Linear Secret Sharing
Actively Secure Multiparty Computation

Peeter Laud, Pille Pullonen

October 29, 2021
Overview

- Access structure
- Adversary structure
- Secret sharing
- Error correcting codes
- Shamir secret sharing
- Linear secret sharing schemes
- MPC from LSSS
- Active security from LSSS
Access Structures

Defines which sets of parties are qualified (privileged) to restore the secret.

Set $p$ of subsets of parties $P$:
- Upwards closed: if $P_1 \in p$ and $P_1 \subset P_2$ then $P_2 \in p$.

Threshold structure - $t$ parties can always reconstruct the secret.

Adversary structure - sets of parties that the adversary can corrupt together.
- For security should never corrupt any set in the access structure.
- If no two sets cover the set of all parties, then we can do information theoretic MPC with passive security.
- If no three sets cover the set of all parties, then we can do information theoretic MPC with active security.
Access Structures

- Defines which sets of parties are qualified (privileged) to restore the secret
- Set $p$ of subsets of parties $\mathcal{P}$
- Upwards closed: if $P_1 \in p$ and $P_1 \subset P_2$ then $P_2 \in p$
Access Structures

- Defines which sets of parties are qualified (privileged) to restore the secret
- Set $p$ of subsets of parties $\mathcal{P}$
- Upwards closed: if $P_1 \in p$ and $P_1 \subset P_2$ then $P_2 \in p$
- Threshold structure - $t$ parties can always reconstruct the secret
Access Structures

- Defines which sets of parties are qualified (privileged) to restore the secret
- Set \( p \) of subsets of parties \( \mathcal{P} \)
- Upwards closed: if \( P_1 \in p \) and \( P_1 \subset P_2 \) then \( P_2 \in p \)
- Threshold structure - \( t \) parties can always reconstruct the secret
- Adversary structure - sets of parties that the adversary can corrupt together
  - For security should never corrupt any set in the access structure
Access Structures

- Defines which sets of parties are qualified (privileged) to restore the secret
- Set $p$ of subsets of parties $\mathcal{P}$
- Upwards closed: if $P_1 \in p$ and $P_1 \subset P_2$ then $P_2 \in p$
- Threshold structure - $t$ parties can always reconstruct the secret
- Adversary structure - sets of parties that the adversary can corrupt together
  - For security should never corrupt any set in the access structure
  - If no two sets cover the set of all parties then we can do information theoretic MPC with passive security
Access Structures

- Defines which sets of parties are qualified (privileged) to restore the secret
- Set $p$ of subsets of parties $\mathcal{P}$
- Upwards closed: if $P_1 \in p$ and $P_1 \subset P_2$ then $P_2 \in p$
- Threshold structure - $t$ parties can always reconstruct the secret
- Adversary structure - sets of parties that the adversary can corrupt together
  - For security should never corrupt any set in the access structure
  - If no two sets cover the set of all parties then we can do information theoretic MPC with passive security
  - If no three sets cover the set of all parties then we can do information theoretic MPC with active security
Q2 and Q3 access structures

- Let us for now define the access structure as the set of unprivileged sets of parties
  - \( \mathcal{A} \in \mathcal{P}(\{P_1, \ldots, P_n\}) \), and \( \mathcal{A} \) is downwards closed
- An access structure \( \mathcal{A} \) is Q2, if for any \( P_1, P_2 \subseteq \{P_1, \ldots, P_n\} \): if \( P_1, P_2 \in \mathcal{A} \), then \( P_1 \cup P_2 \neq \{P_1, \ldots, P_n\} \)
- An access structure \( \mathcal{A} \) is Q3, if for any \( P_1, P_2, P_3 \subseteq \{P_1, \ldots, P_n\} \): if \( P_1, P_2, P_3 \in \mathcal{A} \), then \( P_1 \cup P_2 \cup P_3 \neq \{P_1, \ldots, P_n\} \)

Theorem

*If an access structure is Q2 [Q3], then there exists an “efficient”, information-theoretically, passively [actively] secure MPC protocol for this access structure.*
Construction (for Q2)

- Let $\Pi$ be a 3-party MPC prot. passively secure against 1 party
  - Think of a secret sharing based protocol, e.g. Sharemind
- Let $A = A_0 \cup A_1 \cup A_2$, all three parts of size $\approx |A|/3$
- Let $\Pi_i$ be a $n$-party protocol secure against coalitions in $A_{(i+1) \mod 3} \cup A_{(i+2) \mod 3}$
  - Induction hypothesis: such $\Pi_0, \Pi_1, \Pi_2$ exist
  - Exercise. What is the basis of induction?
- Run $\Pi$, but let the $i$-th party ($i \in \{0, 1, 2\}$) in $\Pi$ be collectively run by all parties $P_1, \ldots, P_n$, using the protocol $\Pi_i$
  - Exercise. Security?
  - Exercise. Where do we use the Q2 property?

Construction for Q3: basically the same. $3 \mapsto 4$
Over half of the parties must be honest (for inf.-theor. security)

- Consider a two-party protocol $\Pi$ for computing the AND of two bits.
- Let $\Pi(b_1, r_1, b_2, r_2)$ be the sequence of messages (and the output bit) exchanged for party $P_i$'s bit $b_i$ and random coins $r_i$.

\[
\forall r_1, r_2^0 \exists r_2^1 : \Pi(0, r_1, 0, r_2^0) = \Pi(0, r_1, 1, r_2^1) \\
\forall r_1, r_2^1 \exists r_2^0 : \Pi(0, r_1, 0, r_2^0) = \Pi(0, r_1, 1, r_2^1) \\
\forall r_1, r_2^0, r_2^1 : \Pi(1, r_1, 0, r_2^0) \neq \Pi(1, r_1, 1, r_2^1)
\]

- Party $P_2$ whose input is $b_2 = 0$ and random coins $r_2^0$ can find $b_1$ as follows:
  - Let $T$ be the exchanged sequence of messages.
  - Try to find such $(b', r', r_2^1)$, that $\Pi(b', r', 1, r_2^1) = T$.
  - If such triple exists then $b_1 = 0$. If not, then $b_1 = 1$.

**Exercise.** Generalize this result to more than 2 parties.
Secret Sharing Scheme

- A vector \( s_0, s_1, \ldots, s_n \) where \( n \geq 1 \)
- \( s_0 \) is the secret, the rest are the shares
- The secret \( s_0 \) is (uniformly) random (in some set)

Access structure (should not be empty)
- The set of all parties is always able to reconstruct the value

Privacy structure - sets of parties that cannot learn anything about the secret
- Empty set is always a privacy structure
- Reasonably, this should contain some non-empty sets

There may be a gap between access and privacy structures
- Some sets of parties that can not fully reconstruct but may learn some information
- Perfect schemes have no such gap

Formally, there can be an overlap if the set of secrets is known independently of the sharing scheme
- For reasonable schemes the secret \( s_0 \) is chosen from some set with at least two elements
Secret Sharing Scheme

- A vector $s_0, s_1, \ldots, s_n$ where $n \geq 1$
  - $s_0$ is the secret, the rest are the shares
  - The secret $s_0$ is (uniformly) random (in some set)
Secret Sharing Scheme

- A vector \( s_0, s_1, \ldots, s_n \) where \( n \geq 1 \)
  - \( s_0 \) is the secret, the rest are the shares
  - The secret \( s_0 \) is (uniformly) random (in some set)
- Access structure (should not be empty)
  - The set of all parties is always able to reconstruct the value

Privacy structure - sets of parties that cannot learn anything about the secret
- Empty set is always a privacy structure
- Reasons: this should contain some non-empty sets
- There may be a gap between access and privacy structures
  - Some sets of parties that can not fully reconstruct but may learn some information
- Perfect schemes have no such gap
- Formally, there can be an overlap if the set of secrets is known independently of the sharing scheme
- For reasonable schemes the secret \( s_0 \) is chosen from some set with at least two elements
Secret Sharing Scheme

- A vector $s_0, s_1, \ldots, s_n$ where $n \geq 1$
  - $s_0$ is the secret, the rest are the shares
  - The secret $s_0$ is (uniformly) random (in some set)
- Access structure (should not be empty)
  - The set of all parties is always able to reconstruct the value
- Privacy structure - sets of parties that cannot learn anything about the secret
  - Empty set is always a privacy structure
  - Reasonably, this should contain some non-empty sets
Secret Sharing Scheme

- A vector $s_0, s_1, \ldots, s_n$ where $n \geq 1$
  - $s_0$ is the secret, the rest are the shares
  - The secret $s_0$ is (uniformly) random (in some set)
- Access structure (should not be empty)
  - The set of all parties is always able to reconstruct the value
- Privacy structure - sets of parties that cannot learn anything about the secret
  - Empty set is always a privacy structure
  - Reasonably, this should contain some non-empty sets
- There may be a gap between access and privacy structures
  - Some sets of parties that can not fully reconstruct but may learn some information
  - Perfect schemes have so such gap
Secret Sharing Scheme

- A vector $s_0, s_1, \ldots, s_n$ where $n \geq 1$
  - $s_0$ is the secret, the rest are the shares
  - The secret $s_0$ is (uniformly) random (in some set)
- Access structure (should not be empty)
  - The set of all parties is always able to reconstruct the value
- Privacy structure - sets of parties that cannot learn anything about the secret
  - Empty set is always a privacy structure
  - Reasonably, this should contain some non-empty sets
- There may be a gap between access and privacy structures
  - Some sets of parties that can not fully reconstruct but may learn some information
  - Perfect schemes have no such gap
- Formally, there can be an overlap if the set of secrets is known independently of the sharing scheme
  - For reasonable schemes the secret $s_0$ is chosen from some set with at least two elements
Shamir Secret Sharing

- Input: element $v \in \mathbb{F}$, threshold $t$, number of parties $n \geq t$
Shamir Secret Sharing

- Input: element $v \in \mathbb{F}$, threshold $t$, number of parties $n \geq t$
- Share:
  - Pick random $a_1, \ldots, a_{t-1} \in \mathbb{F}$
  - Define a polynomial $q(x) = v + \sum_{i=1}^{t-1} a_i \cdot x^i$
  - Compute shares $s_i = q(i) \in \mathbb{F}$ for all parties $\mathcal{P}_i$
Shamir Secret Sharing

Input: element \( v \in \mathbb{F} \), threshold \( t \), number of parties \( n \geq t \)

Share:
- Pick random \( a_1, \ldots, a_{t-1} \in \mathbb{F} \)
- Define a polynomial \( q(x) = v + \sum_{i=1}^{t-1} a_i \cdot x^i \)
- Compute shares \( s_i = q(i) \in \mathbb{F} \) for all parties \( P_i \)

Reconstruct:
- Collect \( s_i = q(i) \) for at least \( t \) values
- Interpolate to reconstruct \( q(x) = v + \sum_{i=1}^{t-1} a_i \cdot x^i \)
  - e.g. Lagrange interpolation
- Recover \( v \) from \( q(x) \)
Shamir Secret Sharing

- Input: element $v \in \mathbb{F}$, threshold $t$, number of parties $n \geq t$
- Share:
  - Pick random $a_1, \ldots, a_{t-1} \in \mathbb{F}$
  - Define a polynomial $q(x) = v + \sum_{i=1}^{t-1} a_i \cdot x^i$
  - Compute shares $s_i = q(i) \in \mathbb{F}$ for all parties $P_i$
- Reconstruct:
  - Collect $s_i = q(i)$ for at least $t$ values
  - Interpolate to reconstruct $q(x) = v + \sum_{i=1}^{t-1} a_i \cdot x^i$
    - e.g. Lagrange interpolation
  - Recover $v$ from $q(x)$
- Both sharing and reconstructing require only linear operations with $v$ and shares $s_i$
Error Correcting Codes

- An error-correcting code $\mathcal{C}$ maps items of length $t$ to codewords of length $n$ for all $x_1, x_2 \in \mathcal{X}_t$ where $x_1 \neq x_2$ we have $\mathcal{C}(x_1) \neq \mathcal{C}(x_2)$ and $\mathcal{C}(x_1)$ and $\mathcal{C}(x_2)$ differ in at least $d$ positions.

- Can detect up to $d - 1$ errors.

- Can correct up to $(d - 1)/2$ errors.

- A code is linear if $\mathcal{C}$ is a linear map.

- For linear codes $d \leq n - t + 1$. 

October 29, 2021

9
Error Correcting Codes

- \((t, n, d)\) error-correcting code
  - Map items of length \(t\) to codewords of length \(n\)
    - \(C : X^t \rightarrow X^n\)
  - for all \(x_1, x_2 \in X^t\) where \(x_1 \neq x_2\) we have \(C(x_1) \neq C(x_2)\)
    - \(C(x_1)\) and \(C(x_2)\) differ in at least \(d\) positions

Can detect up to \(d - 1\) errors
Can correct up to \((d - 1)/2\)

A code is linear if \(C\) is a linear map
For linear codes \(d \leq n - t + 1\)
Error Correcting Codes

- $(t, n, d)$ error-correcting code
  - Map items of length $t$ to codewords of length $n$
    - $C : X^t \to X^n$
  - For all $x_1, x_2 \in X^t$ where $x_1 \neq x_2$ we have $C(x_1) \neq C(x_2)$
    - $C(x_1)$ and $C(x_2)$ differ in at least $d$ positions

- Can detect up to $d - 1$ errors
- Can correct up to $(d - 1)/2$
- A code is linear if $C$ is a linear map
  - For linear codes $d \leq n - t + 1$
Reed-Solomon Codes

- $(t, n, n - t + 1)$ linear code
- Works over a finite field $\mathbb{F}_q$ where $q > n$
Reed-Solomon Codes

- $(t, n, n - t + 1)$ linear code
- Works over a finite field $\mathbb{F}_q$ where $q > n$
- Encoding a word $(v_1, \ldots, v_t) \in \mathbb{F}^t_q$
  - Define a polynomial $p(x) = \sum_{i=1}^{t} v_i x^{i-1}$
  - Pick $n$ different points $a_1, \ldots, a_n$
  - Codeword is $(p(a_1), \ldots, p(a_n))$
Reed-Solomon Codes

- $(t, n, n - t + 1)$ linear code
- Works over a finite field $\mathbb{F}_q$ where $q > n$
- Encoding a word $(v_1, \ldots, v_t) \in \mathbb{F}_q^t$
  - Define a polynomial $p(x) = \sum_{i=1}^{t} v_i x^{i-1}$
  - Pick $n$ different points $a_1, \ldots, a_n$
  - Codeword is $(p(a_1), \ldots, p(a_n))$
- Decoding $(p(a_1), \ldots, p(a_n))$
  - Interpolate the polynomial
Reed-Solomon Error Correction

- Decoding a codeword with less than \((n - t + 1)/2\) errors?
  - The result is not a polynomial with degree \(t - 1\) if we try to decode using all points.
  - Can try subsets of points until you find some that define a polynomial with degree \(t - 1\).
- In reality, see for example Berlekamp–Welch algorithm.
Decoding Reed-Solomon codes

- Suppose that the original codeword was \((s_1, \ldots, s_n)\), corresponding to the polynomial \(p\).
- But we received \((\tilde{s}_1, \ldots, \tilde{s}_n)\).
  - We assume it has at most \((n - t)/2\) errors.
- Find the coefficients for polynomials \(q_0\) and \(q_1\), such that
  - Degree of \(q_0\) is at most \((n + t - 2)/2\). Degree of \(q_1\) is at most \((n - t)/2\).
  - For all \(i \in \{1, \ldots, n\}\): \(q_0(a_i) - q_1(a_i) \cdot \tilde{s}_i = 0\).
  - \(q_0\) and \(q_1\) are not both equal to 0.
- Then \(p = q_0 / q_1\).
- In general, there are more equations than variables, but \(\tilde{s}_i\) are not arbitrary.
Correctness of decoding

Such polynomials $q_0, q_1$ exist:

1. $(s_1, \ldots, s_n), (\tilde{s}_1, \ldots, \tilde{s}_n)$ — original and received codewords. Let $E$ be the set of $i$, where $s_i \neq \tilde{s}_i$. Then $|E| \leq (n - t)/2$.

2. Let $k(x) = \prod_{i \in E} (x - a_i)$. Then $\deg k \leq (n - t)/2$.

3. Take $q_1 = k$ and $q_0 = p \cdot k$. Then $\deg q_0 \leq (n + t - 2)/2$.

4. For all $i \in \{1, \ldots, n\}$ we have

$$q_0(a_i) - q_1(a_i) \cdot \tilde{s}_i = k(a_i)(p(a_i) - \tilde{s}_i) = k(a_i)(s_i - \tilde{s}_i) =$$

$$\begin{cases} k(a_i)(s_i - s_i) = 0, & i \notin E \\ 0 \cdot (s_i - \tilde{s}_i) = 0, & i \in E \end{cases}$$
Correctness of decoding

If $q_0$ and $q_1$ satisfy the equalities and upper bounds on degrees, then $p = q_0 / q_1$:

- Let $q'(x) = q_0(x) - q_1(x)p(x)$. Degree of $q'$ is at most $(n + t - 2)/2$.
- For each $i \notin E$,
  $q'(a_i) = q_0(a_i) - q_1(a_i)p(a_i) = q_0(a_i) - q_1(a_i)\tilde{s}_i = 0$.
  $1 \leq i \leq n$.

- The number of such $i$ is at least $n - (n - t)/2 = (n + t)/2$.

- Thus the number of roots of $q'$ is larger than its degree. Hence $q' = 0$.

- $q_0 - q_1 \cdot p = 0$. 
Shamir vs Reed-Solomon

Shamir Secret Sharing

- Secret share $v$
- Pick random $a_1, \ldots, a_{t-1}$
- Define

$$q(x) = v + \sum_{i=1}^{t-1} a_i x^i$$

- Shares are $(q(1), \ldots, q(n))$
Shamir vs Reed-Solomon

Shamir Secret Sharing

- Secret share $\nu$
- Pick random $a_1, \ldots, a_{t-1}$
- Define

$$q(x) = \nu + \sum_{i=1}^{t-1} a_i x^i$$

- Shares are $(q(1), \ldots, q(n))$

Reed-Solomon Error Correction

- Encode $(a_0, \ldots a_{t-1})$
- Pick $n$ points, e.g. $1, \ldots, n$
- Define

$$p(x) = \sum_{i=0}^{t-1} a_i x^i$$

- Codeword $(p(1), \ldots, p(n))$
**Shamir vs Reed-Solomon**

**Shamir Secret Sharing**
- Secret share \( v \)
- Pick random \( a_1, \ldots, a_{t-1} \)
- Define
  
  \[
  q(x) = v + \sum_{i=1}^{t-1} a_i x^i
  \]

  Shares are \((q(1), \ldots, q(n))\)

**Reed-Solomon Error Correction**
- Encode \((a_0, \ldots a_{t-1})\)
- Pick \( n \) points, e.g. \( 1, \ldots, n \)
- Define
  
  \[
  p(x) = \sum_{i=0}^{t-1} a_i x^i
  \]

  Codeword \((p(1), \ldots, p(n))\)

With Shamir secret sharing we can sometimes detect if some party is submitting a wrong share and do error correction to recover the correct value. For some parameters we get active security.
Error Correction vs Secret Sharing

- Any code defines a secret sharing scheme
  - Shares are the components of the codeword
  - Secret is somehow encoded in the input encoding or as a piece of the codeword that is not given to any party
Error Correction vs Secret Sharing

- Any code defines a secret sharing scheme
  - Shares are the components of the codeword
  - Secret is somehow encoded in the input encoding or as a piece of the codeword that is not given to any party
  - Access structure is given by error correction capabilities
    - If we can correct errors then we can also ignore missing values
Error Correction vs Secret Sharing

- Any code defines a secret sharing scheme
  - Shares are the components of the codeword
  - Secret is somehow encoded in the input encoding or as a piece of the codeword that is not given to any party
  - Access structure is given by error correction capabilities
    - If we can correct errors then we can also ignore missing values
  - Non-trivial to decide on the privacy structure
Linear Secret Sharing Scheme (LSSS)

- Secret sharing scheme with linear reconstruction and sharing operations
Linear Secret Sharing Scheme (LSSS)

- Secret sharing scheme with linear reconstruction and sharing operations
- Can be defined as matrix operations
  - Matrix $M \in \mathbb{F}^{m \times d}$
  - Vector $w \in \mathbb{F}^d$
  - Map $\chi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$ to determine which party gets which share
LSSS Sharing

- **Setup:**
  - Matrix $M \in \mathbb{F}^{m \times d}$
  - Vector $w \in \mathbb{F}^d$
  - Map $\chi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$

- **Sharing $v$:**
  - Pick a vector $a \in \mathbb{F}^d$ such that $v = w^T \cdot a$
  - Compute the shares $s = (s_1, \ldots, s_m)^T = M \cdot a$
  - Give party $P_i$ share $s_j$ if $\chi(j) = i$
LSSS Sharing

- Setup:
  - Matrix $M \in \mathbb{F}^{m \times d}$
  - Vector $w \in \mathbb{F}^d$
  - Map $\chi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$

- Sharing $v$:
  - Pick a vector $a \in \mathbb{F}^d$ such that $v = w^T \cdot a$
LSSS Sharing

- **Setup:**
  - Matrix $M \in \mathbb{F}^{m \times d}$
  - Vector $w \in \mathbb{F}^d$
  - Map $\chi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$

- **Sharing $v$:**
  - Pick a vector $a \in \mathbb{F}^d$ such that $v = w^T \cdot a$
  - Compute the shares

\[
s = (s_1, \ldots, s_m)^T = M \cdot a
\]
LSSS Sharing

- **Setup:**
  - Matrix $M \in \mathbb{F}^{m \times d}$
  - Vector $w \in \mathbb{F}^d$
  - Map $\chi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$

- **Sharing $\nu$:**
  - Pick a vector $a \in \mathbb{F}^d$ such that $\nu = w^T \cdot a$
  - Compute the shares
    \[
    s = (s_1, \ldots, s_m)^T = M \cdot a
    \]
  - Give party $\mathcal{P}_i$ share $s_j$ if $\chi(j) = i$
LSSS Reconstruction

- Party $P_i$ has share $s_j$ if $\chi(j) = i$
- Public setup $(M, w, \chi)$ is known

\[ \text{Set of the linear combinations of the rows} \]
LSSS Reconstruction

- Party $P_i$ has share $s_j$ if $\chi(j) = i$
- Public setup $(M, w, \chi)$ is known
- A set of parties $A$ is in the access structure if the span of rows\(^1\) $j$ in $M$ contain $w$ for $P_i \in A$ and $\chi(j) = i$
  - There is $x_A$ such that $w^T = x_A^T \cdot M_A$
  - $M_A$ is the rows $j$ of $M$ for $P_i \in A$ and $\chi(j) = i$

---

\(^1\)Set of the linear combinations of the rows
LSSS Reconstruction

- Party $P_i$ has share $s_j$ if $\chi(j) = i$
- Public setup $(M, w, \chi)$ is known
- A set of parties $A$ is in the access structure if the span of rows\(^1\) $j$ in $M$ contain $w$ for $P_i \in A$ and $\chi(j) = i$
  - There is $x_A$ such that $w^T = x_A^T \cdot M_A$
  - $M_A$ is the rows $j$ of $M$ for $P_i \in A$ and $\chi(j) = i$
- Giving $s_A = M_A \cdot a$ from the sharing definition
- The parties in $A$ can compute:
  - $x_A^T \cdot s_A = v$
  - Which is correct as

\[
x_A^T \cdot s_A = x_A^T \cdot (M_A \cdot a) = (x_A^T \cdot M_A) \cdot a = w^T \cdot a = v
\]

\(^1\)Set of the linear combinations of the rows
Shamir Sharing as LSSS

- \((t, n)\) Shamir sharing:

\[
M = \begin{pmatrix}
1 & 1 & \ldots & 1^{t-1} \\
1 & 2^1 & \ldots & 2^{t-1} \\
\vdots & \vdots & \ddots & \vdots \\
1 & n & \ldots & n^{t-1}
\end{pmatrix} \in \mathbb{F}^{n \times t}
\]

\[
w = \begin{pmatrix}
1 \\
0 \\
\vdots \\
0
\end{pmatrix} \in \mathbb{F}^{t}
\]

\[
\chi(i) = i
\]
Additive Secret Sharing as LSSS

- $v = s_1 + \ldots + s_n$
- Threshold $t = n$

$$M = \begin{pmatrix}
1 & 0 & \ldots & 0 \\
0 & 1 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 1
\end{pmatrix} \in \mathbb{F}^{n \times n}$$

$$w = \begin{pmatrix}
1 \\
1 \\
\vdots \\
1
\end{pmatrix} \in \mathbb{F}^n$$

$$\chi(i) = i$$
Replicated Secret Sharing as LSSS

- Consider the case \((t, n) = (2, 3)\)

\[
M = \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

\[
w = \begin{pmatrix}
1 \\
1 \\
\end{pmatrix}
\]

\[
\chi(i) = \lceil (i/2) \rceil
\]

- Replicated means that each share \(s_i\) is held by multiple parties
- In this case \(a = a_1 + a_2 + a_3\)
- Party \(\mathcal{P}_i\) has \((a_{i-1}, a_{i+1})\)
Additively Homomorphic LSSS

- LSSS allows parties to compute arbitrary linear functions on the shares without interaction

\[
\begin{align*}
v & = M \cdot a \\
v' & = M \cdot a'
\end{align*}
\]

\[
\begin{align*}
v + v' & = s + s' = M \cdot (a + a') \\
w \cdot (a + a') & = \alpha \cdot v \\
\end{align*}
\]

Any linear combination is a combination of additions and multiplications with constants
Additively Homomorphic LSSS

- LSSS allows parties to compute arbitrary linear functions on the shares without interaction
  - Let \( v \) be shared as \( s = M \cdot a \) and \( v' \) as \( s' = M \cdot a' \)
  - \( v + v' \) is shared by \( s + s' \)
Additively Homomorphic LSSS

- LSSS allows parties to compute arbitrary linear functions on the shares without interaction
  - Let $v$ be shared as $s = M \cdot a$ and $v'$ as $s' = M \cdot a'$
  - $v + v'$ is shared by $s + s'$
    - $s + s' = M \cdot (a + a')$
    - $v + v' = w \cdot (a + a')$
    - Hence, this is like sharing $v + v'$ by choosing $a + a'$ in the algorithm
Additively Homomorphic LSSS

- LSSS allows parties to compute arbitrary linear functions on the shares without interaction
  - Let $v$ be shared as $s = M \cdot a$ and $v'$ as $s' = M \cdot a'$
  - $v + v'$ is shared by $s + s'$
    - $s + s' = M \cdot (a + a')$
    - $v + v' = w \cdot (a + a')$
    - Hence, this is like sharing $v + v'$ by choosing $a + a'$ in the algorithm
  - $\alpha \cdot v$ is shared by $\alpha s$
    - $\alpha s = M \cdot (\alpha a)$ and $\alpha v = w \cdot (\alpha a)$
Additively Homomorphic LSSS

- LSSS allows parties to compute arbitrary linear functions on the shares without interaction
  - Let $v$ be shared as $s = M \cdot a$ and $v'$ as $s' = M \cdot a'$
  - $v + v'$ is shared by $s + s'$
    - $s + s' = M \cdot (a + a')$
    - $v + v' = w \cdot (a + a')$
    - Hence, this is like sharing $v + v'$ by choosing $a + a'$ in the algorithm
  - $\alpha \cdot v$ is shared by $\alpha s$
    - $\alpha s = M \cdot (\alpha a)$ and $\alpha v = w \cdot (\alpha a)$
- Any linear combination is a combination of additions and multiplications with constants
Multiplicative LSSS

- LSSS is multiplicative if the parties can multiply two shared values “without interaction”
Multiplicative LSSS

- LSSS is multiplicative if the parties can multiply two shared values “without interaction”
- Let \( \nu \) be shared as \( s = M \cdot a \) and \( \nu' \) as \( s' = M \cdot a' \)
Multiplicative LSSS

- LSSS is multiplicative if the parties can multiply two shared values “without interaction”
- Let $v$ be shared as $s = M \cdot a$ and $v'$ as $s' = M \cdot a'$
- Define $s \otimes s'$ as the vector of all values $s_i \cdot s'_j$ where $\chi(i) = \chi(j)$
  - There is one party holding both $s_i$ and $s'_j$
Multiplicative LSSS

- LSSS is multiplicative if the parties can multiply two shared values “without interaction”
- Let $v$ be shared as $s = M \cdot a$ and $v'$ as $s' = M \cdot a'$
- Define $s \otimes s'$ as the vector of all values $s_i \cdot s'_j$ where $\chi(i) = \chi(j)$
- There is one party holding both $s_i$ and $s'_j$
- LSSS is multiplicative if there exists a vector $v$ such that for all values and respective sharings we have
  $$v \cdot v' = v^T \cdot (s \otimes s')$$
- If each party computes the part that they can compute on their values then we get some sharing of the multiplication result
Multiplicative LSSS

- LSSS is multiplicative if the parties can multiply two shared values “without interaction”
- Let $v$ be shared as $s = M \cdot a$ and $v'$ as $s' = M \cdot a'$
- Define $s \otimes s'$ as the vector of all values $s_i \cdot s'_j$ where $\chi(i) = \chi(j)$
  - There is one party holding both $s_i$ and $s'_j$
- LSSS is multiplicative if there exists a vector $v$ such that for all values and respective sharings we have
  \[ v \cdot v' = v^T \cdot (s \otimes s') \]

- If each party computes the part that they can compute on their values then we get some sharing of the multiplication result
- Multiplicative LSSS is a good basis for MPC
- Not all LSSS schemes are multiplicative but all LSSS schemes can be transformed to multiplicative schemes
Multiplicative LSSS Examples
Multiplicative LSSS Examples

- Shamir’s scheme with suitably low threshold
  - each party multiplies their shares together
Multiplicative LSSS Examples

- Shamir’s scheme with suitably low threshold
  - each party multiplies their shares together
- Replicated (2, 3) scheme
  - $v = a_1 + a_2 + a_3$
  - $P_1$ has $(a_2, a_3)$, $P_2$ has $(a_1, a_3)$ and $P_3$ has $(a_1, a_2)$

\[
v \cdot v' = (a_1 + a_2 + a_3) \cdot (a_1' + a_2' + a_3') \]
\[
= a_1 a_1' + a_1 a_2' + a_1 a_3'
  + a_2 a_1' + a_2 a_2' + a_2 a_3'
  + a_3 a_1' + a_3 a_2' + a_3 a_3'
\]
\[
= (a_2 a_2' + a_2 a_3' + a_3 a_2') - \text{Computed by } P_1
  + (a_3 a_3' + a_1 a_3' + a_3 a_1') - \text{Computed by } P_2
  + (a_1 a_1' + a_1 a_2' + a_2 a_1') - \text{Computed by } P_3
\]
MPC Multiplication from Multiplicative LSSS

- To compute multiplication:
  - Compute locally using multiplicative properties of the LSSS
MPC Multiplication from Multiplicative LSSS

- To compute multiplication:
  - Compute locally using multiplicative properties of the LSSS
  - Share the results that each party gets from the multiplication
    - The only step that requires communication
MPC Multiplication from Multiplicative LSSS

- To compute multiplication:
  - Compute locally using multiplicative properties of the LSSS
  - Share the results that each party gets from the multiplication
    - The only step that requires communication
  - Combine the shared results to the proper sharing of the multiplication result
- In MPC we need good properties of the shares that the multiplicative LSSS property may not ensure
- Also known as Maurer’s multiplication protocol
Shamir Scheme Multiplication

- Each party multiplies their shares
  - \( c_i = a_i \cdot b_i = f_a(i) \cdot f_b(i) = (f_a \cdot f_b)(i) \)
Shamir Scheme Multiplication

- Each party multiplies their shares
  \[ c_i = a_i \cdot b_i = f_a(i) \cdot f_b(i) = (f_a \cdot f_b)(i) \]
- Result is a correct result for a higher degree polynomial
  - Resharing was needed to reduce the degree of the polynomial
  - Only works for \( t < n/2 + 1 \)
Shamir Scheme Multiplication

- Each party multiplies their shares
  - \( c_i = a_i \cdot b_i = f_a(i) \cdot f_b(i) = (f_a \cdot f_b)(i) \)
- Result is a correct result for a higher degree polynomial
  - Resharing was needed to reduce the degree of the polynomial
  - Only works for \( t < n/2 + 1 \)
- See Gennaro-Rabin-Rabin multiplication protocol from the secret sharing based MPC lecture slides
- Maurer’s protocol was actually a generalization of this protocol
Replicated Scheme Multiplication

- LSSS multiplicative property gave us a sum $v \cdot v' = t_1 + t_2 + t_3$ where party $P_i$ has $t_i$
Replicated Scheme Multiplication

- LSSS multiplicative property gave us a sum $v \cdot v' = t_1 + t_2 + t_3$ where party $P_i$ has $t_i$
- This is correct multiplication result but the result is shared additively and without replication
Replicated Scheme Multiplication

- LSSS multiplicative property gave us a sum $v \cdot v' = t_1 + t_2 + t_3$ where party $P_i$ has $t_i$
- This is correct multiplication result but the result is shared additively and without replication
- Need sharing to get replicated shares again:
  - $P_i$ shares $t_i = t_{i,1} + t_{i,2} + t_{i,3}$ using replicated sharing
Replicated Scheme Multiplication

- LSSS multiplicative property gave us a sum $v \cdot v' = t_1 + t_2 + t_3$ where party $\mathcal{P}_i$ has $t_i$
- This is correct multiplication result but the result is shared additively and without replication
- Need sharing to get replicated shares again:
  - $\mathcal{P}_i$ shares $t_i = t_{i,1} + t_{i,2} + t_{i,3}$ using replicated sharing
  - Party $\mathcal{P}_j$ gets $(t_{i,j-1}, t_{i,j+1})$ as shares from each $\mathcal{P}_i$
Replicated Scheme Multiplication

- LSSS multiplicative property gave us a sum $v \cdot v' = t_1 + t_2 + t_3$ where party $P_i$ has $t_i$
- This is correct multiplication result but the result is shared additively and without replication
- Need sharing to get replicated shares again:
  - $P_i$ shares $t_i = t_{i,1} + t_{i,2} + t_{i,3}$ using replicated sharing
  - Party $P_j$ gets $(t_{i,j-1}, t_{i,j+1})$ as shares from each $P_i$
  - Parties compute $t_1 + t_2 + t_3$ on the newly shared values
    - Each party sums their respective shares
  - The output of the linear combination is a valid replicated sharing for $v \cdot v' = t_1 + t_2 + t_3$
Replicated Scheme Multiplication

Example

- \( P_1 \) has \((a_2, a_3)\), \( P_2 \) has \((a_1, a_3)\) and \( P_3 \) has \((a_1, a_2)\)

\[ v \cdot v' = (a_1 + a_2 + a_3) \cdot (a'_1 + a'_2 + a'_3) \]

\[ = (a_2 a'_2 + a_2 a'_3 + a_3 a'_2) \] – Computed by \( P_1 \), denote as \( t_1 \)

\[ + (a_3 a'_3 + a_1 a'_3 + a_3 a'_1) \] – Computed by \( P_2 \) denote as \( t_2 \)

\[ + (a_1 a'_1 + a_1 a'_2 + a_2 a'_1) \] – Computed by \( P_3 \) denote as \( t_3 \)

- \( P_1 \) shares \( t_1 = t_{1,1} + t_{1,2} + t_{1,3} \) and gives \((t_{1,j-1}, t_{1,j+1})\) to \( P_j \)
- \( P_2 \) shares \( t_2 = t_{2,1} + t_{2,2} + t_{2,3} \) and gives \((t_{2,j-1}, t_{2,j+1})\) to \( P_j \)
- \( P_3 \) shares \( t_3 = t_{3,1} + t_{3,2} + t_{3,3} \) and gives \((t_{3,j-1}, t_{3,j+1})\) to \( P_j \)

- \( P_1 \) has \((t_{1,2}, t_{1,3})\), \((t_{2,2}, t_{2,3})\), \((t_{3,2}, t_{3,3})\), \( P_2 \) has \((t_{1,1}, t_{1,3}), (t_{2,1}, t_{2,3}), (t_{3,1}, t_{3,3})\), \( P_3 \) has \((t_{1,1}, t_{1,2}), (t_{2,1}, t_{2,2}), (t_{3,1}, t_{3,2})\)
Replicated Scheme Multiplication
Example

- $\mathcal{P}_1$ has $(t_{1,2}, t_{1,3}), (t_{2,2}, t_{2,3}), (t_{3,2}, t_{3,3})$, $\mathcal{P}_2$ has $(t_{1,1}, t_{1,3}), (t_{2,1}, t_{2,3}), (t_{3,1}, t_{3,3})$, $\mathcal{P}_3$ has $(t_{1,1}, t_{1,2}), (t_{2,1}, t_{2,2}), (t_{3,1}, t_{3,2})$
- The parties compute $t_1 + t_2 + t_3$ on the secret shared values
  - Each party computes the sum of its respective shares
    - $\mathcal{P}_1$ computes $(t_{1,2} + t_{2,2} + t_{3,2}, t_{1,3} + t_{2,3} + t_{2,3}) = (s_2, s_3)$
    - $\mathcal{P}_2$ computes $(t_{1,1} + t_{2,1} + t_{3,1}, t_{1,3} + t_{2,3} + t_{2,3}) = (s_1, s_3)$
    - $\mathcal{P}_3$ computes $(t_{1,1} + t_{2,1} + t_{3,1}, t_{1,2} + t_{2,2} + t_{2,2}) = (s_1, s_2)$
  - $vv' = s_1 + s_2 + s_3$, the result is shared with replicated scheme
Additive Scheme Multiplication?

- Additive secret sharing is not multiplicative
- and many LSSS schemes are not
Active Security
Active Security

- Parties may deviate from the protocol
- The protocol must ensure that they can not or will be detected if they do
Shamir’s scheme allows to detect some errors.

Shamir’s scheme linear operations do not change the degree of the polynomial.
Shamir’s Linear Operations with Active Security

- Shamir’s scheme allows to detect some errors
- Shamir’s scheme linear operations do not change the degree of the polynomial
- Parties do not interact
  - If a party misbehaves then it only affects its shares
Shamir’s Linear Operations with Active Security

- Shamir’s scheme allows to detect some errors
- Shamir’s scheme linear operations do not change the degree of the polynomial
- Parties do not interact
  - If a party misbehaves then it only affects its shares
- The result of every linear combination can be verified the same as the original shared secrets
Active Security and Shamir

- Shamir’s scheme is also multiplicative
- But we can’t get active security MPC for free
Active Security and Shamir

- Shamir’s scheme is also multiplicative
- But we can’t get active security MPC for free
  - For example, possible to share a wrong value in the multiplication protocol
- Doable with verifiable variations of Shamir
Active Security and Shamir

- Shamir’s scheme is also multiplicative
- But we can’t get active security MPC for free
  - For example, possible to share a wrong value in the multiplication protocol
- Doable with verifiable variations of Shamir
- However, consider Shamir with $t < n/3 + 1$
Active Security and Shamir

- Shamir’s scheme is also multiplicative
- But we can’t get active security MPC for free
  - For example, possible to share a wrong value in the multiplication protocol
- Doable with verifiable variations of Shamir
- However, consider Shamir with $t < n/3 + 1$
  - Then we can use the error correcting properties
  - Number of errors is low
  - Multiplication result can be obtained from only honest parties
    (a.k.a strongly multiplicative secret sharing)
Commitment Schemes

- Cryptographic primitive to commit to a chosen value and be able to later reveal the value
  - The same party that computes the commitment can reveal

Given commitment \[a\] and \[b\] to \(a\) and \(b\) we can compute \[a + b\]

\[a + b\] is a valid commitment to \(a + b\)

The party that computed \[a\] and \[b\] can also reveal the commitment \[a + b\]
Commitment Schemes

- Cryptographic primitive to commit to a chosen value and be able to later reveal the value
  - The same party that computes the commitment can reveal
  - Binding commitment: you can not reveal another value instead
Commitment Schemes

- Cryptographic primitive to commit to a chosen value and be able to later reveal the value
  - The same party that computes the commitment can reveal
  - Binding commitment: you can not reveal another value instead
  - Hiding commitment: seeing the commitment does not reveal the committed value
Commitment Schemes

- Cryptographic primitive to commit to a chosen value and be able to later reveal the value
  - The same party that computes the commitment can reveal
  - Binding commitment: you can not reveal another value instead
  - Hiding commitment: seeing the commitment does not reveal the committed value
- Two main operations – Commit and Reveal
Commitment Schemes

- Cryptographic primitive to commit to a chosen value and be able to later reveal the value
  - The same party that computes the commitment can reveal
  - Binding commitment: you can not reveal another value instead
  - Hiding commitment: seeing the commitment does not reveal the committed value
- Two main operations – Commit and Reveal
- Linearly homomorphic commitment
  - Given commitment \([a]\) and \([b]\) to \(a\) and \(b\) we can compute \([a + b]\)
  - \([a + b]\) is a valid commitment to \(a + b\)
  - The party that computed \([a]\) and \([b]\) can also reveal the commitment \([a + b]\)
Commitment Schemes

- Cryptographic primitive to commit to a chosen value and be able to later reveal the value
  - The same party that computes the commitment can reveal
  - Binding commitment: you can not reveal another value instead
  - Hiding commitment: seeing the commitment does not reveal the committed value
  - Two main operations – Commit and Reveal

- Linearly homomorphic commitment
  - Given commitment \([a]\) and \([b]\) to \(a\) and \(b\) we can compute \([a + b]\)
  - \([a + b]\) is a valid commitment to \(a + b\)
  - The party that computed \([a]\) and \([b]\) can also reveal the commitment \([a + b]\)

- Let \([a]_i\) denote the commitment that can be revealed by \(P_i\)
Commitment Sharing Protocol (CSP)

- Commitment transfer (CTP) from \([a]_i\) to \([a]_j\)
  - \(P_i\) reveals \(a\) from \([a]_i\) to \(P_j\)
  - \(P_j\) computes \([a]_j\)
  - \(P_j\) computes \([a]_i - [a]_j\) and reveals to everyone that it is 0
    - It can do this since it received the revealing information for \([a]_i\)
Commitment Sharing Protocol (CSP)

- Commitment transfer (CTP) from $[a]_i$ to $[a]_j$
  - $\mathcal{P}_i$ reveals $a$ from $[a]_i$ to $\mathcal{P}_j$
  - $\mathcal{P}_j$ computes $[a]_j$
  - $\mathcal{P}_j$ computes $[a]_i - [a]_j$ and reveals to everyone that it is 0
    - It can do this since it received the revealing information for $[a]_i$
- CSP: Convert a committed value $[a]_i$ to commitments on the Shamir’s shares $[a_1]_1, \ldots, [a_n]_n$
  - Run Shamir on $a$ with $f(x) = a + \sum_{j=1}^{t} x^j b_j, \ a_i = f(i)$
  - Commit to $b_j$ as $[b_j]_i$
  - Compute $[a_\ell]_i = [a]_i + \sum_{j=1}^{t} \ell^j [b_j]_i$
Commitment Sharing Protocol (CSP)

- Commitment transfer (CTP) from \([a]_i\) to \([a]_j\)
  - \(P_i\) reveals \(a\) from \([a]_i\) to \(P_j\)
  - \(P_j\) computes \([a]_j\)
  - \(P_j\) computes \([a]_i - [a]_j\) and reveals to everyone that it is 0
    - It can do this since it received the revealing information for \([a]_i\)
- CSP: Convert a committed value \([a]_i\) to commitments on the Shamir’s shares \([a_1]_1, \ldots [a_n]_n\)
  - Run Shamir on \(a\) with \(f(x) = a + \sum_{j=1}^{t} x^j b_j\), \(a_i = f(i)\)
  - Commit to \(b_j\) as \([b_j]_i\)
  - Compute \([a_\ell]_i = [a]_i + \sum_{j=1}^{t} \ell^j [b_j]_i\)
  - Run CTP to get \([a]_j\) from \([a]_j\)
  - This ensures that \(a\) is shared with the polynomial of the right degree
Commitment Sharing Protocol (CSP)

- Commitment transfer (CTP) from $[a]_i$ to $[a]_j$
  - $P_i$ reveals $a$ from $[a]_i$ to $P_j$
  - $P_j$ computes $[a]_j$
  - $P_j$ computes $[a]_i - [a]_j$ and reveals to everyone that it is 0
    - It can do this since it received the revealing information for $[a]_i$

- CSP: Convert a committed value $[a]_i$ to commitments on the Shamir’s shares $[a_1]_1, \ldots [a_n]_n$
  - Run Shamir on $a$ with $f(x) = a + \sum_{j=1}^t x^j b_j$, $a_i = f(i)$
  - Commit to $b_j$ as $[b_j]_i$
  - Compute $[a_\ell]_i = [a]_i + \sum_{j=1}^t \ell^j [b_j]_i$
  - Run CTP to get $[a_j]_j$ from $[a_j]_i$
  - This ensures that $a$ is shared with the polynomial of the right degree

- If any of the Reveals fail then some party is corrupted
Commitment Multiplication Protocol (CMP)

- $\mathcal{P}_i$ proves that $c = ab$ based on $[a]_i$, $[b]_i$ and $[c]_i$
Commitment Multiplication Protocol (CMP)

- $P_i$ proves that $c = ab$ based on $[a]_i$, $[b]_i$ and $[c]_i$
- $P_i$ chooses a random $\beta$ and computes $[\beta]_i$, $[\beta b]_i$
- All other parties collectively choose a random $r \neq 0$
- Parties compute $[r_1] = r[a]_i + [\beta]_i$ and $P_i$ reveals $r_1$
- Parties compute $r_1[b]_i - [\beta b]_i - r[c]_i$, $P_i$ reveals it as 0
Commitment Multiplication Protocol (CMP)

- $P_i$ proves that $c = ab$ based on $[a]_i$, $[b]_i$ and $[c]_i$
- $P_i$ chooses a random $\beta$ and computes $[\beta]_i$, $[\beta b]_i$
- All other parties collectively choose a random $r \neq 0$
- Parties compute $[r_1] = r[a]_i + [\beta]_i$ and $P_i$ reveals $r_1$
- Parties compute $r_1[b]_i - [\beta b]_i - r[c]_i$, $P_i$ reveals it as 0
- If all Reveals are accepted then $c = ab$
  - $r_1 b - \beta b - rc = (ra + \beta)b - \beta b - rc = rab - rc \neq 0$
MPC with Commitments
MPC with Commitments

- Sharing \( v \) by \( P_j \)
  - Run Shamir on \( v \) with \( f(x) = v + \sum_{j=1}^{t} x^j b_j, \ v_i = f(i) \)
  - Commit to \( b_j \) as \([b_j]_i\)
  - Compute \([v\ell]_i = [v]_i + \sum_{j=1}^{t} \ell^j [b_j]_i\)
MPC with Commitments

- Sharing v by \( P_j \)
  - Run Shamir on v with \( f(x) = v + \sum_{j=1}^{t} x^j b_j, \) \( v_i = f(i) \)
  - Commit to \( b_j \) as \([b_j]_i\)
  - Compute \([v_\ell]_i = [v]_i + \sum_{j=1}^{t} \ell^j [b_j]_i\)
  - Run CTP to get \([v_j]_j\) from \([v_j]_i\)
MPC with Commitments

- Sharing $v$ by $P_j$
  - Run Shamir on $v$ with $f(x) = v + \sum_{j=1}^{t} x^j b_j$, $v_i = f(i)$
  - Commit to $b_j$ as $[b_j]_i$
  - Compute $[v_{\ell}]_i = [v]_i + \sum_{j=1}^{t} \ell^j [b_j]_i$
  - Run CTP to get $[v_j]_j$ from $[v_j]_i$

- Addition $a + b$
  - Each party $P_i$ computes $a_i + b_i$ for the new share
  - Using homomorphic properties all parties compute $[a_i + b_i]_i$ for each $i$
MPC with Commitments

- Sharing $v$ by $P_j$
  - Run Shamir on $v$ with $f(x) = v + \sum_{j=1}^{t} x^j b_j$, $v_i = f(i)$
  - Commit to $b_j$ as $[b_j]_i$
  - Compute $[v_\ell]_i = [v]_i + \sum_{j=1}^{t} \ell^j [b_j]_i$
  - Run CTP to get $[v_j]_i$ from $[v_j]_i$

- Addition $a + b$
  - Each party $P_i$ computes $a_i + b_i$ for the new share
  - Using homomorphic properties all parties compute $[a_i + b_i]_i$ for each $i$

- Multiplication
  - Party $P_i$ computes $\overline{c}_i = a_i \cdot b_i$ and commits to it $[\overline{c}_i]_i$
MPC with Commitments

- **Sharing v by P_j**
  - Run Shamir on v with \( f(x) = v + \sum_{j=1}^{t} x^j b_j \), \( v_i = f(i) \)
  - Commit to \( b_j \) as \([b_j]_i\)
  - Compute \([v_l]_i = [v]_i + \sum_{j=1}^{t} \ell^j [b_j]_i\)
  - Run CTP to get \([v_j]_i\) from \([v_j]_i\)

- **Addition a + b**
  - Each party \( P_i \) computes \( a_i + b_i \) for the new share
  - Using homomorphic properties all parties compute \([a_i + b_i]_i\) for each \( i \)

- **Multiplication**
  - Party \( P_i \) computes \( \bar{c}_i = a_i \cdot b_i \) and commits to it \([\bar{c}_i]_i\)
  - Party \( P_i \) performs CMP on \([a_i]_i, [b_i]_i, [\bar{c}_i]_i\)
MPC with Commitments

- Sharing $v$ by $P_j$
  - Run Shamir on $v$ with $f(x) = v + \sum_{j=1}^{t} x^j b_j$, $v_i = f(i)$
  - Commit to $b_j$ as $[b_j]_i$
  - Compute $[v_\ell]_i = [v]_i + \sum_{j=1}^{t} \ell^j [b_j]_i$
  - Run CTP to get $[v_j]_i$ from $[v_j]_i$
- Addition $a + b$
  - Each party $P_i$ computes $a_i + b_i$ for the new share
  - Using homomorphic properties all parties compute $[a_i + b_i]_i$ for each $i$
- Multiplication
  - Party $P_i$ computes $\overline{c}_i = a_i \cdot b_i$ and commits to it $[\overline{c}_i]_i$
  - Party $P_i$ performs CMP on $[a_i]_i, [b_i]_i, [\overline{c}_i]_i$
  - Resharing: CSP on $[\overline{c}_i]_i$ giving $[c_{i1}]_1, \ldots, [c_{in}]_n$ and respective shares $c_{ij}$
MPC with Commitments

- **Sharing v by P_j**
  - Run Shamir on v with \( f(x) = v + \sum_{j=1}^{t} x^j b_j \), \( v_i = f(i) \)
  - Commit to \( b_j \) as \([b_j]_i\)
  - Compute \([v_\ell]_i = [v]_i + \sum_{j=1}^{t} \ell^j[b_j]_i\)
  - Run CTP to get \([v_j]_i\) from \([v_j]_i\)

- **Addition a + b**
  - Each party \( P_i \) computes \( a_i + b_i \) for the new share
  - Using homomorphic properties all parties compute \([a_i + b_i]_i\) for each \( i \)

- **Multiplication**
  - Party \( P_i \) computes \( \overline{c}_i = a_i \cdot b_i \) and commits to it \([\overline{c}_i]_i\)
  - Party \( P_i \) performs CMP on \([a_i]_i, [b_i]_i, [\overline{c}_i]_i\)
  - Resharing: CSP on \([\overline{c}_i]_i\) giving \([c_{i1}]_1, \ldots, [c_{in}]_n\) and respective shares \( c_{ij} \)
  - Recombination: Player \( P_j \) computes \( c_j = \sum_{i=1}^{n} r_i c_{ij} \)
    - There exists some reconstruction vector \( r_i \) for the LSSS
    - \([c_j]_j = \sum r_i[c_{ij}]_j = [\sum r_i c_{ij}]_j\) is computed non-interactively by all \( P_i \)
Shamir’s Secret Sharing as Commitment

- Commit to $a_1, \ldots, a_n$ with $(t, n)$ Shamir sharing scheme

Prove that the degree of the polynomial is $\leq t - 1$

Open (to party $P_j$)

All parties send their shares either to everyone or to designated party $P_j$

Committer sends the polynomial

Opening succeeds to the value $a$ if at most $t - 1$ values are not according to the polynomial

Linearity comes from the Shamir’s scheme

Hiding for all sets of parties less than the threshold

Binding if the polynomial has the right degree
Shamir’s Secret Sharing as Commitment

- Commit to $a$
  - Share $a$ as $a_1, \ldots, a_n$ with $(t, n)$ Shamir sharing scheme
  - Prove that the degree of the polynomial is $\leq t - 1$
Shamir’s Secret Sharing as Commitment

- **Commit to** $a$
  - Share $a$ as $a_1, \ldots, a_n$ with $(t, n)$ Shamir sharing scheme
  - Prove that the degree of the polynomial is $\leq t - 1$
- **Open (to party $P_j$)**
  - All parties send their shares
    - either to everyone or to designated party $P_j$
  - Committer sends the polynomial
  - Opening succeeds to the value $a$ if at most $t - 1$ values are not according to the polynomial
Shamir’s Secret Sharing as Commitment

- Commit to $a$
  - Share $a$ as $a_1, \ldots, a_n$ with $(t, n)$ Shamir sharing scheme
  - Prove that the degree of the polynomial is $\leq t - 1$

- Open (to party $P_j$)
  - All parties send their shares
    - either to everyone or to designated party $P_j$
  - Committer sends the polynomial
  - Opening succeeds to the value $a$ if at most $t - 1$ values are not according to the polynomial

- Linearity comes from the Shamir’s scheme
Shamir’s Secret Sharing as Commitment

- Commit to \( a \)
  - Share \( a \) as \( a_1, \ldots, a_n \) with \((t, n)\) Shamir sharing scheme
  - Prove that the degree of the polynomial is \( \leq t - 1 \)

- Open (to party \( \mathcal{P}_j \))
  - All parties send their shares
    - either to everyone or to designated party \( \mathcal{P}_j \)
  - Committer sends the polynomial
  - Opening succeeds to the value \( a \) if at most \( t - 1 \) values are not according to the polynomial

- Linearity comes from the Shamir’s scheme
- Hiding for all sets of parties less than the threshold
- Binding if the polynomial has the right degree
Proving the Degree of the Polynomial

- \( P_i \) has a random polynomial with \( f(0) = a \) and \( \deg(f) \leq t - 1 \)
Proving the Degree of the Polynomial

- \( \mathcal{P}_i \) has a random polynomial with \( f(0) = a \) and \( \deg(f) \leq t - 1 \)
- Define a bivariate polynomial \( F(x, y) = \sum_{i,j=0}^{t-1} c_{ij} x^i y^j \) where
  - \( F(x, 0) = f(x) \), meaning \( c_{00} = a \) and \( f(x) = \sum c_{i0} x^i \)
  - Symmetric \( c_{kl} = c_{lk} \), meaning \( F(i, j) = F(j, i) \)
Proving the Degree of the Polynomial

- \( \mathcal{P}_i \) has a random polynomial with \( f(0) = a \) and \( \deg(f) \leq t - 1 \)
- Define a bivariate polynomial \( F(x, y) = \sum_{i,j=0}^{t-1} c_{ij}x^i y^j \) where
  - \( F(x, 0) = f(x) \), meaning \( c_{00} = a \) and \( f(x) = \sum c_{i0}x^i \)
  - Symmetric \( c_{k\ell} = c_{\ell k} \), meaning \( F(i, j) = F(j, i) \)
- \( \mathcal{P}_i \) sends \( F(x, j) \) with \( \deg(F(x, j)) \leq t - 1 \) to \( \mathcal{P}_j \), \( \mathcal{P}_j \) can compute \( F(0, j) = F(0, j) \)
Proving the Degree of the Polynomial

- \( P_i \) has a random polynomial with \( f(0) = a \) and \( \deg(f) \leq t - 1 \)
- Define a bivariate polynomial \( F(x, y) = \sum_{i,j=0}^{t-1} c_{ij} x^i y^j \) where
  - \( F(x, 0) = f(x) \), meaning \( c_{00} = a \) and \( f(x) = \sum c_{i0} x^i \)
  - Symmetric \( c_{k\ell} = c_{\ell k} \), meaning \( F(i,j) = F(j,i) \)
- \( P_i \) sends \( F(x, j) \) with \( \deg(F(x, j)) \leq t - 1 \) to \( P_j \), \( P_j \) can compute \( F(0, j) = F(0, j) \)
- Each pair \( P_j \) and \( P_k \) compare \( F(k, j) \) and \( F(j, k) \)
  - They complain and and \( P_i \) publishes \( \sigma = F(k, j) \)
  - \( P_j \) and \( P_k \) compare their values to \( \sigma \), if not equal then call to disqualify \( P_i \)
Proving the Degree of the Polynomial

- \( \mathcal{P}_i \) has a random polynomial with \( f(0) = a \) and \( \deg(f) \leq t - 1 \)
- Define a bivariate polynomial \( F(x, y) = \sum_{i,j=0}^{t-1} c_{ij} x^i y^j \) where
  - \( F(x, 0) = f(x) \), meaning \( c_{00} = a \) and \( f(x) = \sum c_{i0} x^i \)
  - Symmetric \( c_{k\ell} = c_{\ell k} \), meaning \( F(i, j) = F(j, i) \)
- \( \mathcal{P}_i \) sends \( F(x, j) \) with \( \deg(F(x, j)) \leq t - 1 \) to \( \mathcal{P}_j \), \( \mathcal{P}_j \) can compute \( F(0, j) = F(0, j) \)
- Each pair \( \mathcal{P}_j \) and \( \mathcal{P}_k \) compare \( F(k, j) \) and \( F(j, k) \)
  - They complain and \( \mathcal{P}_i \) publishes \( \sigma = F(k, j) \)
  - \( \mathcal{P}_j \) and \( \mathcal{P}_k \) compare their values to \( \sigma \), if not equal then call to disqualify \( \mathcal{P}_i \)
    - In response to disqualification from \( \mathcal{P}_m \), \( \mathcal{P}_i \) publishes \( F(x, m) \)
    - All \( \mathcal{P}_\ell \) check that \( F(\ell, j) = F(j, \ell) \) and \( \deg(F(x, m)) \leq t - 1 \)
    - In case of any inequality the party \( \mathcal{P}_\ell \) calls to disqualify \( \mathcal{P}_i \)
Proving the Degree of the Polynomial

- $\mathcal{P}_i$ has a random polynomial with $f(0) = a$ and $\deg(f) \leq t - 1$
- Define a bivariate polynomial $F(x, y) = \sum_{i,j=0}^{t-1} c_{ij} x^i y^j$ where
  - $F(x, 0) = f(x)$, meaning $c_{00} = a$ and $f(x) = \sum c_{i0} x^i$
  - Symmetric $c_{k\ell} = c_{\ell k}$, meaning $F(i, j) = F(j, i)$
- $\mathcal{P}_i$ sends $F(x, j)$ with $\deg(F(x, j)) \leq t - 1$ to $\mathcal{P}_j$, $\mathcal{P}_j$ can compute $F(0, j) = F(0, j)$
- Each pair $\mathcal{P}_j$ and $\mathcal{P}_k$ compare $F(k, j)$ and $F(j, k)$
  - They complain and $\mathcal{P}_i$ publishes $\sigma = F(k, j)$
  - $\mathcal{P}_j$ and $\mathcal{P}_k$ compare their values to $\sigma$, if not equal then call to disqualify $\mathcal{P}_i$
    - In response to disqualification from $\mathcal{P}_m$, $\mathcal{P}_i$ publishes $F(x, m)$
    - All $\mathcal{P}_\ell$ check that $F(\ell, j) = F(j, \ell)$ and $\deg(F(x, m)) \leq t - 1$
    - In case of any inequality the party $\mathcal{P}_\ell$ calls to disqualify $\mathcal{P}_i$
- If there are at least $t$ disqualifications then the proof fails, otherwise it succeeds
  - Complaining parties update their values to $\sigma$ in case of success
Privacy of the Proof

- Need that if \( P_i \) is honest then adversary learns only \( F(x, j) \) for corrupted \( P_j \) and nothing else.
**Privacy of the Proof**

- Need that if $\mathcal{P}_i$ is honest then adversary learns only $F(x, j)$ for corrupted $\mathcal{P}_j$ and nothing else.
- If $\mathcal{P}_i$ is honest then even if there are complaints it publishes $F(x, j)$ for corrupted party $\mathcal{P}_j$.
  - Adversary already knows this since this is given to $\mathcal{P}_j$ in the protocol.
  - Complain means that one of $\mathcal{P}_j$, $\mathcal{P}_k$ or $\mathcal{P}_i$ is corrupted.
  - Only adversarial party calls to disqualify honest $\mathcal{P}_i$.

Could seeing $t-1$ different $F(x, j)$ reveal the secret?

$F(x, y)$ is uniquely determined by $t^2$ points.
Privacy of the Proof

- Need that if $P_i$ is honest then adversary learns only $F(x, j)$ for corrupted $P_j$ and nothing else.
- If $P_i$ is honest then even if there are complaints it publishes $F(x, j)$ for corrupted party $P_j$.
  - Adversary already knows this since this is given to $P_j$ in the protocol.
  - Complain means that one of $P_j$, $P_k$ or $P_i$ is corrupted.
  - Only adversarial party calls to disqualify honest $P_i$.
- In case of the complaint honest party also publishes $F(j, k)$.
  - But since $P_j$ or $P_k$ is corrupted then adversary has already seen this value.
- Could seeing $t-1$ different $F(x, j)$ reveal the secret?
  - $F(x, y)$ is uniquely determined by $t^2$ points.
Corrupted view of $F(x, y)$

- Could seeing $t - 1$ different $F(x, j)$ reveal the secret?
- Corrupted party has $F(x, j)$ of degree $\leq t - 1$ where they can compute up to $t$ independent points.
- Adversary can corrupt up to $t - 1$ parties.
- In total adversary can have $t(t - 1)$ independent points of the polynomial.
- Choosing any value for $F(0, 0)$ gives a point that can be used to define a new polynomial, hence the secret value is not leaked.
Correctness of the Proof

- Need that if the proof is accepted then the shares of honest parties (set $B$) are defined by $f(x)$ where $\deg(f) \leq t - 1$
- Acceptance means that less than $t$ disqualifications were filed
Correctness of the Proof

- Need that if the proof is accepted then the shares of honest parties (set \( B \)) are defined by \( f(x) \) where \( \text{deg}(f) \leq t - 1 \)
- Acceptance means that less than \( t \) disqualifications were filed
- We have \( t < n/3 + 1 \), hence also at least \( t \) honest parties did not call to disqualify, let them be set \( C \)
Correctness of the Proof

○ Need that if the proof is accepted then the shares of honest parties (set \( B \)) are defined by \( f(x) \) where \( \text{deg}(f) \leq t - 1 \)
○ Acceptance means that less than \( t \) disqualifications were filed
○ We have \( t < n/3 + 1 \), hence also at least \( t \) honest parties did not call to disqualify, let them be set \( C \)
○ For all honest parties \( P_\ell \) and at least \( t \) honest parties \( P_k \) \( k \in C \) we have \( F(\ell, k) = F(k, \ell) \)
  ○ Either they were equal or they were updated to be
○ Let \( r_i \) be the Lagrange interpolation coefficients for \( i \in C \) for polynomials of degree up to \( t - 1 \)
  ○ \( r_i \) exist, otherwise some \( P_k \) \( k \in C \) would have accused the dealer
    ○ due to some mismatch (and complaint) among their own comparisons
○ Define \( f(X) = \sum_{i \in C} r_i F(X, i) \) with \( \text{deg}(f) \leq t - 1 \)
○ \( j \in B : f(j) = \sum_{i \in C} r_i F(j, i) = \sum_{i \in C} r_i F(i, j) = F(0, j) \)
  ○ \( \sum_{i \in C} r_i F(i, j) = F(0, j) \) from Lagrange and \( |C| \geq t \)
○ All points of \( B \) are on the polynomial \( f(x) \) with \( \text{deg}(f) \leq t - 1 \)
Conclusion

- Secret sharing schemes and error correcting codes are similar.
- Multiplicative linear secret sharing is a good basis for secure computation.
- Combining multiplicative property and error correcting properties can give secure computation schemes for active security.
More efficient construction

- A value $v$ is shared by a bivariate polynomial $F$
  - degree $\leq (t - 1)$ in each variable
  - $F(0, 0) = v$
  - $i$-th parth knows $F(X, i)$ and $F(i, Y)$
- Let $v$ be shared by $F$ and $v'$ by $F'$
- In multiplication protocol, $i$-th party shares the value $F(0, i) \cdot F'(0, i)$ using some polynomial $C_i(\cdot, \cdot)$
  - Must verify that degree of $C_i$ is small
    - We just saw this
- Parties verify that $C_i$ indeed shares $F(0, i) \cdot F'(0, i)$
- Parties linearly combine $C_1, \ldots, C_n$ using interpolation coefficients
Equality check for party $P_i$

- $P_i$ picks random bivariate polynomials $D_1, \ldots, D_{t-1}$, such that
  - they have degree $\leq (t - 1)$ in each variable
  - $C(X, 0) = F(X, i) \cdot F'(X, i) - \sum_{k=1}^{t-1} X^k D_k(X, 0)$

- $P_i$ “verifiably shares” the polynomials $D_1, \ldots, D_{t-1}$

- Party $P_j$ verifies the above equality for $X \mapsto j$
  - If something does not match, then there will be complaints broadcast
More efficient equality check

- $P_i$ verifiable shares a single bivariate polynomial $D$, where
  - Degree of $D$ in variable $X$ is $\leq (2t - 2)$
  - Degree of $D$ in variable $Y$ is $\leq (t - 1)$
  - $D(0, 0) = 0$
  - $C(X, 0) = F(X, i) \cdot F'(X, i) - D(X, 0)$

- Degree verification is a bit different, with weaker guarantees
- $D(0, 0)$ must be collectively evaluated. $P_i$ must assist in this
Exercise: Access Structures

Which of the following are valid access structures?

1. \{\mathcal{P}_1, \mathcal{P}_2\}

Which of the access structures are threshold schemes? What are their thresholds?

- Scheme 1 - (2, 2) threshold secret sharing scheme
- Scheme 5 - (2, 4) threshold secret sharing scheme
Exercise: Access Structures

Which of the following are valid access structures?
1. \{\{P_1, P_2\}\}
2. \{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}

7. Valid are 1, 2, 4, 5. Closure of 3 would be an access structure.

Which of the access structures are threshold schemes? What are their thresholds?

Scheme 1 - \((2, 2)\) threshold secret sharing scheme
Scheme 5 - \((2, 4)\) threshold secret sharing scheme
Exercise: Access Structures

Which of the following are valid access structures?

1. $\{\{P_1, P_2\}\}$
2. $\{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}$
3. $\{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}$

Which of the access structures are threshold schemes? What are their thresholds?

Scheme 1 - (2, 2) threshold secret sharing scheme
Scheme 5 - (2, 4) threshold secret sharing scheme
Exercise: Access Structures

Which of the following are valid access structures?

1. \{\{P_1, P_2\}\}
2. \{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}
3. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}
4. \{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}

Which of the access structures are threshold schemes? What are their thresholds?

- Scheme 1 - \((2, 2)\) threshold secret sharing scheme
- Scheme 5 - \((2, 4)\) threshold secret sharing scheme
Exercise: Access Structures

Which of the following are valid access structures?

1. \{\{P_1, P_2\}\}
2. \{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}
3. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}
4. \{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}
5. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}

Which of the access structures are threshold schemes? What are their thresholds?

Scheme 1 - (2, 2) threshold secret sharing scheme
Scheme 5 - (2, 4) threshold secret sharing scheme
Exercise: Access Structures

Which of the following are valid access structures?

1. \{\{P_1, P_2\}\}
2. \{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}
3. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}
4. \{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}
5. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}
6. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}\}

Which of the access structures are threshold schemes? What are their thresholds?

Scheme 1 - \((2, 2)\) threshold secret sharing scheme
Scheme 5 - \((2, 4)\) threshold secret sharing scheme
Exercise: Access Structures

- Which of the following are valid access structures?
  1. \{\{P_1, P_2\}\}
  2. \{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}
  3. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}
  4. \{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\},
      \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}
  5. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\},
      \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}\{P_1, P_3, P_4\}, \{P_2, P_3, P_4\},
      \{P_1, P_2, P_3, P_4\}\}
  6. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\},
      \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}\{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}\}
  7. Valid are 1, 2, 4, 5. Closure of 3 would be an access structure

- Which of the access structures are threshold schemes? What are their thresholds?
Exercise: Access Structures

Which of the following are valid access structures?

1. \{\{P_1, P_2\}\}
2. \{\{P_1\}, \{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_1, P_2, P_3\}\}
3. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}
4. \{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}, \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\}
5. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}\{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}\}
6. \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}, \{P_3, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\}\{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}\}
7. Valid are 1, 2, 4, 5. Closure of 3 would be an access structure

Which of the access structures are threshold schemes? What are their thresholds?

- Scheme 1 - (2, 2) threshold secret sharing scheme
- Scheme 5 - (2, 4) threshold secret sharing scheme
Exercise: Compute additive shares two ways

- Three parties, $t = 3$, sharing modulo 13, $v = 7$
- According to LSSS description of additive sharing

\[
M = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}, \quad w^T = (1, 1, 1), \quad \chi(i) = i
\]
Exercise: Compute additive shares two ways

- Three parties, \( t = 3 \), sharing modulo 13, \( v = 7 \)
- According to LSSS description of additive sharing

\[
M = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}, \quad w^T = (1, 1, 1), \quad \chi(i) = i
\]

- Sharing \( v = (1, 1, 1) \cdot (a_1, a_2, a_3)^T = a_1 + a_2 + a_3 \mod 13 \)
  - Choose \( a_i \) randomly, e.g. let \( a_1 = 6, a_2 = 5, a_3 = 9 \)
  - Shares \( (s_1, s_2, s_3)^T = M \cdot (a_1, a_2, a_3)^T = (6, 5, 9)^T \)
  - Reconstruction
Exercise: Compute additive shares two ways

- Three parties, \( t = 3 \), sharing modulo 13, \( v = 7 \)
- According to LSSS description of additive sharing

\[
M = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}, \quad w^T = (1, 1, 1), \quad \chi(i) = i
\]

- Sharing \( v = (1, 1, 1) \cdot (a_1, a_2, a_3)^T = a_1 + a_2 + a_3 \mod 13 \)
  - Choose \( a_i \) randomly, e.g. let \( a_1 = 6, \ a_2 = 5, \ a_3 = 9 \)
  - Shares \( (s_1, s_2, s_3)^T = M \cdot (a_1, a_2, a_3)^T = (6, 5, 9)^T \)
  - Reconstruction

- By more common definition:
  - Choose \( s_1, s_2 \) randomly, let \( s_3 = v - s_1 - s_2 \mod 13 \)
Exercise: Compute additive shares two ways

- Three parties, \( t = 3 \), sharing modulo 13, \( v = 7 \)
- According to LSSS description of additive sharing

\[
M = \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}, \quad w^T = (1, 1, 1), \quad \chi(i) = i
\]

- Sharing \( v = (1, 1, 1) \cdot (a_1, a_2, a_3)^T = a_1 + a_2 + a_3 \mod 13 \)
  - Choose \( a_i \) randomly, e.g. let \( a_1 = 6, a_2 = 5, a_3 = 9 \)
  - Shares \( (s_1, s_2, s_3)^T = M \cdot (a_1, a_2, a_3)^T = (6, 5, 9)^T \)
  - Reconstruction

- By more common definition:
  - Choose \( s_1, s_2 \) randomly, let \( s_3 = v - s_1 - s_2 \mod 13 \)
  - Can you get the same shares for the two cases?
Computing Shamir LSSS way

- Four parties, $t = 3$, sharing modulo 13, $v = 7$
Computing Shamir LSSS way

- Four parties, $t = 3$, sharing modulo 13, $v = 7$
- According to LSSS description

$$M = \begin{pmatrix} 1 & 1 & 1^2 \\ 1 & 2^1 & 2^2 \\ 1 & 3^1 & 3^2 \\ 1 & 4^1 & 4^2 \end{pmatrix}, \quad w^T = (1, 0, 0), \quad \chi(i) = i$$
Computing Shamir LSSS way

- Four parties, $t = 3$, sharing modulo 13, $v = 7$
- According to LSSS description

$$M = \begin{pmatrix} 1 & 1 & 1^2 \\ 1 & 2^1 & 2^2 \\ 1 & 3^1 & 3^2 \\ 1 & 4^1 & 4^2 \end{pmatrix}, \quad w^T = (1, 0, 0), \quad \chi(i) = i$$

- Sharing $v = (1, 0, 0) \cdot (a_1, a_2, a_3)^T = a_1$
  - Choose random $a_2, a_3$, e.g. let $a_1 = 7, a_2 = 8, a_3 = 1$
  - $(s_1, s_2, s_3, s_4)^T = M \cdot (a_1, a_2, a_3)^T = (3, 1, 1, 3)^T$
  - $P_i$ gets $s_i$
Reconstructing Shamir LSSS way

- Reconstruction by parties $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$
Reconstructing Shamir LSSS way

- Reconstruction by parties $P_1$, $P_2$, $P_3$
  - Need $x_A$ such that $w^T = x_A^T \cdot M_A$

$$M_{P_1, P_2, P_3} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{pmatrix},$$

$$x_A^T \cdot M_A = (x_1 + x_2 + x_3, x_1 + 2x_2 + 3x_3, x_1 + 4x_2 + 9x_3)$$

$$= (1, 0, 0)$$

solve the linear equations

$$x_1 = 3, \quad x_2 = 10, \quad x_3 = 1$$

- Reconstruction: $x_A^T \cdot s_A$

$$(3, 10, 1) \cdot (3, 1, 1)^T = 9 + 10 + 1 = 20 = 7 \mod 13$$
Exercise: Shamir polynomial vs LSSS

- By our initial definition of Shamir, we had to choose a polynomial
- Which polynomial gives the same result as the LSSS matrix definition?
Exercise: Shamir polynomial vs LSSS

- By our initial definition of Shamir, we had to choose a polynomial.
- Which polynomial gives the same result as the LSSS matrix definition?
- Solution: The choice of $a_i$ is the same as the polynomial choice.
Exercise: Design a perfect sharing scheme

- for the access structure
  \[
  \{\{P_1, P_2\}, \{P_2, P_3\}, \{P_1, P_4\}, \{P_1, P_2, P_3\}, \{P_1, P_2, P_4\},
  \{P_1, P_3, P_4\}, \{P_2, P_3, P_4\}, \{P_1, P_2, P_3, P_4\}\} 
  \]

- Hint: Additive secret sharing and Shamir’s sharing are perfect
Exercise: Design a perfect sharing scheme

- for the access structure

\{\{\mathcal{P}_1, \mathcal{P}_2\}, \{\mathcal{P}_2, \mathcal{P}_3\}, \{\mathcal{P}_1, \mathcal{P}_4\}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_4\}, \{\mathcal{P}_1, \mathcal{P}_3, \mathcal{P}_4\}, \{\mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\}, \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\}\}

- Hint: Additive secret sharing and Shamir’s sharing are perfect

- Solution:

  - We only have three sets to focus on \{\mathcal{P}_1, \mathcal{P}_2\}, \{\mathcal{P}_2, \mathcal{P}_3\}, \{\mathcal{P}_1, \mathcal{P}_4\}
  - Use a perfect \((n, n)\) threshold scheme for each
    - e.g. additive scheme: \(v = a_1 + a_2, v = b_1 + b_2, v = c_1 + c_2\)
    - \(\mathcal{P}_1\) gets \(a_1, c_1\)
    - \(\mathcal{P}_2\) gets \(a_2, b_1\)
    - \(\mathcal{P}_3\) gets \(b_2\)
    - \(\mathcal{P}_4\) gets \(c_2\)
Exercise: Design an ideal secret sharing scheme

- for access structure with minimal sets
  \[\{(P_1, P_2), (P_1, P_3), (P_2, P_3), (P_4, P_5)\}\]
  - The real access structure is the closure of this set
- A secret sharing scheme is ideal if the share of a party has the same size as the secret
- Hint: Shamir’s scheme and additive schemes are ideal
Exercise: Design an ideal secret sharing scheme

- for access structure with minimal sets
  \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}, \{P_4, P_5\}\}
  - The real access structure is the closure of this set
- A secret sharing scheme is ideal if the share of a party has the same size as the secret
- Hint: Shamir’s scheme and additive schemes are ideal
- Solution:
  - Use one ideal (2,3) threshold scheme for
    \{\{P_1, P_2\}, \{P_1, P_3\}, \{P_2, P_3\}\}
  - And the other (2,2) threshold scheme for \{P_4, P_5\}
Exercise: Cheating in Shamir-based Commitment

- During reveal $f(x)$ and all values $a_i = f(i)$ were revealed
- Revealing accepted if at most $t - 1$ locations had $a_i \neq f(i)$
- Assume that $n = 3(t - 1)$ or $t = \frac{n+3}{3} \geq n/3 + 1$
  - Contradiction with the requirement of our protocol
  - e.g. $t = 3$, $n = 6$
- How can a corrupted party choose two polynomials $f(x)$ and $f'(x)$ of degree at most $t - 1$ such that it can send values $\bar{a}_i$ to parties and later Reveal either $f(x)$ or $f'(x)$
Exercise: Cheating in Shamir-based Commitment

- During reveal \( f(x) \) and all values \( a_i = f(i) \) were revealed
- Revealing accepted if at most \( t - 1 \) locations had \( a_i \neq f(i) \)
- Assume that \( n = 3(t - 1) \) or \( t = \frac{n+3}{3} \geq n/3 + 1 \)
  - Contradiction with the requirement of our protocol
  - e.g. \( t = 3, n = 6 \)
- How can a corrupted party choose two polynomials \( f(x) \) and \( f'(x) \) of degree at most \( t - 1 \) such that it can send values \( \overline{a}_i \) to parties and later Reveal either \( f(x) \) or \( f'(x) \)
  - Let \( P_1, \ldots, P_{n-t+1} \) be the honest parties
  - Give \( \overline{a}_i = f(i) \) to \( i \leq t - 1 \)
    - These parties will accept the opening of \( f(x) \) and object to \( f'(x) \)
  - Give \( \overline{a}_i = f'(i) \) to \( t \geq i \leq n - t + 1 \)
    - These parties will accept the opening of \( f'(x) \) and object to \( f(x) \)
    - There are \( n - t - 1 - t - 1 = 3(t - 1) - 2(t - 1) = (t - 1) \) parties
  - Corrupted parties will accept either opening
Cheating with Shamir-based Commitment 2

- Using Shamir’s scheme as a commitment required proof of the degree of the polynomial \( \deg(f) \leq t - 1 \)
- Suppose that we now have \( t < \frac{n}{3} + 1 \) as required
- How can a corrupted party choose two polynomials \( f(x) \) and \( f'(x) \) of degree at most \( t - 1 \) such that it can send values \( \bar{a}_i \) to parties and later Reveal either \( f(x) \) or \( f'(x) \)
- Assume, for simplicity, that \( n = 3(t - 1) + 1 \) (e.g. \( t = 2 \))
Cheating with Shamir-based Commitment 2

- Using Shamir’s scheme as a commitment required proof of the degree of the polynomial $\deg(f) \leq t - 1$
- Suppose that we now have $t < n/3 + 1$ as required
- How can a corrupted party choose two polynomials $f(x)$ and $f'(x)$ of degree at most $t - 1$ such that it can send values $\tilde{a}_i$ to parties and later Reveal either $f(x)$ or $f'(x)$
- Assume, for simplicity, that $n = 3(t - 1) + 1$ (e.g. $t = 2$)
  - Similarly to the previous exercise
  - With the addition that $f(i) = f'(i)$ for some values
  - Two degree $t - 1$ polynomials can have up to $t - 1$ common points and still differ
    - We want $f(0) \neq f'(0)$
    - Hence, we can have $f(i) = f'(i)$ for $t - 1$ values of $i$
  - Give half of the parties $f(i)$ and others $f'(i)$
  - In either case only $n - n/2 - (t - 1) = n/2 - t + 1 = \frac{t - 1}{2}$ parties might object to the Reveal
- Hence, proving the degree of the polynomial is important
References and Extra Materials I

  https://doi.org/10.1017/CBO9781107337756
- Smart: Secure MPC based on Secret Sharing 2018
- Laud: Secret-sharing based secure multiparty computation with malicious adversaries 2019
References and Extra Materials II

- Cramer, Damgård, Maurer: General Secure Multi-Party Computation from any Linear Secret-Sharing Scheme 2000

- McEliece, Sarwate: On Sharing Secrets and Reed-Solomon Codes 1981
  http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.2829&rep=rep1&type=pdf

- Benaloh, Leichter: Generalized secret sharing and monotone functions 1988

References and Extra Materials III

- Ben-Or, Goldwasser, Wigderson Completeness theorems for non-cryptographic fault-tolerant distributed computation 1988 https://doi.org/10.1145/62212.62213