Techniques for Efficient Secure Computation Based on Yao's Protocol

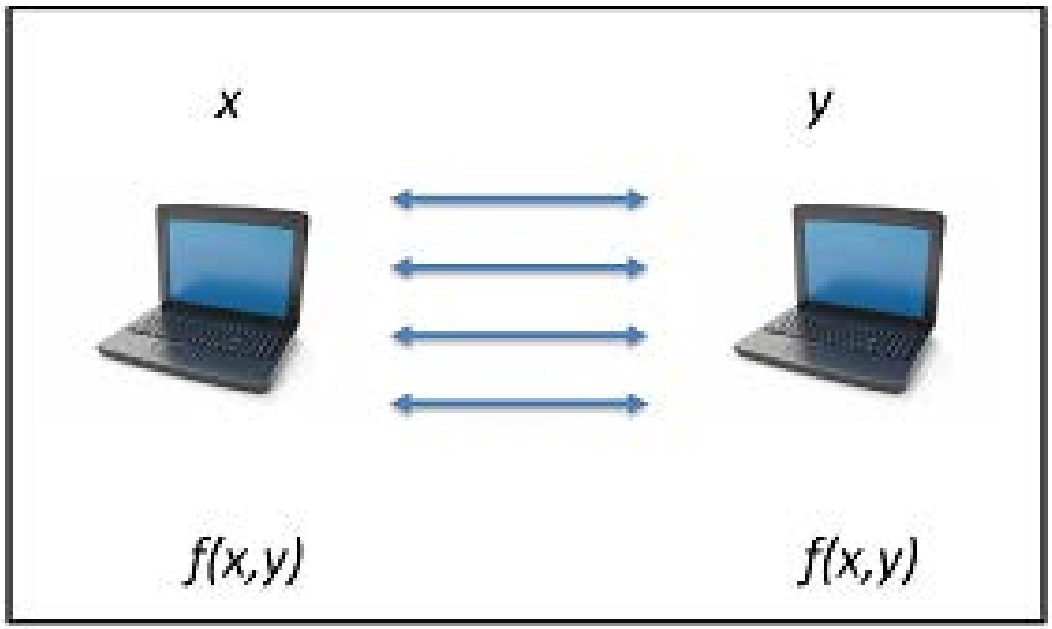
Yehuda Lindell
Bar-Ilan University, Israel

PKC 2013

Secure Computation – Background

A set of parties $P_1, \ldots, P_m$ with private inputs $x_1, \ldots, x_m$ wish to compute a joint function f of their inputs while preserving secure properties such as:

- Privacy: nothing but the output $f(x_1, \ldots, x_m)$ is revealed
- Correctness: the correct output is obtained
- Independence of inputs: no party can choose its input as a function of another party's input



Secure Computation – Background

In an election:

- Privacy means that individual votes are not revealed
- Correctness means that the candidate with the majority vote wins
- Independence of inputs means that you can't vote as a function of the outcome

Secure Computation – Background

Security must hold in the presence of **adversarial behavior**:

- Semi-honest: follows the protocol description but attempts to learn more than allowed
  - Models inadvertent leakage but otherwise gives a weak guarantee

Secure Computation – Background

Security must hold in the presence of **adversarial behavior**:

- Malicious: follows any arbitrary attack strategy
  - Provides a very strong guarantee, but is hard to achieve with respect to efficiency

Security is formalized by comparing the output of a secure protocol to an **ideal world** where an incorruptible trusted party computes the function for the parties

Secure Computation – Feasibility

Despite its stringent requirements, it was shown that essentially **any function can be securely computed**:

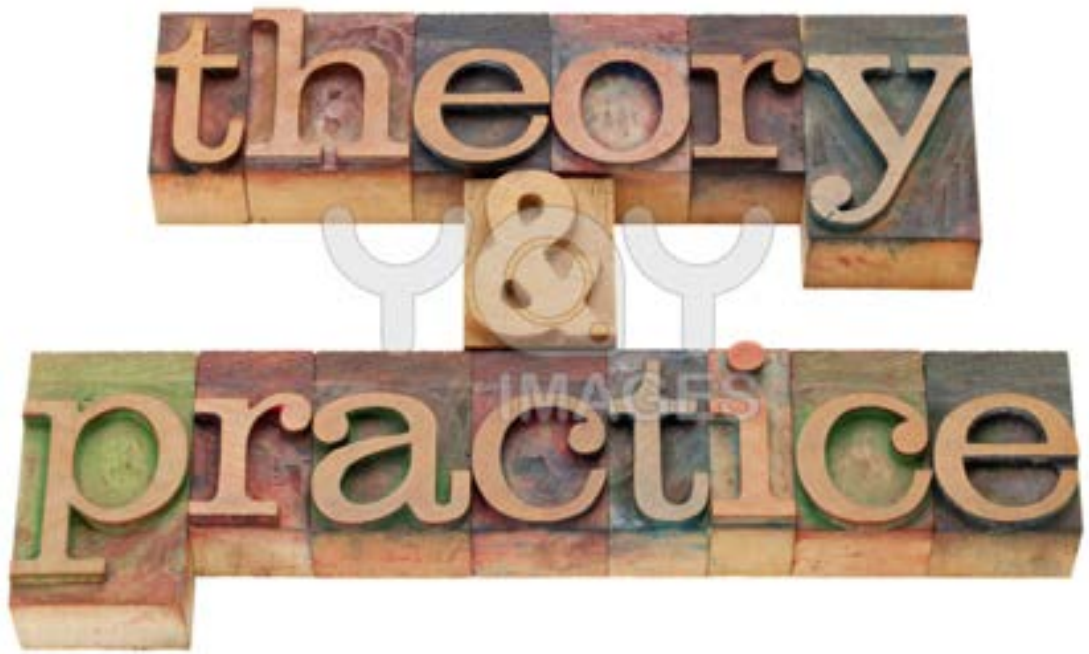
- In the presence of semi-honest adversaries [Yao86,GMW87]
- In the presence of malicious adversaries [GMW87]
- With perfect security where a 2/3 honest majority is guaranteed [BGW88]

Since the 1980s, the **feasibility** of secure computation has been studied heavily:

- Assumptions
- Stronger adversaries (e.g., adaptive corruptions)
- Composition
- And much much more...

Secure Computation – Theory or Practice?

- Due to its broad applicability, secure computation has been a foundational theoretical topic of study since the mid 1980s
  - A rich and beautiful theory has been developed
- Recently, interest has grown with respect to the **practicality** of secure computation
  - Governments, security organizations, industry,...



Secure Computation in Practice?

In the last 5 years there has been incredible progress on making secure computation practical

- Today we can run semi-honest secure computation for problems like secure AES in tens of milliseconds
  - We can run huge computations (on circuits of over a billion gates) in minutes
- We have protocols for malicious adversaries that give amazing amortized complexity
- Every year there are new significant breakthroughs

This is very surprising (and exciting): we now know that secure computation can be practical for a reasonably wide range of problems

- Ten years ago, no one dreamed that this would be possible

Efficient Secure Computation – Semi-Honest Adversaries

From 2004 to 2013

- Yao's protocol from 1986 has a constant number of rounds and uses a few symmetric encryptions per gate
  - For many years, it was assumed that any protocol that is based on a circuit for computing the function cannot be practical
- In 2004, the first implementation of a general secure computation protocol was carried out
  - Fairplay – an implementation of Yao's protocol for semi-honest adversaries
  - It was surprising to many that a circuit-based protocol could even run
  - The billionaires' problem on 32-bit integers took between 1.25 seconds (LAN) and 4.01 seconds (WAN)
  - Median on ten 16-bit numbers (circuit of size 4383 gates) took between 7.09 and 16.63 seconds

Efficient Secure Computation – Semi-Honest Adversaries

From 2004 to 2013

- In 2011, an implementation of Yao for semi-honest adversaries was carried out, using the state-of-the-art algorithmic improvements, and systems optimizations
  - Secure AES computation (with 9,280 non-XOR gates) took just 0.2 seconds overall (after an additional 0.6 seconds of preprocessing that can be used for many executions)
  - In 2013, we can do even better

Secure Computation – Malicious Adversaries

From 2004 to 2013

- In 2004, there were **no** efficient protocols whatsoever (the only way to achieve this level of security was via general zero-knowledge proofs for $\mathcal{NP}$)
  - There were protocols that need exponentiations per gate; e.g., [SchoenmakersTuyls2004]
  - These protocols can be efficient for small circuits but do not scale well
- In 2013, we have a **number of efficient protocols** [NO09,IPS09,DO10,LOP11,BDOZ11,NNOS12,DPSZ12]
  - One important and influential approach is based on Yao's garbled circuits [Y86,LP07,LP11,sS11]
  - This approach appears to still give the lowest latency in a model with no preprocessing
  - In 2012, an implementation of secure AES computation took < 30 seconds on 4-cores, and about 8 seconds on 16-cores

The problem of authentication and one-time passwords:

- Users have devices that compute a PRF of the current time etc. to generate one-time passwords
  - The cryptographic keys for one-time password generation are stored at a server
- A server breach means that all devices must be replaced (very costly and problematic, and so is avoided)
- The danger can be mitigated using secure computation
  - Share the key between two servers
  - In order to verify a one-time password, securely compute AES (without revealing anything about the key), and then verify
- The same method can be used to verify "bank transaction signing"

Secure Computation in Practice

Secure AES Computation

Data Breach at Security Firm Linked to Attack on Lockheed

By CHRISTOPHER DREW and JOHN MARKOFF
Published: May 27, 2011

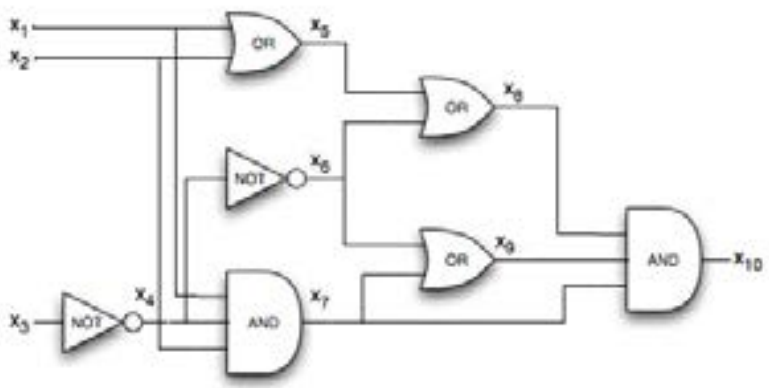
Lockheed Martin, the nation's largest military contractor, has battled disruptions in its computer networks this week that might be tied to a hacking attack on a vendor that supplies coded security tokens to millions of users, security officials said on Friday.

The SecurID electronic tokens, which are used to gain access to computer networks by corporate employees and government officials from outside their offices, are supplied by the RSA Security division of the EMC Corporation.

RSA acknowledged in March that it had sustained a data breach that could have compromised some of its security products. Executives in the military industry said Friday that Lockheed's problems appeared to stem from that data breach and could be the first public signs of damage from it.

General versus Specific Protocols

- A general protocol can be used to compute any functionality (based on the **circuit** or some other general representation)
- For many years it was assumed that general protocols cannot compete with specific protocols
- In some cases, this may be true, but in many cases general protocols are the best we know
  - **And they are good!**
- Efficient general protocols have more applicability, and they save us having to guess what people want to compute
  - For years we talked about elections and auctions, but it appears that one-time password computation is of much more interest



This Talk

Efficient Secure Computation Based on Yao's Protocol

- We will briefly review Yao's basic protocol
- We briefly mention the major techniques for improving efficiency in the semi-honest settings
- We will focus on how to deal with malicious adversaries
  - Understanding the problem and difficulty
  - The cut-and-choose technique and subtleties
  - An optimization to reduce bandwidth
  - New developments

Yao's Garbled Circuits

A **garbling** of a circuit C is an "encryption" of the circuit with the following properties

- Two secret keys are associated with each input wire; one for the 0-bit and one for the 1-bit
- Given a single key for each input wire, it is possible to compute the associated output and nothing else. That is:
  - Given the keys associated with bits $x_1, \ldots, x_n \in \{0,1\}$, it is possible to compute $f(x_1, \ldots, x_n)$
  - Given the keys associated with $x_1, \ldots, x_n \in \{0,1\}$ it is not possible to learn anything beyond $f(x_1, \ldots, x_n)$
- How can garbled circuits be constructed?

A Garbled Gate

Input wires i and j, and output wire ℓ

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

A plain AND gate

x	y	$x \wedge y$
k_i^0	k_j^0	k_ℓ^0
k_i^0	k_j^1	k_ℓ^0
k_i^1	k_j^0	k_ℓ^0
k_i^1	k_j^1	k_ℓ^1

The associated keys (garbled values)

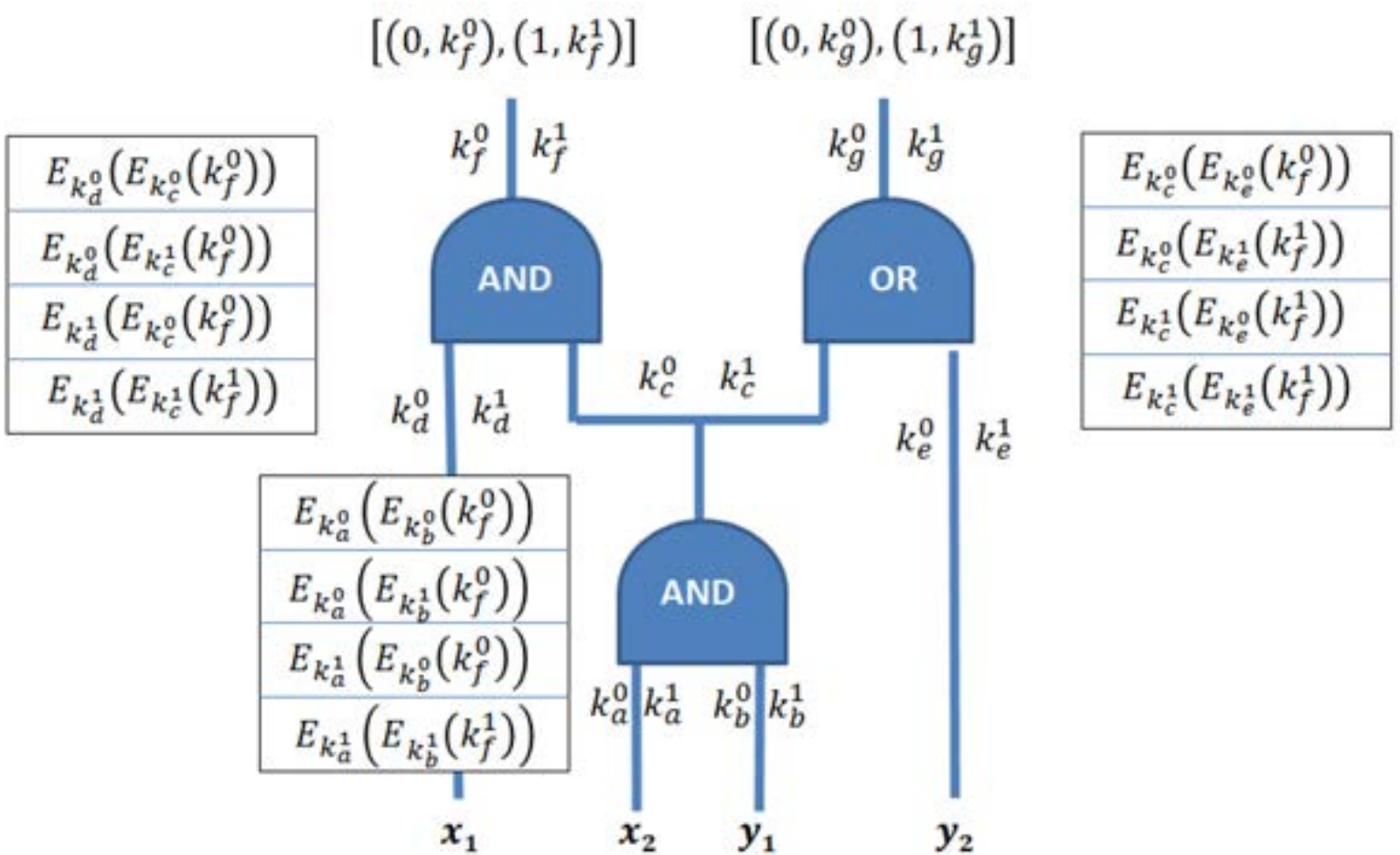
Ciphertexts
$E_{k_i^0}\left(E_{k_j^0}\left(k_\ell^0\right)\right)$
$E_{k_i^0}\left(E_{k_j^1}\left(k_\ell^0\right)\right)$
$E_{k_i^1}\left(E_{k_j^0}\left(k_\ell^0\right)\right)$
$E_{k_i^1}\left(E_{k_j^1}\left(k_\ell^1\right)\right)$

The garbled gate (in random order)

- Given k_i^α and k_j^β for some $\alpha, \beta \in \{0, 1\}$, can obtain $k_\ell^{\alpha \wedge \beta}$
- But, nothing is revealed by this since all keys are random!

A Garbled Circuit

Input wires d, a, b, e and output wires f, g



- Garbled gates can be combined together naturally
- Given one key for every input wire, can compute the entire circuit without learning anything but the output

Garbled Circuits

Correct computation:

- How does the circuit evaluator know which decryption is correct?
- Can include redundancy, but then the evaluator has to try all 4 (on average 2.5)
- Choose random "selector bits" that point to the correct ciphertext (these are random so reveal nothing about the association between the key and the bit)

A protocol for securely computing $f(x, y)$:

- **Inputs:** P_1 has x, and P_2 has y
- Party P_1 constructs a garbled circuit computing the function f and sends it to party P_2
- Party P_1 sends the keys associated with its input x to P_2
- P_1 and P_2 run 1-out-of-2 oblivious transfer for every bit of P_2's input
  - In the ith OT, P_2 inputs y_i (its ith input bit) and P_1 inputs the pair of keys k_i^0, k_i^1 associated with this input wire
  - P_2 receives $k_i^{y_i}$ and learns nothing about $k_i^{\overline{y_i}}$
- Given one key for every input wire, P_2 computes the garbled circuit, obtains the output $f(x, y)$, and sends it to P_1

P_1 corrupted:

- P_1 learns nothing in the OTs and only sees $f(x, y)$
- This view is easy to simulate given the input and output

P_2 corrupted:

- P_2 learns a single key only for every input wire
  - This is trivial for P_1's input wires
  - This follows from the security of OT for P_2's input wires
- From the above, P_2 learns nothing but the output from the garbled circuit
- This view is simulated by constructing a garbled circuit that just outputs the prescribed output

Optimizations for Yao's Circuits

A Brief Look

- **Double-encryption optimizations** [LPS08,BHR12]: garbled gate naively costs 8 encryptions to generate and 2 to evaluate; this can be reduced to a half (at the expense of assumptions)
- **Free XOR gates** [KS08]: it is possible to choose the garbled values so that XOR gates can be computed by just XORing the input wires
- **Garbled row reduction** [PSSW09]: reduce the number of ciphertexts to transmit from 4 to 3 (save bandwidth; a real bottleneck)
- **Circuit optimizations:** make circuits smaller, and with more XOR gates and less AND gates (a new engineering problem)
- **Oblivious transfer extensions** [IKNP03]: compute 128 real OTs once, and derive many OTs from hash calls only
- **Pipelined execution** [HEKM11]: Split the circuit into parts and have the parties compute in parallel

Yao's Protocols with Malicious Adversaries

The Problems

- **The OT must be secure for malicious adversaries**
  - This was a problem 5 years ago: the best protocols required $O(n)$ exponentiations
  - In 2008, this was solved by [PVW] (stand-alone model version in [HL10]): the cost is $11m + 15$ regular DDH exponentiations for m transfers
- **The circuit may not be correctly constructed**
  - This is not just a problem of correctness, but also of privacy
  - The circuit can compute a different function of the evaluator's input, revealing something that should remain secret

Ensuring Correctness of the Circuit

The cut-and-choose paradigm:

- P_1 constructs many copies of the circuit
- P_2 challenges P_1 on half of them
- P_1 opens the requested half and P_2 checks that are correct
- The parties evaluate the remaining circuits and take output

Cut-and-Choose on Yao's Protocol

Opening a Pandora's Box

We solve a problem but generate many new ones:

- The parties compute many circuits: we need to force them to use the same inputs in all
- Opening a circuit means providing all keys on input wires: it may be possible to construct a circuit with two sets of keys – one opening it to the correct circuit and one to a different circuit
- The circuits may be correct, but the garbled keys may not be: P_1 can give invalid 0-keys for the first bit of P_2's input
  - If the first bit of P_2's input is 0, then it cannot compute and so must abort
  - If the first bit of P_2's input is 1, then it computes
  - Thus, P_1 can learn the first bit of P_2's input by observing if it aborts or not
  - This is called a **selective bit attack** [KS06]

Cut-and-Choose on Yao's Protocol

Another Problem

What should P_2 do if not all computed circuits give the same output?

- Observe that a few circuits may be incorrect with good probability!
- If P_2 aborts, then P_1 can carry out the following attack:
  - P_1 generates one garbled circuit that outputs garbage if the first bit of P_2's input is 0; otherwise it computes f
  - With probability $1/2$, this circuit is not checked
  - If the first bit of P_2's input is 0, it aborts
  - If the first bit of P_2's input is 1, it does not abort
  - Thus, P_1 can learn the first bit of P_2's input by observing if it aborts or not
- Thus, P_2 cannot abort, even though it knows that P_1 is trying to cheat!

Strategy for Determining Output

Party P_2 cannot abort, and so takes the majority output

- This is sound since the probability that a majority of the unopened circuits are incorrect is negligible (in the number of circuits)
- But, what is the function bounding the probability of cheating?
  - This is important since it determines the number of circuits, which has a huge ramification on efficiency
- An inaccurate computation:
  - Let s be the number of circuits
  - The adversary succeeds if $\frac{s}{4}$ circuits are incorrect and none of them are chosen to be checked
  - Assume each circuit is checked w.p. $1/2$, this occurs with probability $2^{-s/4}$
  - For security of 2^{-40} need 160 circuits

Bounding the Cheating Probability

- In [LP07] a non-tight bound of $2^{-s/17}$ overall was proven
  - We didn't fully appreciate the ramification of this at the time
- In [LP11] this was improved to $2^{-0.311s}$ and so 128 circuits suffice
- In [sS11] it was shown that by checking 60% of the circuits, this can be further improved to $2^{-0.32s}$ and so 125 circuits suffice
  - In [sS11], they show that this is **optimal** and thus cut-and-choose for Yao is stuck at 125 times the cost of semi-honest Yao

Dealing with Malicious Adversaries

Solving the Other Problems

- **P_2's input consistency in all circuits:** this is easily solved within regular oblivious transfer
- **P_1's input consistency in all circuits:** many different solutions (commitment sets, pseudorandom synthesizer and Diffie-Hellman proof, auxiliary circuits, and more)
- **A circuit with a valid and invalid opening:** commit to all the keys when sending the circuit (commitment may be implicit as well)
- **Selective bit attack:** randomize the inputs [LP07], or incorporate the input keys for P_1 into the checks [LP11]

Solving these problems more efficiently is a very active area of research

The Problem of Bandwidth

An Optimization

- For a circuit of 50,000 gates, 125 copies of the circuits requires sending about 400 MB (and in practice even more)
  - In many cases, this will be the bottleneck (especially over the Internet)
- An optimization proposed by [GMS08]:
  - P_1 chooses a random r_i for the ith garbling and generates the garbled circuit using randomness $PRG(r_i)$
  - P_1 sends P_2 a collision-resistant hash of the garbled circuits
  - To open the ith circuit, P_1 sends the seed r_i only (and P_2 checks the hash)
  - To evaluate the ith circuit, P_1 sends the garbled circuit (and P_2 checks the hash)
- This saves half of the communication (or even 60% using [sS11])

Malicious Security via Yao's Garbled Circuits

Where do we go from here?

- It is still possible to optimize the methods used to enforce input consistency and so on, but the bottleneck of 125 circuits cannot be broken
  - This means that unless massive parallelism is used, the cost of malicious security is going to be high
  - It seems that we have to abandon Yao to go further
- But, the proof of optimality of [sS11] assumes that the protocol works by opening and checking some percentage and taking the majority output from the evaluated circuits
- **Can a variant of cut-and-choose be used to reduce the number of circuits?**

Cut-and-Choose Yao with Fewer Circuits [L13]

- Recall the problem: if P_2 aborts when receiving inconsistent outputs, this can leak information to P_1
- We want to design a strategy so that P_1 can only cheat by making all of the checked circuits correct and all of the evaluated circuits incorrect
- If we succeed, then the cheating probability is just $\binom{s}{\frac{s}{2}}^{-1}$
  - To get 2^{-40} security, 44 circuits suffice
- To further improve this, we can have P_2 choose each circuit to check/evaluate independently at random w.p. $\frac{1}{2}$
  - **This gives an error of 2^{-40} with just 40 circuits!**

Dealing with Inconsistent Outputs

The aim: make cheating possible only if all evaluated circuits are incorrect

- **Observation:** the problem occurs only if P_2 receives *different outputs*
  - If not all the circuits evaluate, but the ones that do yield the same output then there is no problem
  - This holds because unless all evaluated circuits are incorrect, at least one is correct and so the output is correct
- **The idea:** if P_2 receives different outputs, then it will learn P_1's input x
  - In this case, P_2 can locally compute $f(x, y)$ and obtain correct output
  - We stress that P_1 cannot know if P_2 learned $f(x, y)$ because all circuits had the same output or because it learned x

Dealing with Inconsistent Outputs

Continued

Implementing the idea:

- The ith output wire must have the same garbled values in all circuits (checked by P_2 in check circuits)
- P_2 first evaluates all the evaluation circuits
- P_1 and P_2 run a new malicious-secure computation for a small circuit, as follows:
  - P_1 inputs the same x as in the main computation
  - P_2 inputs either garbage or two garbled values on a single wire
  - If P_2's input is two garbled values, then P_2 learns x
- Following this, P_1 opens the check circuits and P_2 checks

The New Secure Computation for a Small Circuit

- The secure computation used is one of the previous protocols, like [LP11]
- The circuit can be made very small, using a specific design (see the paper)
  - To be concrete: $2m + \ell - 1$ non-XOR gates, where m is the output length and ℓ is the input length
- The proof that P_1 uses the same x as before is just a regular **input consistency check** that is applied anyway to the main secure computation
  - The checks don't have any problem going across different circuits
  - We proved our protocol using the method of [LP11] but believe that others will work

Conclusions – Malicious Yao with Fewer Circuits

- **We can now achieve malicious security with much fewer circuits**
  - For error 2^{-40} it suffices to send 40 circuits
  - Together with existing optimizations and techniques, this gives us very fast security for malicious adversaries

- **The big question:**
  - What else can be improved and optimized?
  - I conjecture that we are not finished with Yao yet!

The MPC Lounge

For More Information

The **MPC Lounge** has just been opened:

- The aim of the lounge is to be a resource on efficient secure computation
- The lounge has a Wiki, a blog, and pointers to resources
- It is rather empty right now, but we hope that within the next few months it will fill out
- Go to `mpclounge.org`

Summary

- Efficient secure computation is a reality: there is interest and we have fast protocols
  - I strongly believe that we will start seeing secure computation in use in the near future
- Yao's garbled circuits can yield very fast protocols, but there is still more to do
- We have considered only one approach in this talk (garbled circuits):
  - There are a number of very important other approaches [NO09,IPS09,DO10,LOP11,BDOZ11,NNOS12,DPSZ12]
- **Follow this exciting field and join us: the pace is fast and the competition is growing, but we are doing things that we never believed possible just a few years ago!**

Thank You

Thank You!