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

Notes

- In Part II we will focus on the use of cryptographic techniques that are privacy-preserving.
- Cryptography is primarily associated with security; properties such as *confidentiality*, *integrity*, and *authentication*. Security researchers have, however, always considered privacy to be a sub-branch and there are many interesting ideas and results that "privacy engineers" should be aware of. Part II will provide an introduction to a few of these.
- Cryptography requires a background in mathematics, topics like algebraic structures (groups, rings, fields, lattices etc) and number theory. Many of you will not have this background and there is insufficient time to teach it. So we will simplify the maths and omit formalisation and mathematical proofs. Term 2 includes a module on Cryptography for those interested in a mathematical approach.
- (1) Secure Multi-Party Computation (MPC) are distributed protocols where 2 or more parties jointly compute a function but keep their inputs private. We'll briefly look at FHE, One-time programs and One-time memory.
- (2) Computing on Untrusted Servers - in particular we will look at 3 case-studies. CryptDB, for privacy-preserving SQL queries, Proxy encryption to support multi-user keyword search, and Longitude for sharing location with friends.
- (3) Zero Knowledge Proofs and their use in Zcash, a cryptocurrency that aims to provides better anonymity.

These are not the only topics that privacy researchers are interested in. Below are a few others:

- Privacy techniques for specific platforms such as mobile phones, web, IOT, blockchains etc..
- Anonymous communication, censorship resistance
- Hardware support for privacy enforcement
- Privacy policies, data protection, privacy metrics
- Privacy attacks using adversarial machine learning (and countermeasures!)
- Usable privacy
- ...

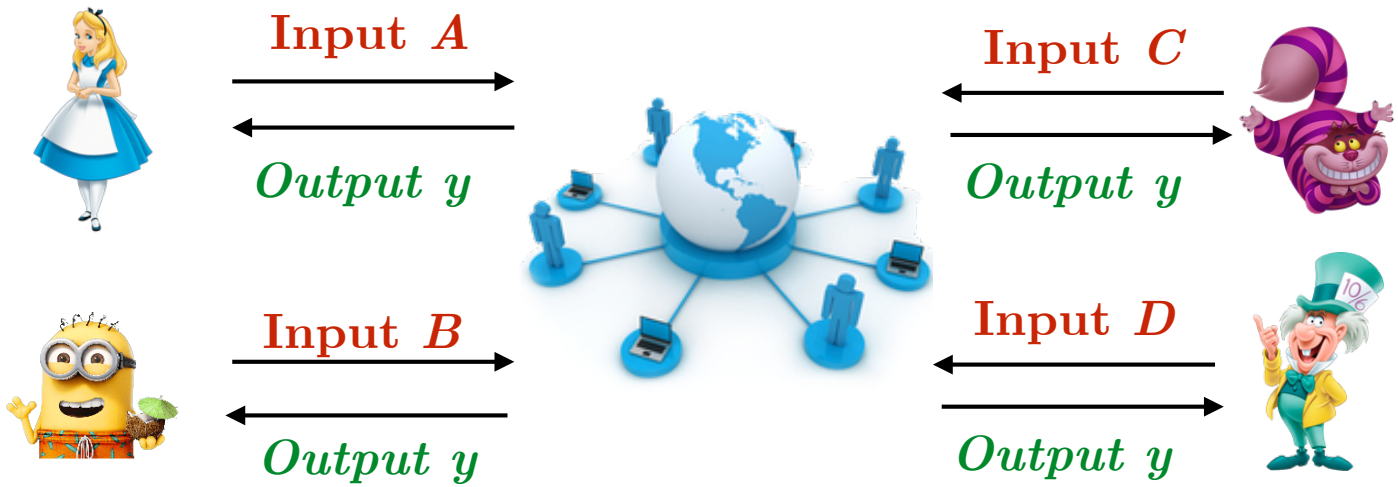
If you're interested in these or other privacy / security topics for a project, then you're welcome to contact me.

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 ...

Notes

- Secure multi-party computation (MPC) is the the first cryptographic approach we'll look at.
- MPC is a *privacy-preserving distributed protocol* where 2 or more parties wish to *jointly-compute* a function but keep their inputs to the function private.
- The function computed is typically the same for each party. That is all parties receive the same output. There are techniques to compute separate functions for each party.
- The formal analysis of an MPC protocol is based on showing that the MPC protocol is equivalent to a protocol that uses a trusted 3rd party where each party securely sends its input to a trusted 3rd party that computes the desired function and then sends the result back to each party.

  This implies that the trusted 3rd party will have access to the inputs of each party. But what if the trusted 3rd party is hacked or ordered by a government to hand-over the data?
- Why use MPC?
- MPC is typically considered when parties

  (i) don't trust each other nor do they
  (ii) don't trust a 3rd party.

  A few examples are listed on the right.

- See Evans et. al for a nice overview of some these
- Private Auctions without an auctioneer. N bidders make offers. Determine highest bidder without revealing offers. Blind auctions
- E-voting without tallying servers.
- Private Set Intersection. N data providers with private data wish to compute an intersection of their common data.
- Genome edit distance. Is there a "genetic match" for Alice's DNA sequence in Bob's database DNA sequences?
- Satellite collisions. Satellite operators consider the trajectories of their satellites a secret, so why not use MPC to avoid collisions?
- Poker without a dealer.
- Many more, ..., Any trusted service that you use is potentially an opportunity for MPC.
- Companies working in the space include. Partisia, Unbound Tech, Cybernetica, Galios, Seplor ...
- *Can you think of some interesting examples of your own?*

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

Notes

- There are many setups that can be used for MPC. We mention three, allow the last FHE is not really MPC.
- In Joint processing, 2 or more parties run the MPC protocol directly. They somehow share they inputs, do the computation and receive the results. The two main MPC approaches that we will look at: Garbled circuits (GC) and secret sharing (SS) can be used for this.
- With Outsourced services, the MPC protocol is used to perform the private computation on a small cluster of untrusted servers. These collect private inputs from clients, jointly perform the computation and then sent the results back, possibly to different clients. The Danish Sugar Beet auction was an early example of a commercial service that did this. Secret sharing MPC is typically used for this.
- The third setup uses Homomorphic Encryption (HE) NOT MPC, and can be used when a client wishes to outsource the storage and processing of its private data to an untrusted server. The input data and output results remain private to the client.

  Partial HE schemes support a limited number of operations but have efficient implementations.

  Fully HE (FHE) schemes support a Turing-complete set of operations but are very very inefficient.

- We'll look at three illustrative examples: (1) computing the average salary of N parties, (2) tallying the votes of N parties, (3) determining who is the richer of two millionaires.
- Then we'll look at Secret Sharing and Garbled Circuits which are more general approaches. We'll then very briefly look at Fully Homomorphic Encryption.
- General approaches can handle quite complex functions on moderately large datasets, for example, secure regression analysis on 100K+ observations and 100+ variables.
- MPC is slowing starting to become more mainstream with a number of general-purpose compilers and libraries for non-experts (see paper below for further details) Performance is also improving each year. Survey Paper: SoK: General Purpose Compilers for Secure Multi-Party Computation 10.1109/SP.2019.00028

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.

Notes

- There are two main approaches for general privacy-preserving multi-party computation, garbled circuits and secret sharing.
- Garbled Circuits (GCs) represent functions as garbled (encrypted) boolean circuits while Secret Sharing (SS) uses arithmetic circuits.
- GCs assume that underlying encryption scheme is secure. For SS any secret sharing scheme can theoretically be used, but the most efficient protocols have used (somewhat) "homomorphic" encryption (see later)
- Of course it is feasible to map arithmetic circuits to boolean circuits and vice-versa. The circuits are topologically sorted for evaluation.
- Recent advances are developing hybrid solutions that use combine Secret Sharing MPC and FHE to provide better security and better performance.

- See Evans for an overview of the state-of-the-art in this area.

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.

Notes

- Cryptographic protocols have many interesting assumptions (e.g. types of adversary, bounds on adversarial power) and theoretical results (e.g. number of corrupt parties that can be tolerated).
- These typically require a background in information theory, and algorithm complexity, etc to fully understand. We'll occasionally state some results but will omit formalisation and mathematic proofs.
- We'll only look at the semi-honest cases (see next slide). In real-world systems we should use MPC protocols that are secure against malicious adversaries.

Adversarial Power

- Computationally unbounded with no limits on the computational power of the adversary, or
- Computationally bounded

Performance:

Algorithm/protocol complexity and real-world performance are also important. For example:

- Computation (time and space)
- Communication costs and latency

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 $D_X(E_X(M))=M$

If B and D collude, then they can know C's value

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** ◀

Notes

Assumptions

- All parties are available and know the topology.
- Public keys have been securely distributed / known / available.
- **Kerchoff's principle** applies -- that the algorithm (protocol) is public knowledge. Security is based on cryptographic values that are kept secret, for example a decryption key. *Security through obscurity* (e.g. propriety non-public algorithms) is not an approach advocated by cryptographers.
- The time and space complexity of the protocol has been ignored. These are important in practice as is communication complexity (e.g. the number and size of messages exchanged) and speed of transmission.

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)`

V11 + V12 + V13 = Votes for Alice; V21 + V22 + V23 = Votes for Bob, V31 + V32 + V33 = Votes for Carol
The total votes are calculated vertically by receiving broadcast from multiparties

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)`

Notes

Why does this work?

Essentially, the protocol carries out the following transformation

$$\sum_{X=1}^{3} v_X \bmod p = \sum_{X=1}^{3}\sum_{Y=1}^{3} v_{XY} \bmod p = \sum_{Y=1}^{3}\sum_{X=1}^{3} v_{XY} \bmod p = \sum_{Y=1}^{3} s_Y = \mathit{Votes}$$

- Note that this will work not just for boolean values but for any integers. The protocol is thus a multiparty addition protocol - so we could use it to compute the average salary also.

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)

Notes

- Andrew Yao's seminal paper presented the first preserving-solution to the millionaire's problem. The solution is not particularly useful. It's celebrated because the paper it was presented in, also hinted at a general MPC approach to evaluating functions that came to be known as garbled circuits. We'll look at garbled circuits later.
- Alice's value is a, Bob's value is b.
- Bob chooses a large random number r (Bob's secret) and encrypts it with Alice's public key EpubA(r)
- Bob sends C=EpubA(r) - b to Alice.
  C is an encryption of Bob's secret.
- Alice decrypts 6 values
  Y[k]=DprivA(C+k) for k=1..6 i.e. for each million,
  Note all 6 elements are random values to Alice. She does not know that Y[3] corresponds to r

RSA in a nutshell

- **Plaintext** m

  Large primes p and q

  Composite $n = p \times q$ (n is public)

  Find e and d such that

  $$e \times d = 1 \bmod (p - 1)(q - 1)$$

  Encryption key e

  Decryption key d

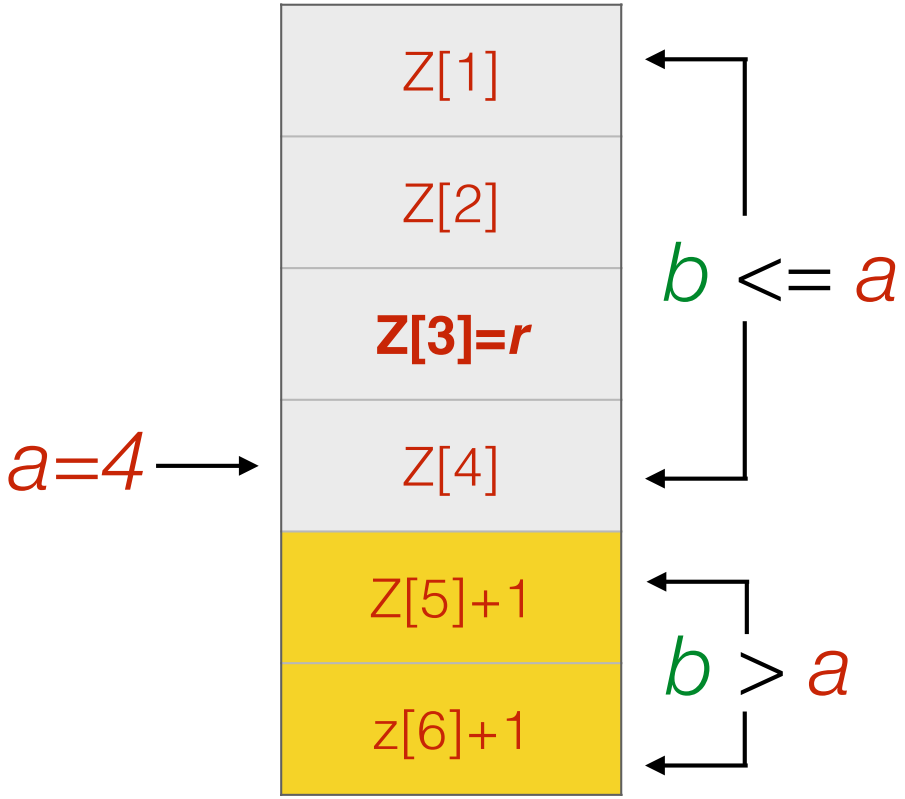
  → **Public Key** $= (e, n)$

  → **Private Key** $= (d, n)$

- **Encryption** $= m^e \pmod n$
  $= c$

- **Decryption** $= c^d \pmod n$
  $= (m^e)^d \pmod n$
  $= m^{ed} \pmod n$
  $= m \pmod n$

```
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
```



Bob can calculate to see if his value Z[3] is calculated correctly = r mod p.
If not, that means Alice is poorer than Bob. If true, than Alice is richer or same rich as Bob.

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*

Notes

- Alice then generates a large random prime p (but smaller than r) and reduces the decrypted Y[k] values mod p

  $$Z[k] = Y[k] \bmod p, \quad \text{for } k=1..6$$

  Alice checks that all elements of Z differ from each other by at least 2, otherwise Alice repeats with a new random prime p.

- Alice sends p plus the following list

  List = [Z1 , Z2 , Z3 , Z4 , Z5+1, Z6+1]

  to Bob

  Note: all values after the a-th position have 1 added.

- Bob checks if the b-th value is his secret, Bob checks if

  List[b] $\equiv$ r (mod p)

- If it is congruent then

  b <= a (Alice is richer than or equal in wealth to Bob). Otherwise

  b > a (Bob is richer than Alice)

- Bobs tells Alice who is richer.

- Effectively Alice adds 1 to values greater than hers (a=4). Bob checks if the one in his position (b=3) has one added to it. If it has then he is richer than Alice.

Example Alice Bob

Wealth a : {1..6} = 4

Wealth b : {1..6} =3

Public Key pubA = (5, 551) ------->

Private Key privA = (101, 551)

r=123 (Bob's secret)

<----- C

C=E_{pubA}(123)-3=123^5-3=16-3=13

Y[1] = D_{privA}(C + 1)
= D_{privA}(13 + 1) = D_{privA}(14) = 127

Y[2] = D_{privA}(13 + 2) = D_{privA}(15) = 250

Y[3] = D_{privA}(13 + 3) = D_{privA}(16) = 123 i.e. $D_{privA}(E_{pubA}(r)-3+3)=r$

Y[4] = D_{privA}(13 + 4) = D_{privA}(17) = 365

Y[5] = D_{privA}(13 + 5) = D_{privA}(18) = 113

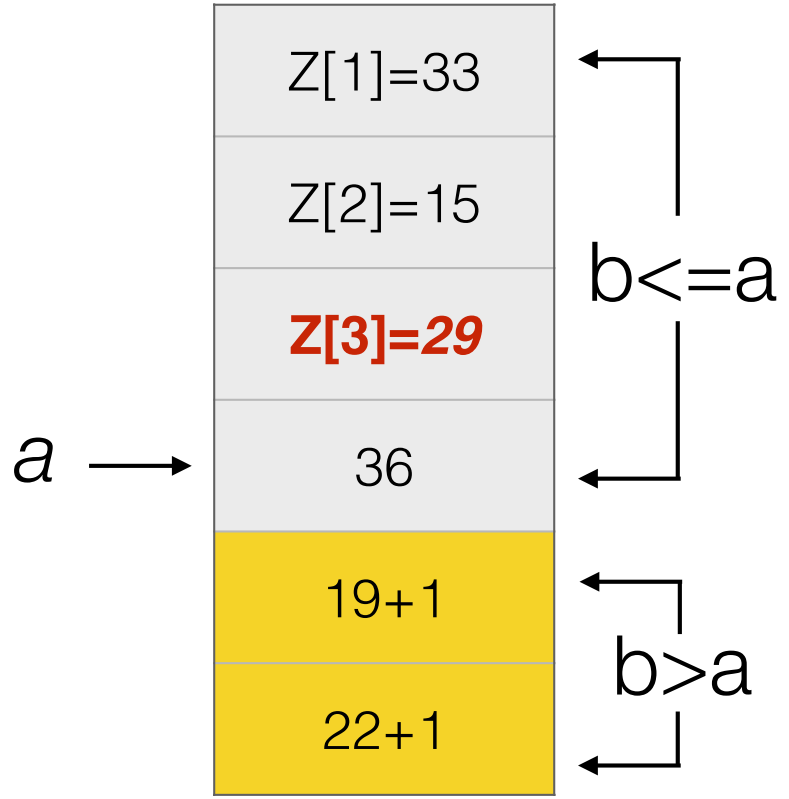
Y[6] = D_{privA}(13 + 6) = D_{privA}(19) = 304

Notes

- Alice runs RSA giving n=19x29=551, e=5, d=101. Public key(5,551)
- Bobs chooses large random number close in magnitude to n but less than n.
- Bob encrypts r using Alice's public key (i.e. $E_{pubA}(r)=r^e \bmod n$) and sends C=$E_{pubA}(r)$ - b to Alice.
  If r=123 then $E_{pubA}(123)$=1235 mod 551=16 and Bob sends C=16-3=13

- Alice generates the following list of values
  Y[k]=DprivA(C+k) for k=1..6 i.e. for each million.
  Y[1]=DprivA(14)=127
  Y[2]=DprivA(15)=250
  Y[3]=DprivA(16)=123
  Y[4]=DprivA(17)=365
  Y[5]=DprivA(18)=113
  Y[6]=DprivA(19)=304
- Alice does not know which value corresponds to b=£3M.

```
p    = large random prime = 47
Z[1] = 127 mod 47 = 33
Z[2] = 250 mod 47 = 15
Z[3] = 123 mod 47 = 29 i.e. r mod 47
Z[4] = 365 mod 47 = 36
Z[5] = 113 mod 47 = 19
Z[6] = 304 mod 47 = 22
```



Send --> p and | 33, 55, 29 = r mod p?, 36, | 20, 23 |

if Z[3]=123 mod 47 then b<=a else b>a

Notes

- Alice picks a large random prime p (less than r).
- Alice computes Z, a list of values mod p from Y.
- If p=47
  Z = [127, 250, 123, 365, 113, 304] mod 47
  = [33, 15, 29, 36, 19, 22]
  Z elements differ from each other by at least 2.
- Alice sends p and list Z but adds 1 to each element in Z corresponding to millions greater than hers (a=4) i.e. to £5M and £6M (last two in list).
- Alice sends p=47, List=[33, 15, 29, 36, 20, 23] to Bob.

- Bob checks if List[b] ≡ r mod p i.e. is 29 ≡ 123 mod 47?
- It is, therefore
  Alice(a) >=Bob(b) otherwise it would be Bob(b)>Alice(a).
- Bobs tells Alice who is richer.

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:

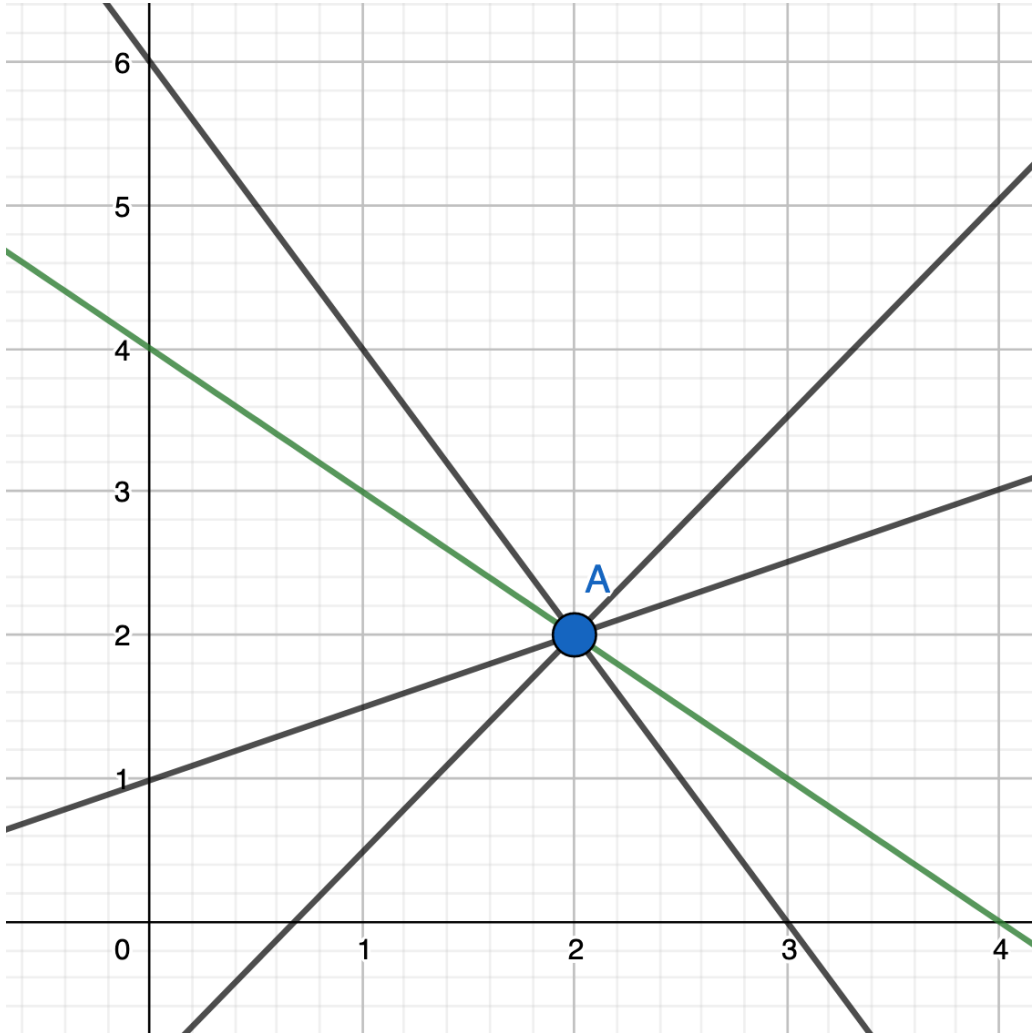
- For any polynominal $P(x)$ of degree upto at most $N-1$ there exist coefficients $r = (r_1, .., r_N)$ (called the recombination vector) such that:

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

(see Recombination vector slide later)

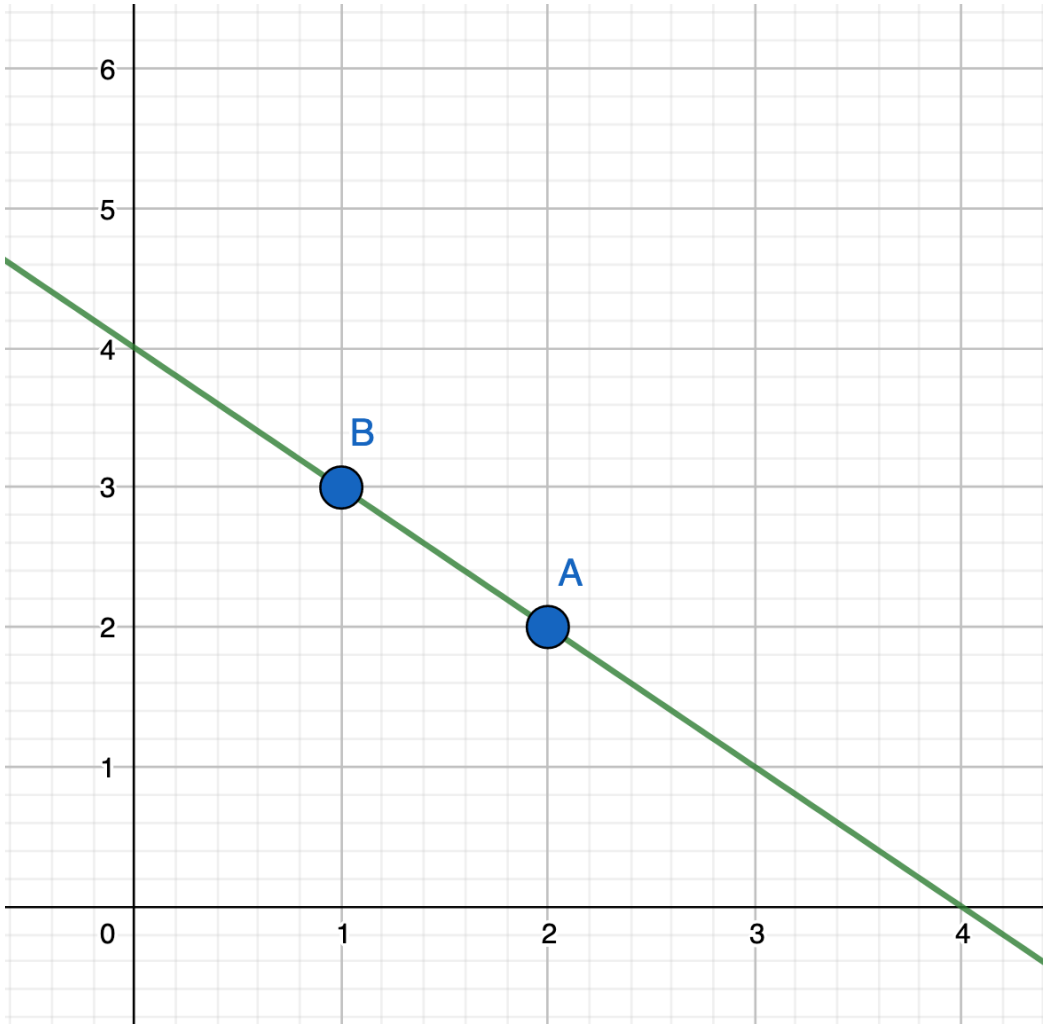
For 1 POINT

- 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)



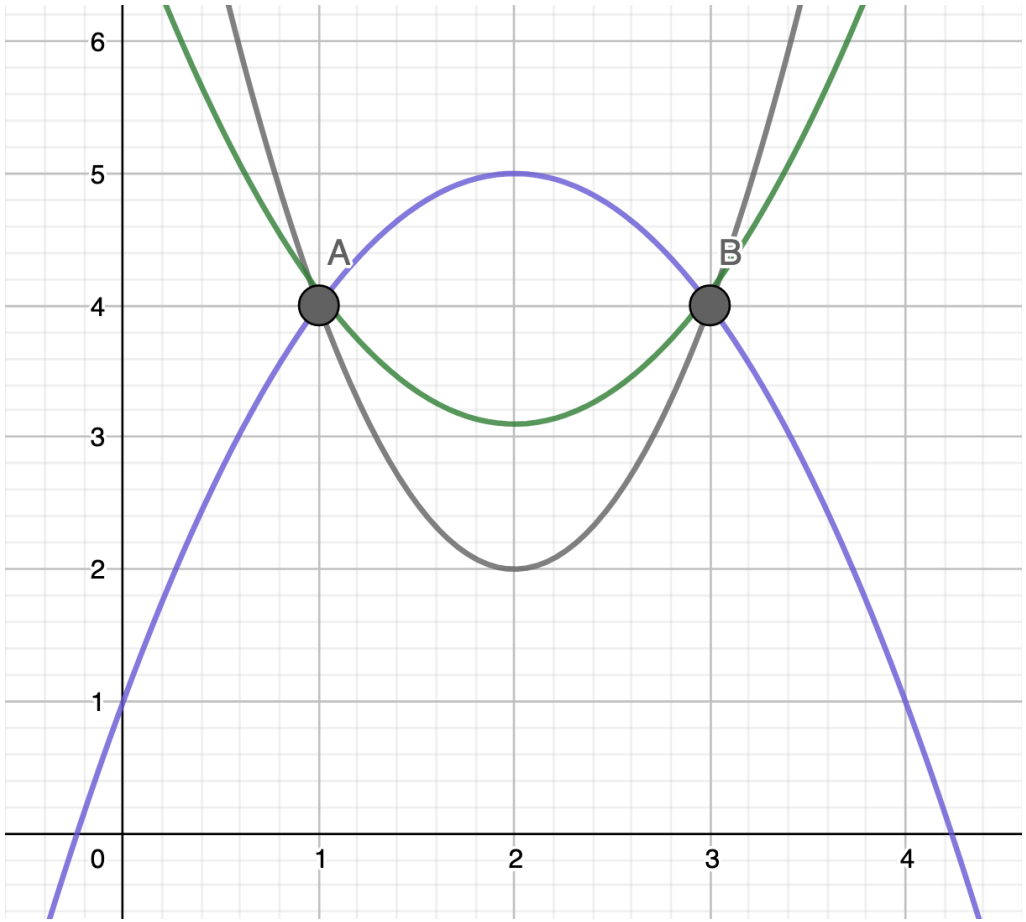
For 2 POINTS

- 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



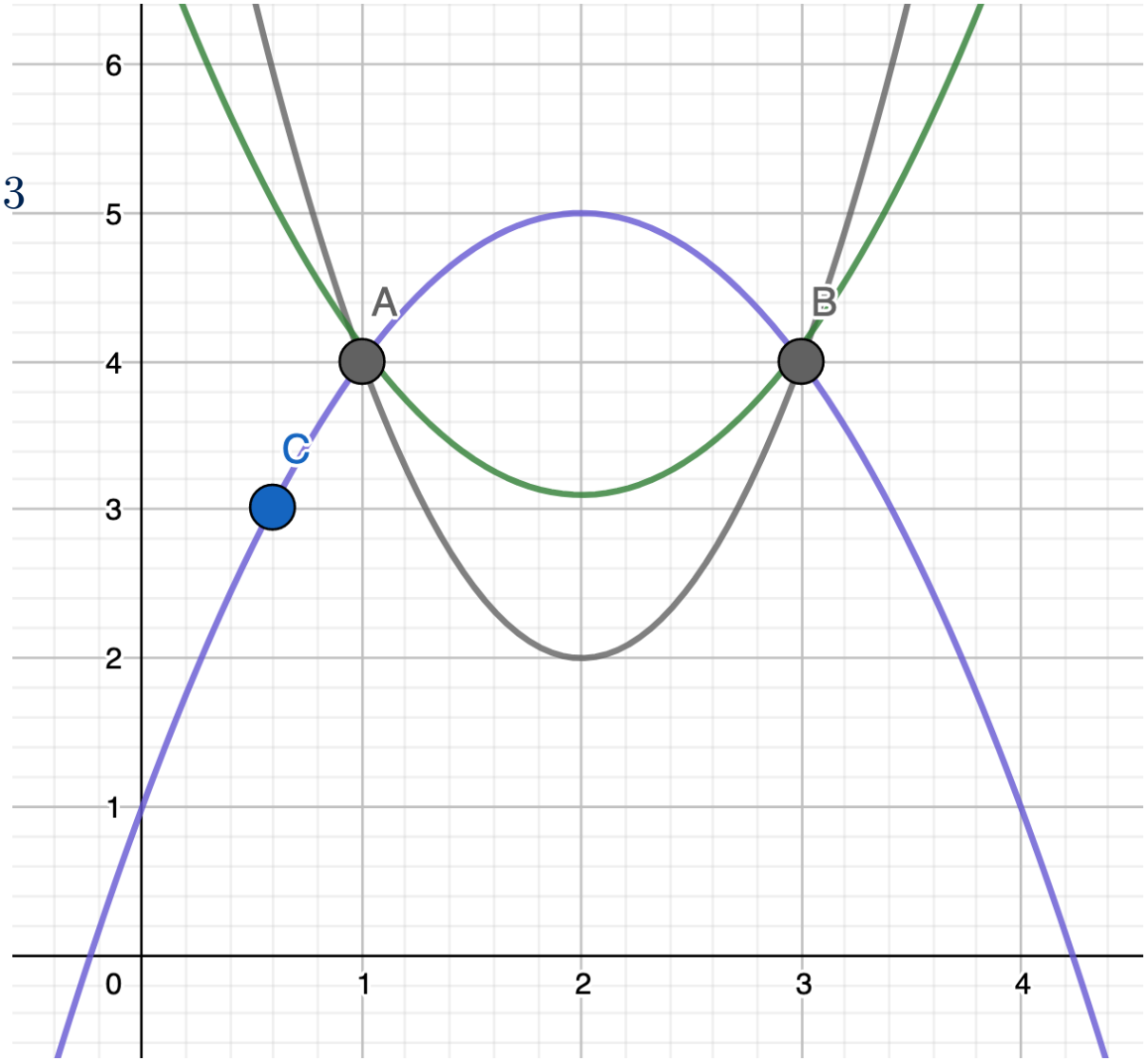
For 2 POINTS

- 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

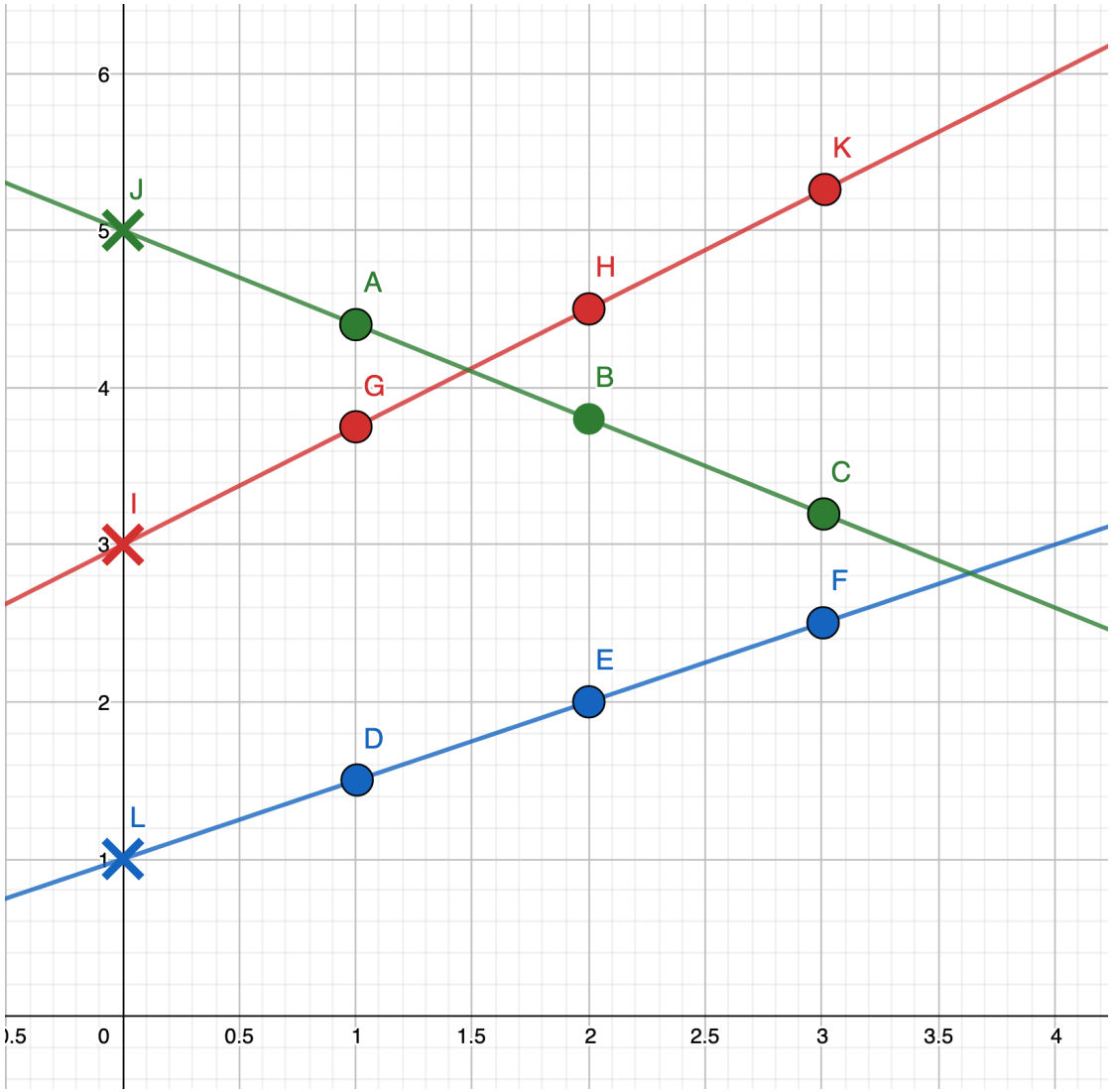
- There is only 1 polynomial $P(x)$ of degree 2 that passes through 3 points.
- There is also only 1 value for $P(0)$ e.g. the value 1 in the plot on the right



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

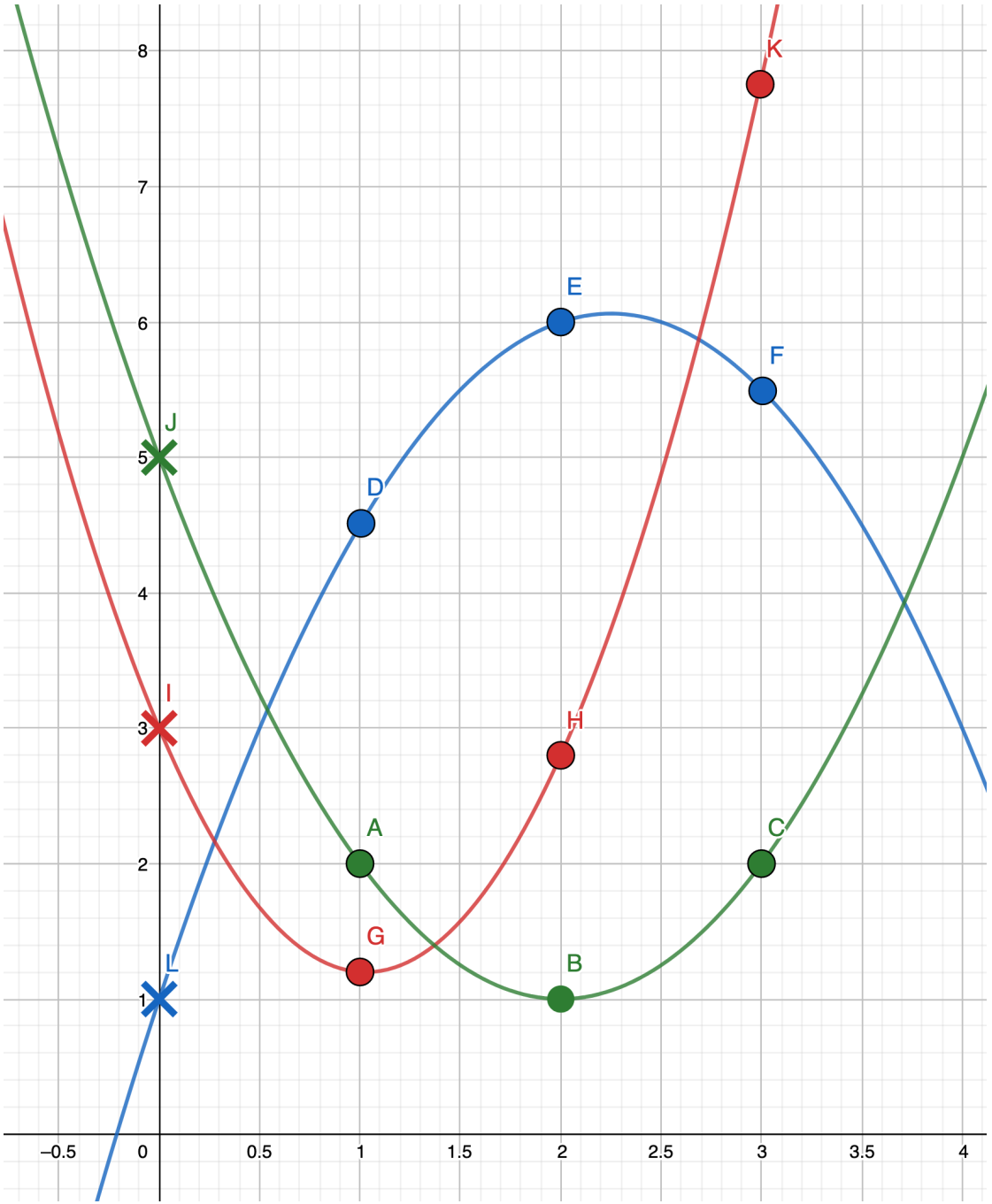
Alice wants to split and share a secret number, for example, J=5 with N other parties, such that J can only be reconstructed if any two of the N parties combine their shares.

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 $J = P(0)$

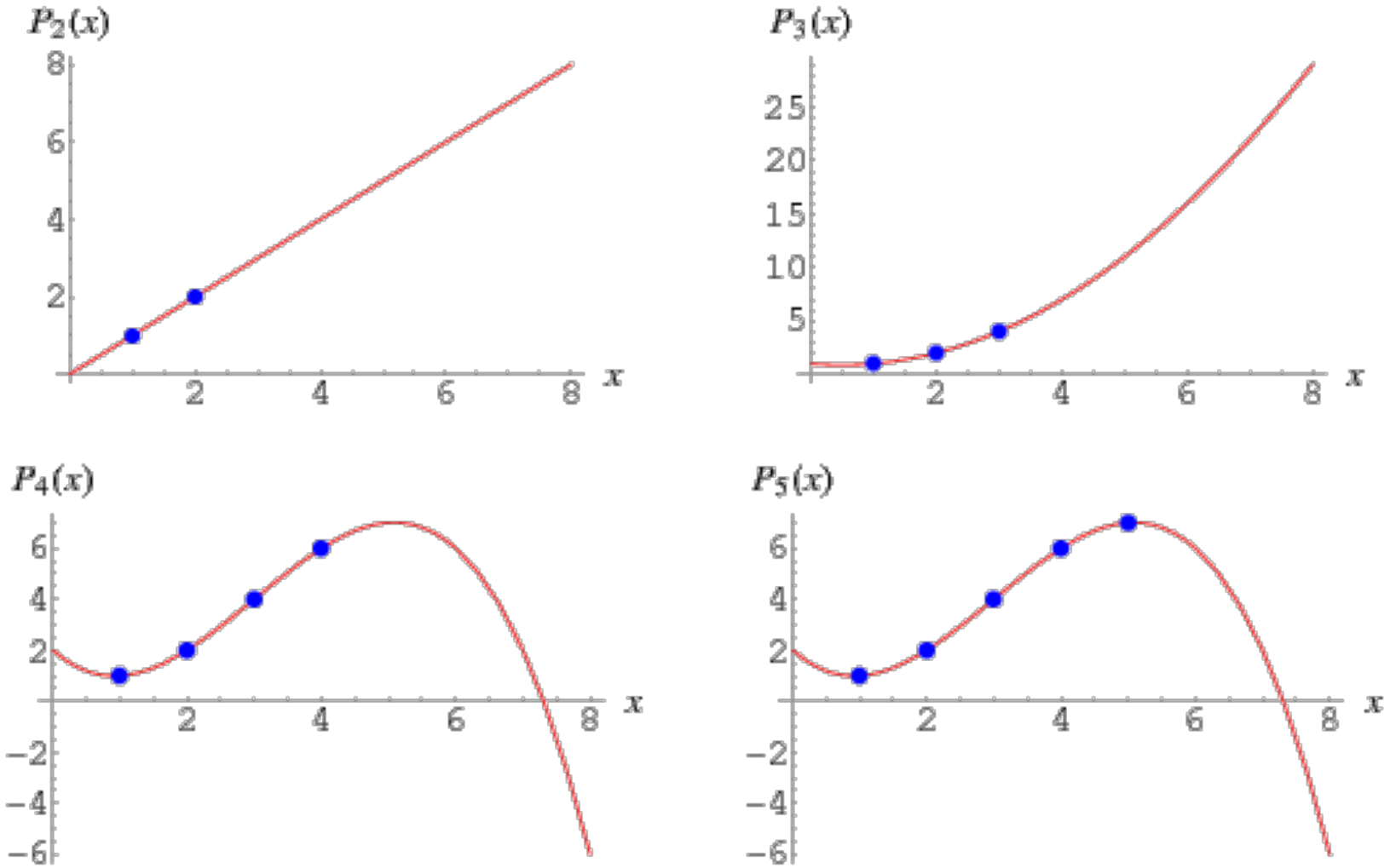


This works for more parties and higher degrees also. For example, to require 3 parties to construct a secret, Alice would generate a random polynomial of degree 2.

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

$$P(0) = \sum_{i=1}^{N} r_i \cdot P(i) \qquad \text{P(0) = 1 * 10 + 2 * (-15) + 4 * 6}$$

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 $\delta_i(0)$.

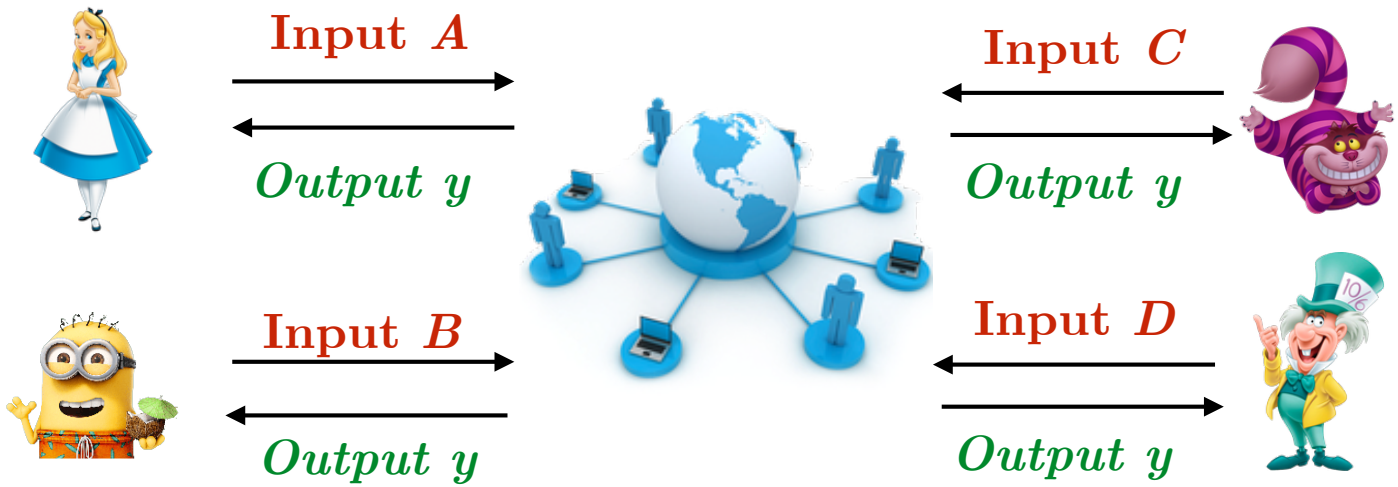
Hence the *same* unique recombination vector works for all polynomials $P(x)$ of degree up to at most $N-1$. This means anyone (every party) can compute it - it is public knowledge.

That's all class

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

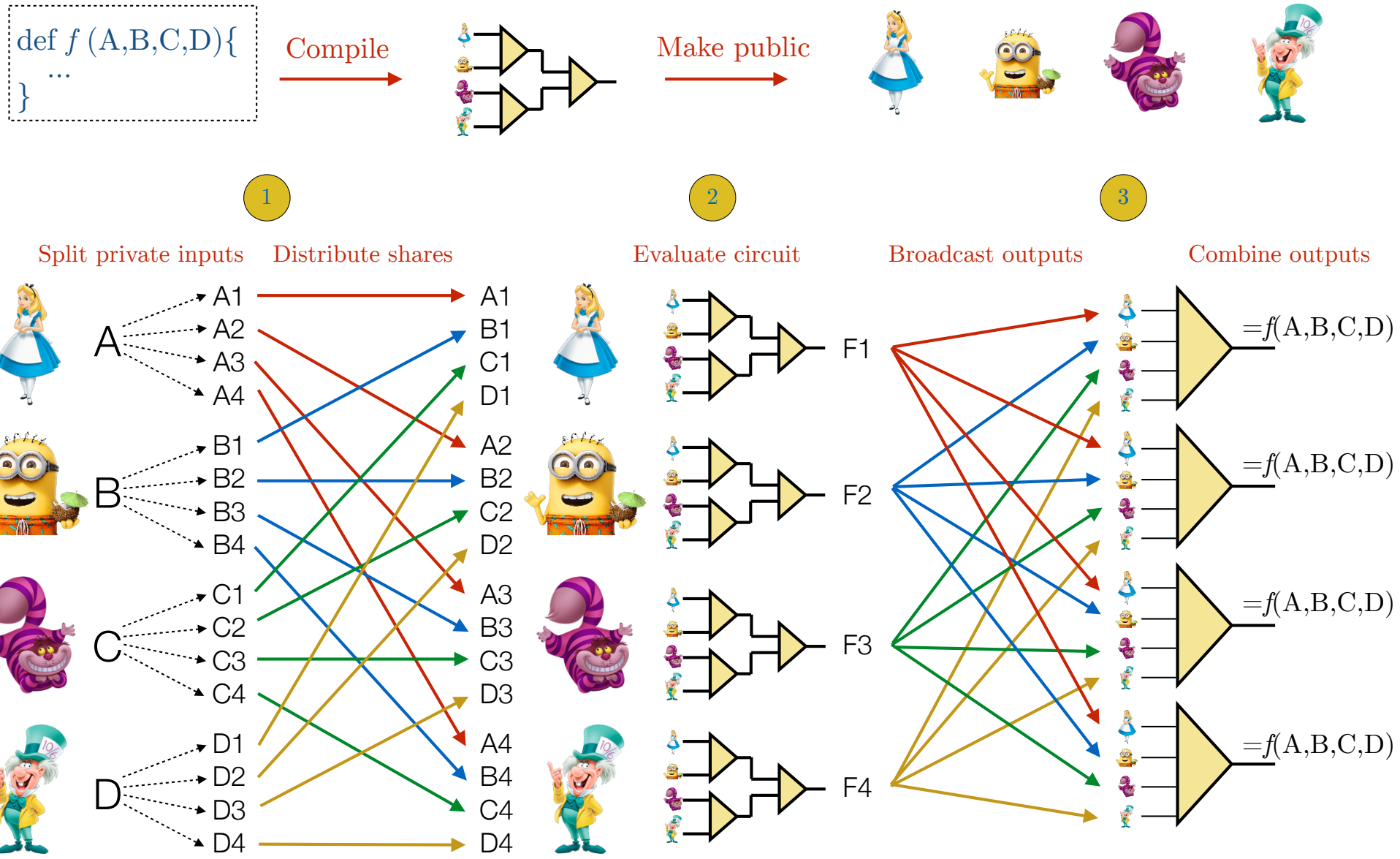
Secure Multi-party Computation (MPC)

$y = f(A, B, C, D)$, 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



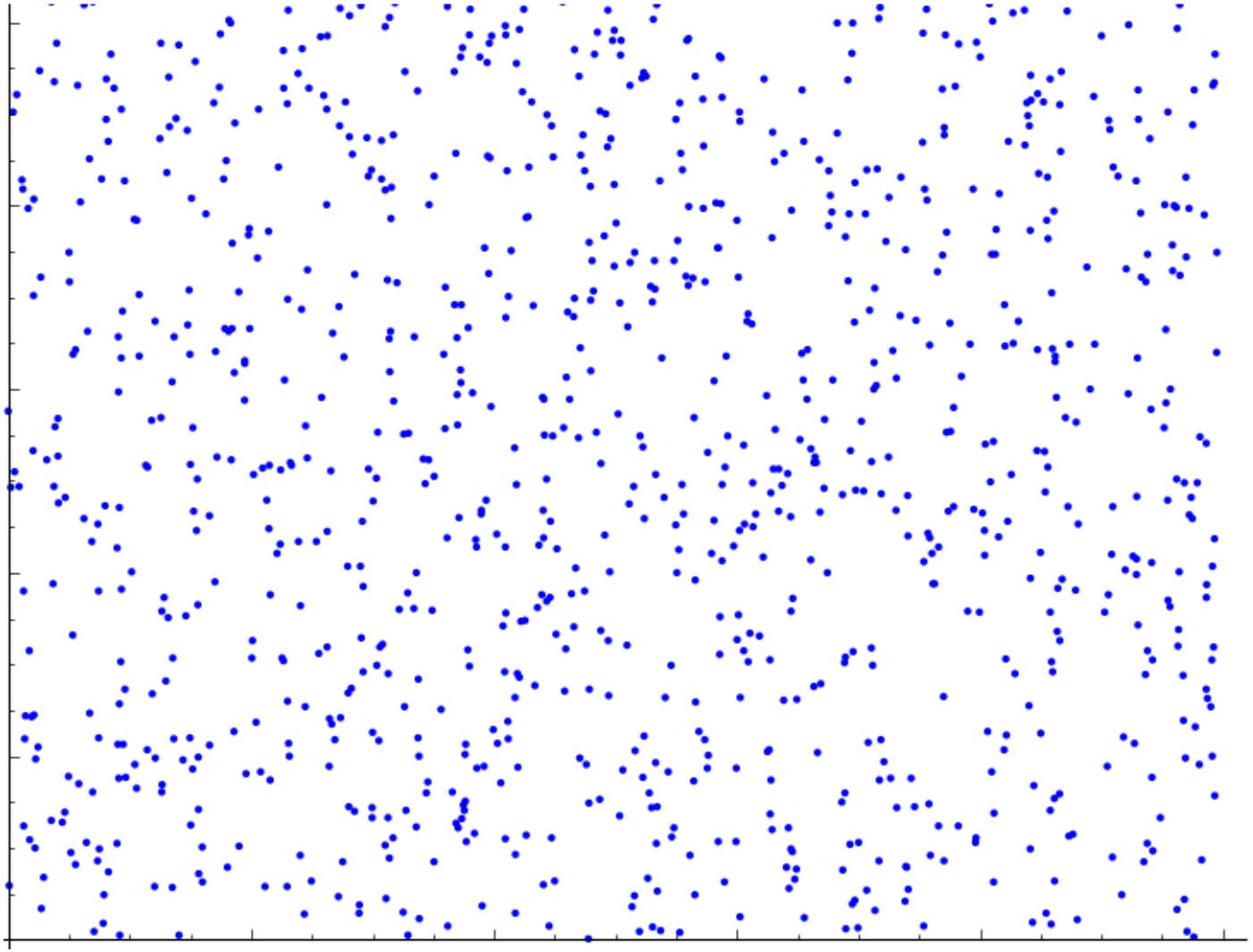
Notes

- BGW (Ben-Or, Goldwasser, Widgerson) is a multi-party protocol that compiles the public function into an arithmetic circuit consisting of addition gates and multiplication gates (over a finite field modulo a prime p denoted F_p). We'll look at the semi-honest version.
- The circuit is sent to, or otherwise made known, to each party (is public). Each party could also compile the function.
- Before evaluating its circuit each party will "split" its private input value into secret shares and send one share to each party e.g. 4 shares if there are 4 parties.
- Each party then "securely" evaluates its circuit using the shares that it receives, using them as the values of the input wires of its circuit. For example, Party A will evaluate its circuit using the shares A1, B1, C1, D1.
- The output of every gate must also be a share of the correct evaluation of the plaintext result of that gate.
- After evaluation of the circuit, each party broadcasts the share of the final output gate of its circuit (F1 to F4 respectively) to all parties.
- Each party then "combines" the shares received to produce the plaintext result of the function!
- But how does the BGW protocol (i) split an input value into shares, (ii) evaluate the ADD and MUL gates, (iii) combine the final shares to produce the function's plaintext result?

- Any computable function with a fixed-length input can specified as a polynomial-sized boolean circuit using AND and NOT gates. This can be changed into an arithmetic circuit.
- **Communication** between parties is secure. Parties will do each step synchronously.
- Note: there is no encryption involved. The BGW protocol effectively "hides" all input values and all intermediate outputs of gates.
- Original paper if interested: *Completeness Theorems for Non-Cryptographic Fault- Tolerant Distributed Computation (extended abstract)*, Michael Ben-Or, Shafi Goldwasser, Avi Wigderson, 1988. https://dl.acm.org/citation.cfm?id=62213

Notes

- Why arithmetic over a finite field? Without going into details, integer arithmetic yields smooth polynomial curves that an adversary can exploit. If we use polynomials over a finite field, we get points that disjoint and random like below. See Wikipedia on Shamir secret sharing for more details.



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.

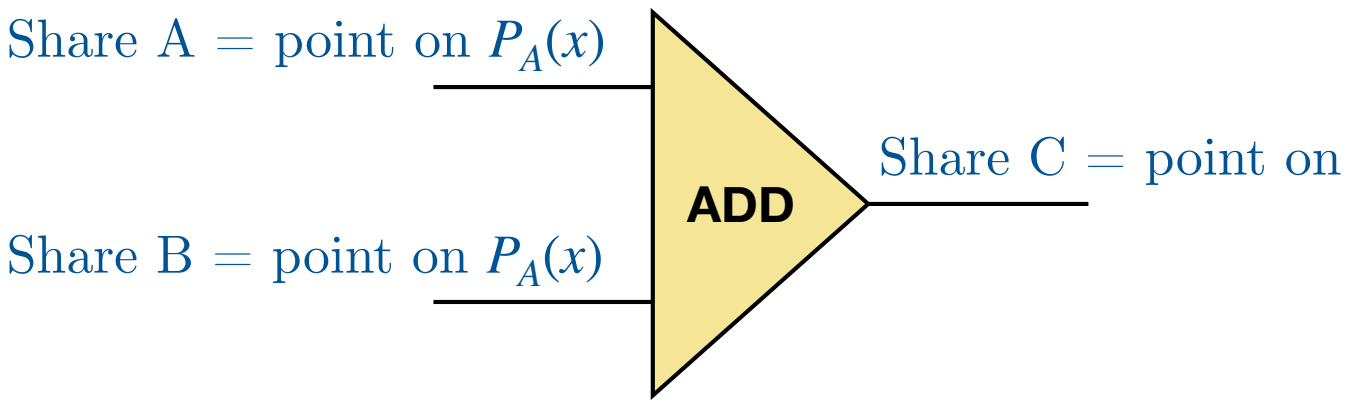
  So (7, 12) means able to achieve perfect security for any group of 6 people

Notes

- In the BGW protocol we will use modular arithmetic over a finite field were p is a prime number and $p > N$ (N=no of parties).
- The coefficients $a_1, \ldots, a_T$ are randomly chosen from a uniform distribution over the integers in $[1, p)$.
- For correct MULtiply gate operation (see later) we also require $2T < N$.
- We typically write the polynomial in increasing powers starting with the secret S.

- In Shamir's original paper and other sources, the formulation (K, N) is chosen and the degree given as $K - 1$.
- We'll use $T + 1$ instead of K and adjust the formulation accordingly.
- Do read Shamir's very readable 2-page paper. "*How to Share a Secret*", Adi Samir, 1979, https://doi.org/10.1145/359168.359176

Phase 2: ADDition Gate



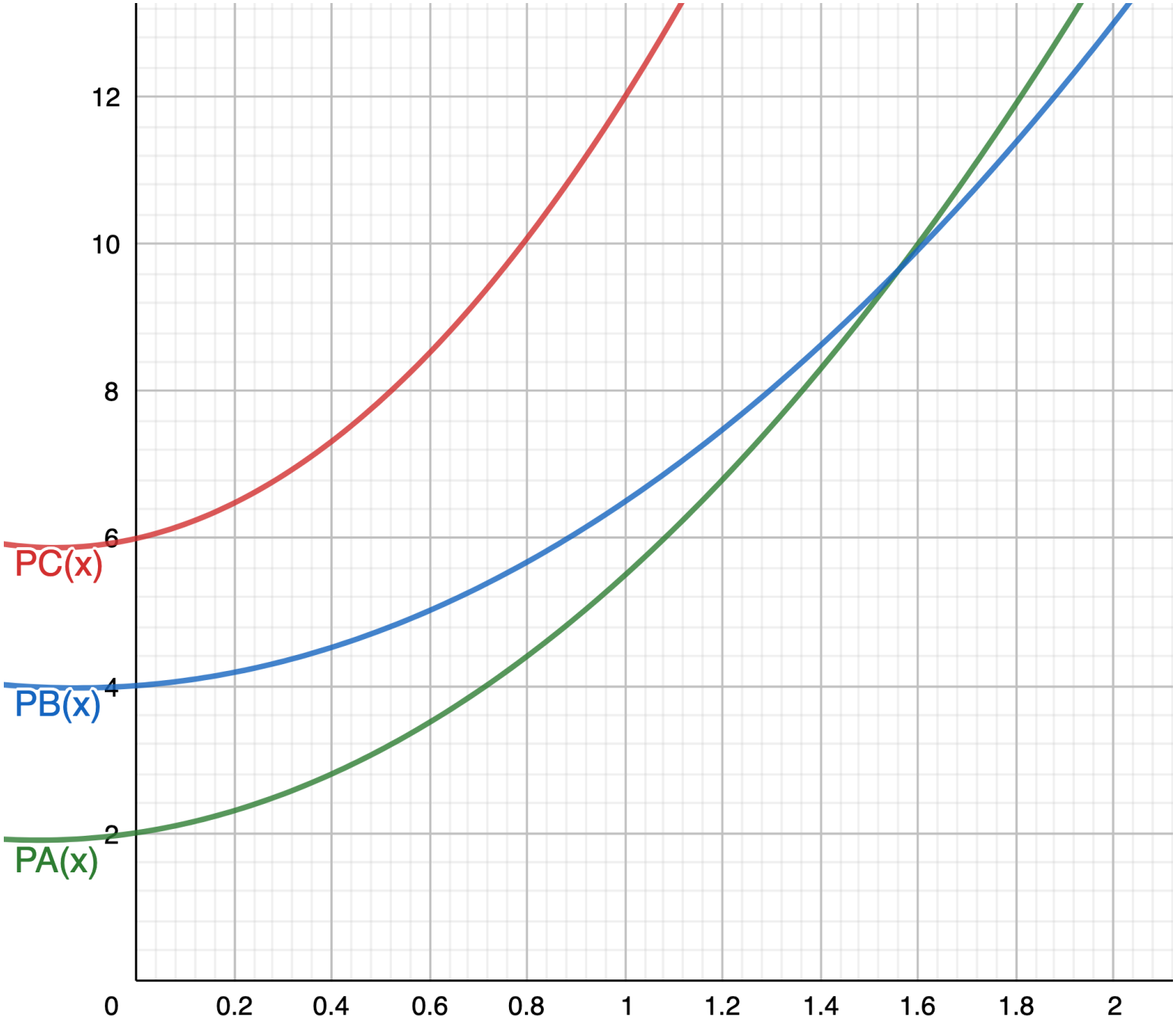
- Example:

$$\text{If} \quad P_A(x) = 2 + 1x + 2.5x^2$$

$$\text{and} \quad P_B(x) = 4 + 0.5x + 2x^2$$

$$\text{then} \quad P_C(x) = 6 + 1.5x + 4.5x^2$$

- 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)

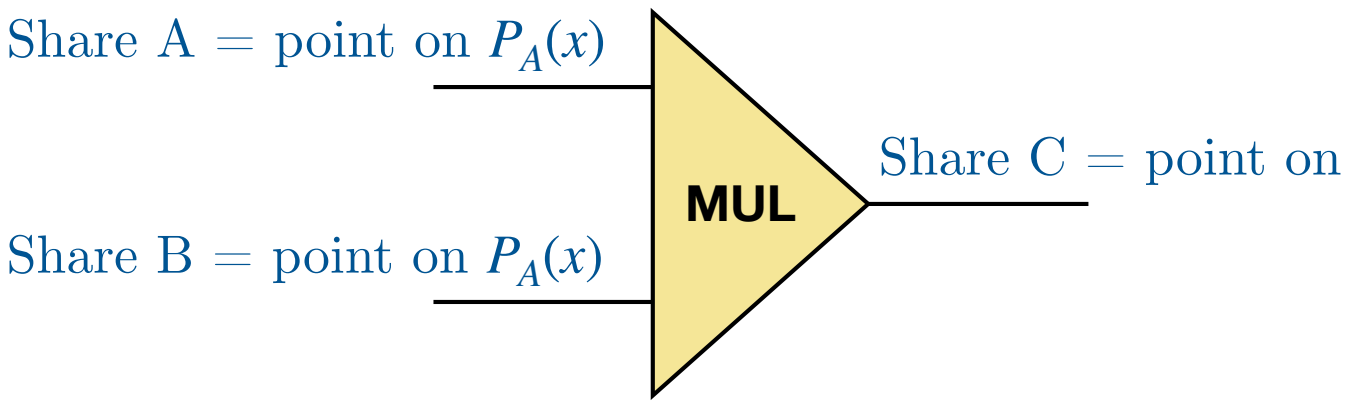
Notes

- Once the shares of the circuit's inputs are distributed the parties proceed to evaluate their circuit gate by gate.
- Each gate will have a share on each of its input wires corresponding to points on a different randomised polynomial.
- Each gate produce a share on its output wire of either the sum (for an ADD gate) or the product (for a MUL gate) of the input shares, and corresponds to a point on a new randomised polynomial of degree T.
- For an ADD gate each party computes its own share of the output wire value from its input wire shares using a simple modulo prime addition.
- This means that for ADD that no interaction is needed between the parties (its non-interactive).
- Multiplication by a constant gates can be implemented in a similar way.

Why does this hide A+B?

- The input shares correspond to points on randomised polynomials where the constant term corresponds to the secret value, e.g. $P_A(0) = 2$ and $P_B(0) = 4$ for the example.
- Adding the values corresponds to adding the polynomials and will produce a new randomised polynomial where the constant term corresponds to the sum, e.g. $P_C(0) = 6$
- This is an example of a **homomorphic property**:
  If $P_A(1), \ldots, P_A(N)$ are shares of A, and $P_B(1), \ldots, P_B(N)$ are shares of B then $P_A(1) + P_B(1), \ldots, P_A(N) + P_B(N)$ are the corresponding shares of $A + B$

Phase 2a: MULtiplication Gate



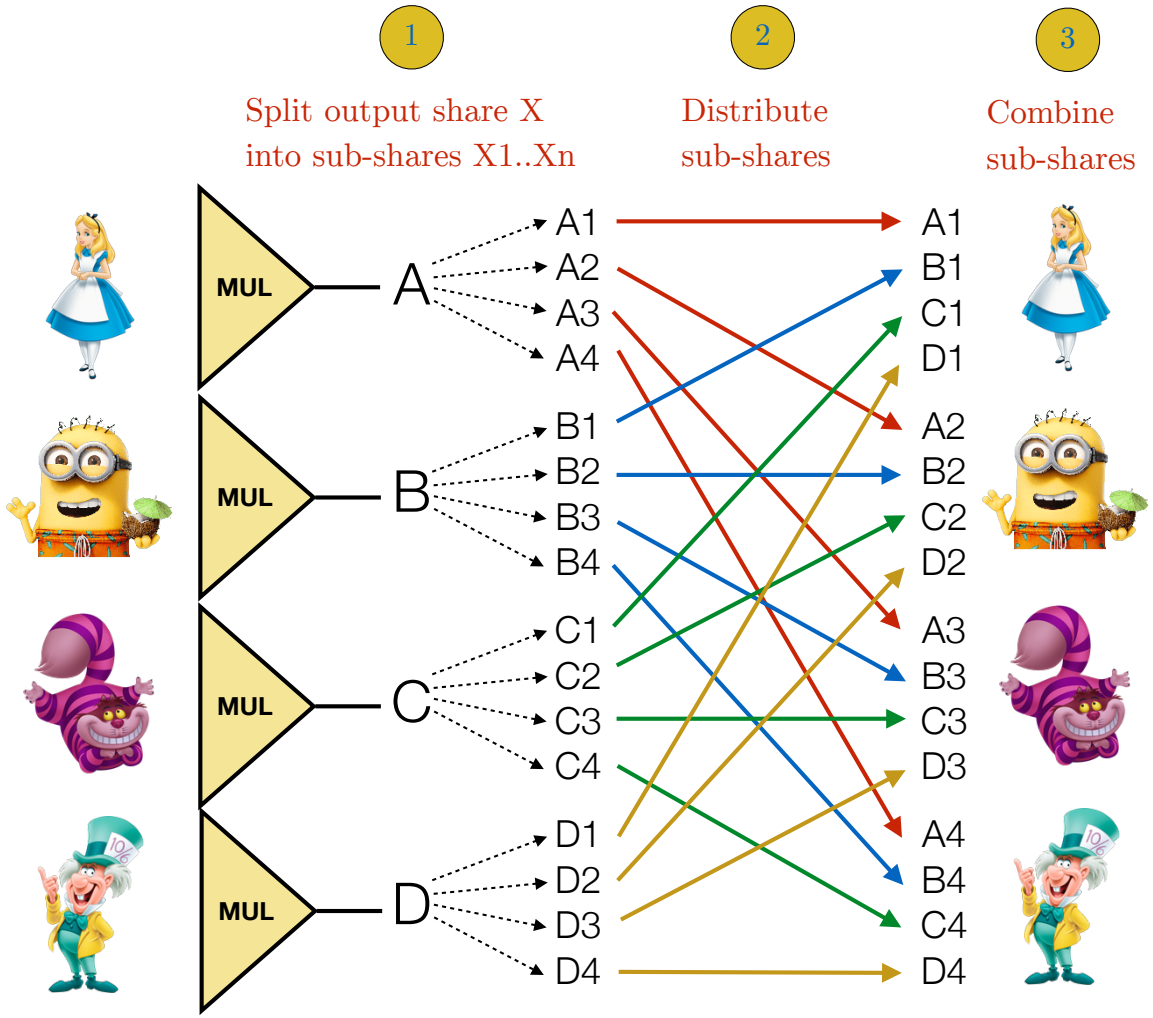
Multiplication is more complex. Consider, if $P_A(x) = 2 + 4x$ and $P_B(x) = 3 + 2x$ then $P_C(x) = 6 + 16x + 8x^2$. Here, the output share, 6, is correct. However the degree of the polynomial, 2, is too high. In this case we need the degree to be 1.

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.

Notes

- Following multiplication of the input shares we need to carry out a degree reduction step that reduces the output polynomial $P_C(x)$ of degree 2T to a new output Polynomial $Q_C(x)$ of degree T where $Q_i(1), \ldots, Q_i(N)$ are the shares for parties 1 to N.
- There is a second, less obvious problem, $P_C(x)$ is not random. $P_C(x)$ is a composition of 2 polynomials, so leaks information, that an adversary can use, we need the output polynomial to be irreducible (cannot be factored).
- For the second problem, BGW randomises the coefficients of the output polynomial.

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.

Notes

Why does this work?

- Parties can perform a linear recombination on $N > 2T + 1$ points of the polynomial to generate a new Shamir sharing of $H(0) = C = A \cdot B$ of degree T as follows:

$$H(x) = \sum_{i=1}^{N} \delta_i(x) \cdot Q_i(x) = \sum_{i=1}^{N} r_i \cdot Q_i(x)$$

  where $(r_1, \ldots, r_N)$ is the publicly-computable recombination vector for all polynomials of degree at most $N - 1$. Continuing we then have:

$$H(0) = \sum_{i=1}^{N} r_i \cdot Q_i(0) = \sum_{i=1}^{N} r \cdot S_j = C$$

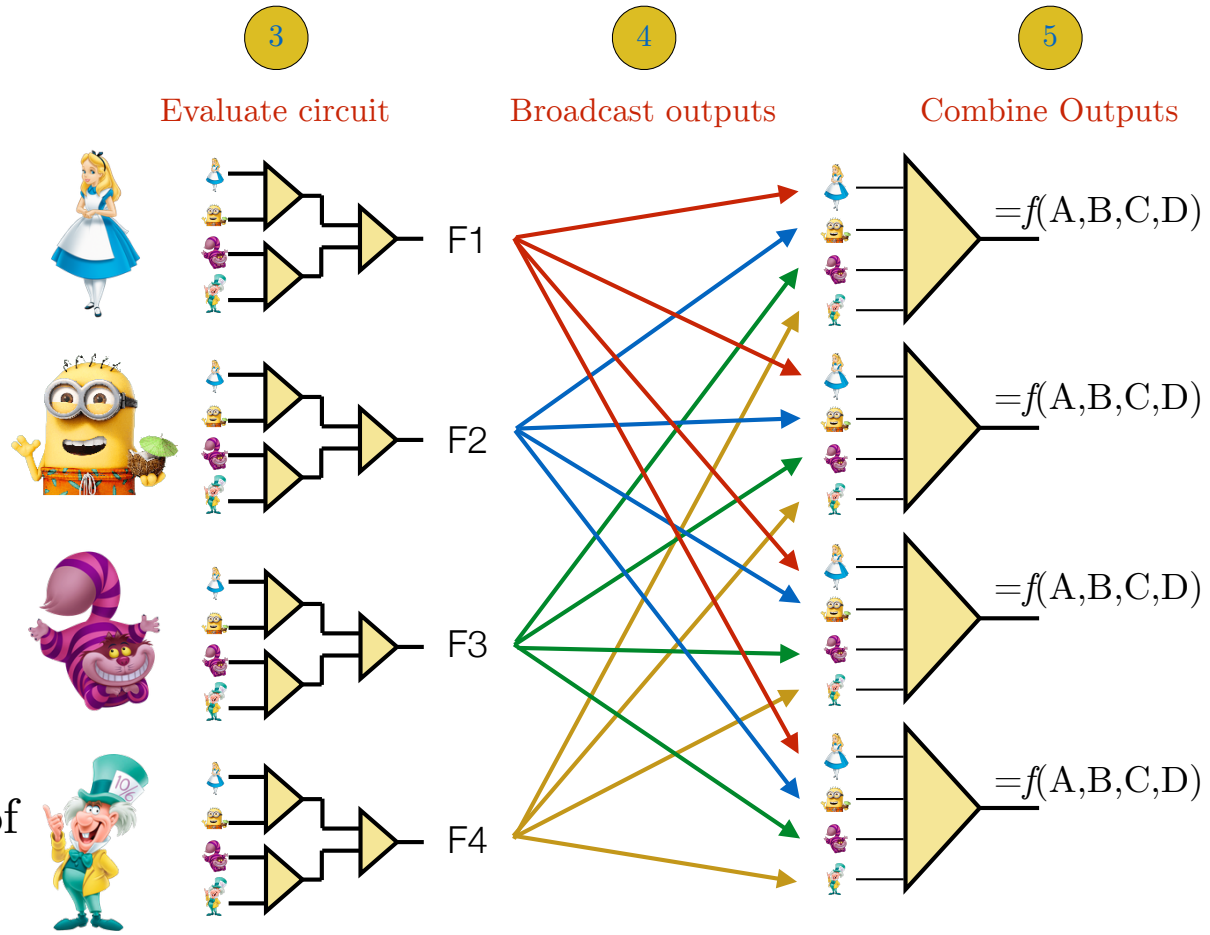
- And since $Q_i(x)$ is of degree T, so is $H(x)$. Note also that the shares of $H(X)$ are the shares of C:

$$H(j) = \sum_{i=1}^{N} r_i \cdot Q_i(j) = \sum_{i=1}^{N} r_i \cdot Q_i(j) = C_j$$

- Note: Most papers and other books will normally describe a method that uses the inverse of a so-called Vandermonde in matrix operations to do degree reduction. This can be shown to be equivalent to the use a recombination vector and maybe easier to understand follow if you have a background in linear algebra.

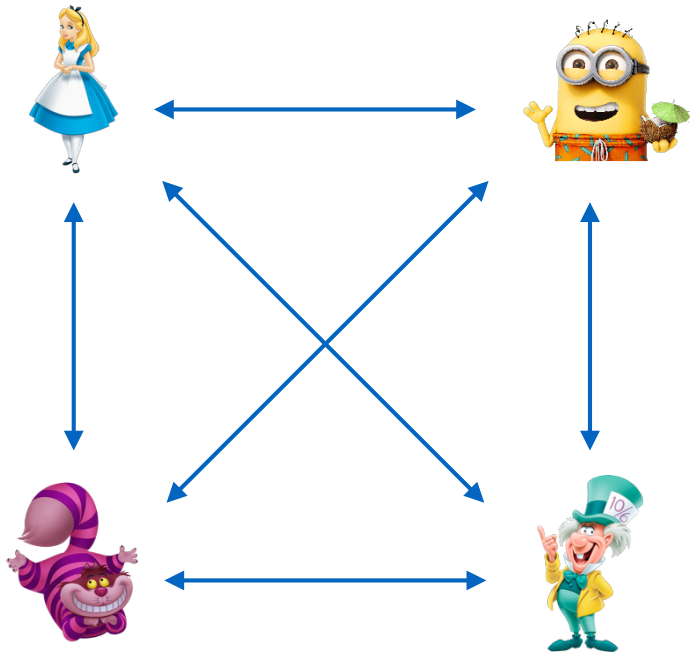
Phase 3: Final result

- 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