Privacy Engineering

Part II

Naranker Dulay & Rami Khalil

n.dulay@imperial.ac.uk
https://www.doc.ic.ac.uk/~nd/peng

November 2020 (v1)

Privacy Engineering 70018

Materials

0. **COURSEWORK**

Draft Specification

The deadline for submission is Thursday 19th November 2020.

1. **SECURE MULTI-PARTY COMPUTATION**

Videos, Slides, Tutorial,

Secret Sharing and Garbled Circuits, in "Cryptography Made Simple", Nigel Smart, Chapter 22 (sections 1 to 3). Springer, 2016. Free download within Imperial domain. If the link doesn't work, download the e-book via a College library search. Also download the errata for the book. In particular, please note the corrections for page 447. These corrections are important if you are trying to understand the example and/or replicate some of the tables on the page.

A Pragmatic Introduction to Secure Multi-Party Computation, David Evans, Vladimir Kolesnikov and Mike Rosulek. A very good up-to-date overview. Recommended.

Millionaires' Problem, In "Protocols for secure

News

Welcome

This is the web page for Part II of the course.
Bookmark for future reference.

Provisional Teaching Schedule

```
Fri Nov  6 -- coursework spec available
Mon Nov  9 1000-1200 review secret sharing
Fri Nov 13 1000-1200 review garbled circuits

Mon Nov 16 1000-1200 tutorial MPC
Thu Nov 19 --coursework deadline
Fri Nov 20 1000-1200 review CoUS

Mon Nov 23 1000-1200 review ZK
Fri Nov 27 1000-1200 tutorial CoUS, ZK

Mon Dec 14 1400-1600 Exam
```

Pre-recorded videos will made available several days before review sessions.

Links

Notes

- Welcome to Part II of the course.
- Part II will include pre-recorded videos of varying lengths plus live sessions on the videos and for the tutorials.
- Part II will also include an individual assessed coursework.
- I hope you enjoy Part II of the course.

- RESOURCES
- Links to slides and other materials will be provided at https://www.doc.ic.ac.uk/~nd/peng. Bookmark it. Materials and CATE will include links to this page.
- The website will also include links to e-books, Papers, and web pages.
- The textbook "Cryptography Made Simple", Nigel Smart, Springer, 2016 will be used as the primary source for the material in Part II. You can download a pdf via a College Library search.
- Use Piazza to ask questions and start discussions.
- You are welcome to contact me by email if you have any questions or concerns about the course. You should get a reply from me by the end of the next day, other resend your email.

Topics

Naranker Dulay

Secure Multi-party Computation (MPC)
Shamir Secret Sharing, Garbled Circuits
Fully Homomorphic Encryption (FHE)

Coursework - *Implementation of the BGW MPC Protocol*

SQL queries over encrypted data - CryptDB
Multiuser keyword search on encrypted data - Proxy encryption
Privacy-preserving location sharing - Longitude

Rami Khalil

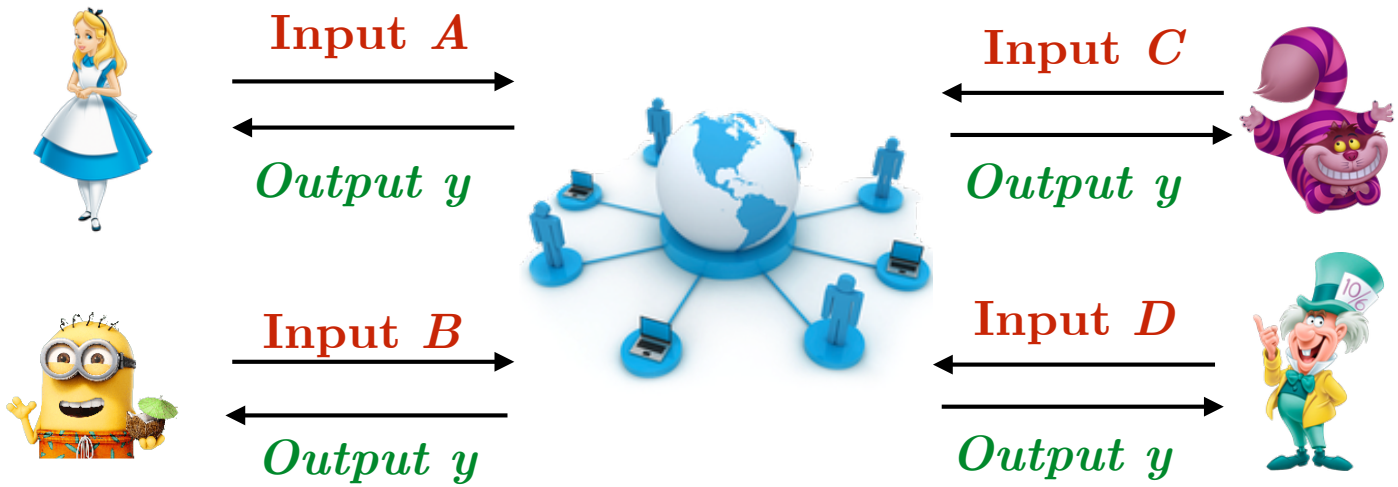
Zero Knowledge Proofs - Zcash

That's all class

2. Multi-party Computation (MPC) Introduction

Secure Multi-party Computation (MPC)

$y = f(A, B, C, D)$, for 4 parties



Given n parties, can we design a protocol which computes a public function f over inputs from each party such that the inputs remain private unless they would be revealed by the function anyway.

Applications: Auctions, E-Voting Machine Learning, Genomics, Satellite collision avoidance, Private set intersection, Network security monitoring, Spam filtering, many more ...

Example MPC Setups

1. Joint Processing

Garbled Circuit (GC) *or Secret Sharing* (SS)

2

Inputs

Computation

Outputs

GC
SS

Mutually distrusting parties

2. Outsourced Services

Secret Sharing (SS)

N

Inputs

SS

Computation

3+

M

Outputs

Untrusted servers

Mutually distrusting parties

3. Outsourced Processing

Homomorphic Encryption (HE)

1

Input

Output

HE

Computation

1×

Distrusting party

Untrusted server

Secret Sharing vs Garbled Circuits

Secret Sharing (SS)

- Express the function to be computed as an arithmetic circuit, e.g. addition and multiplication gates.
- Secret sharing is better suited for multi-party (3+) outsourced service setups.
- Secret sharing has high communication costs so we need keep the number of parties low.
- We'll look at the BGW secret sharing protocol

Garbled Circuits

- Express the function to be computed as an "encrypted" boolean circuit, e.g. with ANDs, ORs, XORs, etc
- Garbled circuits are more efficient than for 2-party joint processing setups. They get too complex for more parties.
- We will look Yao's Garbled Circuits protocol for 2-parties.

Adversarial Models

- ***Honest-but-curious*, Semi-Honest Passive**

  - Parties follow protocol, but are interested (i.e. curious) in breaking privacy of other parties.

  - Can collude with other parties.

  - Said to be a ***corrupt*** party.

- ***Malicious*, Byzantine, Active**

  - Parties can deviate from the protocol e.g. lie about inputs, quit protocol early.

  - Can also collude with other parties

The BGW protocol is information theoretically secure:

- For **honest-but-curious parties**, BGW can tolerant a minority **($<n/2$) of corrupt parties.**
- For **malicious parties,** BGW can tolerate: ($<n/3$) **corrupt parties.**
- **Information theoretically secure** - system cannot be broken even if adversary has unlimited computational power. Also known as Perfect security and Unconditional security.
- **Computationally secure** - adversary would require an unreasonably large amount of computational power to break the system.

Example 1: Privacy-Preserving Average Salary?

A group of people wish to calculate their average salary without anyone learning the salary of anyone else.

$E_X(M)$. Encrypt M with X's public key.
$D_X(M)$. Decrypt M with X's private key.

Recall that $D_X(E_X(M))=M$

Salaries of Alice, Bob, Carol, Dave are a, b, c, d

Alice generates a large secret random number r and then initiates the protocol:

Alice → Bob:	$E_{Bob}(a+r)$	Bob decrypts to get $a+r$
Bob → Carol:	$E_{Carol}(a+r+b)$	Carol decrypts to get $a+r+b$
Carol → Dave:	$E_{Dave}(a+r+b+c)$	Dave decrypts to get $a+r+b+c$
Dave → Alice:	$E_{Alice}(a+r+b+c+d)$	Alice decrypts to get $a+r+b+c+d$

Alice subtracts r to get total $a+b+c+d$, divides by 4 and informs everyone.

▶ Identify at least 3 situations where this protocol fails ◀

Example 2: Privacy-Preserving Voting

We want to compute the number of YES votes (yes=1, no=0) for 3 voters Alice, Bob and Carol whose votes are `V1`, `V2` and `V3` respectively.

- Alice generates 2 random numbers `V12` and `V13` in the range $0..p$ (p is a large prime agreed in advance). Alice computes `V11 = V1 - V12 - V13 (mod p)`
  i.e. V1=V11+V12+V13 (mod p) `Vxy` values are the **shares** of `Vx` (the **secret**)
- Alice sends (V11, V13) to Bob, and sends (V11, V12) to Carol
- Bob and Carol generate shares and distribute them in a similar fashion. We have
- `Alice=(V11, V12, V13)  Bob=(V11,  V13)  Carol=(V11, V12,  )`
  `(  V22, V23)  (V21, V22, V23)  (V21, V22,  )`
  `(  V32, V33)  (V31,  V33)  (V31, V32, V33)`

Example 2: Privacy-Preserving Voting

- `Alice=(V11, V12, V13)  Bob=(V11,  V13)  Carol=(V11, V12,  )`
  `(  V22, V23)  (V21, V22, V23)  (V21, V22,  )`
  `(  V32, V33)  (V31,  V33)  (V31, V32, V33)`

- Alice computes then broadcasts S2 (column 2) and S3 (column3)

  `S2 = V12 + V22 + V32 (mod p) and S3 = V13 + V23 + V33 (mod p)`

  Bob computes then broadcasts S1 (column 1) and S3 (column3)

  `S1 = V11 + V21 + V31 (mod p) and S3 = V13 + V23 + V33 (mod p)`

  Carol computes then broadcasts S1 (column 1) and S2 (column 2)

  `S1 = V11 + V21 + V31 (mod p) and S2 = V12 + V22 + V23 (mod p)`

- Everyone computes the final tally as follows `Votes = S1 + S2 + S3 (mod p)`

Example 3: Yao's Millionaires' problem

Two millionaires Alice and Bob wish to know who is the richer of the two without disclosing their wealth.

Let's say Alice has a=£4M, Bob has b=£3M, a and b are integers in the range 1..6

Alice a : {1..6} = 4 — Bob b : {1..6} = 3

Public Key pubA = (e, n) >>>>>>>>>>>> r = large random number

Private Key privA = (d, n) (r is secret to Bob)

<<<<< C <<<<< $C = E_{pubA}(r) - b = r^e - b$

$Y[1] = D_{privA}(C + 1) = D_{privA}(EpubA(r) - b + 1)$

$Y[2] = D_{privA}(C + 2) = D_{privA}(EpubA(r) - b + 2)$

$Y[3] = D_{privA}(C + 3) = DprivA(EpubA(r) - b + 3) = DprivA(EpubA(r)) = r$

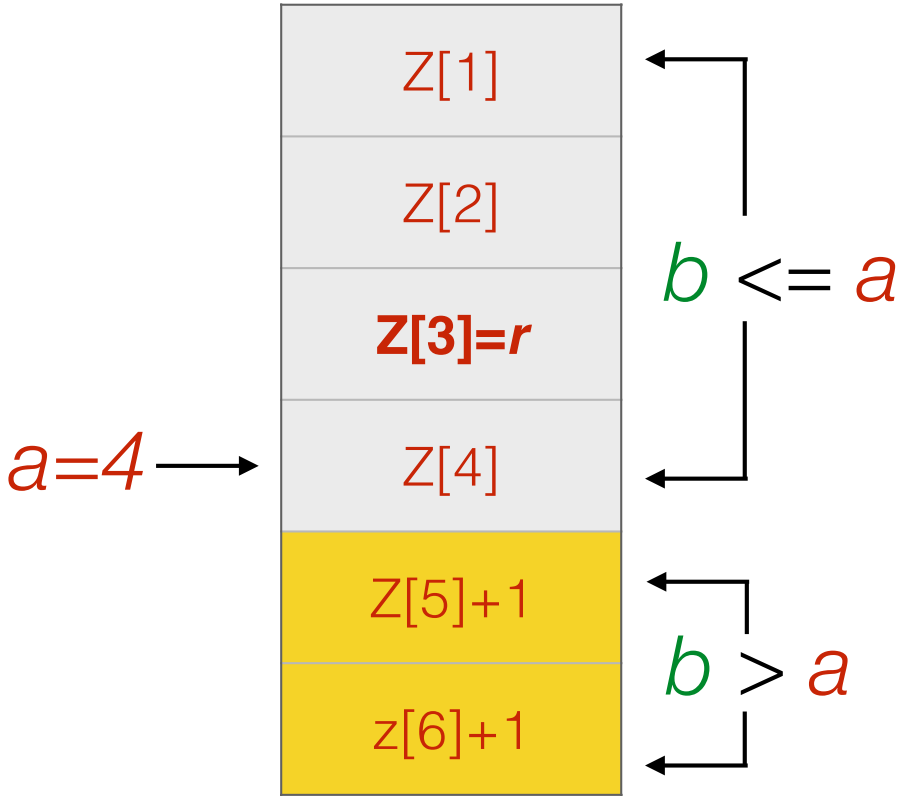
$Y[4] = D_{privA}(C + 4) = DprivA(EpubA(r) - b + 4)$

$Y[5] = D_{privA}(C + 5) = DprivA(EpubA(r) - b + 5)$

$Y[6] = D_{privA}(C + 6) = DprivA(EpubA(r) - b + 6)$

Alice Bob

```
p    = large random prime
Z[1] = Y[1] mod p
Z[2] = Y[2] mod p
Z[3] = Y[3] mod p = r mod p
Z[4] = Y[4] mod p
Z[5] = Y[5] mod p
Z[6] = Y[6] mod p
```



Send p and | Z[1], Z[2], Z[3] = r mod p, Z[4], | Z[5]+1, Z[6]+1

if Z[3]=r mod p **then** *b*<=*a* **else** *b*>*a*

That's all class

3. Preliminary - Lagrange Interpolation

Lagrange Interpolation 1

- A polynomial $P(x)$ of degree T can be uniquely determined by a set of $T+1$ (or more) distinct points on the polynomial curve $P(x)$.
- For T points there are an infinite number of polynomials of degree T that pass through the T points.
- Lets see this in action and why its of interest for MPCs ...

A second result that we'll use:

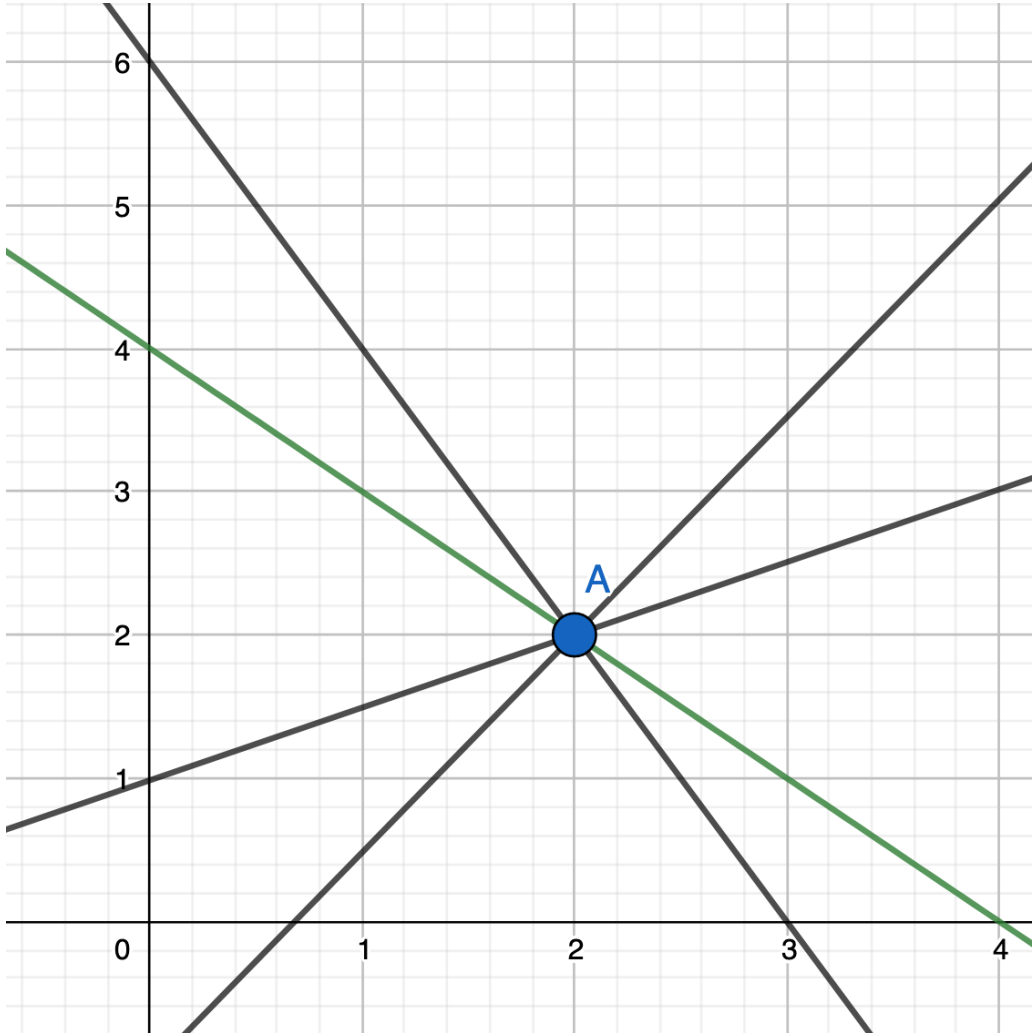
- For any polynominal $P(x)$ of degree upto at most $N-1$ there exist coefficients $r = (r_1, .., r_N)$ (called the recombination vector) such that:

$$P(0) = \sum_{i=1}^{N} r_i \cdot P(i)$$

(see Recombination vector slide later)

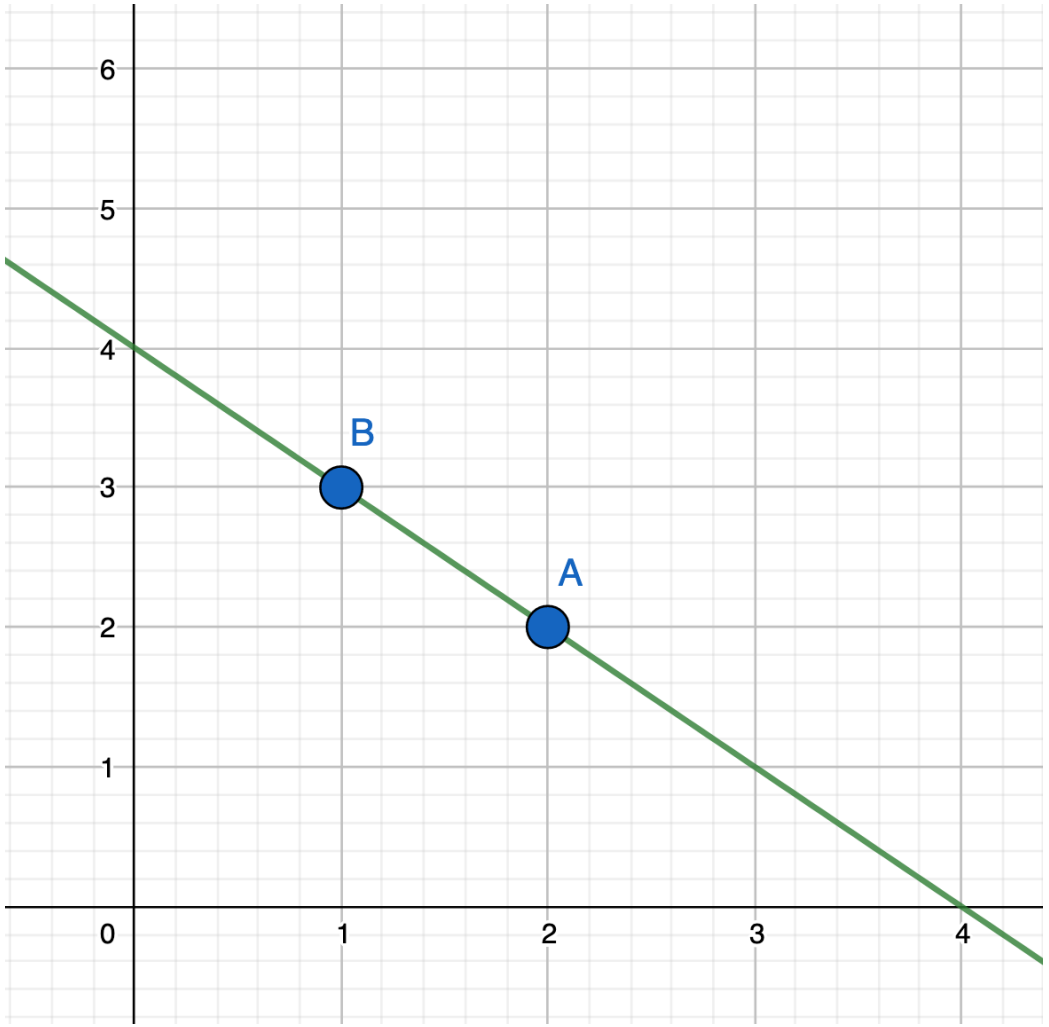
For 1 POINT

- There are an infinite numbers of polynomials $P(x)$ of degree 1 (i.e. straight lines) that pass through the point.
- There are also infinite number of values of $P(0)$ (e.g. including the values 1, 4, and 6 in the plot on the right)



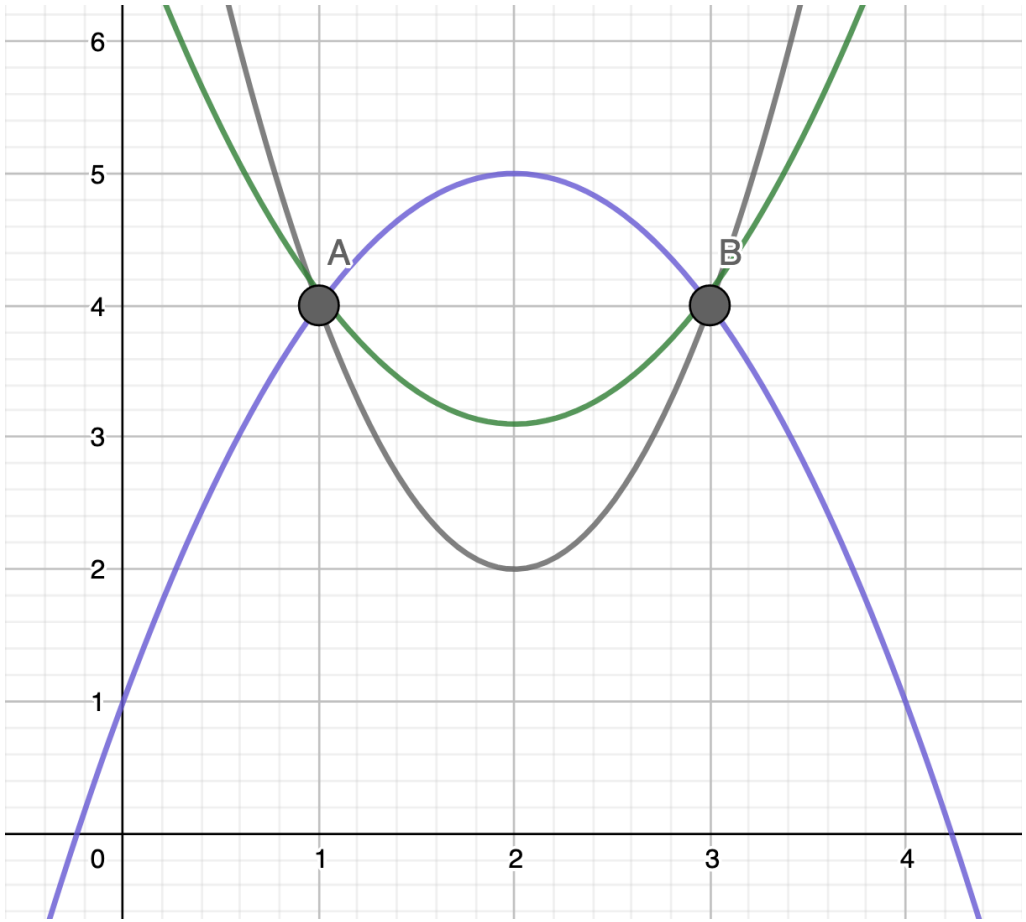
For 2 POINTS

- There is only 1 polynomial $P(x)$ of degree 1 that passes through 2 points.
- There is also only 1 value for $P(0)$ e.g. the value 4 in the plot on the right



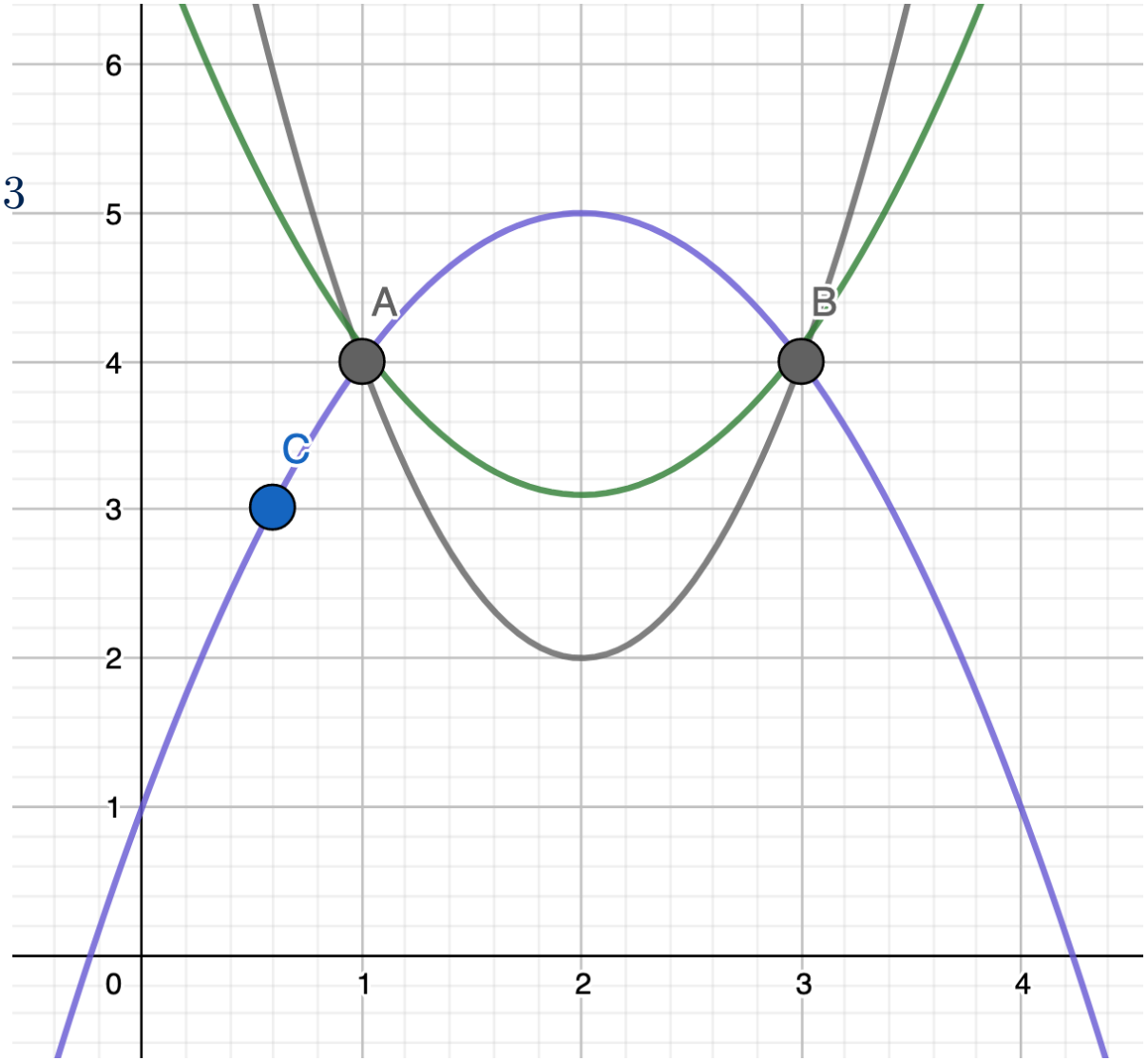
For 2 POINTS

- There are an infinite numbers of polynomials $P(x)$ of degree 2 (parabola) that pass through the 2 points.
- There are also infinite number of values of $P(0)$ (including the value 1 in the plot on the right)



For 2 POINTS

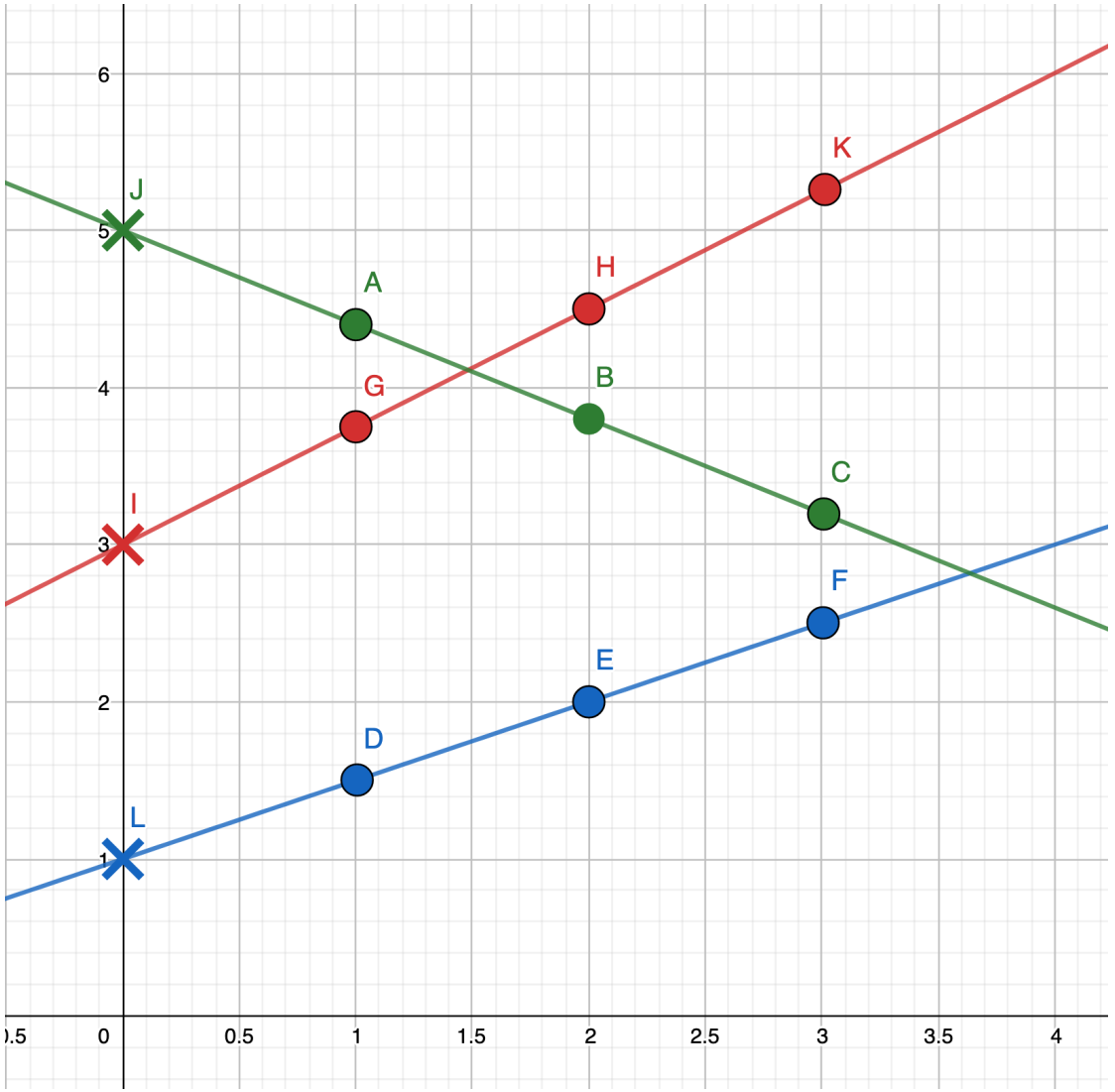
- There is only 1 polynomial $P(x)$ of degree 2 that passes through 3 points.
- There is also only 1 value for $P(0)$ e.g. the value 1 in the plot on the right



- This extends to polynomials of higher degree.
- Yes, but how could we use?

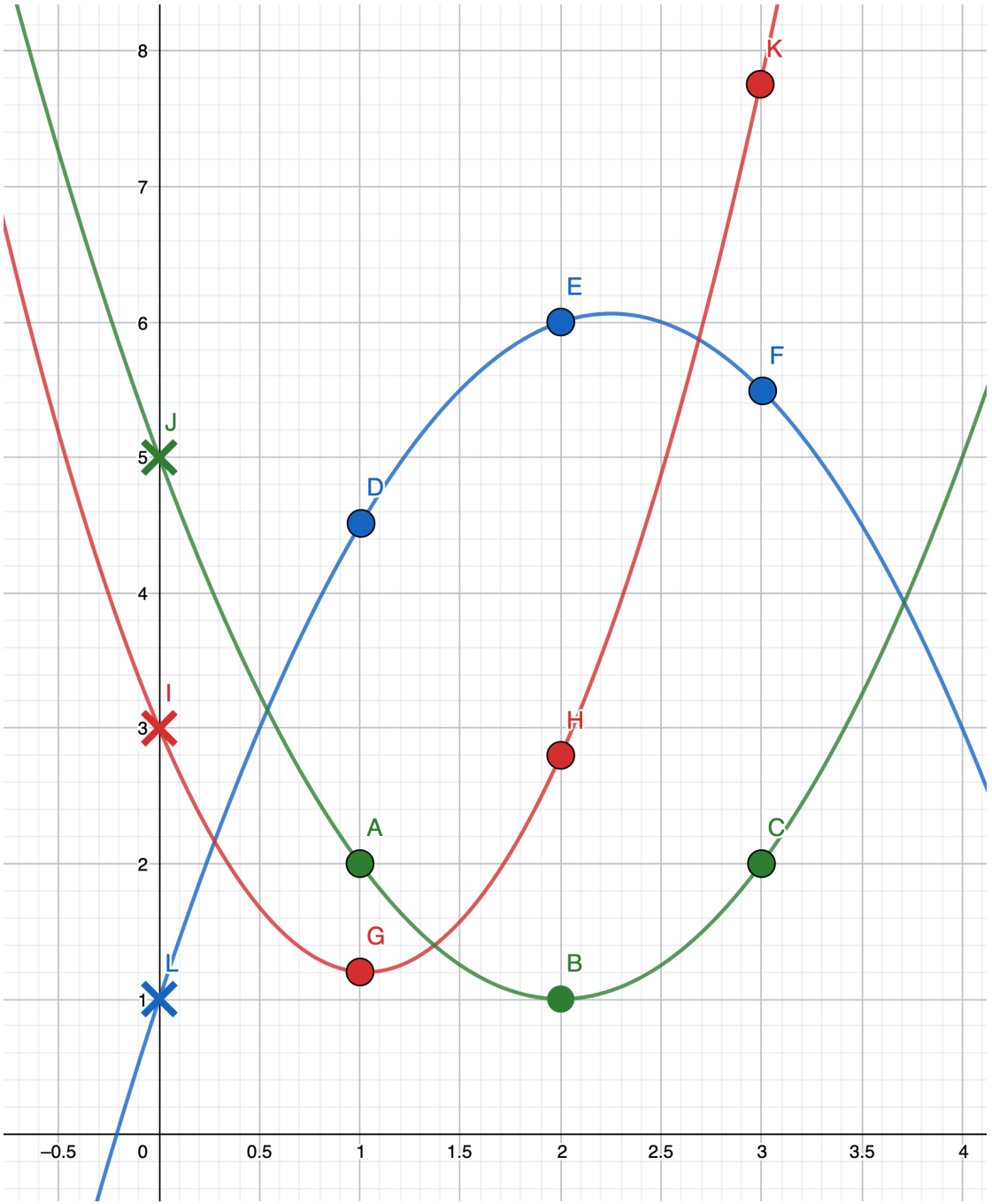
Alice wants to split and share a secret number, for example, J=5 with N other parties, such that J can only be reconstructed if any two of the N parties combine their shares.

To do this Alices only needs to generate a random polynomial of degree 1 and then send different values of $P(x)$ to each party. For example $A = P(1)$ to party 1, $B = P(2)$ to party 2, $C = P(3)$ to party 3. Now any 2 of the parties can reconstruct the polynomial $P(x)$ and then compute the secret number $J = P(0)$

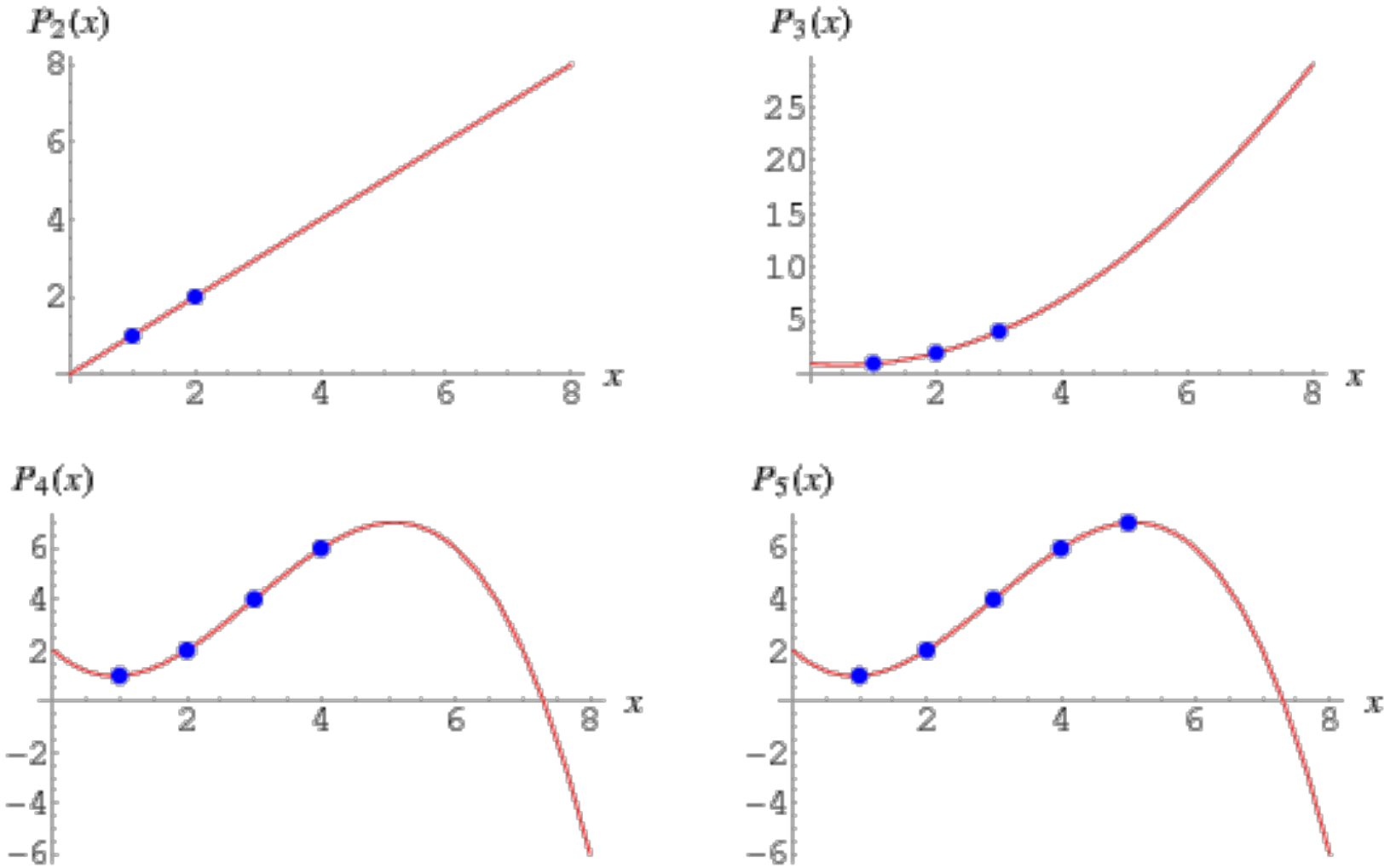


This works for more parties and higher degrees also. For example, to require 3 parties to construct a secret, Alice would generate a random polynomial of degree 2.

This brings to the question - how can polynomial $P(x)$ of degree T be uniquely determined by a set of $T+1$ (or more) distinct points on the polynomial curve $P(x)$.



We can use Lagrange Interpolation to construct a unique polynomial of degree T from T+1 points, e.g $P_2(x)$ below is of degree 1 while $P_5(x)$ is of degree 4.



Wolfram MathWorld

Lagrange Interpolation 2

- For T+1 points the unique Lagrange Interpolation Polynomial of degree T is

$$P(x) = \sum_{i=1}^{T+1} \delta_i(x) \cdot P(i) \text{ where } \delta_i(x) = \prod_{j=1, j\neq i}^{T+1} \frac{x - x_j}{x_i - x_j}$$

  $\delta_i(x)$ are called Lagrange basis polynomials. This works, because for each term of $\delta_i(x)$, we have $\delta_i(x_j) = 1$ when $j = i$ and $\delta_i(x_j) = 0$ when $j \neq i$

- For Shamir secret sharing (see later) we will use x values of 1, 2, .., for parties 1,2, 3, ... so this can be further simplified to

$$\delta_i(x) = \prod_{j=1, j\neq i}^{T+1} \frac{x - j}{i - j} \text{ or } \delta_i(x) = \prod_{j=1, j\neq i}^{T+1} \frac{j - x}{j - i}$$

- For $P(0)$ we can simplify $P(x)$ to

$$P(0) = \sum_{i=1}^{T+1} \delta_i(x) \cdot P(i) \text{ where } \delta_i(x) = \prod_{j=1, j\neq i}^{T} \frac{x_j}{x_j - x_i}.$$

Example

- **Example**. Construct the unique polynomial that fits the following points (3,1), (4, 2), (5,4)?

$$\delta_1(x) = \frac{(4-x)(5-x)}{(4-3)(5-3)} = \frac{20-9x+x^2}{2} = 10 - 4.5x + 0.5x^2$$

$$\delta_2(x) = \frac{(3-x)(5-x)}{(3-4)(5-4)} = \frac{15-8x+x^2}{-1} = -15 + 8x - x^2$$

$$\delta_3(x) = \frac{(3-x)(4-x)}{(3-5)(4-5)} = \frac{12-7x+x^2}{2} = 6 - 3.5x + 0.5x^2$$

$$P(x) = 1(10 - 4.5x + 0.5x^2) + 2(-15 + 8x - x^2) + 4(6 - 3.5x + 0.5x^2)$$

$$P(x) = 4 - 2.5x + 0.5x^2 \qquad P(0) = 4 = Secret$$

Recombination vector = $[\delta_1(0),\ \delta_2(0),\ \delta_3(0) = [10,\ -15,\ 6]$

- *Note:* For the BGW MPC protocol we will use modular arithmetic over a prime number p.

  Redo the example above using modular arithmetic for p=5. Chapter 1 of Smart's book provides an introduction to modular arithmetic.

Lagrange Interpolation 3

Recombination vector

- For Lagrange interpolation there exists an easily computable **recombination vector** $r = (r_1, \ldots, r_N)$ such that

$$P(0) = \sum_{i=1}^{N} r_i \cdot P(i)$$

for *all* polynomials $P(x)$ of degree upto at most $N-1$, namely

$$r_i = \delta_i(0)$$

Note $\delta_i(x)$ does not depend on $P(x)$ so neither does $\delta_i(0)$.

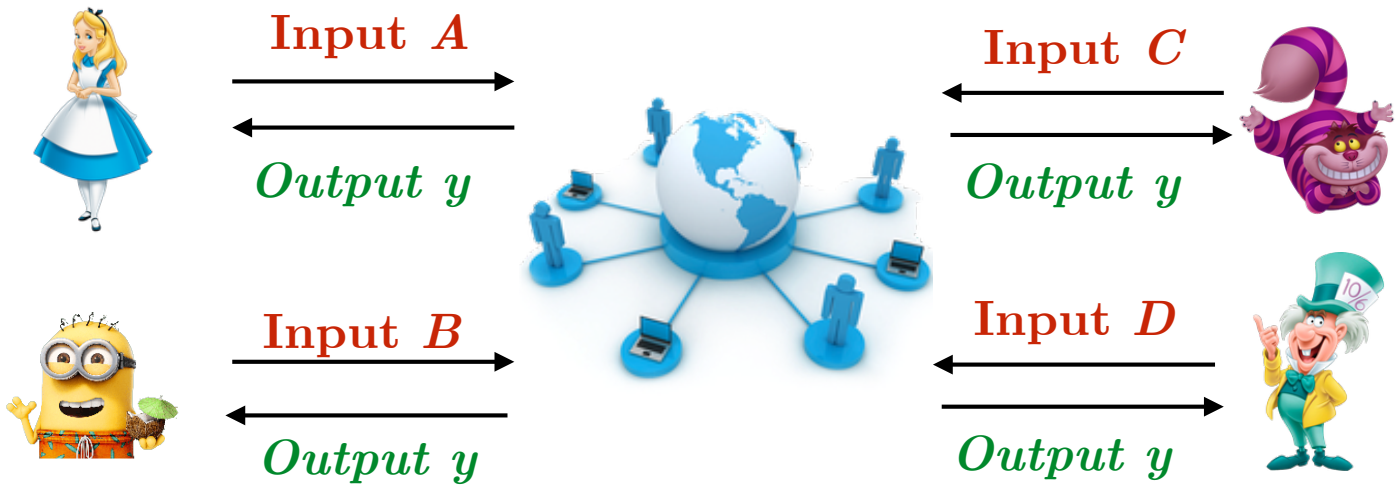
Hence the *same* unique recombination vector works for all polynomials $P(x)$ of degree up to at most $N-1$. This means anyone (every party) can compute it - it is public knowledge.

That's all class

4. BGW (Ben-Or, Goldwasser, Widgerson) MPC Protocol

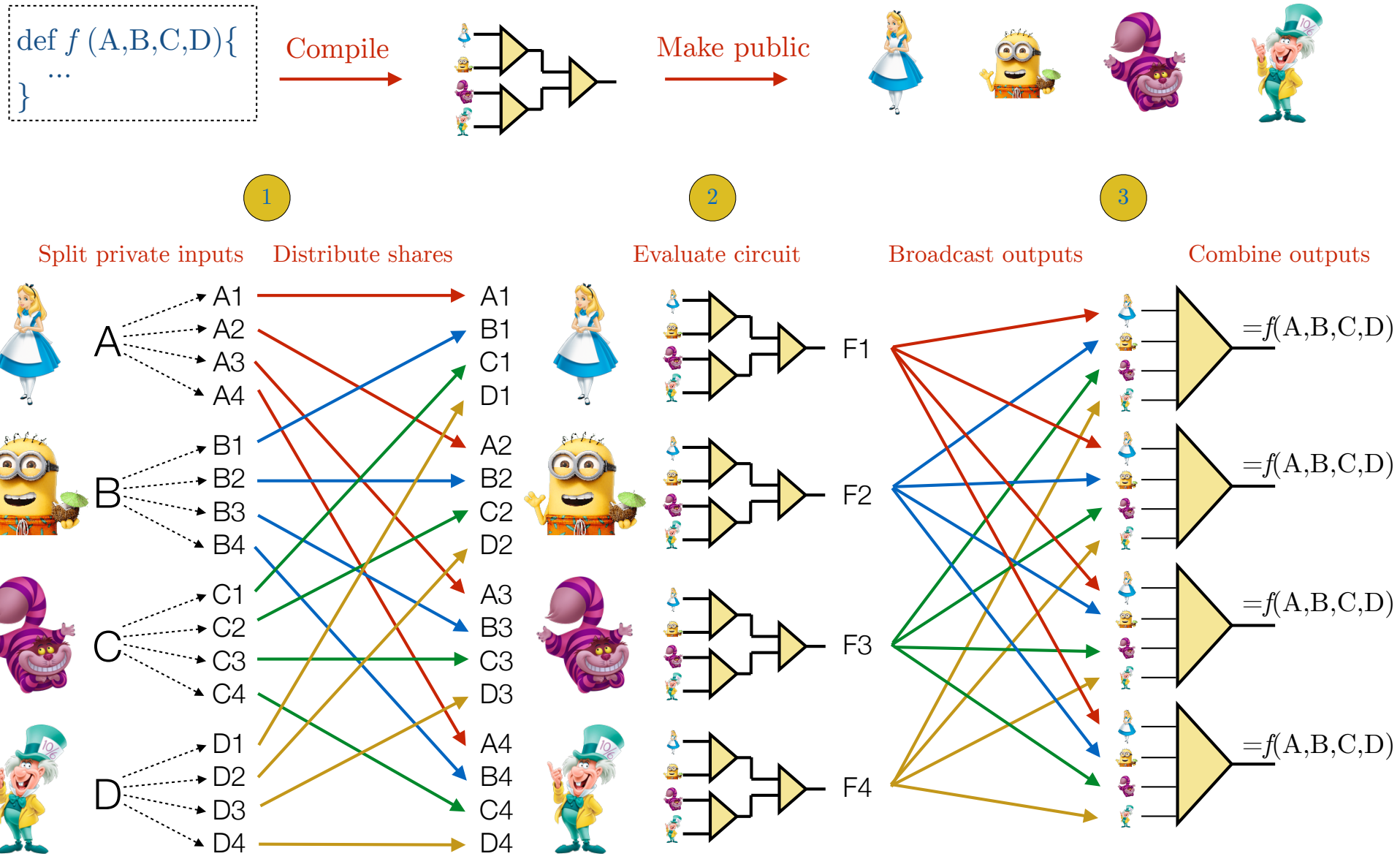
Secure Multi-party Computation (MPC)

$y = f(A, B, C, D)$, for 4 parties



Given n parties, can we design a protocol which computes a public function f over inputs from each party such that the inputs remain private unless they would be revealed by the function anyway.

BGW protocol using Secret Sharing - Overview



Phase 1: (T+1, N) Shamir Secret Sharing

In BGW the value of each input wire (a party's secret) and each gate output wire will be "hidden" using a random polynomial of degree T by sharing the value among N parties. For each input wire value we apply Shamir's ($T+1$ *shares from* N) threshold secret sharing scheme:

Given a secret S (the private value on the party's input wire) **each party** i performs:

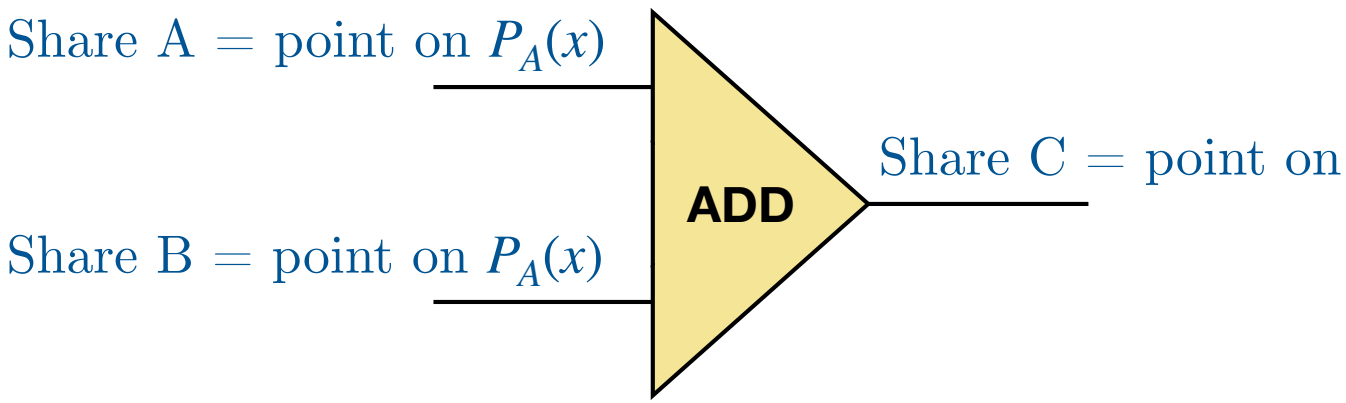
- Selects random coefficients $a_1, a_2, \ldots, a_T$
- Defines a $P_i(x) = S + a_1 x + a_2 x^2 + a_3 x^3 \ldots + a_T x^T$
- Sends the share $S_j = P_i(j)$ to Party j. The N shares are known as a **SHARING** of S.
- Recall that given $T+1$ shares we can combine the shares for a wire to reconstruct a new polynomial P for the wire using *Lagrange Interpolation* and then recover the secret S by computing $P(0) = S$.

- Perfect Privacy

  The polynomial $P(x)$ and hence the secret S cannot be reconstructed with T or less shares. Further nothing about the secret is revealed even if the adversary has **unbounded computational** resources (i.e. Shamir secret sharing is **information-theoretically secure)**

- $T+1$ shares (i.e. parties) are needed to reconstruct the secret.
- For example, given 12 parties with shares computed from a random polynomial of degree 6, then any 7 parties can recover the secret using their 7 shares, 6 or parties cannot recover the secret or learn anything about the shares. The scheme is a (7, 12) threshold sharing scheme.

Phase 2: ADDition Gate



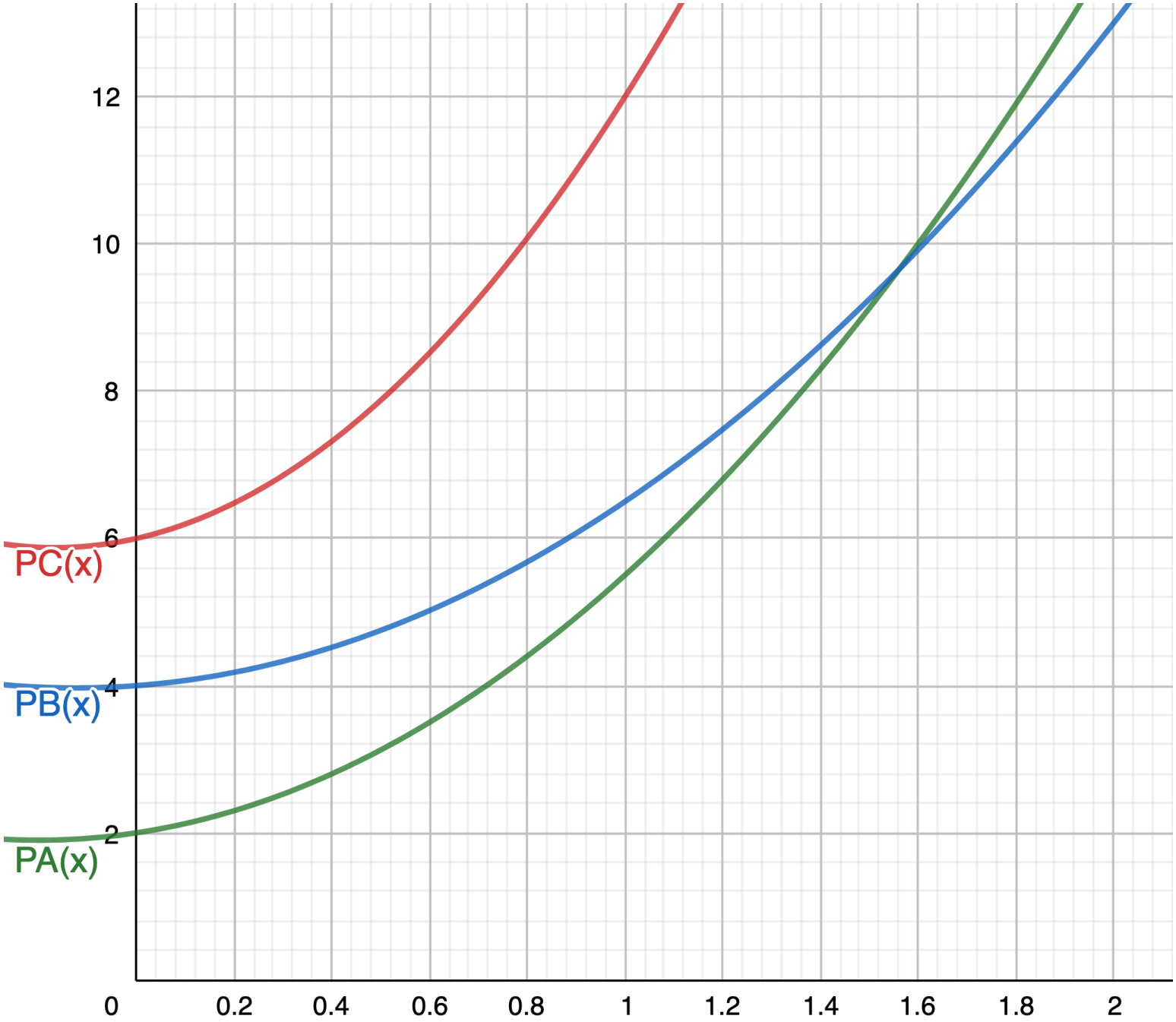
- Example:

  If $P_A(x) = 2 + 1x + 2.5x^2$

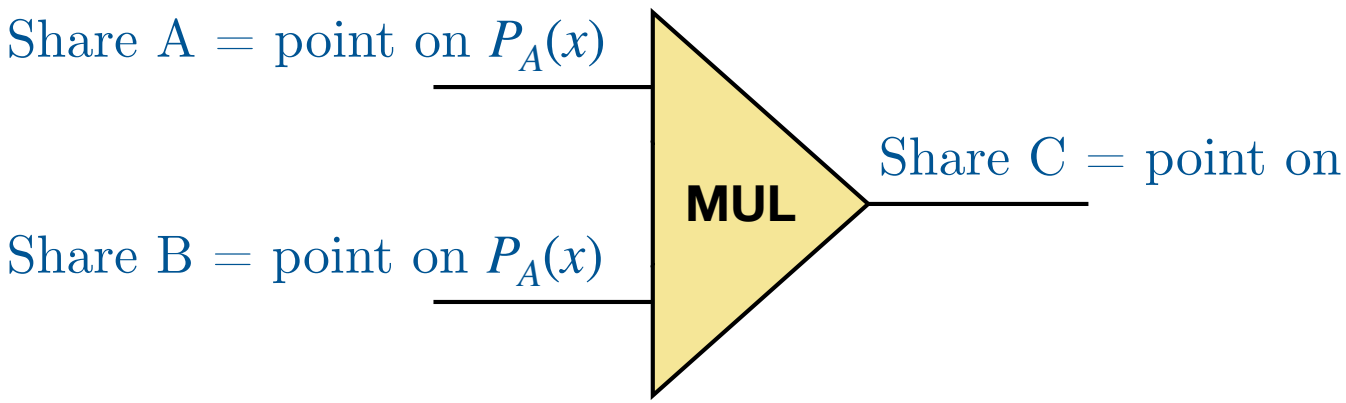
  and $P_B(x) = 4 + 0.5x + 2x^2$

  then $P_C(x) = 6 + 1.5x + 4.5x^2$

- We can see this graphically also:


12
10
8
6
4
2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
PC(x)
PB(x)
PA(x)

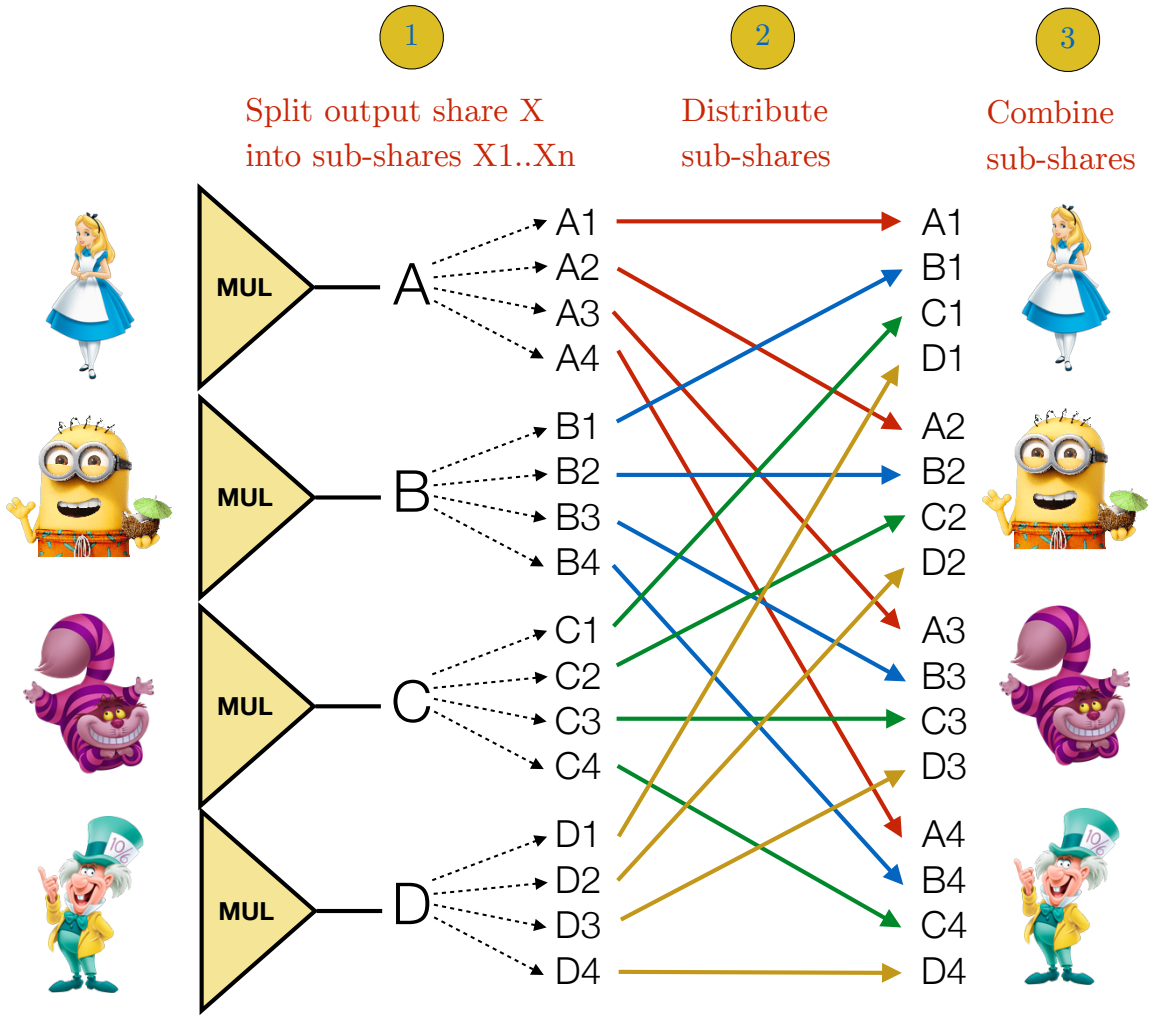
Phase 2a: MULtiplication Gate



Multiplication is more complex. Consider, if $P_A(x) = 2 + 4x$ and $P_B(x) = 3 + 2x$ then $P_C(x) = 6 + 16x + 8x^2$. Here, the output share, 6, is correct. However the degree of the polynomial, 2, is too high. In this case we need the degree to be 1.

More generally the degree of the output polynomial for multiplication will be $2T$ which is too high. We need the degree of the output polynomial $P_C(x)$ to be T.

Phase 2b: MULiplication gate (degree reduction)



Phase 2b: MULtiplication gate (degree reduction)

- How can we both reduce the degree of $P_C(x)$ and randomise its coefficients of $P_C(x)$, while keeping the constant term?
- We know that any point on a polynomial of degree $2T$ can expressed as a linear combination of $2T+1$ points on the polynomial given its (public) recombination vector.
- So why not apply Shamir secret sharing and Lagrange interpolation to the share?

Degree reduction protocol

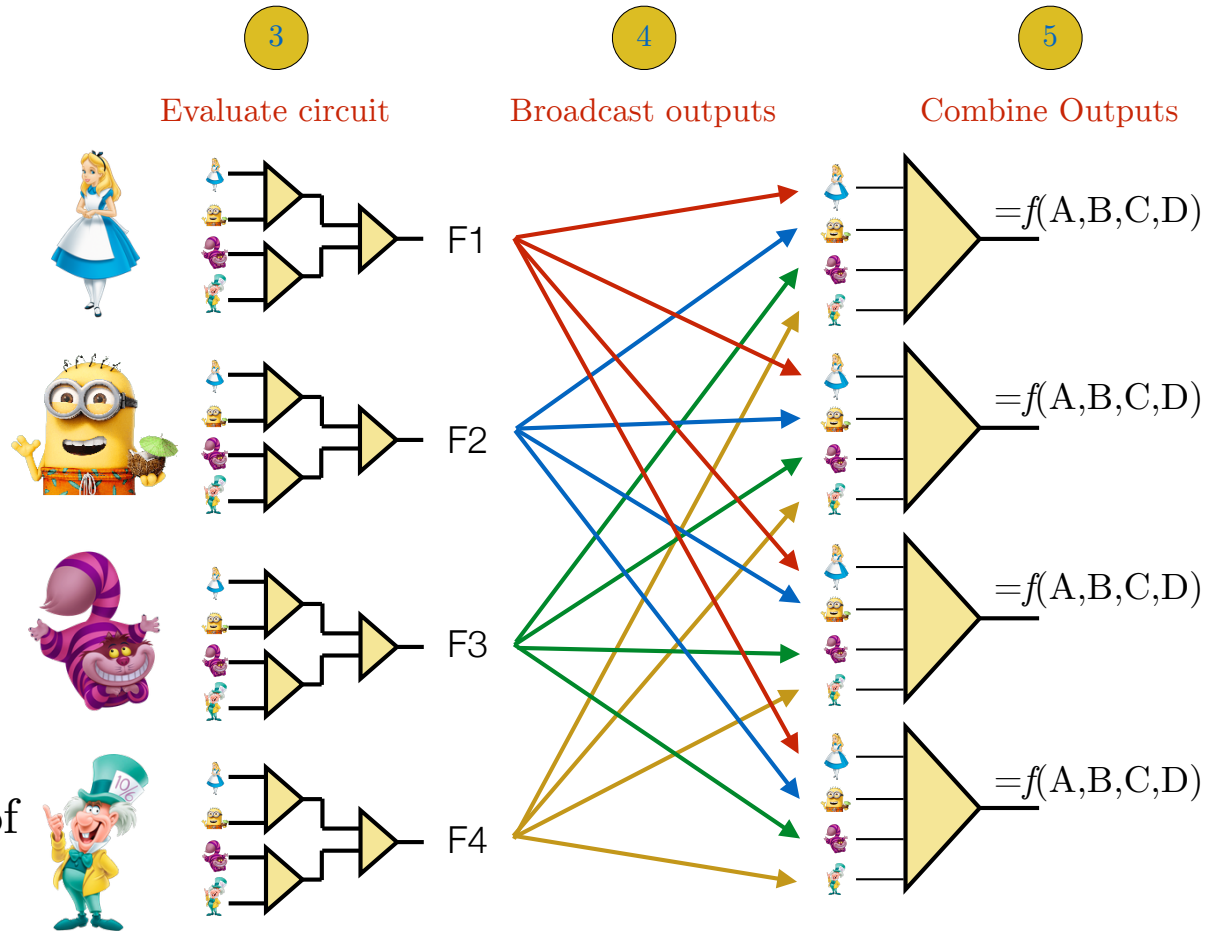
Given a share S (the value of the MULtiply gate's output wire) **each party** i:

- Selects random coefficients $a_1, a_2, \ldots, a_T$
- Defines $Q_i(x) = S + a_1 x + a_2 x^2 + a_3 x^3 \ldots + a_T x^T$
- And sends the sub-share $S_j = P(j)$ to Party j.

- After receiving $T+1$ sub-shares each party can combine the sub-shares using a recombination vector to construct a new reduced polynomial Q of degree T.
- Equivalently each party can use *Lagrange Interpolation* to construct Q and then recover the share using $Q(0) = S$.
- The T+1 shares are essentially points on the reduced degree random polynomial Q.
- Note: in contrast to ADD gates which involve no communication between parties, N^2 messages are sent for each MUL gate.

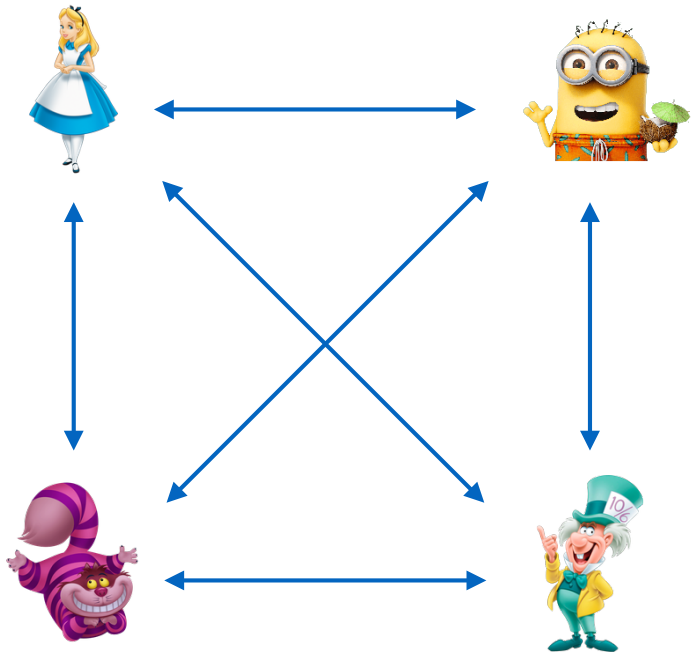
Phase 3: Final result

- In the final step each party sends its share of the circuit's output wire to every other party.
- The parties can then use Lagrange Interpolation to construct the polynomial $P(x)$ corresponding to the shares received and recover the circuit's output value using $P(0) = result$
- Note: $O(n^2)$ messages are needed for each MUL gate. The number of rounds is linear in the depth of the circuit counting the MUL gates only.



Coursework Overview

- For the coursework, you'll be implement the BGW protocol!
- Each party will run as a Linux(or Mac) process and use TCP to communicate with other parties.
- Python code to create processes and to do interprocess communication (party-to-party) will be provided.
- Two sample circuits will be provided.
- You will need to add code that each party executes to run the BGW protocol. You will also need to submit an original circuit of your own.
- In your code, each party will need to (1) share its private value with other parties using Shamir secret sharing, (2) evaluate the circuit gate-by-gate carrying out the correct procedure for ADD gates and MUL gates, (3) send, receive and combine the circuits output shares to produce the final result.
- The involves managing the inputs and outputs of gates and writing code for Lagrange interpolation. All calculations will need to done using modular arithmetic



- A typical Python solution is about 500 lines (300 provided, 200 that you write).
- Although relatively straightforward, do start the coursework early.
- There is no tutorial on this topic. You'll learn a lot more doing this coursework.
- Have fun and good luck!

That's all class