Chapter 4: Approaches for Private Computation

Lecture PETs4DS:
Privacy Enhancing Technologies for Data Science

Dr. Benjamin Heitmann and Prof. Dr. Stefan Decker
Informatik 5
Lehrstuhl Prof. Decker

RWTH AACHEN UNIVERSITY

Overview of chapter

- **High-level Concepts**
- **Secure Multi-party Computation**
- **Yao's Garbled Circuits**
- **Limitations**

High-level Concepts

Based on:
Yakoubov, Sophia, et al. "A survey of cryptographic approaches to securing big-data analytics in the cloud." *High Performance Extreme Computing Conference (HPEC),* IEEE, 2014.

Motivation

- Cloud computing enables organisations to outsource computation.
  - Enables storage and analysis of data on shared resources.
  - Easy to handle fluctuations in volume and velocity of data
- Total size of market for public cloud services is ~205 billion USD in 2016
- However, cloud computing introduces many new threats:
  - Cloud infrastructure might be provided by untrusted entities.
  - Tampering with data or computation is possible.
- **How can we address these new threats?**
- **Is private computation possible even in the cloud?**

Common Node Roles for Big Data Analytics in Cloud Computing

Many Big Data analytics applications involve the following node roles:

- **Input node (I)**
  - Supplies raw data
  - Examples: aggregated data from sensors or users of a system
- **Compute node (C)**
  - Performs computation on the data
- **Storage node (S)**
  - Store data between computations: input & temporary & output data
- **Result node (R)**
  - Receives computation result
  - Sends result to client or makes automated decision

Running Example: Genomic Data Analysis

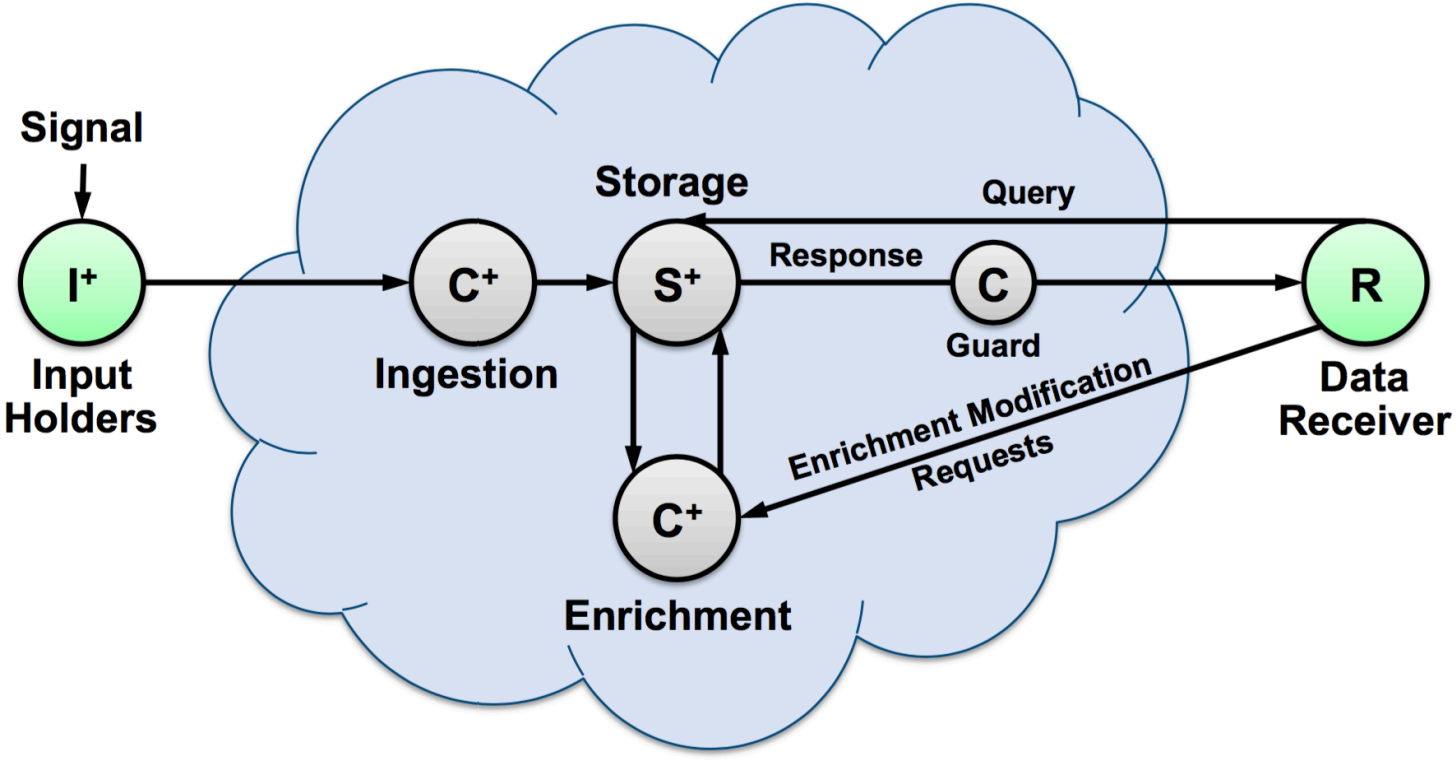
- Sensitive data: reference data sets for genomic sequences.
- Maintained by agencies like National Institutes for Health (NIH) in US

- **Data analytics application:**
  - Allow scientists to check correlation of genetic sequence to reference sequence
  - Correlation can be checked using sequence metadata
  - However, sequence metadata needs to be computed
  - Requires private computation

Running Example: Genomic Data Analysis

- Data is generated by genetic sequencers -> **Input node**
- Genetic sequences need to be stored for use as input / intermediary / output data -> **Storage node**
- The incoming data needs to be pre-processed for organizing it -> **Compute node**
- Meta-data on genetic sequences provides enrichment of data -> **Compute node**
- Output results are presented to scientists -> **Result node**
- Output needs to be checked to control unintended disclosure of data **-> Compute node**

Running Example: Genomic Data Analysis



Desirable Security and Privacy Properties of Cloud Computing

- **Availability**
  - Data owners have access to their data and to computation resources
  - Main goal of cloud infrastructure
  - Already solved by infrastructure
- **Confidentiality**
  - All sensitive data remains secret from adversaries and untrusted entities
  - Corresponds to Disclosure threat in LINDDUN
  - Not guaranteed!
- **Integrity**
  - All outputs of computation are correct.
  - Any unauthorised modification of sensitive data can be traced.
  - Not guaranteed!

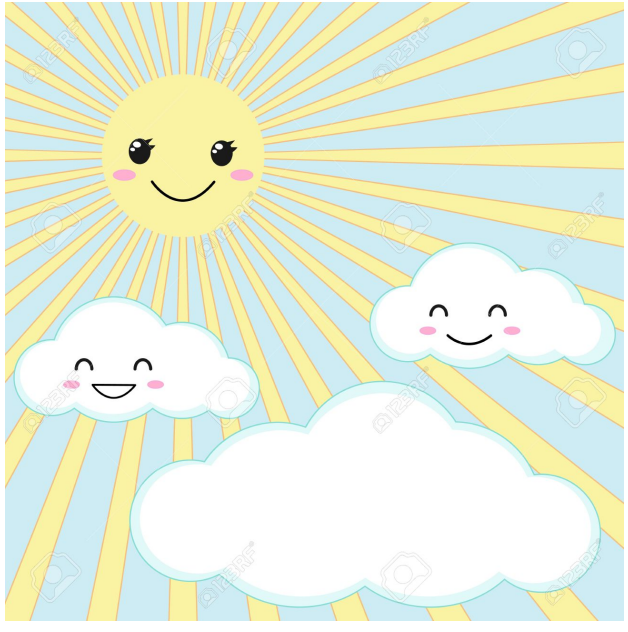
Cloud Computing Scenarios

- **Untrusted Cloud**
  - Cloud provider makes no guarantees
  - Confidentiality or integrity of data might not be maintained by cloud nodes
  - Most common type of cloud infrastructure today, e.g. Amazon Web Services (AWS)
  - Any kind of adversary is possible
  - Corresponds to public cloud deployment model

Cloud Computing Scenarios

- **Trusted Cloud**
  - Cloud is completely controlled by organisation (it is in-house / on-site)
  - Air-gap between cloud and outside network
  - Clients data stays confidential and can not leave cloud
  - Cloud nodes can still be corruputed
  - Adversary can threaten data integrity
  - Very common in government use cases

  - Corresponds to private cloud deployment model



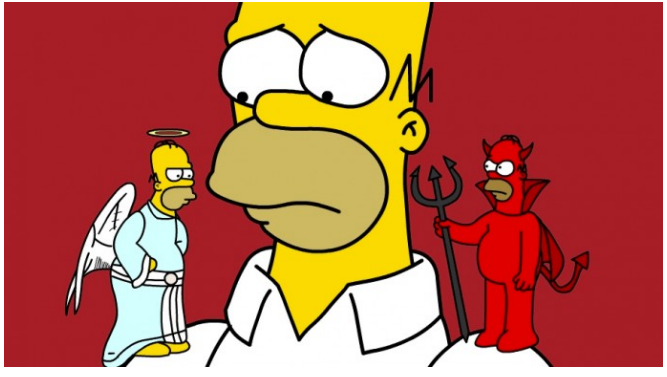
Cloud Computing Scenarios

- **Semi-Trusted Cloud**
  - Client can not fully trust the cloud
  - Not the whole cloud is malicious
  - Parts of cloud could be controlled by adversary
  - Common when cloud provider maintains security, but does not guard against other threats
  - Corresponds to public & private & hybrid cloud deployment model

Adversary types

The two most important types of adversary

- **Honest-but-curious (HBC) adversary**
  - Controlled nodes perform all computation and protocol like honest node
  - Adversary tries to learn additional information
  - Can combine information from multiple controlled nodes
  - Can learn information which normally no single party can learn



- **Malicious adversary**
  - Controlled nodes deviate arbitrarily from computation / protocol.
  - Examples: send malformed messages, incorrect computation, active collusion.
  - Active effort to violate confidentiality or integrity.

- A single adversary could control multiple nodes

Approaches to Achieve Computational Privacy

- **Homomorphic Encryption (HE)**
  - Encrypt data before sending to cloud
- **Verifiable Computation (VC)**
  - Prove that results are correct
- **Secure Multi-Party Computation (SMPC)**
  - Compute results together without sharing private data

Homomorphic Encryption: Use Case and Cloud Scenario

- **Genomic data analytics example**
  - NIH wants to outsource computation to Amazon Elastic Compute Cloud (EC2)
  - Amazon EC2 provides an untrusted cloud
  - No guarantees regarding confidentiality

- **How to enable private computation on untrusted cloud?**
  - Is it possible to use encrypted data for computation without decrypting it?

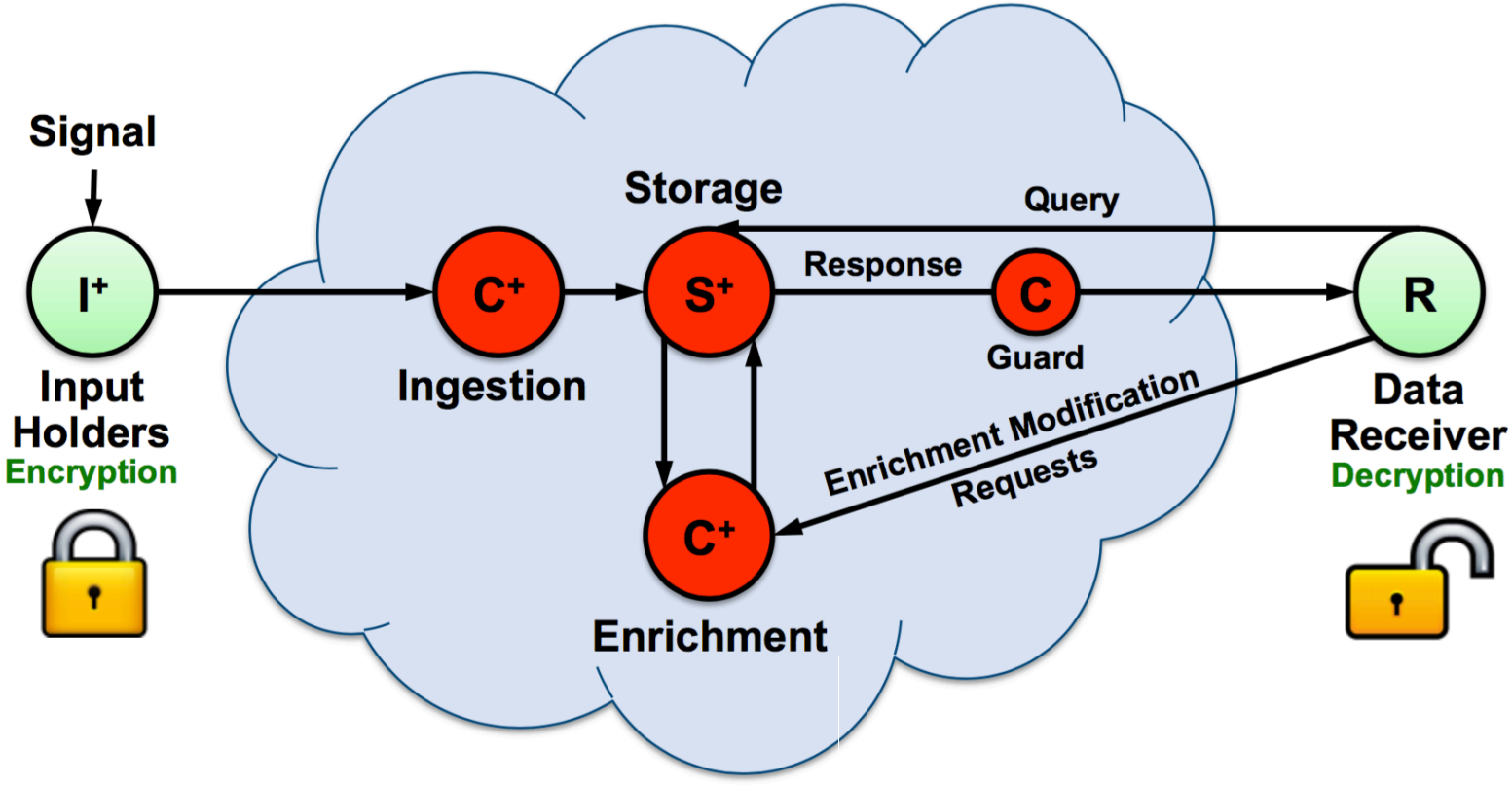
Homomorphic Encryption: Use Case and Cloud Scenario

- **Homomorphic encryption:**
  - Encryption of message m under key k: $E_k(m)$
  - Decryption under key k: $D_k(\,)$
  - **Requirement for homomorphic encryption:**
    For function f there is f' such that $D_k\big(f'(E_k(m))\big) = f(m)$

- Fully Homomorphic Encryption (FHE) enables any kind of computation
- Somewhat Homomorphic Encryption (SHE) enables only some operations, e.g. multiplication

- **Limitations:**
  - Slow: matrix-vector multiplication for 256 integers takes 26 seconds [HElib]
  - Only one key: all input and result nodes have to share the same key

Homomorphic Encryption: Illustration



Homomorphic Encryption: Overview of Properties

- **Confidentiality:** yes
  - Compute and storage nodes do not have access to the unencrypted data
- **Integrity:** no
  - Modification of data is possible
  - Faulty computation is possible
- **Adversary type addressed:** malicious
- **Requires interaction:** no
  - HC is possible with only one party providing data

Verifiable Computation: Use Case and Cloud Scenario

- **Genomic data analytics example**
  - NIH has funding to build its own cloud with no connection to outside networks (air gap)
  - No data can leave this private cloud
  - Confidentiality of data is automatically guaranteed
  - Private cloud can be compromised, e.g. through malware or supply chain attack
  - Integrity of data and computation still needs to be protected
- **Is it possible to guarantee integrity in a private cloud?**
- Solved for data storage, e.g. with checksums and broadcasting of checksums
- Can we guarantee integrity of computation results ?

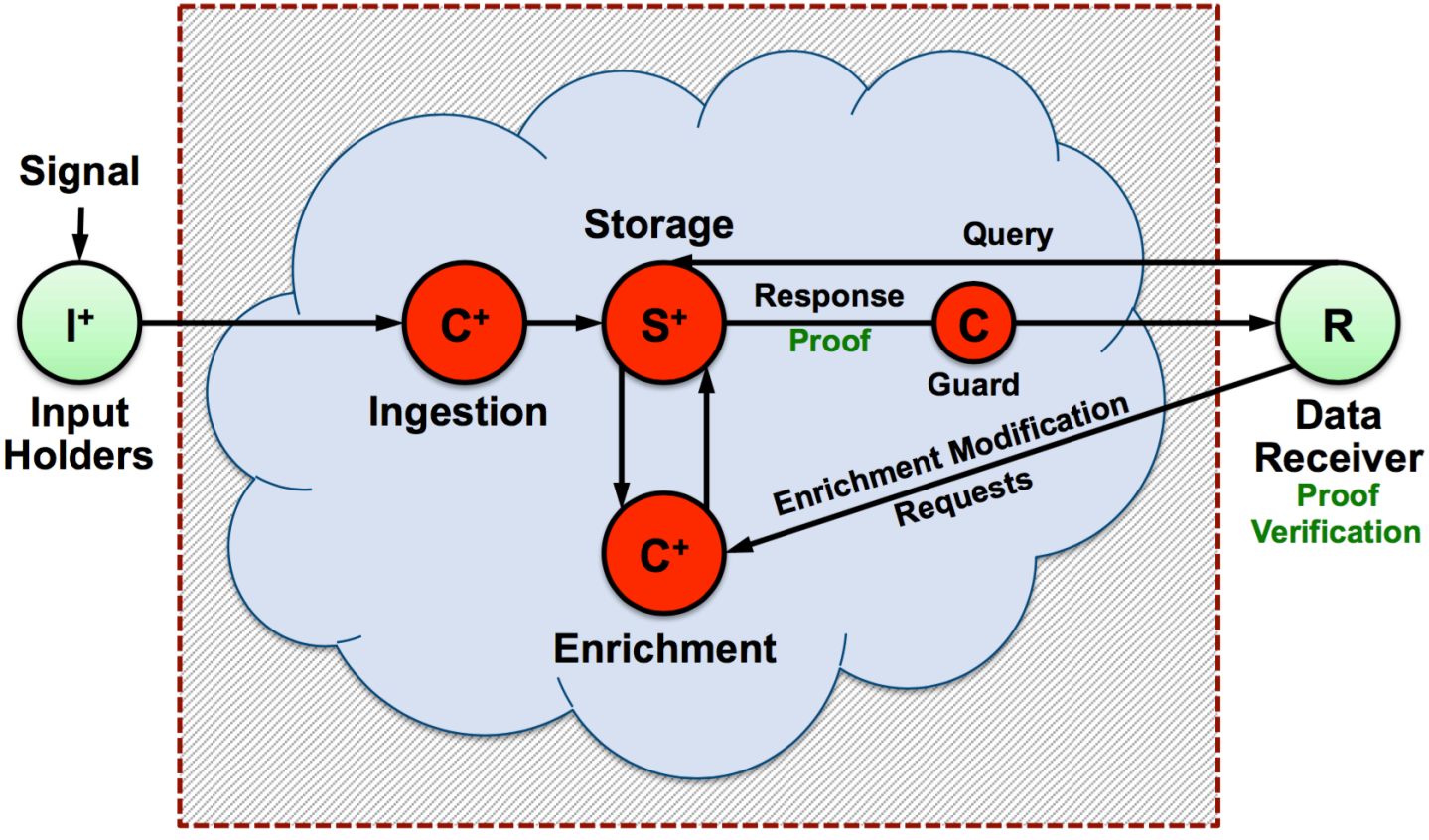
Verifiable Computation: Details

- Verifiable computation allows data owner to check integrity of computation.

- **How does verifiable computation work?**
  - **Data owner** gives data with specification of computation to *prover*
  - **Prover** has more computation resources than data owner
  - Prover performs computation as specified
  - Prover returns result of computation together with proof of correctness.
  - Data owner then verifies proof.
  - VC requires proofs to be easier to verify than the proven computation.
  - Details of such proofs not covered by this lecture.

- **Limitations:**
  - VC does not involve encryption of data.
  - Slow: Computation is usually faster then constructing proof.
  - Pinocchio library: computation with 277.000 multiplications, construction of proof takes 144 sec, verification of proof takes 10 ms.

Verifiable Computation: Illustration



Verifiable Computation: Overview of Properties

- **Confidentiality:** no
  - If VC is used in a private cloud, confidentiality is already guaranteed by isolation of cloud
- **Integrity:** yes
  - Computation nodes can prove correctness of results
- **Adversary type addressed:** malicious
- **Requires interaction:** no

Secure Multi-Party Computation: Use Case and Cloud Scenario

- **Genomic data analytics example**
  - After building a private cloud, the NIH discovers they need even more computing resources
  - Private cloud is expanded with computation nodes from public cloud
  - Result is a semi-trusted cloud.
  - Not all nodes will be controlled by adversary.
  - However, both HBC and malicious adversaries are possible.

- **Is it possible for multiple nodes to collaboratively perform a computation?**
- **If yes, can this be done without actually sharing the secret data?**

Secure Multi-Party Computation: Details

How does SMPC work?

- All nodes split their secret data into shares.
- Shares are distributed to the other nodes.
- Every node then calculates intermediate results and shares them with all other nodes.
- This allows verifying the correctness of intermediate results.
- Each node then can use the public intermediate results with his private shares to determine result.
- All nodes will get the same result.

- No node learns something they don’t already know, except for the result of the computation.

Secure Multi-Party Computation: Limitations

Limitations of SMPC:

- Slow, but not as much overhead incurred as for HE
- Requires more than two parties, but does not scale well with number of parties

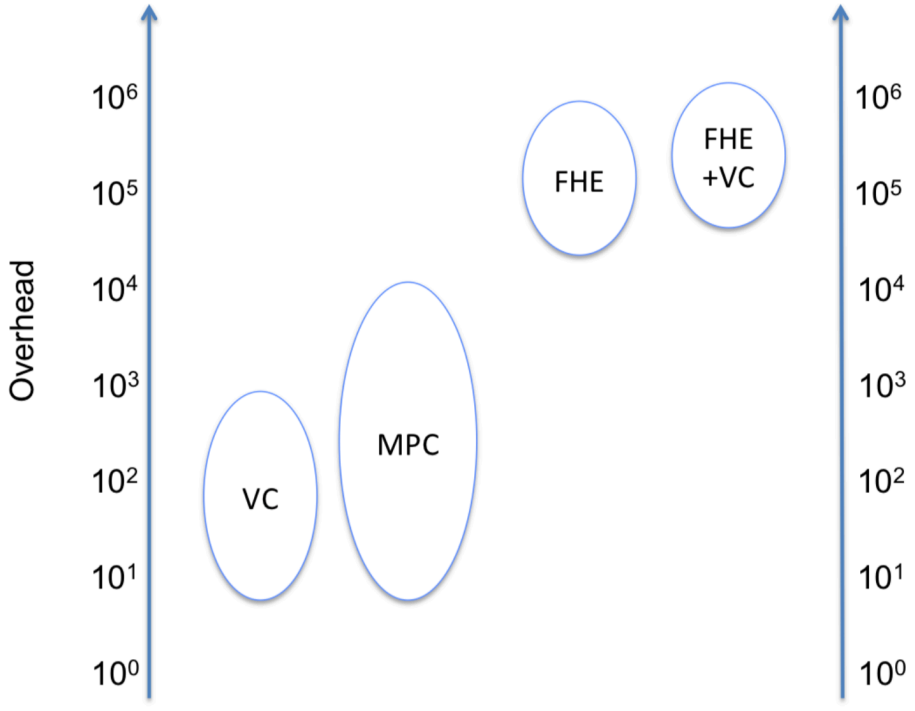
Secure Multi-Party Computation : Overview of Properties

- **Confidentiality:** yes
  - No node learns anything new, besides the result of the computation.
- **Integrity:** yes
  - Intermediate results have to be shared, which allows verification of correctness.
- **Adversary type addressed:** malicious and HBC
  - Most SMPC schemes allow $t < \frac{n}{2}$ HBC adversaries or $t < \frac{n}{3}$ malicious adversaries, but higher bounds also possible.
- **Requires interaction:** yes
  - SMPC requires 3 nodes or more.
  - Confidentiality and integrity of SMPC are enforced through interaction between nodes.

Comparison of Approaches for Private Computation

Approach	Adversary Type	Confidentiality	Integrity	Requires Interaction
Homomorphic Encryption (HE)	Malicious	YES	NO	NO
Verifiable Computation (VC)	Malicious	NO	YES	NO
Secure Multi-Party Computation (SMPC)	Honest-but-curious (HBC) or Malicious	YES	YES	YES

Comparison of Performance Overhead Incurred by Approaches



Graphical depiction of the multiplicative performance overheads over unsecured computation incurred by HE, VC and multi-party computation (MPC).

MPC and SMPC are the same.

Summary

- **Cloud computing** provides shared resources for computation
- Different cloud deployment scenarios have different **confidentiality and integrity requirements**
- **Adversaries** can be honest but curious or malicious.
- Three promising **approaches to address threats** in cloud computing are:
  - Homomorphic encryption (HE)
  - Verifiable computation (VC)
  - Secure Multi-Party Compuation (SMPC)

- Each approach addresses different **requirements**, and provides different **guarantees**
- All approaches incur significant **performance overheads**
- SMPC is the most **promising** approach with the least overhead

Secure Multi-Party Computation

based on:
Cramer, Ronald, Ivan Damgard, and Jesper Buus Nielsen. "Secure multiparty computation and secret sharing." *Cambridge University Press* (2015).

For the purpose of this lecture, the terms ”Secure Multi-Party Computation” and “Multi-Party Computation” are the same. These ideas have a history of approx. 30 years, and there was a shift from the term MPC to SMPC over time.

Motivation for Secure Multi-Party Computation

- **There are many situations with the following properties:**
  - Several parties are involved
  - Every party has a secret
  - Confidential information from all parties needs to be combined
  - All parties need to compute a function on the input of all secrets
  - No party trusts any of the other parties
- **Ideally, all parties want to obtain same result of a function without sharing any secrets.**
- Lets look at some examples…

Motivating Example: Auctions

- **Most simple form of auction:**
  - An item is for sale.
  - The highest bid wins.
- **Requirement for fair auction:**
  - The maximum amount of every bidder has to stay secret.
  - If the auctioneer would know it, he has incentives to use this information to get more money.
- **However, the result of the auction could be computed without central broker:**
  - Input for each bidder: highest possible bid
  - Output: winner of auction and the price he would pay (2nd highest bid)

Motivating Example: Procurement

- Inverted auction
- Public institution asks for bids on a public contract
  - E.g. new university buildings

- The lowest bid wins (“cheapest offer”)

- **Bidder motivation:**
  - Bidders want to get largest price
  - Bidders also don’t want competitors to know their bid
- Institution also does not want bidders to communicate about bids

- **Result** of process can be easily computed with true values of all bids

Motivating Example: Benchmarking

- Companies in the same line of business want to know if they are doing well in **comparison**.
  - **Metrics** for this could be: profit relative to size, average salaries, productivity.
- **Benchmark analysis** takes this data from several participating companies
  - Data is used to determine “ideal” performance
  - Results of comparison between each company and ideal is reported
- No company in a benchmark wants their metrics to be **leaked** to competition
- However, a **central party** can easily compute benchmark given all private data.

Motivating Example: Voting

- **Every voter has one vote**
  - Input per voter is either 0 or 1
- **Function to be calculated:**
  - Sum of all inputs
- **Requirement for a fair vote:**
  - Everybody only votes once
  - Every vote stays secret
  - Nobody knows the vote of anybody else
- **Is this possible without a central broker ?**

Do we have to trust somebody?

- **How are these situations handled currently?**
  - All parties agree on a trusted central broker
  - The broker gets all the secrets and informs all parties of the result.
  - Then the broker destroys the data and forgets the result.
- **Of course, in practice this is very hard to realise.**
  - The broker is a high value target for attackers.
  - Payment to broker is often higher than value of selling data.
- **Fundamental question:**
  Can this kind of computation be done without relying on a trusted party?

Secure Multi-Party Computation (SMPC): Pre-conditions and Assumptions

- The parties or players are called $P_1, \ldots, P_n$
- Each player P_i holds secret input x_i
- All players agree on a function f that takes n inputs

- Goal: compute $y = f(x_1, \ldots, x_n)$ while satisfying the following two conditions:
  1. **Correctness:** the correct value of y is computed
  2. **Privacy:** y is the only new information that is released.
- **Computing f securely** means achieving correctness and privacy at the same time

- Other assumptions (for now):
  - All players follow the given protocol.
  - Any pair of players can communicate securely.

Implementing Voting through Secure Addition

- Simple special case:
  - Each x_i is a natural number
  - $f(x_1, \ldots, x_n) = \sum_{i=1}^{n} x_i$

- This simple function has many **meaningful applications**. For instance:
  - Players user their input to **vote** on a decision.
  - $x_i = 0$ means **no** and $x_i = 1$ means **yes**

- **Secure computation of f** gives us the expected properties of a paper vote:
  - **Correctness:** the sum is correct.
  - **Private:** NO information besides the result is leaked, and NO information on the vote of a player is leaked.

Secret Sharing

- Provides way for any party spread their secret to all players
- Together they hold full information about s (sufficient to reconstruct s)
- Yet, no player has any information about s on their own

Steps for P_1 to share his secret s with P_2 and P_3:

- Choose prime p and define the finite field $\mathbb{Z}_p = \{0,1, \dots, p-1\}$
- P_1 chooses numbers r_1, r_2 uniformly at random in $\mathbb{Z}_p$
- Then set $r_3 = s - r_1 - r_2 \; mod \; p$
- Alternative: P_1 chooses r_1, r_2, r_3 uniformly at random so that $s = r_1 + r_2 + r_3 \; mod \; p$

- Now P_1 sends the following messages using a private channel:
  - P_1 sends r_1, r_3 to P_2
  - P_1 sends r_1, r_2 to P_3
  - P_1 sends r_2, r_3 to P_1 (himself)

Informal discussion of Privacy of Addition Protocol

Did any information about s get leaked besides the shares?

- Neither P_2 nor P_3 have any idea about value of s
- Situation for P_2:
  - P_2 knows r_1, r_3
  - P_2 does **not** know r_2
  - P_2 knows $s = r_1 + r_2 + r_3 \, mod \, p$
  - Any $r_2 \in \mathbb{Z}_p$ is equally likely to be missing share
  - Any $s \in \mathbb{Z}_p$ is equally likely to be the secret
  - In summary: P_2 gained no new knowledge besides r_1, r_3
- The same applies to P_3
- **This secret sharing schema is private**

Informal discussion of Correctness of Addition Protocol

- Can we check if reconstructing the secret results in the correct number?

- Yes, if all parties pool their information
- Every party sends all other parties all their shares
- Having access to all three shares allows reconstructing the secret:

$$s = r_1 + r_2 + r_3 \; mod \; p$$

- All parties can check the shares which the others are sending
- All parties can compare value of s

- **This secret sharing schema is correct**

Protocol for Secure Addition

Protocol Secure Addition

Participants are $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$, input for P_i is $x_i \in Z_p$, where p is a fixed prime agreed upon in advance.

1. Each P_i computes and distributes shares of his secret x_i as described in the text: he chooses $r_{i,1}, r_{i,2}$ uniformly at random in $\mathbb{Z}_p$, and sets $r_{i,3} = x_i - r_{i,1} - r_{i,2} \bmod p$.
2. Each P_i sends privately $r_{i,2}, r_{i,3}$ to P_1, $r_{i,1}, r_{i,3}$ to P_2, and $r_{i,1}, r_{i,2}$ to P_3 (note that this involves P_i sending "to himself"). So P_1, for instance, now holds $r_{1,2}, r_{1,3}$, $r_{2,2}, r_{2,3}$ and $r_{3,2}, r_{3,3}$.
3. Each P_j adds corresponding shares of the three secrets – more precisely, he computes, for $\ell \neq j$, $s_\ell = r_{1,\ell} + r_{2,\ell} + r_{3,\ell} \bmod p$, and announces s_ℓ to all parties (hence two values are computed and announced).
4. All parties compute the result $v = s_1 + s_2 + s_3 \bmod p$.

An Example using the Protocol for Secure Addition

- Choose prime $p = 11$ with $\mathbb{Z}_{11} = \{0,1, \dots, 10\}$

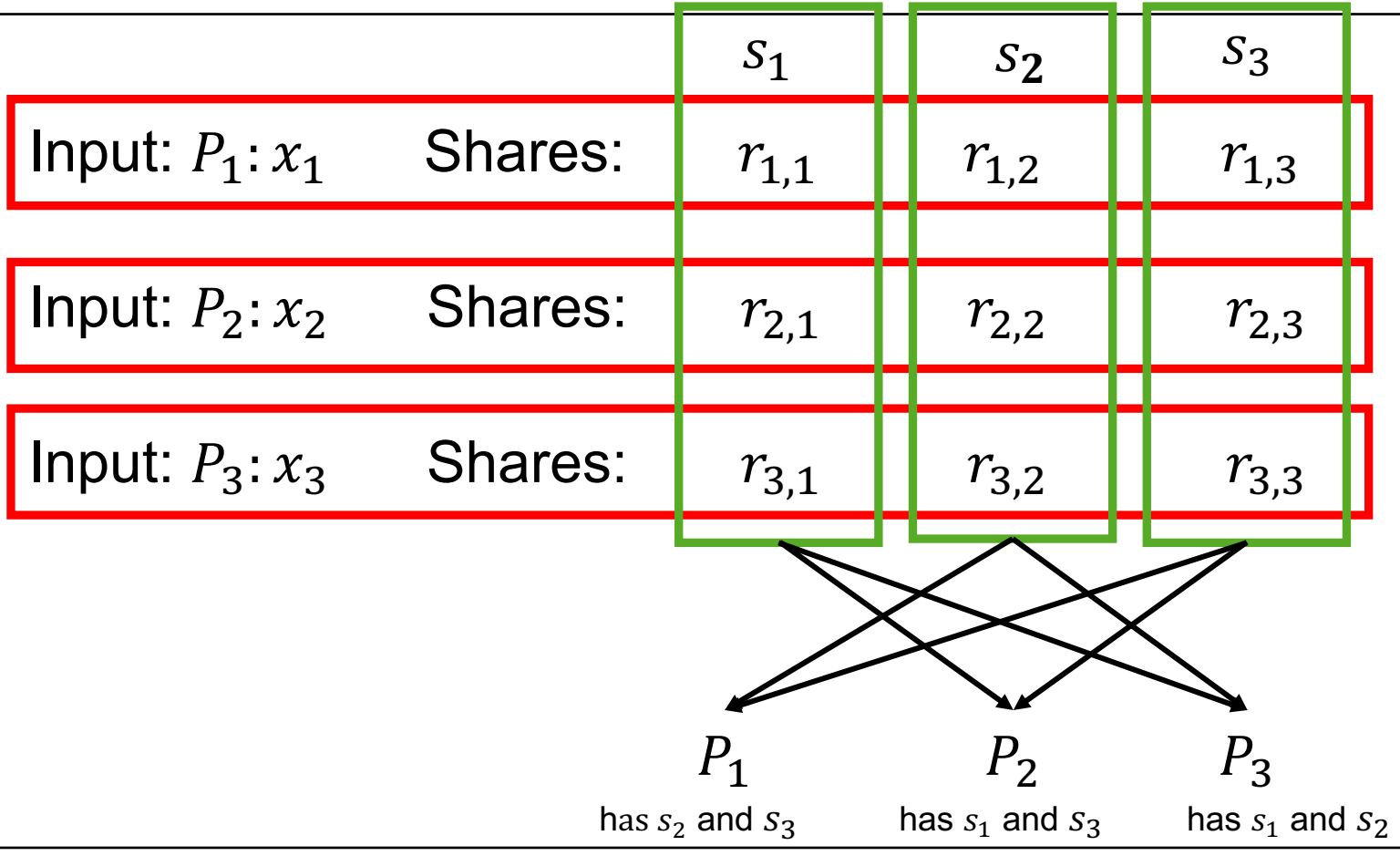
- Input $P_1: x_1 = 1$
  - P_1 chooses $r_{1,1}$, $r_{1,2}$ uniformly at random in $\mathbb{Z}_p$
  - $r_{1,1} = 2$
  - $r_{1,2} = 7$
  - $r_{1,3} = x_1 - r_{1,1} - r_{1,2} \; mod \; p = 1 - 2 - 7 \; mod \; 11 = 3$

- Input $P_2: x_2 = 0$
  - P_2 chooses $r_{2,1}$, $r_{2,2}$ uniformly at random in $\mathbb{Z}_p$
  - $r_{2,1} = 3$
  - $r_{2,2} = 6$
  - $r_{2,3} = x_2 - r_{2,1} - r_{2,2} \; mod \; p = 0 - 3 - 6 \; mod \; 11 = 2$

An Example using the Protocol for Secure Addition

- Input $P_3: x_3 = 0$
  - P_3 chooses $r_{3,1}$, $r_{3,2}$ uniformly at random in $\mathbb{Z}_p$
  - $r_{3,1} = 5$
  - $r_{3,2} = 9$
  - $r_{3,3} = x_3 - r_{3,1} - r_{3,2} \; mod \; p = 0 - 5 - 9 \; mod \; 11 = 8$

Distribution of Shares for Secure Addition



Results for the Secure Addition Example

Each player computes his two s_i "verification shares":

- $s_1 = r_{1,1} + r_{2,1} + r_{3,1} \; mod \; p = 2 + 3 + 5 \; mod \; 11 = 10$
- $s_2 = r_{1,2} + r_{2,2} + r_{3,2} \; mod \; p = 7 + 6 + 9 \; mod \; 11 = 0$
- $s_3 = r_{1,3} + r_{2,3} + r_{3,3} \; mod \; p = 3 + 2 + 8 \; mod \; 11 = 2$

- Then each player broadcasts his s_i to all other players
- Each player gets send two versions of every s_i
- This allows verifying that the shares are correct and that the result will be correct

Now every player can compute the result s of the addition locally:

- $s = s_1 + s_2 + s_3 \; mod \; 11 = 10 + 0 + 2 \; mod \; 11 = 1$

Result: all players know that there was only one yes vote

Secure Multiplication: The Matchmaking Scenario

- Useful application for secure multiplication:
  - Two players: Alice and Bob
  - Alice wants to know if Bob wants to go on a date with her
  - $x_i = 0$ means **no** and $x_i = 1$ means **yes**
  - Both players want to avoid embarrassment from rejection
  - Both players provide the input for $f(a, b) = ab \ mod \ p$
  - Plausible deniability regarding negative result: could be one zero input, could be two zeros.
- Basically, this could replace Tinder as central broker with a peer to peer protocol.
- We can build a protocol for secure multiplication based on the addition protocol
  - But it needs a third player

Idea for secure multiplication

- Pre-conditions:
  - $a, b \in \mathbb{Z}_p$ have been secretly shared to three players as described before
  - $a = a_1 + a_2 + a_3 \; mod \; p$
  - $b = b_1 + b_2 + b_3 \; mod \; p$

- Could we compute a,b if we have all shares but not a,b themselves ?

$$ab = a_1b_1 + a_1b_2 + a_1b_3 + a_2b_1 + a_2b_2 + a_2b_3 + a_3b_1 + a_3b_2 + a_3b_3 \bmod p \; .$$

P_3 can compute these products

P_2 can compute these products

P_1 can compute these products

- Every player can locally compute one number and then all players use the secure addition protocol to determine the sum.

The Protocol for Secure Multiplication

Protocol SECURE MULTIPLICATION

Participants are $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$, input for P_1 is $a \in \mathbb{Z}_p$, input for P_2 is $b \in \mathbb{Z}_p$, where p is a fixed prime agreed upon in advance. P_3 has no input.

1. P_1 distributes shares a_1, a_2, a_3 of a, while P_2 distributes shares b_1, b_2, b_3 of b.
2. P_1 locally computes $u_1 = a_2b_2 + a_2b_3 + a_3b_2 \bmod p$, P_2 computes $u_2 = a_3b_3 + a_1b_3 + a_3b_1 \bmod p$, and P_3 computes $u_3 = a_1b_1 + a_1b_2 + a_2b_1 \bmod p$.
3. The players use Protocol SECURE ADDITION to compute the sum $u_1 + u_2 + u_3 \bmod p$ securely, where P_i uses u_i as input.

An Example using the Protocol for Secure Multiplication

- Choose prime $p = 11$ with $\mathbb{Z}_{11} = \{0,1, \dots, 10\}$

- Input Alice: $P_1: a = 1$
  - P_1 chooses a_1, a_2 uniformly at random in $\mathbb{Z}_p$
  - $a_1 = 9$
  - $a_2 = 1$
  - $a_3 = 1 - 9 - 1 \mod 11 = 2$

- Input Bob: $P_2: b = 1$
  - P_2 chooses b_1, b_2 uniformly at random in $\mathbb{Z}_p$
  - $b_1 = 7$
  - $b_2 = 10$
  - $b_3 = 1 - 7 - 10 \mod 11 = 6$

An Example using the Protocol for Secure Multiplication

- P_1 gets shares b_2, b_3, and already has all shares of a.
- P_2 gets shares a_1, a_3, and already has all shares of b.
- P_3 gets shares a_1, a_2, b_1, b_2 and has no other shares.

- P_1 locally computes $u_1 = a_2b_2 + a_2b_3 + a_3b_2 \; mod \; p$
- P_2 locally computes $u_2 = a_3b_3 + a_1b_3 + a_3b_1 \; mod \; p$
- P_3 locally computes $u_3 = a_1b_1 + a_1b_2 + a_2b_1 \; mod \; p$

- Now all players perform the secure addition protocol for input u_i of party P_i to determine $u = u_1 + u_2 + u_3 \; mod \; p$

What if Players Do Not Follow Instructions

Lets look at some attacks on these two simple protocols:

- **Collusion:**
  - Players just need information about one missing share.
  - Two players could easily collude → no protection from honest-but-curious adversaries.

- **Choice of inputs:**
  - In matchmaking scenario, Alice could provide input 1, even if she is not interested.
  - Then she learns the input of Bob and breaks his privacy.
  - Bob can **not protect against this.**
  - For SMPC we have to assume that all players have incentive/motivation to provide meaningful and truthful input.

What if Players Do Not Follow Instructions

Deviation from protocol:

1. A player could choose shares which don't express his secret input
   - That means the player is sharing a different number
   - No protection against players arbitrarily choosing their input

2. When a player sends his shares to the other player, he might modify a share he sends, e.g. $r_{1,1} \neq r'_{1,1}$
   - This is easily detected when each party broadcasts their verification shares s_i
   - This will result in unequal verification shares $s_i \neq s'_i$

3. When broadcasting verification shares, one player could modify an s_i
   - Same reasoning: every s_i is broadcast by more than one player, so this can be detected

- **Summary:** Deviation of a single party from the protocol can be detected by the other two players.

What to Improve for the Simple Addition / Multiplication Protocol ?

Towards General Solutions

- We have learned simple protocols for secure addition and multiplication
- However, there were strong assumptions:
  - Exactly 3 parties
  - Players always follow the protocol
  - Only a single player can be corrupt

SMPC Protocol with Passive Security (CEPS)

SMPC Protocol with Passive Security

Properties of the protocol:

- Allows any number of players $n \geq 3$
- Protocol works as long as $t < n/2$ (less then half of nodes are colluding)
- We still assume that all players follow protocol.

- This is called **semi-honest or passive security.**

Shamir's Secret Sharing Scheme

- In order to allow more than 3 parties, we need a **different way to share secrets.**

Requirements:

- Shamir's secret sharing scheme is based on polynomials in a finite field $\mathbb{F}$
- The only necessary restriction is that $|\mathbb{F}| > n$ (n is the number of players)
- We will assume that $\mathbb{F} = \mathbb{Z}_p$ for some prime $p > n$

Shamir's Secret Sharing Scheme

Idea for sharing a secret:

- The value $s \in \mathbb{F}$ is shared by choosing a random polynomial $f_s(X) \in \mathbb{F}[X]$
- The degree of $f_s(X)$ is at most t (t is the number of HBC adversaries tolerated)
- The polynomial is chosen so that $f_s(0) = s$
- Each player P_i gets sent the share $s_j = f_s(j)$

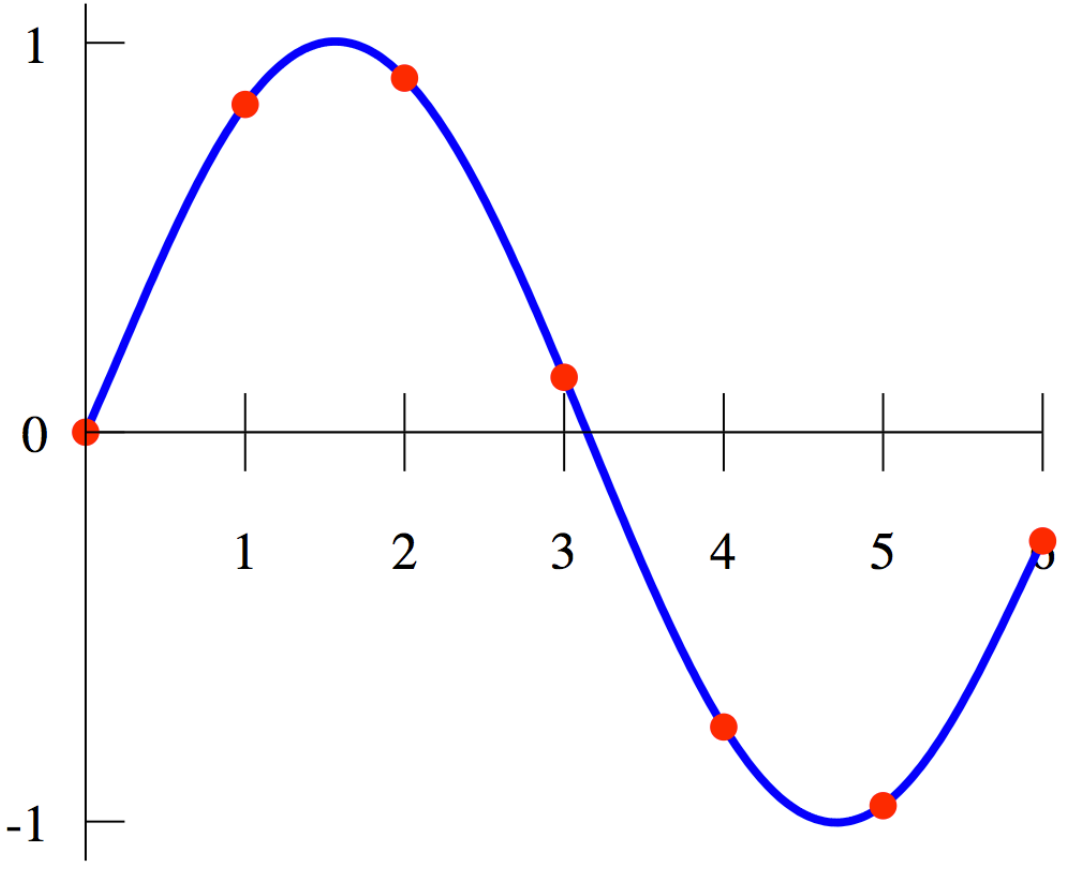
Properties of this secret sharing scheme:

- Any set of t or fewer shares contains no information on secret s
- Any t+1 shares allow reconstructing secret s

This is achieved by using **Lagrange interpolation**

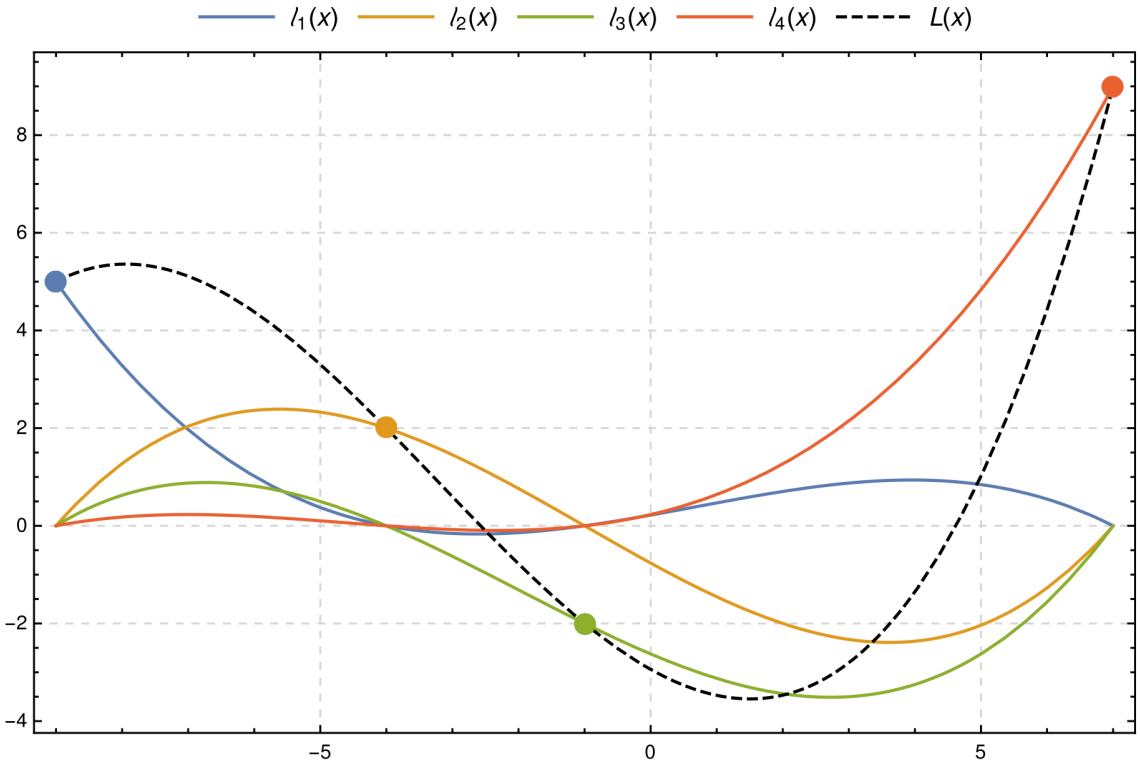
Illustration of Lagrange Interpolation: Goal

- Goal: we want a polynomial which goes through all the red dots.



Intuition for Lagrange Interpolation: How to Construct the Polynomial

- In this example, the **dashed line** shows a polynomial which goes through 4 points: $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$.
- We can interpolate this polynomial with **4 other** polynomials.
- Each colored line shows a polynomial with the properties:
  - $l_i(x_i) = 1$
  - $l_i(x_j) = 0$, for $i \neq j$



The Lagrange Interpolation of L(x) is $L(x) = \sum_{i=1}^{4} L(x)\delta_i(x) = \sum_{i=1}^{4} l_i(x)$

How do we find the $\delta_i(x)$?

Intuition for Lagrange Interpolation

- Every $\delta_i(x)$ should fulfill
  - $\delta_i(x_i) = 1$ for $i = j$
  - $\delta_i(x_j) = 0$ for $i \neq j$
- Second condition: $(x - x_2)(x - x_3) \dots (x - x_n) = 0, \forall x \in \{x_2, \dots, x_n\}$
- First condition: $\frac{(x - x_2)(x - x_3) \dots (x - x_n)}{(x_1 - x_2)(x_1 - x_3) \dots (x_1 - x_n)} = 1$ for $x = x_1$
- In $\mathbb{R}$ we get the the formula: $\delta_i(x) = \prod_{\substack{0 \leq m \leq k, \\ m \neq j}} \frac{x - x_m}{x_j - x_m}$

Lagrange Interpolation in a Finite Field

- Now, use only discreet numbers, as we are in $\mathbb{F}$, so $x_j = j$
- We get the following formula for $\delta_i(x)$:

$$\delta_i(\mathbf{X}) = \prod_{j \in C, j \neq i} \frac{\mathbf{X} - j}{i - j} .$$

- With $C = \{j \mid j \in \mathbb{F}\}$ and $|C| = n + 1$ at most

Lagrange Interpolation Without a Polynomial

- We don't actually need to approximate a polynomial:

$$\text{Given any set of values } \{y_i \in \mathbb{F} |\ i \in C\},$$

$|C| = l + 1$, we can construct a polynomial h with $h(i) = y_i$ of degree at most l as

$$h(\mathtt{X}) = \sum_{i \in C} y_i \delta_i(\mathtt{X}) \ .$$

- With $\delta_i(\mathtt{X}) = \prod_{j \in C, j \neq i} \frac{\mathtt{X} - j}{i - j}$.

Recovering the value of h(0)

- The value of h(0) can be recovered with any t + 1 pairs of (x_i, y_i) for a polynomial of degree t
- There is a vector r, such that $h(0) = \sum_{i=1}^{n} r_i h(i)$

  with $r_i = \delta_i(0)$.

- This r=(r_1, … , r_n) is called the recombination vector.

- $\delta_i(X)$ and $\delta_i(0)$ do not depend on h(X).
- The same recombination vector works for all polynomials with the same degree and the same set C.
- Therefore the recombination vector is public information which all players can compute.

Shamir's Secret Sharing Scheme – Example 1

The Setup for Sharing a Secret with Five Players

We look at an example. Assume that we have five parties $\mathsf{P}_1, \ldots, \mathsf{P}_5$ and that we want to tolerate $t = 2$ corrupted parties. Assume that we work in $\mathbb{F} = \mathbb{Z}_{11}$ and want to share $s = 7$. We pick $a_1, a_2 \in_{\mathrm{R}} \mathbb{F}$ uniformly at random, say they become $a_1 = 4$ and $a_2 = 1$, and then we define

$$h(\mathtt{X}) = s + a_1\mathtt{X} + a_2\mathtt{X}^2 = 7 + 4\mathtt{X} + \mathtt{X}^2 \ . \tag{3.2}$$

Shamir's Secret Sharing Scheme – Example 2

Sharing the secret

$$h(\mathtt{X}) = s + a_1\mathtt{X} + a_2\mathtt{X}^2 = 7 + 4\mathtt{X} + \mathtt{X}^2 \ .$$

Then we compute $s_1 = h(1) = 7+4+1 \bmod 11 = 1$, $s_2 = h(2) = 19 \bmod 11 = 8$, $s_3 = h(3) = 6$, $s_4 = h(4) = 6$, $s_5 = h(5) = 8$. So, the sharing is

$$[s] = (1, 8, 6, 6, 8) \ .$$

Shamir's Secret Sharing Scheme – Example 3

Computing $\delta_3(X)$

Assume now that someone is given just the shares s_3, s_4, s_5. Since $3 > 2$, she can use Lagrange interpolation to compute the secret.

We first compute

$$\delta_3(\mathtt{X}) = \prod_{j=4,5} \frac{\mathtt{X} - j}{3 - j} = \frac{(\mathtt{X} - 4)(\mathtt{X} - 5)}{(3 - 4)(3 - 5)} = (\mathtt{X}^2 - 9\mathtt{X} + 20)((3-4)(3-5))^{-1} \pmod{11}.$$

We have that $(3-4)(3-5) = 2$ and $2 \cdot 6 \bmod 11 = 1$, so $((3-4)(3-5))^{-1} \bmod 11 = 6$. So,

$$\delta_3(\mathtt{X}) = (\mathtt{X}^2 - 9\mathtt{X} + 20)6 = (\mathtt{X}^2 + 2\mathtt{X} + 9)6 = 6\mathtt{X}^2 + 12\mathtt{X} + 54 = 6\mathtt{X}^2 + \mathtt{X} + 10 \pmod{11}.$$

Shamir's Secret Sharing Scheme – Example 4

Checking the validity of $\delta_3(X)$

$$\delta_3(\mathtt{X}) = \prod_{j=4,5} \frac{\mathtt{X} - j}{3 - j} = \frac{(\mathtt{X} - 4)(\mathtt{X} - 5)}{(3 - 4)(3 - 5)}$$

$$\delta_3(\mathtt{X}) = (\mathtt{X}^2 - 9\mathtt{X} + 20)6 = (\mathtt{X}^2 + 2\mathtt{X} + 9)6 = 6\mathtt{X}^2 + 12\mathtt{X} + 54 = 6\mathtt{X}^2 + \mathtt{X} + 10 \pmod{11}.$$

We check that

$$\delta_3(3) = 6 \cdot 3^2 + 3 + 10 = 67 = 1 \pmod{11},$$

$$\delta_3(4) = 6 \cdot 4^2 + 4 + 10 = 110 = 0 \pmod{11},$$

$$\delta_3(5) = 6 \cdot 5^2 + 5 + 10 = 165 = 0 \pmod{11},$$

as it should be.

Shamir's Secret Sharing Scheme – Example 5

Computing $\delta_4(X)$

We then compute

$$\begin{aligned}\delta_4(\mathtt{X}) = \prod_{j=3,5} \frac{\mathtt{X}-j}{4-j} &= \frac{(\mathtt{X}-3)(\mathtt{X}-5)}{(4-3)(4-5)} \\ &= (\mathtt{X}^2 - 8\mathtt{X} + 15)(-1)^{-1} \\ &= (\mathtt{X}^2 + 3\mathtt{X} + 4)10 \\ &= 10\mathtt{X}^2 + 8\mathtt{X} + 7\ .\end{aligned}$$

We can check that $\delta_4(3) = 121 = 0 \pmod{11}$, $\delta_4(4) = 199 = 1 \pmod{11}$, and $\delta_4(5) = 297 = 0 \pmod{11}$.

Shamir's Secret Sharing Scheme – Example 6

Computing $\delta_5(X)$

We then compute

$$\begin{aligned}\delta_5(\mathtt{X}) = \prod_{j=3,4} \frac{\mathtt{X}-j}{5-j} &= \frac{(\mathtt{X}-3)(\mathtt{X}-4)}{(5-3)(5-4)} \\ &= (\mathtt{X}^2 - 7\mathtt{X} + 12)(2)^{-1} \\ &= (\mathtt{X}^2 + 4\mathtt{X} + 1)6 \\ &= 6\mathtt{X}^2 + 2\mathtt{X} + 6\ .\end{aligned}$$

We can check that $\delta_5(3) = 66 = 0 \pmod{11}$, $\delta_5(4) = 110 = 0 \pmod{11}$, and $\delta_5(5) = 166 = 1 \pmod{11}$.

Shamir's Secret Sharing Scheme – Example 7

Assembling $h(X)$ from the delta functions

It is now clear that if for any s_3, s_4, s_5 we let

$$h(\mathbf{X}) = s_3 \cdot \delta_3(\mathbf{X}) + s_4 \cdot \delta_4(\mathbf{X}) + s_5 \cdot \delta_5(\mathbf{X}) \ ,$$

then $h(3) = s_3 \cdot 1 + s_4 \cdot 0 + s_5 \cdot 0 = s_3$, $h(4) = s_3 \cdot 0 + s_4 \cdot 1 + s_5 \cdot 0 = s_4$ and $h(5) = s_3 \cdot 0 + s_4 \cdot 0 + s_5 \cdot 1 = s_5$, which implies that if $s_3 = f(3)$, $s_4 = f(4)$ and $s_5 = f(5)$ for some quadratic polynomial, then $h(\mathbf{X}) = f(\mathbf{X})$. This allows to compute $h(\mathbf{X})$ from the three shares.

Shamir's Secret Sharing Scheme – Example 8

Determining all parameters of h(X)

$$\begin{aligned} h(\mathtt{X}) &= s_3\delta_3(\mathtt{X}) + s_4\delta_4(\mathtt{X}) + s_5\delta_5(\mathtt{X}) \\ &= (6s_3 + 10s_4 + 6s_5)\mathtt{X}^2 + (s_3 + 8s_4 + 2s_5)\mathtt{X} + (10s_3 + 7s_4 + 6s_5)\ . \end{aligned}$$

Since we consider $h(\mathtt{X})$ of the form $h(\mathtt{X}) = s + a_1\mathtt{X} + a_2\mathtt{X}^2$, we have that

$$\begin{aligned} s &= 10s_3 + 7s_4 + 6s_5 \bmod 11 \\ a_1 &= s_3 + 8s_4 + 2s_5 \bmod 11 \\ a_2 &= 6s_3 + 10s_4 + 6s_5 \bmod 11\ , \end{aligned}$$

which is then the general formula for computing $h(\mathtt{X})$ from the three shares $s_3 = h(3), s_4 = h(4), s_5 = h(5)$.

Shamir's Secret Sharing Scheme – Example 9

Plugging in the numbers to determine all parameters of h(X)

In our concrete example we had the shares $s_3 = 6$, $s_4 = 6$ and $s_5 = 8$. If we plug this in we get

$$\begin{aligned} s &= 10 \cdot 6 + 7 \cdot 6 + 6 \cdot 8 \bmod 11 = 150 \bmod 11 = 7 \\ a_1 &= 6 + 8 \cdot 6 + 2 \cdot 8 \bmod 11 = 70 \bmod 11 = 4 \\ a_2 &= 6 \cdot 6 + 10 \cdot 6 + 6 \cdot 8 \bmod 11 = 144 \bmod 11 = 1 \ , \end{aligned}$$

which gives exactly the polynomial in (3.2).

$$h(\mathtt{X}) = s + a_1\mathtt{X} + a_2\mathtt{X}^2 = 7 + 4\mathtt{X} + \mathtt{X}^2 \ .$$

Shamir's Secret Sharing Scheme – Example 10

Determining the recombination vector for h(X)

If we had only been interested in finding the secret s and not the entire polynomial we would only need the equation $s = 10s_3 + 7s_4 + 6s_5 \bmod 11$. We see that $r = (10, 7, 6)$ is the recombination vector for finding $h(0)$ from $h(3), h(4), h(5)$ when $h(\mathbf{X})$ is a polynomial of degree at most 2.

- $r = (10,7,6)$ is the recombination vector for set $C = \{3,4,5\}$, and this set C is the only information required to calculate r.

A Passively Secure Protocol

Protocol Requirements

- Evaluate function :

$$f : \mathbb{F}^n \rightarrow \mathbb{F}^n, (x_1, \ldots, x_n) \rightarrow (y_1, \ldots, y_n)$$

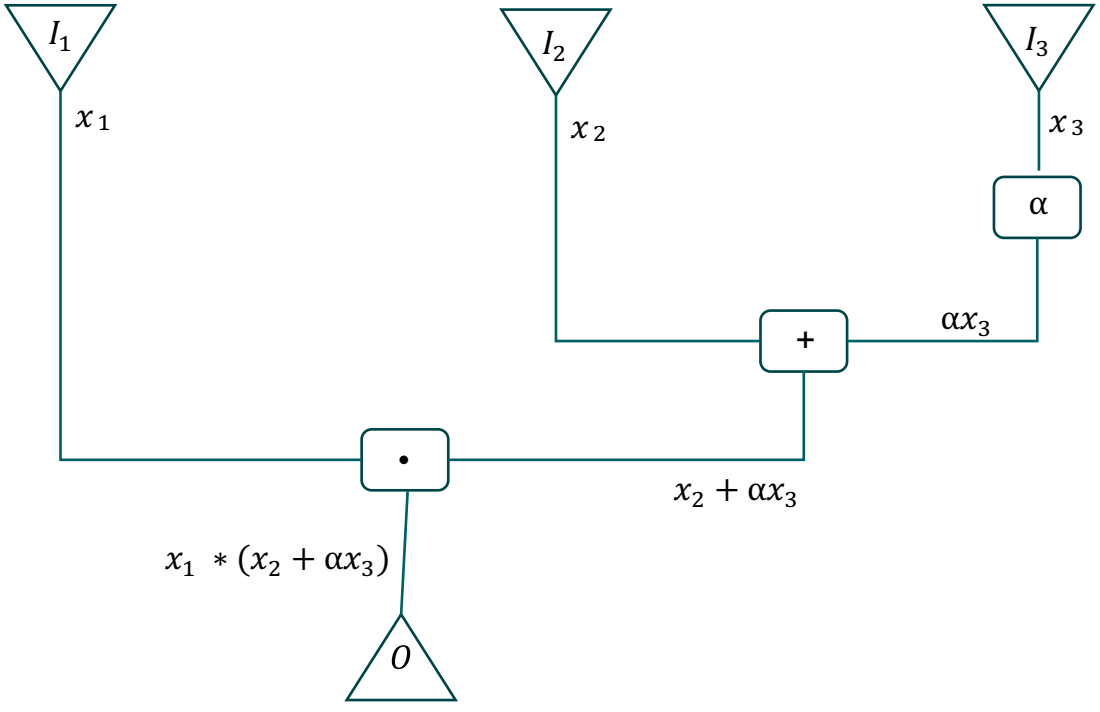
- Each party has one input and one output to $\mathbb{F}$
  - we assume this for clarity of explanation

- Every $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$ can be described as **arithmetic circuit.**

Arithmetic Circuits

- An Arithmetic Circuit is an acyclic directed graph
  - $\text{No}des \mathrel{\widehat{=}} \text{Gates}$
  - $\text{Edges} \mathrel{\widehat{=}} \text{Wires}$
  - Gates have at most 2 in-coming Wires
  - $P_1, \ldots, P_n$ are represented through n input gates with no in-coming and 1 out-coming wire
  - Multiply-by-constant gate
  - Addition gate
  - Multiplication gate
  - Exactly one output gate

Arithmetic Circuits



Prelimaries

We define $[a; f]_t = \big(\mathrm{f}(1), \dots, \mathrm{f}(\mathrm{n})\big)$

- $\mathrm{a} \in \mathbb{F}$
- f is a polynomial over f with $f(0) = a$ and degree at most t
- $f(1), \dots, f(n)$ denote the corresponding shares to $P_1, \dots, P_n$

Circuit Evaluation with Passive Security (CEPS)

Preliminaries

- Addition
  - $[a; f]_t + [b; g]_t = (f(1) + g(1), \dots, f(n) + g(n))$
- Multiplikation with scalar
  - $\alpha[a; f]_t = (\alpha f(1), \dots, \alpha f(n))$
- Multiplikation
  - $[a; f]_t * [b; g]_t = (f(1)g(1), \dots, f(n)g(n))$

Circuit Evaluation with Passive Security (CEPS)

Preliminaries

- Addition
  - $[a; f]_t + [b; g]_t = [a + b; f + g]_t$
- Multiplikation with scalar
  - $\alpha[a; f]_t = [\alpha a, \alpha f]_t$
- Multiplikation
  - $[a; f]_t * [b; g]_t = [ab; fg]_{2t}$

Circuit Evaluation with Passive Security (CEPS)

Preliminaries

- A player distributes $[a; f_a]_t$
  - Choose random polynom $f_a(X)$ of degree $\leq t$ with $f_a(0) = a$
  - Send shares $f_a(j)$ to P_j for $j = 1, \dots, n$

- The players hold $[a; f_a]_t$
  - The players have obtained shares of value a, based on polynomial f_a

- The players compute $[a; f_a]_t + [b; f_b]_t$
  - P_i computes $f_a(i) + f_b(i)$
  - Now $P_1, \dots, P_n$ hold $[a + b; f + g]_t$

Circuit Evaluation with Passive Security (CEPS)

The Protocol

1. Input Sharing
   - Each player P_i holding input $x_i \in \mathrm{F}$ distributes $[x_i; f_{x_i}]_t$

2. Invariant
   - Computing with the circuit on inputs x_1, ..., x_n assigns a unique value for each wire
   - Let $a \in \mathbb{F}$ be the value of a specific wire
     → After processing the gate every player holds $[a\,; f_a]_t$ for a polynomial f_a of degree t

3. Computation Phase (Next slides)

4. Output reconstruction
   - Every player holds a result $[y; f_y]_t$ for the output gate
   - y is the value assigned to the output gate
   - Each P_j sends $f_y(j)$ to P_i to reconstruct $f_y(0)$ through Lagrange interpolation

Circuit Evaluation with Passive Security (CEPS)

Computation Phase

- Addition gate
  - Input : $[a; f_a]_t$ and $[b; f_b]_t$
  - Output : $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$

- Multiply-by-constant gate
  - Input : $[a; f_a]_t$
  - Output : $\alpha[a, f_a] = [\alpha a; \alpha f_a]_t$

Circuit Evaluation with Passive Security (CEPS)

Computation Phase

- Multiplication gate
  - Input : $[a; f_a]_t$ and $[b; f_b]_t$
  - Output : $[a * b; f_a * f_b]_{2t}$

Problem ?

$f_{ab} = f_a * f_b$ has degree $2t$

  - Problem with increasing degree means
    - Imagine several multiplication gates → degree can reach any $x \in \mathbb{N}$
  - How to keep a degree of t ? Ideas ?
  - Solution : Define a new polynom h with degree t, $h(0) = ab$ and do interpolation over $[f_{ab}(1), \ldots, f_{ab}(n)]$

Circuit Evaluation with Passive Security (CEPS)

Computation Phase

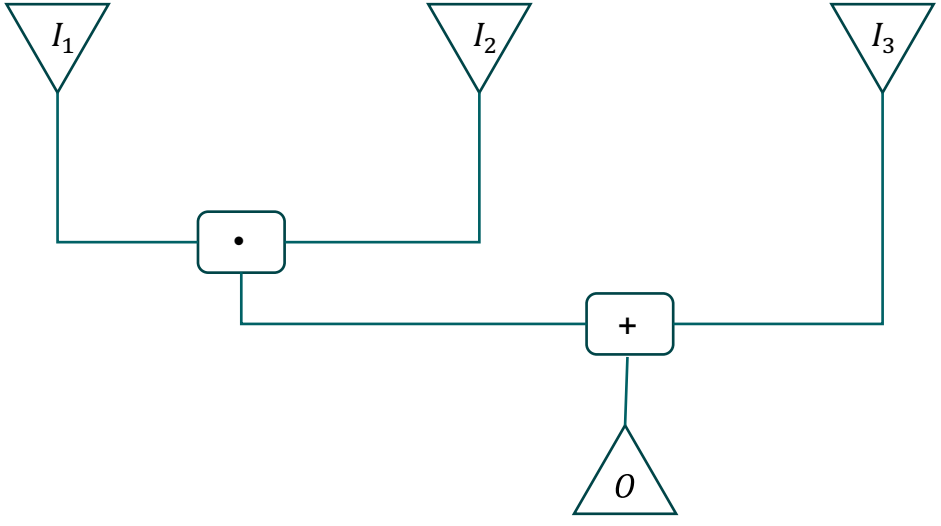
- Multiplication gate
  - Input : $[a; f_a]_t$ and $[b; f_b]_t$
    1. Compute $[a; f_a]_t * [b; f_b]_t = [ab; f_a f_b]_{2t}$
    2. Define $h \stackrel{\text{def}}{=} f_a f_b$. Then $h(0) = f_a(0) f_b(0) = ab$, so all parties hold $[ab; h]_{2t}$
       $\Rightarrow P_i$ holds $h(i)$ and each P_i distributes $[h(i); f_i]_t$
    3. Note that $\deg(h) = 2t \leq n - 1$
    4. Let r be the recombination vector ($h(0) = \sum_i r_i h(i)$ for any polynomial h of degree $\leq n - 1$) , then

$$\sum_i r_i [h(i); f_i]_t = [\sum_i r_i h(i); \sum_i r_i f_i] = [h(0); \sum_i r_i f_i]_t = [ab; \sum_i r_i f_i]_t$$

  - Output : $[ab; \sum_i r_i f_i]_t$

Example CEPS

- Three parties P_1, P_2, P_3
- Calculate $O = (I_1 * I_2) + I_3$ in $\mathbb{F}_7$



Example CEPS

1. Input : $I_1 = 2, I_2 = 2, I_3 = 3$ in $\mathbb{F}_7$

2. t = 1 (which means only one party can be corrupted)

3. P_1 distributes $[I_1; f_1]_t$ with $f_1 = 2 + x \quad \rightarrow [3,4,5]$
   P_2 distributes $[I_2; f_2]_t$ with $f_2 = 2 + 2x \quad \rightarrow [4,6,1]$
   P_3 distributes $[I_3; f_3]_t$ with $f_3 = 3 + 3x \quad \rightarrow [6,2,5]$

4. The players compute $I_1 * I_2$

	I_1	I_2	I_3
P_1	3	4	6
P_2	4	6	2
P_3	5	1	5

$\Longrightarrow$

	$I_1 * I_2$	I_3
P_1	5	6
P_2	3	2
P_3	5	5

Example CEPS

5. P_1 distributes $[I_1 I_2; f_1]_t$ with $f_1 = 5 + 2x \rightarrow [0,2,4]$
   P_2 distributes $[I_1 I_2; f_2]_t$ with $f_2 = 3 + x \rightarrow [4,5,6]$
   P_3 distributes $[I_1 I_2; f_3]_t$ with $f_3 = 5 - x \rightarrow [4,3,2]$

6. Lagrange Interpolation

$$\delta_1(0) = \frac{0-2}{2-1} * \frac{0-3}{3-1} = 3 \quad , \quad \delta_2(0) = \frac{0-1}{1-2} * \frac{0-3}{3-2} = 4 \, , \quad \delta_3(0) = \frac{0-1}{1-3} * \frac{0-2}{2-3} = 1$$

$$\Rightarrow r = (3,4,1)$$

$\Rightarrow P_1$ calculates $f_{I_1 I_2}(1) = 3 * 0 + 4 * 4 + 1 * 4 = 6$
P_2 calculates $f_{I_1 I_2}(2) = 3 * 2 + 4 * 5 + 1 * 3 = 1$
P_3 calculates $f_{I_1 I_2}(3) = 3 * 4 + 4 * 6 + 1 * 2 = 3$

$\Rightarrow$ The players hold $[I_1 I_2, f_{I_1 I_2}]_t$

Example CEPS

7. The players compute $I_1 * I_2 + I_3$

	$I_1 * I_2$	I_3
P_1	6	6
P_2	1	2
P_3	3	5

$\Longrightarrow$

	$I_1 * I_2 + I_3$
P_1	5
P_2	3
P_3	1

8. Output reconstruction

$$\delta_1(0) = \frac{0-2}{1-2} = \frac{-2}{-1} = 2 \quad , \quad \delta_2(0) = \frac{0-1}{2-1} = \frac{-1}{1} = 6$$

$$\Rightarrow r = (2,6)$$

$$\Rightarrow\ 2 * 5 + 6\ * 3 \mathrel{\widehat{=}} 28 \mathrel{\widehat{=}} 0 \mathrel{\widehat{=}} 2 * 2 + 3 = I_1 * I_2 + I_3$$

Summary: CEPS protocol for SMPC

Properties of the protocol:

- Allows any number of players $n \geq 3$
- Protocol works as long as $t < n/2$ (less then half of nodes are colluding)
- We still assume that all players follow protocol.
- This is called **semi-honest or passive security.**

- Uses Shamir's Secret Sharing Scheme.
  - Which is based on Lagrange Interpolation of Polynomials.

- Allows arithmetic circuits of any depth with addition and multiplication and skalar multiplication.