Computer and Network Security

Lorenzo Rossi

January 4, 2022

Contents

Part I

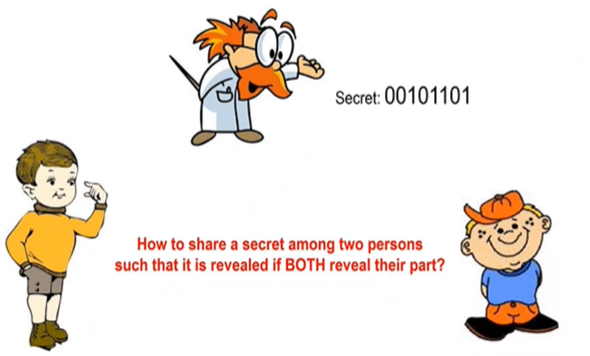
Third Midterm

Chapter 1

Secret Sharing

1.1 Trivial Secret Sharing

Supponiamo di avere un segreto e vogliamo dividerne la conoscenza in due persone (dette shareholders). Inoltre, vogliamo si viene a conoscenza del segreto se e solo se entrambe le parti rivelano la loro porzione di segreto. Chi



fornisce il segreto viene detto **dealer**, mentre chi riceve le porzioni del segreto sono detti **share**.
Nel caso in cui avessimo diviso il segreto in parti uguali, è una pessima idea poiché per indovinare il segreto abbiamo $\frac{1}{2^{N_{bit}}}$ probabilità di indovinare la password ed ora, avendo diviso il segreto in parti uguali, abbiamo una probabilità molto maggiore $\frac{1}{2^{\frac{N_{bit}}{2}}}$.

1.1.1 XOR Secret Sharing

Possiamo fare di meglio:

1. Prendi il segreto i.e.0010.1101;
2. Genera una sequenza casuale **key** i.e.1011.0100;
3. XOR il segreto e il valore casuale **one time pad** i.e.1001.1001;
   Fino ad ora abbiamo applicato un *Vernam cipher*.
4. Diamo ad uno share la sequenza casuale, mentre ad un altro diamo il valore dello XOR;
5. L'unione fra gli share da la chiave.

Importante. *Il conoscere la chiave, cioè il valore casuale, non mi da alcuna informazione riguardante la chiave.*
Lo stesso discorso vale per il valore dello XOR poiché, come dimostrato nel ***perfect secrecy****, l'operatore di XOR tra una stringa pseudocasuale e un valore casuale non da informazioni su quale sia la password.*
Questi due aspetti rappresentano un requisito di sicurezza.

1.1.2 Modular Secret Sharing

Un altro possibile schema è quello di utilizzare le somme modulari:

1. Prendi il segreto S in bit, trasformalo in digit i.e.0010.1101$rightarrow$45;
2. Genera $RAND \mod N$ i.e.$RAND \mod 256 \to 180$;
3. Esegui $S - RAND \mod N$ i.e. $S - RAND \mod 256 \to 121$;

Importante. *Questo schema è equivalente ad One Time Pad poiché abbiamo sommato un numero pseudocasuale con un numero casuare (in modulo). In altre parole, la probabilità di indovinare S conoscendo il valore casuale o il valore della somma è uguale alla probabilità di indovinare senza sapere nulla.*

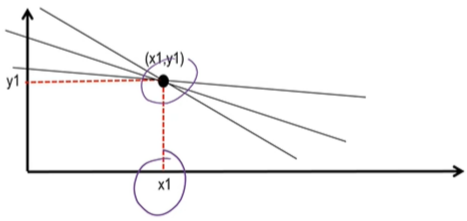
Questo metodo è più facile da implementare per essere condiviso con N shareholders. In particolare, genero 3 quantità truly random ed effettua la differenza tra il segreto e queste 3 quantità modulo N. Nel caso un attacker, riuscisse ad ottenere un numero sufficiente di share non può comunque ottenere la password, ma al più la differenza tra il segreto e le shares non prese.
Da qui è possibile definire il concetto di **perfect secrecy**: un avversario, conoscendo n-1 shares deve ancora possedere la probabilità di indovinare il segreto pari a quella di indovinare il segreto da zero.

1.2 Shamir Secret Sharing

Fino ad ora abbiamo costruito uno schema detto (n,n) secret sharing scheme in cui il primo parametro è il numero delle persone necessarie a rilevare il segreto e il secondo parametor è il numero di parti:il segreto viene rilevato solo se tutte le n parti forniscono il segreto.
Un altro schema è (t,n) secret sharing scheme:il segreto è rilevato quando qualsiasi t delle n parti fornisce il segreto. Questo secondo problema è molto più complicato del trivial secret sharing.

1.2.1 Idea:Schema (2,n)

Il problema è quello di modellare uno schema per cui, conoscendo 2 degli n shareholders, posso ricostruire il segreto. Questo problema è riconducibile a quello di conoscere quanti punti sono necessari per definire una linea:ovviamente 2. Infatti conoscendo un solo punto (shares) ho infinite rette passanti per quel punto e quindi è impossibile ricondurci



al segreto;tuttavia, conoscendo 2 punti (shares), tra essi passa solamente una sola retta e conseguentemente posso conoscere il segreto. Abbiamo comunque mantenuto la proprietà di poter avere un numero maggiore di 2 per ottenere il segreto, ma al minimo sono 2.

1.2.2 Procedura: Schema$(2, n)$

- **Dealer:** costruisce la linea:
  1. Coefficiente a:scelto casualemnte;
  2. Segreto S:noto;

$$y = S + ax$$

  Per esempio: $a = 15 \quad S = 39$

- Distribuisci le shares ai n partecipanti scegliendo casualmente il valore x_i da introdurre nell'equazione della retta:
  - Shareholder 1: $x_1 = 1 \to share = (1, 54)$;
  - Shareholder 2: $x_2 = 2 \to share = (2, 69)$;
  - Shareholder 3: $x_3 = 3 \to share = (3, 84)$;
  - ...

Importante. *La y viene calcolata in base alla funzione della retta;tuttavia, i punti degli shareholder sono mantenuti con (x,y) e il valore delle x_i possono essere noti a priori a patto che la y sia nascosta.*

1.2.3 Procedura: Ricostruzione (2,n)

- Ricezione di due shares:$P_i = (x_i, y_i) \quad P_j = (x_j, y_j)$;
- Interpola i punti per ricostruire l'equazione della retta:

$$\frac{y - y_i}{y_i - y_j} = \frac{x - x_i}{x_i - x_j}$$

  Ottenendo:

$$y = y_i + \frac{x - x_i}{x_i - x_j}(y_i - y_j)$$

- Bisogna sostituire $x = 0$ per ottenere il segreto $y = S$;

1.2.4 Estensione al caso (t, n)

Estendendo il discorso precedentemente introdotto, ci si riconduce al caso di polinomi di grado $t-1$ unicamente definiti da t punti:

- Linea:2 punti;
- Parabola (quadratic):3 punti;
- Cubiche:4 punti;
- ...

1.2.5 Generalizzazione: Schema (t, n)

- **Dealer**:
  1. Genera un polinomio casuale $p(x)$ di grado $t - 1$;
  2. Imposta il segreto s come il termine noto del polinomio:

$$p(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-2} x^{t-2} + a_{t-1} x^t - 1$$

     con s il segreto e i coefficienti delle x generati truly random;
  3. Distribuisci uno share ad ogni shareholders:

$$(x_i, y_i) \rightarrow y_i = p(x_i)$$

- **Ricostruzione**: Colleziona t shares su n disponibili e calcola il segreto utilizzando *l'Interporlazione di Lagrange* con $x = 0$:

$$s = \sum_{shares\ x_i} y_i \Lambda_{x_i} \quad with \quad \Lambda_{x_i} = \Lambda_{x_i}(0) = \prod_{shares\ x_k \neq x_j} \frac{-x_k}{x_i - x_k}$$

  L'interpolazione di Lagrange si basa sul concetto che qualsiasi polinomio di grando t-1 con t punti noti, può essere decomposto come:

$$y = \sum_{i=1}^{t} y_i \Lambda_i(x)$$

  In cui $\Lambda_i(x)$ è la base del polinomio calcolata come:

$$\Lambda_i(x) = \prod_{m=1, m \neq 1}^{l} \frac{x - x_m}{x_i - x_m} \quad \Lambda_i(x_i) = 1; \quad \Lambda_i(x_m) = 0 \quad for\ m \neq i$$

1.2.6 Segretezza

Per discutere di quanto sia sicuro questo schema dobbiamo ricordare che in questo ambito la segretezza è così definita:

Finché si conoscono (t-1) shares non si dovrebbe avere nessuna informazione sul segreto che stiamo condividendo.

Lo schema di Shamir in questo senso non è sicuro poiché se conoscessi a priori il range in cui è compreso il segreto, potrei ciclare su uno share mancante per ottenere un segreto nel range voluto.

***Esempio* 1.** *Effttuiamo uno schema (3,4) in cui per conoscere il segreto dobbiamo conoscore almeno 3 share su 4. Dato che utilizziamo l'interpolazione di Lagrange il polinomio sarà di grado $t-1$ e il termine noto sarà s:*

$$y = 3x^2 + 52x + 32;$$

Abbiamo 4 shareholders, quindi dobbiamo generare 4 punti,generando un valore casuale x e sostituendolo nell'equazione precedente. Avendo posto rispettivamte i valori $1, 2, 3, 4$, si ottengono i seguenit punti:

$$(1, 87), (2, 148), (3, 215), (4, 288)$$

Ora, occorre calcolare i valori di lambda, supponendo di aver collezionato x_1, x_2, x_3, come

$$\Lambda_i(x) = \prod_{m=1, m\neq 1}^{l} \frac{x - x_m}{x_i - x_m} \quad \Lambda_i(x_i) = 1; \quad \Lambda_i(x_m) = 0 \quad for\ m \neq i$$

Ottenendo:

$$\Lambda_1(x) = \frac{(x - x2)(x - x3)}{(x1 - x2)(x1 - x3)}$$

$$\Lambda_2(x) = \frac{(x - x1)(x - x3)}{(x2 - x1)(x2 - x3)}$$

$$\Lambda_3(x) = \frac{(x - x1)(x - x2)}{(x3 - x1)(x3 - x2)}$$

Ora, per ricostruire il segreto occorre applicare

$$y = \sum_{i=1}^{t} y_i \Lambda_i(x)$$

Quindi:

$$s = y_1\Lambda_{x_1}(0) + y_2\Lambda_{x_2}(0) + y_3\Lambda_{x_3}(0) = 87(30) + 148(-3) + 215(1) = 32$$

Ora supponiamo di non sapere uno share (d) e vogliamo verificare se questo schema garantisce secrecy o meno. Sostituendo imponiamo:

$$s = y_1\Lambda_{x_1}(0) + d\Lambda_{x_2}(0) + y_3\Lambda_{x_3}(0) = 476 - 3d$$

Ipotizziamo che il range in cui vive s è noto e compreso tra 0 e 100. Possiamo indovinare il segreto? Si, basta ciclare sulle d:

- *Con* $d = 125 \rightarrow s = 101$*;*
- *Con* $d = 126 \rightarrow s = 98$*;*
- *Con* $d = 127 \rightarrow s = 95$*;*
- *Da varie prove si capisce che d è nel range* $126 \leq d \leq 158$*;*

Quindi, conoscere 2 su 3 in uno schema 3 su 4 ci permette di escludere tutti i valori d non ammissibili.

1.2.7 Real Shamir Secret Sharing

Lo schema reale utilizza l'aritmetica modulare (con p numero primo) invece di quella reale e le operazioni effettuate sia con il segreto sia con il polinomio devono essere scelti nel campo dei numeri primi.
L'interpolazione rimane uguale.

Importante. *La regola per scegliere il numero primo p deve essere più grande del dominio del segreto per avere un segreto uniformemente distribuito e non è necessario che sia grande.*

Esempio 2. *La nuova costruzione corretta che utilizza il modulo è la seguente.*
Supponiamo di avere un segreto $s \in [0, 100]$ *in uno schema* $(3, 4)$.

1. *Scegliamo il primo numero primo maggiore dell'intervallo in cui è compreso s.*

$$p = 101$$

2. *Il segreto che vogliamo inviare è:* $s = 32$.

3. *Il polinomio sarà di grado* $t - 1$ *e con termine noto s:*

$$y = Mod[32 + 52x + 3x^2, 101] = (32 + 52x + 3x^2) \mod 101$$

4. *Generiamo i valori pergli shareholders:*

$$\begin{array}{c} x_1 = 1 \to y_1 = y/.x \to x_1 = 87 \\ x_2 = 2 \to y_2 = y/.x \to x_2 = 47 \\ x_3 = 3 \to y_3 = y/.x \to x_3 = 13 \\ x_4 = 6 \to y_4 = y/.x \to x_4 = 48 \end{array}$$

5. *Calcoliamo i valori* $\Lambda_i(0)$ *presupponendo di conoscere le share di* $1, 2, 4$, *sostituendo* $x = 0$ *e ovviamente considerando il modulo:*

$$\begin{array}{c} \Lambda_1(x) = Mod[(0 - x2) * (0 - x_4) * PowerMod[(x_1 - x2) * (x_1 - x4), -1, 101], 101] = 63 \\ \Lambda_2(x) = Mod[(0 - x1) * (0 - x_4) * PowerMod[(x_2 - x1) * (x_2 - x4), -1, 101], 101] = 49 \\ \Lambda_4(x) = Mod[(0 - x1) * (0 - x_2) * PowerMod[(x_4 - x1) * (x_4 - x2), -1, 101], 101] = 91 \end{array}$$

Importante.

$$\Lambda_i(x) = \prod_{m=1, m \neq 1}^{l} \frac{x - x_m}{x_i - x_m} \quad \Lambda_i(x_i) = 1; \quad \Lambda_i(x_m) = 0 \quad for\ m \neq i$$

Questa formula imporrebbe di scrivere il denominatore sotto il segno di frazione, ma questo non è possibile se si effettua il modulo. Quindi, quello che occorre fare è effettuare l'inversa del modulo: in mathematica si utilizza PowerMod. Per esempio per $\Lambda_1(0)$*:*

$$\Lambda_1(x) = Mod[(0-x2)*(0-x_4)*PowerMod[(x_1-x2)*(x_1-x4), -1, 101], 101] = (\frac{(0 - x2) * (0 - x_4)}{(x_1 - x2) * (x_1 - x4)}) \mod p$$

In cui:

$$\frac{1}{(x_1 - x_2) * (x_1 - x_4)} \mod 101 = ((x_1 - x_2)(x_1 - x4))^{-1} \mod 101$$

6. *La forma per ricostruire il segreto è la seguente:*

$$Mod[y_1\Lambda_1(x) + y_2\Lambda_2(x) + y_3\Lambda_3, 101] = (y_1\Lambda_1(x) + y_2\Lambda_2(x) + y_3\Lambda_3) \mod 101 = 32$$

7. *Verifichiamo ora che sia unconditially secure:finché ho anche uno share mancante, allora il segreto potrebbe essere qualsiasi. In particolare, supponiamo che non sia noto* $d = y_1$ *e cicliamo su d da 0 a 100, sapendo che il segreto è compreso in questo intervallo:*

$$Mod[d * \Lambda_1(x) + y_2 * \Lambda_2(x) + y_3 * \Lambda_3(x)/.d \to Range[0, 100]]$$

Come osserviamo i possibili valori sono molteplici e uniformamente distribuiti tra 0 e 100:

```
Out[=]= {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
  14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
  25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
  36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
  47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,
  58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
  69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
  80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
  91, 92, 93, 94, 95, 96, 97, 98, 99, 100}
```

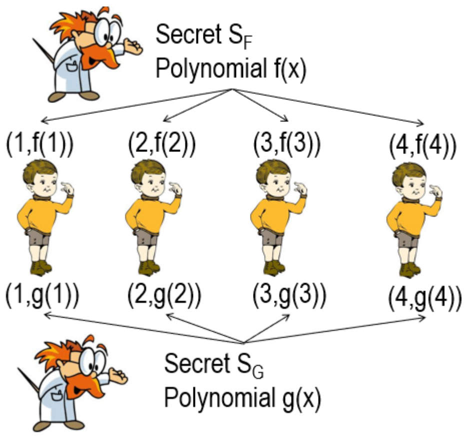
1.3 Secret Sharing: Details

- Gli shares non possono essere più piccoli del segreto, ma al più larghi quanto il segreto. A tale scopo, intuitivamente la conoscenza di uno share deve aggiungere informazioni al segreto, riducendone l'entropia. Quindi, dati t-1 shares, non si può determinare nulla riguardo al segreto ed, inoltre, lo share finale deve contenere quanta più informazione quanta ne ha il segreto stesso.
- Shamir Scheme Ideal quando lo share ha la stessa dimensione del segreto. Vi sono esempio di schemi con chiavi maggiore del segreto come lo schema di *Blackley*.

1.4 Secret Sharing for secure multiparty computation

1.4.1 Homomorphic Property

Assumiamo uno schema (3,4) scheme e supponiamo di avere un Dealer che genera un segreto S_F e un polinomio $f(x)$. Il dealer condividere a 4 shareholders (parties) gli shares. In parallelo, un altro Dealer genera un altro segreto S_G con un altro polinomio $g(x)$ e anche lui genera e condivide gli shares.
Il nostro obiettivo è calcolare S_F+S_G: approccio sarebbe quello di ricostruire inizialmente entrambi i segreti per peoi effettuarne la somma;tuttavia grazie allo schema di Shamir **la somma degli shares è uguale alla somma dei segreti** (ovviamente applicando la formula di ricostruzione) . Quindi, la propietà homomorphic risiede nel fatto che è possibile calcolare $S_F + S_G$ senza sapere i due segreti: effettuare calcoli sui segreti senza rivelare niente dei segreti.

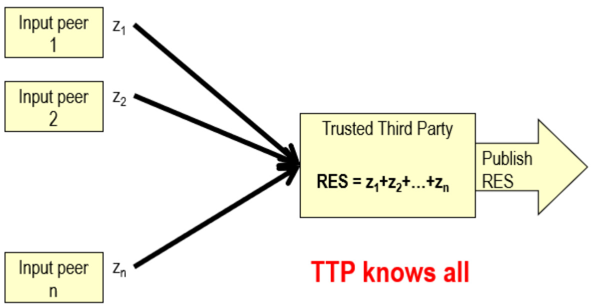


1.4.2 SMC:Secure Multiparty Computation

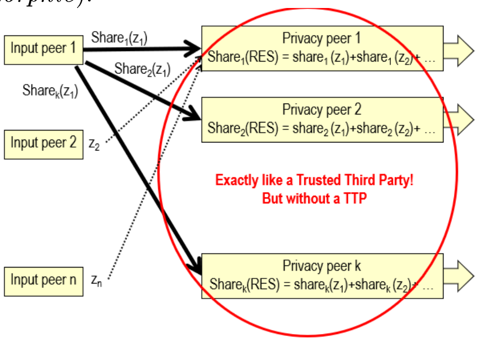
SMC (**Secure Multiparty Computation**) l'obiettivo è quello di calcolare il risultato di una funzione senza rivelare i dati in input. Funziona nel seguente modo:

- Date N parti $P_1, P_2, \ldots, P_n$ ognuna delle quali con valore z_i;
- Calcola la funzione $f(z_1, z_2, \ldots, z_n)$. Il suo risultato è pubblico, ma non si deve dare alcuna informazione riguardo agli input;
- *se l'operazione è una funzione lineare al più pesata da dei coefficienti, allora diventa banale e identico al Secret Sharing Scheme classico*;

Schematicamente, senza l'utilizzo di SMC: La terza parte deve essere trusted e conosce tutto il segreto.



Al contrario, con SMC si ha l'assenza di trusted third parties poiché il segreto è noto solo dall'unione dei privacy peers (*applicando la proprietà homomorphic*):

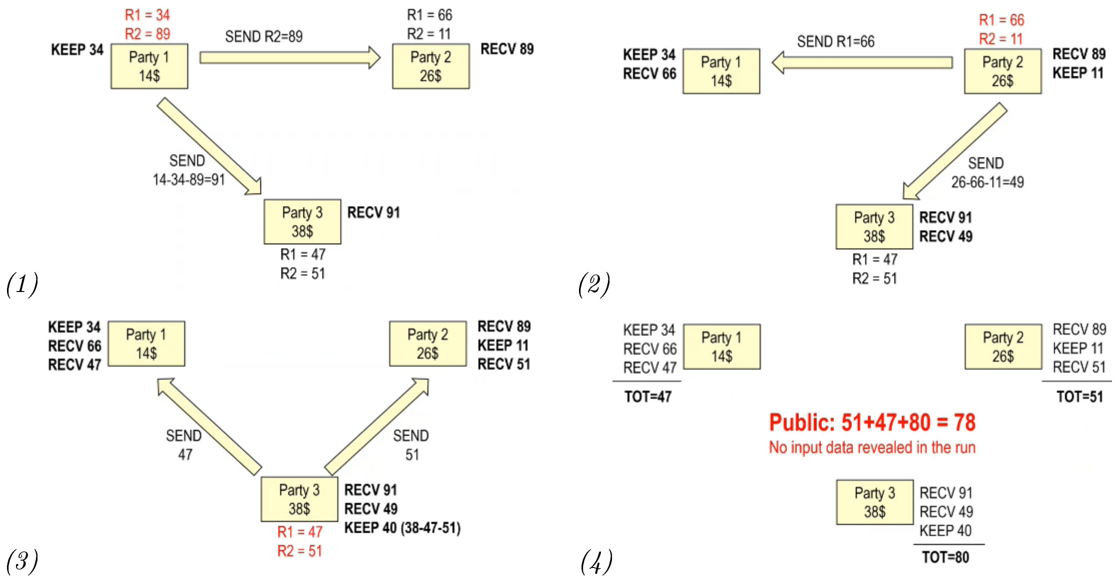


Importante.
- *Si necessitano almeno di 3 peers poiché se ce ne fossero 2 ad una parte basterebbe calcolare il segreto tramite complementarieta;*
- *Ci spossono essere molteplici end users;*
- *Ci devono essere almeno 2 privacy peers, ma più ce ne sono maggiore è la sicurezza e robusto alle collisione;*
- *Soglia sul numero di peers pari a $2 \leq t \leq k$ se è uno schema (t,k).*

1.4.3 Costruzione

- Input peer i:
  1. Input data z_i;
  2. Genera un polinomio $p_i(x)$ di grado $t-1$ con z_i termine noto;
  3. Invia privatamente gli shares $p_i(1), \ldots, p_i(k)$ ai privacy peer $1, \ldots, k$;
- Privace peer m:
  1. Collezione gli input shares $p_1(m), \ldots, p_n(m)$;
  2. Calcola $RES = p_1(m) + \ldots, p_n(m)$;
  3. Pubblica lo share aggregato $RES(m)$;
- Public:
  1. Ricostruisci RES da un numero sufficiente di $RES(m)$ con l'interpolazione di Lagrange.

***Esempio* 3.** *Versione distribuita dello schema precedente:*



Chapter 2

Verifiable Secret Sharing

Quando si immette uno share non è possibile capire se è un valore corretto o meno (*cheating*). Il nostro obiettivo è capire se vi sono delle tecniche per verificare le operazioni crittografiche.
L'**honest-but-curious model** è un modello in cui un attacker segue le regole per ottenere il segreto (quello classico è quello in cui l'attacker cheatta e questo viene detto malicious) . Quindi, il nostro obiettivo è quello di avere dei modi per rilevare e bloccare i cheaters che possono essere sia i dealer che i players.
Si ha bisogno di verifiable secret sharing quando un party può verificare qualora il dealer share è consistente (*rilevare malicious dealer*) oppure le parti possono verificare qualora il segreto rivelato è consistente (*detect cheating parties*) .

2.1 VSS:Feldman VSS Scheme

2.1.1 Feldman Scheme:dealer

- Inizia con un ordinary Shamir scheme:
  1. Genera un polinomio casuale $p(x)$ con grado $(t-1)$, con $P(0) = s$:
  $$p(x) = s + a_1x + a_2x^2 + \cdots + a_{t-2}x^{t-2} + a_{t-1}x^{t-1}$$
  2. Distribuisci uno share ad ognuna delle n parti:
  $$(x_i, y_i) \quad y_i = p(x_i)$$
- Per ogni coefficiente del polinomio pubblichiamo in chiaro i seguenti termni detti **commitments**:
  $$c_0 = g^s; c_1 = g^{a_1}; \ldots; c_{t-1} = g^{a_{t-1}} \mod p$$
  Importante. *Se utilizziamo p, grande numero primo, e i coefficienti del polinomio in* Large Fields *allora i coefficienti c_i non rivelano nulla del segreto poiché si basano sul problema del discrete log (sono difficili da calcolare).*

Commitments

Un commitment è una costruzione crittografica che deve possedere due proprietà:

- *Hiding*: un reicever, ottenuto il commitment, non dovrebbe conoscere nulla riguardo il segreto; **COMMIT PHASE**
- *Binding*: il commitment può essere aperto solo con il valore del segreto. Quindi, il mittente non può barare e cambiarlo. **REVEAL PHASE**

Anche $C = H(x)$ è una sorta di commitment poiché possiede entrambe le proprietà, ma in aggiunta fornisce cumputationally hiding e computationally binding (*con abbastanza tempo posso trovare una collisione*).
Il Feldman Commitment $c = g^x$ è un commitment:

- Hiding computazionale:
  1. Dato $c = g^x \mod p$, computazionalmente legato al ricevitore senza conoscere x (*x deve essere preso in intervallo grande*);
- Perfectly Binding:

1. Il mittente non può trovare alcun x' tale che $g^{x'} = c$;
2.

Importante. *Feldman VSS è solo computazionalmente sicuro. Ciò implica che se s è piccolo,* $c_0 = g^s$ *rivela informazioni sul segreto.*

2.1.2 Feldman Scheme:verifier

- La parte i riceve lo share (x_i, y_i):

1. Le altre parti possono verificare se la parte i è onesta senza sapere il segreto s;
2. Le altre parti, dato che il dealer in questo caso è onesto, possono caloclare:

$$\begin{aligned} c_0 \cdot c_1^{x_i} \cdot c_2^{x_i^2} \cdot c_{t-1}^{x_i^{t-1}} = \\ &= (g^s) \cdot (g^{a_1})^{x_i} \cdot (g^{a_2})^{x_i^2} \cdot \dots \cdot (g^{a_{t-1}})^{x_i^{t-1}} = \\ &= g^s \cdot g^{a_1 \cdot x_i} \cdot g^{a_2 \cdot x_i^2} \cdot \dots \cdot g^{a_{t-1} \cdot x_i^{t-1}} = \\ &= g^{s + a_1 xi + a_2 x_i^2 + \dots + a_{t-1} x_i^{t-1}} = g^p(x_i) \end{aligned}$$

3. così facendo le altre parti effettuano una homomorphic computation di $p(x_i)$ all'esponente ottenendo $g^{p(x_i)}$;
4. Ora per verificare le parti hannno y_{x_i}, calcolando $g^{y_{x_i}}$ e se è uguale a quello calcolato al passo precedente, allora lo share è verificato.

2.2 Pedersen Commitment

Non è possibile avere uno schema perfectly hiding:uno schema in cui un commitment sia contemporaneamente perfectly hiding e perfectly binding.
Tuttavia, esistono commitment che sono perfectly hiding, ma computationally binding. L'approccio di Feldman funzionava poiché si ha un commitment che è anche homomorphic.
Quindi, vogliamo uno scheme perfectly hiding, ma che sia anche homomorphic: **Pedersen Commitment**.

Definizione 1 (Pedersen Commitment)**.** *Dati g e h pubblici:*

$$Commit(a, r) = g^a \cdot h^r \mod p$$

In cui a è il segreto, r numero scelto truly random.

Abbiamo ottenuto la proprietà homomorphic poiché:

$$\begin{aligned} Commit(a + b, r_a + r_b) = \\ &= g^a h^{r_a} \cdot g^b h^{r_b} = \\ &= Commit(a, r_a) \cdot Commit(b, r_b) \end{aligned}$$

Abbiamo ottenuto anche il perfectly hiding. Infatti, dato il commitment $c = g^a h^r$ questo permette di nasconodere qualsiasi valore di a: per qualsiasi $a' \neq a$, possiamo trovare un unico r' tale che:

$$Commit(a', r') = g^{a'} h^{r'} = g^a h^r = Commit(a, r)$$

Questo commitment è solamente computationally binding:il mittente non dovrebbe essere in grado di trovare un a', ma si può arrivare ad un trapdoor commitment scheme.

- Sia $h = g^w$ i.e.$w = \log_g h$;
- Sappiamo a, r, ci viene dato a' e cerchiamo un r' tale che:

$$\begin{aligned} g^a h^r = g^{a'} h^{r'} &\implies g^a g^{wr} = g^{a'} g^{wr'} \implies \\ &\implies g^{a+wr} = g^{a'+wr'} \implies \\ &\implies a + wr = a' + wr' \mod q \implies \\ &\implies r' = w^{-1}(a - a' + wr) = r' = (a - a')w^{-1} + r \end{aligned}$$

2.3 Pdersen VSS:dealer

- Genera due polinomi casuali:

$$f(x) = s + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$$
$$f'(x) = r + b_1x + b_2x^2 + \cdots + b_{t-1}x^{t-1}$$

- Dai ad ogni parte lo share x_i, y_i, z_i

$$y_i = f(x_i) = s + a_1x_i + \cdots + a_{t-1}x^{t-1}$$
$$z_i = f'(x_i) = r + b_1x_i + \cdots + b_{t-1}x^{t-1}$$

- Pubblica i commitment di Pedersen:

$$\begin{aligned} c_0 &= g^s h^r \\ c_1 &= g^{a_1} h^{b_1} \\ &\vdots \\ c_{t-1} &= g^{a_{t-1}} h^{b_{t-1}} \end{aligned}$$

2.4 Pedersen VSS:verifier

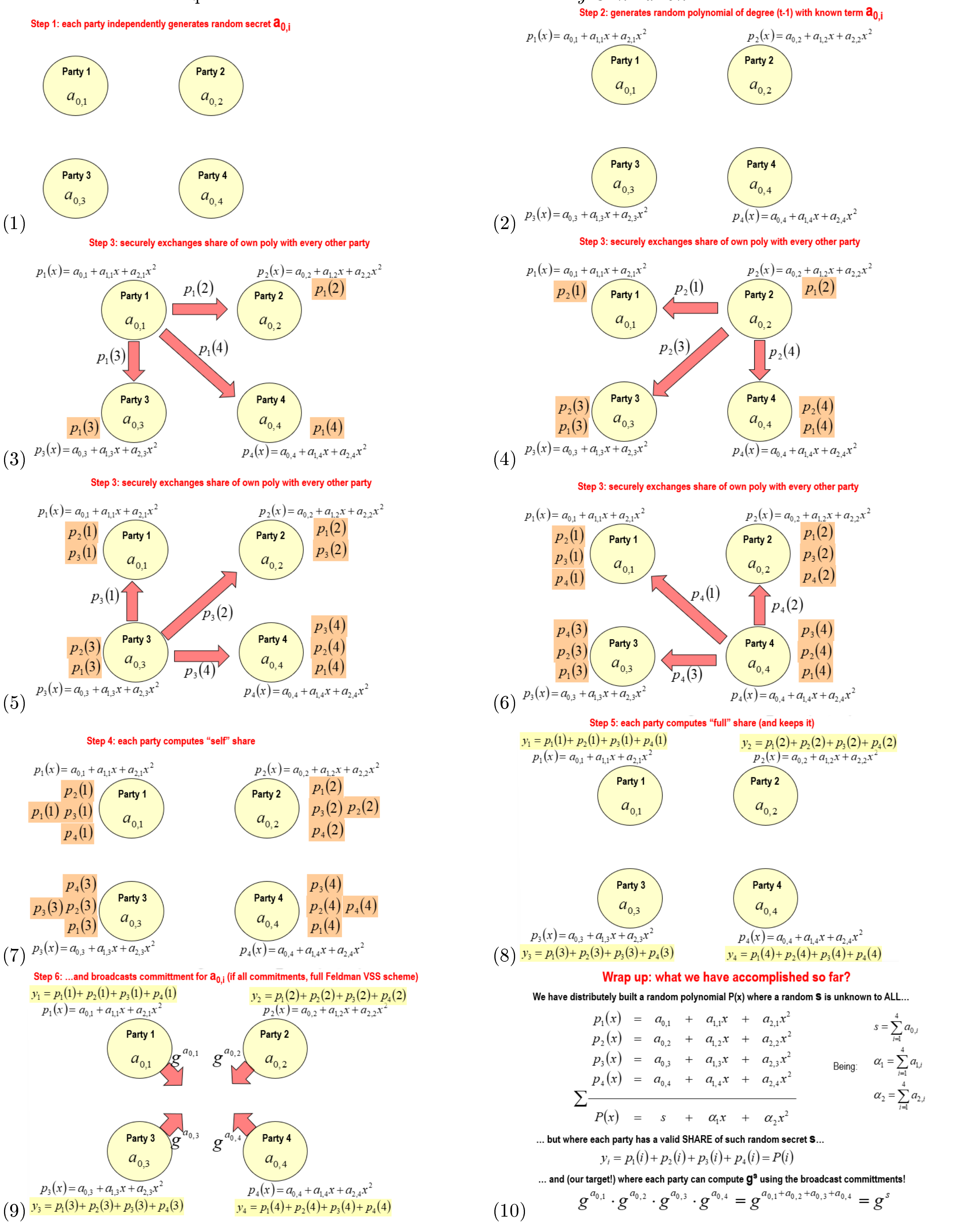
- La parte i-esima riceve lo share x_i, y_i, z_i;
- Verifica in $\mod p$ che:

$$\begin{aligned} c_0 \cdot c_1 \cdot x_i \cdot \cdots \cdot c_{t-1}^{x_i^{t-1}} &= (g^s h^r) \cdot (g^{a_1} h^{b_1})^{x_i} \cdot \cdots \cdot (g^{a_{t-1}} h^{b_{t-1}})^{x_i^{t-1}} = \\ &= g^s \cdot g^{a_1 x_i} \cdot g^{a_2 x_i^2} \cdot \cdots \cdot g^{a_{t-1} x_i^{t-1}} \cdot h^r \cdot h^{b_1 x_i} \cdot \cdots \cdot h^{b_{t-1} x_i^{t-1}} = \\ &= g^{s + a_1 x_i + a_2 x_i^2 + \cdots + a_{t-1} x_i^{t-1}} \cdot h^{r + b_1 x_i + b_2 x_i^2 + \cdots + b_{t-1} x_i{}^{t-1}} = \\ &= g^{y_i} h^{z_i} \end{aligned}$$

- Se l'equazione è soddisfatta allora siamo riusciti a verificare;

2.5 Distribuited Key Generation

Nei sistemi crittografici basati su DLog, si ha: x chiave privata e g^x chiave pubblica. Vogliamo generare una coppia $(Pub_K, Priv_K)$ tale che tutti conoscono $Pubb_K$, ma nessuno conosce $Priv_K$.
Ciò ci è utile in tutti i casi in cui non vogliamo rilevare la chiave privata o deve essere ricostruita in seguito.
Lo schema che realizza questa idea è chiamato **DKG** *Distribuited Key Generation*.



Chapter 3

Multiplicative Group $\mod p$

3.1 Gruppo

Un **gruppo** $(G, \circ)$ è una struttura algebrica in cui G definisce l'insieme degli elementi (*membri del gruppo*) e $\circ$ è l'operazione del gruppo. Questa operazione deve soddisfare:

- **Chiusura**:presi due elemnti g_1, g_2 del gruppo, allora $g_x = g_1 \circ g_2$ deve appartenere al gruppo;
- **Identità**:deve esiste un membro del gruppo tale che $g \circ I = I \circ g = g$;
- **Inversa**:per ogni g esiste g^{-1} tale che $g \circ g^{-1} = I$;
- **Associtativa**:per qualsiasi g_1, g_2, g_3 deve valere $(g_1 \circ g_2) \circ g_3 = g_1 \circ (\circ g_3)$

Inoltre, se l'operazione è anche commutativa, si ha un **Gruppo Abeliano**.

3.1.1 Gruppo Z_p^*

Z_p^* è la sintassi utilizzata per indicare il gruppo moltiplicativo modulo p:gli elementi di questo gruppo **finito** sono $p-1$ composti da $\{1, 2, \ldots, p-1\}$ e l'unica operazione è la moltiplicazione (*se l'avessimo considerata avremmo avuto un campo F_p*).
Infatti, questo insieme di elementi con questa operazione, è un gruppo poiché è chiuso, è associativo, indentità e commutatività (Gruppo Abeliano) e presenta l'inversa.

***Esempio* 4** (Z_{11}^*)**.**
- *$p-1 = 10$ elementi:$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$;*
- *Inverse in modulo p:*

$$\begin{array}{ll} \Rightarrow 1 \rightarrow 1 & \\ \Rightarrow 2 \rightarrow 6 & 6 \rightarrow 2 \\ \Rightarrow 3 \rightarrow 4 & 4 \rightarrow 3 \\ \Rightarrow 5 \rightarrow 9 & 9 \rightarrow 5 \\ \Rightarrow 7 \rightarrow 8 & 2 \rightarrow 7 \\ \Rightarrow 10 \rightarrow 10 & \end{array}$$

- *Per grandi gruppi si utilizza l'algoritmo di Euclide esteso.*

3.1.2 Gruppi moltiplicativi:exponentiation

L'esponenziazion è una operazione che appartiene al gruppo Z_p^* poiché è l'applicazione della stessa operazione moltiplicativa più volte:$x^k = x \circ x \circ x \circ x \circ \ldots \circ x$(k volte).

Definizione 2 (Generatore di un Gruppo)**.** *Il generatore del gruppo è un valore g tale che $\{g^0, g^1, \ldots, g^{m-1}\}$ siano tutti gli elementi del gruppo*

Importante. *Se m è primo, allora quasiasi membro del gruppo è un generatore ad eccezione dell'identità.*
*Z_p^* non è un **Prime-Order Group** infatti se p è primo, non lo è $p-1$.*

Esempio 5 (Z_{11}^*).

- *$p-1 = 10$ elementi:*$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$*;*
- *Quali sono i generatori?* $\{g^1, g^2, g^3, \dots, g^{10}\}$*?*
-

$$\begin{gathered} g = 2 \rightarrow \{2,4,8,5,10,9,7,3,6,1\} \quad \textit{OK,è un generatore} \\ g = 3 \rightarrow \{3,9,5,4,1,3,,9,5,4,1\} \quad \textit{NO, è un sottogruppo di ordine 5} \\ etc\dots \end{gathered}$$

- *La cardinalità di sottogruppi deve essere uno dei fattori del numero degli elementi del gruppo: se $(p-1) = kq$ è il numero di elementi, allora potremmo incontrare sottogruppi di cardinalità k o q.*

3.2 Strong Primes

In crittografia siamo interessati a prendere *strong primes.*

Definizione 3. *Un numero primo p è detto **strong prime** se $p = 2q + 1$ con q numero primo.*

Quindi, qualsiasi sia x ad eccezione di 1 e p-1:

- Genera l'intero gruppo;
- Genera un sottogruppo di ordina primo q.

3.3 Quadratic Residue Subgroup

Definizione 4. *Sia $x \in Z_p^*$ è un **quadratic residue** se ammette la radice quadrata in Z_p^*.*

Per esempio esiste a tale che $a^2 \mod p = x$.
Se soddisfo la definizione allora sono il generatore di un sottogruppo;altrimenti sono il generatore dell'intero gruppo.

- QR forma un sottogruppo di ordine $\frac{p-1}{2}$;
- QR Test: **Legendre Symbol**:$a \in QR$ se $a^{\frac{p-1}{2}} \mod p = 1$ (se pari a -1 allora sono un generatore)

Importante. *Vedere applicazioni nel file matematica 41-vss-example-1-qr e -correct.*

Chapter 4

Threshold and policy-based cryptography

La **threshold cryptography**è un tipo di crittografia in cui l'encryption o una singnature può essere decryptata solo quando vi sono un certo numero di partecipanti. Si può anche definire come group crpyto composta da VSS con tecniche standard di criptografia.

4.1 Threshold Encryption

4.1.1 Public Key Encryption with DLOG

Vogliamo applicare il DLOG con la public encryption: schema El Gamal (DH adattato). Si è fatta questa scelta poiché viene facilmente implementato nelle curve ellittiche.

4.1.2 El-Gamal:background

Lo schema di El-Gamal modifica il protocollo di accordo delle chiavi in un cipher asimmetrico.

4.1.3 El-Gamal:Sketch

- Operazioni in $\mod p$ con p numero primo elevato;
- g generatore di gruppo;
- s chive privata;
- $h = g^s$ chiave pubblica;
- r valore casuale;

Importante.

- *g^s, g^r noti a tutti:il primo pubblico, il secondo nel ciphertext;*
- *s noto solo dal receiver;*
- *r noto solo dal transmitter;*
- *g^{sr} nessun altro può calcolarlo*

Quindi per **cifrare**:

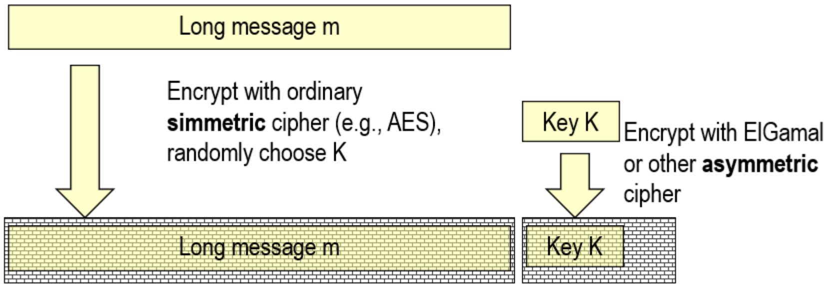
$$(R, c) = (g^r, m \cdot h^r)$$

Per **decifrare**:

$$m = c \cdot R^{-s} = \frac{c}{(g^r)^s} = \frac{m \cdot h^r}{g^{rs}} = \frac{m \cdot g^{sr}}{g^{rs}}$$

4.1.4 Asymmetrc Criptography

Schema su come si applica El-Gamal:



4.1.5 ECIES=Hybrid Encryption 5g

ECIES sta per *Elliptic Curver Integrated Encryption Scheme* e venne per la prima volta standardizzato nell'IMSI del 5G. Assumiamo che nella nostra SIM ho installato la chiave pubblica del provider della SIM (Trusted) g^{HN} (Home Network). Nel momento dell'autenticazione **Encrypt-then-MAC**:

- generiamo x casuale, il coefficente ephemeral g^x;
- calcoliamo $K = HKDF(g^{HNx})$ ed inviamo $AES_k(SUP)$ (SUP è l'IMSI);
- aggiungiamo $HMAC$ (integrità);
- Il messaggio sarà quindi $(g^x, HMAC(AES_k(MSG)))$

In ricezione:

- Riceviamo $(g^x, HMAC(AES_k(MSG)))$:
  1. Con g^x e la chiave privata dell'Home Network ricostruiamo la chiave $(g^x)^{HN}$;
  2. Deriviamo la chiave $K = HKDF(g^{HNx})$;
- Decrypto il dato in ingresso

4.2 Threshold El-Gamal

L'idea di El-Gamal è quella di distribuire gli shares della chiave privata s e ricostruirla quando si necessità di decriptare solo quando vi sono un certo numero (threshold) di riceiver che cooperano e questo messaggio può essere letto solamente da queste parti. Ovviamente, nessuno possiede la chiave privata.
Quindi, il mittente invia il messaggio, i receiver ricevono il messaggio e ricostruiscono il segreto per decryptarlo. Questo apporoccio non è molto conveniente poiché può essere usato solamente una volta, nel momento in cui si ripete siamo esposti a rischi.
Ricordiamo che la **encryption** è:

$$(g^{r_1}, m_1 \cdot h^{r_1})$$

La **decryption** è:

$$m_1 = \frac{c}{(g^{r_1})^s}$$

Ma se volessi decryptare anche un altro messaggio, il mittente dovrebbe generare altri due coefficienti da condividere cosicché il receiver possa decryptare:

$$m_2 = \frac{c_2}{(g^{r_2})^s}$$

Se non facessì così ed utilizzerei sempre lo stesso segreto **perdo** la sicurezza semantica. Tuttavia, notiamo che per decriptare i messaggi il denominatore viene sempre elevato per il segreto s . Esiste un modo per mantenere il segreto e quindi calcolare il denominatore del decrypt senza rivelare il segreto?Una soluzione potrebbe essere quella di interpolare gli shares all'esponente sfruttando la proprietà degli esponenziali:

$$A^x = A^{x_1} \cdot A^{x_2} = A^{x_1+x_2}$$

In particolare:

- Manentiamo:
  1. Il polinomio (Pedersen Scheme): $p(x) = s + a_1x + a_2x + \cdots + a_{t-2}x^{t-2} + a_{t-1}x^{t-1}$;
  2. Gli shares:$(x_i, y_i) \quad y_i = p(x_i)$;
  3. La formula per ricostruire il segreto:
  $$y = \sum_{shares\ x_i}^{t} y_i \Lambda_{x_i}$$
  In cui Λ_{x_i} è la base del polinomio calcolata come:
  $$\Lambda_{x_i} = \Lambda_{x_i}(0) = \prod_{x_k \neq x_i}^{l} \frac{-x_k}{x_i - x_k}$$
- Effettuato all'esponente:
$$\prod A^{y_i \Lambda_{x_i}} = A^{\sum y_i \Lambda_{x_i} = A^s}$$