CIS 2018

Efficient MPC

Correlated Randomness
and Arithmetic Circuits

Claudio Orlandi, Aarhus University

We’re hiring!

- **PhD students, postdocs, assistant professors (tenure track), associate professors**
- **Topics**: blockchain, differential privacy, zero-knowledge proofs, secure multiparty computation, formal verification, language design and semantics for smart contracts, …
- More info at https://iacr.org/jobs/

AARHUS UNIVERSITET

Online Poker

2♠, 5♠ ,2♥,5 ♥,J♦

Q♠,Q♣,7♣,3♥,2♦

10 ♠,9♣, 8♣,7♦,6♦

3♠, 4♠,7♥,Q ♦,10♦

Poker with Pirates

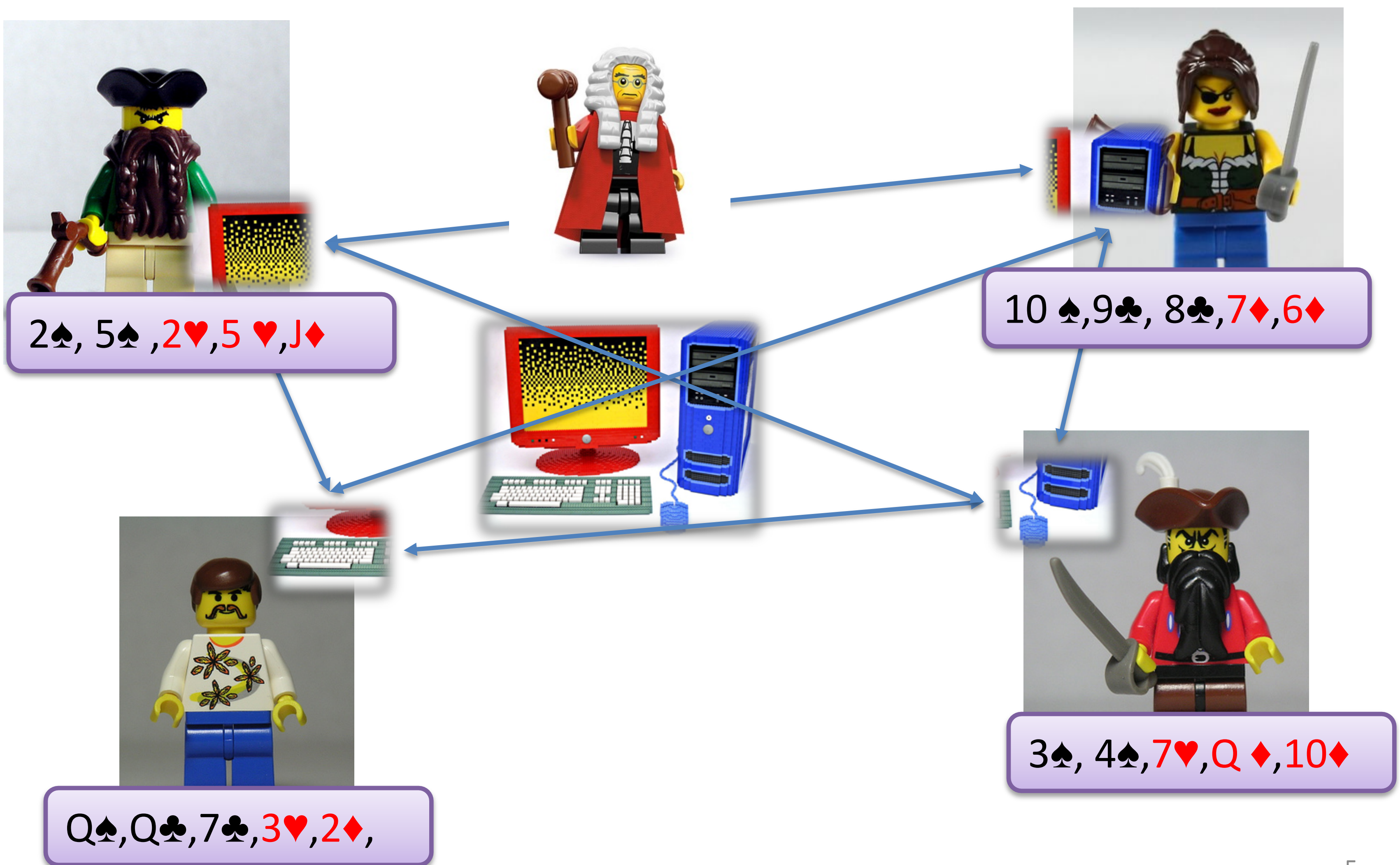
2♠, 5♠ ,2♥,5 ♥,J♦

Q♠,Q♣,7♣,3♥,2♦,

10 ♠,9♣, 8♣,7♦,6♦

A♠,A♣,A♥,A♦,K♦

Secure Computation



Hospitals and Insurances

Syge mister millioner af kroner

Af CHARLOTTE BEDER
Offentliggjort 19.02.09 kl. 08:39

Danskerne går årligt glip af 80 mio. kr., fordi de ikke aner, at de er forsikret ved kritisk sygdom.

Brystundersøgelse. Foto: Colourbox

Relaterede artikler

- Nyt system sikrer syge 80 mio. kr.
- Forsikringsklager i bund
- Boom i sundhedsforsikringer

Hundredvis af alvorligt syge danskere går hvert år glip af millioner af kroner, fordi de ikke har overblik over deres forsikringsdækning.

Derfor kontakter de ikke deres pensions- eller forsikringsselskab, når de bliver ramt af kræft, blodpropper eller anden kritisk sygdom. Og så får de aldrig den check på typisk mellem 50.000 og 200.000 kr., som de har ret til, lyder det fra forsikrings- og pensionsbranchen.

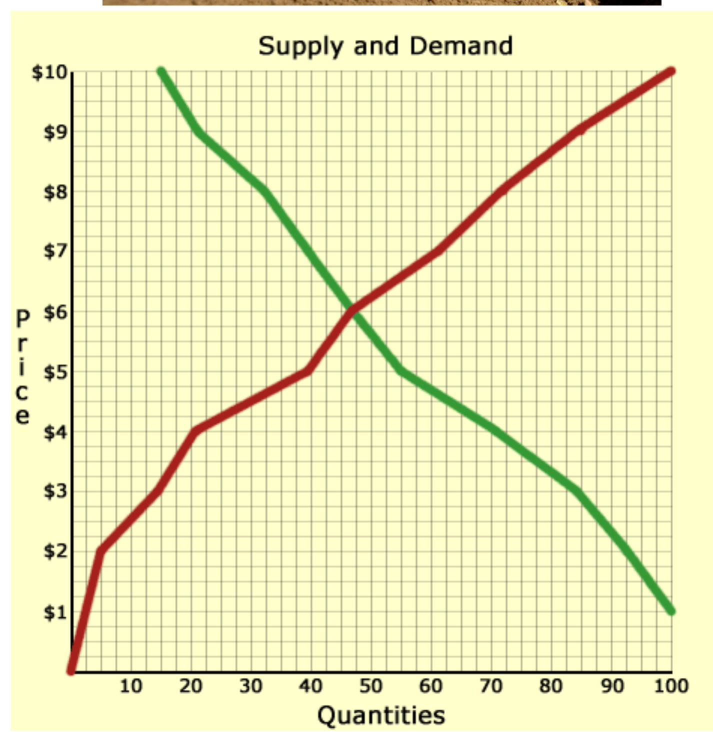
»Forudsætningen er, at systemet skrues sammen på en måde, så selskaberne ikke får andre oplysninger om kunderne, end de bør få. For det enkelte individ må ikke miste kontrollen over egne helbredsoplysninger,« siger jurist Lars Kofod.

- **Problem:** Sick people forget to claim compensations from insurance
- **Solution:** Insurances and hospitals could periodically compare their data to find and help these people
- **Privacy Issue:** insurance and medical records are sensitive data! No other information than what is strictly necessary must be disclosed!

MPC Goes Live (2008)

Bogetoft et al.
"Multiparty Computation Goes Live"

- January 2008
- **Problem**: determine market price of sugar beets contracts
- 1200 farmers
- Computation: 30 minutes



Last decade: commercial interest and social value of MPC

- **Estonian study on student dropout**
- **Boston women workforce councile, study on wage gap**

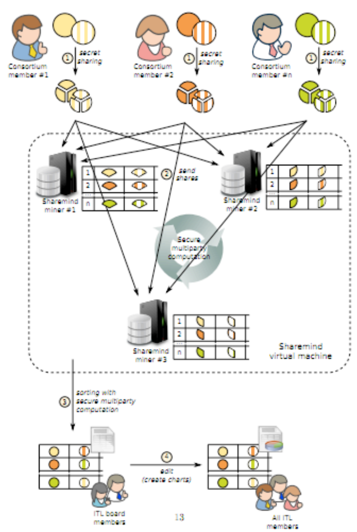


Figure 3: Data flow and visibility in the improved solution using the SHAREMIND framework.

sharemind

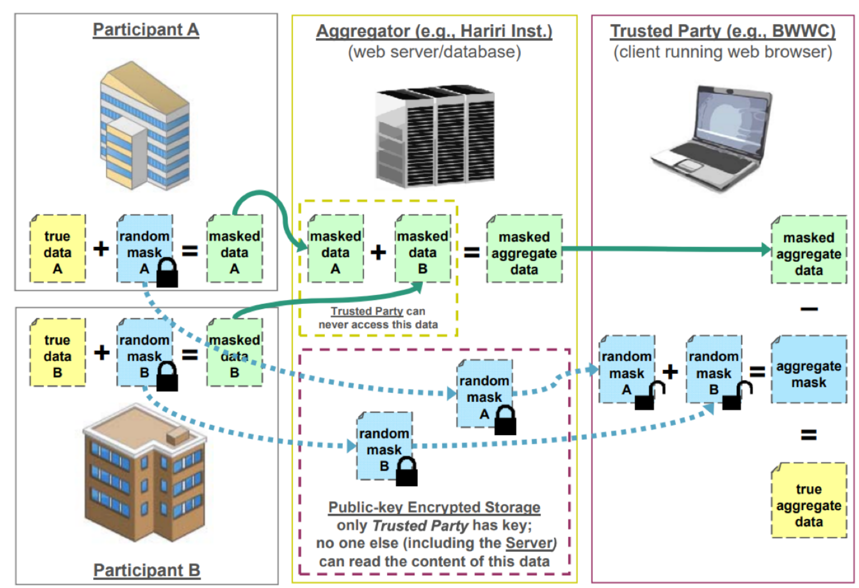
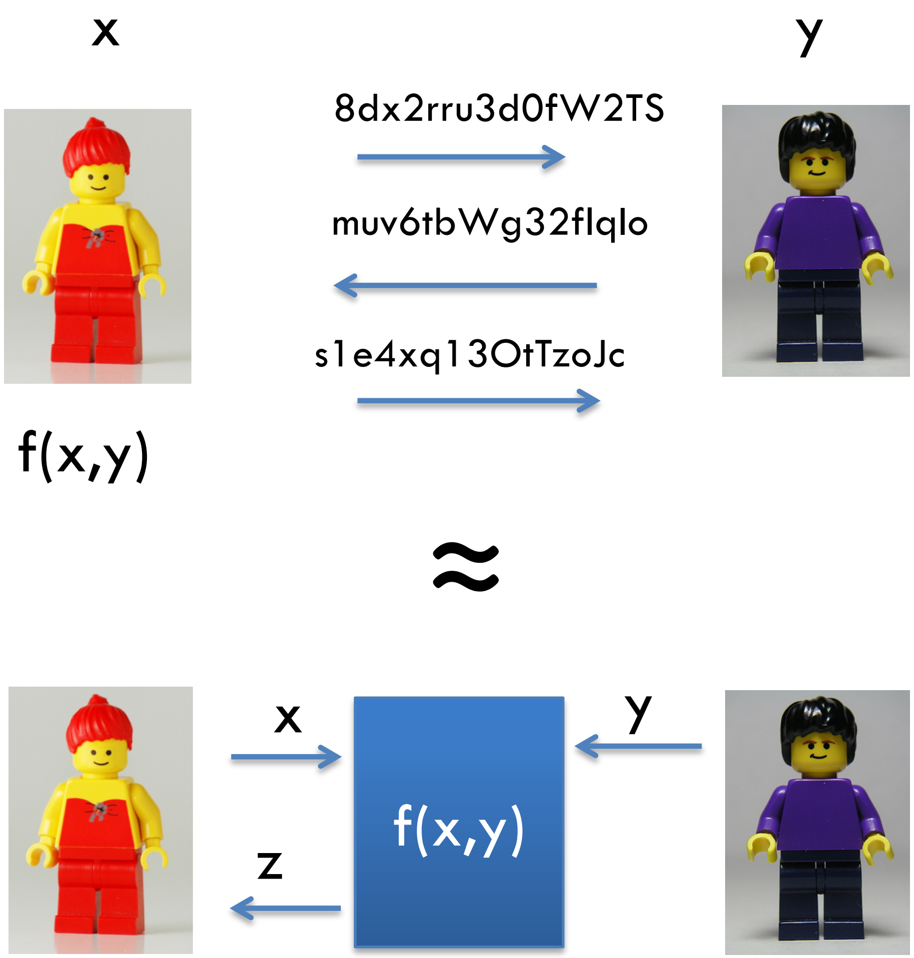


Figure 1: Illustration of a deployment of the protocol implementation for two participants.

Secure Computation



- *Privacy*
- *Correctness*
- *Input independence*
- …

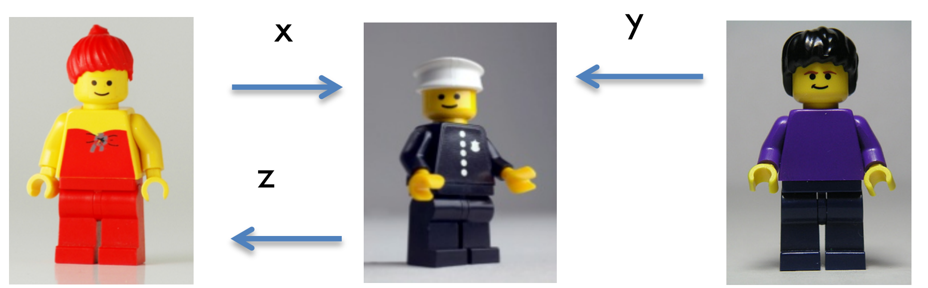
Part 1: Correlated Randomness and Arithmetic Circuits

- **Warmup: One-Time Truth Tables**

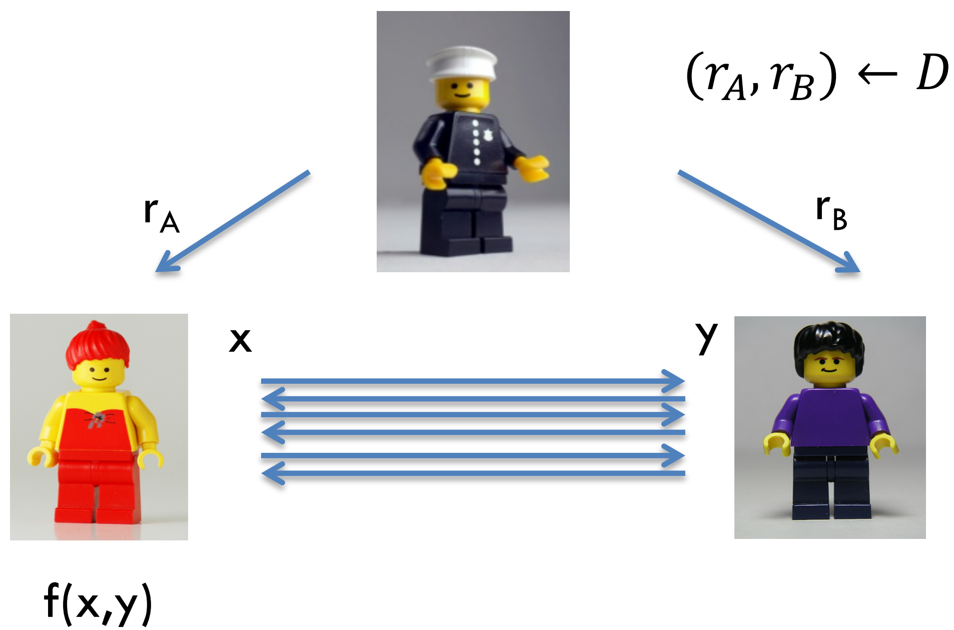
- Arithmetic Black Box and Evaluating Circuits with Beaver's trick

- Simple Unconditionally Secure Protocols

Trusted Party



Trusted Dealer



"The simplest 2PC protocol ever"

$$(r_A, r_B) \leftarrow D$$

r_A r_B

x y

$f(x,y)$

“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

1) Write the truth table of the function F you want to compute

y

x		0	1	2	3
	0	3	2	2	2
	1	3	0	0	4
	2	1	0	0	4
	3	1	1	4	4

“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

2) Pick random (r, s), rotate rows and columns

s=3

r=1

	0	1	2	3
0	1	4	4	1
1	2	2	2	3
2	0	0	4	3
3	0	0	4	1

“The simplest 2PC protocol ever” OTTT (Preprocessing phase)

3) Secret share the truth table i.e.,

Pick T1 at random, and let

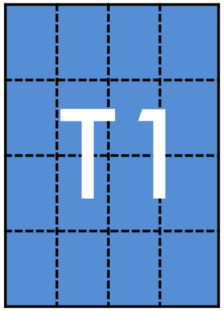
T2 =

1	4	4	1
2	2	2	3
0	0	4	3
0	0	4	1

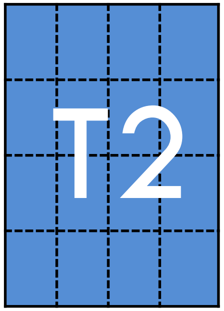
\- T1

“The simplest

“Privacy”:
inputs masked w/uniform random values



, r



, s

$u = x + r$ →

$v = y + s$ ←

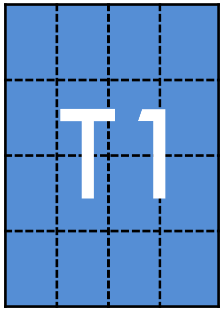
$z_2 = T2[u,v]$ ←

Correctness:
by construction

output $f(x,y) = T1[u,v] + z_2$

“The simplest 2PC ” OTTT

Simulated view, given x and f(x,y) (but not y)



, r

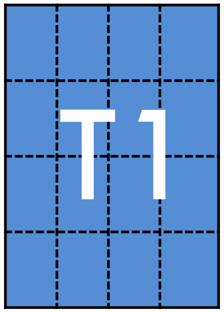
$u = x + r$ →

v (random) ←

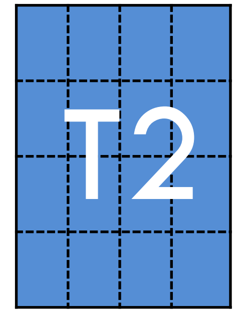
$z_2 = f(x,y) - T1[u,v]$ ←

output $f(x,y) = T1[u,v] + z_2$

What about active security?



, r



, s

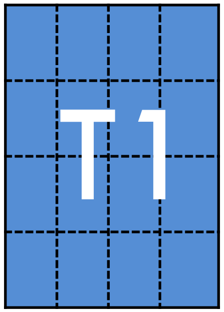
$u = x + r$ →

$v = y + s$ ←

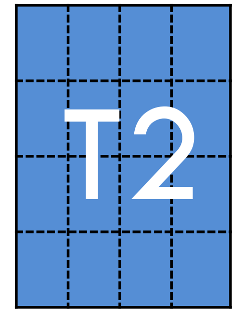
$z_2 = T2[u,v]$ ←

output $f(x,y) = T1[u,v] + z_2$

What about active security?



, r



, s

u = x + r →

← v = y + s + e1

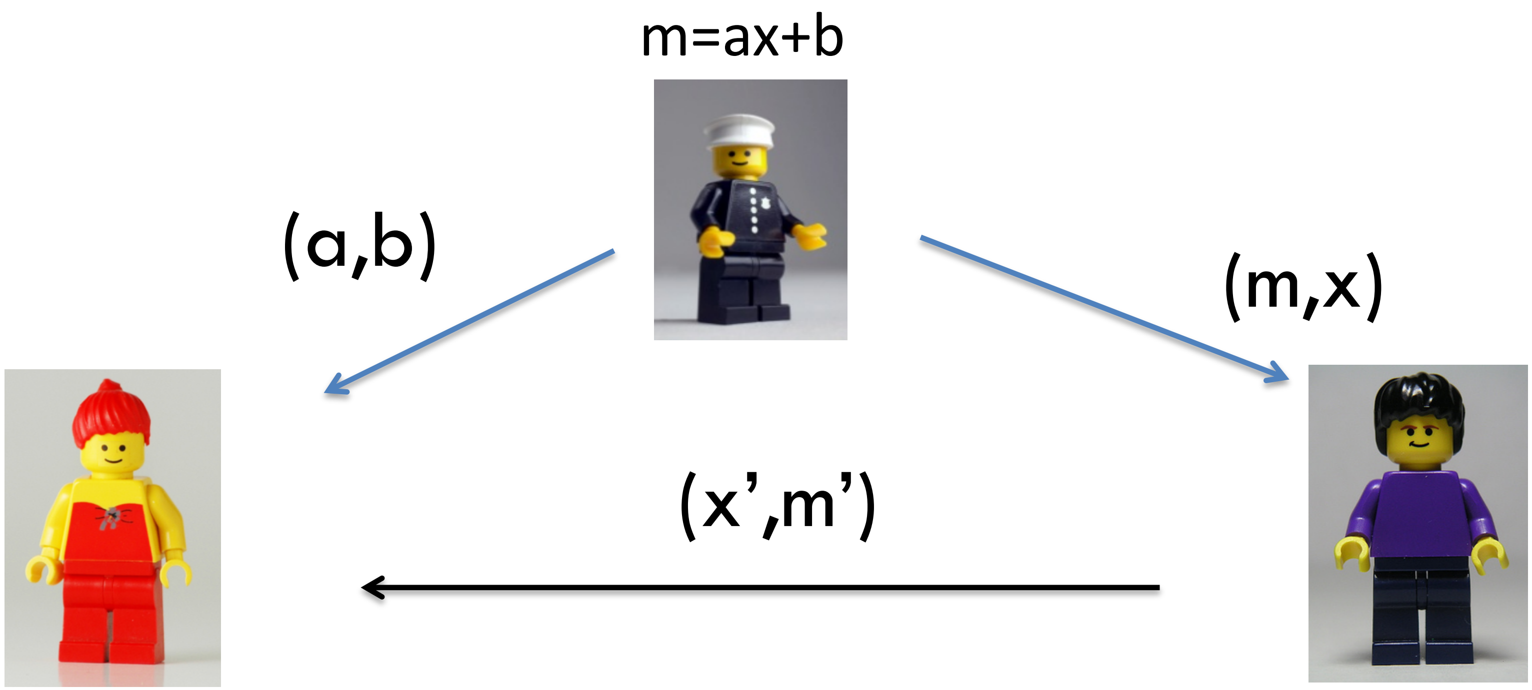
← T2[u,v] + e2

Is this cheating?

- $v = y + s + e1 = (y+e1) + s = y' + s$
  - Input substitution, not cheating according to the definition!
- $M2[u,v] + e2$
  - Changes output to $z' = f(x,y) + e2$
  - Example: $f(x,y)=1$ iff $x=y$ *(e.g. pwd check)*
  - $e2=1$ the output is 1 whp *(login without pwd!)*
    - *Clearly breach of security!*

Force Bob to send the right value

- **Problem:** Bob can send the wrong shares
- **Solution:** use MACs
  - e.g. m=*ax+b with (a,b)* ← *F (e.g., F=*$\mathbb{Z}_p$ *with p≥*2^k *prime)*



Abort if m'≠ax'+b

OTTT+MAC

T1, r A B

T2, s M

u = x + r →

← v = y + s

← T2[u,v], M[u,v]

If (M[u,v]=A[u,v]*T2[u,v]+B[u,v])

output f(x,y) = T1[u,v] + T2[u,v]

else

abort

Statistical security vs. malicious Bob w.p. $1-2^{-k}$

“The simplest 2PC protocol ever” OTTT

- **Optimal communication complexity ☺**

- **Storage exponential in input size ☹**

➔Represent function using circuit instead of truth table!

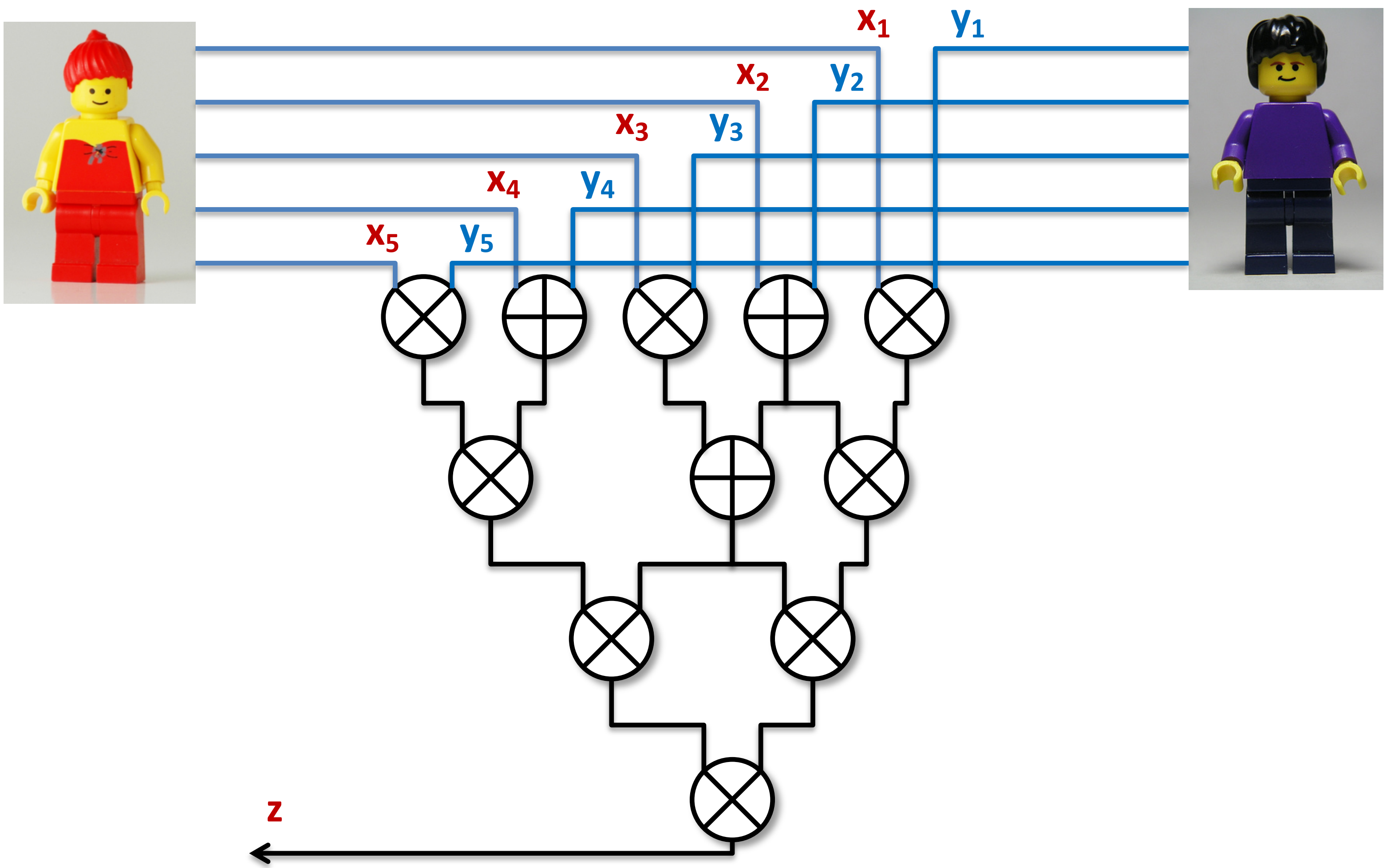
Part 1: Correlated Randomness and Arithmetic Circuits

- ~~Warmup: One-Time Truth Tables~~

- **Arithmetic Black Box and Evaluating Circuits with Beaver's trick**

- Simple Unconditionally Secure Protocols

Circuit based computation



What kind of circuit?

- **Boolean**
  - Addition & Multiplication modulo 2 (XOR, AND)
- **Arithmetic: which modulo?**
  - In a field ($\mathbb{Z}_p$, GF(2^k))?
  - Determined by Public Key (e.g., Paillier, LWE, …)
  - Arbitrary? (e.g., modulo 2^{32})

The Arithmetic Black Box (ABB)

- **A reactive functionality which allows to manipulate secret values**

- **Often a good abstraction:**
  - if you want to implement some algorithm in MPC, you might not care too much about how operation are implemented, just what the ”interface” is.

ABB: Basic Commands

- **[x] ← Input(P_i, x)**
  - Party P_i inputs a secret value x, all other parties get a "handle/pointer" to [x]
- **x ← Open(P_j, [x])**
  - If all parties agree, party P_j learns the secret value contained in [x]
- **[z]←Add([x],[y])** // or [z]=[x]+[y]
  - If all parties agree, a new handle [z] is created such that z=x+y
  - [z]←Add(c,[x]), [z]←Mul(c,[x]) easy from Add
- **[z]←Mul([x],[y])** // or [z]=[x]*[y]
    - If all parties agree, a new handle is created such that z=x*y

ABB: Advanced (Efficient) Commands

- **[r] ← Rand()**
  - Generate a random handle for r
  - Could have been implemented by $[r_i]$←Input(P_i,r_i) and [r] ← $[r_1]$+…+$[r_n]$
- **b ← IsZero([x])**
  - Could be implemented by [z]=[x]*[r] for random r, then open z and check if = 0.
- **$[x_1]$,…,$[x_n]$ ← BitsOf([x])**
  - Useful and typically expensive

- **Exercise**: how would you implement these?
  - **[b] ← IsZero([x])** // b=1 iff x=0
  - **b ← Equality([x],[y])** // b=1 iff x=y
  - **b ← IsBit([x])** // b=1 iff x∈{0,1}

Beaver's random triples trick

[z]←Mul([x],[y]):

1. ([a],[b],[c])←RandMul()

   *Creates random tuple such that c=a*b*

2. *e=Open([a]+[x])*
3. *d=Open([b]+[y])*

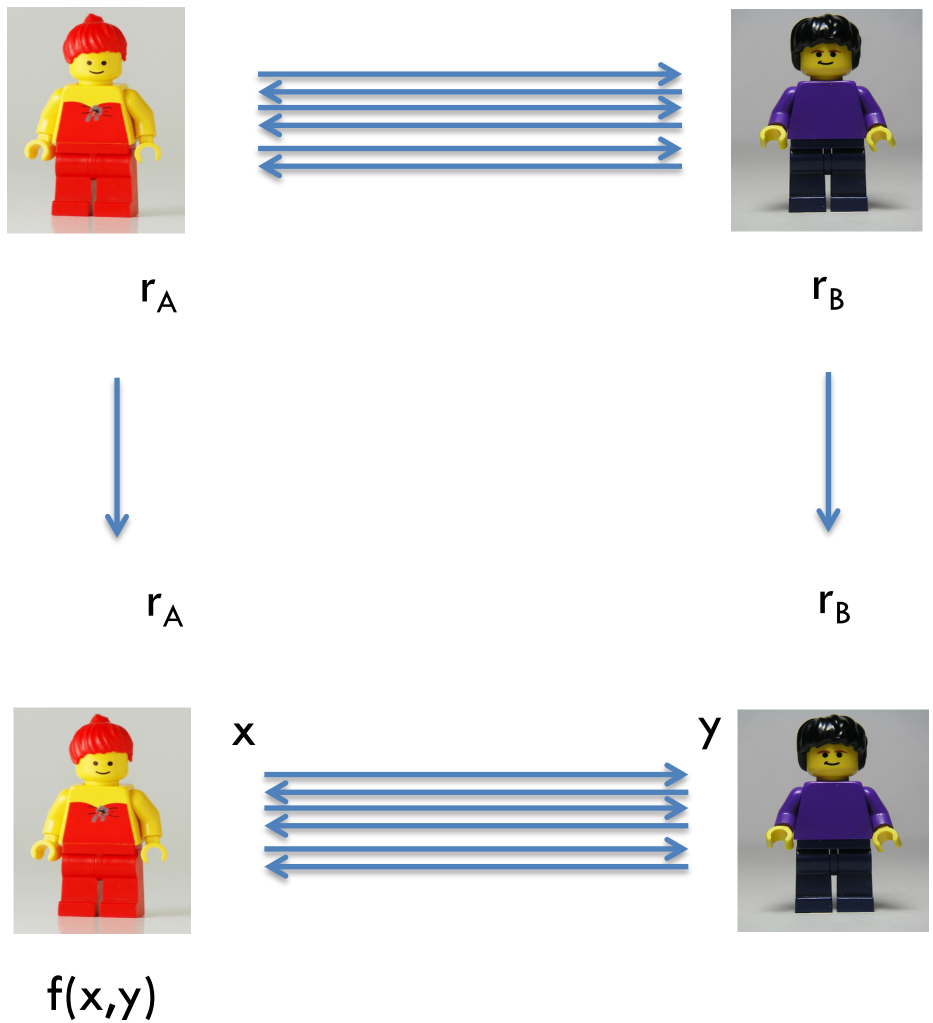
Is this secure?
e,d are "one-time-pad" encryptions of x and y using a and b

4. Compute [z] = [c] + e[y] + d[x] - ed

   ab + (ay+xy) + (bx+xy) - (ab+ay+bx+xy)

Beaver and Preprocessing

Preprocessing



- Independent of ***x,y***
- Tipically only depends on ***size of f***
- Uses public key crypto technology ***(slower)***

Online Phase

- Uses only information theoretic tools ***(order of magn. faster)***

Implementing the Arithmetic Black Box

- How to implement the basic commands?
  - Input, Add, Mul/RandMul

- In the remaining time:
  - **Additive Secret Sharing**
    - **Passive Security**
    - Active Security
  - Replicated Secret Sharing
  - Shamir Secret Sharing

Invariant

- For each ***wire x*** in the circuit we have
  - [x] := (x_1, x_2) *// read "x in a box"*
  - Where Alice holds x_1
  - Bob holds x_2
  - Such that $x_1+x_2=x$

- Notation overload:
  - x is both the r-value and the l-value of x
  - use n(x) for name of x and v(x) for value of x when in doubt.
  - Then [n(x)] = (x_1,x_2) such that $x_1+x_2=v(x)$

Circuit Evaluation

1) [x] ← Input(A,x) :

- chooses random x_2 and send it to Bob
- set $x_1=x+x_2$ mod M // symmetric for Bob
  // mod omitted from now on

Alice only sends a random value! "Clearly" secure

2) z ← Open(A,[z]):

- Bob sends z_2
- Alice outputs $z=z_1+z_2$ // symmetric for Bob

Alice should learn z anyway! "Clearly" secure

Circuit Evaluation

2) [z]← Add([x],[y]) // at the end z=x+y

- Alice computes $z_1 = x_1 + y_1$
- Bob computes $z_2 = x_2 + y_2$

No interaction! “Clearly” secure

“for free” : only a local addition!

Circuit Evaluation

2a) [z]← Mul(c,[x]) // at the end z=c*x

- Alice computes $z_1 = c*x_1$
- Bob computes $z_2 = c*x_2$

2c) [z]← Add(c,[x]) // at the end z=c+x

- Alice computes $z_1 = c+x_1$
- Bob computes $z_2 = x_2$

Circuit Evaluation (Online phase)

3) Multiplication?

How to compute [z]=[xy] ?

How do we compute this?

Alice, Bob should compute

$$z_1 + z_2 = (x_1+x_2)(y_1+y_2)$$

$$= x_1y_1 + x_2y_1 + x_1y_2 + x_2y_2$$

Alice can compute this

Bob can compute this

RandMul() with Trusted Dealer

Pick random

a_1, a_2, b_1, b_2, c_1

and

$c_2 = (a_1+a_2)(b_1+b_2) - c_1$

a_1, b_1, c_1

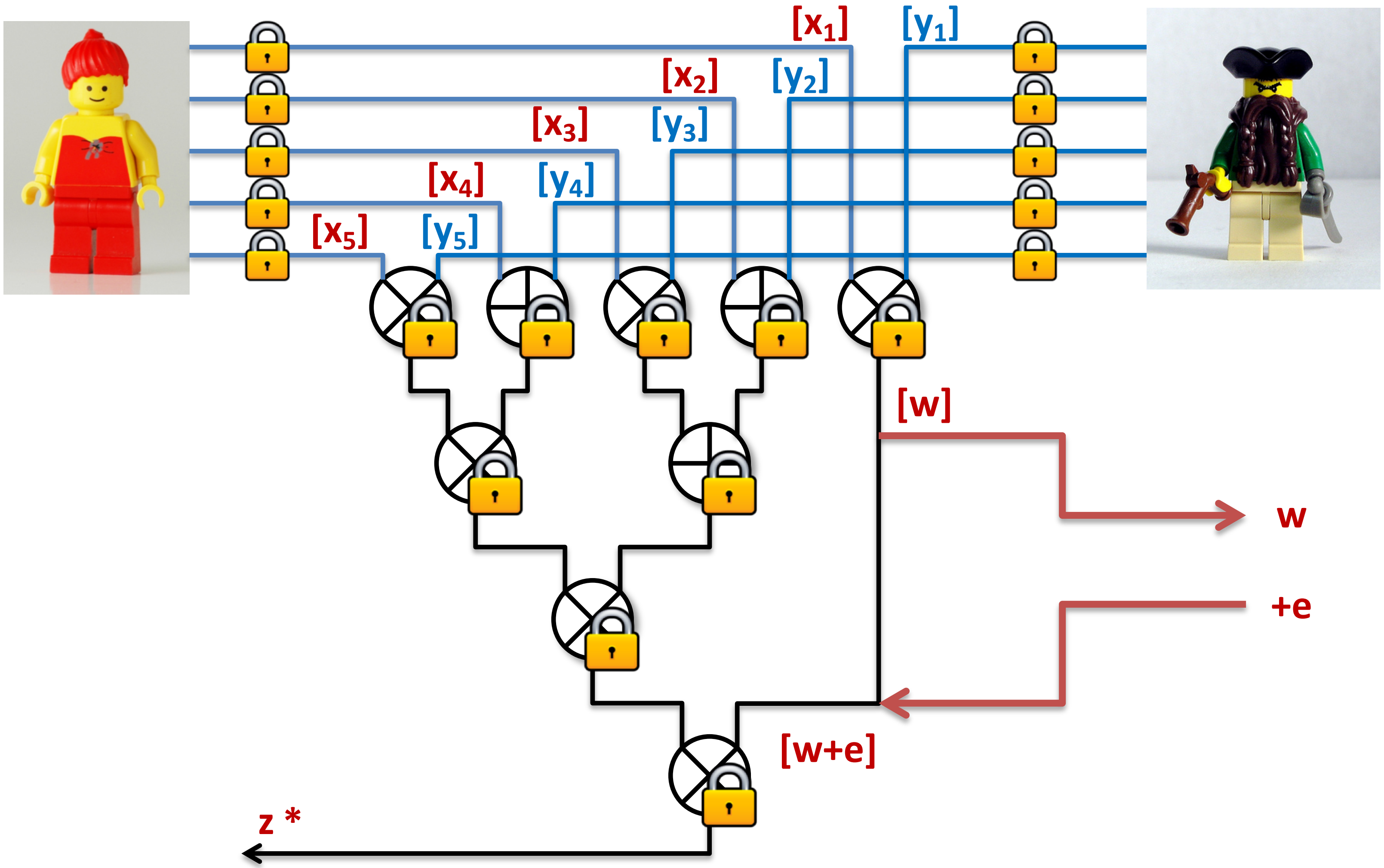
a_2, b_2, c_2

Implementing the Arithmetic Black Box

- How to implement the basic commands?
  - Input, Add, Mul/RandMul

- In the remaining time:
  - **Additive Secret Sharing**
    - Passive Security
    - **Active Security**
  - Replicated Secret Sharing
  - Shamir Secret Sharing

Secure Computation



Active Security?

- **“Privacy?”**
  - even a malicious Bob does not learn anything ☺
- **“Correctness?”**
  - a corrupted Bob can change his share during any “Open” (both final result or during multiplication) leading the final output to be incorrect ☹

Problem

2) z ← Open(A,[z]):

- Bob sends $z_2 + e$
- Alice outputs $z = z_1 + z_2 + e$ // error change output distribution in way that cannot be simulated by input substition

Authenticated Shares

- **Passive share**: [x] means
  - Alice has x_1, Bob has x_2,

$$x_1 + x_2 = x$$

- **MAC on Share** $[\![x]\!]$ (BeDOZa, TinyOT, …):
  - [x] plus:
  - Bob has a MAC key (Δ_2, K_2), Alice has a MAC M_1 :

$$M_1 = \Delta_2 \, x_1 + K_2$$

  - (Symmetric for Bob)

Authenticated Shares

- **Is the representation $[\![x]\!]$ still linear?**
  Yes, if Δ_1, Δ_2 are "global" **keys**

$$[\![x]\!] = ([x], (\Delta_1, K_1(x), M_1(x)), (\Delta_2, K_2(x), M_2(x)))$$

$$[\![y]\!] = ([y], (\Delta_1, K_1(y), M_1(y)), (\Delta_2, K_2(y), M_2(y)))$$

$$[\![z]\!] = ([x+y], (\Delta_1, K_1(x)+K_1(y), M_1(x) + M_1(y)), (\Delta_2, K_2(x)+K_2(y), M_2(x) + M_2(y)))$$

Better MACs for MPC

- **SPDZ:**
  - **Problem**: with MAC on Share you need to store a MAC for every other party!
  - **Solution**: MAC value directly instead
  - $[\![x]\!] = ([x], [M(x)], [\Delta])$ with $M(x) = \Delta x$ **(**Δ is global)
- **MiniMAC:**
  - **Problem**: MAC must be large for unpredictability. If working in small field, need to have multiple MACs per value.
  - **Solution**: Compute MAC on vector of bits instead
- **SPDZ2K:**
  - **Problem**: MACs don't work modulo power of 2's (not a field).
  - **Solution**: compute MAC modulo 2^{k+s}
- …

Implementing the Arithmetic Black Box

- How to implement the basic commands?
  - Input, Add, Mul/RandMul

- In the remaining time:
  - **Additive Secret Sharing**
    - Passive Security
    - **Active Security**
  - Replicated Secret Sharing
  - Shamir Secret Sharing

Implementing the Arithmetic Black Box

- How to implement the basic commands?
  - Input, Add, Mul/RandMul

- In the remaining time:
  - Additive Secret Sharing
    - Passive Security
    - Active Security
  - **Replicated Secret Sharing**
  - Shamir Secret Sharing

n=3 parties
t≤1 corruptions

Replicated Secret Sharing

- [x] means:
  - $x = x_1+x_2+x_3$ where
  - P_1 knows (x_1,x_2)
  - P_2 knows (x_2,x_3)
  - P_3 knows (x_3,x_1)

No party alone can reconstruct the secret

- [x] ← Input(P_i,x)
  - P_i picks random shares and distributes them.
- x ← Open(P_i,[x])
  - Everyone sends their shares to P_i who reconstructs.
- [x] ← Add([x],[y])
  - Everyone locally adds their shares.

n=3 parties
t≤1 corruptions

Replicated Secret Sharing

- [z]=Mul([x],[y])

Goal, compute random such that

$$z = (x_1+x_2+x_3)(y_1+y_2+x_3)$$

$$= \begin{matrix} x_1y_1 + x_2y_1 + x_3y_1 + \\ x_1y_2 + x_2y_2 + x_3y_2 + \\ x_1y_3 + x_2y_3 + x_3y_3 \end{matrix}$$

P_1 P_2 P_3

n=3 parties
t≤1 corruptions

Replicated Secret Sharing

- [z]=Mul([x],[y])
  - P_1 computes $z_1 = x_1y_1 + x_2y_1 + x_1y_2$
    - Symmetric for P_2, P_3, …
  - $[z_1] \leftarrow$ Input(P_1,z_1) *// Why resharing?*
    - Symmetric for P_2, P_3, …
  - $[z]=[z_1]+[z_2]+[z_3]$

Implementing the Arithmetic Black Box

- How to implement the basic commands?
  – Input, Add, Mul/RandMul

- In the remaining time:
  – Additive Secret Sharing
    - Passive Security
    - Active Security
  – Replicated Secret Sharing
  – **Shamir Secret Sharing**

Shamir vs. Replicated Secret Sharing

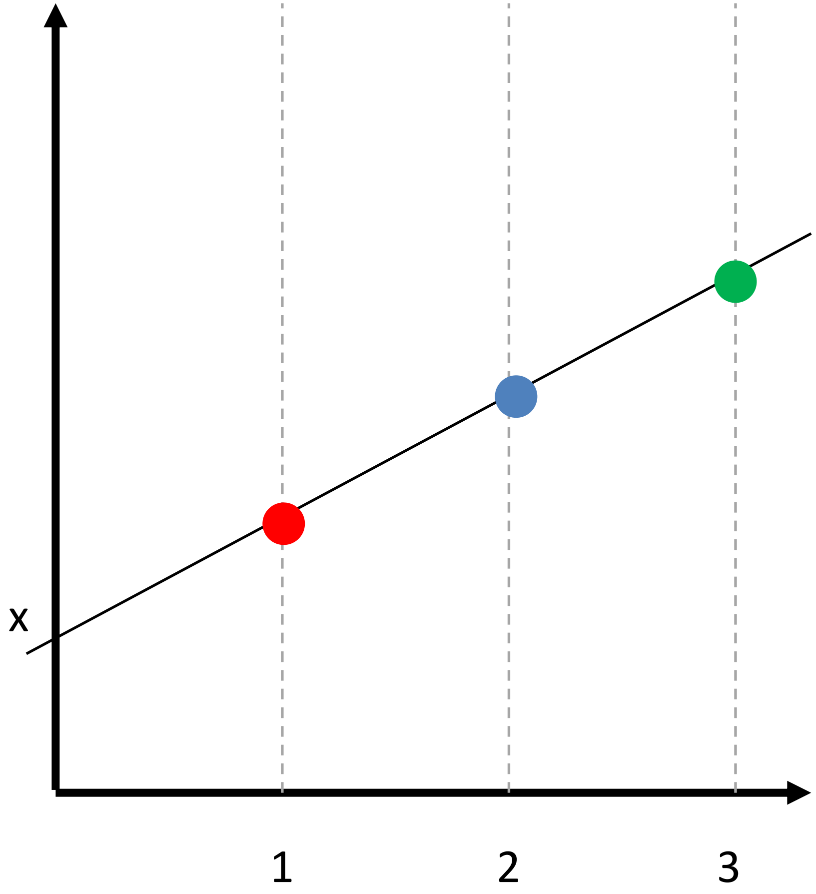
- **Share size:**
  - Shamir is optimal (size of share = size of secret)
  - RSS scales horribly with the number of parties

- **Generality:**
  - Shamir works only in fields
  - RSS works in any ring

n=3 parties
t≤1 corruptions
Computations in field

Shamir Secret Sharing

- [x] means:
  - $x=p(0)$ where
  - $p(\alpha) = x_0 + x_1\alpha$
  - P_1 knows $p(1)$
  - P_2 knows $p(2)$
  - P_3 knows $p(3)$

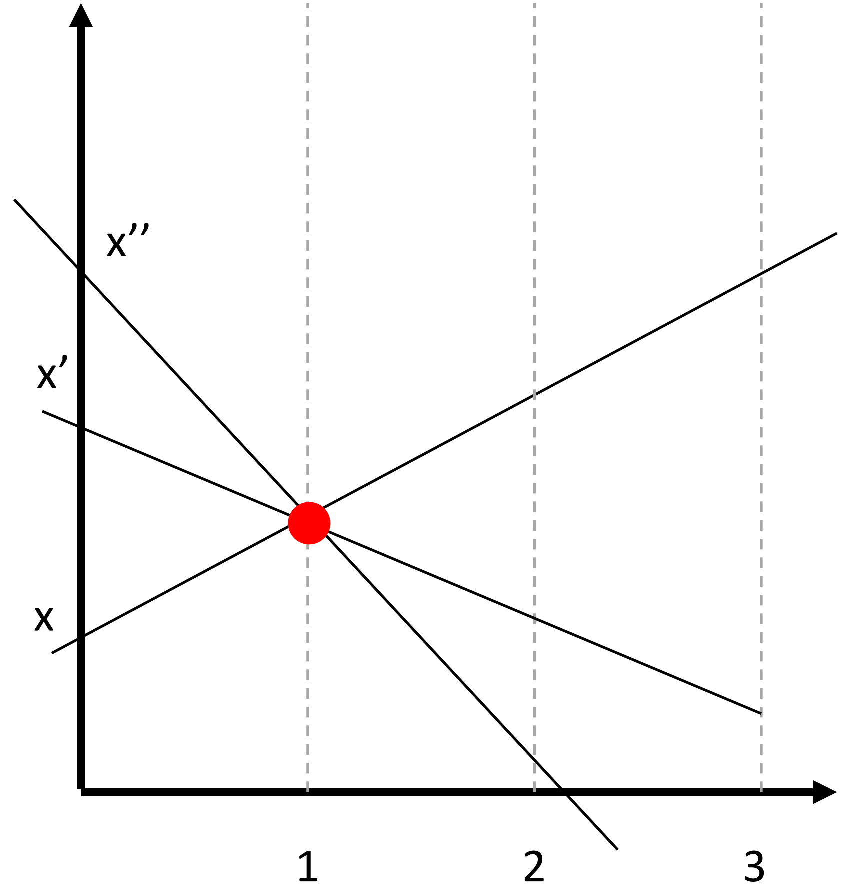


n=3 parties
t≤1 corruptions
Computations in field

Shamir Secret Sharing

- [x] means:
  - $x=p(0)$ where
  - $p(\alpha) = x_0 + x_1\alpha$
  - P_1 knows $p(1)$
  - P_2 knows $p(2)$
  - P_3 knows $p(3)$

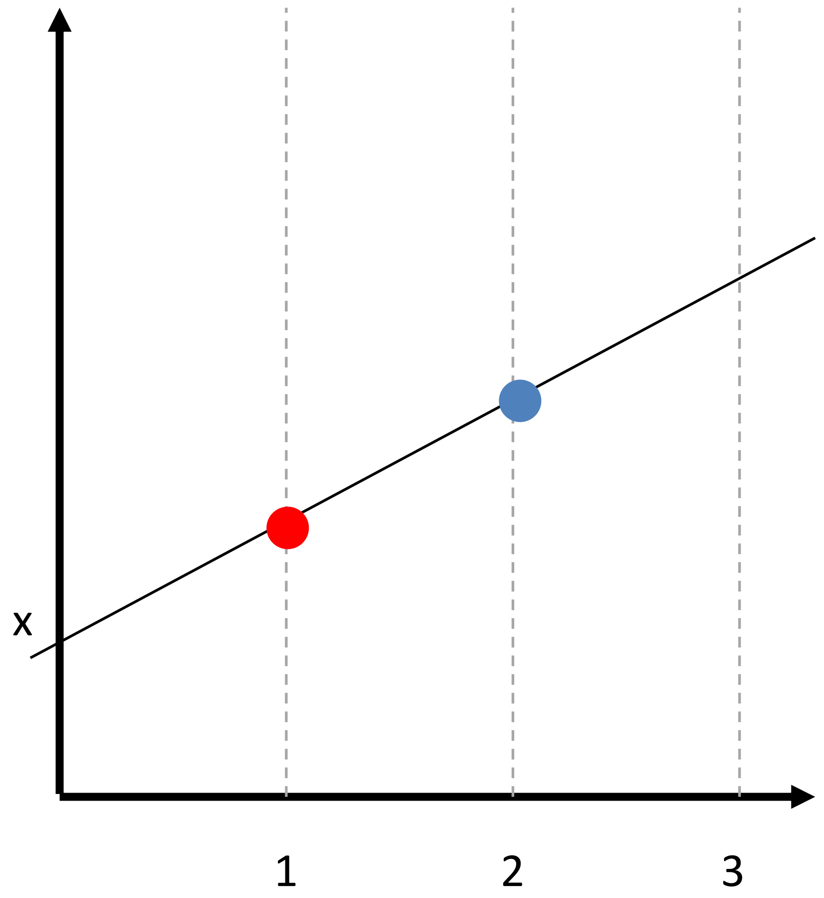
No party alone can reconstruct the secret



n=3 parties
$t \leq 1$ corruptions
Computations in field

Shamir Secret Sharing

- [x] means:
  - $x=p(0)$ where
  - $p(\alpha) = x_0 + x_1\alpha$
  - P_1 knows $p(1)$
  - P_2 knows $p(2)$
  - P_3 knows $p(3)$



Any two parties can reconstruct x

Reconstruction - Details

- Given $p(1)$, $p(2)$ one can reconstruct $p(x)$ as

  $p(\alpha)=\delta_1(\alpha)p(1)+\delta_2(\alpha)p(2)$

- $\delta_i(\alpha)$ is a poly s.t.

  $\delta_i(i)=1$

  $\delta_i(j)=0$ for all j in the reconstruction set (except i)

- In our case
  $\delta_1(\alpha)= (\alpha -2)(1-2)^{-1}$
  $\delta_2(\alpha)= (\alpha -1)(2-1)^{-1}$

- To reconstruct secret enough to compute $p(0)=\delta_1(0)p(1)+\delta_2(0)p(2)$

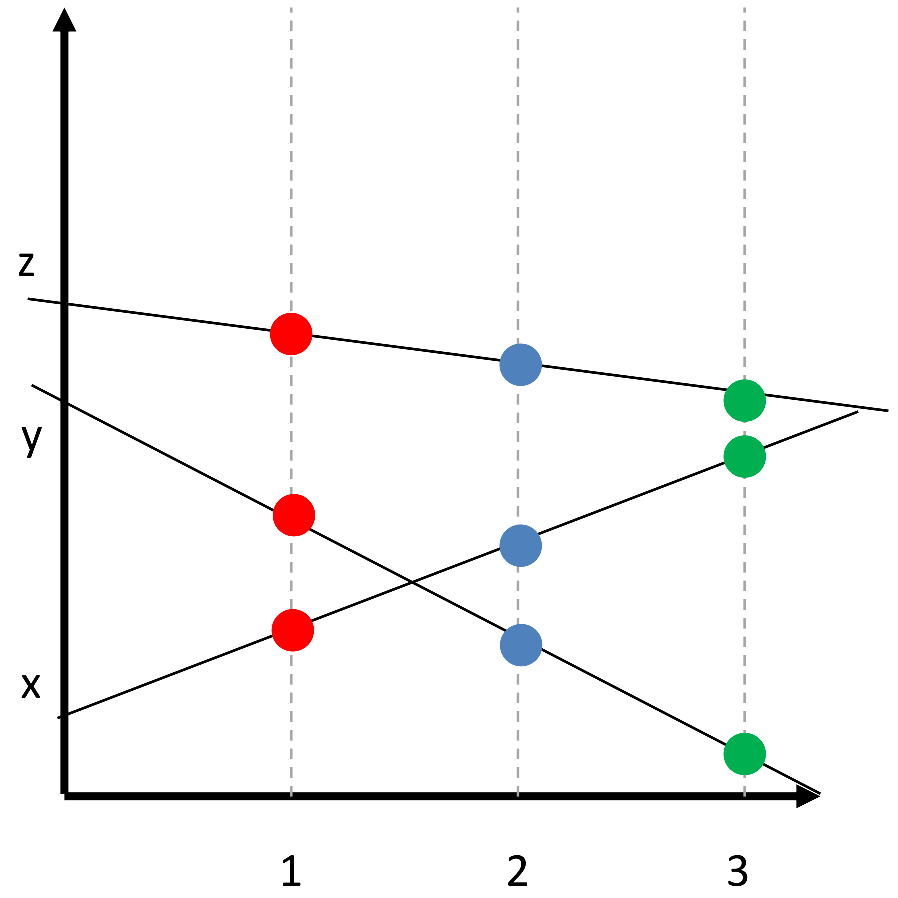
- (Generalizes to any other degree)

n=3 parties
t≤1 corruptions
Computations in field

Shamir Secret Sharing

- [z]=Add([x],[y]) means:
  - x=p(0), y=q(0)
  - $p(\alpha) = x_0 + x_1\alpha$
  - $q(\alpha) = y_0 + y_1\alpha$

  - P_1 computes p(1)+q(1)
  - P_2 computes p(2)+q(2)
  - P_3 computes p(3)+q(3)



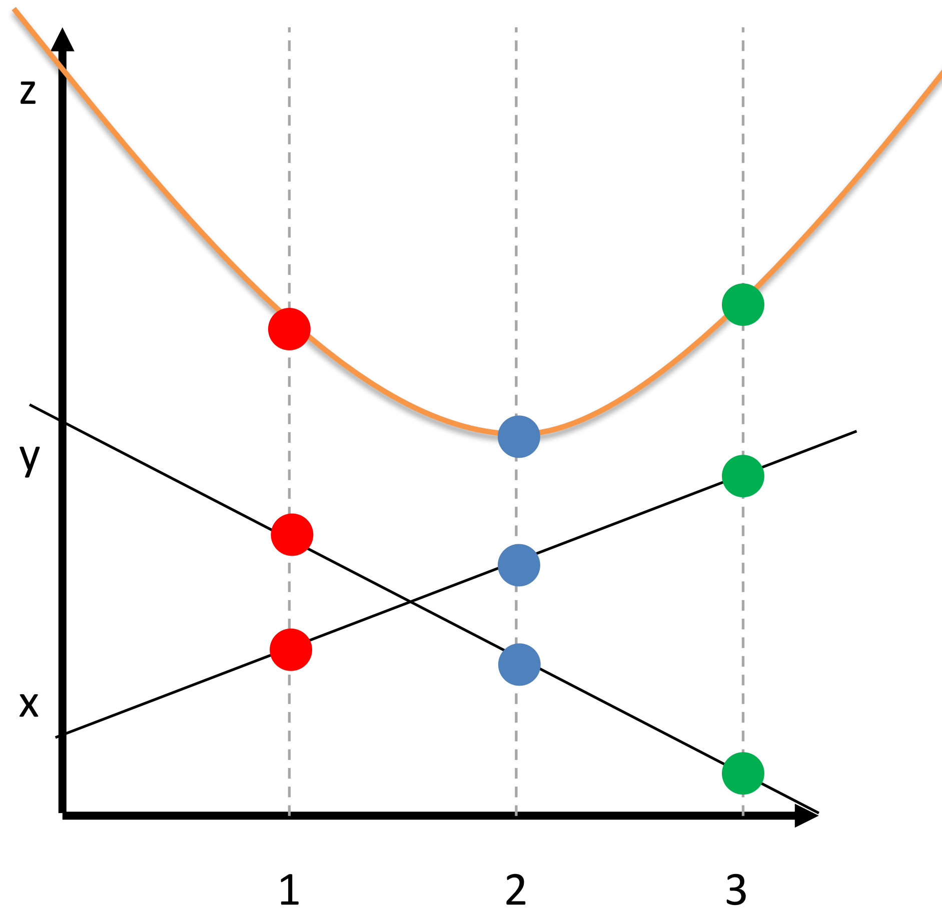
n=3 parties
t≤1 corruptions
Computations in field

Shamir Secret Sharing

- [z]=Mul([x],[y]) (part 1):
  - x=p(0), y=q(0)
  - $p(\alpha) = x_0 + x_1\alpha$
  - $q(\alpha) = y_0 + y_1\alpha$

  - P_1 computes t(1)=p(1)*q(1)
  - P_2 computes t(2)=p(2)*q(2)
  - P_3 computes t(3)=p(3)*q(3)

- t(0)=xy (as desired)
- But t has the wrong degree!
- $t(\alpha) = t_0 + t_1\alpha + t_2\alpha^2$



n=3 parties
t≤1 corruptions
Computations in field

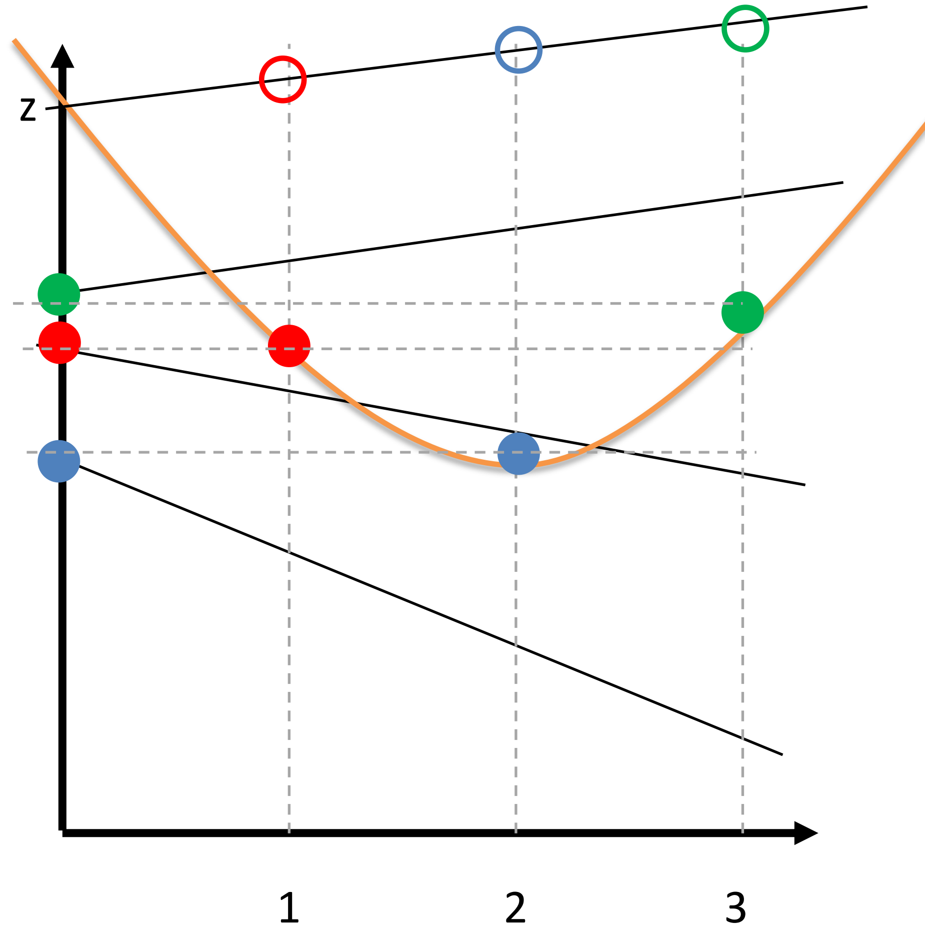
Shamir Secret Sharing

- $[z]=\mathrm{Mul}([x],[y])$ (part 2):
  - $[z_1] \leftarrow$ Input(P_1,t(1))
  - Symmetric for P_2, P_3
  - Then reconstruct i.e.

$[t(0)]=\delta_1[t(1)]+\delta_2[t(2)]+ \delta_2[t(3)]$

  - But t(0)=z, so we're done!

- Exercise: find the the values $\delta_1,\delta_2,\delta_3$ (Hint, the degree is different this time!)



Recap

- Simple protocols with trusted dealer
  - OTTT
  - Circuit evaluation with random triples
  - Active security via information theoretic MACs
- Simple protocols for 3 parties, 1 corruption
  - Replicated Secret Sharing
  - Shamir Secret Sharing

Coming up next:

- How to get rid of the trusted dealer?
  - Protocols for secure multiplication
  - OT and OT extension
- Efficiency of 2PC based on garbled circuits
  - Garbling techniques
  - Techniques for Active Security
- If time (and patience) allows
  - Anonymity in Cryptocurrencies

Primary References

- Cryptographic Computing, lecture notes, http://orlandi.dk/crycom (with theory and programming exercises)
- On the Power of Correlated Randomness in Secure Computation (Ishai et al.)
- Semi-homomorphic Encryption and Multiparty Computation (Bendlin et al.)
- Secure multi-party computation made simple (Maurer)
- A Full Proof of the BGW Protocol for Perfectly-Secure Multiparty Computation (Asharov et al.)
- A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority (Lindell et al.)

Other References

- A New Approach to Practical Active-Secure Two-Party Computation (Nielsen et al.)
- Web-based Multi-Party Computation with Application to Anonymous Aggregate Compensation Analytics (Lapets et al.)
- Multiparty Computation Goes Live (Bogetoft et al.)
- Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation (Bogdanov et al.)
- Efficient Multiparty Protocols Using Circuit Randomization (Beaver)
- How to Share a Secret (Shamir)
- Chaum et al. (Multiparty Unconditionally Secure Protocols)
- SPDℤ2k: Efficient MPC mod 2k for Dishonest Majority (Cramer et al.)
- Constant-Overhead Secure Computation of Boolean Circuits using Preprocessing (Damgård et al.)
- Multiparty Computation from Somewhat Homomorphic Encryption (Damgård et al.)
- Primitives and applications for multi-party computation (Toft)
- Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Ben-Or et al.)