Computer and Network Security

Lorenzo Rossi

January 16, 2022

Contents

Part I

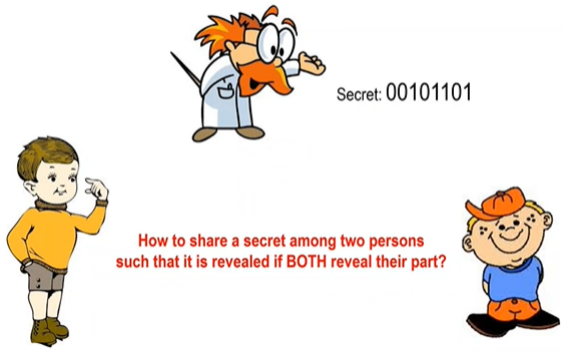
Third Midterm

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.$RAND \mod 256 \rightarrow 180$;
3. Esegui $S - RAND \mod N$ i.e. $S - RAND \mod 256 \rightarrow 121$;

Importante. *Questo schema è equivalente ad One Time Pad poiché abbiamo sommato un numero pseudocasuale con un numero casuale (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.*

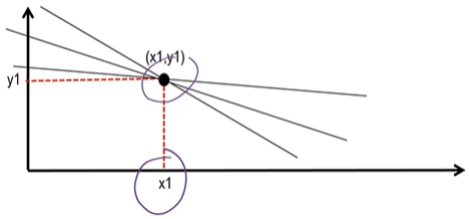
Questo metodo è più facile da implementare per essere condiviso con N shareholders. In particolare, genero 3 quantità truly random ed effettua la differenza tra il segreto e queste 3 quantità modulo N. Nel caso un attacker, riuscisse ad ottenere un numero sufficiente di share non può comunque ottenere 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 parametro 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 casualmente;
  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: $x_1 = 1 \rightarrow share = (1, 54)$;
  - Shareholder 2: $x_2 = 2 \rightarrow share = (2, 69)$;
  - Shareholder 3: $x_3 = 3 \rightarrow share = (3, 84)$;
  - ...

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'interpolazione 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 grande 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. *Effettuiamo uno schema (3,4) in cui per conoscere il segreto dobbiamo conoscere 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 rispettivamente i valori $1, 2, 3, 4$, si ottengono i seguenti 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 per gli shareholders:*

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

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

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

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 unconditionally 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 uniformemente 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}
```

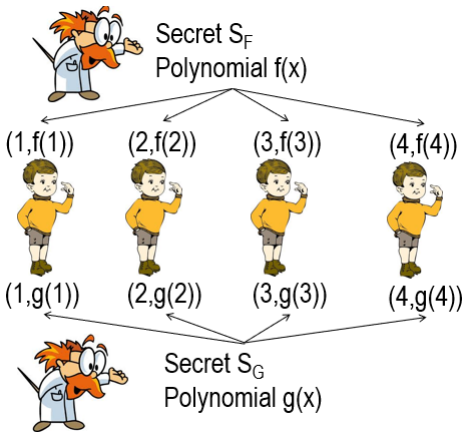
1.3 Secret Sharing: Details

- 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 poi 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 proprietà 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.

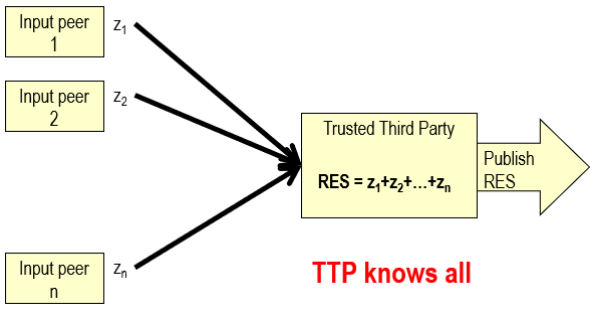


1.4.2 SMC:Secure Multiparty Computation

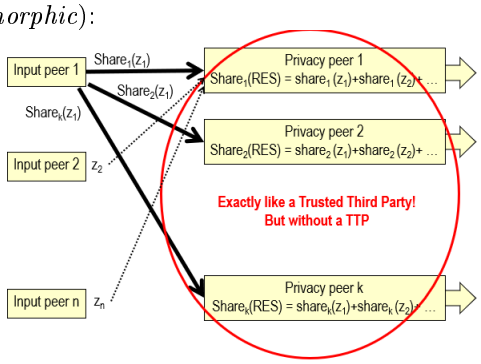
SMC (**Secure Multiparty Computation**) l'obiettivo è quello di calcolare il risultato di una funzione senza rivelare i dati in input. Funziona nel seguente modo:

- 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*):



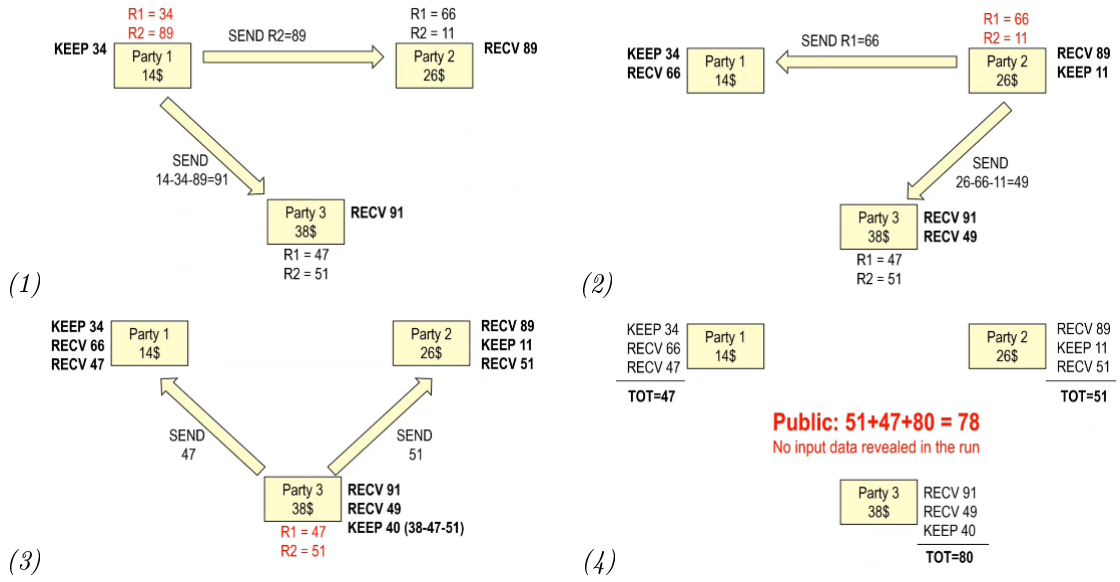
Importante. • *Si necessitano almeno di 3 peers poiché se ce ne fossero 2 ad una parte basterebbe calcolare il segreto tramite complementarietà;*

- *Ci possono 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 $1, \ldots, k$;
- Privacy 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 $P(0) = s$:

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

  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 termini 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 receiver, 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 computationally 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 calcolare:

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

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 hanno 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 nascondere 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 Pedersen VSS:dealer

- Genera due polinomi casuali:

$$f(x) = s + a_1 x + a_2 x^2 + \cdots + a_{t-1} x^{t-1}$$
$$f'(x) = r + b_1 x + b_2 x^2 + \cdots + b_{t-1} x^{t-1}$$

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

$$y_i = f(x_i) = s + a_1 x_i + \cdots + a_{t-1} x^{t-1}$$
$$z_i = f'(x_i) = r + b_1 x_i + \cdots + b_{t-1} x^{t-1}$$

- Pubblica i commitment di Pedersen:

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

2.4 Pedersen VSS:verifier

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

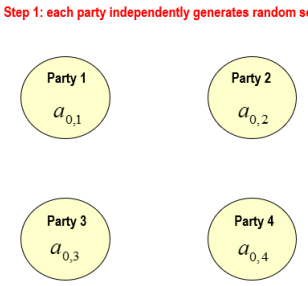
$$\begin{aligned} c_0 \cdot c_1 \cdot x_i \cdot \cdots \cdot c_{t-1}^{x_i^{t-1}} &= (g^s h^r) \cdot (g^{a_1} h^{b_1})^{x_i} \cdot \cdots \cdot (g^{a_{t-1}} h^{b_{t-1}})^{x_i^{t-1}} = \\ &= g^s \cdot g^{a_1 x_i} \cdot g^{a_2 x_i^2} \cdot \cdots \cdot g^{a_{t-1} x_i^{t-1}} \cdot h^r \cdot h^{b_1 x_i} \cdot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

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

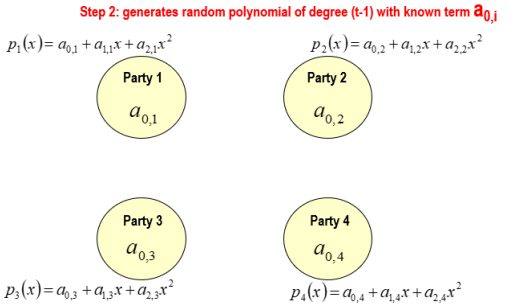
2.5 Distributed Key Generation

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** *Distributed Key Generation*.

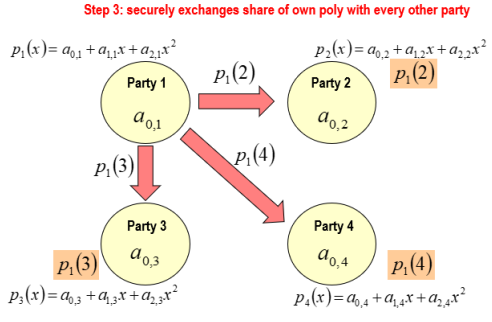
(1)



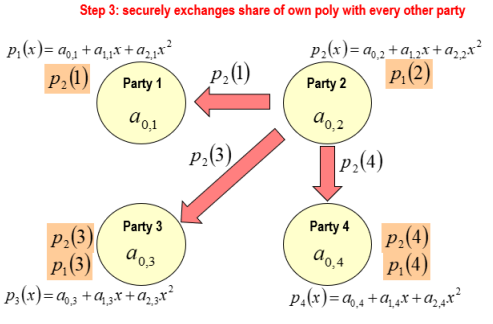
(2)



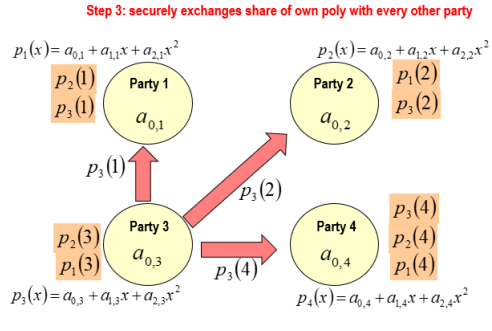
(3)



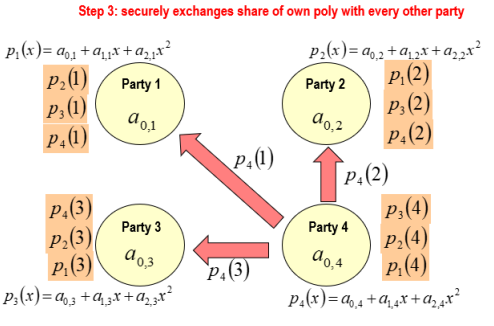
(4)



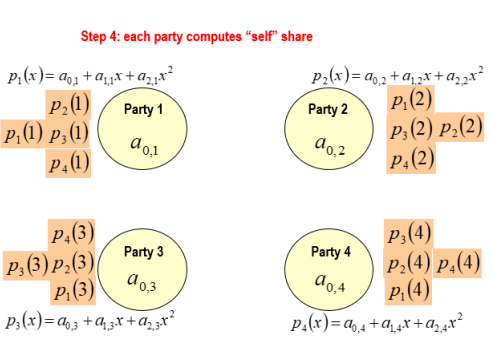
(5)



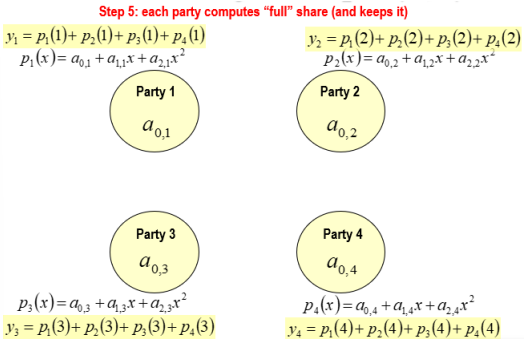
(6)



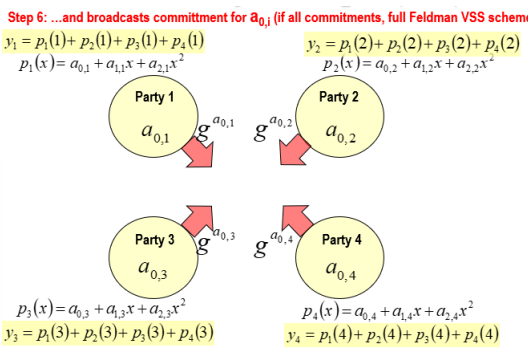
(7)



(8)



(9)



(10)

Wrap up: what we have accomplished so far?

We have distributely built a random polynomial P(x) where a random **s** is unknown to ALL...

$$\begin{aligned} p_1(x) &= a_{0,1} + a_{1,1}x + a_{2,1}x^2 \\ p_2(x) &= a_{0,2} + a_{1,2}x + a_{2,2}x^2 \\ p_3(x) &= a_{0,3} + a_{1,3}x + a_{2,3}x^2 \\ p_4(x) &= a_{0,4} + a_{1,4}x + a_{2,4}x^2 \\ \sum & \\ P(x) &= s + \alpha_1 x + \alpha_2 x^2 \end{aligned}$$

Being: $s = \sum_{i=1}^{4} a_{0,i}$, $\alpha_1 = \sum_{i=1}^{4} a_{1,i}$, $\alpha_2 = \sum_{i=1}^{4} a_{2,i}$

... but where each party has a valid SHARE of such random secret **s**...

$$y_i = p_1(i) + p_2(i) + p_3(i) + p_4(i) = P(i)$$

... and (our target!) where each party can compute g^s using the broadcast committments!

$$g^{a_{0,1}} \cdot g^{a_{0,2}} \cdot g^{a_{0,3}} \cdot g^{a_{0,4}} = g^{a_{0,1}+a_{0,2}+a_{0,3}+a_{0,4}} = g^s$$

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 elementi 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 $g \circ g^{-1} = I$;
- **Associativa**: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, identità 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:*

$$\Rightarrow 1 \to 1$$
$$\Rightarrow 2 \to 6 \qquad 6 \to 2$$
$$\Rightarrow 3 \to 4 \qquad 4 \to 3$$
$$\Rightarrow 5 \to 9 \qquad 9 \to 5$$
$$\Rightarrow 7 \to 8 \qquad 2 \to 7$$
$$\Rightarrow 10 \to 10$$

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

3.1.2 Gruppi moltiplicativi:exponentiation

L'esponenziazione 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 qualsiasi 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?* $\{g^1, g^2, g^3, \ldots, g^{10}\}$*?*
-

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

- *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 signature può essere decryptata solo quando vi sono un certo numero di partecipanti. Si può anche definire come group crypto 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 chiave 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**:

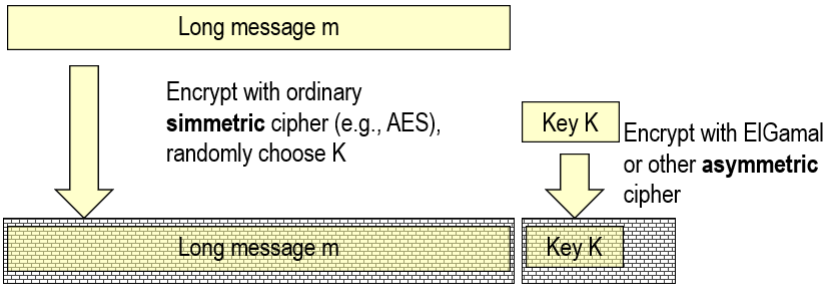
$$(R, c) = (g^r, m \cdot h^r)$$

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 Asymmetric Cryptography

Schema su come si applica El-Gamal:



4.1.5 ECIES=Hybrid Encryption 5g

ECIES sta per *Elliptic Curve 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 coefficiente 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 $K = HKDF(g^{HNx})$;
- Decrypt 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 receiver 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 approccio 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 facessi 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:

- Manteniamo:
  1. Il polinomio (Pedersen Scheme): $p(x) = s + a_1 x + a_2 x + \cdots + a_{t-2} x^{t-2} + a_{t-1} x^{t-1}$;
  2. Gli shares:$(x_i, y_i) \quad y_i = p(x_i)$;
  3. La formula per ricostruire il segreto:
  $$y = \sum_{shares}^{t} y_i \Lambda_{x_i}$$
  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:
$$\prod A^{y_i \Lambda_{x_i}} = A^{\sum y_i \Lambda_{x_i} = A^s}$$

4.2.1 Soluzione

- Ogni parte possiede uno share $(x_i, y_i) \quad y_i = p(x_i)$;
- Le parti prendono g^r dal ciphertext $g^r, m \cdot h^r$;
- Calcolano il coefficiente di Lagrange:
$$\Lambda_{x_i} = \Lambda_{x_i}(0) = \prod_{x_k \neq x_i}^{l} \frac{-x_k}{x_i - x_k}$$
- Calcola exponent share $(g^r)^{y_i \Lambda_{x_i}}$
- Un numero sufficiente di shares permette di ricostruire il termine di decryption:
$$\prod (g^r)^{y_i \Lambda_{x_i}} = (g^r)^{\sum y_i \Lambda_{x_i}} = (g^r)^s = g^{rs}$$

4.3 Threshold Signature

Si utilizza la threshold signature ogni qual vola in cui vogliamo che t su n membri di un gruppo possono firmare un messaggio garantendo, di conseguenza, che la validità di un messaggio viene garantita da più parti, che un membro del gruppo è certificato dagli altri e la fiducia viene riposta in più certification authority. Inoltre, se si hanno meno di t membri è impossibile forgiare una signature.

4.3.1 RSA Signature

Ricordiamo il funzionamento di RSA:

- Genera due numeri primi grandi **p,q (Segreti)**;
- Calcola RSA Module:$N = pq$ **Può essere pubblico**;
- Calcola $\Phi(N) = (p-1)(q-1)$ **Segreto**;
- Genera chiave arbitraria e coprimi con $\Phi(N)$ e prendi come **chiave pubblica** $1 < e < \Phi(N)$
- Genera **chiave privata** d tale che $e \times d = 1 \mod \Phi(N)$;
- Firma il messaggio $[m, H(m)^d]$

Adattandolo in una threshold function:

- **Dealer**:
$$f(x) = d + a_1 x + a_2 x^2 + \cdots + a_{t-2} x^{t-2} + a_{t-1} x^{t-1}$$

- Shares della parte P_i:
$$(x_i, y_i = f(x_i)) \mod \Phi(N)$$
;
- Firma il messaggio $m, H(m)$;
- Ora tutte le part hanno uno share del tipo:
$$H(m)^{y_i \Lambda_{x_i}} \mod N$$
;
- Ricostruzione della firma:
$$\prod H(m)^{y_i \Lambda_{x_i}} = H(m)^{\sum y_i \Lambda_{x_i}} = H(m)^d$$

Importante. *Vedi file mathematica 43-thresholdRSA.nb per vedere che questa costruzione è sbagliata;*

La costruzione è sbagliata perché stiamo operando in modulo e quindi l'operazione di divisione all'interno del coefficiente di Lagrange è sbagliata dato che occorre utilizzare l'inversa del modulo e questa non è sempre possibile (a meno che il modulo e il numero da invertire siano coprimi) . Oltre a questo non possiamo calcolare il modulo poiché non conosciamo $\Phi(N)$. In particolare:
$$\Lambda_{x_i} = \Lambda_{x_i}(0) = \prod_{x_k \neq x_i}^{l} \frac{-x_k}{x_i - x_k} = \frac{\alpha_{x_i}}{\beta_{x_i}}$$
Quindi occorre calcolare:
$$H(m)^{y_i \Lambda_{x_i}} = H(m)^{y_i \Lambda_{x_i} \beta_{x_i}^{-1}} \mod N$$
In cui:
$$\beta_{x_i} \cdot \beta_{x_i} = 1 \mod \Phi(N)$$
ma noi non siamo a conoscenza di $\Phi(N)$ e non sempre l'inversa è calcolabile.

RSA:signature Shoup scheme

Tuttavia, è possibile evitare di effettuare l'inversa tramite lo schema di Shoup:

- Assumi $x_i = i$ e L giocatori;
- Osservando il denominatore del polinomio di Lagrange, nel caso peggiore abbiamo:
$$\Lambda_i(x) = \prod_{shares\ k \neq i} \frac{x-k}{i-k} = \frac{something(x)}{(i-1)(i-2)\cdots(u-(-1))(i-(i+1))\cdots(i-L)}$$
- Il denominatore può essere suddiviso in $i!(L-i)!$ che moltiplicato per $L!$ ritorna **sicuramente** un intero:
$$L! \cdot \Lambda_i(x) := \overline{\Lambda_i}(x)$$
- Modificando la costruzione del capitolo precedente, calcoliamo gli shares computabili INTERI:
$$H(m)^{y_i \overline{\Lambda_{x_i}}}$$
- Ricostruiamo la firma con un $L!$ aggiuntivo come:
$$\prod H(m)^{y_i \overline{\Lambda_{x_i}}} = H(m)^{L! \sum y_i \Lambda_{x_i}} = H(m)^{d \cdot L!} \mod N$$

Importante. *L'**RSA Modulus Attack** si basa sull'errore di un client nell'aver utilizzato per cifrare lo stesso messaggio m con differenti chiavi pubbliche. Questo perché si utilizza due volte lo stesso modulo N: il messaggio m può essere decifrato calcolando due coefficienti r, s ottenuti applicando l'algoritmo di Euclide esteso.*
Infatti, eseguendo $e_a \cdot r + e_b \cdot s = \gcd e_a, e_b$ si trova r, s che per decifrare il messaggio vanno utilizzati nei rispettivi ciphertext come esponente e moltiplicarli tra di loro.
$$(m^{e_a})^r \cdot (m^{e_b})^s = m^{e_a + r + e_b s} = m$$
L'unica cosa che rimane da levare è $L!$ e a tale scopo utilizziamo l'idea del **RSA Modulus Attack**:

- Sia $L! = \Delta$;
- Abbiamo $H(m)^{d \cdot L!} = H(m)^{d \cdot \Delta}$;
- Vogliamo $H(m)^d = y$, ma abbiamo $H(m)^{d \cdot \Delta} = y^{\Delta}$;
- Inoltre $H(m) = H(m)^{de} = y^e$;
- Se e e Δ sono coprimi, possiamo applicare l'attacco e ricostruire il messaggio.

4.4 Mobile Devices Resilient to Capture

è un dispositivo che non può essere usato da nessuno ad eccezione del proprietario legittimo. Assumiamo che il nucleo del dispositivo è una chiave segreta e i possibili approcci di sicurezza possono essere i seguenti:

- Lock/Unlock la chiave con password:soluzione debole poiché si può effettuare un dictionary attack;
- Salvare il segreto in un sistema hardware sicuro:soluzione costosa e bisogna che sia sicuro anche su aspetti di attacchi di tipo side channels;
- Scaricare dinamicamente la chiave dalla repository di rete:soluzione non adatta poiché occorre fidarsi della repository;

La soluzione a questo problema venne fornita da **MacKenzie+Reiter**

4.5 MacKenzie+Reiter

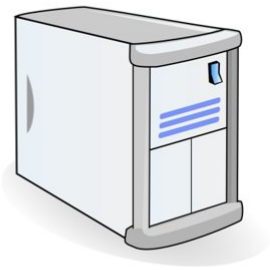
L'assunzione è che il dispositivo deve essere connesso quando viene utilizzato.La soluzione da loro proposta si basa su 3 giocatori: me stesso (utilizzo la password), computer (salva il segreto) e un server remoto (untrusted server). Inoltre, come altro fondamento, si utilizza il concetto di **capture-protection server**:il server, sebbene sia non trusted, conferma che il dispositivo rimane al proprietario prima di consentire l'utilizzo della chiave. Si compone di due livelli.

4.5.1 Scenario

Password P

Many possible attack scenarios
- **password known**
- **device stolen**
- **server cracked (no trusted server!)**

Good: Solution resilient to any crack
Better: Solution resilient to more than 1 crack!



Secret Key SKd
Public Key PKd

Secret Key SKs
Public Key PKs

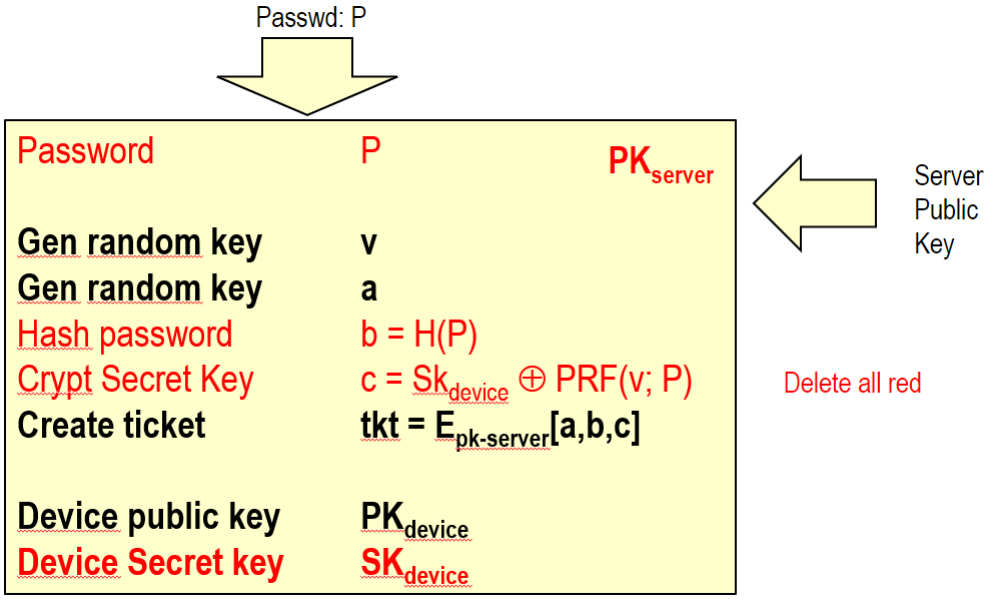
4.5.2 Basic Solution

La soluzione base risolve tutte le combinazioni di attacchi precedenti tranne una in cui bisogna applicare tecniche più avanzate.

Tickets

- Assunzioni: il dispositivo deve essere connesso in rete;
- Protection: cifra la chiave del dispositivo cosicché possa essere decifrata solamente con la cooperazione del server;
- Idea:
  1. Invia il ticket cifrato al server (contiene dati per autenticare l'utente e non si ha bisogno che siano salvati);
  2. Usa il contenuto del ticket per autenticare l'utente:
  3. Usa il contenuto del ticket per decifrare parzialmente la chiave del dispositivo.

Protocol Inizialization

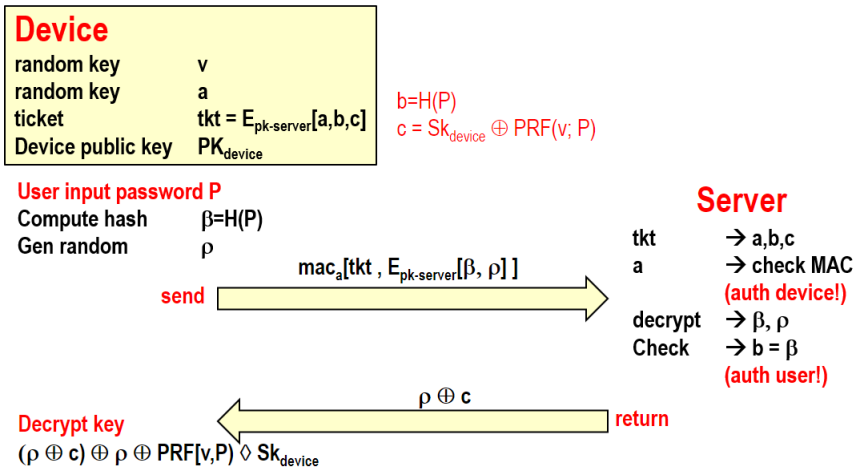


Il tempo in cui registriamo il dispositivo inseriamo la vera password P e salvata temporaneamente nel dispositivo.Sappiamo la chiave pubblica del serve PK_{server} e quella pubblica PK_{device} e privata SK_{device} del dispositivo.

- generiamo due chiavi casuali v, a;
- Eseguiamo l'hash della password $b = H(P)$;
- Cifriamo la chiave segreta $c = SK_{device} \bigoplus PRF(v, P)$;
- Crea il ticket $tkt = E_{PK_{server}[a,b,c]}$;

Importante. *Tutto ciò che è in rosso viene cancellato appena letto.*

Key Retrival



Accedo al dispositivo inserendo la password P calcolo localmente $\beta = H(P)$ e genero il valore casuale ρ . Ora invio al server un messaggio che contiene:

- $tkt = E_{PK_{server}[a,b,c]}$;
- $E_{PK_{server}}[\beta, \rho]$;

Autentico questo messaggio tramite HMAC con chiave a.
Ora chiedo assistenza al server:

- Apro il ticket e prelevo il valore a;
- Verifico che il messaggio è stato inviato effettivamente da quel dispositivo controllando il MAC;
- Decrypt l'hash della password β, ρ;
- Controllo che la password originale sia uguale a quella ricevuta $\beta = b$;
- Ora so che l'utente e il device sono reali;
- Il server ritorna $p \bigoplus s$;

Il dispositivo decifra la chiave $(\rho \bigoplus c) \bigoplus \rho \bigoplus PRF(v, P) = SK_{device}$ avendo così il dispositivo autorizzato.

Attacchi

Con lo schema appena proposto ci siamo protetti da qualsiasi tipo di attacco eccetto uno (*device stolen and password known*).

- **Server cracked e password nota**:
  1. Se la chiave v nel dispositivo è ancora segreta allora non asi può ottenere SK poiché cifrata con PRF (v,P)
- **Device cracked/stolen**:
  1. L'attacker deve inviare un hash di una password valida;
  2. L'attacker può effettuare un dictionary attack online, ma è facilmente rilevabile e il MAC è verificato;
- **Device and Server Cracked**:
  1. Manca solamente la password;
  2. L'attacker può effettuare un dictionary attack online.

Per risolvere tutti i problemi fino ad ora proposti si utilizza il secret sharing nella seguente maniera.

4.5.3 Advanced Solution

Secret Sharing (2,2) for RSA

$n = p \cdot q$

$d \leftarrow \text{random,secretkey}$

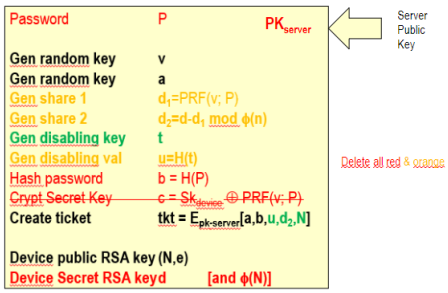
$d_1 \leftarrow \text{random,share1}$

$d_2 = d - d_1 \bmod \phi(n) \leftarrow \text{share2}$

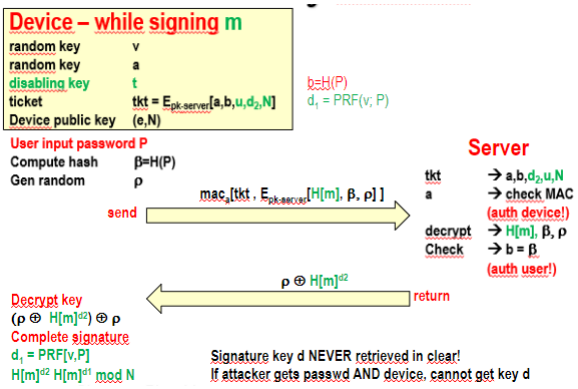
$$H(m)^{d_1} \cdot H(m)^{d_2} = H(m)^{d_1+d_2} = H(m)^{d_1+d-d_1} = H(m)^{d} \bmod n$$

Case (2,2): use trivial secret share
NO Shoup's problem to overcome in reconstruction (no Lagrange now!!)

Protocol Initialization



Key Retrieval



➔**Suffices to keep backup of**

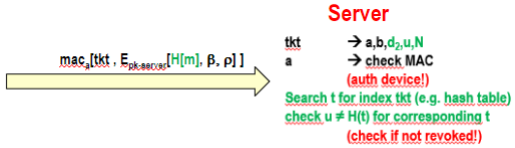
⇨t

⇨tkt

➔**If device stolen, send them to server**

⇨Server keeps blacklist

Key Disabling



Chapter 5

Linear Secret Sharing & Access Control Matrix

Vogliamo generalizzare il secret sharing in modo tale che, al posto di avere threshold, abbiamo una policy arbitraria.

5.1 Revisiting Shamir Scheme

- Segreto s;
- Share i:
$$y_i = s + a_1 x_1 + a_2 x_i^2 + \cdots + a_{t-2} x_i^{t-2} + a_{t-1} x_i^{t-1}$$
- In forma vettoriale: prodotto scalare
$$y_i = [1, x_i, x_i^2, \ldots, x_i^{t-2}, x_i^{t-1}] \bullet [s, a_1, a_2, \ldots, a_{t-2}, a_{t-1}]$$
- I coefficienti a_i sono casuali, rinominiamoli come:
$$y_i = [1, x_i, x_i^2, \ldots, x_i^{t-2}, x_i^{t-1}] \bullet [s, r_1, r_2, \ldots, r_{t-2}, r_{t-1}]$$

5.1.1 Shamir Scheme in Matrix Form

$$Ax = b$$

- A è una matrice $n \times t$ data;
- x è un vettore di dimensione t formato da $[secret, rand, rand], \ldots$;
- b è un vettore di dimensione n formato dagli shares risultanti.

Esempio 6. *Schema Shamir in forma matriciale* $3, 4$*:*

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Matrice ***Vandermonde***

Ricostruzione del segreto

- Sistema lineare t righe, schema per esempio (3,4):
$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_4 & x_4^2 \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_4 \end{bmatrix}$$
- L'incognita è il vettore:
$$\begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix}$$

Detto ciò, dato che è un sistema lineare, non abbiamo bisogno di utilizzare l'interpolazione di Lagrange ma possiamo semplicemente:

$$\begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_4 & x_4^2 \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \\ y_4 \end{bmatrix}$$

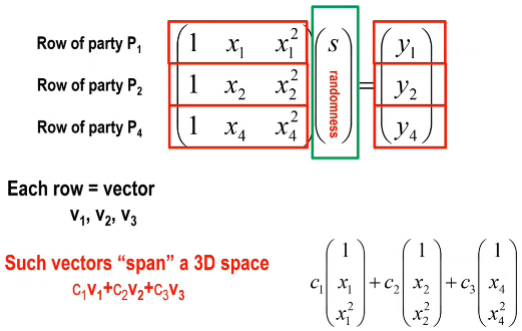
La cui inversa vale:

$$\begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_4 & x_4^2 \end{bmatrix}^{-1} = \begin{bmatrix} \frac{x_2x_4}{(x_1-x_2)(x_1-x_4)} & \frac{x_1x_4}{(x_2-x_1)(x_2-x_4)} & \frac{x_1x_2}{(x_4-x_1)(x_4-x_2)} \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

I cui elementi sono i coefficienti di Lagrange che, moltiplicati con il vettore b si ottiene il segreto:

$$s = y_1 \frac{x_2x_4}{(x_1-x_2)(x_1-x_4)} + y_2 \frac{x_1x_4}{(x_2-x_1)(x_2-x_4)} + y_3 \frac{x_1x_2}{(x_4-x_1)(x_4-x_2)}$$

Ricostruzione del segreto:alternativa



I vettori v_1, v_2, v_3 devono essere basi dello spazio vettoriale, se così non fosse non si potrebbe ricostruire il segreto poiché la loro combinazione lineare non fornisce tutto lo spazio.
Quindi, si può ricostruire il segreto se è possibile ricostruire il segreto se esiste una combinazione lineare tra i vettori v_1, v_2, v_3 pari a $(1, 0, 0)$:

$$c_1 \begin{bmatrix} 1 & x_1 & x_1^2 \end{bmatrix} + c_2 \begin{bmatrix} 1 & x_2 & x_2^2 \end{bmatrix} + c_3 \begin{bmatrix} 1 & x_4 & x_4^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

In forma matriciale:

$$\begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ x_1^2 & x_2^2 & x_3^2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Che la sua risoluzione ci porta a:

$$c_1 = \frac{x_2x_4}{(x_1-x_2)(x_1-x_4)}; \quad c_2 = \frac{x_1x_4}{(x_2-x_1)(x_2-x_4)}; \quad c_3 = \frac{x_1x_2}{(x_4-x_1)(x_4-x_2)};$$

Questi coefficienti rispettano la combinazione lineare e quindi:

$$\{c_1 \begin{bmatrix} 1 & x_1 & x_1^2 \end{bmatrix} + c_2 \begin{bmatrix} 1 & x_2 & x_2^2 \end{bmatrix} + c_3 \begin{bmatrix} 1 & x_4 & x_4^2 \end{bmatrix}\} \begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} = s$$

Quindi, siamo riusciti a trasformare il problema di shamir in un problema di span.

5.2 Generalizzazione

Nel **Linear Secret Sharing Scheme (LSSS)** è una generalizzazione dello span problem in cui la matrice A è arbitraria.

5.2.1 Trivial Secret Share is LSSS

Esempio 7. *Schema* $(3, 3)$

$$\begin{bmatrix} 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s \\ r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} s - r_1 - r_2 \\ r_1 \\ r_2 \end{bmatrix}$$

Esempio 8. *Schema* $(3,3)$ *Span program*

$$c_1 \begin{bmatrix} 1 & -1 & 0 \end{bmatrix} + c_2 \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} + c_3 \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$c_1 = c_2 = c_3 = 1$$

Ricostruzione del segreto:

$$c_1(s - r_1 - r_2) + c_2 r_1 + c_3 r_2 = (s - r_1 - r_2) + r_1 + r_2 = s$$

Theorem 5.2.1 (LSSS-homomorphic)**.** *Qualsiasi LSSS (Linear Secret Sharing Scheme) è* ***homomorphic****.*

Proof. Supponiamo di avere:

$$x_a = (s_a, r_{1a}, r_{2a}, \ldots) \quad y_a = (share_{1a}, share_{2a}, \ldots)$$

$$x_b = (s_b, r_{1b}, r_{2b}, \ldots) \quad y_b = (share_{1b}, share_{2b}, \ldots)$$

$$Ax_a = ya \quad Ax_b = y_b$$

$$y_a + y_b = Ax_a + Ax_b = A(x_a + x_b) = A(s_a + s_b, rand, rand, \ldots)$$

□

5.2.2 Monotone Span Programs

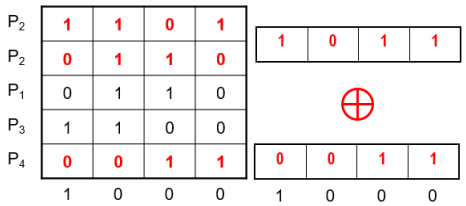
Le operazioni fatte fino ad ora, le operazioni sono legate al campo vettoriale in cui si opera.Proviamo ad eseguire uno Span Program in GF2 (si può avere o 0 o 1): il segreto è un bit e la sua probabilità di essere indovinato è del 50. Facciamo un esempio in GF2.

Esempio 9.

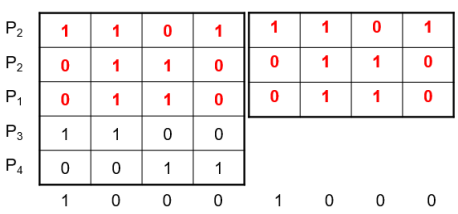
- *5 parties;*
- *Vettori 4-dimensionali;*
- *La seconda parte ha 2 vettori;*

P_2	1	0	1	1
P_2	0	1	1	0
P_1	0	1	1	0
P_3	0	0	1	1
P_4	1	1	0	0
	0	0	0	1

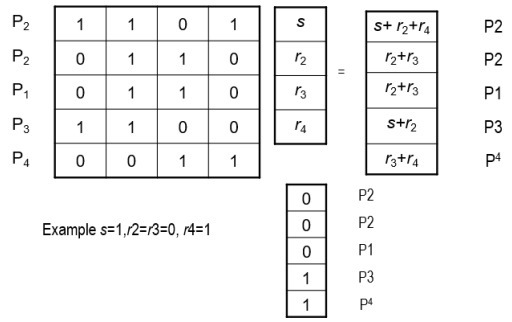
Vogliamo capire se una qualche combinazione lineare di questi vettori ci porta al valore $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$ *e quindi capace di farmi ricostruire il segreto. Quindi, il programma accetta un insieme* B *se e solo se lo span delle righe assegnate a* B *consentono di arrivare al vettore* $\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$ *. Supponiamo che* $\{P_2, P_4\}$ *si mettono insieme*



e sommo i 3 vettori, ottengo il vettore che cercavamo quindi posso ricostruire il segreto.
Supponiamo ora che si uniscano P_1, P_2, non è possibile ricostruire il segreto poiché lo span di questi vettori non ritorna il vettore $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$



Esempio 10. *Span Programs→Secret Sharing*



Importante. *Abbiamo effettuato la moltiplicazione fra le due matrici per ottenere il termine di destra.*

5.2.3 LSSS and Access Structure

Ogni struttura di accesso monotona può essere implementata con questo schema:

- Partecipanti $P = \{P_1, P_2, P_3, P_4\}$;
- Struttura monotona di accesso: $A \subseteq 2^P$;

Definizione 5 (Struttura di accesso monotona)**.** *Una* ***struttura di accesso monotona*** *è una struttura in cui non sono presenti negazioni:si ottiene la condizione verificata solo se non vi sono presenti negazioni nella policy implementata.*

Esempio 11.

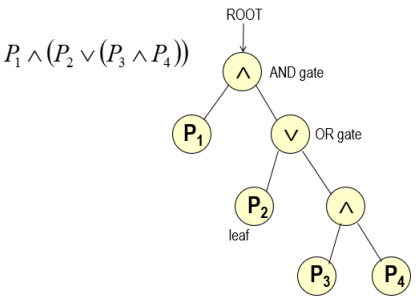
$$A = \{\{P_1, P_2\}, \{P_1, P_3, P_4\}\}$$

Con questa notazione, si accede al segreto solo se si coordinano $\{P_1, P_2\}$ *o* $\{P_1, P_3, P_4\}$ (Come si può vedere non vi sono negazioni)

Ogni predicato booleano (AND o OR) si può implementare un LSSS che supporta quel determinato predicato. Tuttavia, non vi è una regola precisa di implementare un predicato:in base a come viene scritta la logica si ha uno schema ideale o meno.

LSSS matrix frmo AC Predicate

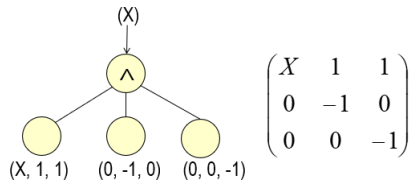
Vogliamo ottenere il segreto se la condizione $P_1 \wedge P_2 \vee (P_3 \wedge P_4)$ è verificata e lo vogliamo implementare secondo uno schema di Secret Sharing. Scriviamo la policy come un albero:



1. Radice;
2. Da sinistra a destra esegui:
   - Ogni parte P_i rappresenta la foglia dell'albero;
   - Gli operatori sono i nodi interni dell'albero;
   - Ogni operatore OR o AND collega la foglia all'albero;

Qual è la matrice che implementa questa policy? A tale scopo, procediamo preliminarmente ad analizzare le singole operazioni.

AND Gate Supponiamo che X sia il segreto che vogliamo dividere in 3 parti in modo tale che il segreto venga ricostruito solo se tutti e tre condividono il proprio share: (3,3) Trivial Secret Scheme.



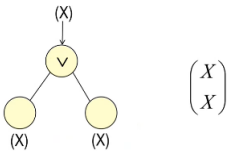
Importante. *La matrice* $\begin{bmatrix} X & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$ *viene ottenuta tenendo conto che le righe sono le parties e che lo span (combinazione lineare) delle righe debba ritornare* $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$ *(X,0,0). Quindi era valida anche la seguente matrice:*

$$\begin{bmatrix} X & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{\textit{Relativi Pesi:}} \quad c_1 = 1, c_2 = 1, c_3 = -1$$

Infatti:

$$c_1P_1 + c_2P_2 + c_3P_3 = 1\begin{bmatrix}X & -1 & 1\end{bmatrix} + 1\begin{bmatrix}0 & 1 & 0\end{bmatrix} - 1\begin{bmatrix}0 & 0 & 1\end{bmatrix} = \begin{bmatrix}X & 0 & 0\end{bmatrix}$$

OR Gate Sia X il segreto che stiamo condividendo a due parties.La condizione OR ci dice che per ricostruire il segreto, deve essere presente una delle due parti.
Quindi possiamo modellare la matrice come $\begin{bmatrix}X\\X\end{bmatrix}$.



Ricostruzione della matrice La condizione era $P_1 \wedge P_2 \vee (P_3 \wedge P_4)$.
Quello che manca ora è di riassumere tutti i vettori che sono stati assegnati durante la ricostruzione della matrice:

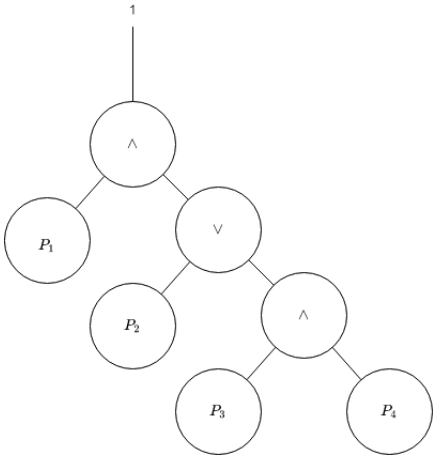


Figure 5.1: Fase 1:assumiamo il segreto di una dimensione e quest'ultimo deve essere diviso in due per essere ricostruito.

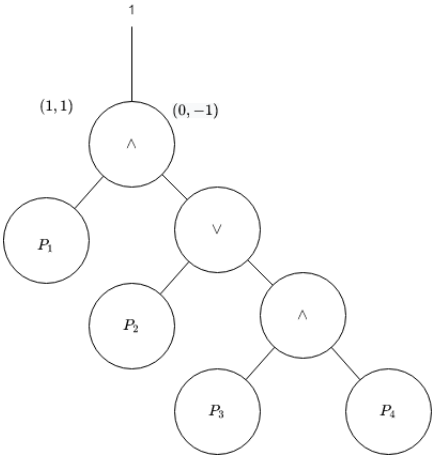


Figure 5.2: Fase 2: Calcolo gli shares necessari e li spartisco nei rami $P_1 \to (1,1)$ e $\wedge \to (0,-1)$

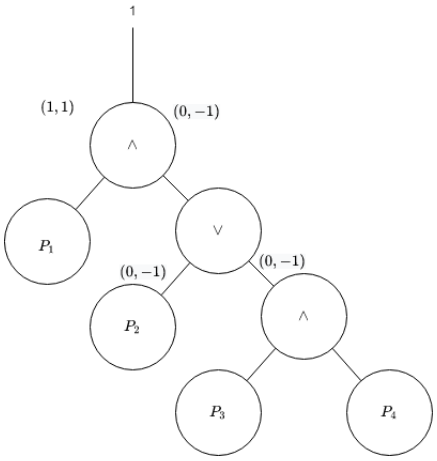


Figure 5.3: Fase 3: dato che non so se ci sono delle parti o meno prendo lo share e lo copio in entrambi i rami **poiché uno dei due deve conoscere il segreto per risalire l'albero**.

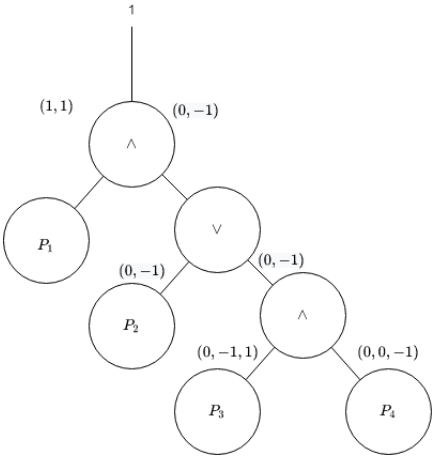


Figure 5.4: Fase 4: Mi trovo di fronte ad un AND e quindi il segreto deve essere conosciuto da entrambi. Aumentiamo la dimensione del vettore e generiamo due valori consoni per ottenere il valore del padre cioè $(0,-1)$, come nella fase 1 si è scelto 1 e -1

$$\begin{aligned} P_1 &: \begin{bmatrix}1 & 1 & 0\end{bmatrix} \\ P_2 &: \begin{bmatrix}0 & -1 & 0\end{bmatrix} \\ P_3 &: \begin{bmatrix}0 & -1 & 1\end{bmatrix} \\ P_4 &: \begin{bmatrix}0 & 0 & -1\end{bmatrix} \end{aligned}$$

Per effettuare le verifiche si pone ad uno la/le parties che dovrebbero contribuire a ricostruire il segreto, si effettuano le somme e non si deve ottenere il vettore $\begin{bmatrix}1 & 0 & 0\end{bmatrix}$.

Esempio 12 (Esercizio). *Verifica che la policy proposta rispetta la condizione data:*

$$(A \wedge C \wedge D) \vee (B \wedge C)$$

$$\begin{pmatrix} A & 1 & -1 & -1 & . \\ C & 0 & 1 & 0 & . \\ D & 0 & 0 & 1 & . \\ B & 1 & . & . & -1 \\ C & 0 & . & . & 1 \end{pmatrix}$$

5.3 Conclusione

- **LSSS/MSP: generalization to arbitrary access structures**
  - ⇨Must be monotone
    - →If parties A+B may access, also A+B+C may
    - →Cannot model policies such as A+B+NOT(C)
      - » Issue when revocation needed
- **Sub-optimal**
  - ⇨Parties may need more than 1 share
  - ⇨Minimum overhead: open research issue
- **Improved constructions**
  - ⇨Explicit threshold gates [Liu,Cao, 2010], but prime fields…
- **Applications**
  - ⇨Dramatic! Most modern crypto
    - →E.g. attribute based cryptography

Chapter 6

Elliptic Curve Crypto

6.1 Introduction

La public key cryptography si basa sul fatto di rendere una operazione facile in un verso, ma difficile nell'altro. Questo concetto viene detto **computational security**.
Le curve ellittiche servono ad ottenere computational security anche in assenza di operazioni con modulo:invece del modulo si utilizza il DLog. La complessità per invertire l'operazione del DLOG in $Z^*(p)$ è $\exp(O^{\frac{1}{3}})$; nel caso in cui si cambia il gruppo in $(Z(p), +)$ è polinomiale infatti:

- $Z(p)$ è l'insieme dei numeri $\{0, \ldots, p-1\}$;
- $+$ è la somma $\mod p$ i.e.$3+9 \mod 11 = 1$;
- è un gruppo abeliano poiché soddisfa tutte le proprietà dei gruppi e quella di essere abeliano;
- Supponiamo che la moltiplicazione venga espressa come somma di k volte e che ci venga dato il risultato i.e.$k \cdot 3 \mod p = result$;
- Il corrispondente **Discrete Log Problem (DLOG)** consiste nel trovare k tale che $k \cdot 3 \mod p = result$; *(nel caso si abbia una moltiplicazione si ha g^x)*;
- Quindi questo problema è risolvibile polinomialmente tramite un algoritmo.

Dato che questo in questo gruppo, il problema è risolvibile polinomialmente e quindi in maniera facile non è adatto nella crittografia. Tuttavia, si introduce un problema più difficile nei **gruppi delle curve ellittiche** con difficoltà computazionale $\exp(O^{\frac{1}{2}})$ (*Polland Rho*). La vera convenienza delle curve ellittiche è per la sua **scalabilità**.

6.2 Elliptic Curves

Definizione 6 (Curve Ellittiche)**.** *Una **curve ellittica** è una particolare curva cubica di terzo ordine che, nella sua forma più generale (*Weierstrass Expression*) ha questa struttura:*

$$y^2 = x^3 + ax + b \quad where \quad 4a^3 + 27b^2 \neq 0$$

Per avere un gruppo dobbiamo avere un insieme di curve e una operazione che è chiusa rispetto all'insieme. Questa operazione è $+$ o anche detta $\circ$: tale che $P + Q = R$ con R appartenente alla curva.

6.3 Elliptic point addition $P + Q$

Per definire l'operazione di addizione si seguono i seguenti passi: Per verificare la validità di questa operazione dobbiamo

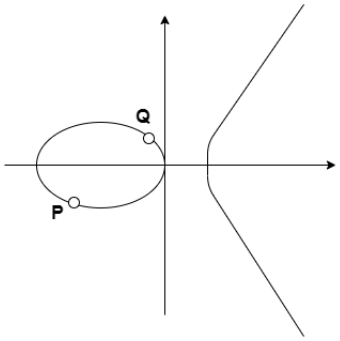


Figure 6.1: Prendiamo due punti (per ora distinti appartenenti alla curva ellittica)

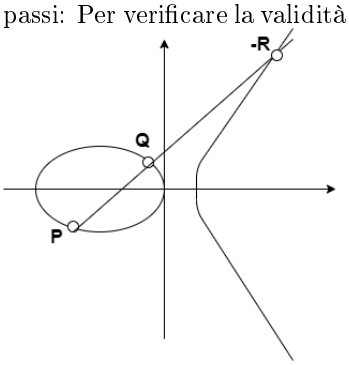


Figure 6.2: Tracciamo una retta passanti per i due punti e, dato che la curva è cubica, la retta intersecherà la curva ellittica in un altro punto chiamato -R

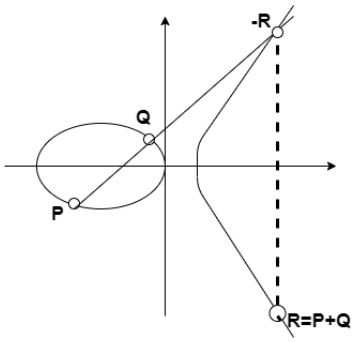


Figure 6.3: Tracciamo il negativo di R che intersecherà la curva ellittica in R. Questo punto lo definiamo **R**=**P**+**Q**

verificare le proprietà di gruppo:

- **Chiusura**:presi due elementi 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 $g \circ g^{-1} = I$;
- **Associativa**:per qualsiasi g_1, g_2, g_3 deve valere $(g_1 \circ g_2) \circ g_3 = g_1 \circ (\circ g_3)$

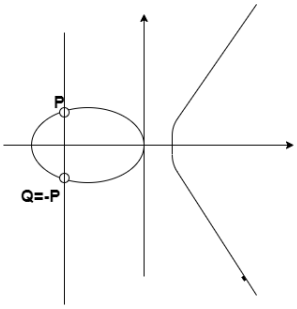
6.3.1 Elliptic Curves: chiusura

La chiusura dello operazione rispetto all'insieme non è sempre definita poiché, se prendessimo 2 punti posti verticalmente e li sommiamo come prima, la retta non interseca la curva ellittica. Per ovviare a questo problema introduciamo un punto aggiuntivo detto **O**=-**O** che è posto all'infinito.

Importante. *Se vogliamo effettuare P+P, introduciamo la tangente alla curva e la sua prossima intersezione la consideriamo come -R.*

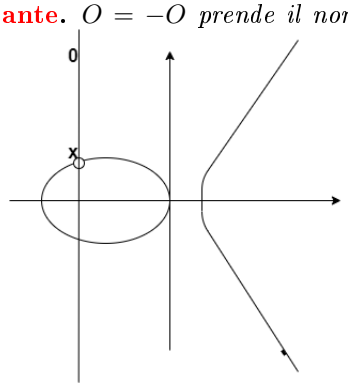
6.3.2 Elliptic Curves: inversa

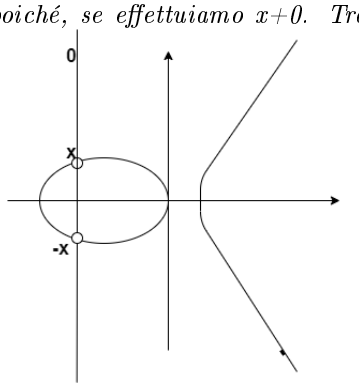
Supponiamo di voler sommare P e -P, il risultato sarà il punto all'infinito $O = -O$ poiché non si interseca la curva. Dato che $P - P = 0$, possiamo considerarla come l'inversa di P. Essa viene segnata con il -.



Importante. *L'inversa viene segnata con il $-P$ al posto di P^{-1} poiché nel secondo caso corrisponderebbe all'inversa del gruppo moltiplicativo e non additivo.*

Importante. *$O = -O$ prende il nome di 0 (zero) poiché, se effettuiamo $x+0$. Tracciamo la retta che interseca la curva, segniamo il punto (*in questo caso coincide con x*) e ne prendiamo il suo opposto $-x$.*

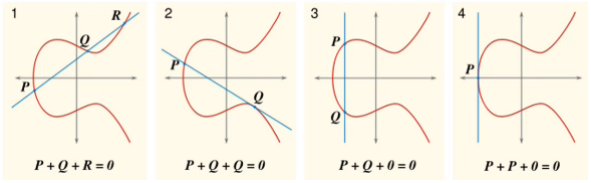




6.3.3 Elliptic Curves:associatività,identità,commutativa

Queste 3 proprietà sono banalmente verificate.

Theorem 6.3.1. *Se 3 punti di una curva ellittica appartengono alla stessa linea, allora la loro somma è zero.*

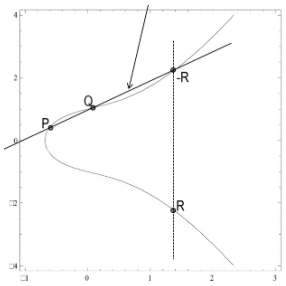


6.3.4 Elliptic Curves:Procedura

Un'altra alternativa di effettuare le operazioni con le curve ellittiche è quella di procedere per via analitica in questo modo:

$$
\begin{aligned}
P &= (x_1, y_1) \\
Q &= (x_2, y_2) \\
R &= P + Q = (x_3, y_3) \\
x_3 &= \lambda^2 - x_1 - x_2 \\
y_3 &= \lambda(x_1 - x_3) - y_1 \\
\lambda &= \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & P = Q \end{cases}
\end{aligned}
$$

Proof. Perché $P \neq Q$.
La retta passante tra P e Q è:



$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) = \lambda(x - x_1) + y_1$$

Sostituiamo l'equazione della retta nell'equazione della curva ellittica.

$$y^2 = x^3 + ax + b$$
$$(\lambda(x - x_1) + y_1)^2 = x^3 + ax + b$$

λ, x_1, y_1 sono noti, mentre x è l'incognita. Non vi è una procedura per risolvere in maniera generica una equazione d terzo grado, ma possiamo ricondurci alle soluzioni poiché sappiamo che i punti della curva sono P e Q. Quindi la soluzione generale è nella forma:

$$(x - x_1)(x - x_2)(x - x_3) = x^3 - (x_1 + x_2 + x_3)x^2 + \cdots$$

La cui incognita è x^3 e il termine $(x_1 + x_2 + x_3)$ che è pari a λ (noto) ci porta a risolvere:

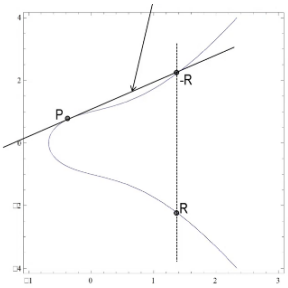
$$\lambda^2 = x_1 + x_2 + x_3 \rightarrow x_3 = \lambda^2 - x_1 - x_2$$

Per quanto riguarda la y_3, viene ottenuta dall'equazione della curva:

$$y_3 = \lambda(x_1 - x_3) - y_1$$

□

Proof. Perché $P = Q$.
Nell'equazione della retta non vi è più il coefficiente angolare, ma la tangente:



$$y = y_1 + y_1'(x - x_1) + y_1$$
$$2y\frac{dy}{dx} = 3x^2 + a \rightarrow \frac{dy}{dx} = \frac{3x^2 + a}{27}$$
$$\lambda = y_1' = \frac{3x_1^2 + a}{2y_1}$$

Il resto prosegue come nel caso $P \neq Q$. □

6.4 Elliptic Curve Group

6.4.1 Elliptic Curves Modular Integers

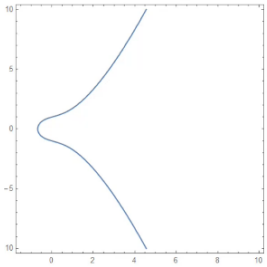
Restringiamo l'uso delle curve ellittiche non a numeri reali, ma fra interi in modulo.

Definizione 7. *Una **elliptic curves modular integers** è una curva ellittica $E_p(x, y)$ che soddisfa le seguenti condizioni:*

- *x, y siano numeri interi;*
- $y^2 \mod p = x^3 + ax + b \mod p$.

Dato che questo campo è finito, allora il numero di punti della curva è finito.

Esempio 13. *Curva ellittica* $x^3 + x + 1 \mod 5$
Le condizioni da soddisfare sono le seguenti e devono essere soddisfatte da tutti i punti:tutti i punti (i, j)*tali che*



- $(i, j) \in [0, 4]$*;*
- $j^2 = i^3 + i + 1 \mod 5$*;*
- *Punto all'infinito*

Quindi:

$$E(Z_5) = \{O, (0,1), (0,4), (2,1), (2,4), (3,1), (3,4), (4,2), (4,3)\}$$

Supponiamo di prendere un punto come punto iniziale:

$P = (0, 1)$
$P + P = (0, 1) + (0, 1)$
Passo 1: calcolare λ
$\lambda = \frac{3x_1^2 + a}{2y_1} = 1 \cdot 2^{-1} = 1 \cdot 3 = 3 \mod 5$
$2^{-1} \mod 5 = 2x = 1 \mod 5 = 3$
Passo 2: calcolare x_3
$x_3 = \lambda^2 - x_1 - x_1 = 3^2 = 4 \mod 5$
Passo 3:calcolare y_3
$y_3 = \lambda(x_1 - x_3) - y_1 = 3(0 - 4) - 1 = 2 \mod 5$
Risultato:$(4, 2)$

Iterando:

$$\begin{aligned} P &= (0,1) \\ 2P &= (0,1) + (0,1) = (4,2) \\ 3P &= (4,2) + (0,1) = (2,1) \\ 4P &= (2,1) + (0,1) = (3,4) \\ 5P &= (3,4) + (0,1) = (3,1) \\ 6P &= (3,1) + (0,1) = (2,4) \\ 7P &= (2,4) + (0,1) = (4,3) \\ 8P &= (4,3) + (0,1) = (0,4) \\ 9P &= (0,4) + (0,1) = O \end{aligned}$$

Notiamo che partendo da P, il gruppo è ciclico.

6.4.2 Elliptic Curves Group

In conclusione, $(E(Z_p), +)$ è un gruppo per cui:

1. L'addizione è chiusa su un insieme E definita dalla curva ellittica;
2. L'addizione è commutativa;
3. O è l'identità;
4. Ogni punto $P \in E$ possiede l'inversa $-P$;
5. Vale la proprietà associativa;
6. è un gruppo abeliano;

Proprietà Elliptic curver group in crypto

Sia P un punto della curva ellittica appartenente al gruppo Z_p e k intero mod group order:

- Dato P e k è facile calcolare $k \cdot P$;

- Dato P e kP è difficile calcolare k;
- Questo problema è equivalente al problema del DLog inZ_p:se abbiamo la notazione moltiplicativa $k \to P^k$.
- In criptografia si ha una forte dipendenza dalla curva scelta: si estendono questi discorsi oltre che in Z_p, ma anche nel campo di Galua.

Goal: compute [1437]P

Express in bits

⇨1437_{10}=10110011101_2

Initialize first line

⇨Differs if 0 or 1

Start from 2° lsb to msb

⇨For every bit

→Double;

→If 1: add to result

Complexity:

⇨O(nbit) x2

⇨O(nbit/2) additions

lsb−>msb	*double*	*result*
1	$1P$	$+1P = P$
0	$2P$	.
1	$4P$	$+4P = 5P$
1	$8P$	$+8P = 13P$
1	$16P$	$+16P = 29P$
0	$32P$	.
0	$64P$	.
1	$128P$	$+128P = 157P$
1	$256P$	$+256P = 413P$
0	$512P$	.
1	$1024P$	$+1024P = 1437P$

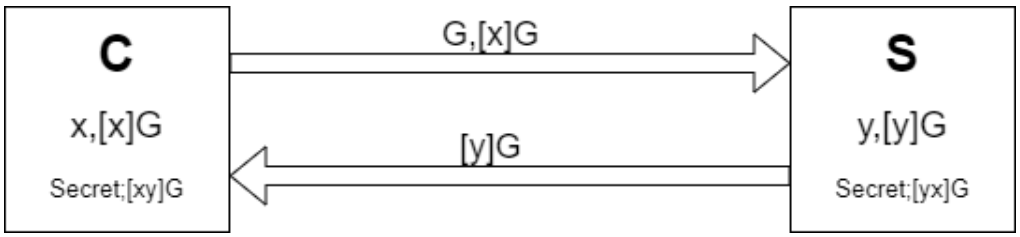
Chapter 7

ECDH/ECDSA

In qualsiasi problema crittografico, si calcola $g^x \mod p$ che nelle curve ellittiche corrisponde a moltiplicare x volte g punto appartenente al gruppo della curva.

7.1 ECDH:Elliptic Curves Diffie-Hellman

Lo scambio chiavi di Diffie-Hellman con le curve ellittiche funziona esattamente come Diffie-Hellman generale, ma con la differenza che si utilizzare il gruppo delle curve ellittiche. Quindi ECDH è un protocollo di accordo delle chiavi.



1. C genera x casuale e calcola $[x]G$ in cui G è una curva ellittica;
2. Invia $G, [x]G$;
3. S genera y casuale e calcola $[y]G$, per la proprietà di gruppo (additiva) può calcolare il segreto $[yx]G$ tramite la conoscenza di y;
4. Invia $[y]G$;
5. C calcola il segreto $[xy]G$;

7.2 ECDSA

ECDSA è un protocollo per la firma digitale a modulo di un numero primo da 256 e 384 bit basato sul DLog.

7.2.1 DSA

Nella versione standard di **DSA** si utilizzano le operazioni modulari in $Z*(p), p > 2$ numero primo e q strong prime cioè: se p è primo $p = 2q + 1$. Inoltre, lo span di $g^x \mod p$ comprende tutti gli elementi del gruppo.

Importante. *Chiave privata d, chiave pubblica* $y = g^d \mod p$

Quando si firma, dobbiamo effettuare le seguenti operazioni:

- Generare un valore casuale $k \in (1, q)$;
- Calcolare $r = (g^k \mod p) \mod q$;
- Calcolare $s = k^{-1}(H((m) + dr)) \mod q$;
- la firma:(r, s);

Per verificare la firma, occorre:

- Calcolare $u_1 = s^{-1}H(m) \mod q$;
- Calcolare $u_2 = s^{-1}r \mod q$;
- Verificare $((g^{u_1}y^{u_2}) \mod p) \mod q = r$
- $g^{u_1}y^{u_2} = g^{u_1}g^{d \cdot u_2} = g^k = M$
- $u_1 + d \cdot u_2 = \frac{H(m)k}{H(m)+rd} + \frac{d \cdot r \cdot k}{H(m)+rd} = k$

Problematiche

Nel caso in cui k viene predetto la costruzione porta a dei rischi.Infatti, un attacco che è possibile fare è dovuto all'errore di ottimizzazione nel ricalcolo della curva ellittica in ogni firma:k viene predetto o riutilizzato. Quindi, k non deve essere predicibile poiché, altrimenti, si può firmare un qualsiasi messaggio casuale, si può rubare la chiave privata e quindi perdere la propria identità digitale.In particolare:

Algorithm 1 K predicted

Ensure: $(r, s),\ k$
$s = k^{-1}[H(m) + rd] \mod n$
$sk = H(m) + rd \mod n$
$[sk - H(m)]r^{-1} = d \mod n$
d viene calcolato

Invece, nel caso in cui si ripete k:

$$\text{from } (r,s_1),(r,s_2) \text{ - same unknown } k = \text{same } r$$

$$\begin{cases} s_1 = k^{-1}\left[H(m_1) + rd\right] \\ s_2 = k^{-1}\left[H(m_2) + rd\right] \end{cases} \mod n \quad \Rightarrow \quad \begin{cases} k = s_1^{-1}\left[H(m_1) + rd\right] \\ k = s_2^{-1}\left[H(m_2) + rd\right] \end{cases} \mod n$$

$$s_1^{-1}\left[H(m_1) + rd\right] = s_2^{-1}\left[H(m_2) + rd\right]$$

$$s_2 H(m_1) + s_2 rd = s_1 H(m_2) + s_1 rd$$

$$d(s_2 - s_1)r = s_1 H(m_2) - s_2 H(m_1)$$

$$d = \left[s_1 H(m_2) - s_2 H(m_1)\right]r^{-1}(s_2 - s_1)^{-1} \mod n$$

$$d \text{ readily computed!!}$$

Figure 7.1: K riutilizzato

7.2.2 ECDSA:setup

In ECDSA, dati:

- $E(Z_p)$ curva ellittica con p numero primo;
- n differente da p e numero grande;
- P generatore del gruppo;
- $Q = [d]P$:
  1. d è un intero casuale tra $[1, n-1]$ **chiave privata**;
  2. $Q = dP$ punto sulla curva *chiavepubblica*.

7.2.3 ECDSA:signature generation

Algorithm 2 Signature generation

Seleziona casualmente un intero $k \in [1, n-1]$:

- unico per ogni firma;
- non predicibile;

Calcola il puntto sulla curva ellittica $kP = (x_{1,y_1})$:

- x_1 è un intero $\mod P$

Calcola $r = x_1 \mod n$ è un numero e non un punto della curva ellittica
Calcola $k^{-1} \mod n$
Effettua l'hash del messaggio i.e. SHA
Firma:$(r, s) \quad s = k^{-1}(H(m) + dr)$

7.2.4 ECDSA:verification

Algorithm 3 Signature verification

Ensure: Ricevo m (r, s) con $s = k^{-1}(H(m) + dr)$
Prendo il messaggio m e calcolo $H(m)$
Prendo $s = \frac{H(m)+rd}{k}$ e calcolo $w = s^{-1} \mod n = \frac{k}{H(m)+rd}$
Calcola:

- $u_1 = H(m)w \mod n$
- $u_2 = rw$

Calcola il punto della curva ellttica:

- $u_1P + u_2Q = (x_0, y_0)$
- è possibile grazie alla proprietà homomorphic:

$$u_1P + u_2Q = u_1P + u_2dP = (u_1 + u_2d)P = \frac{H(m)k}{H(m) + rd} + \frac{rkd}{H(m) + rd} = \frac{k(H(m) + rd)}{H(m) + rd} = kP$$

Calcola $v = x_0 \mod n$
verifica che $v = r$

7.2.5 Considerazioni

L'operazione più costosa rispetto a quella modulare è quella fatta fra le curve ellittiche poiché dipende da quante operazioni si devono effettuare sulla curva ellittica. Infatti:

- una per generare la chiave pubblica Q;
- calcolare kP;

Quindi per questo si è tentati di riutilizzare la stessa k perché, così facendo, si abbassa il costo computazionale dovuto all'utilizzo delle curve ellittiche.

Chapter 8

Bilinear Maps

Il **bilinear maps** viene utilizzato nelle tecniche di crittografia dette **pairing-based crypto**: si hanno molti gruppi i cui punti vengono trasformati in altri gruppi.

8.1 Definizioni

Definizione 8 (Bilinear Map)**.** *Siano G_1, G_2, G_t gruppi ciclici dello stesso gruppo. Una **mappa bilineare** da $G_1 \times G_2 \to G_t$ è una funzione $e : G_1 \times G_2 \to G_t$ tale che per tutti $u \in G_1,\ v \in G_2,\ a, b \in \mathbb{Z}$ valga:*

$$e(u^a, v^b) = e(u, v)^{ab}$$

Importante. *Vengono dette **pairings** perché associano coppie di elementi di G_1, G_2 con gli elementi di G_t.*

Definizione 9 (Admissible Bilinear Map)**.** *Sia $e : G_1 \times G_2 \to G_t$ e g_1, g_2 generatori di G_1, G_2 rispettivamente. La mappa e è una **mappa bilineare ammissibile** se $e(g_1, g_2)$ genera G_t e e è efficientemente calcolabile.*

8.2 Relazione tra G_1, G_2, G_t

- G_1, G_2, G_t sono tutti gruppi isomorphic di un altro gruppo dato che hanno in comune lo stesso gruppo ciclico;
- Sono differenti poiché i loro elementi vengono calcolati in maniera differenti;
- Di solito, $G_1 = G_2$;
- etc. . . ;

8.3 Decisional Diffie-Hellman

Definizione 10 (Decisional Diffie-Hellman)**.** *Dati g, g^a, g^b e $\alpha = \{g^{ab}, g^{random}\}$. Sapendo che è possibile conoscere il valore α con un coin flip (una tra le due alternative), è possibile distinguere quale sia il valore? (il vero valore di DH o quello casuale)*
Un gruppo soddisfa il Decisional Diffie-Hellman DDH se non è possibile distinguere tra il numero casuale e il vero coefficiente di Diffie-Hellman.

Se siamo in grado di calcolare Diffie-Hellman, allora non si soddisfa DDH e DH ma non vale il contrario.
Questo problema è differente da quello computazionale di DH.A tale scopo, supponiamo che stiamo affrontando questo problema con ECDDH:

- Dati G, G^a, G^b con bilinear pairing:
$$e(g^a, g^b) = e(g, g)^{ab}$$
(in modulo);
- Obiettivo: rompere DDH

Il pairing non ci consente, di norma, di calcolare il termine g^{ab} poiché è una operazione della curva ellittica, ma trasforma i punti in altri. Quindi, non si rompe Diffie-Hellman.
Supponiamo di avere g, g^a, g^b (*Punti della curva ellittica*) e chiamiamo il generatore del gruppo $g_t = e(g, g)$ (*Punti in*

Z_p). Ora, se prendo due punti g^a, b^b e applico pairing ottengo $e(g^a, g^b) = g_t^{ab}$ ma non ho rotto DH dato che non è un punto sulla curva ellittica. Tuttavia, posso prende il termine α e considerare questo scenario:

$$e(g, \alpha) = \begin{cases} e(g^1, g^{ab}) = g_t^{1 \cdot ab} \\ e(g, g^Z) = g_t^z \neq g^{ab} \end{cases}$$

Assunto che precedentemente il valore g_t^{ab}, se il valore α da noi scelto ci porta allo stesso risultato allora era il coefficiente di Diffie-Hellman.

8.4 Discrete Log

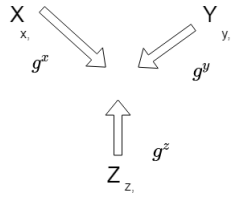
Theorem 8.4.1. *Se esiste una mappa bilineare $e : G \times G \to G_t$, allora il problema DLOG in G non è più difficile del DLOG in G_t*

8.5 Group Diffie-Hellman

Vogliamo provare applicare Diffie-Hellman per key agreement con più parties. Di base ciò non è possibile a meno che non si apportino delle modifiche ed occorre scambiare più di un messaggio fra le parties.

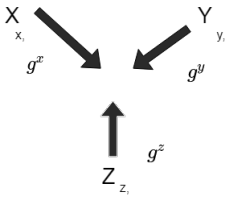
8.5.1 Joux Scheme

Si applica il pairing per effettuare uno schema Diffie-Hellman con più di due parties.
Dati X, Y, Z parti con relativi valori x, y, z e concessa la possibilità di utilizzare pairings $e(g, g) = g_t$ vogliamo effettuare un key agreement.



(a) Fase 1: si X, Y, Z generano e pubblicano i valorig^x, g^y, g^z

.



(b) Fase 2: per esempio Z riceve g^x, g^y, potendo così calcolare g^{xz}, g^{yz}
ma non può ancora calcolare g^{xzy} (Diffie-Hellman), ma è come se stessimo affrontando un problema di Secret Sharing: posso applicare il pairing.
Ottenendo:$e(g^{xz,g^y}) = e(g, g)^{xyz}$. Questo rappresenta il segreto condiviso.

Importante. *Questo procedimento funziona solamente a 3. Se lo dovessimo fare anche per uno schema a 4 parties non funziona per lo stesso problema che si aveva nello schema a 2.*

8.6 Identity-based Encryption

Nell'**identity-based encryption** vogliamo uno schema crittografico a chiave pubblica in cui la chiave pubblica **PK** è una stringa leggibile (per esempio nome utente).

8.6.1 Schema Boneh e Franklin: Identity Based Encryption

- Sia G un gruppo di ordine q primo, $e : G \times G \to G_t$ una mappa bilineare, e g un generatore di G;
- Sia $\hat{g} = e(g, g) \in G_t$;
- Siano $h_1 : 0, 1* \to G$ e $h_2 : G_t \to 0, 1*$ hash functions: la prima è una hash function che prende una stringa di bit e ne effettua l'hash su un punto di una curva ellittica;la seconda funzione hash prende un punto su una curva ellittica e ne effettua l'hash in una stringa di bit.

Importante. *Questi parametri sono tutti pubblici.*

8.6.2 Scheme

Supponiamo che Alice voglia mandare dei dati cifrati a BOB, BOB riceve il messaggio e la sua public key è il suo nome $PK_{BOB} = BOB$ e un PKG (public key generator) s chiave privata e g^s chiave pubblica.

PKG
(Private Key Generator)
$(SK_{PKG} = s, PK_{PKG} = g^s)$

Scheme:Encryption

Se alice vuole Inviare un messaggio m a BOB, genera un numero casuale $r \leftarrow \mathrm{R}Z_q$ e calcola:

Alice
(Invia dati cifrati a BOB)
(r, g^r)

BOB
(riceve i dati)
PK_{BOB}="BOB"
(t, g_t)

$$\begin{aligned} ENC(g, g^s, BOB, m) &= (g^r, m \bigoplus h_2(e(h_1(BOB), g^s)^r)) \\ &= (g^r, m \bigoplus h_2(e(h_1(BOB), g)^{rs})) \end{aligned}$$

PKG, genera la chiave privata di BOB come:

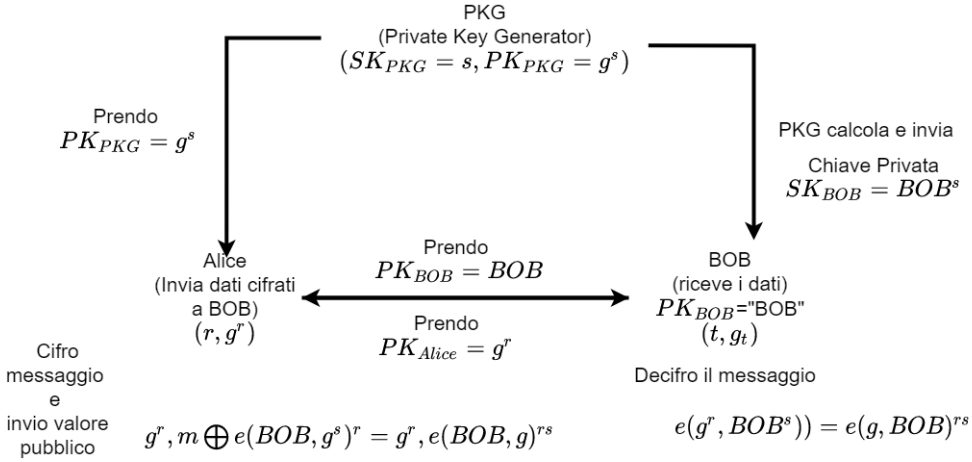
$$MakeKey(s, BOB) = h_1(BOB)$$

Scheme:Decryption

Dato un messaggio cifrato $(u, v) = (g^r, m \bigoplus h_2(e(h_1(BOB), g))^{rs})$ e la chiave privata $w = h_1(BOB)$, BOB decifra come segue:

$$\begin{aligned} DEC(u, v, w) &= v \bigoplus h_2(e(w, u)) \\ &= m \bigoplus h_2(e(h_1(BOB), g))^{rs} \bigoplus h_s(e(h_1(BOB)^s, g^r)) \\ &= m \bigoplus h_2(e(h_1(BOB), g)^{*rs}) \bigoplus h_2(e(h_1(BOB), g)^{rs}) \\ &= m \end{aligned}$$

Summary



8.6.3 Considerazioni

Con questa schema ci siamo tolte tutte le problematiche dovute alla presenza dei Certification Authority, ma con l'impiego dei PKG stiamo dando molto potere a una terza parte che ha il compito di generare (e non più solamente di verificare) la chiave privata del destinatario. PKG potrebbe prendere l'identità di BOB.
TUttavia per ovviare a questo problema si utilizzano più PKG che condividono lo stesso segreto s:distributed PKGs. Questo sistema è più semplice da gestire e da setuppare e quindi è più conveniente.

Esempio 14 (Esercizio)**.** *Spiegare come si può implementare un Distributed PKG*