CSE/CEN 598
Hardware Security & Trust

Secure Computation Approaches: Security Protocols

Prof. Michel A. Kinsy

Foundations of Secure Computing

- Security protocols
  - Multi-party computation, zero-knowledge, oblivious transfer, security models, etc.
- Homomorphic encryption (HE)
  - Hardware and software implementations
- Design and implementation of trusted platform modules (TPMs)
  - TPM-based anonymous authentication, signature, encryption, identity management, etc.
- Trusted execution environments (TEEs)
  - TEE-based security and privacy techniques, vulnerability and countermeasures of TEE, distributed TEE, decentralized TEE, etc.

Foundations of Secure Computing

- Security protocols
  - Multi-party computation, zero-knowledge, oblivious transfer, security models, etc.
- Homomorphic encryption (HE)
  - Hardware and software implementations
- Design and implementation of trusted platform modules (TPMs)
  - TPM-based anonymous authentication, signature, encryption, identity management, etc.
- Trusted execution environments (TEEs)
  - TEE-based security and privacy techniques, vulnerability and countermeasures of TEE, distributed TEE, decentralized TEE, etc.

Threshold Secret Sharing Scheme

- Select
  - *p* a large prime number and
  - S as the secret value
  - $s_1,\ldots,s_{k-1}$ a set of randomly numbers from [0, p-1]
- A (k, n) threshold polynomial can be written by

$$s(x) \equiv S+s_1x+s_2x^2+\ldots+s_{k-1}x^{k-1} \pmod p$$

- Send $(x_i, s(x_i))$ to the *i*-th participant
- Secret sharing in distributed systems provides
  - Fault-tolerant
  - Multi-factor authentication
  - Multi-party authorization

Threshold Secret Sharing Scheme

- Secret Reconstruction
  - To reconstruct the secret S, one needs to collect at least k partial secrets
  - The secret can then be reconstructed using Lagrange interpolation

$$s(x) \equiv \sum_{j=1}^{k} \left[s(x_j) \prod_{i=1, i \neq j}^{k} \frac{x - x_i}{x_j - x_i} \right] \bmod p$$

- The scheme can be extended to support share renewal and share recovery

Oblivious Transfer

- Oblivious Transfer refers to the technique of transferring a specific piece of data based on the receiver's selection

Alice ———— $\{M_0, M_1\}$ ————→ Bob

Alice sends two messages to Bob

Bob elects to see one of them and only one M_s with $s \in \{0,1\}$

- Alice does not know which one of the two Bob has selected
- Bob is also oblivious to the content of the non-selected message

Oblivious Transfer

- Oblivious Transfer refers to the technique of transferring a specific piece of data based on the receiver's selection

Alice —— $\{M_0, M_1, \ldots, M_{n-1}\}$ ——→ Bob

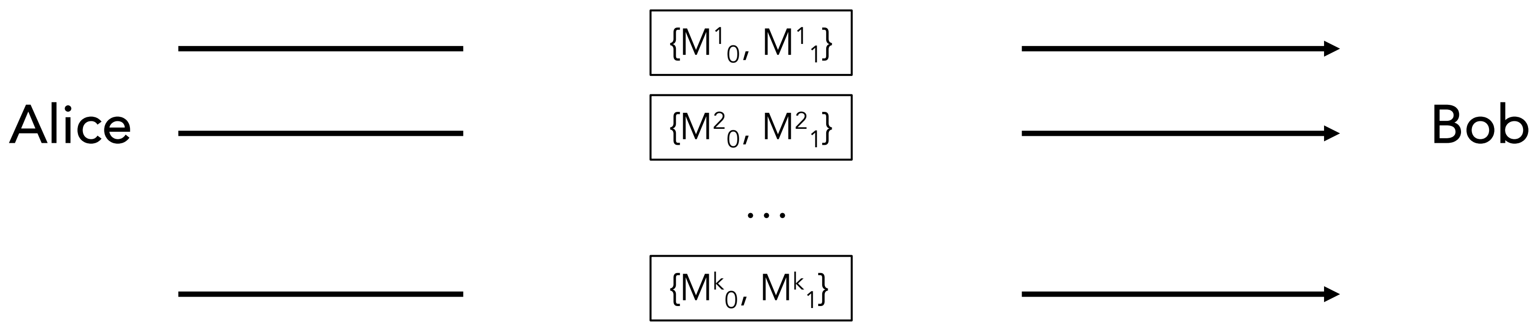
Alice sends two messages to Bob

Bob elects to see one of them and only one M_s with $s \in \{0,1,\ldots,n-1\}$

- Alice does not know which one of the *n* Bob has selected
- Bob is also oblivious to the content of the non-selected message

Oblivious Transfer

- Oblivious Transfer refers to the technique of transferring a specific piece of data based on the receiver's selection



Alice sends two-k messages to Bob

Bob elects to see one-k of them

M^k_s with $s \in \{0,1\}^k$

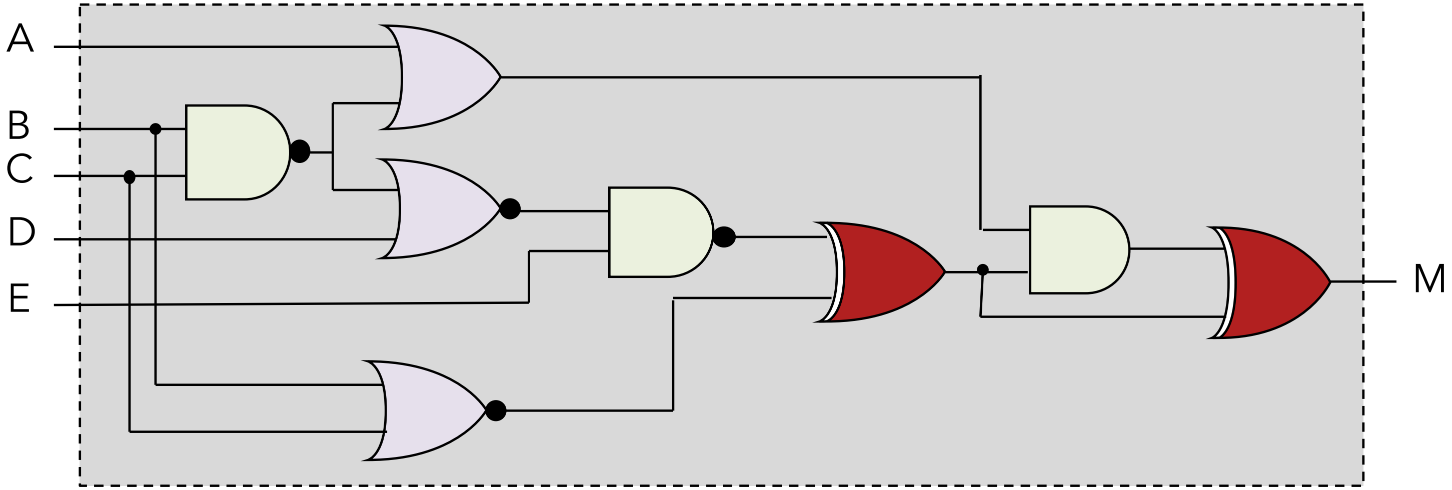
- There are algorithms for optimizing these straightforward implementations

Oblivious Transfer

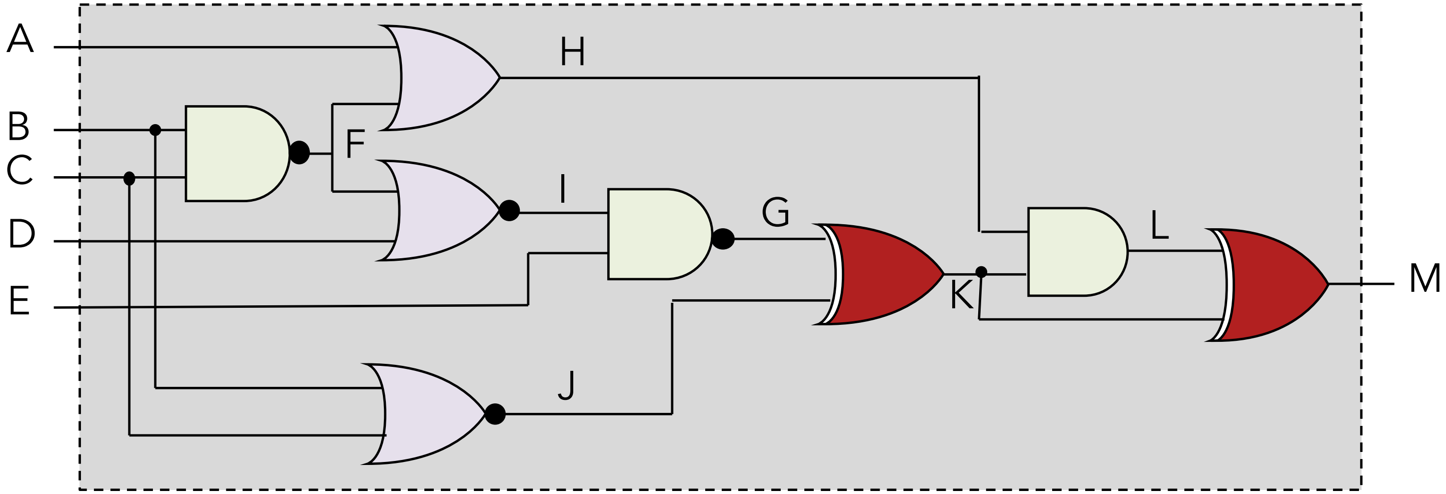
- Oblivious transfer is the necessary and sufficient condition for multiparty computation
- How can one practically perform this oblivious transfer?
  - For that let us introduce garbled circuits
    - Garbling is a process by means of which the Boolean gate truth table is obfuscated

Garbled Circuit

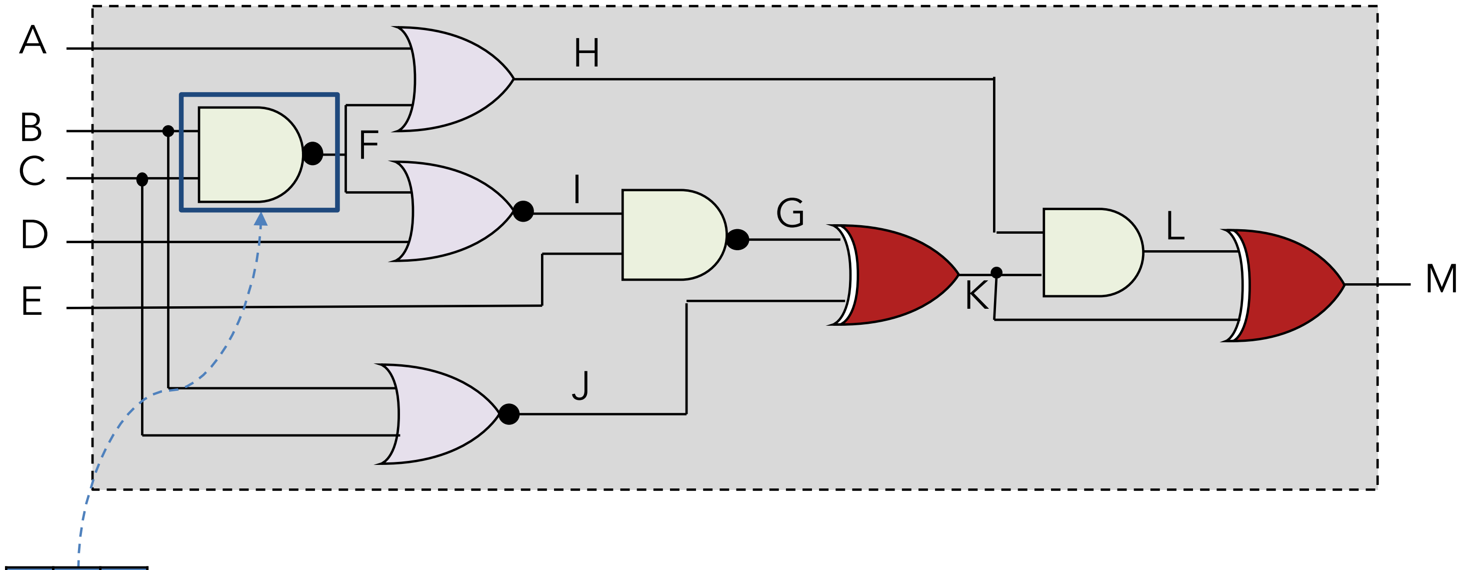
Garbled Circuit



Garbled Circuit

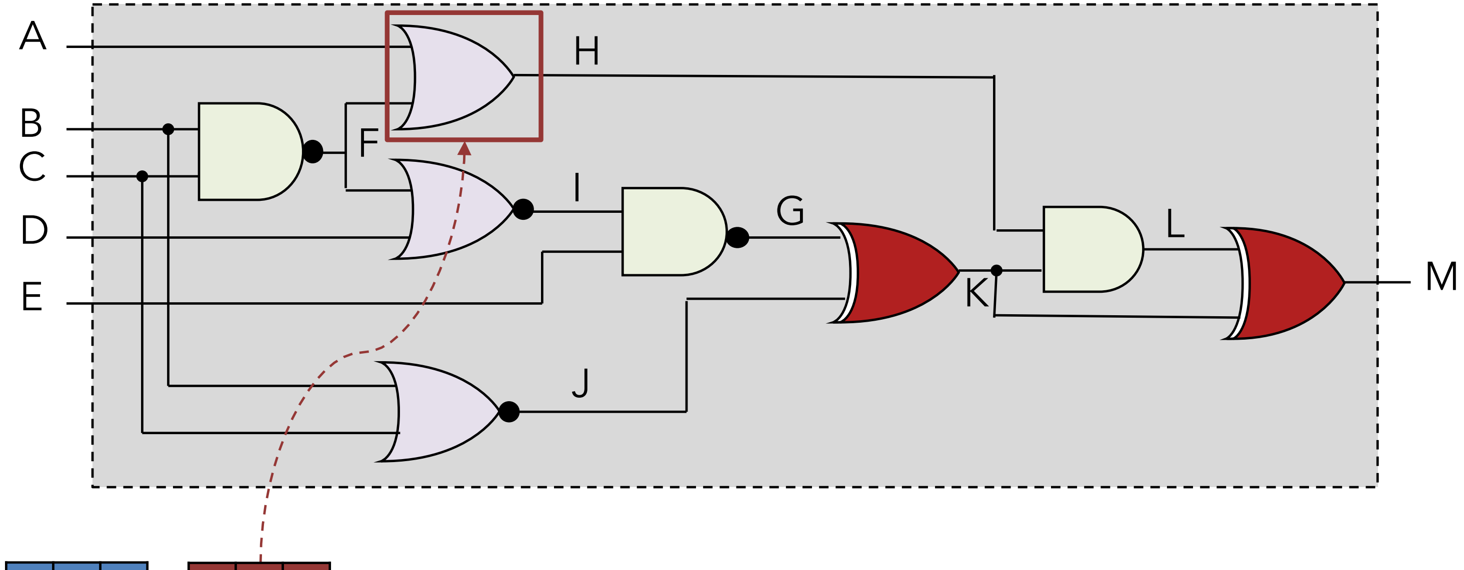


Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

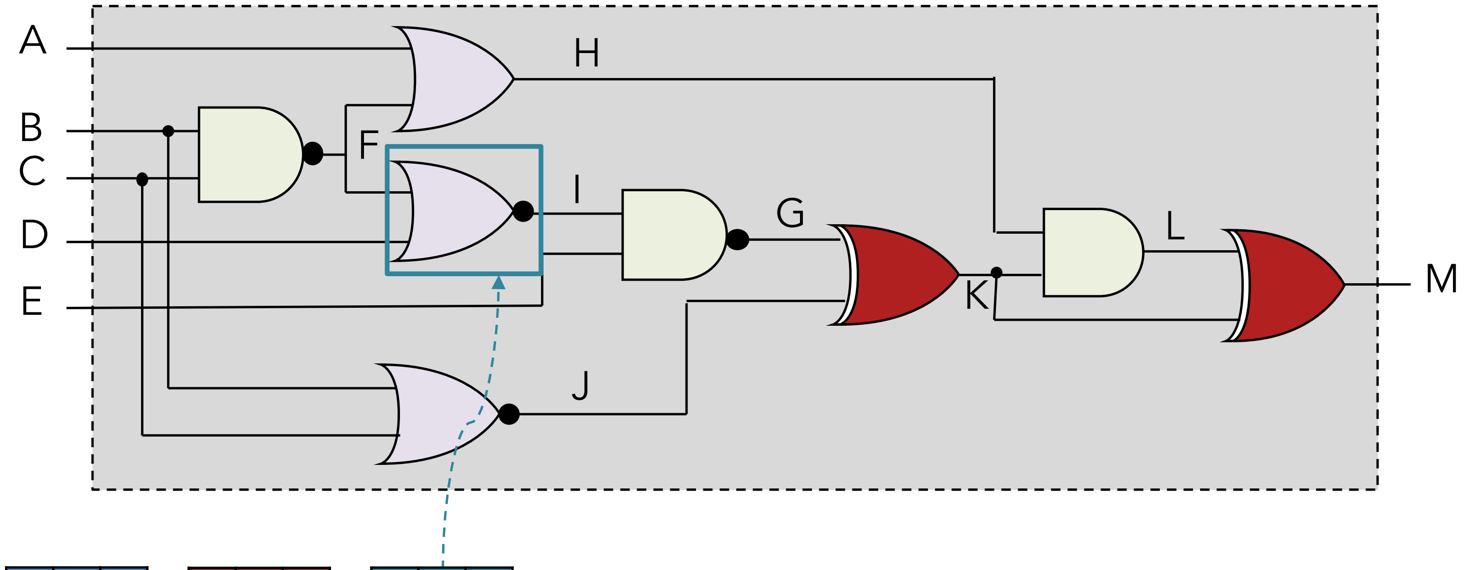
Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

Garbled Circuit

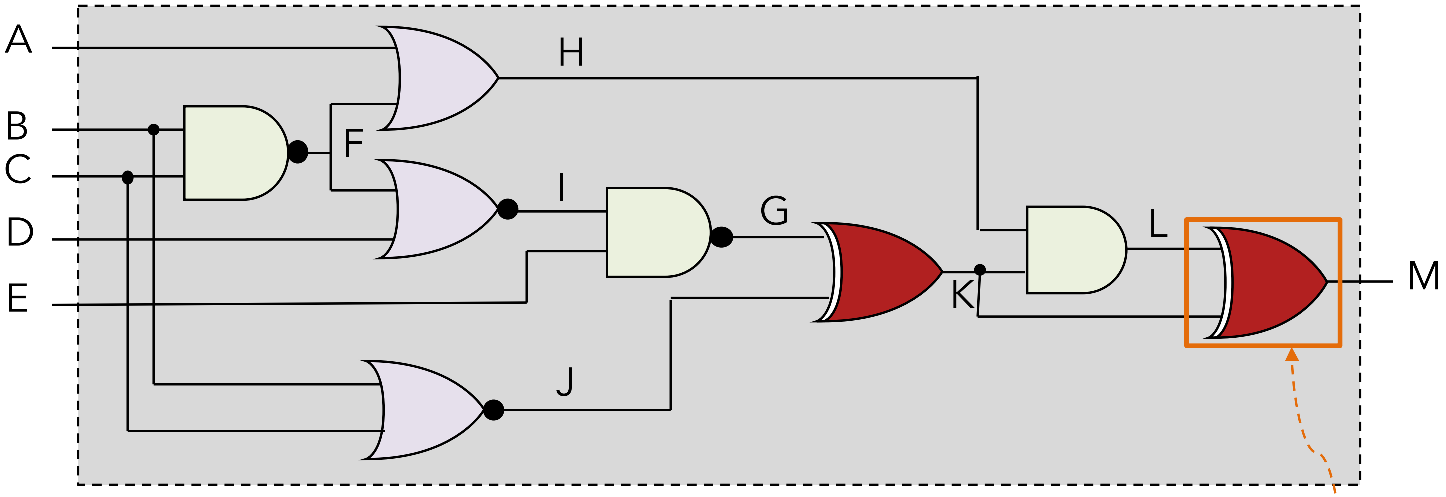


B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

B	C	J
0	0	1
0	1	0
1	0	0
1	1	0

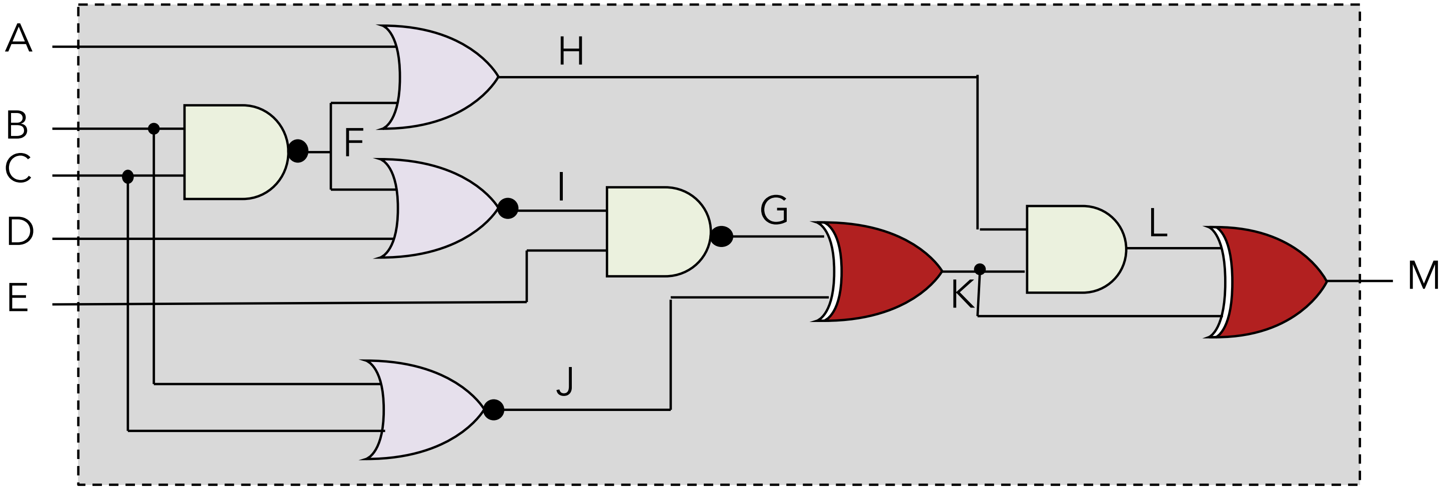
I	E	G
0	0	1
0	1	1
1	0	1
1	1	0

G	J	K
0	0	0
0	1	1
1	0	1
1	1	0

H	K	L
0	0	0
0	1	0
1	0	0
1	1	1

L	K	M
0	0	0
0	1	1
1	0	1
1	1	0

Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

B	C	J
0	0	1
0	1	0
1	0	0
1	1	0

I	E	G
0	0	1
0	1	1
1	0	1
1	1	0

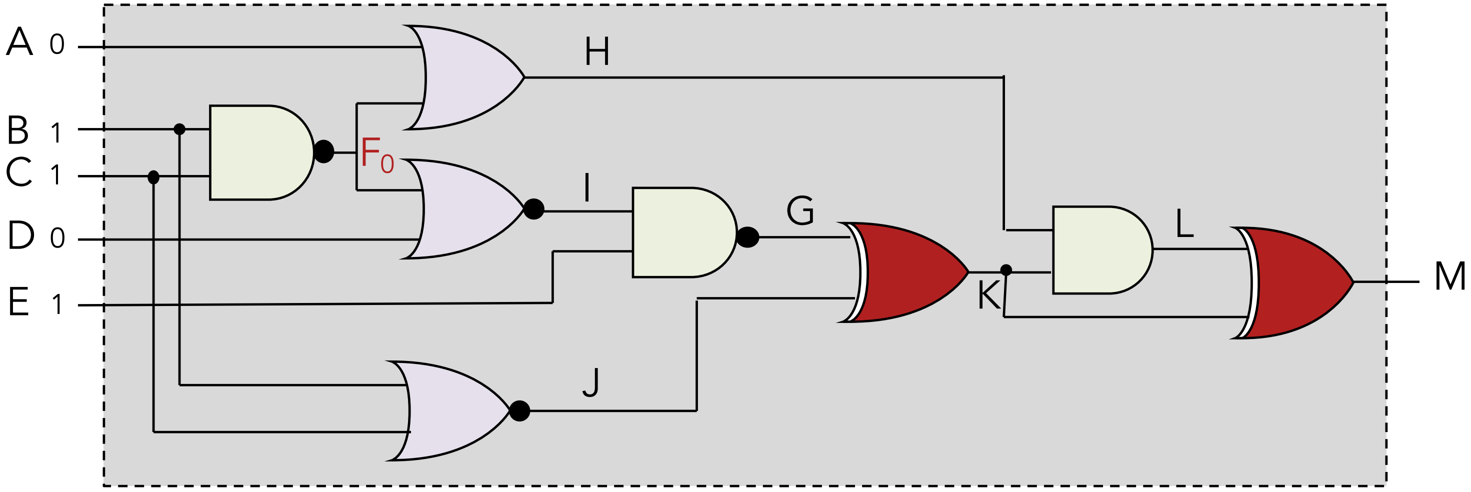
G	J	K
0	0	0
0	1	1
1	0	1
1	1	0

H	K	L
0	0	0
0	1	0
1	0	0
1	1	1

L	K	M
0	0	0
0	1	1
1	0	1
1	1	0

EB_0,C_0 (F_1)	EA_0,F_0 (H_0)	EF_0,D_0 (I_1)	EB_0,C_0 (J_1)	EI_0,E_0 (G_1)	EG_0,J_0 (K_0)	EH_0,K_0 (L_0)	EL_0,K_0 (M_0)
EB_0,C_1 (F_1)	EA_0,F_1 (H_1)	EF_0,D_1 (I_0)	EB_0,C_1 (J_0)	EI_0,E_1 (G_1)	EG_0,J_1 (K_1)	EH_0,K_1 (L_0)	EL_0,K_1 (M_1)
EB_1,C_0 (F_1)	EA_1,F_0 (H_1)	EF_1,D_0 (I_0)	EB_1,C_0 (J_0)	EI_1,E_0 (G_1)	EG_1,J_0 (K_1)	EH_1,K_0 (L_0)	EL_1,K_0 (M_1)
EB_1,C_1 (F_0)	EA_1,F_1 (H_1)	EF_1,D_1 (I_0)	EB_1,C_1 (J_0)	EI_1,E_1 (G_0)	EG_1,J_1 (K_0)	EH_1,K_1 (L_1)	EL_1,K_1 (M_0)

Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

EB_0,C_0 (F_1)
EB_0,C_1 (F_1)
EB_1,C_0 (F_1)
EB_1,C_1 (F_0)

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

EA_0,F_0 (H_0)
EA_0,F_1 (H_1)
EA_1,F_0 (H_1)
EA_1,F_1 (H_1)

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

EF_0,D_0 (I_1)
EF_0,D_1 (I_0)
EF_1,D_0 (I_0)
EF_1,D_1 (I_0)

B	C	J
0	0	1
0	1	0
1	0	0
1	1	0

EB_0,C_0 (J_1)
EB_0,C_1 (J_0)
EB_1,C_0 (J_0)
EB_1,C_1 (J_0)

I	E	G
0	0	1
0	1	1
1	0	1
1	1	0

EI_0,E_0 (G_1)
EI_0,E_1 (G_1)
EI_1,E_0 (G_1)
EI_1,E_1 (G_0)

G	J	K
0	0	0
0	1	1
1	0	1
1	1	0

EG_0,J_0 (K_0)
EG_0,J_1 (K_1)
EG_1,J_0 (K_1)
EG_1,J_1 (K_0)

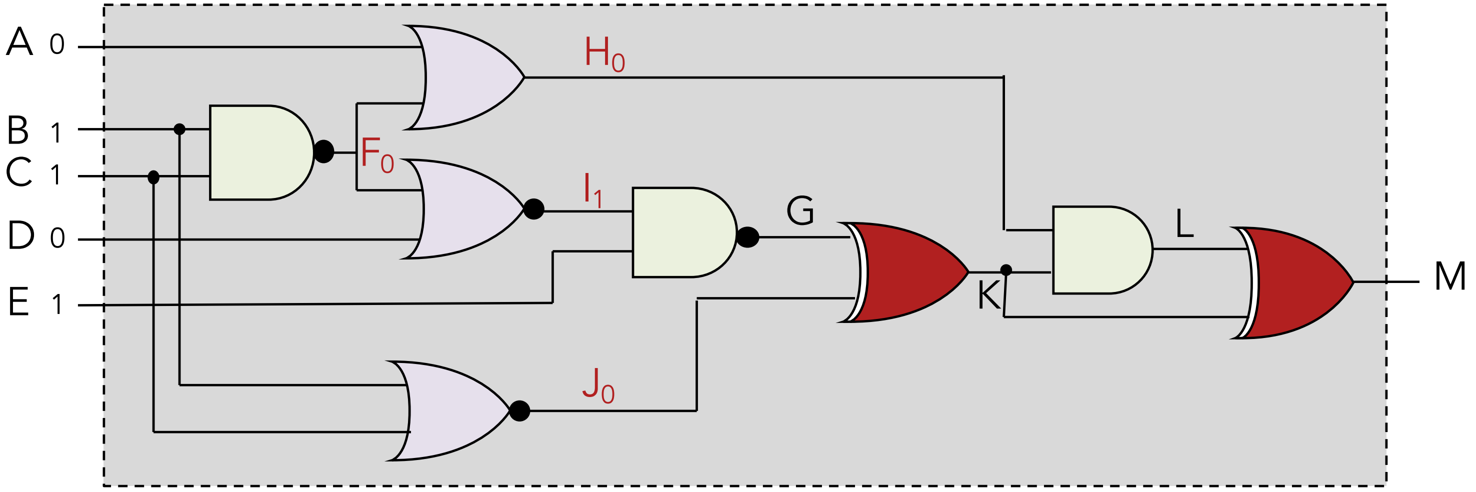
H	K	L
0	0	0
0	1	0
1	0	0
1	1	1

EH_0,K_0 (L_0)
EH_0,K_1 (L_0)
EH_1,K_0 (L_0)
EH_1,K_1 (L_1)

L	K	M
0	0	0
0	1	1
1	0	1
1	1	0

EL_0,K_0 (M_0)
EL_0,K_1 (M_1)
EL_1,K_0 (M_1)
EL_1,K_1 (M_0)

Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

$E_{B_0,C_0}(F_1)$
$E_{B_0,C_1}(F_1)$
$E_{B_1,C_0}(F_1)$
$E_{B_1,C_1}(F_0)$

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

$E_{A_0,F_0}(H_0)$
$E_{A_0,F_1}(H_1)$
$E_{A_1,F_0}(H_1)$
$E_{A_1,F_1}(H_1)$

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

$E_{F_0,D_0}(I_1)$
$E_{F_0,D_1}(I_0)$
$E_{F_1,D_0}(I_0)$
$E_{F_1,D_1}(I_0)$

B	C	J
0	0	1
0	1	0
1	0	0
1	1	0

$E_{B_0,C_0}(J_1)$
$E_{B_0,C_1}(J_0)$
$E_{B_1,C_0}(J_0)$
$E_{B_1,C_1}(J_0)$

I	E	G
0	0	1
0	1	1
1	0	1
1	1	0

$E_{I_0,E_0}(G_1)$
$E_{I_0,E_1}(G_1)$
$E_{I_1,E_0}(G_1)$
$E_{I_1,E_1}(G_0)$

G	J	K
0	0	0
0	1	1
1	0	1
1	1	0

$E_{G_0,J_0}(K_0)$
$E_{G_0,J_1}(K_1)$
$E_{G_1,J_0}(K_1)$
$E_{G_1,J_1}(K_0)$

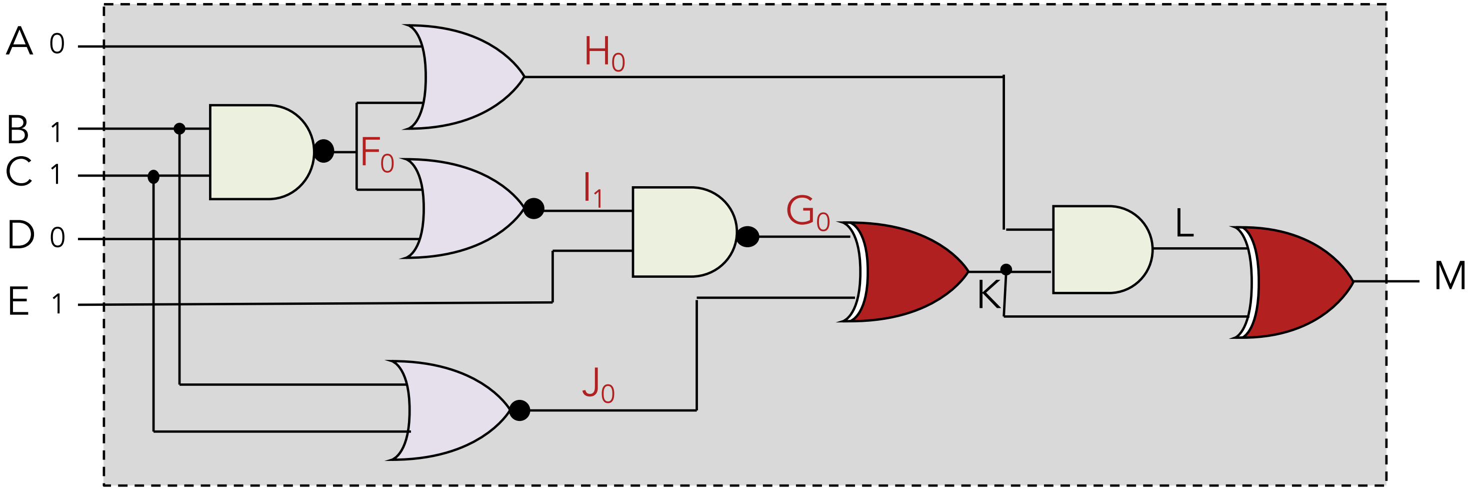
H	K	L
0	0	0
0	1	0
1	0	0
1	1	1

$E_{H_0,K_0}(L_0)$
$E_{H_0,K_1}(L_0)$
$E_{H_1,K_0}(L_0)$
$E_{H_1,K_1}(L_1)$

L	K	M
0	0	0
0	1	1
1	0	1
1	1	0

$E_{L_0,K_0}(M_0)$
$E_{L_0,K_1}(M_1)$
$E_{L_1,K_0}(M_1)$
$E_{L_1,K_1}(M_0)$

Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

$EB_0,C_0\ (F_1)$
$EB_0,C_1\ (F_1)$
$EB_1,C_0\ (F_1)$
$EB_1,C_1\ (F_0)$

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

$EA_0,F_0\ (H_0)$
$EA_0,F_1\ (H_1)$
$EA_1,F_0\ (H_1)$
$EA_1,F_1\ (H_1)$

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

$EF_0,D_0\ (I_1)$
$EF_0,D_1\ (I_0)$
$EF_1,D_0\ (I_0)$
$EF_1,D_1\ (I_0)$

B	C	J
0	0	1
0	1	0
1	0	0
1	1	0

$EB_0,C_0\ (J_1)$
$EB_0,C_1\ (J_0)$
$EB_1,C_0\ (J_0)$
$EB_1,C_1\ (J_0)$

I	E	G
0	0	1
0	1	1
1	0	1
1	1	0

$EI_0,E_0\ (G_1)$
$EI_0,E_1\ (G_1)$
$EI_1,E_0\ (G_1)$
$EI_1,E_1\ (G_0)$

G	J	K
0	0	0
0	1	1
1	0	1
1	1	0

$EG_0,J_0\ (K_0)$
$EG_0,J_1\ (K_1)$
$EG_1,J_0\ (K_1)$
$EG_1,J_1\ (K_0)$

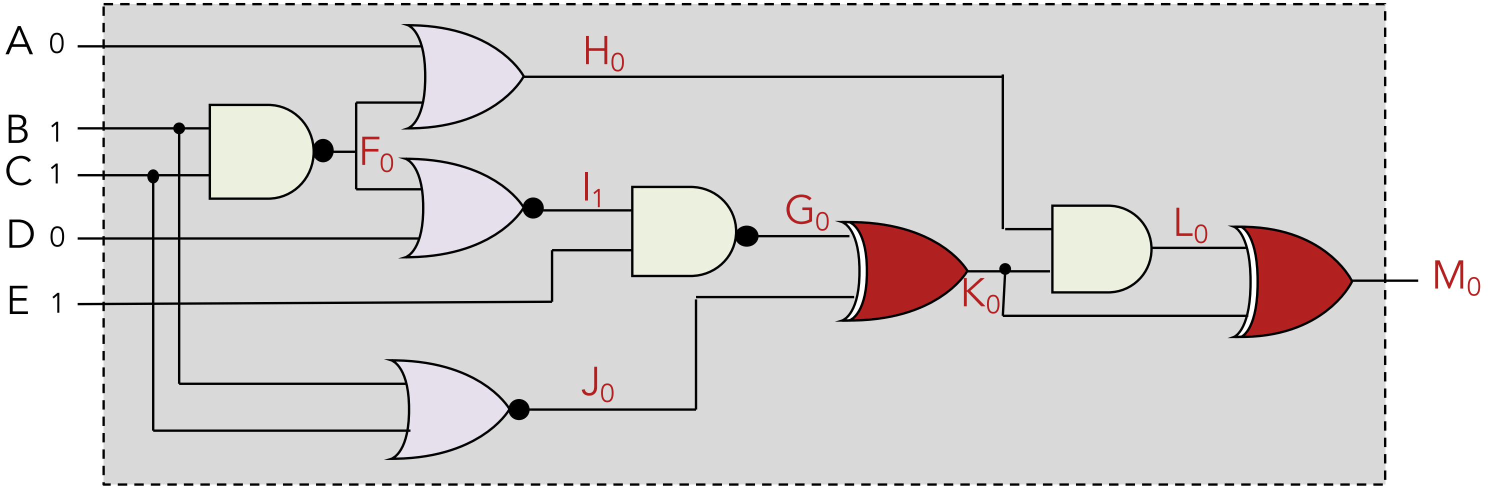
H	K	L
0	0	0
0	1	0
1	0	0
1	1	1

$EH_0,K_0\ (L_0)$
$EH_0,K_1\ (L_0)$
$EH_1,K_0\ (L_0)$
$EH_1,K_1\ (L_1)$

L	K	M
0	0	0
0	1	1
1	0	1
1	1	0

$EL_0,K_0\ (M_0)$
$EL_0,K_1\ (M_1)$
$EL_1,K_0\ (M_1)$
$EL_1,K_1\ (M_0)$

Garbled Circuit



B	C	F
0	0	1
0	1	1
1	0	1
1	1	0

E_{B_0,C_0} (F_1)
E_{B_0,C_1} (F_1)
E_{B_1,C_0} (F_1)
E_{B_1,C_1} (F_0)

A	F	H
0	0	0
0	1	1
1	0	1
1	1	1

E_{A_0,F_0} (H_0)
E_{A_0,F_1} (H_1)
E_{A_1,F_0} (H_1)
E_{A_1,F_1} (H_1)

F	D	I
0	0	1
0	1	0
1	0	0
1	1	0

E_{F_0,D_0} (I_1)
E_{F_0,D_1} (I_0)
E_{F_1,D_0} (I_0)
E_{F_1,D_1} (I_0)

B	C	J
0	0	1
0	1	0
1	0	0
1	1	0

E_{B_0,C_0} (J_1)
E_{B_0,C_1} (J_0)
E_{B_1,C_0} (J_0)
E_{B_1,C_1} (J_0)

I	E	G
0	0	1
0	1	1
1	0	1
1	1	0

E_{I_0,E_0} (G_1)
E_{I_0,E_1} (G_1)
E_{I_1,E_0} (G_1)
E_{I_1,E_1} (G_0)

G	J	K
0	0	0
0	1	1
1	0	1
1	1	0

E_{G_0,J_0} (K_0)
E_{G_0,J_1} (K_1)
E_{G_1,J_0} (K_1)
E_{G_1,J_1} (K_0)

H	K	L
0	0	0
0	1	0
1	0	0
1	1	1

E_{H_0,K_0} (L_0)
E_{H_0,K_1} (L_0)
E_{H_1,K_0} (L_0)
E_{H_1,K_1} (L_1)

L	K	M
0	0	0
0	1	1
1	0	1
1	1	0

E_{L_0,K_0} (M_0)
E_{L_0,K_1} (M_1)
E_{L_1,K_0} (M_1)
E_{L_1,K_1} (M_0)

Secure Computation Approaches

Multi-Party Computation (MPC)

Pros
- Low compute requirements
- Easy to accelerate
- Provably secure
- Supports multiple threat models
- Easy to map existing algorithms

Cons
- High communication costs
- High latency
- Information theoretic proofs are weaker than PKE ones

Fully Homomorphic Encryption (FHE)

Pros
- Very low communication costs
- Requires a single round of communications, i.e., "fire and forget"
- Useful when one side is limited in compute / memory / storage
- Provably secure – relies on strength of PKE

Cons
- Very high computational requirements
- Harder to accelerate
- Mapping existing algorithms to FHE may be difficult

Trusted Execution Environments (TEE)

Pros
- No communication required
- Trivial to accelerate
- Great support for existing software

Cons
- Weaker security guarantees
- Cannot stop determined adversaries
- Historically plagued by vulnerabilities and breaches
- Long term deployment is difficult – TEE's can 'run out' of entropy / CRP's, etc.

Secure Computation Approaches

Multi-Party Computation (MPC)

Pros

- Low compute requirements
- Easy to accelerate
- Provably secure
- Supports multiple threat models
- Easy to map existing algorithms

Cons

- High communication costs
- High latency
- Information theoretic proofs are weaker than PKE ones

Fully Homomorphic Encryption (FHE)

Pros

- Very low communication costs
- Requires a single round of communications, i.e., "fire and forget"
- Useful when one side is limited in compute / memory / storage
- Provably secure – relies on strength of PKE

Cons

- Very high computational requirements
- Harder to accelerate
- Mapping existing algorithms to FHE may be difficult

Trusted Execution Environments (TEE)

Pros

- No communication required
- Trivial to accelerate
- Great support for existing software

Cons

- Weaker security guarantees
- Cannot stop determined adversaries
- Historically plagued by vulnerabilities and breaches
- Long term deployment is difficult – TEE's can 'run out' of entropy / CRP's, etc.

Secure Multiparty Computation

- For the Two-party secure multiparty computation
- Assume
  - Alice has x, Bob has y, and they want to compute two functions $f_A(x,y)$ and $f_B(x,y)$
    - It could be the same function f(x,y)
  - The desired outcome is that at the end of the protocol
    - Alice learns the result of her function $f_A(x,y)$ and not Bob's input y
    - Bob learns the result of his function $f_B(x,y)$ and not Alice's input x

Secure Multiparty Computation

- For the Two-party secure multiparty computation
- Assume
  - Alice has x, Bob has y, and they want to compute two functions $f_A(x,y)$ and $f_B(x,y)$
    - It could be the same function f(x,y)
- Illustration
  - Alice represents the function f(x,y) as a garbled circuit
  - She then sends the circuit and values corresponding to her input bits to Bob
  - Bob evaluates the circuits using the sent Alice's bits and his own input bits
  - He then transfers the result to Alice

Secure Multiparty Computation

- For the Two-party secure multiparty computation
- Assume
  - Alice has x, Bob has y, and they want to compute two functions $f_A(x,y)$ and $f_B(x,y)$
    - It could be the same function
- The set up for the n-party secure multiparty computation makes the same assumptions
  - Here instead of just Alice and Bob, there are n parties
  - Each party with a private input
  - And they want to jointly compute the function

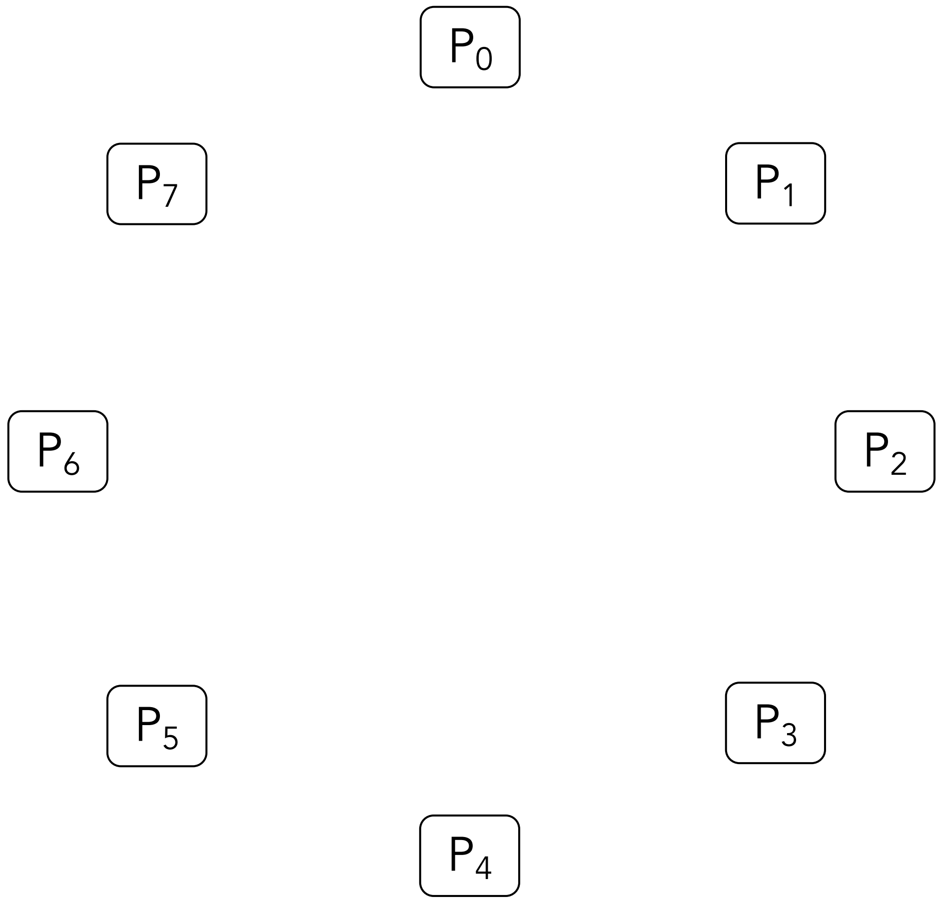
$$f_{X_i}=(x_1, \ldots, x_n)$$

Secure Multiparty Computation

- Validity
  - Secure function evaluation (SFE) system must be able to correctly computed
    - For example, result must be computed with inputs from at least all correct parties
- Privacy
  - P_1 and P_2 cannot know each others input ip_1, ip_2
- Agreement
  - Result must be same for all parties (P_1 and P_2)
- Termination
  - All active parties (P_1 and P_2) eventually receive final result
- Fairness
  - P_1 should not be able to learn the result while denying it to P_2

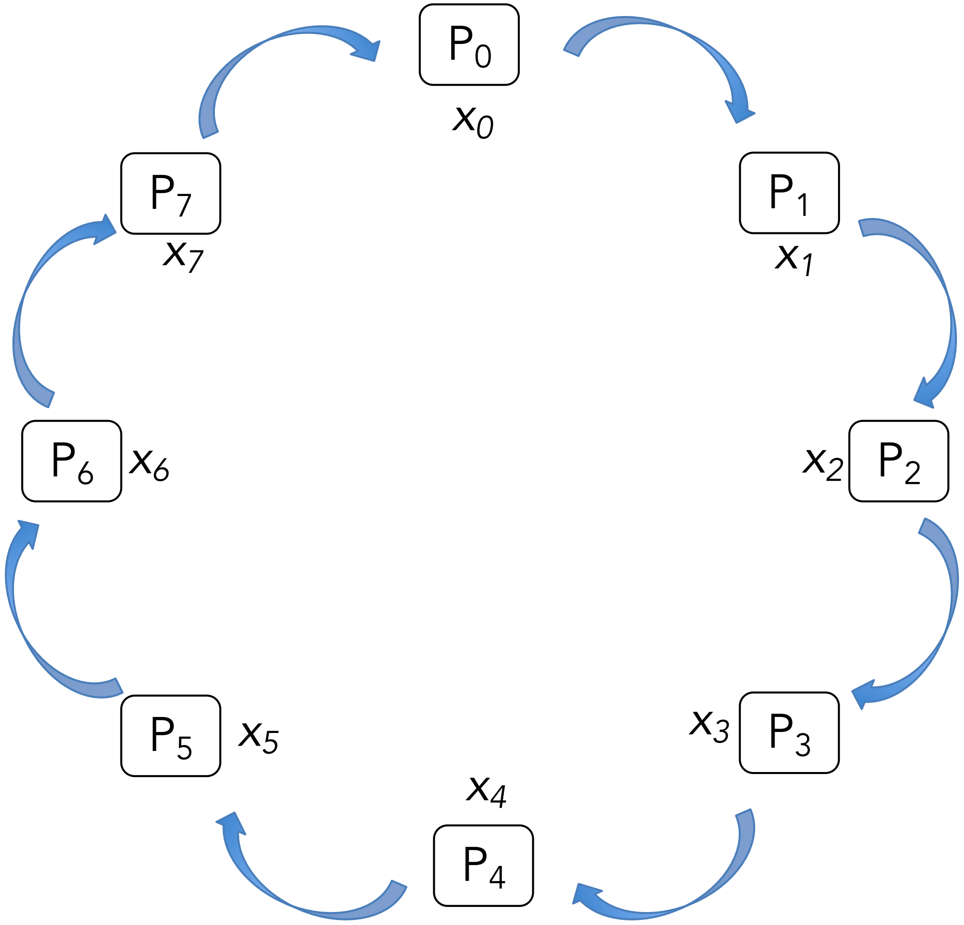
Secure Multiparty Computation

- Construction of the computation
  - Let us have 8 parties $P_1, \ldots, P_7$ that want to perform a joint computation



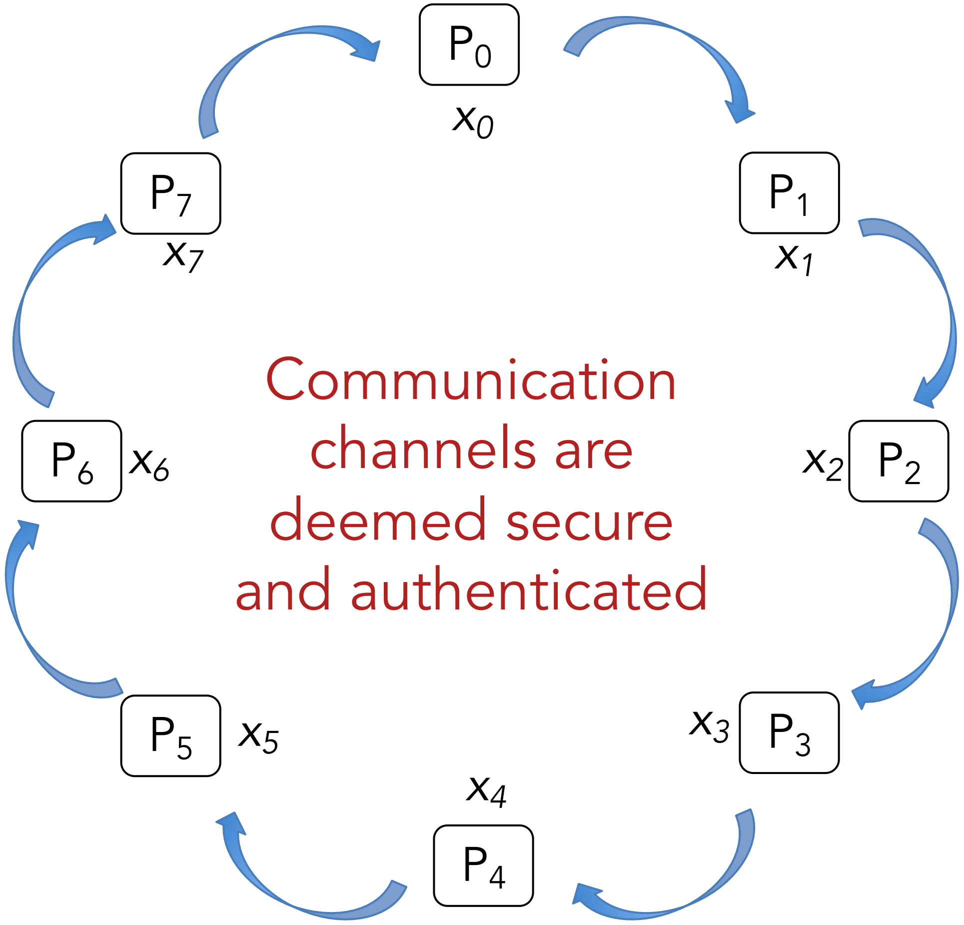
Secure Multiparty Computation

- Construction of the computation
  - Let us have 8 parties $P_0, ..., P_7$ that want to perform a joint computation
  - Each party P_i with $i \in [0..7]$, has private input x_i



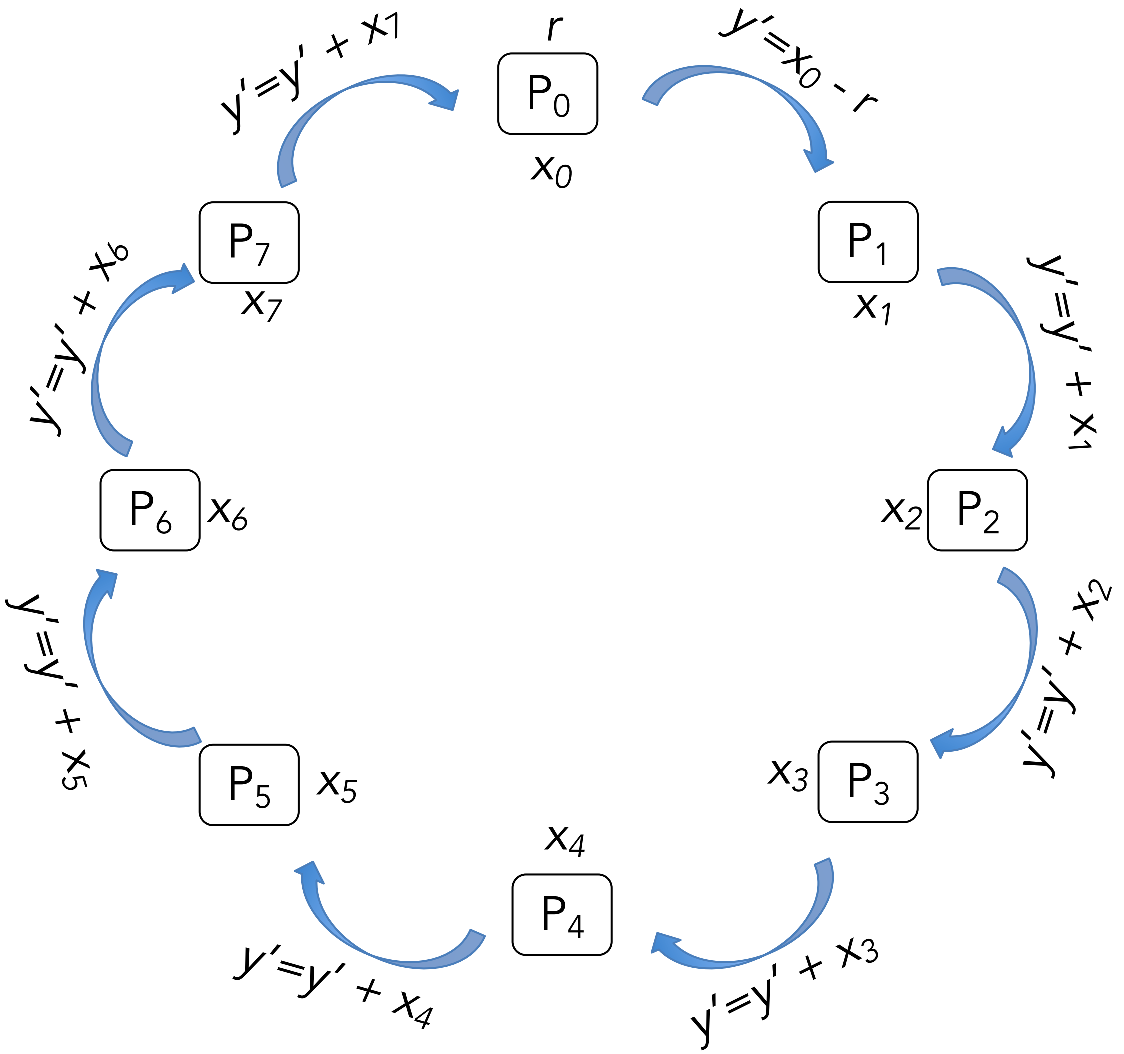
Secure Multiparty Computation

- Construction of the computation
  - Let us have 8 parties $P_0, . . . , P_7$ that want to perform a joint computation
  - Each party P_i with $i \in [0..7]$, has private input x_i



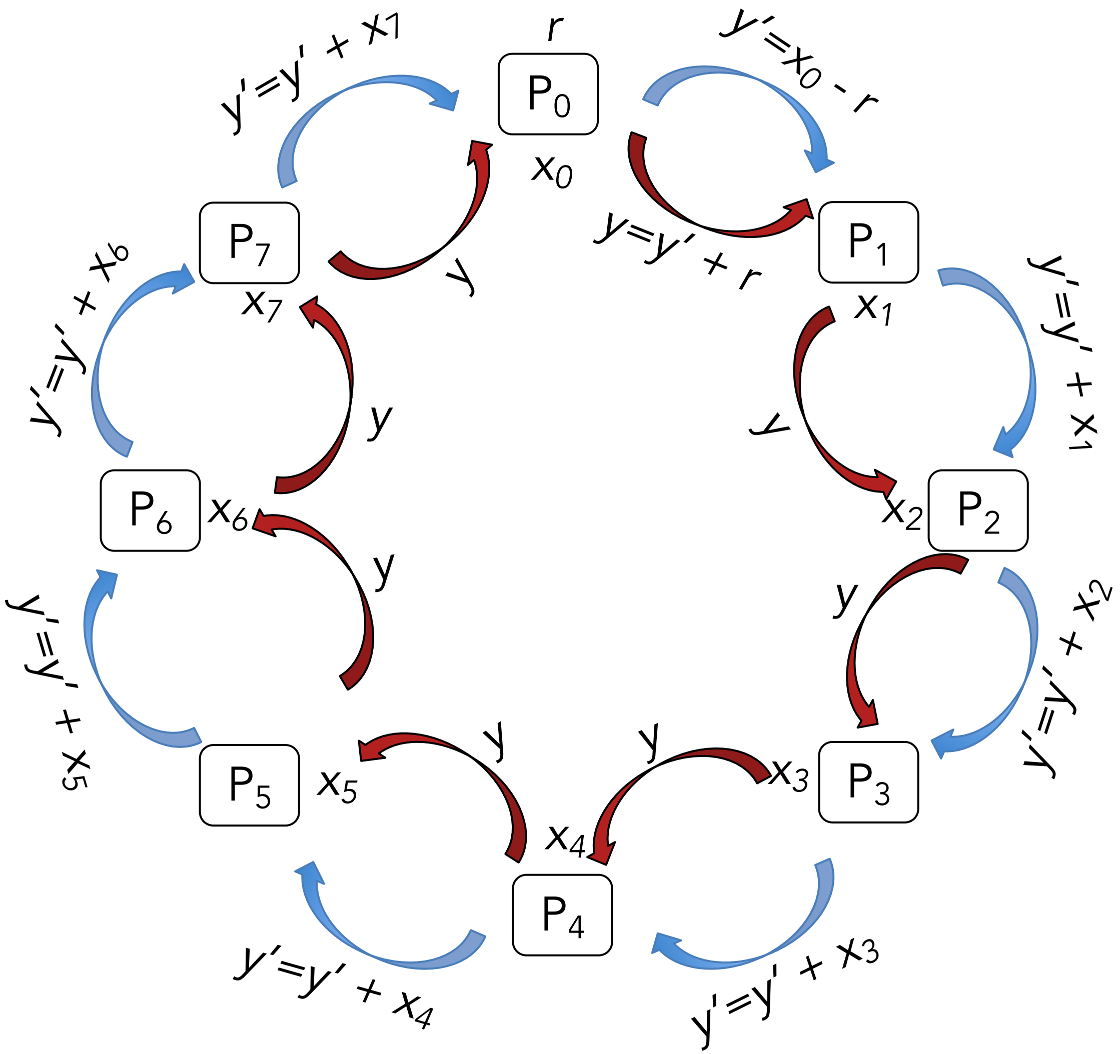
Secure Multiparty Computation

- Construction of the computation
  - r is a random number



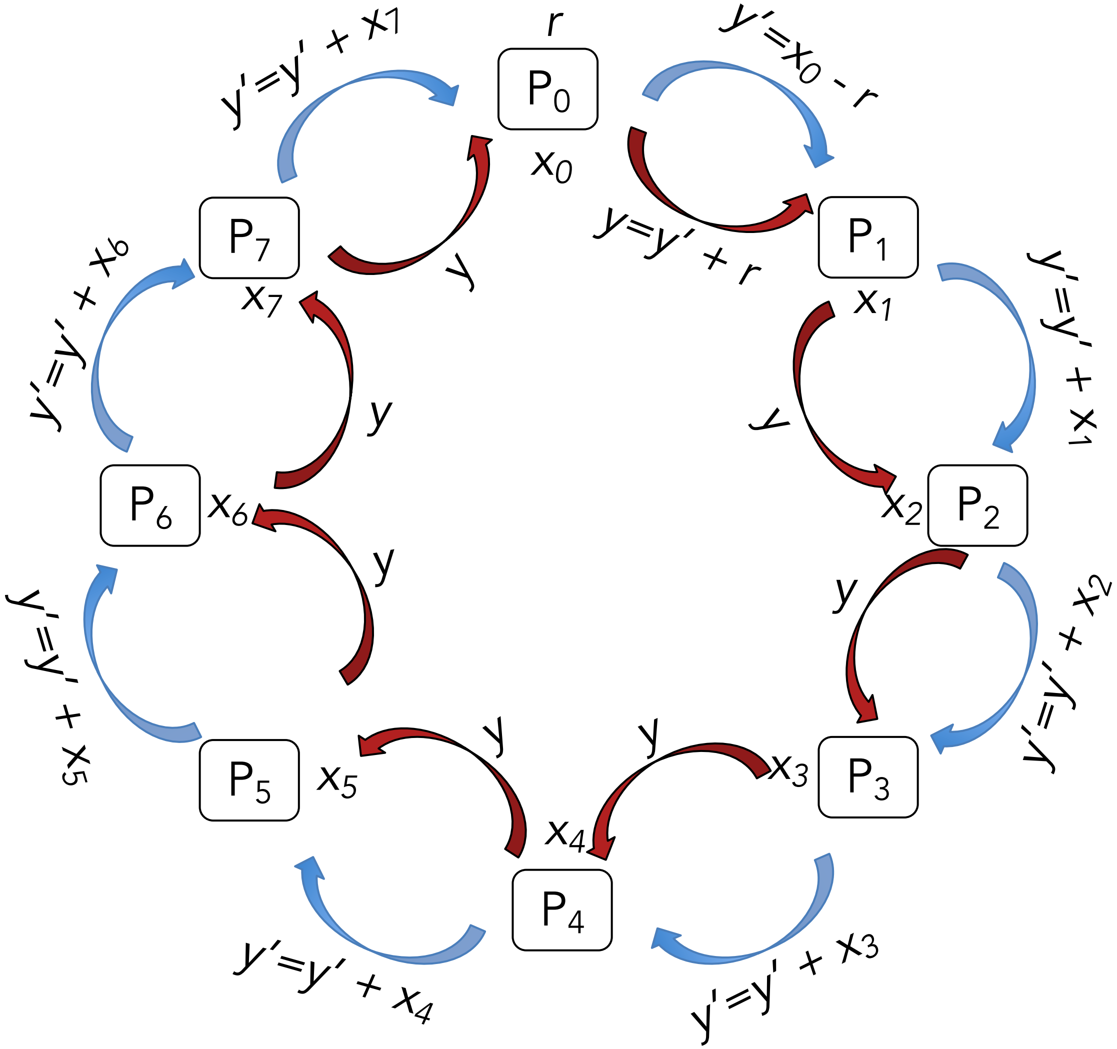
Secure Multiparty Computation

- Construction of the computation
  - r is a random number



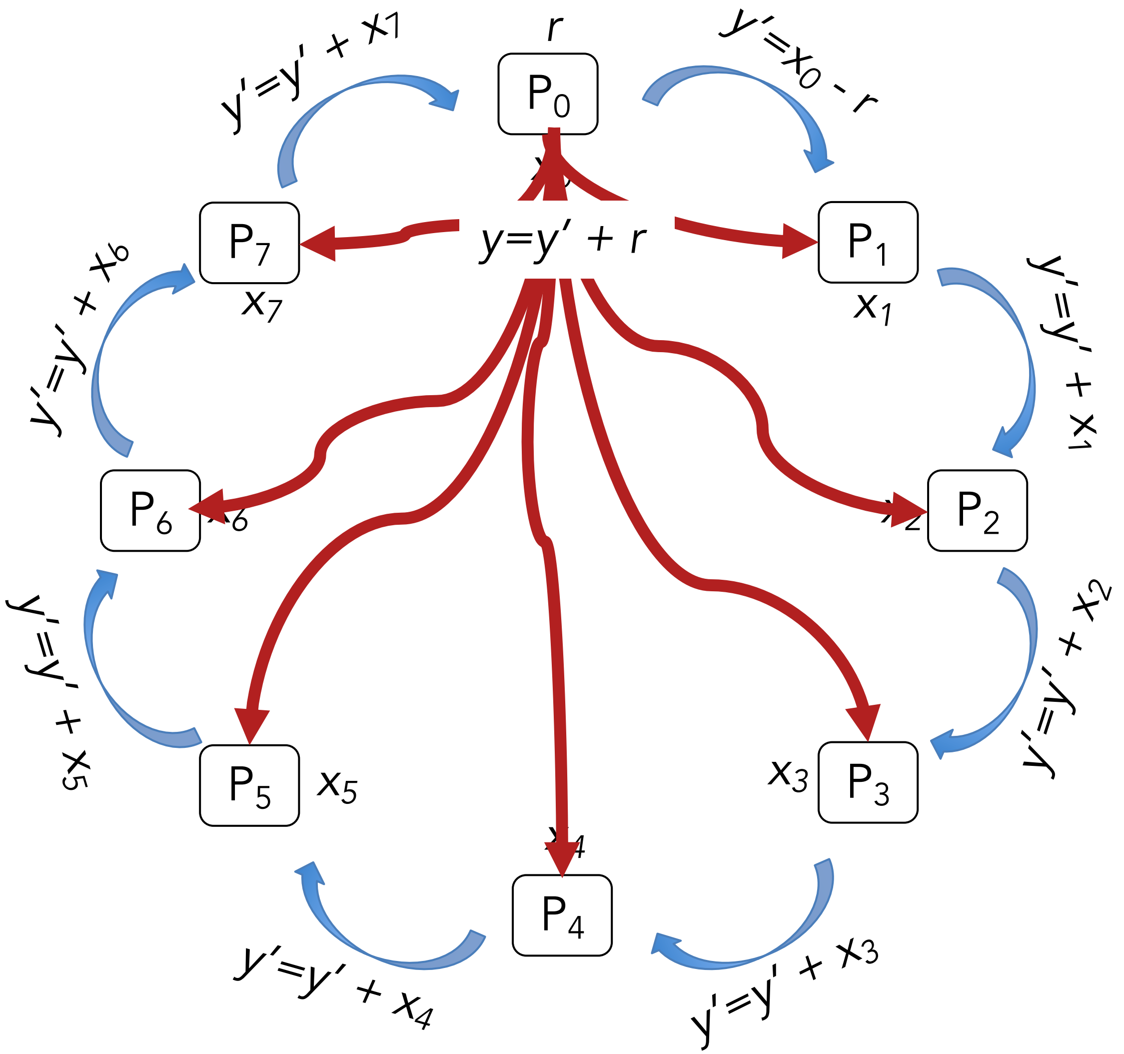
Secure Multiparty Computation

- Construction of the computation
  - r is a random number
  - If any P_i is semi-honest or malicious, then these messages may not be passed along properly or be modified in a way that break the protocol



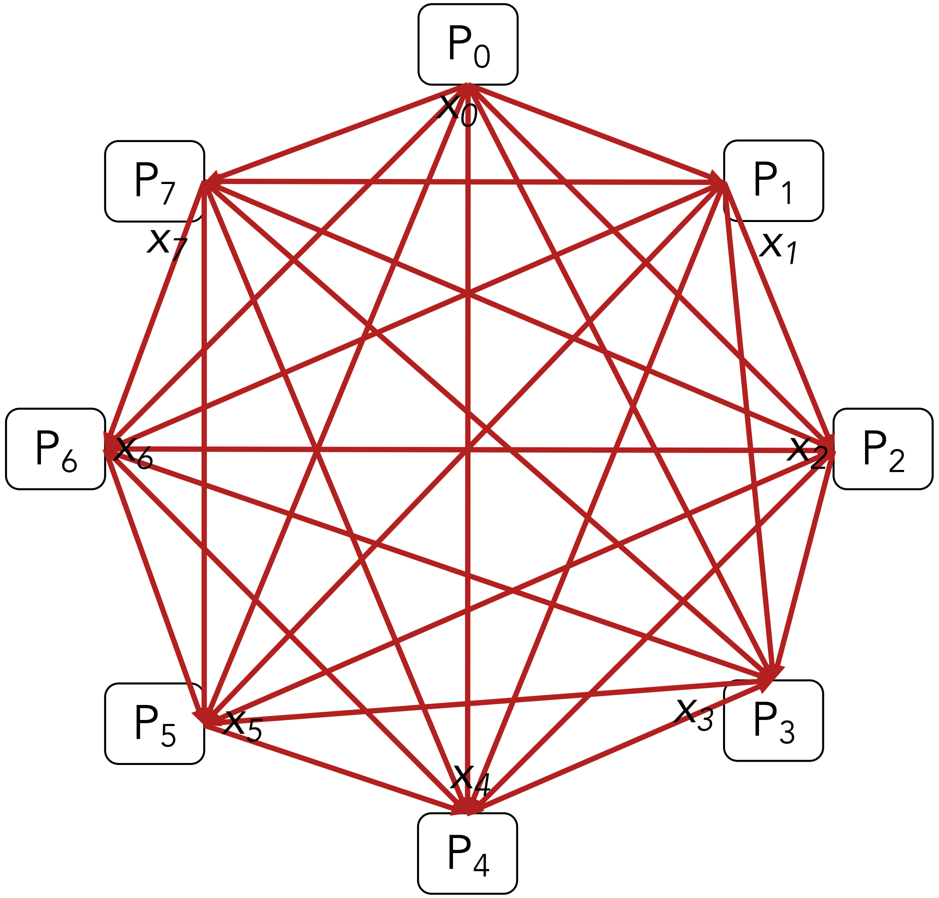
Secure Multiparty Computation

- Construction of the computation
  - Result distribution could be faster



Secure Multiparty Computation

- Construction of the computation
  - Even fast compute



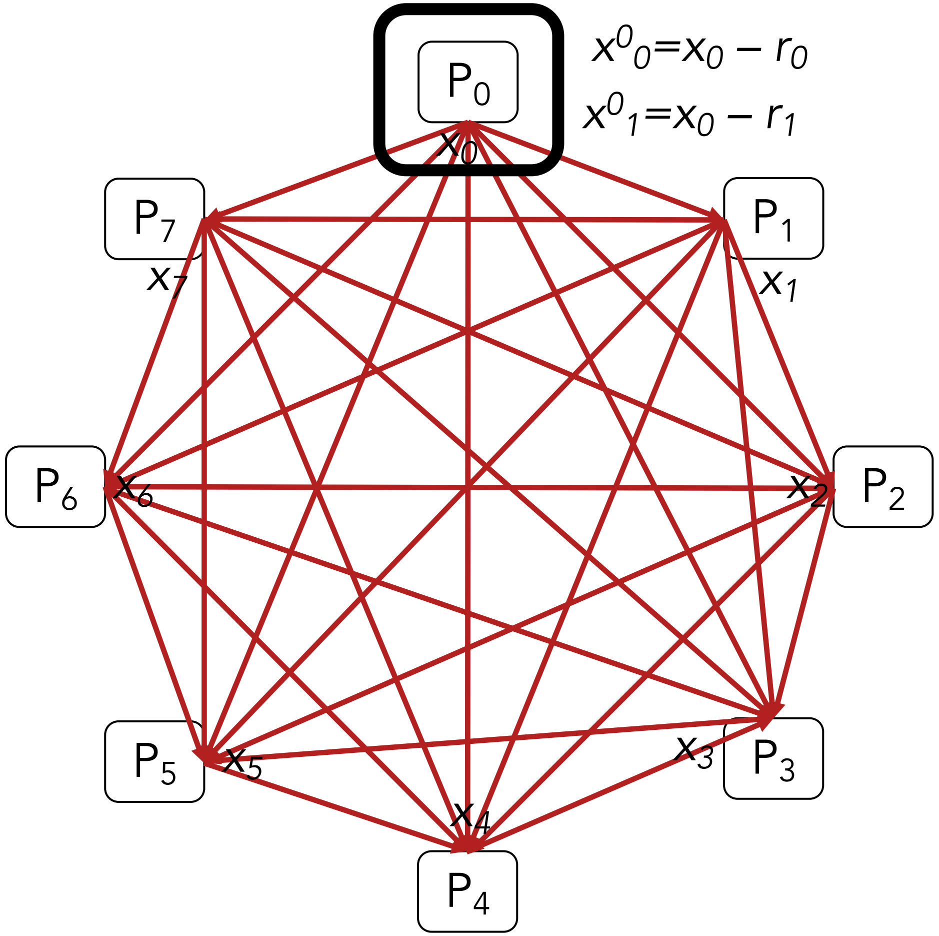
Secure Multiparty Computation

- Construction of the computation
  - *The parties can use a linear secret sharing scheme to create a distributed state of their inputs*
  - *For each party, the random variables r_i are different*

$x^0_2 = x_0 - r_2$ $x^0_3 = x_0 - r_3$

$x^0_4 = x_0 - r_4$ $x^0_5 = x_0 - r_5$

$x^0_6 = x_0 - r_6$ $x^0_7 = x_0 - r_7$



Secure Multiparty Computation

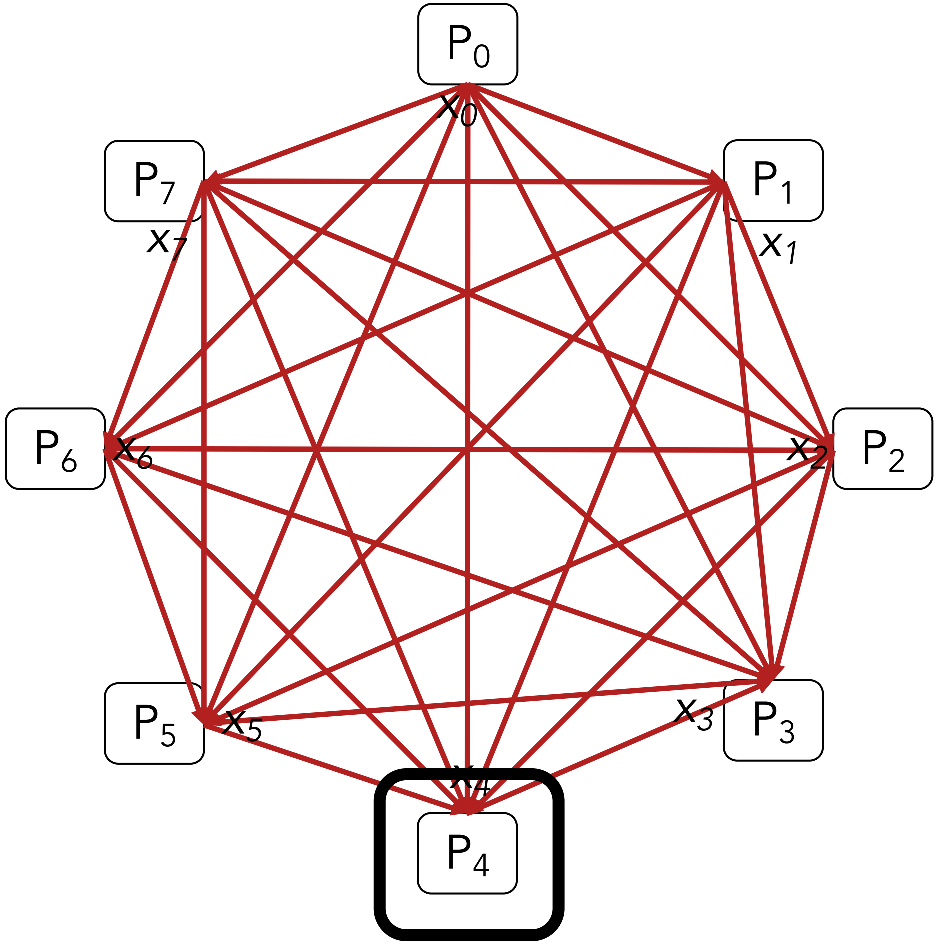
- Construction of the computation
  - *The parties can use a linear secret sharing scheme to create a distributed state of their inputs*
  - *For each party, the random variables r_i are different*

$x^4_0 = x_4 - r_0$ $x^4_3 = x_4 - r_3$

$x^4_1 = x_4 - r_1$ $x^4_4 = x_4 - r_4$

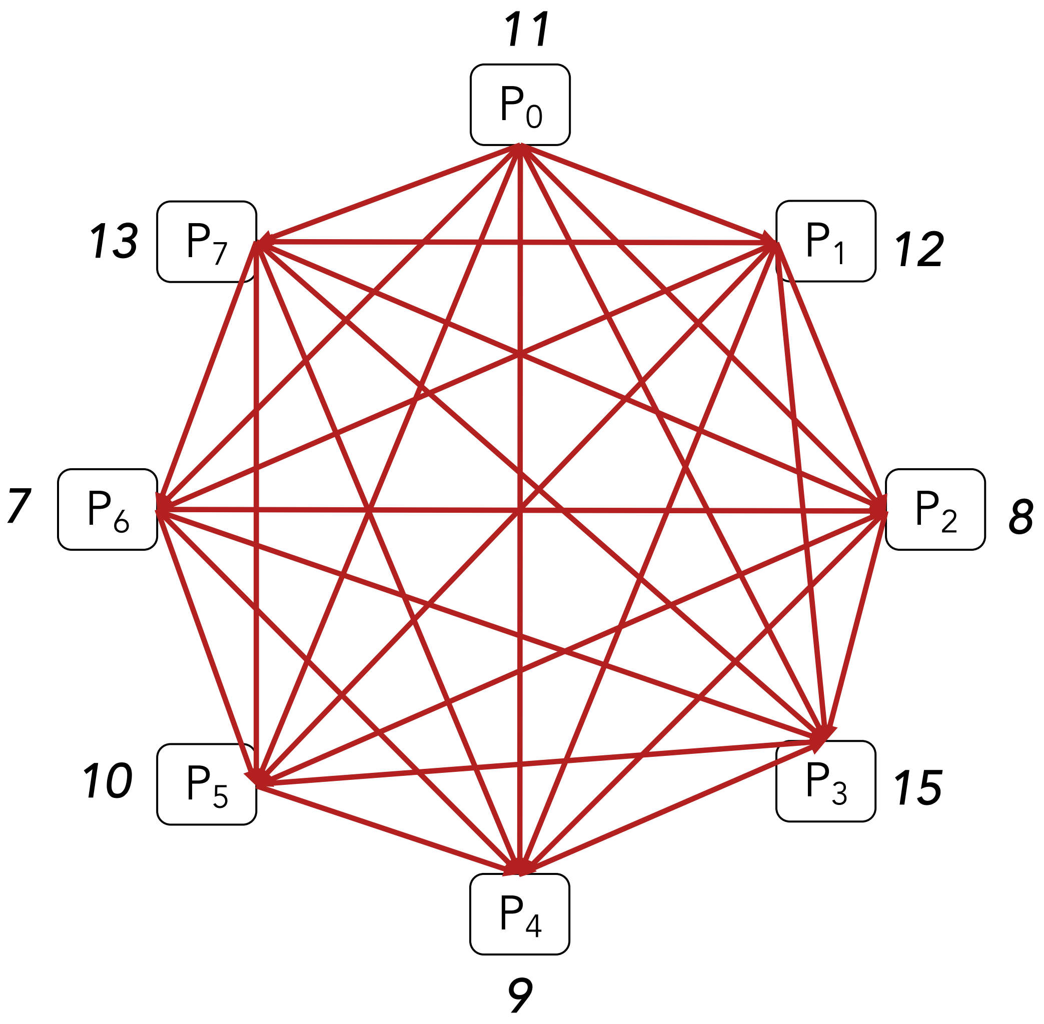
$x^4_2 = x_4 - r_2$ $x^4_5 = x_4 - r_5$

$x^4_6 = x_4 - r_6$ $x^4_7 = x_4 - r_7$



Secure Multiparty Computation

- Construction of the computation
  - Let us have 8 parties P_1, . . . , P_7 that want to perform a joint computation
  - Let us do summation



Secure Multiparty Computation

Private Inputs		P_0	P_1	P_2	P_3	P_4	P_5	P_6	
11	P_0								
12	P_1								
8	P_2								
15	P_3								
9	P_4								
10	P_5								
7	P_6								
13	P_7								

Local Total

Secure Multiparty Computation

Private Inputs		P_0	P_1	P_2	P_3	P_4	P_5	P_6	
11	P_0	-1	1	4	3	1	0	3	0
12	P_1								
8	P_2								
15	P_3								
9	P_4								
10	P_5								
7	P_6								
13	P_7								

				Local Total					

Secure Multiparty Computation

Private Inputs		P_0	P_1	P_2	P_3	P_4	P_5	P_6	
11	P_0	-1	1	4	3	1	0	3	0
12	P_1	3	-5	1	2	4	0	2	5
8	P_2	1	0	0	0	2	3	1	1
15	P_3	4	3	1	-4	3	2	2	4
9	P_4	1	1	3	0	2	0	1	1
10	P_5	2	4	0	1	2	-2	3	0
7	P_6	1	0	5	2	0	1	-5	3
13	P_7	1	2	3	2	1	1	3	0

Local Total

Secure Multiparty Computation

Private Inputs		P_0	P_1	P_2	P_3	P_4	P_5	P_6	
11	P_0	-1	1	4	3	1	0	3	0
12	P_1	3	-5	1	2	4	0	2	5
8	P_2	1	0	0	0	2	3	1	1
15	P_3	4	3	1	-4	3	2	2	4
9	P_4	1	1	3	0	2	0	1	1
10	P_5	2	4	0	1	2	-2	3	0
7	P_6	1	0	5	2	0	1	-5	3
13	P_7	1	2	3	2	1	1	3	0
85		**12**	**5**	**17**	**6**	**15**	**5**	**10**	**14**

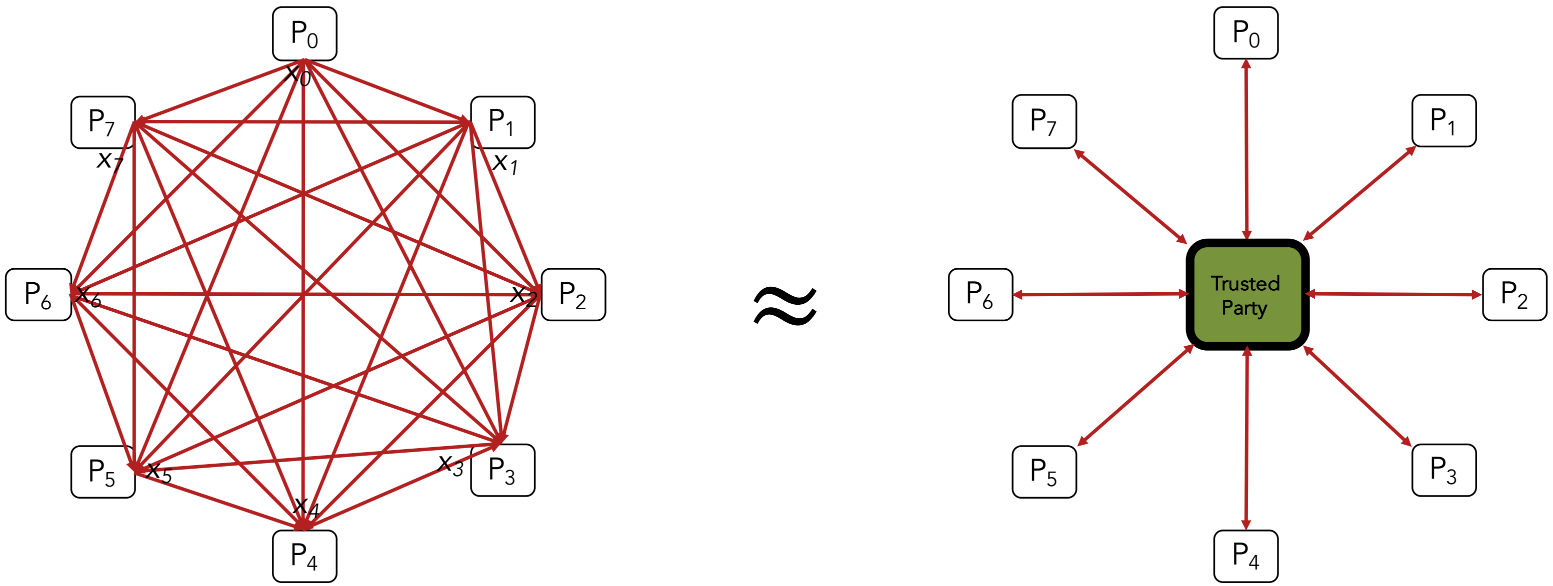
Local Total

Secure Multiparty Computation

- There are two major adversary models for secure computation
  - Semi-honest/passive model
    - Follows all required steps
    - Looks for all advantageous information leaked
    - Assumed to be selfish
  - Fully malicious/active model
    - Arbitrarily deviates from the protocol
    - Aborts the protocol at anytime
    - Takes any step that runs counter to the desirable outcome

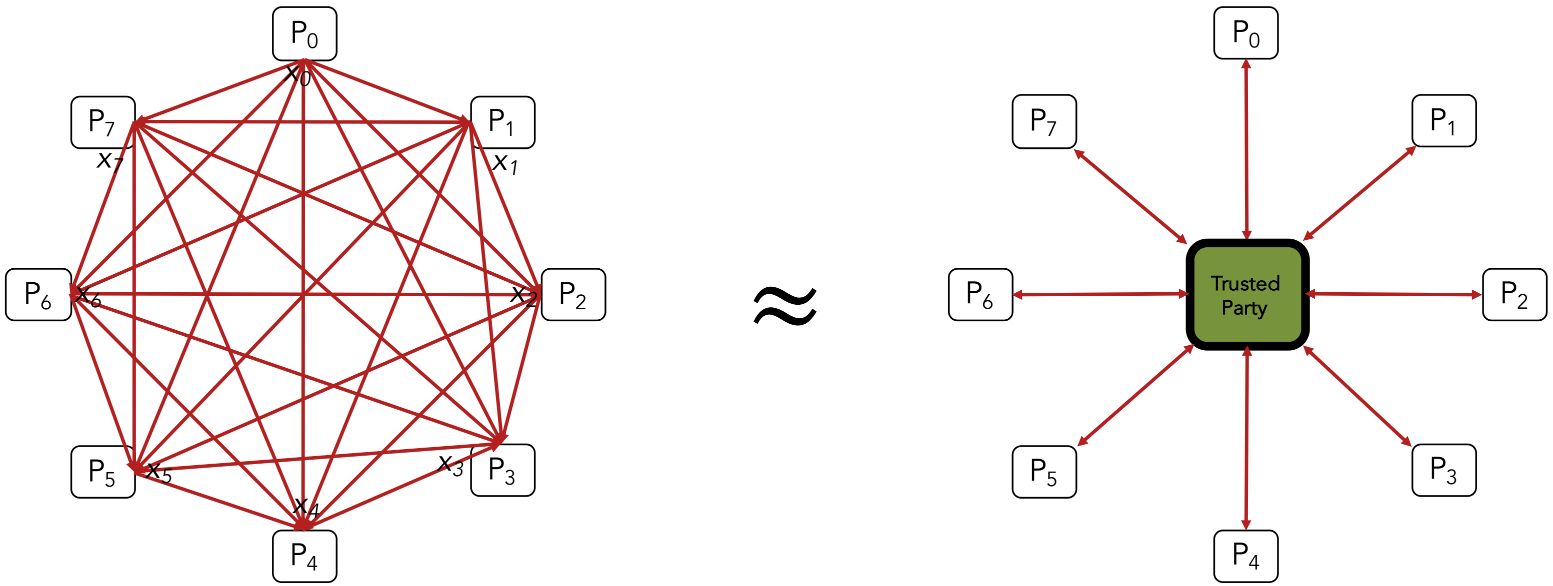
Secure Multiparty Computation

- The multiparty computation is secure if it emulates the trusted central party model to a negligible error range
  - If the two are shown to be indistinguishable
  - Trusted party/Ideal/Simulated model



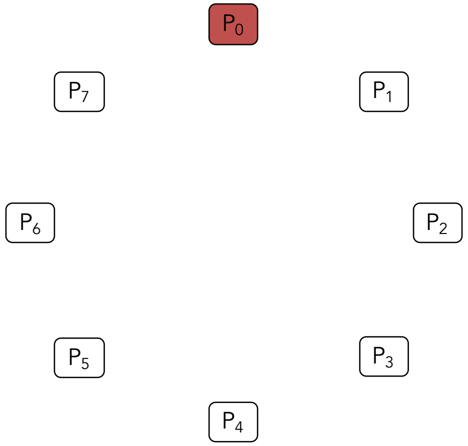
Secure Multiparty Computation

- The security multiparty computation protocol is also evaluated though the simulated model
  - For example, the assumption that parties communicate through secure and authenticated channels holds for both settings

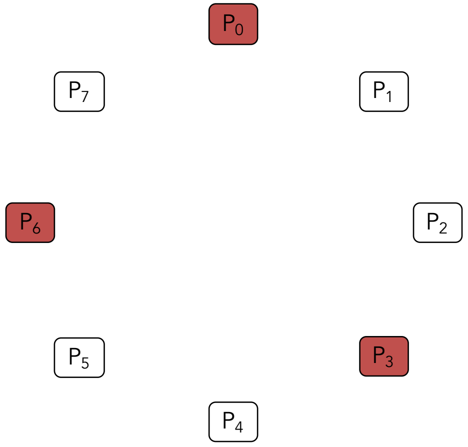


Secure Multiparty Computation

- Dealing with semi-honest and malicious



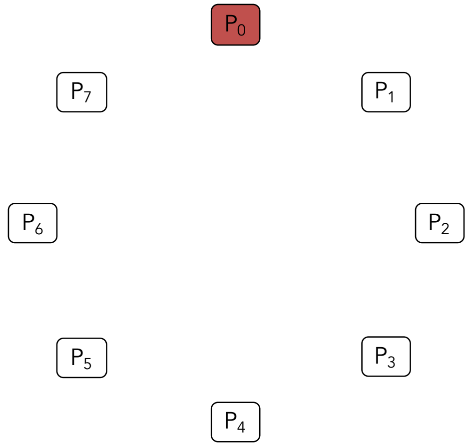
D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC '88)

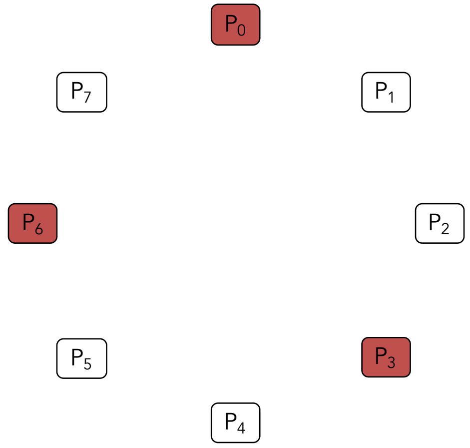


M. Ben-Or, S. Goldwasser, and A. Wigderson Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC '88)

Secure Multiparty Computation

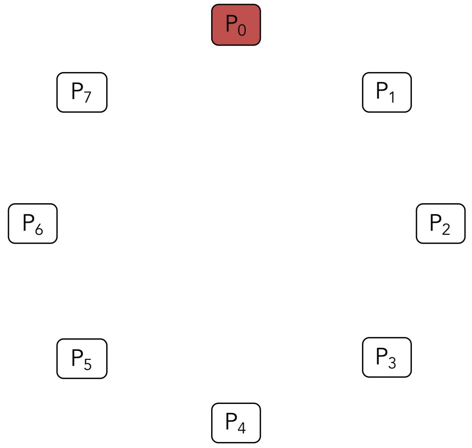
- Dealing with semi-honest and malicious
  - Any function $f(x_1, \ldots, x_n)$ can be securely computed in a semi-honest setting if the majority is honest
    - The passive adversary controls less than n/2 of the parties
  - Any function $f(x_1, \ldots, x_n)$ can be securely computed if the adversary actively controls less than n/3 of the parties

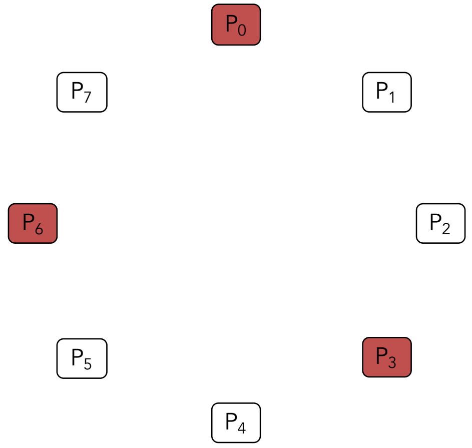




Secure Multiparty Computation

- It is a rich area of research
  - Secure multiparty computation over groups, fields, rings
  - Authentication of the communication channels
  - Synchronous versus asynchronous messaging
  - And many more sub-topics





Secure Multiparty Computation

- Commitment
  - Let p and q be two large prime numbers such that q divides p-1
  - Generator g of the order-q subgroup of Zp*
  - A secret s from Zp such that $y=g^s \bmod p$
  - Where the values p,q,g, and y are public
  - There is only one secret s in the system residing with Bob

Alice

Alice commits to some x∈Zq
Then selects a random r ∈Zq

{M}

$M = g^x y^r \bmod p$

Bob

Bob now has M

Secure Multiparty Computation

- Commitment
  - Let p and q be two large prime numbers such that q divides p-1
  - Generator g of the order-q subgroup of Zp*
  - A secret s from Zp such that $y=g^s \bmod p$
  - Where the values p,q,g, and y are public
  - There is only one secret s in the system residing with Bob

Alice

Alice commits to some x∈Zq
Then selects a random r ∈Zq

→ {M} →
$M = g^x y^r \bmod p$

Bob

Bob now has M

Alice reveals x and r

→ {x, r} →

Bob can verify that
$M= g^x(g^s)^r=g^{x+sr} \bmod p$

Secure Multiparty Computation

- Zero-Knowledge
  - Let p and q be two large prime numbers such that q divides p-1
  - Generator g of the order-q subgroup of Zp*

Alice

Alice knows a number s such that **$M = g^s$ mod p** and wants to prove it to Bob

{U= g^r mod p }

r is random number ∈ [1..q]

Bob

Bob also know M

Secure Multiparty Computation

- Zero-Knowledge
  - Let p and q be two large prime numbers such that q divides p-1
  - Generator g of the order-q subgroup of Zp*

Alice

Alice knows a number s such that $M = g^s \bmod p$ and wants to prove it to Bob

Alice → Bob: $\{U = g^r \bmod p\}$

r is random number ∈ [1..q]

Bob

Bob also know M

Bob → Alice: {a}

a is random number ∈ [1..q]

Secure Multiparty Computation

- Zero-Knowledge
  - Let p and q be two large prime numbers such that q divides p-1
  - Generator g of the order-q subgroup of Zp*

Alice

Alice knows a number s such that **$M = g^s \bmod p$** and wants to prove it to Bob

Alice → Bob: {U= g^r mod p }

r is random number ∈ [1..q]

Bob

Bob also know M

Bob → Alice: {a}

a is random number ∈ [1..q]

Alice → Bob: {x}

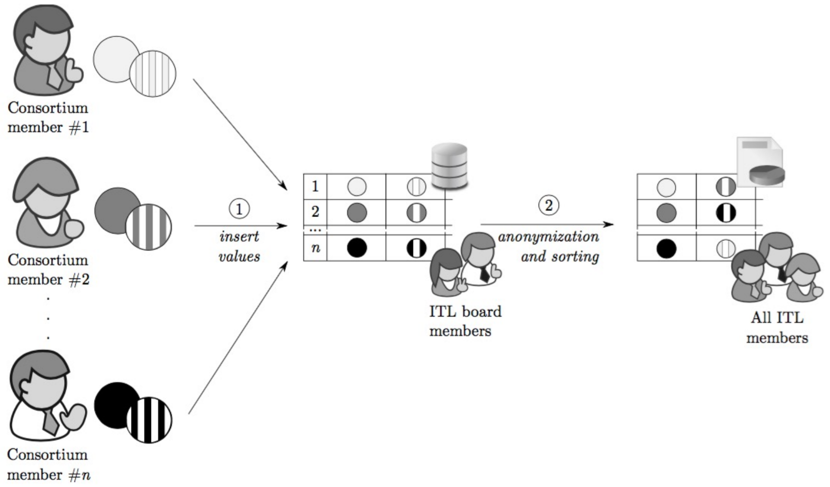
x = r + sa

Alice now shows that she knows s without revealing the value

Bob can verify that
$U = g^x M^{-a}$
$= g^{r+sa}(M)^{-a}$
$= g^{r+sa}(g^s)^{-a} \bmod p$
$= g^r \bmod p$

Secure Multiparty Computation

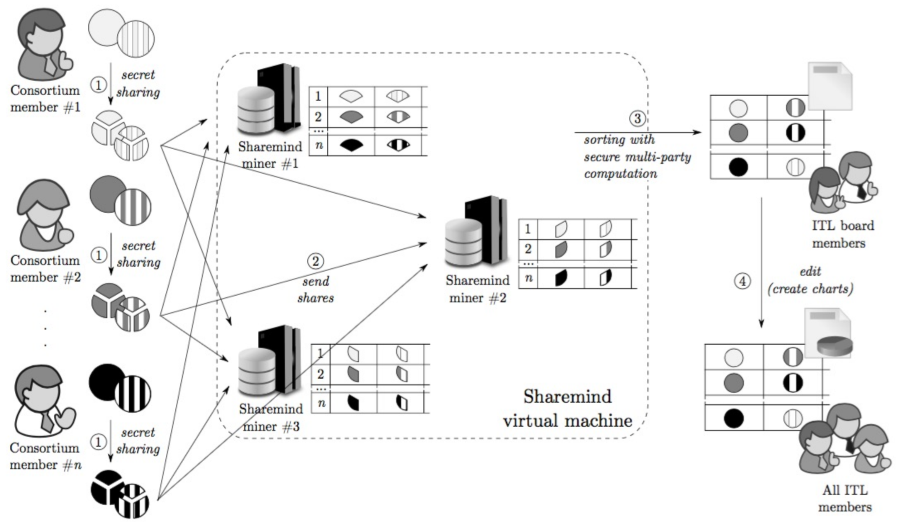
- Use Case
  - In order to analyze the economic situation of an industrial sector, a secure system is needed for jointly collecting and analyzing sensitive financial data
  - The financial data should be kept
    - Confidential
    - Anonymous



Deploying secure multi-party computation for financial data analysis
D. Bogdanov, R. Talviste and J. Willemson

Secure Multiparty Computation

- Use Case
  - Improved version
    - Data stored/sorted separately on three servers
    - No single party has access to original data
    - Anonymous to the board members



Deploying secure multi-party computation for financial data analysis
D. Bogdanov, R. Talviste and J. Willemson

Secure Computation Approaches

Multi-Party Computation (MPC)

Pros

- Low compute requirements
- Easy to accelerate
- Provably secure
- Supports multiple threat models
- Easy to map existing algorithms

Cons

- High communication costs
- High latency
- Information theoretic proofs are weaker than PKE ones

Fully Homomorphic Encryption (FHE)

Pros

- Very low communication costs
- Requires a single round of communications, i.e., "fire and forget"
- Useful when one side is limited in compute / memory / storage
- Provably secure – relies on strength of PKE

Cons

- Very high computational requirements
- Harder to accelerate
- Mapping existing algorithms to FHE may be difficult

Trusted Execution Environments (TEE)

Pros

- No communication required
- Trivial to accelerate
- Great support for existing software

Cons

- Weaker security guarantees
- Cannot stop determined adversaries
- Historically plagued by vulnerabilities and breaches
- Long term deployment is difficult – TEE's can 'run out' of entropy / CRP's, etc.

Upcoming Lectures

- Secure Computation Approaches
  - Trusted Execution Environment (TEE)
  - Homomorphic Encryption