Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation

Michael Ben-Or
Shafi Goldwasser
Avi Wigderson

Lecture: Mickey Hakimi

The Model:

- A complete synchronous network of n processors.
- The communication channels between players are secure.
- A function *f* from n inputs to n outputs, the player *i* holds the *i-th* input.

Synchronous network

- The network is controlled by a global clock, divided into rounds.
- In one round of computation each of the players can do:
  - An arbitrary amount of local computation.
  - Send a message to each of the players.
  - Read all messages that were sent to it in this round.

Synchronous network

- There are two kinds of synchronous network:

1. In each round all processors send there messages exactly at the same time.
2. A processor can receive a message before he sends his own message.

- We will deal with the second kind of network.

The problem:

- Compute the output of the function *f* such that player *i* obtains the *i-th* output at the end, **but nothing else**.

- We distinguish two kinds of faults:
  1. Gossip (honest but curious).
  2. Byzantine.

The kind of faults

- Gossip
  - Act according to their predetermined programs.
  - Try to learn as much as they can by sharing the information they received.
  - Two kinds:
    1. Use the original received input.
    2. Change the original input with a new one.

    The protocol we will see handles both kinds.

The kind of faults

- Byzantine
  - Can use totally different programs.
  - Collaborating to acquire more information or even sabotage the computation.

Definitions

- *t-private* – Any set of at most *t* players cannot compute after the protocol more than they could jointly compute solely from their set of private inputs and outputs.

- *t-resilient* – no set of *t* or less players can influence the correctness of the outputs of the remaining players.

Today – Part 1: The Gossip case

- Theorem 1:
  - For every (probabilistic) function F and $t < n/2$ there exists a *t-private* protocol.

- Theorem 2:
  - There are functions for which there are no $n/2$ *private* protocols.

Today – Part 2: The Byzantine case

- Theorem 3:
  - For every probabilistic function and every $t<n/3$ there exist a protocol that is both *t-resilient* and *t-private*.

- Theorem 4:
  - There are functions for which there are no *n/3 resilient* protocols.

Known results:

- $t<n/3$ – BGW, absolute correctness (later on).
- $t<n/2$ – An algorithm presented by Rabin-Ben Or. This algorithm has an a small (exponential) probability of error.
- Both assume the existence of a broadcast channel.

Part 1: The Gossip case

- Here we take t such that $t<n/2$.
- The parameter t is known in advance by all players.

Theorem 1:

- For every (probabilistic) function F and $t < n/2$ there exists a *t-private* protocol.

Proof:

- Let $P_0,\ldots,P_{n-1}$ be the set of players.
- Let $n>2t+1$.
- Let E be some finite field with $|E|>n$.
- Assume all inputs are from E and that we can compute F using the operations x,+ and constants from E.

The protocol stages:

1. The input stage
   - Every player will bring his input to the computation.
2. The computation stage.
   - The players will simulate the circuit computing *F*, gate by gate.
3. The final stage.
   - The players reveal their secret shares of the final value.

The input stage:

- Each player has an input s.
- Divide the input into n parts. Each player will receive one part.
- Requirement: Any group of players of size$<t$ cannot reconstruct s using their parts.

Shamir's Secret Sharing Scheme

- The idea:
  - 2 points are sufficient to define a line.
  - 3 points are sufficient to define a parabola.
  - 4 points are sufficient to define a cubic curve.
  - ....
  - k points to define a polynomial of degree k-1.

The Input Stage

- Let $\alpha_0,\ldots\alpha_{n-1}$ be some n distinct non zero points in the field E.
- Each player holding an input $s \in E$, will select t random elements $a_i \in E$, for $i=1,\ldots,t$ setting

$$f(x) = s + a_1 x + \ldots + a_t x^t$$

- *Each player* P_i *will receive his share - the value*

$$s_i = f(\alpha_i)$$

Using Shamir's Secret Sharing Scheme:

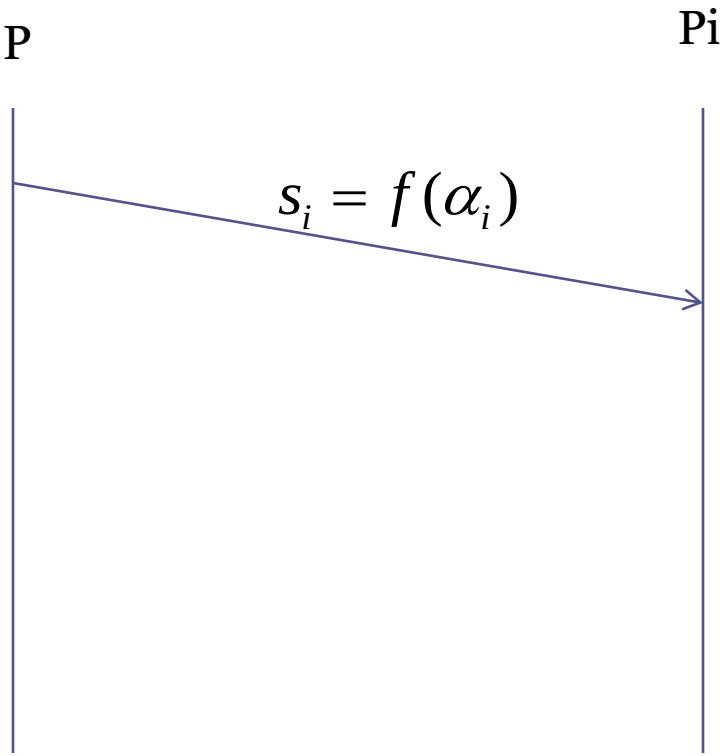
- The sequence $(s_0,\ldots,s_{n-1})$ is a sequence of *t-wise* independent random variables uniformly distributed over *E*.

- The value of the input *s* is independent from any group of the shares $\{s_i\}$ of size $\leq t$.

The Input Stage

Secret s:

$$f(x) = s + a_1 x + \ldots + a_t x^t$$



The computation stage

- We would like that each player will compute the function F gate by gate using the input shares he received in the input stage.

- Every gate will be evaluated as new share of the gate output value.

The computation stage

- Let a,b be two secrets shared by the polynomials $f(x)$, $g(x)$.
- Let $c \in E, c \neq 0$ be some constant.
- We need to show how to compute:
  1. $c \cdot a$
  2. $a+b$
  3. $a \cdot b$

The computation stage

- The linear operations (1+2) are simple:
  - $h(x)=c \cdot f(x)$
    - There exist: $h(\alpha_i) = c \cdot f(\alpha_i)$, $h(0) = c \cdot a$
    - $h(x)$ encodes $c \cdot f(x)$.
  - $k(x)=f(x)+g(x)$
    - There exist: $k(\alpha_i) = f(\alpha_i) + g(\alpha_i)$, $k(0) = a + b$
    - $k(x)$ encodes $f(x)+g(x)$.

The multiplication step

- A player holding $f(\alpha_i)$ and $g(\alpha_i)$ wants to compute a · b.
- We note that the free coefficient of the polynomial $h(x) = f(x) \cdot g(x)$ is a · b.

- There are two problems with *h(x):*
  - The degree of *h(x)* is *2t*.
  - *h(x)* is not a random polynomial.

The multiplication step – GRR (Gennaro, Rabin, Rabin)

- The product of $f(x)$ and $g(x)$ is

$$f(x)g(x) = \gamma_{2t}x^{2t} + \ldots + \gamma_1 x + ab = f_{ab}(x)$$

- We can write:

$$A \begin{Bmatrix} ab \\ \gamma_1 \\ . \\ . \\ . \\ \gamma_{2t} \end{Bmatrix} = \begin{Bmatrix} f_{ab}(\alpha_1) \\ f_{ab}(\alpha_2) \\ . \\ . \\ . \\ f_{ab}(\alpha_{2t+1}) \end{Bmatrix}$$

- A is (2t+1)x(2t+1) VDM matrix, $a_{i,j} = \alpha_i^{\,j-1}$.

The multiplication step – GRR

- The matrix A is VDM, it is not singular and has an inverse.
- Let the first row in the inverse matrix be:

$$(\lambda_1,\ldots,\lambda_{2t+1})$$

- The previous equation implies that

$$a \cdot b = \lambda_1 f_{ab}(\alpha_1) + \ldots + \lambda_{2t+1} f_{ab}(\alpha_{2t+1})$$

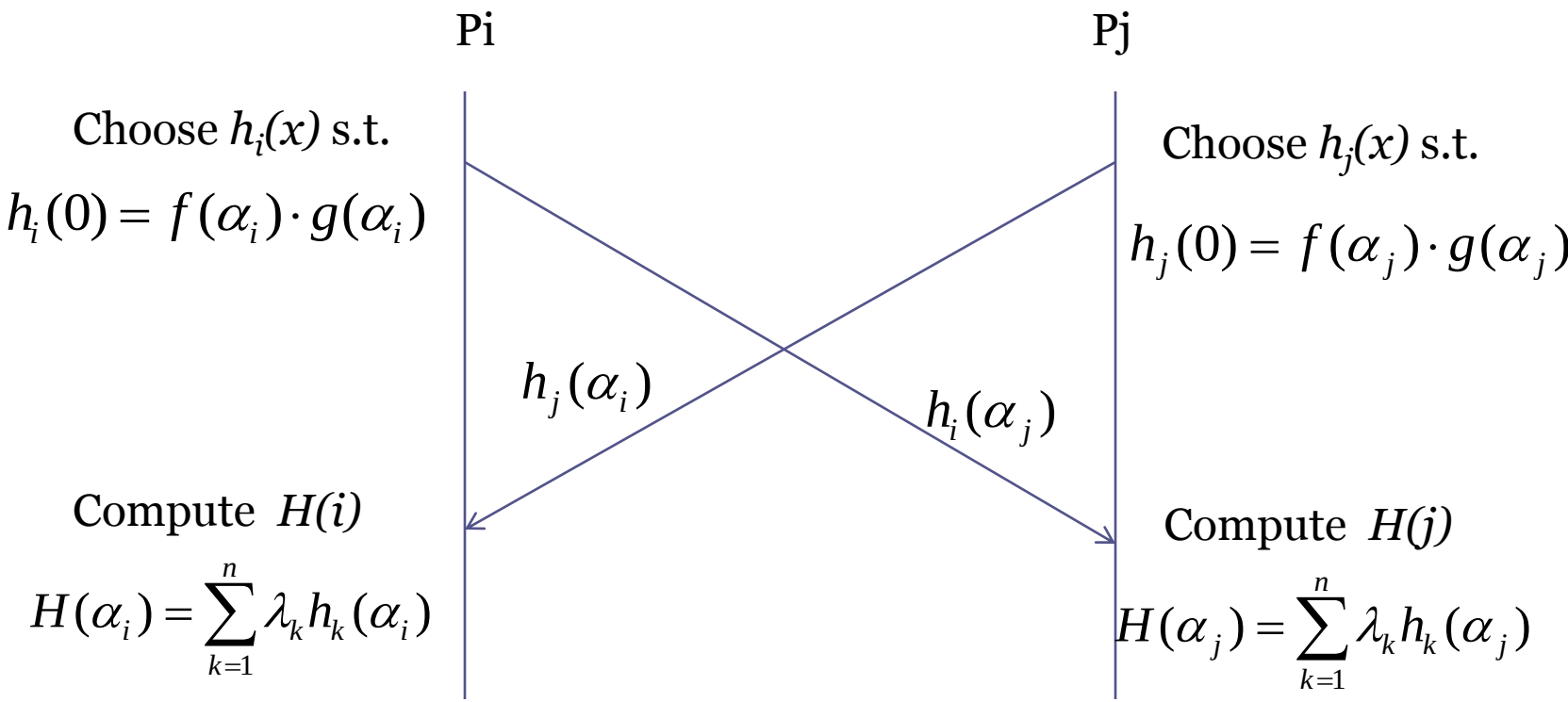
The multiplication step – GRR

- Every player P_i chooses a random polynomial $h_i(x)$ of degree t such that $h_i(0) = f_{ab}(\alpha_i)$.
- We define: $H(x) = \sum_{i=1}^{2t+1} \lambda_i h_i(x)$
- Note that $H(0) = a \cdot b$.

The multiplication step – GRR

- Furthermore $H(\alpha_j) = \sum_{i=1}^{2t+1} \lambda_i h_i(\alpha_j)$.
- Each player sends his share $h_i(x)$ to all players. Now each player can compute his share in the polynomial $H(x)$ used for sharing $a \cdot b$.
- The polynomial $H(x)$ is of degree t and it is random because n-t polynomials $h_i(x)$ were chosen by good players randomly.

The multiplication step



The final step

- In the end of the computation, we will have the value of F shared among the players.
- If we want only one player to know the output, all the shares will be sent to him. This player can compute the interpolation polynomial $f(x)$ and use $f(0)$ as the result.
- Note: The set of all shares doesn't contain any information about the input that doesn't follow from the value of $f(0)$.

Theorem 1 – Proof:

- Correctness:
  - Follows immediately from the correctness of each step in the computation stage.

- Secrecy:
  - Recall: '*A protocol is secure for some task if it "emulates" an "ideal process" where the parties hand their inputs to a "trusted party", who locally computes the desired outputs and hands them back to the parties*'.

Reminder: Definition of security

- A protocol π *emulates the ideal process for evaluating F if for any (PPT) adversary A there exists a (PPT) adversary S such that for any set of external inputs:*

[The inputs and outputs of all parties and A in π] ~
[The inputs and outputs of all parties and S in ideal process for F]

Secrecy

- The protocol functionality:
  1. Retrieve x_i from P_i, $1 \le i \le n$.
  2. Set $(y_1,..,y_n)=F(x_1,...,x_n)$
  3. Output y_i to P_i, for $1 \le i \le n$.

Secrecy - proof sketch

- Any P_i which is a good player will act the same in both protocol execution and the ideal process.

- Any P_i which is a bad player cannot tell the difference between *n-1* shares and *n-1* random values. The output of P_i is the same in both runs.

Secrecy - proof sketch

- For any sequence of inputs $x_1,\ldots x_n$ for the parties, the following two random variables are identically distributed:
  - The adversary view of an execution of the protocol with inputs $x_1,\ldots x_n$.
  - The same view, where each field element except for the inputs and outputs of the corrupted parties is replaced with an independently chosen new random element in the field.
- Corollary: The environment cannot tell between the runs.

Theorem 2:

- There are functions for which there are no n/2 private protocols.

Proof:

- Two players each holding one input bit cannot compute the function OR of their bits without leaking information.

Part 2: The Byzantine case

Theorem 3:

- For every probabilistic function and every $t<n/3$ there exist a protocol that is both *t-resilient* and *t-private*.

Using the previous protocol

- Using the previous protocol, we are facing 3 new problems:
  1. How to can we compute the secret now that some of the shares are false ones?
  2. How can we trust that the shares we get are valid ones?
  3. How can we trust the shares we get in the multiplication step (the $h_i(x)$)?

Sharing a secret with cheaters:

- Here we take $n=3t+1$.
- Requirements:
  A. Any set of at most t players does not have any information about the secret.
  B. It is easy to compute the secret from all its shares, even if up to t pieces are wrong or missing.

The scheme:

- For a secret *s*, choose a random polynomial *f(x)*

$$f(x) = s + a_1 x + \ldots + a_t x^t$$

- Each player Pi, i=0,.., n-1 will receive the piece

$$s_i = f(\alpha_i)$$

- As in [Sh] the s_i's are *t*-wise independent.

Requirement A is met!

Error Correcting code:

- Reed Solomon decoding
  - Given n distinct elements $x_1,\ldots,x_n \in E$ and $y_1,\ldots,y_n \in E$, parameter k, and $t \le (n-k)/2$.
  - Find a polynomial p of degree less than k such that

$$p(x_i) \neq y_i$$

    for at most t values.

- In our case $k=t$ and the requirement that $n \ge 3t$ is met.

Known solutions:

1. Peterson algorithm (1960) – The binary case.
2. Gorenstein and Zierler – generalized the last to the non-binary case. Runs in $O(n^3)$.
3. Berlekamp and Massey – Speeded the algorithm to run in $O(n^2)$.
4. Currently the fastest runs in $O(n \log n)$.

We will see a simple version of Welch and Berlekamp that runs in $O(n^3)$.

Error locating polynomial

- Definition – Given a vector x and received a vector y that is within a distance of t of some polynomial p of degree less than k. ***E(x)*** is called error locating polynomial if it has degree t and satisfies $E(x_i) = 0$ if $p(x_i) \neq y_i$.

- Note: This polynomial always exist.

More definitions

- We define another "hard-to-find" polynomial

$$N(x) = p(x) \cdot E(x)$$

- This equation holds the following property:

$$\forall i \quad N(x_i) = p(x_i) \cdot E(x_i) = y_i E(x_i)$$

  - If $E(x_i) = 0$ then $p(x_i) \cdot E(x_i) = y_i E(x_i) = 0$.
  - When $E(x_i) \neq 0$, $p(x_i) = y_i$ and we have $p(x_i) \cdot E(x_i) = y_i E(x_i)$

The Welch-Berlekamp algorithm

- Step 1: Find polynomials N and E satisfying the following, if they exist:

$$(*) \quad \begin{array}{l} \deg_x(E) = t \\ \deg_x(N) < k + t \\ \forall i \in [n] \quad N(x_i) = y_i E(x_i) \end{array}$$

- Step 2: Output $N(x)/E(x)$.

The Welch-Berlekamp algorithm – step 1

- We need to find coefficients $< N_0,\ldots,N_{k+t-1} >$ and $< E_0,\ldots,E_t >$ such that for all i

$$\sum_{j=0}^{k+t-1} N_j x_i^j = y_i \sum_{j=0}^{t} E_j x_i^j$$

- Solving then system above is just solving a linear system, except that we need $E_t \neq 0$. We can overcome this by setting $E_t = 1$ and divide all coefficients by E_t.

Correctness:

- Claim: There exist polynomials E and N satisfying the properties (*) such that $N(x)/E(x)=p(x)$.

- Proof: We take E to be the error-locating polynomial for p and the given set of points, we define N to be $N(x)=p(x)E(x)$.

Correctness:

- Claim: Any solutions *(N,E)* and *(N',E')* satisfy

$$\frac{N(x)}{E(x)} = \frac{N'(x)}{E'(x)}$$

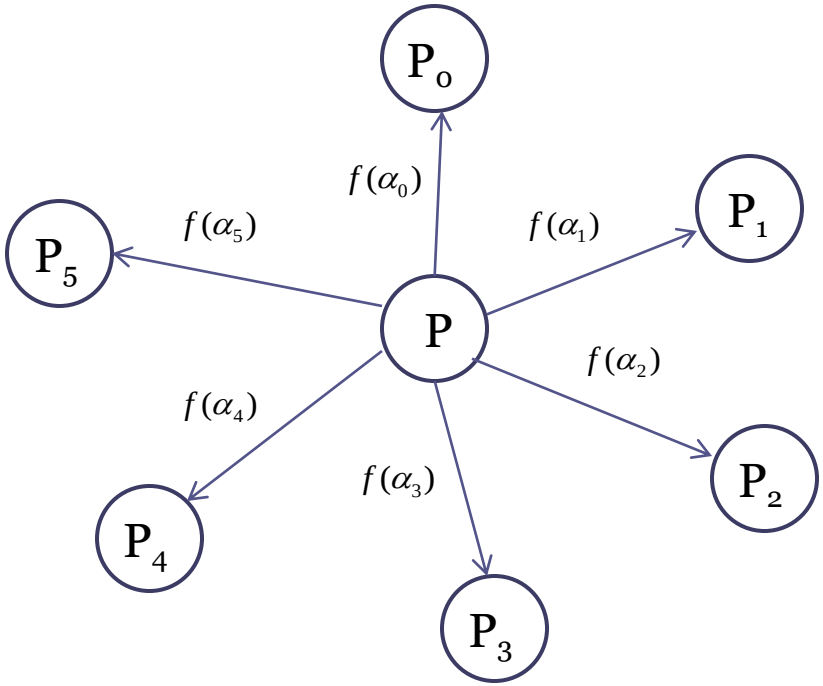
- Proof: The degree of the polynomials *N(x)E'(x)* and *N'(x)E(x)* is at most *k-1+2t*. We have $y_i E(x_i) = N(x_i)$ and $N'(x_i) = y_i E'(x_i)$. Multiplying the two we get $y_i E(x_i) N'(x_i) = y_i E'(x_i) N(x_i)$, and we get $E(x_i) N'(x_i) = E'(x_i) N(x_i)$.

The Byzantine case: Using Error Correcting code

- We want to compute the secret s from all its shares even if up to *t* pieces are wrong or missing.

- Using the Welch-Berlekamp algorithm with our shares we can compute the polynomial f and compute $s=f(0)$.

Verifying a secret



We need to verify that the secret shares are shares of a real secret. Verify the s_i are all on the same polynomial.

Verifying a secret – Absolute Verification

- Instead of just sending the shares $\{s_i\}$, the dealer *P* selects a random polynomial *f(x,y)* of degree *t* in both *x* and *y*.
- The only restriction of *f(x,y)* is that $f(0,0) = s$.
- *P* sends the polynomials $f_i(x) = f(x, \alpha_i)$, and the polynomials $g_i(y) = f(\alpha_i, y)$.
- The secret shares are $s_i = f_i(0)$.

Verifying the secret

- After receiving the polynomials $f_i(x)$ and $g_i(y)$, the player P_i sends to each P_j

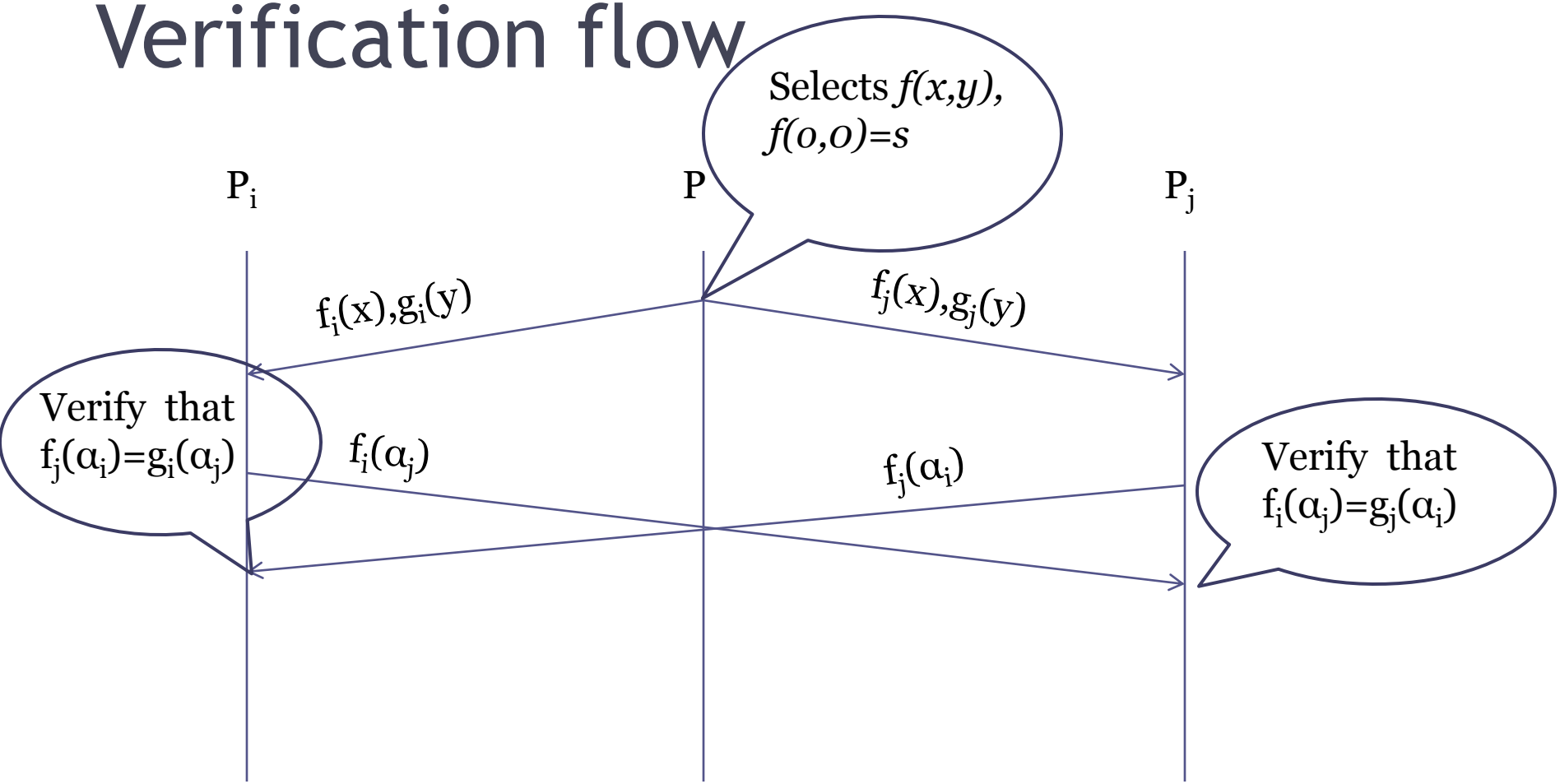
$$s_{i,j} = f_i(\alpha_j) = f(\alpha_j, \alpha_i) = g_j(\alpha_i)$$

- If the dealer P is correct, then all the points in the sequence $(s_{0,j}, s_{1,j}, \ldots, s_{n-1,j})$, are points on the polynomial $g_i(y)$.
- P_j compares the incoming values with his own.

Verifying the shares

	f_o	f_1	....	f_i	f_{n-1}
g_o				$f_i(\alpha_o)$	
g_1				$f_i(\alpha_1)$	
....				...	
g_j	$g_j(\alpha_o)$	$g_j(\alpha_1)$	...	$f(\alpha_j, \alpha_i)$	$g_j(\alpha_n)$
g_{n-1}				$f_i(\alpha_n)$	

Verification flow



Verifying the secret

- Each player P_j broadcasts the coordinates *(i,j)* he had to correct.
- If P_j detects more than *t* wrong incoming values, or had to correct his own value, the dealer is faulty.
- In such case, P_j requests *P* to make $f_j(x), g_j(y)$ public.

Verifying the secret

- Making $f_j(x), g_j(y)$ public, makes $s_{k,j}$ and $s_{j,k}$ public for $0 \leq k < n$.
- If a player P_i now observes that the new information contradicts his own, he requests his information to be made public, and so on.
- If more than t players ask to make their information public, the dealer is faulty. In that case the players pick a default value (zero).

Verifying the secret

- If t or less players have complaint, then there is at least $t+1$ good players who are satisfied. That uniquely defines the polynomial $f(x,y)$.

- The complaining players use the information that was made public.

Hidden assumption

- In order for a good player to complain on invalid data such that all the other players (not just the dealer) will hear him, we need a broadcast channel.

- Broadcast channel – The ability to send a message and to be sure that everybody received it.
- This model assume the existents of a broadcast channel.

Private Information

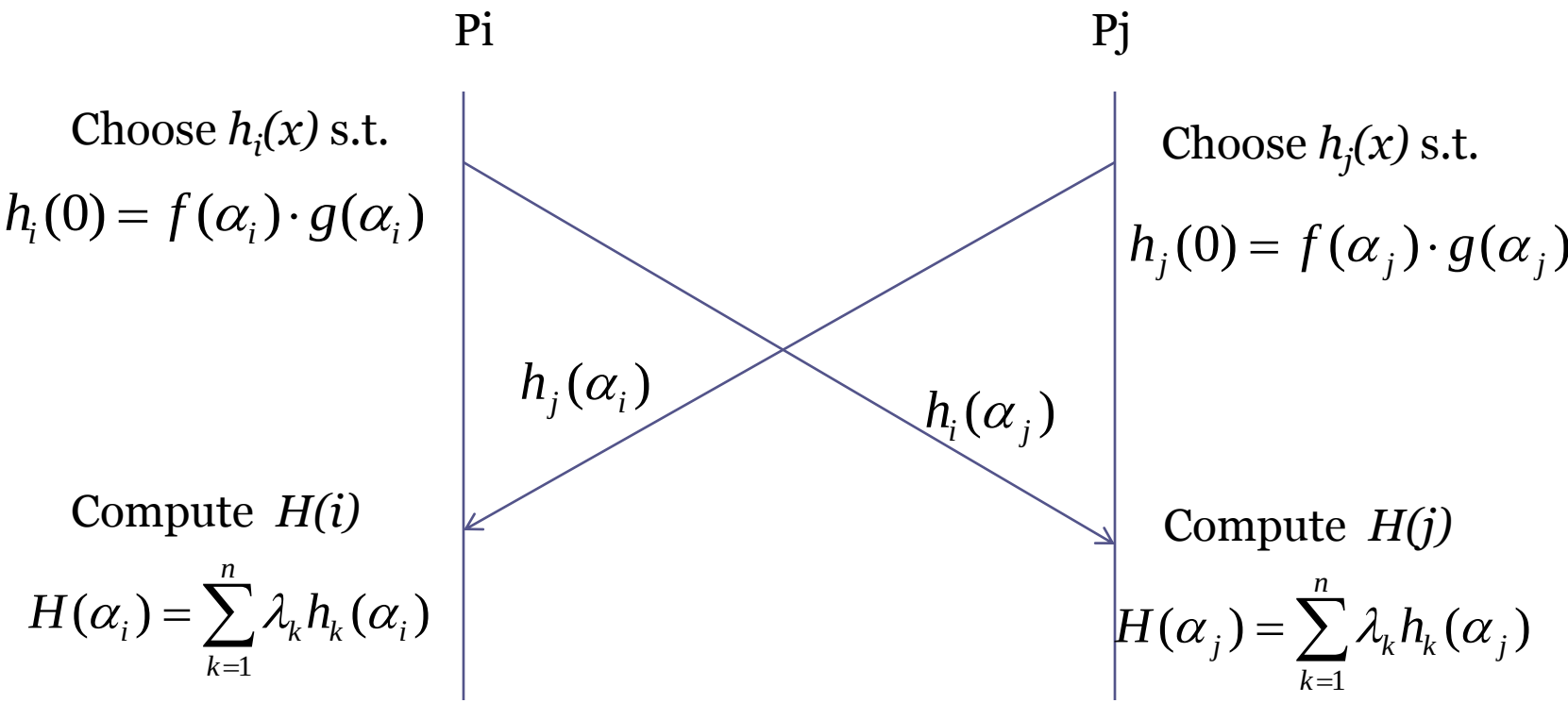
- If the dealer is a good player, then none of the good players revealed any piece of information.
- If the dealer is faulty, we don't need to protect his private information.

The computation stage – Byzantine case

- There is no change when computing $c \cdot a$ and $a+b$ because it is done locally (no communication involved).

- The problem is with the multiplication step.

Recall: The multiplication step



The multiplication step – Byzantine case

- Each player P_i has $c_i = h_i(0)$ and distributes it using VSS.
- Each player receives $c_{i,j}$ which is the share i of c_j
- The player P_j which receives the shares $(c_{j,0}, c_{j,1}, \ldots, c_{j,n})$ needs to verify that the shares are real shares of the real c_i's.

Global View: The shares each player receives

Shares of a code word

Po	…	Pi	…	Pn
Co,o	…	Ci,o	…	Cn,o
…	…	…	…	…
Co,n	…	Ci,n	…	Cn,n

The multiplication step – Byzantine case

- The c_i's are n points on a polynomial of degree $2t$.
- The players need to verify the shares by finding the syndrome of the codeword.
- By finding the syndrome each player can locally compute the error location vector.
- This can be done using a variant of Berlekamp and Welch. For that we assume $n>4t+1$.

Finding the syndrome:

- Let P be a polynomial with coefficients

$$\overline{P} = (P_0, \ldots, P_{2t})$$

- Let p be a codeword.

- Using the van-der monde matrix we can compute the codeword by $p = \overline{P} \cdot V$.
- Let e be the error vector, and q be the correct codeword, that is $p+e=q$.

Finding the syndrome:

- We can now compute the syndrome by

$$q \cdot V^{-1} = \overline{Q} = (Q_0, \ldots, Q_{2t+1}, \ldots, Q_n)$$

- Note that the last 2t+1 coefficients are affected only by the error vector.

- In our case the players compute only $Q_{2t+1}, \ldots, Q_n$, that is in order to keep the secrecy of the polynomial.

Finding the error vector:

- Let $\overline{Q}' = (0, \ldots 0, Q_{2t+1}, \ldots, Q_n)$.
- Let $q' = Q' \cdot V$.
- Using BW we can get the codeword p'.
- We then get q'-p'=e'.
- It is easy to check that e'=e.

The multiplication step – Byzantine case

- Note: The computation was done using only linear operations. Each player can perform these computation using it's shares.

- The players can decide to ignore the players who sent invalid shares, or to correct them using the error vector.

Theorem 4:

- There are functions for which there is no *n/3-resilient* protocol

- Proof:

  Even if we allow broadcast as a primitive, the theorem above is true. There are function for 3 players that cannot be computed resiliently when one player is bad.

Thank you!