



# APPLICATIONS OF POLYNOMIAL INTERPOLATION (SECURE MULTI-PARTY COMPUTATION)

Pradeep Kumar Pal  
(2019811001)

Project Group 8

## **ABSTRACT**

Secure Multi-party computation is a subfield of cryptography which enables a group to jointly perform computations over their inputs without disclosing their private inputs. First introduced by Andrew Yao in the 1980s, secure multi-party computation (SMPC) has developed from a theoretical model to an important tool for building applications which requires large scale privacy preservation of the inputs. The participants agree on a function to be computed, and then use a SMPC protocol to jointly compute the output of that function on their secret inputs without revealing them. This report aims to provide an introduction to the foundations of SMPC, its development over the years, discuss its usage in a simplistic real-life situation with focus on the algebra involved and then finally enlist some present day applications while also highlighting the work of the fellow group members that has helped to understand SMPC.

# **INTRODUCTION**

## **Polynomial Interpolation**

Polynomial functions have the ability to be visualized graphically which adds to its understanding from the nature of the plot. But often certain graphical data contains gap, while data on the either sides of the gap is available. Using interpolation, an estimate can be made of the missing values, while there is a trade off between having a better fit and having a smooth well-behaved fitting function. One of the basic methods of interpolation is to draw straight lines between known data points and function is considered to be a combination of those straight lines, aka, linear interpolation. As can be expected, this trivial method brings in a lot of error and hence is required a more sophisticated approach. This job is served by using a polynomial function to connect the points. A polynomial by its definitive meaning is an expression having a sum of terms, each distinct term includes a variable raised to some exponent and multiplied by a coefficient.

If a set of data contains  $n$  known points, then there exists exactly one polynomial of degree  $n-1$  or smaller that passes through all of those points. In some sense, the polynomial's graph serves to 'fill in the curve' to account for the missing points between the known points.

## **Applications of Polynomial Interpolation**

As can be thought of this method can be extended to use in approximating complicated curves, like the ones used in typography, provided a few points are given. More applications include evaluating natural algorithm and trigonometric functions. While it also forms the basis for algorithms in numerical quadrature and schemes like Secure Multi-party computations, which will be discussed in more details hereafter.

## **Secure multi-party computation**

Secure multi-party computation (aka secure computation, multi-party computation (MPC), or privacy-preserving computation) is a subfield of cryptography which aims to create methods to allow parties to jointly compute a function over their inputs while keeping the inputs private. This is different from traditional cryptographic tasks, which assures security and integrity of communication or storage and the adversary is outside the system of participants, the cryptography in this model protects participants' privacy from each other. The term secure computation is used to broadly encompass all methods for performing computation on data while keeping that data secret. A computation method may also allow participants to confirm the result is indeed the output of the function on the provided inputs, which is known as verifiable computation. There are two main types of secure and verifiable computation: outsourced computation and multi-party computation. We will briefly discuss outsourced computation before discussing multi-party computation.

## Outsource computation

In this kind of computation, one party owns the data and wants to obtain the result of computation on that data and sends it to the second party which on receiving it stores the data in an encrypted form, performs computation on the encrypted data, and provides the encrypted results to the data owner, without learning anything about the input data, intermediate values, or final result at any point of time. The results are then sent to the data owner in the encrypted form itself who then decrypts the returned results to obtain the output.

## Multi-party computation

While the outsourced computation limits its usage for multiple parties, as in that case, it would be needed the data to be encrypted by one of the parties who then are at the liberty to look into the data. Hence secure multi-party computation (MPC) enables a group of independent parties to jointly compute a function using all of their private inputs without disclosing them. The major difference from outsourced computation is that all of the participants are data owners who participate in the computation.

## Brief history of MPC

The idea of secure computation was first introduced by Andrew Yao in 1982 in his paper titled, 'Protocols for Secure Computations (Extended Abstract)'. In: 23rd Annual Symposium on Foundations of Computer Science. IEEE Computer Society Press. 160–164. This paper introduced a general notion of secure computation, in which  $m$  parties want to jointly compute a function  $f(x_1, x_2, \dots, x_m)$  where  $x_i$  is the  $i^{\text{th}}$  party's private input. In a series of talks over the next few years, Yao introduced the Garbled Circuits Protocols which remains the basis for many of the most efficient MPC implementations till now.

Despite its theoretical interest, it was not until 2000s that algorithmic developments and computing costs were favourable for it to become realistic to build practical systems using a general purpose multi party computation. Fairplay (Malkhi *et al.*, 2004) made the first remarkable implementation of general purpose MPC, by demonstrating the possibility that a privacy-preserving program could be expressed in a high level language and can be compiled to executables that could be run by the data owners as a multi party protocol. Increasingly efficient protocols for MPC have been proposed, and MPC is now considered as a practical solution to various real-life problems such as distributed voting, private bidding and auctions, sharing of signature or decryption functions and private information retrieval published by Claudio Orlandi: Is multiparty computation any good in practice? (ICASSP 2011). The first large-scale and practical application of multi-party computation (demonstrated on an actual auction problem) took place in Denmark in January 2008 (Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielse, Jakob Pagter, Michael Schwartzbach and Tomas Toft (2008). "Multiparty Computation Goes Live". *Cryptology ePrint Archive* (Report 2008/068)).

## Applications of MPC

Below are enlisted some of the real life situations where MPC has been/can be applied.

### Scenario 1: Private Dating

Alice and Bob meet at a pub

- If both of them want to date together – they will find out
- If Alice doesn't want to date – she won't learn his intentions
- If Bob doesn't want to date – he won't learn her intentions



### Scenario 2: Private Auction

Many parties wish to execute a private auction

- The highest bid wins
- Only the highest bid (and bidder) is revealed



### Scenario 3: Private Set Intersection

Intelligence agencies holds lists of potential terrorists

- They would like to compute the intersection
- Any other information must remain secret

Play online poker reliably



MIS



FBI

### Scenario 4: Online Poker



In all scenarios the easiest real life solution is if an external trusted third party works. But trusting a third party is a very strong assumption. Can we do better? We would like a solution with the same security guarantees, but without using any trusted party. This role is played by MPC to use a protocol to emulate the trusted party.

### The general protocol

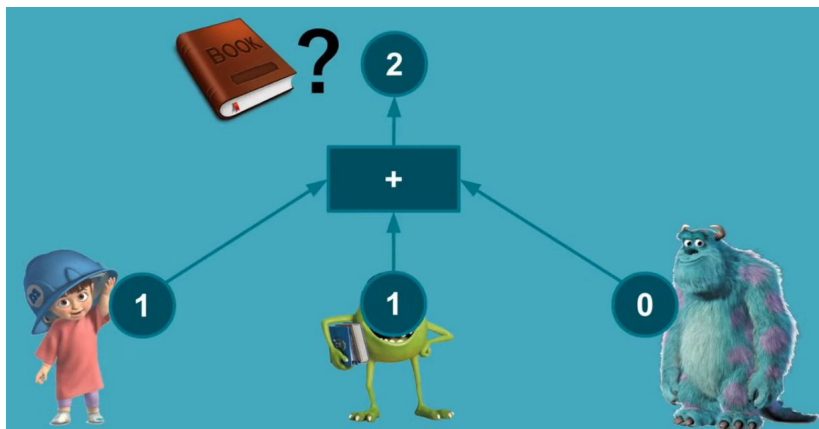
- All the parties  $P_1, \dots, P_n$  (modeled as interactive teams).
- Party  $P_i$  has private input  $x_i$ .
- The parties wish to jointly compute a (known) function  $y = f(x_1, \dots, x_n)$ .
- The computation must preserve certain security properties, even if some of the parties collude and maliciously attack the protocol.

Some more Applications:

1. **Yao's millionaires problem** : Yao in his paper introduces the problem as, "two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each other's wealth." So the goal is to compute the Boolean result of  $x_1 \leq x_2$  where  $x_1$  is the first party's private input and  $x_2$  is the second party's private input.
2. **Voting** : Secure electronic voting, in a simple form, is simply computation of the addition function which tallies the vote. Privacy of the vote are essential for similar technical reasons.
3. **Secure Machine learning** : MPC can be used to enable privacy in both the inference and training phases of machine learning systems. Oblivious model inference allows a client to submit a request to a server holding a pre-trained model, keeping the request private from the server  $S$  and the model private from the client  $C$ . An example of recent work in this setting include MiniONN (Liu et al., 2017), which provided a mechanism for allowing any standard neural network to be converted to an oblivious model service using a combination of MPC and homomorphic encryption techniques. In the training phase, MPC can be used to enable a group of parties to train a model based on their combined data without exposing that data. Hybrid approaches have been designed that combine MPC with homomorphic encryption (Nikolaenko et al., 2013b; Gascón et al., 2017) or develop custom protocols to perform secure arithmetic operations efficiently (Mohassel and Zhang, 2017). These approaches can scale to datasets containing many millions of elements.
4. **Other applications** : Privacy-preserving network security monitoring (Burkhardt et al., 2010), privacy-preserving genomics (Wang et al., 2015a; Jagadeesh et al., 2017), private stable matching (Doerner et al., 2016), contact discovery (Liet et al., 2013; De Cristofaro et al., 2013), ad conversion (Kreuter, 2017), and spam filtering on encrypted email (Gupta et al., 2017).

### Application to a real life problem:

**Anonymous Voting** : One night Boo, Mike and Sully (from the movie 'Meet your twin brother Neo' !!) are studying hard and the question arises whether they should read another book or go to sleep. They decide to do a simple majority voting. So everyone who wants to study another book votes 1 and everyone who wants to go to sleep votes 0. Then the votes are added up and let's say, it is found that the majority wants to study. Now they want the computer to sum up the votes without revealing the data.



**Note :** Since the outcome is known to everyone, the person who voted 0 knows that other two voted 1. But it is not true other way round.

Here, privacy preserving requirement is the only property of SMPC.

### Summing polynomials :

- We consider polynomials  $g$  and  $h$  of degree  $k$ .
- Define  $f = g + h$
- Then, a)  $F$  is also of degree  $k$ .  
b)  $f(x) = g(x) + h(x)$  for each value of  $x$ .


We can verify this using a simple example as follows:

$$\begin{aligned}g &= 1 + 2x + 3x^2 \\h &= 2 - 1x + 1x^2, \text{ then} \\f &= 3 + 1x + 4x^2\end{aligned}$$


This is cross verified by taking  $x=2$ ,  $g(2)= 17$ ,  $h(2)=4$  and  $f(2)=21$ .

So, how do we combine this with secret sharing to achieve SMPC.


1. We start with each party encode the vote as a secret polynomial of degree (i.e 1 less than the number of participants,  $n-1$ )
2. Since Boo voted 1, the polynomial would be


$$b(x) = 1 + ? x + ? x^2$$

Since Mike voted 1, the polynomial would be


$$m(x) = 1 + ? x + ? x^2$$

Since Sully voted 0, the polynomial would be

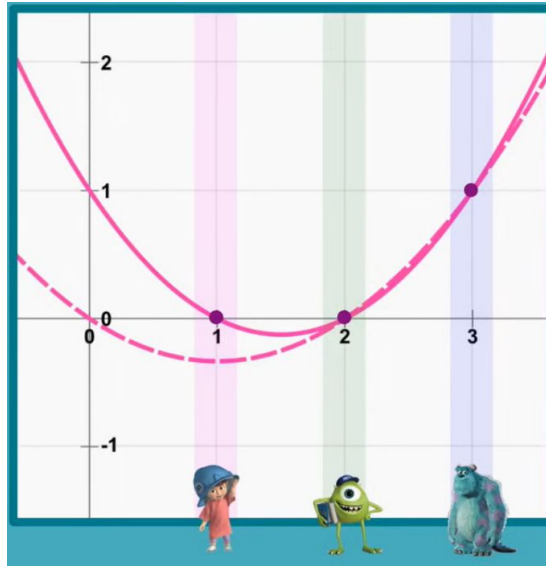

$$s(x) = 0 + ? x + ? x^2$$

Now if we sum up these polynomials, together by property of addition, it will have the form  $2 + ?x + ?x^2$ , and in other words,  $f(0) = 2$ .

Since it is a polynomial of degree 2 we know we can determine the polynomial if we have 3 points or shares on its line. Lets see how it works in our example and how we can get 3 points the polynomial  $f$  so we can find the outcome of the vote.

We start with the polynomial for Boo, lets say  $b(x) = 1 - \frac{3}{2}x + \frac{1}{2}x^2$ .

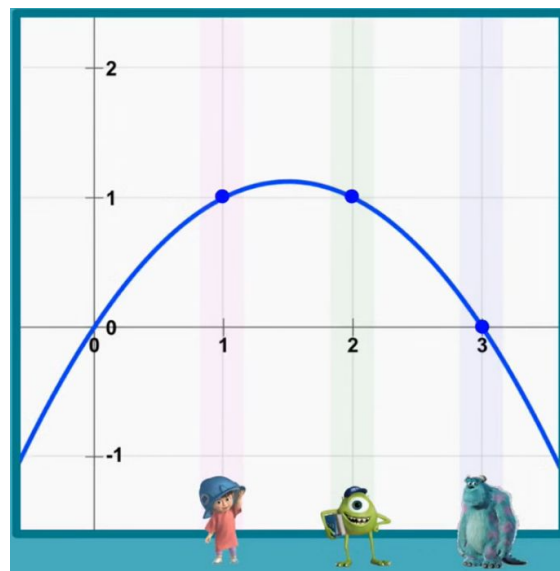
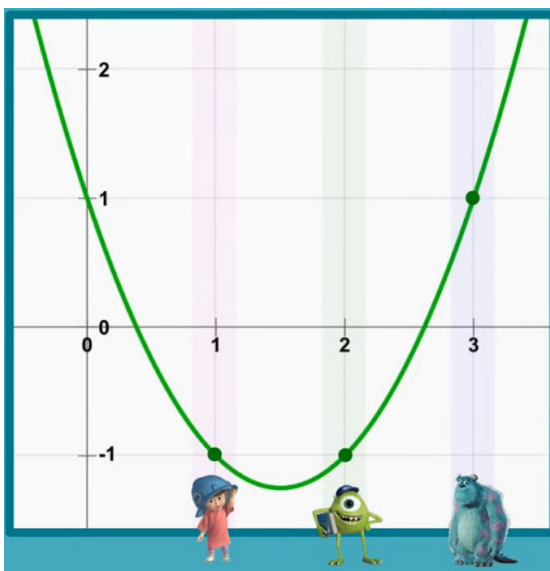
**NOTE :** The coefficients are only for the completeness of the equation. The important part here is that for  $x=0$ ,  $b(x) = 1$  i.e. the vote of Boo. Graphically the polynomial would look like this,



As with the secret sharing scheme, Boo gives shares (values of  $x$  where it intersects  $x$  axis) of the polynomial to Mike and Sully and keeps one share with herself. The shares are highlighted.

At this point it is important that all the parties are well behaving and do not reveal the shares received from Boo. At this point in the protocol, Secret sharing guarantees that if Mike and Sully put their shares together without knowing Boo's share it is equally likely that it goes to 0 or 1 (dashed line).

Next Mike picks his polynomial  $m(x) = 1 - 3x + x^2$  making sure that  $m(0) = 1$  and giving Boo and Sully their share of the polynomial.

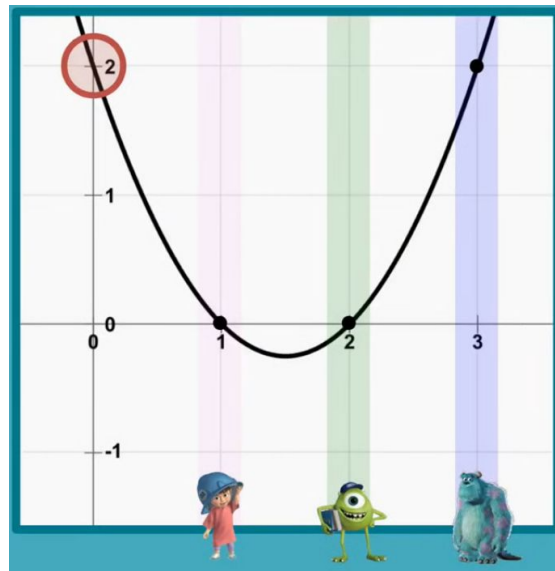


Finally Sulley picks his polynomial  $s(x) = 0 + \frac{3}{2}x - \frac{1}{2}x^2$  satisfying  $m(0) = 0$  and gives Boo and Mike their shares.

So, Boo knows all the shares of  $x=1$ , Mike knows all shares for  $x=2$  and Sully knows for  $x=3$  while not sharing their information with the other parties.



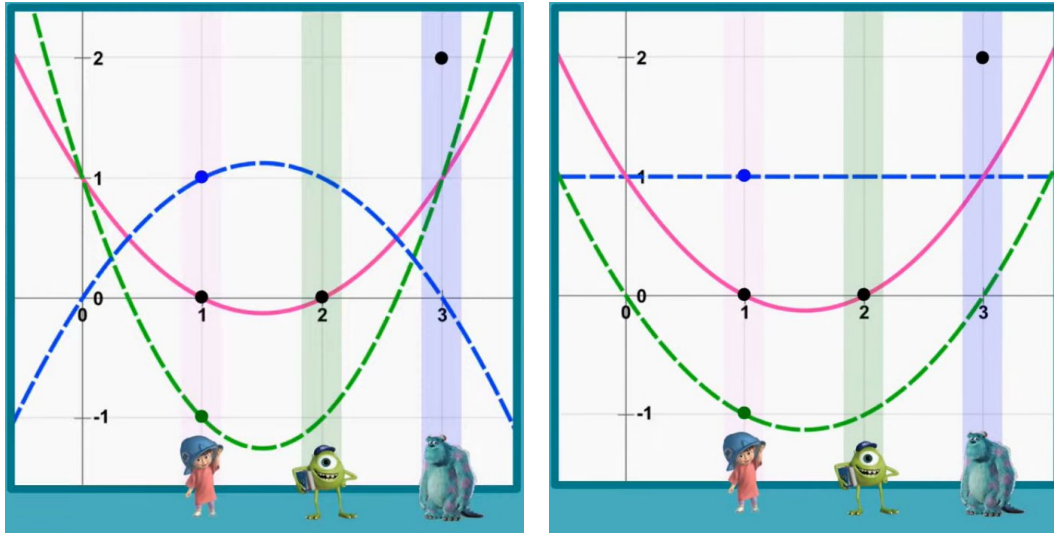
Now, let's recall that we need 3 points of the summed polynomial  $f$  in order to use graph interpolation to find out the shape of  $f$  as well as its value at  $f(0)$  which is the result of the voting, i.e. the outcome. So the question arises that if we have 3 distinct points of the polynomial  $f$ . Remember that from the property of addition of polynomials for each values of  $x$  we can sum the value of the individual polynomial at  $x$  and we get the value of  $f(x)$ . So from the individual shares, Boo can compute  $f(1)$ , Mike can compute  $f(2)$  and Sully can compute  $f(3)$  by simple addition. So, Boo computes  $f(1) = b(1) + m(1) + s(1) = 0 - 1 + 1 = 0$  and reveals the values of adding the shares, so now Mike and Sully know the value of  $f(1) = 0$ . Similarly Mike adds up  $f(2) = b(2) + m(2) + s(2) = 0 + 1 - 1 = 0$  and reveals the value of  $f(2)$ . Finally Sully adds up the values  $f(3) = b(3) + m(3) + s(3) = 1 + 1 + 0 = 2$  and announce the value of  $f(3)=2$ . So, now we now have 3 points for the polynomial  $f$  and by using Lagrange's interpolations to find out the second degree polynomial determined by these points. On doing so we obtain the polynomial of the form  $f(x) = 2 - 3x + x^2$  which has the the form



It is notable that the polynomial has a value 2 at  $f(0)$  hence satisfies our condition i.e. the outcome of the vote and since all three parties now have the 3 points they can now compute this outcome. It is also remarkable to see that the polynomial  $f(x)$  is the sum of the three secret polynomials  $b(x)$ ,  $m(x)$  and  $s(x)$ .

**Cross verification of secrecy :** Let us now cross verify that Boo does not learn that who among Mike and Sully voted 0. At the end of the computation all the information that Boo has is the shares she received from other parties, the value of her own polynomial and the 3 points on the polynomial  $f$ . In the actual realm of the protocol both possibilities are equally likely. It could be that Mike voted 1 and Sully voted 0 and the other way around. This holds true as there exists freedom on the choices of the coefficients which will keep in line with the information that Boo observed.





*Left holds true for Sully voted 0 and Right hold true for Mike voted 0. The dashed line indicate the freedom of choosing coefficients that would lead to different polynomials while still giving the same shares to all of them.*

But the values of the secret polynomial  $f$  at  $x=0$  are now different. So Boo would have to consider both the possibilities side by side and will not be able to confirm who among Mike and Sully voted 0. Hence the property of privacy preservation is conserved.

### Shortcomings :

There are a number of shortcoming to the simplistic approach used in the last example,

1. There is no verification of the secret polynomial used by the different parties, say, any party could vote -2 instead of 0 and 1.
2. Asynchronous sharing : Since the parties announce the share of each party one by one, this makes it possible for the last party to simply lie about the sum's value of the share to ensure that the final outcome sums up to a value to whichever party desires. Sully can announce a different value for  $f(3)$  such that the  $f(0)$  i.e. the voting results is 0.
3. We have only looked at simple addition of polynomials, doing things like multiplications make it more sophisticated as then the degree of the polynomial will then increase.

### Current applications :

An implementation of the authenticated garbling scheme reports malicious security 2PC at over 0.8 million gates per second on a 10Gbps LAN (Wang et al., 2017b). With honest-majority malicious 3PC the performance that can be achieved is even more remarkable. Araki et al.(2017) reported a 3-machine 20-core cluster that can evaluate a billion gates per second over a 10Gbps channel. Despite this progress, and many emerging real world uses, MPC is not yet widespread in practice. There remain several challenges like computation cost (Asharov et al., 2012), leakage trade offs

(Zahur et al.(2015), meaningful trust, to overcome before MPC can be deployed for a wide range of privacy-preserving applications.

## References :

1. A Pragmatic Introduction to Secure Multi-Party Computation, David Evans, Vladimir Kolesnikov, Mike Rosulek
2. Secure Multi-party computation Introduction, Ran Cohen.
3. Secure Multi-Party computation, Oded Goldreich, Final (incomplete) draft, version 1.4), 2002.
4. <https://www.cs.virginia.edu/~evans/pragmaticmpc/pragmaticmpc.pdf>
5. Secure Multiparty computation (MPC), Yehuda Lindell
6. A. C. Yao, "Protocols for secure computations," *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, Chicago, IL, USA, 1982, pp. 160-164
7. Malkhi, D., N. Nisan, B. Pinkas, and Y. Sella. 2004. "Fairplay-Secure Two-Party Computation System". In: USENIX Security Symposium.
8. Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielse, Jakob Pagter, Michael Schwartzbach and Tomas Toft (2008). "Multiparty Computation Goes Live". *Cryptology ePrint Archive* (Report 2008/068).