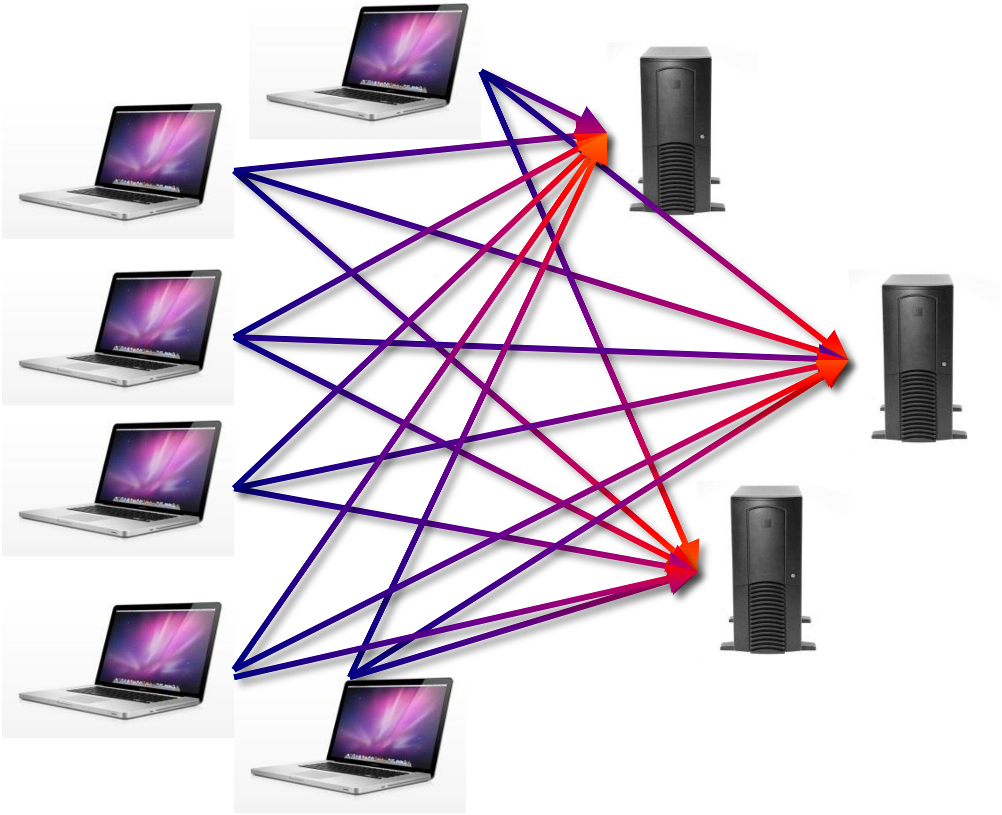
- Note: fully homomorphic encryption does not solve the problem

A Hack around this Issue

- Most of the parties are “input providers”
- Some parties/servers do the actual computation

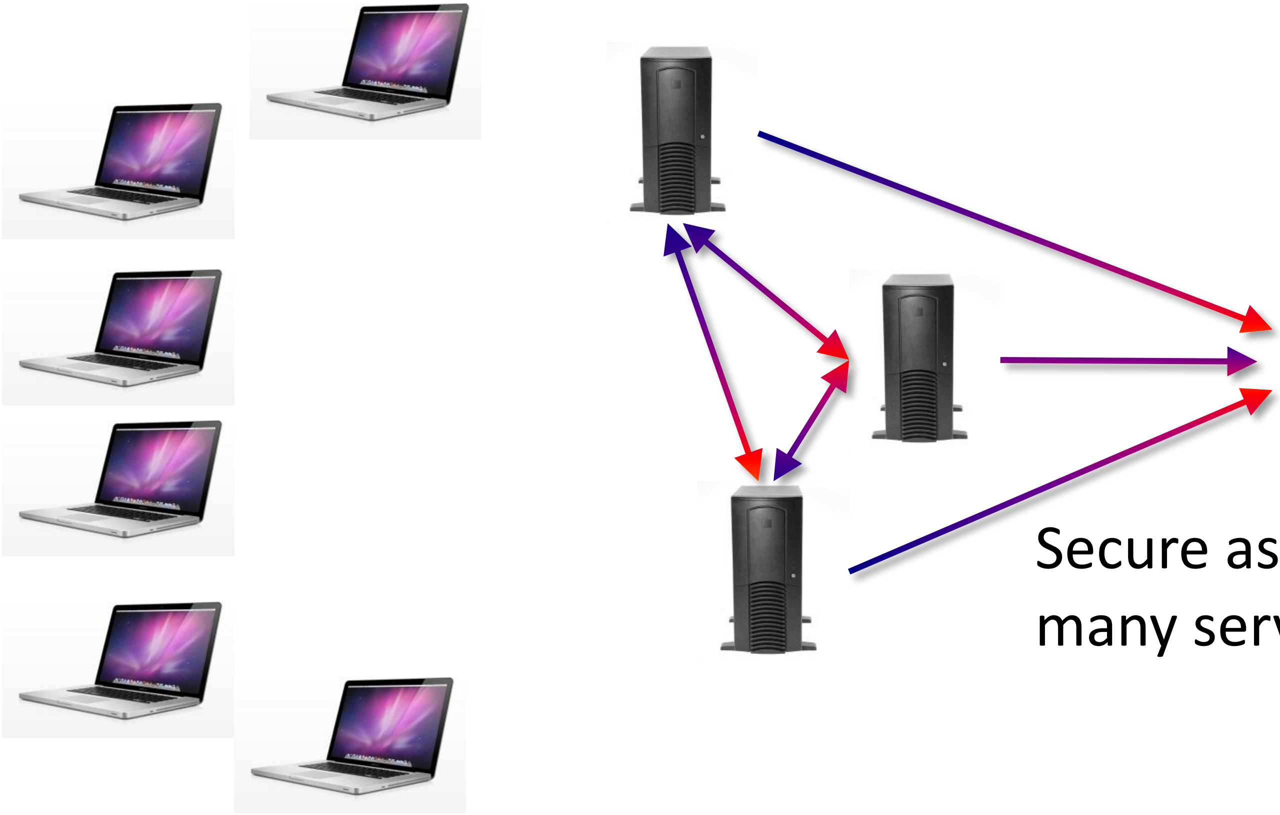
A Hack around this Issue

- Hack: Input providers send shares of the their inputs to the servers (non-interactively, whenever they want)



A Hack around this Issue

- Servers then do the actual computation (consisting of multiple interactive rounds)

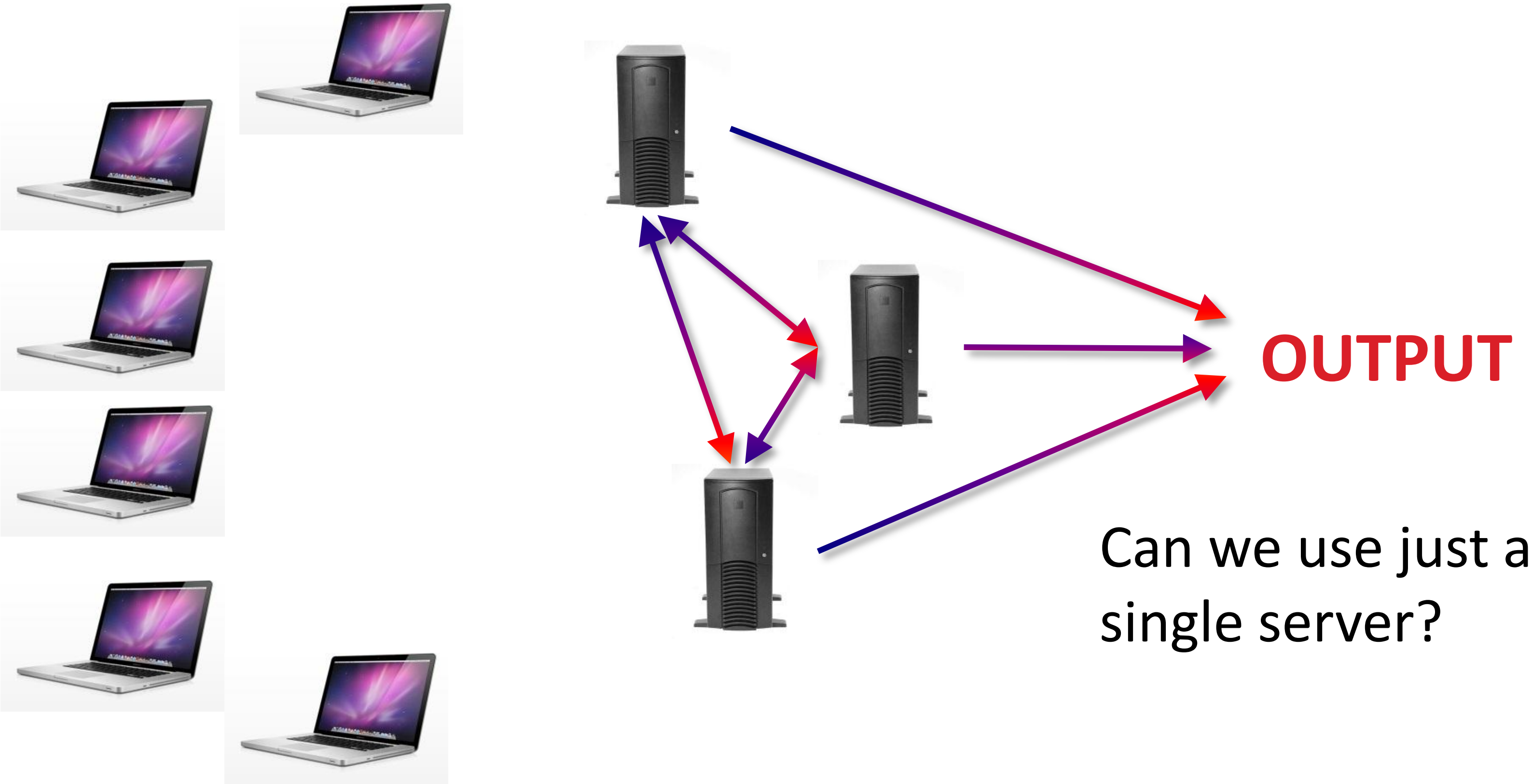


OUTPUT

Secure as long as not too many servers collude

A Hack around this Issue

- A client-server model
- Auctions [NPS99,BCD+09], surveys [FPRS04], FairplayMP [BNP09]



Can we use just a single server?

Related Work

- Non-interactive cryptocomputing [SYY99]
- Secure function collection [IKYZ09]

Our Model

- Parties
  - One server $\boldsymbol{S}$
  - $\boldsymbol{n}$ parties $\boldsymbol{P_1}, \dots, \boldsymbol{P_n}$
- Communication model
  - Each party interacts with the server **exactly once**
    - In all protocols, interaction is a single round between server and party, but this is not essential
    - **Order** may be important… (in some protocols)
  - At the end, the server obtains the output

Limitations

- A problem
  - A corrupted server can take the message computed by $\boldsymbol{P_1}$ and play all the roles of $\boldsymbol{P_2, \ldots, P_n}$ itself, with any set of inputs
    - When computing AVG this would reveal $\boldsymbol{P_1}$'s input
  - Conclusion: It is **not possible** to solve this problem in the plain model
- Solution
  - We solve this problem by assuming a known public-key infrastructure
  - The secret key of $\boldsymbol{P_i}$ is needed to run $\boldsymbol{P_i}$'s instructions, preventing $\boldsymbol{S}$ from doing the above

Limitations

- Consider $\boldsymbol{n}$ parties who wish to compute $\boldsymbol{y = f(x_1, \dots, x_n)}$ where the $\boldsymbol{j}$th party has input $\boldsymbol{x_j}$
  - Consider the residual function
    $$\boldsymbol{g_i(X_{i+1}, \dots, X_n) = f(x_1, \dots, x_i, X_{i+1}, \dots, X_n)}$$
  - Clearly, the server and the last $\boldsymbol{n - i}$ parties must be able to compute $\boldsymbol{g_i(x_{i+1}, \dots, x_n)}$ for any set of legit inputs $\boldsymbol{x_{i+1}, \dots, x_n}$

- Consider a semi-honest adversary running $\boldsymbol{P_{i+1}, \dots, P_n, S}$
  - It must be able to compute $\boldsymbol{g_i(X_{i+1}, \dots, X_n)}$ for **any** set of inputs $\boldsymbol{x_{i+1}, \dots, x_n}$
  - This is more than allowed in “classic” secure comp.

Limitations

- This is an inherent limitation of the model
  - Honest parties $\boldsymbol{P_{i+1}, \dots, P_n}$ and $\boldsymbol{S}$ must be able to compute $\boldsymbol{g_i(X_{i+1}, \dots, X_n)}$ for **any** set of inputs $\boldsymbol{x_{i+1}, \dots, x_n}$
  - Thus, a semi-honest adversary controlling these parties can also do that

- We must therefore allow this in the security definition

Function Decomposition

- A decomposition of a function $f(x_1, \dots, x_n)$ is a vector of functions

$$f_1(\lambda, x_1), f_2(y_1, x_2), \dots, f_n(y_{n-1}, x_n)$$

such that

$$f_n(\cdots f_2(f_1(x_1), x_2) \cdots x_n) = f(x_1, \dots, x_n)$$

y_i is the state of the server after interaction with P_i

- Intuition:
  - At each stage, P_i computes f_i
  - If S and $P_{i+1}, \dots, P_n$ are corrupted in our setting, then they can compute $f_i(x_{i+1}, \dots, x_n)$ on any inputs they like, but nothing else

Decompositions

- Are all decompositions equally good?
- Consider $f_1(x_1) = x_1, \ldots, f_i(y_{i-1}, x_i) = (y_{i-1}, x_i), \ldots$ (the **identity** function)
  - If P_n and S are corrupted, then all inputs are revealed

- Consider the AVG function, and $f_i(y_{i-1}, x_i) = y_{i-1} + x_i$
  - Given y_i we can learn nothing more than sum of first i
  - But this can be also learned from setting $x_{i+1} = \cdots = x_n = 0$

- This latter decomposition seems better

Minimum Disclosure

- A decomposition $\boldsymbol{f_1, \dots, f_n}$ of $\boldsymbol{f}$ is minimum disclosure if there exists a simulator $\boldsymbol{S}$, s.t. for every vector $\overline{\boldsymbol{x}} = (\boldsymbol{x_1, \dots, x_n})$ and every $\boldsymbol{i}$, $\boldsymbol{S}$ with oracle access to $\boldsymbol{g_i}(\overline{\boldsymbol{x}}) = \boldsymbol{g}(\boldsymbol{x_1, \dots, x_i}, \cdot, \dots, \cdot)$ outputs $\boldsymbol{f_i}(\boldsymbol{x_1, \dots, x_i})$
- Note that if $\boldsymbol{P_{i+1}, \dots, P_n}$ and $\boldsymbol{S}$ are corrupted then they inherently have access to $\boldsymbol{g_i}(\overline{\boldsymbol{x}}) = \boldsymbol{g}(\boldsymbol{x_1, \dots, x_i}, \cdot, \dots, \cdot)$
- If the decomposition can be computed from access to $\boldsymbol{g}(\boldsymbol{x_1, \dots, x_i}, \cdot, \dots, \cdot)$ then learning it does not disclose new information. Therefore it is minimum disclosure.

- Not all functions have minimum disclosure decompositions

Not all Functions have a Minimum Disclosure Decomposition

- Assuming that one-way funcs exist, there is a function for which no minimum disclosure decomposition exists
- $F(k, x) = ENC_k(x) = f_2(f_1(k), x)$

- A corrupt server and P_2 have access to an oracle $f_2(X) = ENC_k(X)$

- Intuitively, if they can compute $f_1(k)$ given access to encryption oracle, then the encryption is insecure.
  - This can be shown by a formal reduction

Minimum Disclosure Examples

- The sum function $f(x_1, \ldots, x_n) = x_1 + \cdots + x_n$
  - The decomposition $f_i(x_1, \ldots, x_n) = x_1 + \cdots + x_i$ is minimal disclosure

- Proof of minimal disclosure
  - Given access to the oracle $g_i(x_{i+1}, \ldots, x_n) = f(x_1, \ldots, x_i, x_{i+1}, \ldots ., x_n)$
  - Compute $g_i(0, \ldots, 0) = x_1 + \cdots + x_i = f_i(x_1, \ldots, x_i)$, which is the decomposition.

- We couldn't have proved a similar reduction for the id decomposition

Minimum Disclosure Examples

- Binary symmetric functions
  - Depend only on Hamming weight of input
  - E.g., AND, OR, PARITY, MAJORITY, THRESHOLD
- 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, the output is a function of the weight

Minimum Disclosure Binary Symmetric

- Define $y_1 = f_1(x_1)$ to be the truth table, with the first 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 Binary Symmetric

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

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

- A similar condensed truth table exists for symmetric functions over (non-binary) constant-size domains
  - The table size is $\binom{n+c-1}{n} = O(n^{c-1})$ for an input domain of size c

Definition of Security

- We follow the real/ideal paradigm

- Security is defined by comparing a real execution to an ideal execution with a trusted party
    - A protocol is secure if no real adversary can do more than an ideal adversary

Definition of Security

- Real execution – as described
- Ideal execution (for a given decomposition)
  - **Honest server:**
    - All parties give inputs; server gets output
  - **Corrupt server + arbitrary number of corrupt parties:**
    - As above, except that adversary is also given $\boldsymbol{f_i(x_1, \dots, x_i)}$ where $\boldsymbol{P_i}$ is the last honest party
- A protocol securely computes a decomposition if there exists an ideal simulator such that real and ideal executions are indistinguishable
  - The protocol is **optimally private** if the decomposition is minimum disclosure

Questions

- Can the notion of optimally private protocols be achieved?

- If yes,
  - Under what assumptions?
  - At what cost?

Practical Optimal Protocols

- Binary symmetric functions
- Main tool – layer rerandomizable encryption
  - Denote $\boldsymbol{E_{pk}(x; r)}$ and $\boldsymbol{E_{pk_{1,\dots,}pk_{n+1}}(x; r_1, \dots, r_{n+1}) = E_{pk_1}\left(\cdots E_{pk_{n+1}}(x; r_{n+1}) \cdots ; r_1\right)}$
  - This is **layer rerandomizable** if there exists an efficient procedure that rerandomizes **all** layers (given public keys)
  - Must be able to remove encryptions one by one.
  - Useful to hide which table entry was removed
- From any randomizable, (e.g. additively homomorphic) encryption
  - $\boldsymbol{E_{pk_{1,\dots,}pk_{n+1}}(x; r_1, \dots, r_{n+1}) = E_{pk_1}(x_1; r_1), \dots, E_{pk_{n+1}}(x_{n+1}; r_{n+1})}$ s.t. $x = x_1 + \cdots + x_{n+1}$
  - Rerandomize each separately, decrypt one at a time

More Efficient Layer Rerand. Encryption

- Using El Gamal (DDH assumption)
  - Generator $\boldsymbol{g}$, prime-order group of order $\boldsymbol{q}$
  - Public keys $\boldsymbol{h_1 = g^{\alpha_1}, \dots, h_n = g^{\alpha_n}, h_{n+1} = g^{\alpha_{n+1}}}$
  - Define $\boldsymbol{H_{i,n+1} = \prod_{j=i}^{n+1} h_j = g^{\sum_{j=i}^{n+1} \alpha_j}}$

  - To encrypt: $\boldsymbol{E_{H_{1,n+1}}(m) = (u, v) = \left(g^r, \left(H_{1,n+1}\right)^r \cdot m\right)}$
  - To decrypt the first key and rerandomize
    - Decrypt: compute $\boldsymbol{u' = u}$ and $\boldsymbol{v' = v \cdot u^{-\alpha_1}}$
    - Rerandomize: compute $\boldsymbol{u'' = u' \cdot g^s}$ and $\boldsymbol{v'' = v' \cdot \left(H_{2,n+1}\right)^s}$
    - And so on, for each party

The Protocol (Semi-Honest)

- Server $\boldsymbol{S}$ encrypts the truth table under all parties’ keys
  - Using rerandomizable layer encryption
- Server $\boldsymbol{S}$ sends table to first connecting party $\boldsymbol{P_1}$
- $\boldsymbol{P_1}$ removes first or last row, decrypts every entry of the truth table using its key and rerandomizes it, and sends to $\boldsymbol{S}$
- Server $\boldsymbol{S}$ receives and sends to $\boldsymbol{P_2}$
- $\boldsymbol{P_2}$ removes first or last row, rerandomizes every entry of the truth table, and sends to $\boldsymbol{S}$
- …
- After $\boldsymbol{P_n}$ concludes, it sends $\boldsymbol{S}$ an encryption (under $\boldsymbol{S}$’s key) of the remaining row = output
- Rerandomization is important…

Example

- Majority function with 5 parties

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

Example

- **The server S** computes the encrypted concise truth table

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

Example

- 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

- $\boldsymbol{P_1}$ with input $\boldsymbol{x_1} = \mathbf{0}$ erases, removes its key and rerandomizes

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

Example

- $\boldsymbol{P_2}$ with input $\boldsymbol{x_2 = 1}$ erases

$$\boldsymbol{E_{pk_2,\dots,pk_6}(0; r_2, \dots, r_6)}$$

$$\boldsymbol{E_{pk_2,\dots,pk_6}(0; r_2, \dots, r_6)}$$

$$\boldsymbol{E_{pk_2,\dots,pk_6}(1; r_2, \dots, r_6)}$$

$$\boldsymbol{E_{pk_2,\dots,pk_6}(1; r_2, \dots, r_6)}$$

Example

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

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

Example

- $\boldsymbol{P_3}$ with input $\boldsymbol{x_3 = 1}$ 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

- $\boldsymbol{P_3}$ with input $\boldsymbol{x_3 = 1}$ erases, removes its key and rerandomizes

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

Example

- $\boldsymbol{P_4}$ with input $\boldsymbol{x_4} = \mathbf{0}$ erases

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

Example

- $\boldsymbol{P_4}$ with input $\boldsymbol{x_4} = \mathbf{0}$ erases, removes its key and rerandomizes

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

Example

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

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

- Note that parties could connect to server in an arbitrary order, since decryption and rerandomization can be done in any order.

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

- The server knows the initial encryptions of the table entries. Even if it colludes with $\boldsymbol{P_2}$ it does not which entry was removed by $\boldsymbol{P_1}$, because of rerandomization

Concrete Cost (Semi-Honest)

- S encrypts table at a cost of $\mathbf{2}$ exponentiations per truth table entry (equals $\boldsymbol{2n}$ exponentiations)
- Each $\boldsymbol{P_i}$ computes $\boldsymbol{n-i}$ decryptions and rerandomizations; each costs $\mathbf{3}$ exponentiations
- We can do $\mathbf{1000-2000}$ exponentiations per second, making this protocol practical even for thousands of users
  - Main problem could be sequentiality if many parties come at the same time

Security for Malicious

- Possible attacks
  - Server can send an incorrect truth table or a truth table with unique values in each row
  - A corrupted $\boldsymbol{P_i}$ can generate a new truth table from scratch
- We prevent all of the above using non-interactive zero-knowledge and signatures
  - Made efficient using Fiat-Shamir on Sigma protocols (Fiat-Shamir requires random oracle model ☹)
- All messages are signed to prevent replacement

Zero-Knowledge Proofs

- For, say, MAJ function, server proves that the first half of the table entries are encryptions to 0 and the rest are to 1
  - This statement can be reduced to proving that $\boldsymbol{n+1}$ tuples are Diffie-Hellman tuples $(\boldsymbol{O(n)}$ exponentiations)

- Each party needs to prove an OR of two statements.
- Namely that it generated a rerandomization of the ciphertexts, after decrypting its key and removing either the first or last table entry
  - Using [CDS], this is double the cost of a Diffie-Hellman proof

Concrete Cost (Malicious)

- Pretty efficient!
- Server computes $\mathbf{2}\boldsymbol{n}+\mathbf{4}(\boldsymbol{n}+\mathbf{1})$ exponentiations
- Each $\boldsymbol{P_i}$ computes less than $\mathbf{8}\boldsymbol{n}^{\mathbf{2}}$ exponentiations
  - It has to verify **all** previous proofs and generate own

- With $\boldsymbol{n}=\mathbf{100}$, each party computes at most $\mathbf{80,000}$ exponentiations (many less)
  - Can be done in about a minute

- Certainly practical for Program Committee vote
  - 40 parties: less than 12,800 exponentiations each, taking about 10 seconds

Additional Practical Protocols

- Symmetric functions over $\mathbb{Z}_{\boldsymbol{c}}$
  - Extension of previous protocol
- Sum function over large domain
  - Uses additively homomorphic encryption
- Selection functions
  - Different ideas…

A General Result

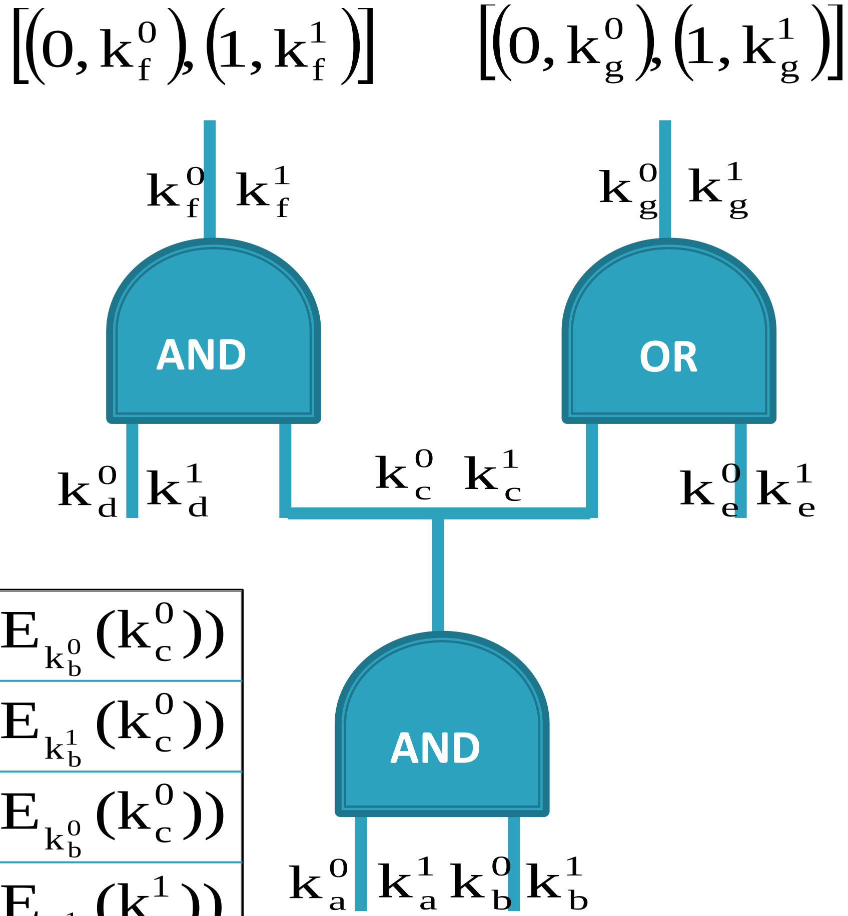
- Theorem:
  - Any decomposition $f_1, \ldots, f_n$ can be securely computed, under the DDH assumption and assuming NIZK (for security against malicious adversaries)
- The main tool: rerandomizable Yao circuits
  - As in the i-Hop homomorphic encryption [GHV], with some modifications
- Given a garbled circuit, rerandomize all labels
  - This involves rerandomizing ciphertexts AND keys
  - As in i-Hop, this can be done using [BHHO]
  - Labels are balanced bit strings, encrypted bit by bit

Protocol Outline

- $\boldsymbol{P_1}$'s instructions
    - Generate a rerandomizable Yao circuit for $\boldsymbol{f_1}$
    - Encrypt the input labels (determining its own input) under all parties' public keys (as in the efficient solution)
    - (For malicious: prove correct behavior and sign)

- Any other party $\boldsymbol{P_i}$
    - Decrypt with its key all encrypted labels of previous parties
    - Add a rerandomizable Yao circuit for $\boldsymbol{f_i}$, and join it to the previous circuit
    - Rerandomize the previous circuit (so that even its creator won't recognize its new labels)
    - (For malicious: prove correct behavior and sign)

Yao Circuit



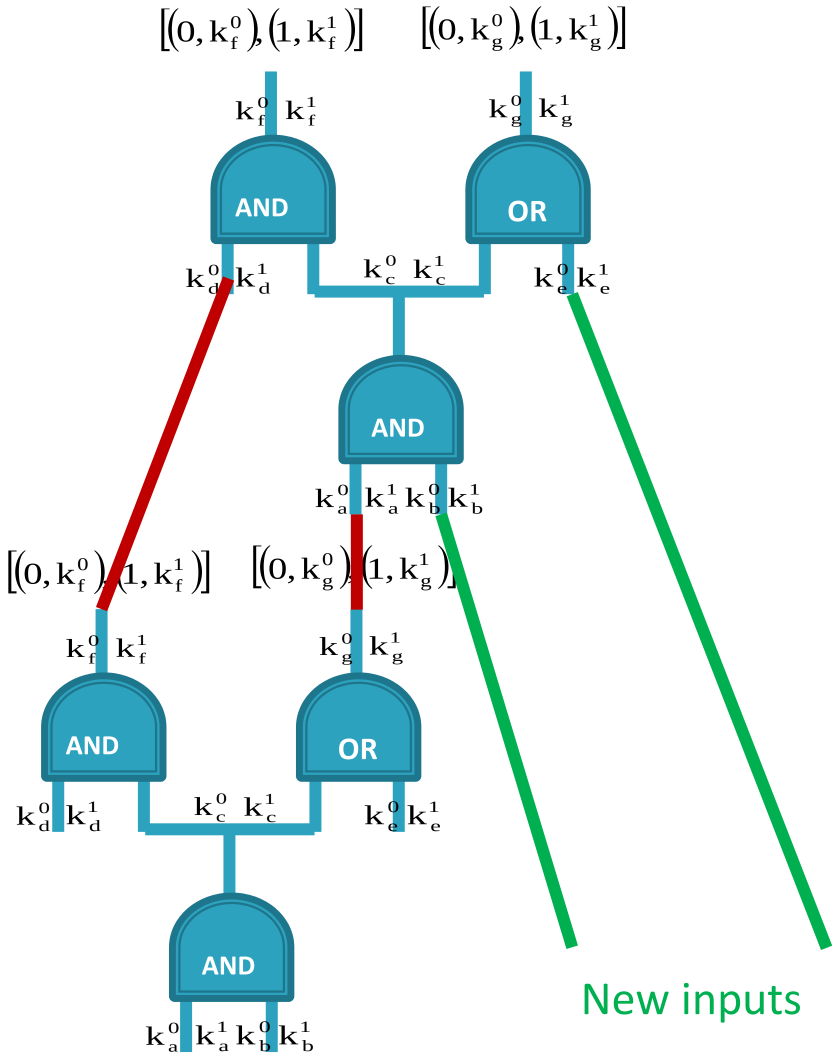
$E_{k_d^0}(E_{k_c^0}(k_f^0))$
$E_{k_d^0}(E_{k_c^1}(k_f^0))$
$E_{k_d^1}(E_{k_c^0}(k_f^0))$
$E_{k_d^1}(E_{k_c^1}(k_f^1))$

$E_{k_c^0}(E_{k_e^0}(k_g^0))$
$E_{k_c^0}(E_{k_e^1}(k_g^1))$
$E_{k_c^1}(E_{k_e^0}(k_g^1))$
$E_{k_c^1}(E_{k_e^1}(k_g^1))$

$E_{k_a^0}(E_{k_b^0}(k_c^0))$
$E_{k_a^0}(E_{k_b^1}(k_c^0))$
$E_{k_a^1}(E_{k_b^0}(k_c^0))$
$E_{k_a^1}(E_{k_b^1}(k_c^1))$

Yao Circuit Composition

- In order to compose, the output keys must equal the input keys
  - I.e., $k_d^0 = k_f^0$ and so on
- This can be done because the output labels (and associated values) are given
- New party also rerandomizes all intermediate values of circuits



New inputs

Security

- By the security of the Yao circuit construction, nothing can be learned without any input labels
  - While there is still at least one honest party, its public key hides all input labels

- Given the input labels of all the honest parties, the remaining corrupted parties only see random intermediate values and the output of the last circuit.
- Can now only compute the remaining circuit on different inputs of their own

Summary

- Fully interactive secure computation is a problem in practice
  - A one-pass client/server is essential for many applications, and is also interesting from a theoretical point of view
- There are inherent limitations to the model
  - Captured by computing function decompositions
- Under the DDH assumption
  - Any function decomposition can be computed
  - Highly efficient and practical protocols exist for many natural problems in this setting
  - New results [GMRW]: sparse polynomials, branching programs