Progetto #4

Secure Distributed Storage

Un problema combinatoriale [Liu, 68]

- Undici scienziati stanno lavorando ad un progetto segreto
- Vorrebbero rinchiudere i documenti relativi al progetto in una cassaforte
- La cassaforte dovrebbe essere aperta solo se almeno sei degli scienziati sono presenti
- Quale il più piccolo numero di:
  - Lucchetti necessari per chiudere la cassaforte?
  - Chiavi dei lucchetti che ogni scienziato deve possedere?

Soluzione

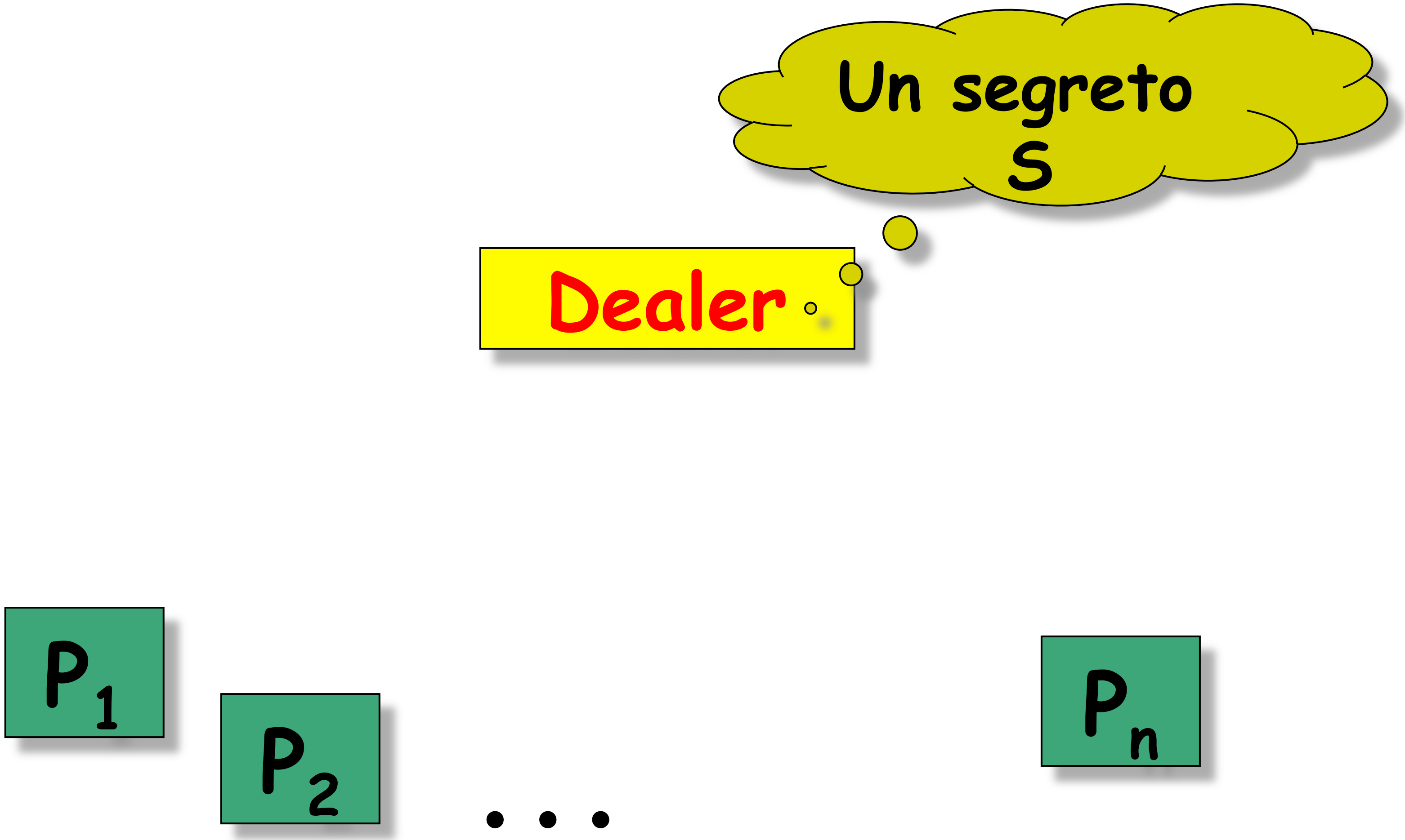
- Si può provare che la soluzione minimale usa 462 lucchetti e 252 chiavi per scienziato

- Tali numeri sono chiaramente non praticabili e diventano esponenzialmente peggiori quando il numero degli scienziati aumenta

Secret Sharing

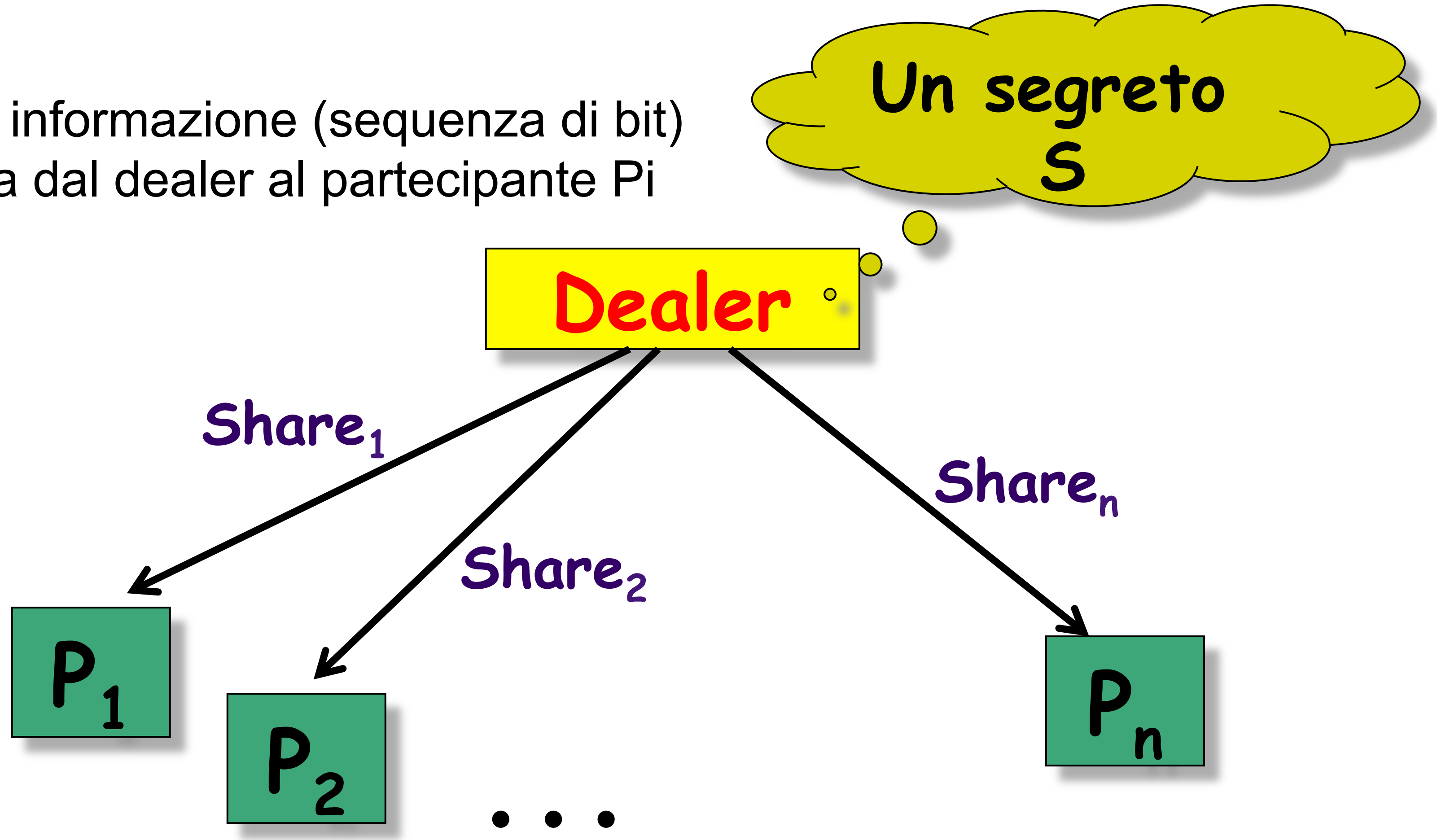
- Shamir, nel 1979 generalizzò il problema di Liu
- Un'entità fidata (*dealer*) vuole dividere un dato segreto S (la combinazione della cassaforte) tra *n* persone (*partecipanti*) in modo che
  - Un qualsiasi insieme di k o più partecipanti può facilmente calcolare S; ma,
  - La collusione di meno di k partecipanti lascia il segreto S completamente indeterminato
    - Tutti i suoi possibili valori sono equiprobabili
- Tale schema è chiamato *schema a soglia (k, n)*

Lo scenario

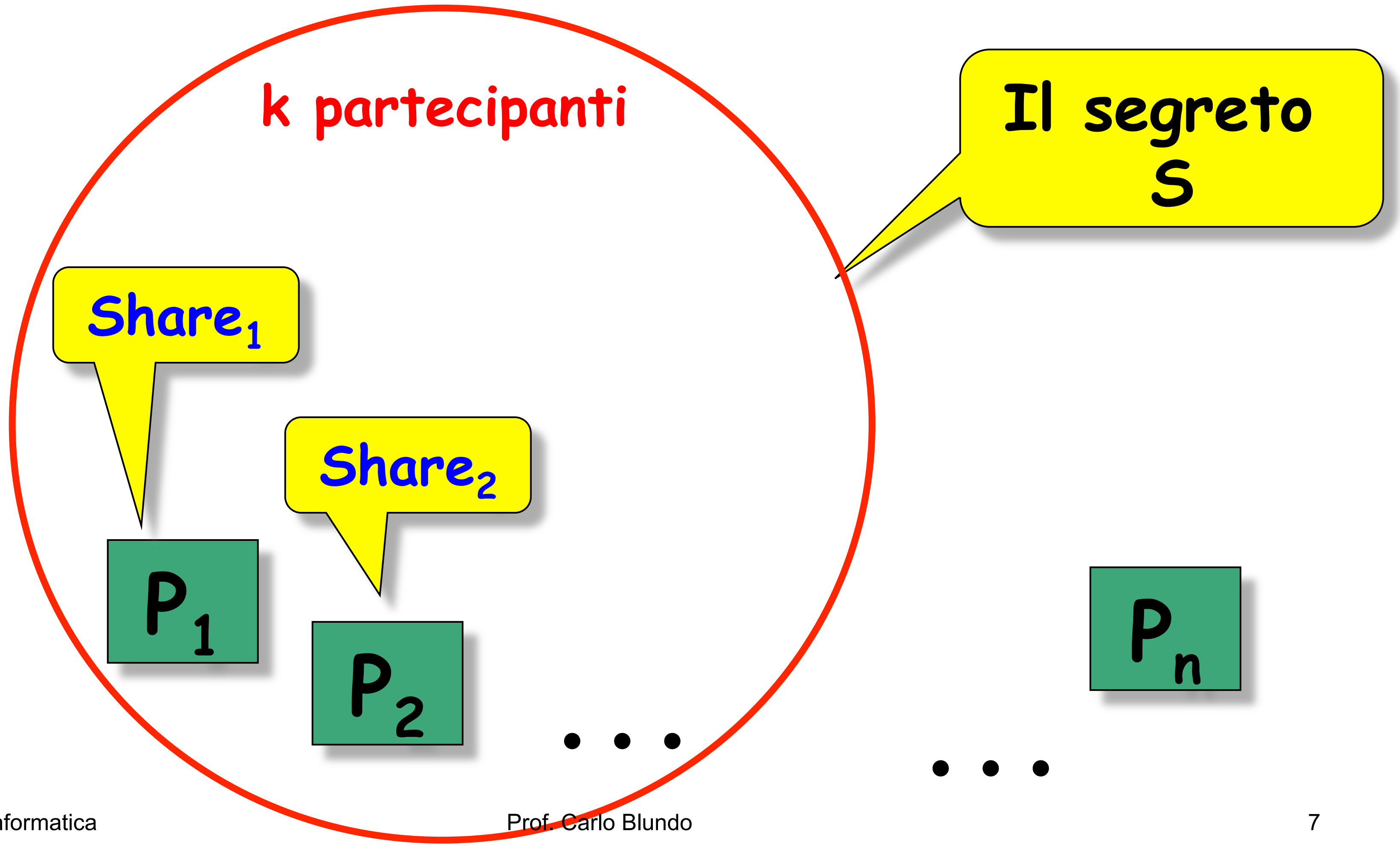


Distribuzione delle share

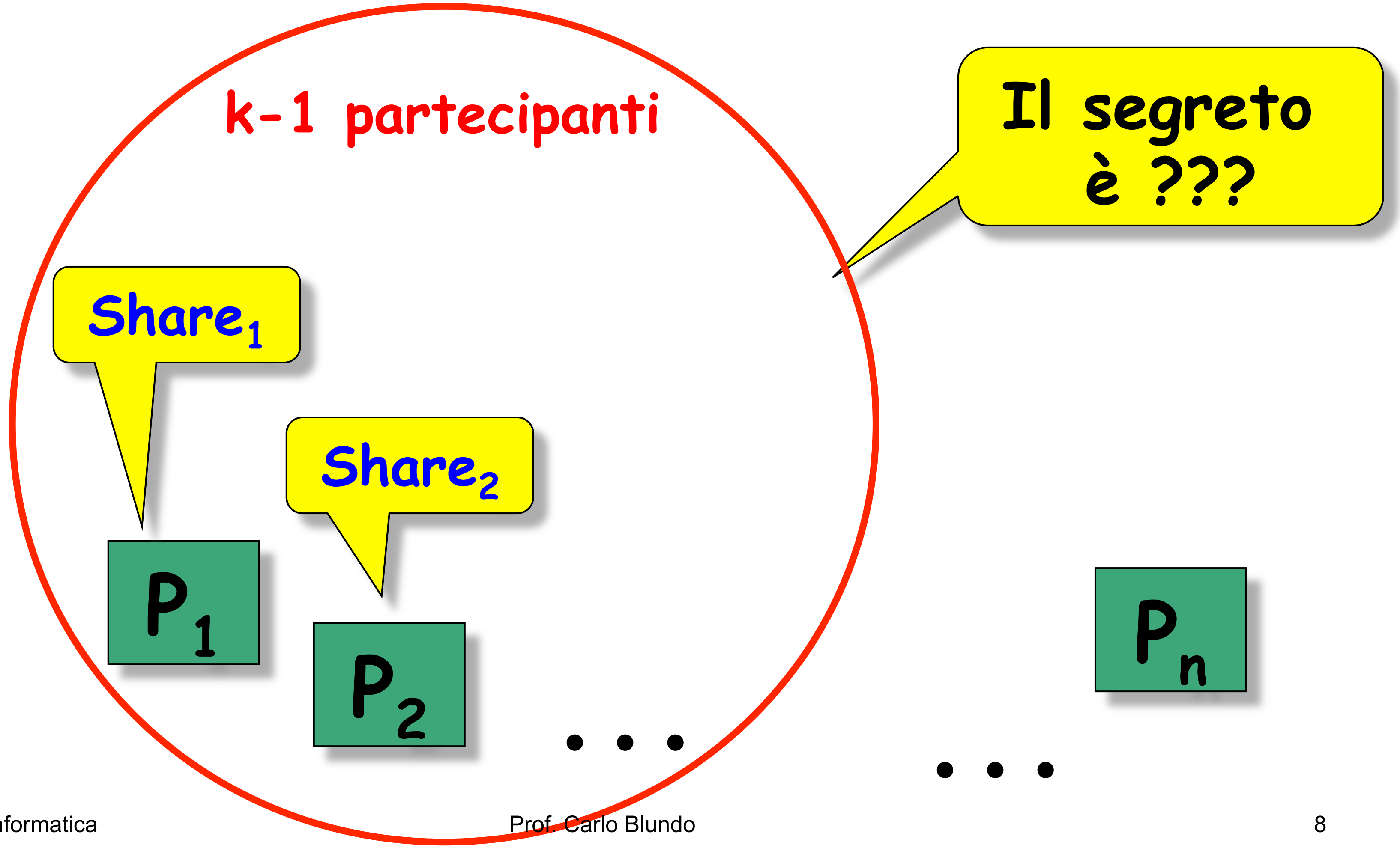
Share$_i$ = informazione (sequenza di bit) distribuita dal dealer al partecipante Pi



Ricostruzione del segreto



Sicurezza dello schema



Un semplice esempio

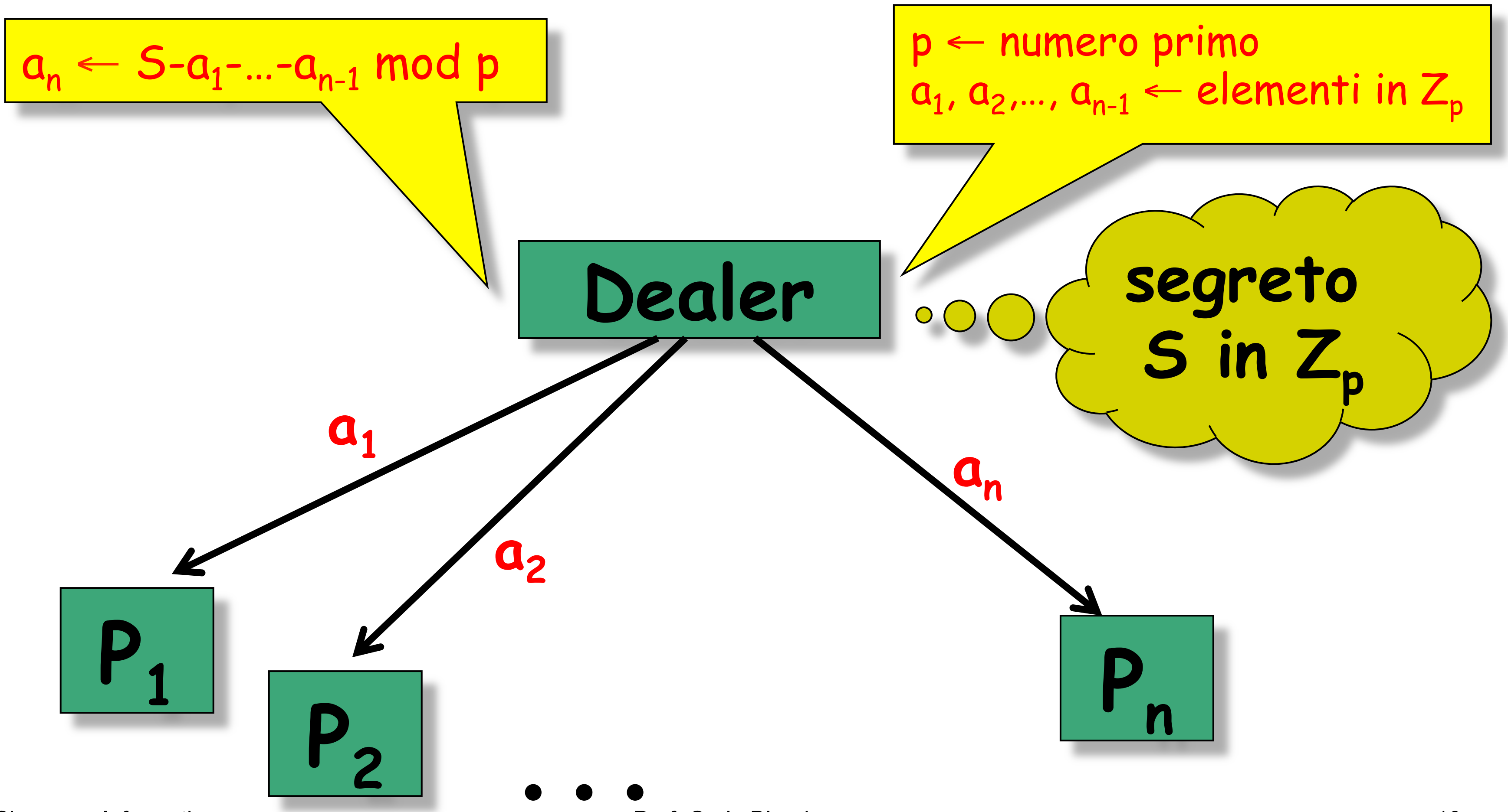
- Consideriamo un gruppo di n partecipanti
- Ad ogni partecipante è data una share s_i, che è una stringa casuale di bit di lunghezza fissata ($s_i \in \{0,1\}^t$)
- Il segreto s è la stringa di t bit

$$s = s_1 \oplus s_2 \oplus \cdots \oplus s_n.$$

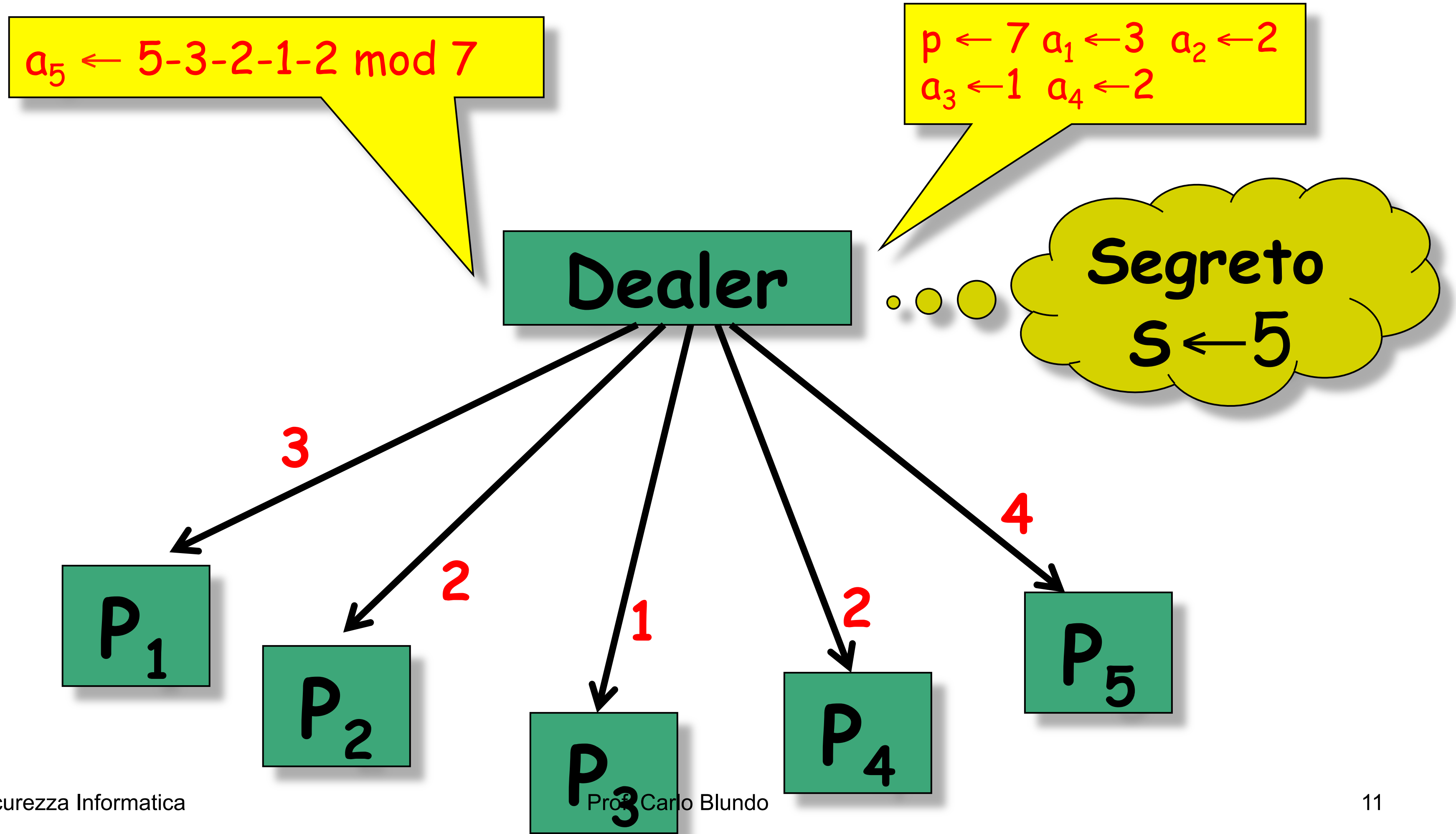
- Tutte le share sono necessarie per recuperare (ricostruire) il segreto
- Un qualsiasi insieme di n-1 partecipanti non ha alcuna informazione sul segreto s

Uno schema a soglia (n,n)

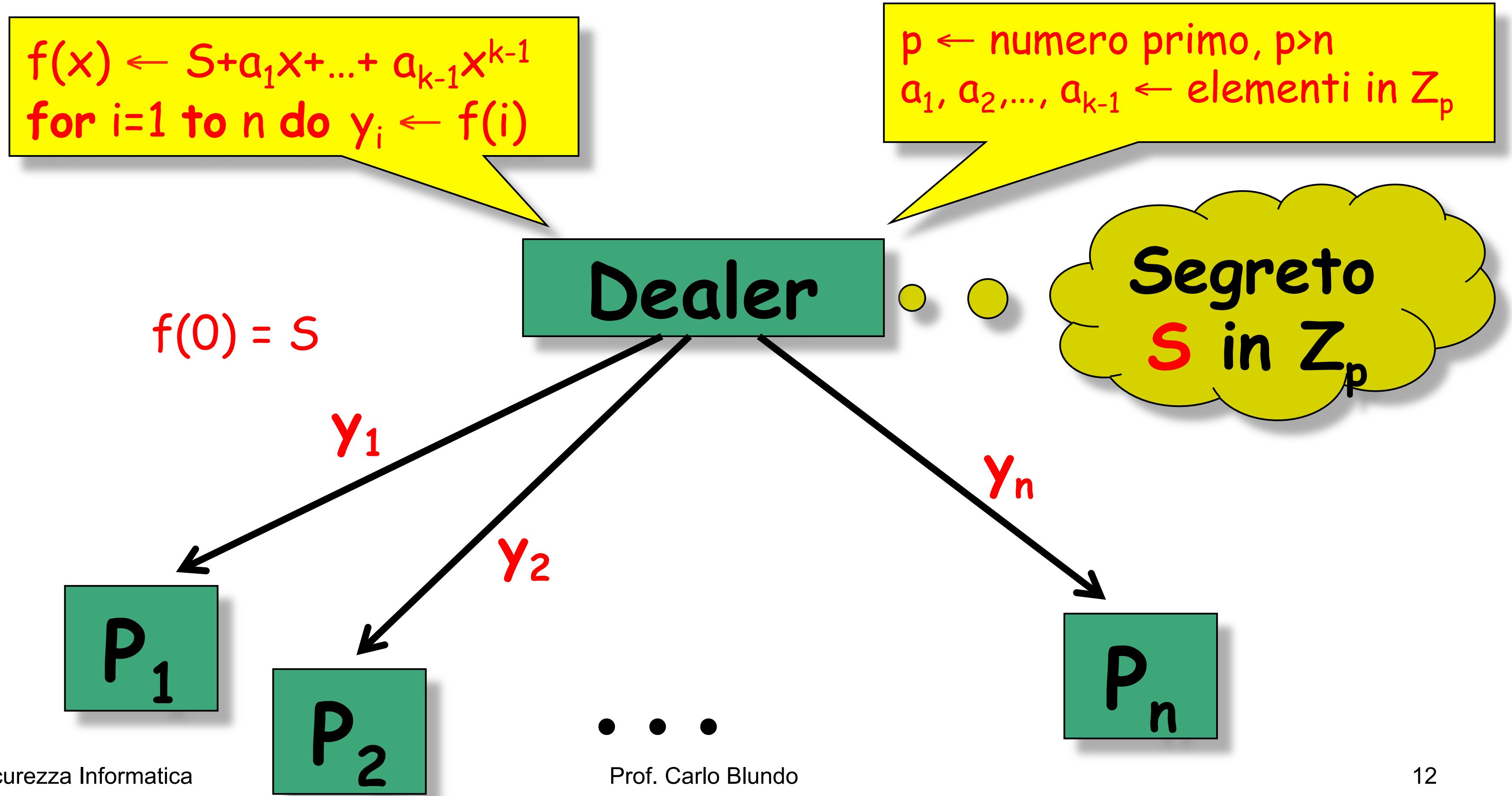
(n,n)-threshold scheme



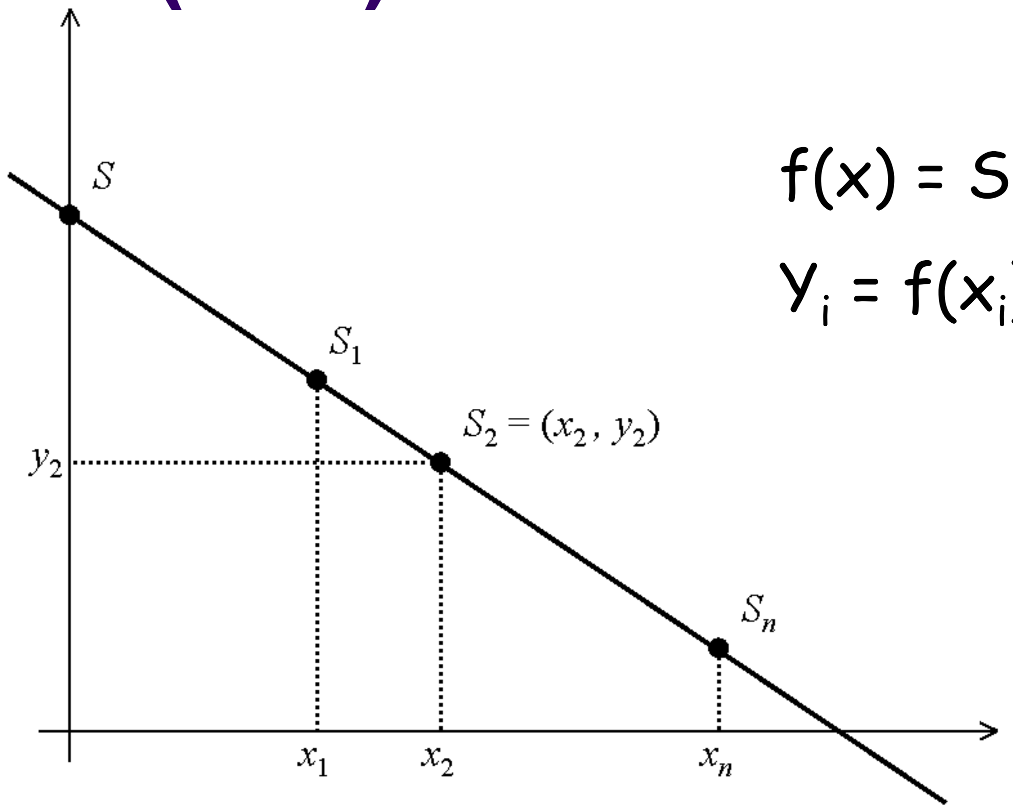
Schema soglia (5,5)



Schema a soglia (k,n) di Shamir



Esempio (k=2)



f(x) = S + ax

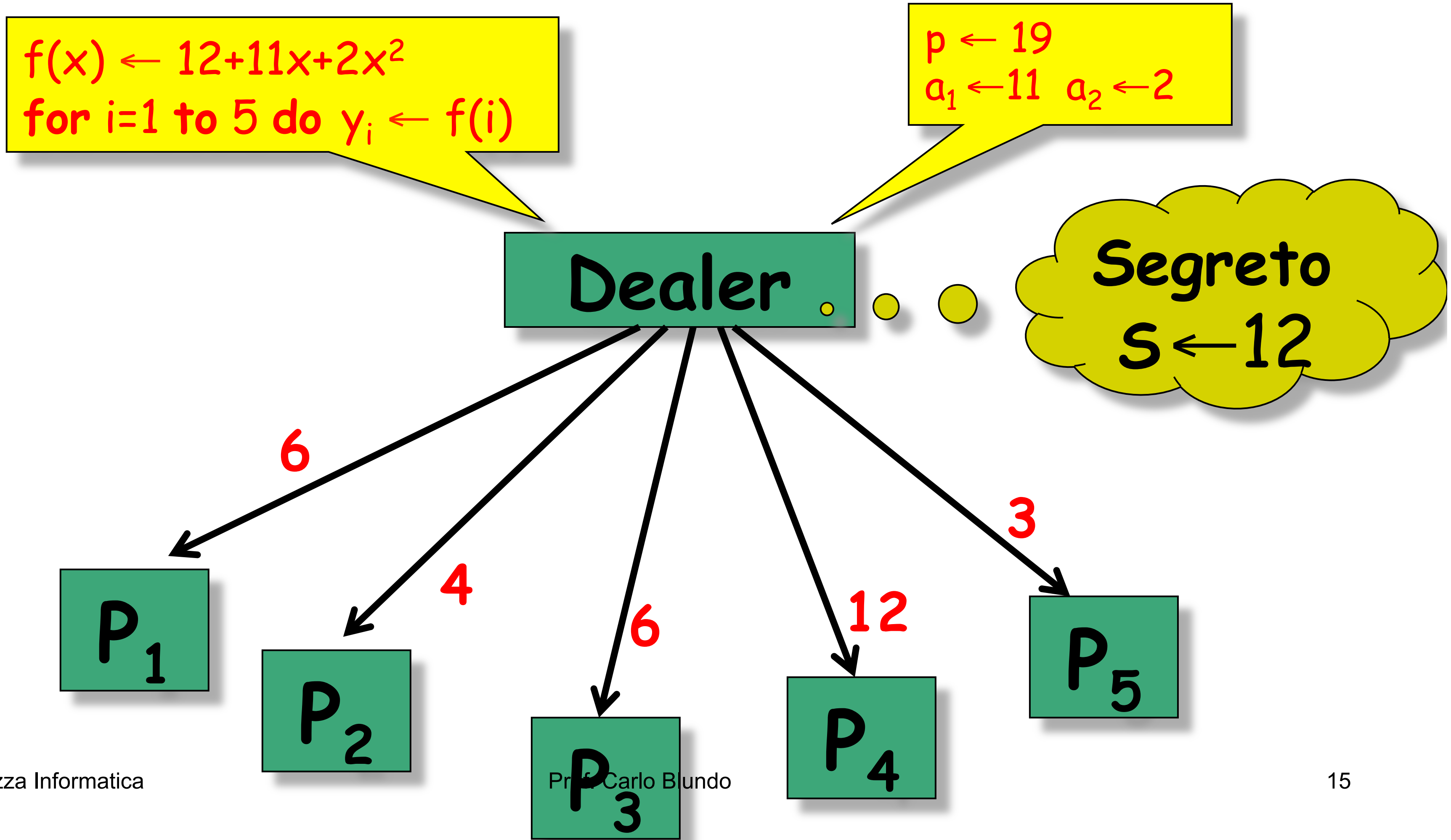
$y_i = f(x_i)$

- Con una singola share (punto), il segreto può essere un qualsiasi punto in [0,p) con uguale probabilità

Intuizione del perché funziona

- Affinché si possa interpolare (calcolare il polinomio f(x)), i coefficienti del polinomio sono scelti uniformemente in Z_p con p primo maggiore di n
- Dato un polinomio f di grado k-1
  - k punti sono sufficienti per calcolare il polinomio f
    - Interpolazione di Lagrange
  - Il segreto S è f(0)
  - Dati k-1 punti, qualsiasi segreto è equiprobabile

Schema a soglia (3,5)



Correttezza

- Ogni gruppo di k partecipanti può recuperare il segreto
- Essi possono interpolare f(x) e valutando il polinomio in 0 ottenere il segreto S (S = f(0))
  - Conoscono k punti del polinomio di grado k-1
- Tali partecipanti *conoscono* un sistema di
  - k equazioni: $y_i = f(i) = S + a_1 i + \ldots + a_{k-1} i^{k-1}$ per $i = i_1, i_2, \ldots i_k$
  - k incognite: $S, a_1, \ldots, a_{k-1}$
- Tali partecipanti non devono risolvere il sistema dato che possono usare la formula di interpolazione di Lagrange

Matrice di Vandermonde

- La matrice dei coefficienti del sistema di equazioni è una matrice di Vandermonde

$$\begin{cases} S+a_1i_1+a_2i_1^2\ldots+ a_{k-1}i_1^{k-1} = y_{i1} \\ S+a_1i_2+a_2i_2^2\ldots+ a_{k-1}i_2^{k-1} = y_{i2} \\ \\ S+a_1i_k+a_2i_k^2\ldots+ a_{k-1}i_k^{k-1} = y_{ik} \end{cases}$$

$$\begin{matrix} 1 & i_1 & i_1^2 & \cdots\cdots & i_1^{k-1} \\ 1 & i_2 & i_2^2 & \cdots\cdots & i_2^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & i_k & i_k^2 & \cdots\cdots & i_k^{k-1} \end{matrix}$$

$$\det A = \prod_{1\leq j<t\leq k} (i_j - i_t)$$

Ogni termine del prodotto è diverso da zero

Interpolazione di Lagrange

- f(x) polinomio di grado k-1 tale che
  - $f(0)=S$ e $f(i_j) = y_{i_j}$, per j=1,2,…,n
- Possiamo scrivere f(x) come

$$f(x) = \sum_{j=1}^{k} y_{i_j} \prod_{\substack{1\le t\le k \\ t\ne j}} \frac{x - i_t}{i_j - i_t}$$

- Abbiamo bisogno solo di f(0)=S, quindi

$$f(0) = \sum_{j=1}^{k} y_{i_j} \prod_{\substack{1\le t\le k \\ t\ne j}} \frac{i_t}{i_t - i_j}$$

Termine noto date le identità dei partecipanti

Altro esempio di (3,5)-TS

$$
\begin{aligned}
n &= 5\\
k &= 3\\
S &= 7\\
a_0 &= S\\
a_1 &= 3\\
a_2 &= 5\\
p &= 11
\end{aligned}
$$

$$q(x) = 5x^2 + 3x + 7 \pmod{11}$$

$$
\begin{aligned}
S_1 &= q(1) = 5(1)^2 + 3(1) + 7 \pmod{11} \equiv 4\\
S_2 &= q(2) = 5(2)^2 + 3(2) + 7 \pmod{11} \equiv 0\\
S_3 &= q(3) = 5(3)^2 + 3(3) + 7 \pmod{11} \equiv 6\\
S_4 &= q(4) = 5(4)^2 + 3(4) + 7 \pmod{11} \equiv 2\\
S_5 &= q(5) = 5(5)^2 + 3(5) + 7 \pmod{11} \equiv 4
\end{aligned}
$$

- Il partecipante i-esimo riceve la share S_i

... continuazione

- Supponiamo che i partecipanti con share $S_1=4$, $S_2=0$ e $S_5=5$ decidono di ricostruire il segreto, allora abbiamo

$$P(x) = [4\frac{(x-2)(x-5)}{(1-2)(1-5)} + 0\frac{(x-1)(x-5)}{(2-1)(2-5)} + 4\frac{(x-1)(x-2)}{(5-1)(5-2)}] \pmod{11}$$

$$P(x) = [(x-2)(x-5) + 4(x-1)(x-2)] \pmod{11} = 5x^2 + 3x + 7 \pmod{11}$$

$$S = P(0) = 7$$

Progetto#4: parte 1

- Sviluppare una classe Java **SecretSharing** che implementa lo schema di Shamir (k, n)
- Rappresentare gli elementi in Z_p con la classe **BigInteger**
  - p numero primo maggiore di n
  - Le identità dei partecipanti sono valori distinti in [1, p)
  - Il partecipante P_i avrà identità i
    - P3 = new BigInteger(“3");

Funzioni utili

- probablePrime(int bitLength, **Random** rnd)
  - Restituisce un BigInteger positivo di bitLength bit che è primo con grande probabilità
  - p=BigInteger.probablePrime(bitLength, *new Random());*

- Per generare numeri primi potete usare la funzione genPrime() specificata nella prossima slide

```java
private  BigInteger genPrime() {
        BigInteger p=null;
        boolean ok=false;

       do
        {
          p=BigInteger.probablePrime(this.modLength, new Random());
          if(p.isProbablePrime(this.CERTAINTY))
                    ok=true;
        }while(ok==false);

        return p;
  }
```

Nella classe dove è definita la funzione genPrime, aggiungere

```java
private final int CERTAINTY = 50;
    // I numeri primi sono generati con probabilità 1-2^(-CERTAINTY)
private int modLength;
```

Generare numeri casuali in Z_p

```
private BigInteger randomZp() {
   BigInteger r;
    do {
    r = new BigInteger(modLength, new Random());
          }
 while (r.compareTo(BigInteger.ZERO) < 0 || r.compareTo(this.p) >= 0);

    return r;
}
```

Per restituire un intero casuale in Z^*_p

```
while (r.compareTo(BigInteger.ZERO) <= 0 || r.compareTo(this.q) >= 0);
```

Progetto #4: parte 2

- Sviluppare un servizio di Secure Distributed Storage per conservare file usando servizi di archiviazione cloud tipo Dropbox, Onedrive, Google Drive, …
- Non è necessario sviluppare un'architettura client-server
  - Il client e i server saranno simulati da cartelle in un filesystem

Specifiche - server

- Un file del client è conservato su n server
- Per ricostruire il file è necessario il contributo di almeno k server
- Un qualsiasi insieme di t server (con t<k) condividendo le informazioni in loro possesso non sono in grado di ricostruire il file originale
- Il nome del file su ogni server deve essere casuale

Specifiche - client

- Il client deve essere in grado di verificare se il file (share) restituito dal server sia stato modificato
- Il client non deve conservare copia del file, ma solo il suo nome, quanti server sono necessari per la ricostruzione del file e di eventuali informazioni per verificare l'integrità del file

Test

- Sviluppare
  - una funzione per distribuire il file sui server
  - una funzione per recuperare il file dai server
- Testare il servizio sviluppato con
  - n=5 e k=2
  - n=5 e k=3

Altri Dettagli

- Non ce ne sono
- Decidete tutto voi
- Non chiedetemi niente
  - Chiarirò dubbi solo sul Secret Sharing, ma solo dopo che avete studiato il materiale messo a disposizione sulla piattaforma