Cut-and-Choose Yao-Based Secure Computation in the Online/Offline and Batch Settings

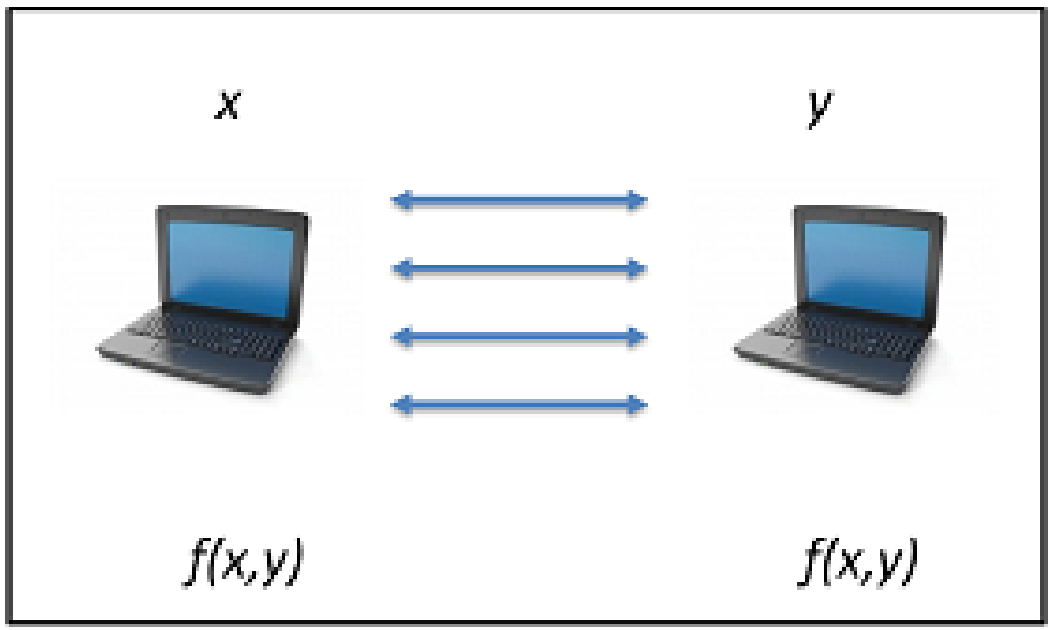
Yehuda Lindell
Bar-Ilan University, Israel

Technion Cryptoday 2014

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

Applications

- **Privacy:**
  - Secure DNA comparison
  - Secure set intersection (compare lists of friends, customers, etc.)
- **Security:**
  - Compute AES without revealing key
  - Private database search and private searchable encryption
  - User authentication without revealing credentials

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

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

In the last 5-10 years, the **efficiency** of secure computation has been a topic of intensive study

- Semi-honest general secure computation is extraordinarily fast
- Security for malicious adversaries has made great progress; but this is still a difficult problem

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)

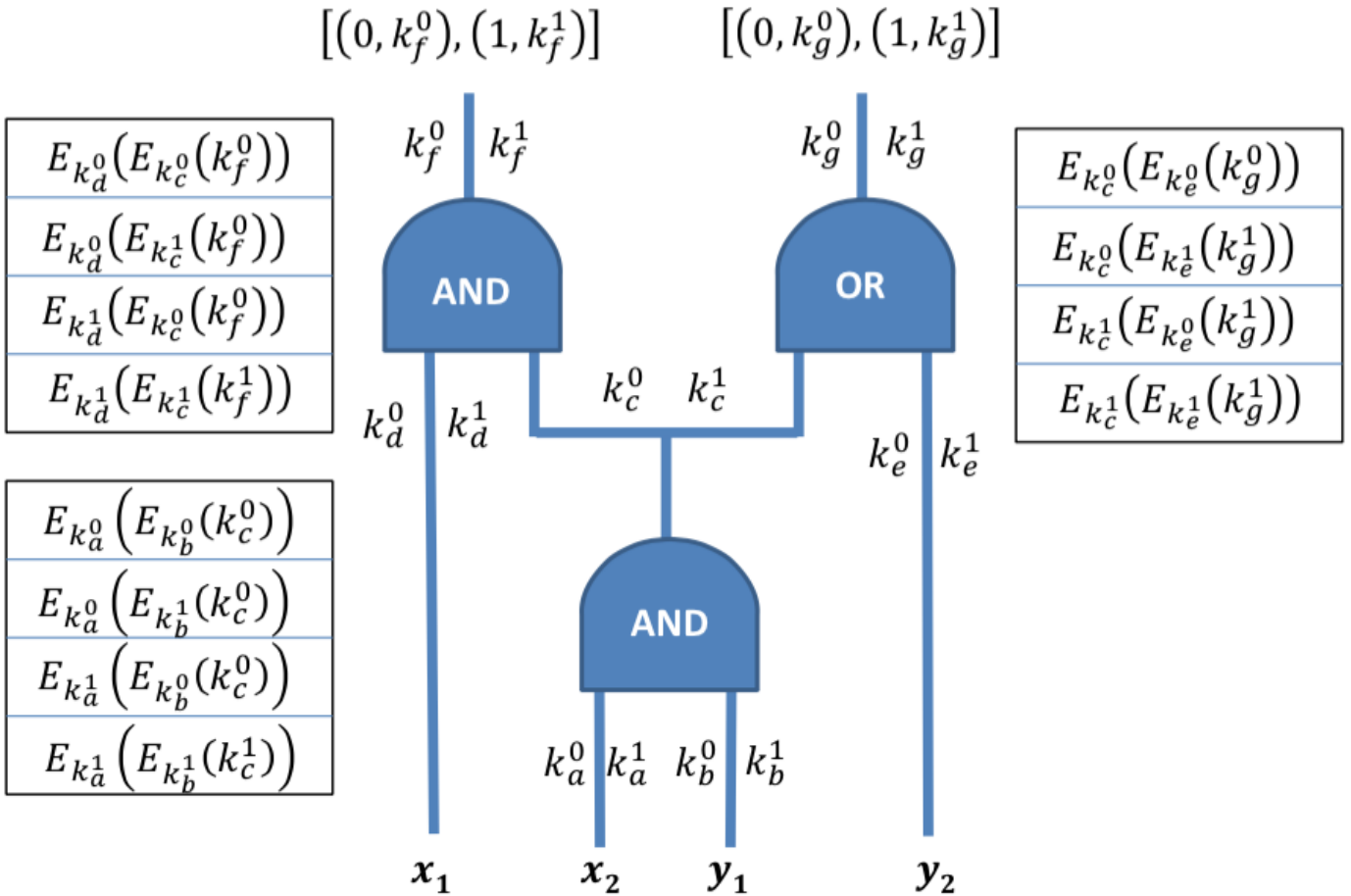
Ciphertexts
$E_{k_i^0}\left(E_{k_j^0}\left(k_\ell^0\right)\right)$
$E_{k_i^0}\left(E_{k_j^1}\left(k_\ell^0\right)\right)$
$E_{k_i^1}\left(E_{k_j^0}\left(k_\ell^0\right)\right)$
$E_{k_i^1}\left(E_{k_j^1}\left(k_\ell^1\right)\right)$

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 Optimizations

There are **many** optimisations:

- Evaluation while only decrypting one ciphertext
- Reducing number of ciphertexts in a gate from 4 to 3 to 2
- Free-XOR and FlexOR
- Reducing the number of encryptions in each gate
- Utilising the AES-NI chip
- Reducing the size of the circuit

Current state of the art:

- Can garble and compute the AES circuit (about 7000 AND gates, and 15,000 XOR gates) in a few hundred microseconds on a standard i7 PC

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

Yao's Protocols with Malicious Adversaries

The Main Problem – Circuit Correctness

Party P_1 may construct an **incorrect circuit**

- 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 – 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
- 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 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
- In [LP11] proved $2^{-0.311s}$ and so 128 circuits suffice
- In [sS11] showed that when checking 60% of the circuits, error is $2^{-0.32s}$ and so 125 circuits suffice (and this is **optimal**)

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

The cheating recovery technique:

- If P_2 receives the same output in all circuits, then this is its output
  - This is fine as long as at least one of the circuits is correct
- If P_2 receives two different outputs, then this releases a trapdoor to P_2 that enables it to learn P_1's actual input x
  - P_1 just locally computes $f(x, y)$ which is certainly correct
  - "Trapdoor" is actually a bootstrapping secure computation: uses 128 circuits, but each is very small

Cheating probability

- As long as at least one evaluation circuit is correct, P_1 cannot cheat
- Thus, P_1 can cheat only if all check circuits are correct and all evaluation circuits are not
- Cheating probability for s circuits: 2^{-s}

Batched and Online/Offline Yao

Presented at CRYPTO 2014; joint work with Ben Riva

- **Consider a setting where many executions take place**
  - **Batch:** many executions run at the same time
  - **Online/offline:** prepare in offline stage, run Yao execution fast in online
- **Amortizing cut-and-choose**
  - P_1 can only cheat if all evaluation circuits are incorrect
  - The aim: make the probability that all evaluation circuits are incorrect be low
  - The idea:
    - Generate many circuits for many executions
    - Open a percentage to check
    - Randomly permute the remainder into buckets for evaluation
- **Why does this help?**
  - If P_1 generates **many** incorrect circuits, it will be caught
  - If P_1 generates **few** incorrect circuits, the probability that a bucket will contain only incorrect circuits is very small

Amortizing Cut-and-Choose

Notation:

- Number of executions: N
- Size of each bucket: B
- Error parameter: s (allow 2^{-s} error)

Naive method: set $B = s$ and use $N \cdot B = N \cdot s$ circuits

- This requires s circuits per evaluation

Randomly permute evaluation circuits:

- Use $2N \cdot B$ circuits; open half
- What value of B is needed?
  - Asymptotically: $\mathcal{O}\left(\frac{s}{\log N}\right)$ circuits per evaluation
  - Concretely: s is small and N is large, so $\frac{s}{\log N} << s$
    - For $s = 40$ and $N = 512$, need 10 circuits per evaluation (5 checked and 5 evaluated)
    - For $s = 40$ and $N = 2^{19}$, only 6 circuits per evaluation

Amortizing Cut-and-Choose

Opening a half is not optimal:

Number of Executions	Probability of Not Checking	Bucket Size	Average circuits per Execution
32	0.75	10	13.34
1,024	0.65	5	7.69
1,024	0.85	6	7.06
1,048,576	0.65	3	4.62
1,048,576	0.98	4	4.08
2^{30}	0.99	3	3.03

The same method is also used for the cheating recovery circuit:

- For N =1024, need 24 circuits per execution (instead of 125)
- For N =1,048,576, need 12 circuits per execution

Amortizing Cut-and-Choose

Summary:

- For 1024 executions, the online phase requires the evaluation of 5 full circuits (and 24 tiny bootstrapping circuits)
- A single AES circuit can be evaluated in $\approx \mathbf{350}$ **microseconds**; with multithreading should yield an online phase of just a few milliseconds

Combinatorics of Multiple Cut-and-Choose

An inaccurate analysis: $N = 1000$ executions, buckets of size 10, open half

- Adversary wins if there exists an all-bad bucket
- Adversary constructs 20,000 circuits
  - If at least 40 are bad, probability that none are checked is 2^{-40}
  - If less than 40 are bad, these are randomly thrown into 1000 buckets
  - If there is independence (**which there isn't**) then each ball is bad w.p. $40/10000 = 0.004$
  - If there is independence, a bucket gets 10 bad balls with probability $0.004^{10} \approx 2^{-80}$

Challenges in Online/Offline and Batch Yao

- **Input consistency:**
  - Current methods for ensuring P_2's input consistency are applied to all circuits before opening
  - The inputs are only determined after opening in online/offline; mapping of circuits to buckets is unknown
- **Cheating recovery:**
  - All circuits have same output values on wires
  - In multiple execution setting, this is not possible (and certainly different values need to be obtained on different wires)
- **Moving checks to the offline phase:**
  - We wish to check consistency of P_1's and P_2's inputs in the offline phase
  - The input is not yet determined; how is this possible?

Changing the Functionality

Instead of computing $f(x, y)$, the parties compute

$$f'((x_1, x_2), (y_1, y_2)) = f(x_1 \oplus x_2, y_1 \oplus y_2)$$

where x_1, y_1 are private, and x_2, y_2 are public

Private versus public inputs:

- Consistency of public inputs is easy:
  - P_1 commits to garbled values on wires in their **correct** order
  - P_2 verifies that decommitments are to all first, or all second
  - P_2 can also simply tell P_1 its public input in this stage
- Consistency of private inputs is hard, and uses known methods

Online/offline:

- Parties choose random x_1, y_1 for private inputs in offline
- After receiving x, y, parties define $x_2 = x \oplus x_1$ and $y_2 = y \oplus y_1$ and reveal x_2, y_2 to each other

Reducing the Online Work

- **Our protocol moves almost all work to the offline phase**
  - In online send garbled values, and almost nothing else; compute a few circuits and hashes only
- **This raises a problem regarding adaptive Yao circuits**
  - Standard Yao simulation requires that the input/output of the corrupted be known before constructing a "fake" circuit
  - In the standard malicious setting, this is achieved by running the the OT that determines the input before sending the circuit
- **In the online/offline setting, the inputs have not yet been fixed**
  - In the ROM, there exist solutions to this
  - In the standard model, we do not know how to achieve online time that is independent in the circuit size
- Solving adaptive Yao has important applications here

Cut-and-Choose Yao over the Years

Progress on the number of circuits (for 2^{-40} error):

- Lindell-Pinkas 2007: error $2^{-s/17}$; requires 680 circuits (but conjectures to be $2^{-s/4}$) and so 160 circuits
- Lindell-Pinkas 2011: error $2^{-0.311s}$; requires 128 circuits
- shelat-Shen 2011: error $2^{-0.32s}$; requires 125 circuits (optimal for this approach)
- Lindell 2013: error 2^{-s}; requires 40 circuits (Huang-Katz-Evans 40 circuits in **each direction**)
- Lindell-Riva 2014: amortize cost, **for 2^{-s} error and 1000 executions, 7690 circuits overall and just 5 evaluations in online**

Summary

Online/Offline and Batch Executions with Yao

- Cut-and-choose can be vastly improved when amortized
- Online/offline secure computation can be carried out also with Yao
- This is competitive with the SPDZ and TinyOT approaches since the online phase has few rounds
- Adaptive Yao is an obstacle for standard-model protocols

Thank You

Thank You!