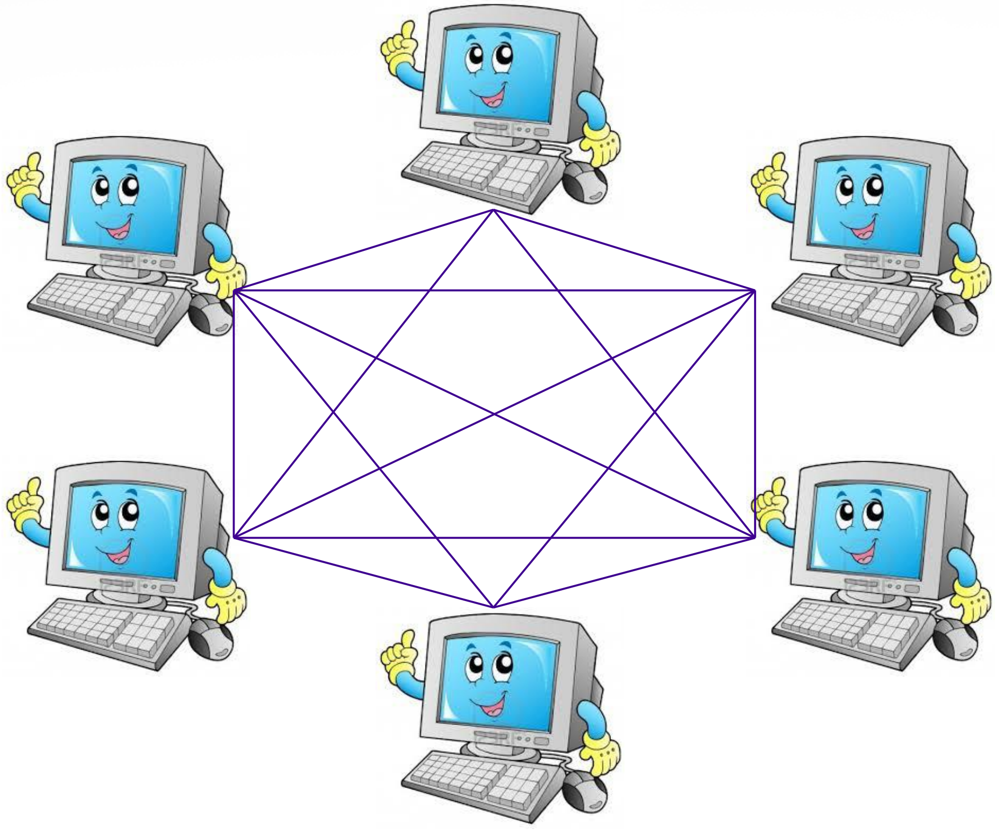
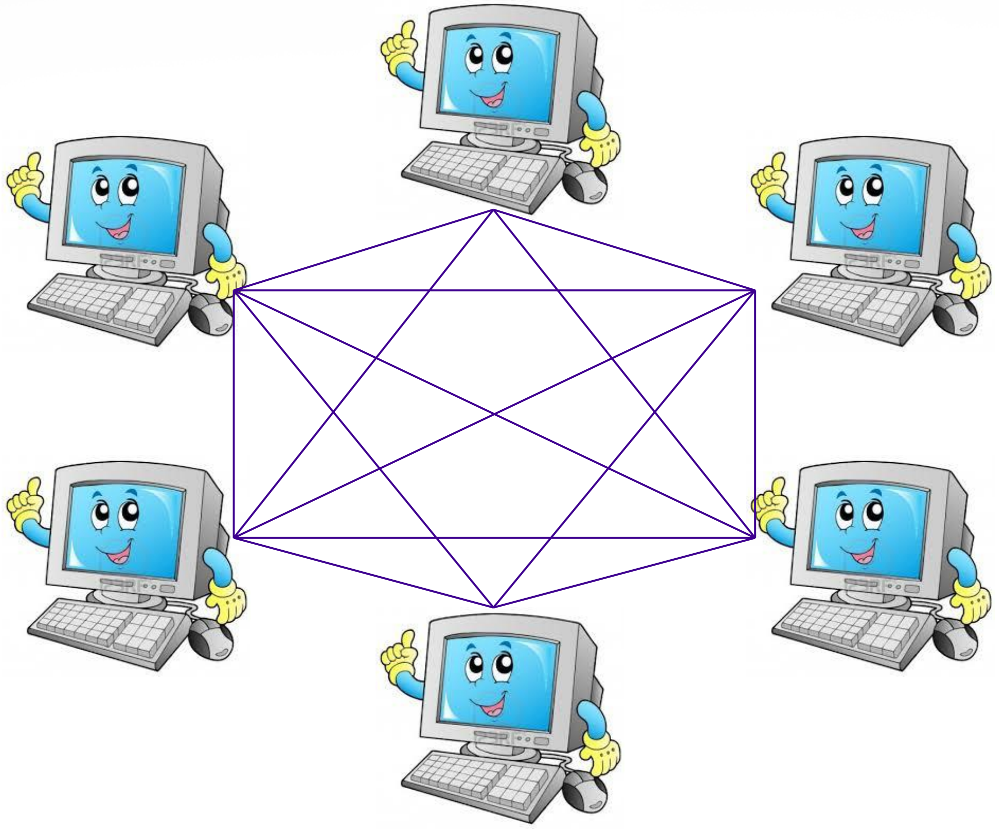
Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions

Sandro Coretti (New York University)
Juan Garay (Yahoo Research)
Martin Hirt (ETH Zurich)
Vassilis Zikas (RPI)

Secure Multi-Party Computation (MPC)
[Yao82, GMW87, BGW88, CCD88, RB89,…]

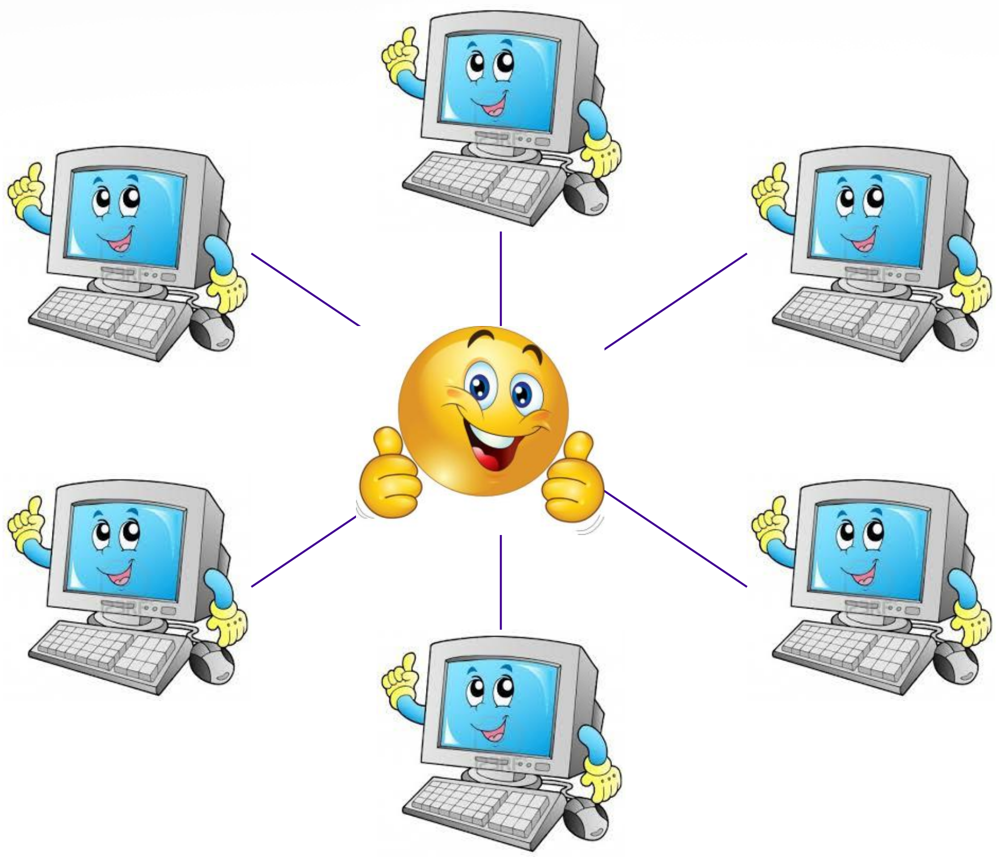


Secure Multi-Party Computation (MPC)
[Yao82, GMW87, BGW88, CCD88, RB89,…]

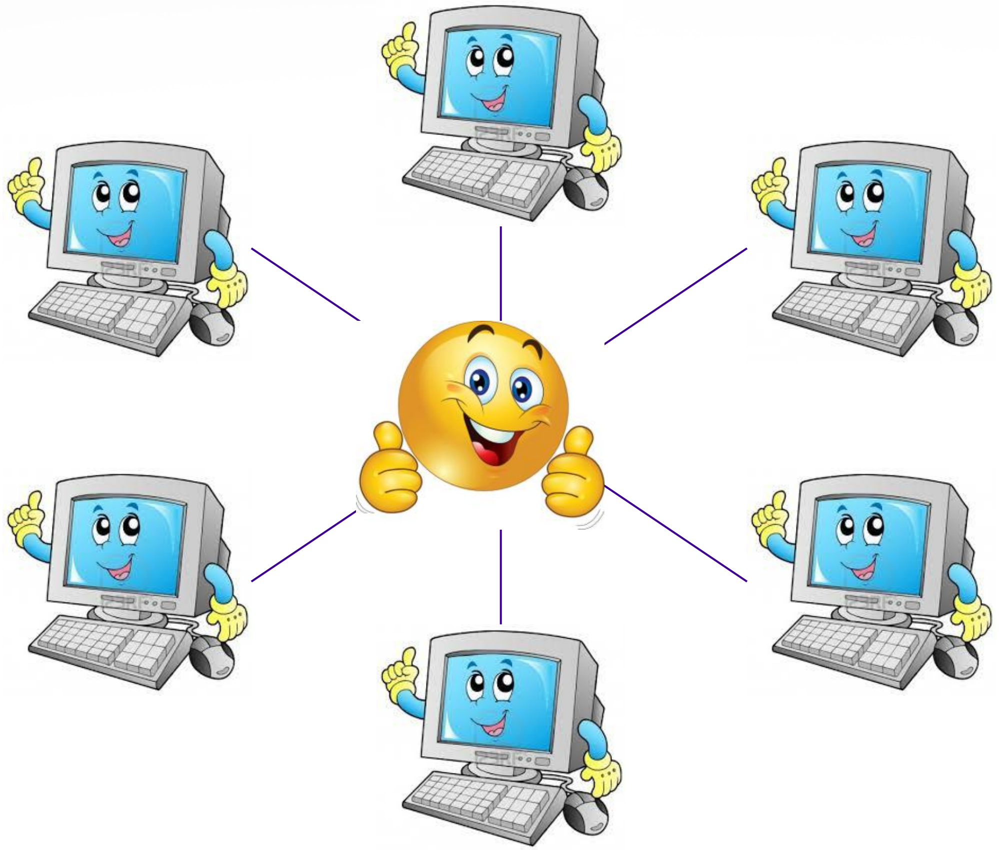


Mutually distrustful parties wish to evaluate function of their inputs

Secure Multi-Party Computation (MPC) (2)
[GMW87, C00, C01,…]

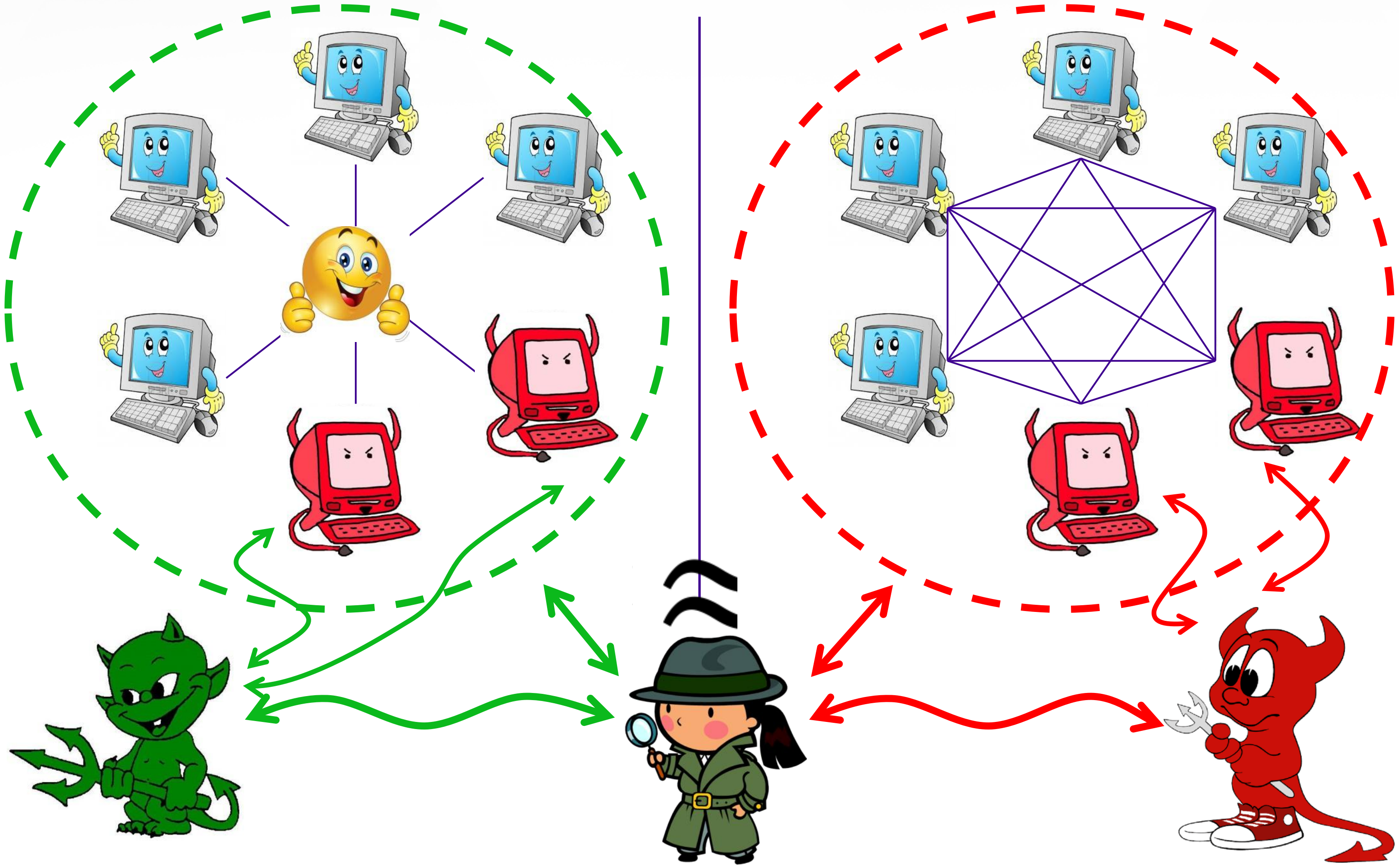


Secure Multi-Party Computation (MPC) (2)
[GMW87, C00, C01,…]



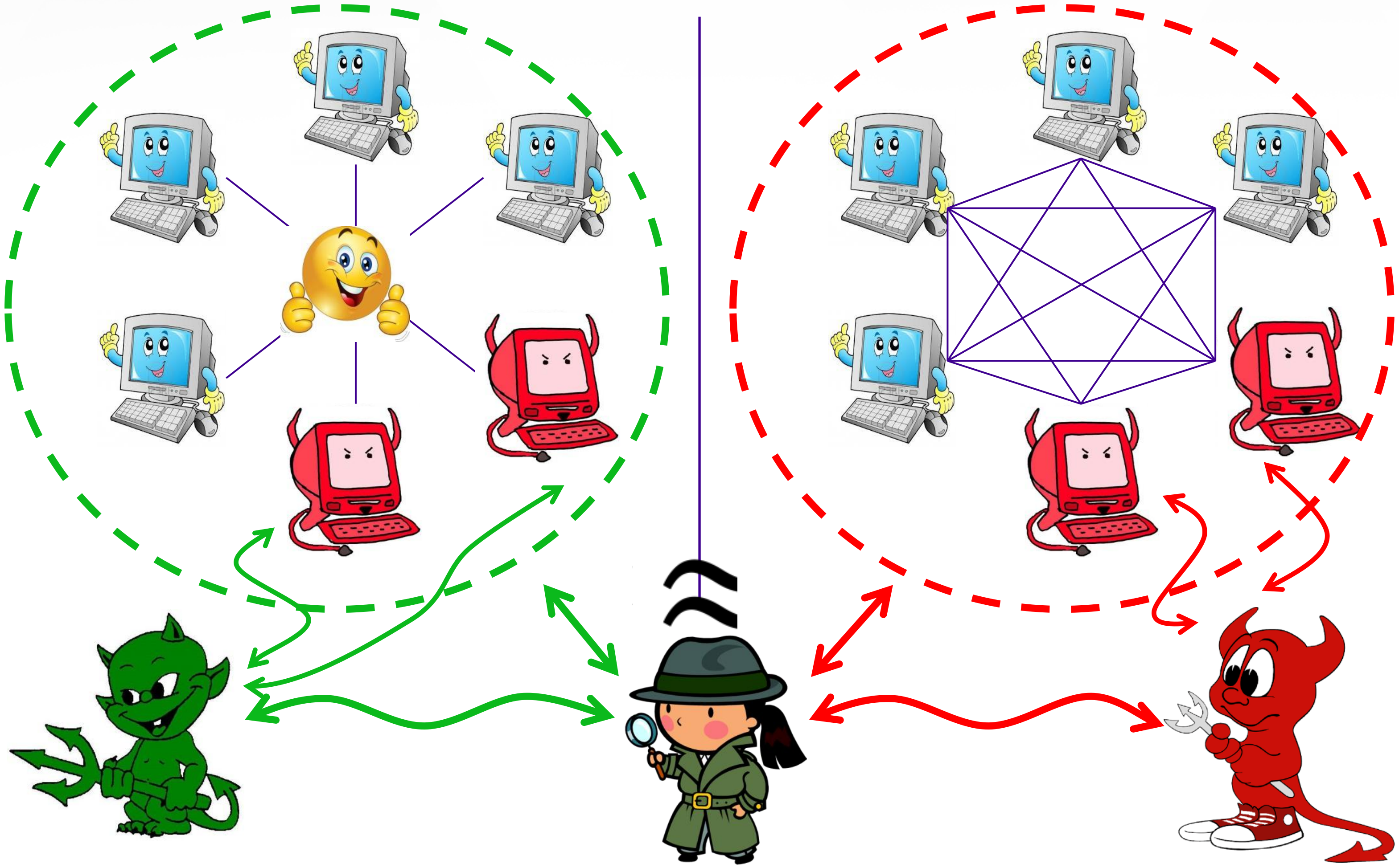
MPC protocol should emulate a trusted third party

Secure Multi-Party Computation (MPC) (3)



YAHOO!

Secure Multi-Party Computation (MPC) (3)



Simulation-based security definition in the Universal Composability (UC) framework [C01]

YAHOO!

Synchronous Communication Network

- Each pair of parties connected by secure channels
- Protocol proceeds in rounds
- Messages sent in particular round guaranteed to arrive by beginning of next round

Synchronous Communication Network

- Each pair of parties connected by secure channels
- Protocol proceeds in rounds
- Messages sent in particular round guaranteed to arrive by beginning of next round
- “Plain” UC framework is inherently asynchronous
  - Adversary has full control over message delivery; may choose to delete messages sent between honest parties
  - “Synchronous” UC using clock functionality and bounded-delay channels [KMTZ13]

YAHOO!

Asynchronous Communication Network

- Synchronous network: great for analysis
  - (Partially) Synchronized clocks + bounded network latency → “timeouts” (T)
  - Round length typically (much) higher than average transmission time

YAHOO!

Asynchronous Communication Network

- Synchronous network: great for analysis
  - (Partially) Synchronized clocks + bounded network latency → “timeouts” (T)
  - Round length typically (much) higher than average transmission time
- UC asynchrony: overly pessimistic

Asynchronous Communication Network

- Synchronous network: great for analysis
  - (Partially) Synchronized clocks + bounded network latency → “timeouts” (T)
  - Round length typically (much) higher than average transmission time
- UC asynchrony: overly pessimistic

“It takes advantage of the nature of information being easy to spread but hard to stifle.”

Asynchronous Communication Network

- Synchronous network: great for analysis
  - (Partially) Synchronized clocks + bounded network latency → “timeouts” (T)
  - Round length typically (much) higher than average transmission time
- UC asynchrony: overly pessimistic

“It takes advantage of the nature of information being easy to spread but hard to stifle.”

Satoshi Nakamoto

YAHOO!

Asynchronous Communication Network (2)

- Each pair of parties connected by secure channels
- Messages sent guaranteed to arrive only *eventually*
- Adversary may:
  - Delay message delivery by arbitrary finite amount of time
  - Reorder messages
  - **Note:** No deletions! (Unlike UC)
- Model considered early on in fault-tolerant distributed computing (e.g., [FLP83]) and asynchronous MPC [BCG93,…]

YAHOO!

Asynchronous Communication Network (2)

- Each pair of parties connected by secure channels
- Messages sent guaranteed to arrive only *eventually*
- Adversary may:
  - Delay message delivery by arbitrary finite amount of time
  - Reorder messages
  - **Note:** No deletions! (Unlike UC)
- Model considered early on in fault-tolerant distributed computing (e.g., [FLP83]) and asynchronous MPC [BCG93,…]
- "Opportunistic": protocols terminate as quickly as the network allows

Asynchronous Communication Network (2)

- Each pair of parties connected by secure channels
- Messages sent guaranteed to arrive only *eventually*
- Adversary may:
  - Delay message delivery by arbitrary finite amount of time
  - Reorder messages
  - **Note:** No deletions! (Unlike UC)
- Model considered early on in fault-tolerant distributed computing (e.g., [FLP83]) and asynchronous MPC [BCG93,…]
- "Opportunistic": protocols terminate as quickly as the network allows
- To date: Asynchronous MPC with eventual delivery not modeled in UC

This Work

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)

This Work

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)
- *Constant-round* MPC protocol
  - I.e., round complexity independent of circuit's multiplicative depth
  - Based on standard assumptions (PRFs)
  - Tolerates $t < n/3$ corruptions
  - Adaptive adversary

Prior Work: Constant-Round MPC Protocols

- Synchronous model:
  - Based on circuit garbling [Yao86, BMR90, DI05, IPS08]
  - Based on FHE [AJLTVW12]
  - $t < n/2$ corruptions
  - Assume broadcast channel (cf. [FL82, BE03, CCGZ16])

Prior Work: Constant-Round MPC Protocols

- Synchronous model:
  - Based on circuit garbling [Yao86, BMR90, DI05, IPS08]
  - Based on FHE [AJLTVW12]
  - $t < n/2$ corruptions
  - Assume broadcast channel (cf. [FL82, BE03, CCGZ16])
- Asynchronous model (recall: eventual delivery):
  - Based on FHE [Coh16]
  - $t < n/3$ corruptions
  - *Static* security
  - Assume A-BA
  - (Other known protocols are GMW-based → circuit depth)

YAHOO!

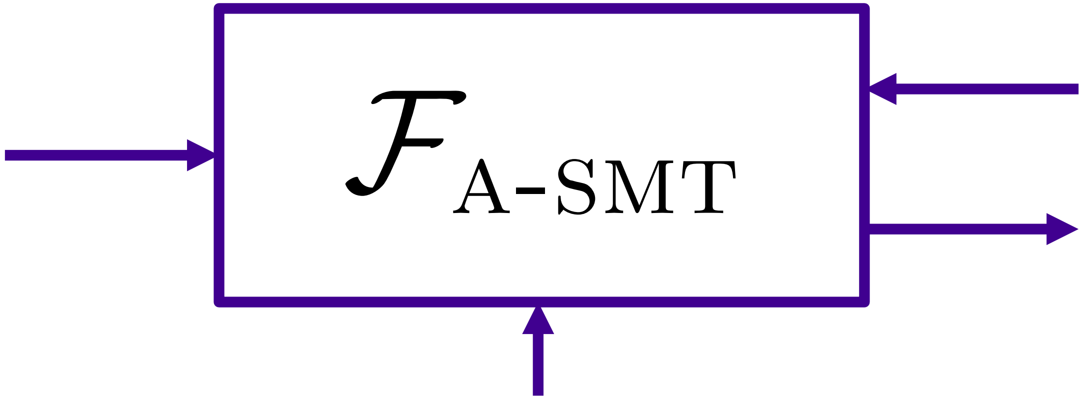
This Work

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)
- *Constant-round* MPC protocol
  - I.e., round complexity independent of circuit's multiplicative depth
  - Based on standard assumptions (PRFs)
  - Tolerates $t < n/3$ corruptions
  - Adaptive adversary

Modeling Asynchronous Communication in UC

Sender

Input messages



Receiver

- Poll for messages: *T* = *T*-1
- If *T* = 0, first message in buffer output

A-SMT Functionality:

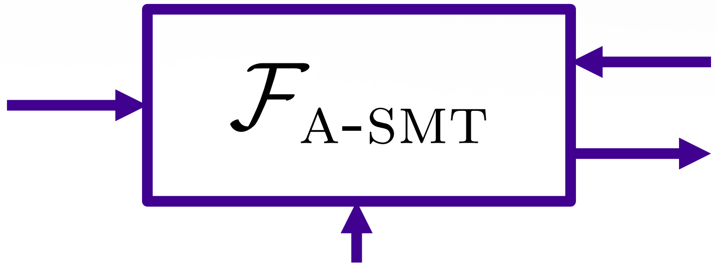
- Stores messages in buffer
- Maintains delay *T*

Adversary

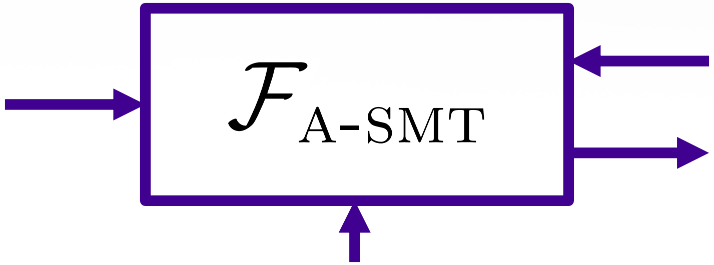
- Reorder messages in buffer
- Increase *T*, specified in **unary**

Modeling Asynchronous Communication in UC (2)

- **Protocol execution:**
  - Party either sends message or
  - polls A-SMT channels in round-robin fashion



Modeling Asynchronous Communication in UC (2)



- **Protocol execution:**
  - Party either sends message or
  - polls A-SMT channels in round-robin fashion

- **Round complexity:** Maximum number of times any party switches between sending and polling

Modeling Asynchronous Secure Function Evaluation in UC

Parties P

- Provide input
- Poll for output: *T* = *T*-1
- If *T* = 0, first message in buffer output

$$\mathcal{F}_{\text{A-SFE}}$$

A-SFE Functionality:

- Collects inputs and computes output
- Maintains delay *T*

Adversary

- Decide on set of *n-t* **input providers**
- Increase *T*, specified in unary

YAHOO!

Modeling Asynchronous Byzantine Agreement in UC

Parties P

- Provide input
- Poll for output: *T* = *T*-1
- If *T* = 0, first message in buffer output

$\mathcal{F}_{\text{A-BA}}$

A-BA Functionality:

- Maintains delay *T*
- Collects inputs and computes output
  - If there is agreement in *C* output corresponding value
  - Otherwise, output a value specified by attacker

Adversary

- Decide on set *C* of *n*-*t* **input providers**
- Increase *T*, specified in unary

YAHOO!

This Work

- Formalize asynchronous model with *eventual delivery* in the UC framework
  - Asynchronous round complexity
  - Basic communication resources: async. secure channel (A-SMT) and async. Byzantine agreement (A-BA)
- *Constant-round* MPC protocol
  - I.e., round complexity independent of circuit's multiplicative depth
  - Based on standard assumptions (PRFs)
  - Tolerates $t < n/3$ corruptions
  - Adaptive adversary

Our Constant-Round Async. MPC Protocol

- UC-realizes A-SFE in (A-SMT, A-BA)-hybrid model
- Function computed specified by Boolean circuit
- Computational security against adversary adaptively corrupting up to $t < n/3$ parties (optimal [BCG93, Can95])
- Constant-round
- Black-box from one-way functions

YAHOO!

Protocol Overview

- Three phases for computing Boolean circuit C:

I. Compute distributed version of garbled circuit

  - Evaluate **constant-depth** function using asynchronous (unconditionally secure) MPC protocol by [BKR94] (whose round complexity depends on depth of evaluated circuit)

II. With output from Phase I, complete circuit garbling

III. *Locally* evaluate garbled circuit

Circuit Garbling [Yao86,BMR90]

- **Idea:** Associated with every wire w of **Boolean** circuit C:
  - mask m_w (to hide actual value on wire) and
  - two keys $k_{w,0}$, $k_{w,1}$
- Evaluate circuit on masked values while maintaining invariant:

If masked value is z, $k_{w,z}$ is known and $k_{w,1-z}$ is secret

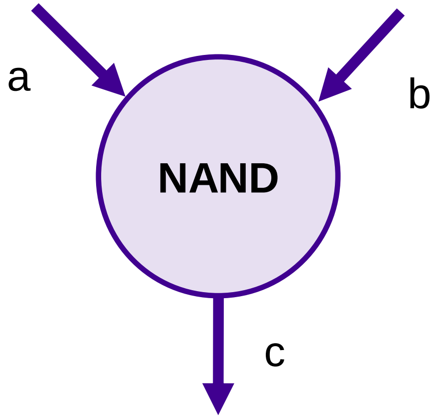
YAHOO!

Circuit Garbling [Yao86,BMR90] (2)

z_1	z_2	Masked Output Bit z	Garbled Entry
0	0	$((0 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$E(k_{a,0}, k_{b,0}, z \,\|\|\, k_{c,z})$
0	1	$((0 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$E(k_{a,0}, k_{b,1}, z \,\|\|\, k_{c,z})$
1	0	$((1 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$E(k_{a,1}, k_{b,0}, z \,\|\|\, k_{c,z})$
1	1	$((1 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$E(k_{a,1}, k_{b,1}, z \,\|\|\, k_{c,z})$

To evaluate garbled circuit, use:

- Masked values on input wires and corresponding keys
- Masks of output wires



YAHOO!

Issue 1

- Evaluating encryption function in MPC → non-black-box

Issue 1

- Evaluating encryption function in MPC → non-black-box
- **Solution**: "Distributed encryption" [DI05]

YAHOO!

Issue 1

- Evaluating encryption function in MPC → non-black-box
- **Solution**: "Distributed encryption" [DI05]

Regular encryption: $E(k,m)$

YAHOO!

Issue 1

- Evaluating encryption function in MPC → non-black-box
- **Solution**: "Distributed encryption" [DI05]

Regular encryption: $E(k,m)$

Distributed encryption:

- Use sub-keys $k_1,\ldots,k_n$ instead of k
- Secret-share m
- Give i^{th} share m_i and k_i to party P_i
- P_i computes $E(k_i,m_i)$ and sends to all

Circuit Garbling with **Distributed Encryption**

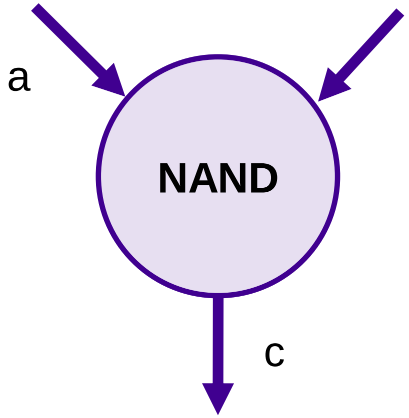
- **Idea:** Associated with every wire w of circuit C:
  - mask m_w (to hide actual value on wire) and
  - two key *sets* $\boldsymbol{k}_{w,0}$, $\boldsymbol{k}_{w,1}$, **each consisting of *n* subkeys**
- Evaluate circuit on masked values while maintaining invariant:

If masked value is z, $\boldsymbol{k}_{w,z}$ is known and $\boldsymbol{k}_{w,1-z}$ is secret.

YAHOO!

Circuit Garbling **without** Distributed Encryption

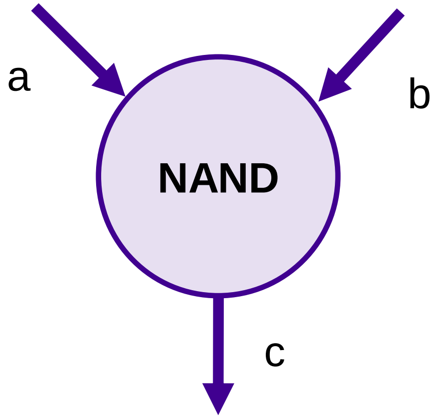
z_1	z_2	Masked Output Bit z	Garbled Entry
0	0	$((0 + m_a)$ NAND $(0 + m_b)) + m_c$	$E(k_{a,0}, k_{b,0}, z \,\|\|\, k_{c,z})$
0	1	$((0 + m_a)$ NAND $(1 + m_b)) + m_c$	$E(k_{a,0}, k_{b,1}, z \,\|\|\, k_{c,z})$
1	0	$((1 + m_a)$ NAND $(0 + m_b)) + m_c$	$E(k_{a,1}, k_{b,0}, z \,\|\|\, k_{c,z})$
1	1	$((1 + m_a)$ NAND $(1 + m_b)) + m_c$	$E(k_{a,1}, k_{b,1}, z \,\|\|\, k_{c,z})$



Circuit Garbling **with** Distributed Encryption

z_1	z_2	Masked Output Bit z	Garbled Entry
0	0	$((0 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$[\, z \,, \mathbf{k}_{c,z} \,]$
0	1	$((0 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$[\, z \,, \mathbf{k}_{c,z} \,]$
1	0	$((1 + m_a) \text{ NAND } (0 + m_b)) + m_c$	$[\, z \,, \mathbf{k}_{c,z} \,]$
1	1	$((1 + m_a) \text{ NAND } (1 + m_b)) + m_c$	$[\, z \,, \mathbf{k}_{c,z} \,]$

Instead of encrypting garbled entry, compute secret-sharing of (each component of) it



YAHOO!

Phase I: Setting the Stage for Garbling with Distributed Encryption

Phase I: Described by (randomized) constant-depth function that

- Randomly chooses masks and subkeys
- Computes masked inputs and corresponding subkeys based on player inputs and masks
- Computes shared function tables (can be done in parallel)
- Outputs to P_i:
  - Masked inputs and corresponding subkeys
  - i^{th} shares of all shared function tables
  - Masks of output wires

YAHOO!

Phase I: Setting the Stage for Garbling with Distributed Encryption (2)

- Actual Phase I: Evaluate Phase I function using [BKR94] protocol
- Round complexity of [BKR94] depends on depth of evaluated circuit
- **But:** Phase I function is constant-depth!

Issue 2

- [BKR94] protocol evaluates arithmetic circuits
- Phase I function described by Boolean circuit
- → Conversion to circuit over extension field of GF(2)
  - Replace each NAND gate with inputs x,y by a computation of $1-xy$
- Ensure that all inputs are 0,1 as follows:
  - After input phase, for every input x, jointly open $x - x^2$ [BGN05]
  - If result is 0, accept x, otherwise replace by 0

YAHOO!

Protocol Overview

- Three phases for computing Boolean circuit C:

  I. Compute distributed version of garbled circuit
     - Evaluate **constant-depth** function using asynchronous (unconditionally secure) MPC protocol by [BKR94] (whose round complexity depends on depth of evaluated circuit)

  II. With output from Phase I, complete circuit garbling

  III. Locally evaluate garbled circuit

YAHOO!

Phases II + III: Encrypting and Evaluating

- Phase II: Compute encryption of garbled entries
  - Each party P_i locally encrypts its shares with the appropriate subkeys and sends resulting ciphertexts to all

Phases II + III: Encrypting and Evaluating

- Phase II: Compute encryption of garbled entries
  - Each party P_i locally encrypts its shares with the appropriate subkeys and sends resulting ciphertexts to all
- Phase III: Locally evaluate garbled circuit
  - Decryption of a function table entry with decryption subkeys $k_1,\ldots,k_n$:
    - Upon receiving encrypted share from P_i, decrypt it with k_i
    - Wait until $2t+1$ shares on degree-t polynomial received and interpolate

Recap: ***Constant-Round*** Async. MPC Protocol

- UC-realizes A-SFE in (A-SMT, A-BA)-hybrid model
- Function computed specified by Boolean circuit
- Computationally secure against adversary adaptively corrupting up to $t < n/3$ parties (optimal [BCG93, Can95])
- Constant-round
- Black-box from one-way functions

Full Version

- S. Coretti, J. Garay, M. Hirt and V. Zikas, “Constant-Round Asynchronous Multi-Party Computation Based on One-Way Functions.” Cryptology ePrint Archive Report 2016/208

http://eprint.iacr.org/2016/208

YAHOO!

Thanks!