SAPIENZA
UNIVERSITÀ DI ROMA

Condivisione di Segreti Dinamica: Un'implementazione

Facoltà di Ingegneria dell'informazione, Informatica e Statistica

Corso di Laurea in Informatica

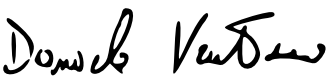
Candidato

Leonardo Danella

Matricola 1885686

Relatore

Prof. Venturi Daniele



Anno Accademico 2021/2022

Tesi non ancora discussa

Condivisione di Segreti Dinamica: Un'implementazione
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Email dell'autore: danella.1885686@studenti.uniroma1.com

Sommario

Gli schemi di condivisione di un segreto con soglia k (threshold) permettono ad un entità, che chiameremo *dealer* di condividere un segreto s tra n parti in modo tale che solo un sottoinsieme di cardinalità almeno $k = k(n)$ possa ricostruire il segreto.

Komargodski, Naor e Yogev hanno proposto uno schema efficiente per condividere un segreto tra un numero illimitato di parti in modo che solo sottoinsiemi di k parti possano recuperare il segreto, dove k è qualsiasi costante fissa. Questa struttura di accesso è nota come k-threshold.

Successivamente, Komargodski, Paskin-Cherniavsky hanno risolto due dei problemi lasciati aperti nella pubblicazione precedente, in particolare: la possibilità di avere uno schema efficiente per la struttura di accesso a soglia dinamica, in cui gli insiemi qualificati sono di dimensioni crescenti all'aumentare del numero di parti, e come traslare un qualsiasi schema con un k-threshold in uno schema che sia robusto.

Indice

Capitolo 1

Introduzione

Gli schemi a condivisione di segreto sono dei metodi che permettono ad un dealer, che mantiene un'informazione segreta, di distribuire quest'ultima tra n parti in modo che dei loro sottoinsiemi qualificati di cardinalità predefinita possano ricostruire il segreto, mentre altri non possano scoprire nulla. Questa collezione di sottoinsiemi qualificati è chiamata **struttura d'accesso**.
Questa tipologia di schemi è stata scoperta ed introdotta per la prima volta nel 1979, contemporaneamente e indipendentemente, da Shamir [Sha79] e da Blakley [Bla79], trattando le strutture d'accesso con una soglia (threshold) di k su n, con $1 \leq k \leq n$. Con struttura d'accesso a soglia di k su n si intende che per ricostruire un messaggio suddiviso in n parti, è sufficiente avere almeno k di queste.
Le loro costruzioni sono efficienti sia per quanto riguarda la grandezza dei messaggi da mandare alle n parti, sia per quanto riguarda la complessità computazionale negli algoritmi di condivisione e di ricostruzione. Inoltre grazie allo studio di Ito, Saito, e Nishizeki [ISN93] sappiamo che per ogni struttura d'accesso monotona esiste uno schema di condivisione del segreto. Nel loro schema la grandezza dei messaggi è proporzionale al secondo livello di complessità della profondità della struttura d'accesso quando vista come una funzione booleana (e quindi le dimensioni sono esponenziali per la maggioranza delle strutture). Benaloh e Leichter [BL88] hanno proposto uno schema con grandezza polinomiale rispetto alla complessità della formula monotona della struttura d'accesso. Karchmer e Wigderson [KW93] hanno generalizzato questa costruzione in modo tale che la grandezza sia polinomiale nella complessità monotona dello span del programma.
La limitazione di tutti questi schemi è il fatto che sia necessario conoscere in anticipo un limite superiore sul numero di participanti. Nella maggioranza dei casi ciò può causare problemi, poiché non sempre è un dato conosciuto, oppure anche avendolo si possono avere conseguenze disastrose. Inoltre va considerato anche che se si dovesse avere un upper-bound n, dovremmo comunque avere delle dimensioni per gli shares il più piccoli possibile, poiché potrebbe esserci la possibilità che il numero di partecipanti sia di molto minore al limite superiore di n. Inoltre, la struttura d'accesso potrebbe cambiare nel tempo, costringendo il dealer a ricondividere il segreto.
Considereremo quindi il caso in cui il numero di partecipanti non sia conosciuto, cercando di dare a chi arriva come t-esimo un messaggio criptato in funzione di t, in modo tale che sia il più piccolo possibile. Chiameremmo questo tipo di struttura d'accesso **evolving**: le parti arrivano una alla volta e, nel caso più generale, un sottoinsieme qualificato viene rivelato al dealer solo quando tutte le parti di quel sottoinsieme sono presenti (in casi speciali il dealer conosce la struttura di accesso per iniziare, solo che non ha un limite superiore al numero di partecipanti). Affinché

questo abbia senso, assumiamo che le modifiche alla struttura di accesso siano monotone, vale a dire che i partecipanti vengono solo aggiunti e gli insiemi qualificati rimangano tali.
Nel loro lavoro, [KNY16] hanno mostrato che qualsiasi struttura d'accesso evolutiva può essere realizzata, anche se la dimensione del $t - esimo$ share è 2^{t-1}. Successivamente, considerano la struttura di accesso evolutiva con un k-threshold, per ogni $k \in N$, dove in qualsiasi momento una delle k parti possono ricostruire il segreto ma nessun sottoinsieme di $k-1$ parti può capire nulla sul segreto, ed hanno mostrato uno schema efficiente in cui la dimensione della condivisione della t-esima parte è delimitata da circa $k \cdot \log t$ bits. Il loro schema ha dimostrato di essere ottimale in termini di dimensione dello share per $k = 2$. Prima del loro studio era stato tentato un approccio da Csirmaz e Tardos nel 2012 [CT12] per il 2-threshold che risultava però inefficiente nella grandezza del messaggio condiviso, poiché consisteva nel dare al t-esimo partecipante un bit casuale b_t e tutti i bit $s \oplus b_1, ..., s \oplus b_{t-1}$. Ciò permette ovviamente ad ogni coppia di partecipanti di ricostruire il segreto, e assicura che uno solo di loro non possa ricostruirlo. Generalizzare quest'idea per un valore arbitrario di k causa la grandezza del messaggio condiviso di essere circa t^{k-1}. La costruzione in [KNY16] è meno "ingenua" della precedente, dato che assegna ad ogni parte una generazione, dove la g-esima generazione corrisponde a 2^g partecipanti (quindi sappiamo che consistono in una grandezza geometricamente crescente). Per ognuna di esse viene eseguito una condivisione del segreto per un 2-threshold standard. Partiamo dicendo che sappiamo con certezza a quale generazione appartiene il t-esimo partecipante: $g = \log_2 t$ e che la grandezza della generazione è $SIZE(g) \leq t$.

Capitolo 2

Secret Sharing

2.1 Threshold schemes

Nell'impostazione classica della condivisione del segreto, molti schemi sono noti per strutture ad accesso generale, a seconda della loro rappresentazione [ISN93, BL88, KW93]. Tutti questi schemi risultano con condivisioni di dimensioni esponenziali per strutture ad accesso generale. Uno dei problemi aperti più importanti nell'area del secret sharing è dimostrare la necessità di quote grandi, cioè trovare una struttura di accesso (anche non esplicita) che richieda quote di dimensioni esponenziali. Anche lo schema introdotto in [KNY16] per strutture generali di accesso evolutive risulta con quote di dimensioni esponenziali. Poiché qualsiasi struttura di accesso può essere resa evolutiva, non si può sperare di ottenere qualcosa di meglio che esponenziale in generale (a meno che non ci sia una rivoluzione nell'impostazione classica).

Ci sono diversi schemi per la struttura di accesso a soglia:

2.1.1 Schema di Shamir

In questo schema, qualsiasi t su n (dove $1 \leq t \leq n$) shares può essere usato per recuperare il segreto. Il sistema si basa sull'idea che è possibile adattare un unico polinomio di grado $t-1$ a qualsiasi insieme di t punti che giacciono sul polinomio. Ci vogliono due punti per definire univocamente una linea retta, tre punti per definire completamente una quadratica, quattro punti per definire una curva cubica, e così via. Cioè, ci vogliono t punti per definire un polinomio di grado $t-1$. Il metodo consiste nel creare un polinomio di grado $t-1$ con il segreto come primo coefficiente e gli altri coefficienti scelti a caso. Poi trovare n punti sulla curva e darne uno a ciascuno dei giocatori. Quando almeno t degli n giocatori rivelano i loro punti, ci sono informazioni sufficienti per adattare ad essi un polinomio di grado $t-1$, il cui primo coefficiente è il segreto.
La condivisione del segreto introdotta da Shamir è un perfetto schema con un $(k, n) - threshold$. In questo schema, l'obiettivo è quello di dividere un segreto $\boldsymbol{S}$ (ad esempio la combinazione di una cassaforte, o la formula segreta della Coca-Cola) in n pezzi di dati $S_1, ..., S_n$ (chiamati *shares*) in maniera tale che:

1. Avendo un qualsiasi sottoinsieme di k o più degli S_i renda $\boldsymbol{S}$ facilmente computabile

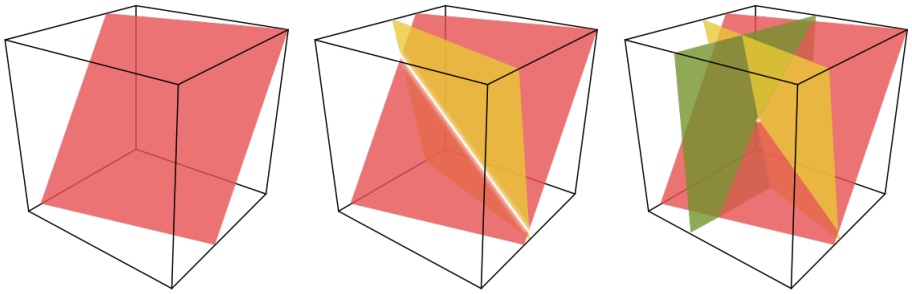
2. Conoscere $k-1$ o meno S_i renda impossibile ottenere qualsiasi informazione su $\boldsymbol{S}$

L'idea essenziale dello schema si basa sul teorema di interpolazione di Lagrange, in particolare che k punti sono sufficienti per determinare univocamente un polinomio di grado inferiore o uguale a k-1. Per esempio, 2 punti sono sufficienti per definire una linea, 3 punti sono sufficienti per definire una parabola, 4 punti per definire una curva cubica e così via. Assumiamo che il nostro segreto $\boldsymbol{S}$ possa essere rappresentato come un elemento a_0 di un campo finito $GF(q)$. Scegliamo a caso k-1 elementi dal $GF(q)$ e costruiamo il polinomio $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{k-1}x^{k-1}$. Costruiamo qualsiasi n punti da esso, per esempio impostiamo $i = 1, \ldots, n$ per recuperare $(i, f(i))$. Ad ogni partecipante viene dato un punto (in input un intero non nullo del polinomio e l'output intero corrispondente). Dato un qualsiasi sottoinsieme di k di queste coppie, possiamo ottenere a_0 usando l'interpolazione, con una possibile formulazione come segue:

$$f(0) = \sum_{j=0}^{k-1} y_j \prod_{\substack{m=0 \\ m \neq j}}^{k-1} \frac{x_m}{x_m - x_j}$$

2.1.2 Schema di Blakley

Due linee non parallele nello stesso piano si intersecano esattamente in un punto. Tre piani non paralleli nello spazio si intersecano esattamente in un punto. Più in generale, n qualsiasi iperpiani (n-1)-dimensionali non paralleli si intersecano in un punto specifico. Il segreto può essere codificato come qualsiasi singola coordinata del punto di intersezione. Se il segreto è codificato usando tutte le coordinate, anche se sono casuali, allora un insider (qualcuno in possesso di uno o più degli iperpiani (n - 1)-dimensionali) ottiene informazioni sul segreto poiché sa che deve trovarsi sul suo piano. Se un insider può ottenere più conoscenze sul segreto di quante ne possa ottenere un outsider, allora il sistema non ha più la sicurezza teoretica dell'informazione. Se viene usata solo una delle n coordinate, allora l'insider non sa più di un outsider (ad esempio, che il segreto deve trovarsi sull'asse x per un sistema a 2 dimensioni). Ad ogni giocatore vengono date abbastanza informazioni per definire un iperpiano; il segreto viene recuperato calcolando il punto di intersezione dei piani e poi prendendo una determinata coordinata di tale intersezione.
Lo schema di Blakley è meno efficiente in termini di spazio di quello di Shamir; mentre gli shares di Shamir sono ciascuno tanto grande quanto il segreto originale, quelli di Blakley sono k volte più grandi, dove k è il numero di soglia dei partecipanti. Lo schema di Blakley può essere migliorato aggiungendo restrizioni su quali piani sono utilizzabili come azioni.
Lo schema risultante è equivalente al sistema polinomiale di Shamir.



Lo schema di Blakley in tre dimensioni: ogni share è un piano, e il segreto è il punto in cui tre shares si intersecano. Due shares non sono sufficienti per determinare il segreto, anche se forniscono abbastanza informazioni per restringere il campo alla linea in cui entrambi i piani si intersecano.

2.2 Modello e definizioni formali

Preso un intero $n \in \mathbb{N}$, denotiamo con $[n]$ l'insieme $1, \cdots, n$. Con log indichiamo il logaritmo in base 2, ed assumiamo che il $\log 0 = -\infty$. Preso un insieme $\boldsymbol{X}$, indichiamo con $x \leftarrow \boldsymbol{X}$ il processo di campionamento di un valore x dalla distribuzione uniforme su $\boldsymbol{X}$. Una funzione $\mathbf{neg} : \mathbb{N} \to \mathbb{R}^+$ viene definita *trascurabile* se per ogni costante $c > 0$ esiste un intero N_c tale che $\mathbf{neg}(\lambda) < \lambda^{-c}$ per ogni $\lambda > N_c$.
Iniziamo descrivendo leggermente l'impostazione standard della perfetta condivisione del segreto. Sia $P_n = 1, \cdots, n$ un insieme di n partecipanti. Una collezione del sottoinsieme $A \subseteq 2^{P_n}$ è monotona se per ogni $B \in A$, con $B \subseteq C$, è vero che $C \in A$.

Definizione 2.2.1 (Struttura d'accesso). Una struttura d'accesso $A \subseteq 2^{P_n}$ è una collezione monotona di sottoinsiemi. Sottoinsiemi in A sono definiti *qualificati*, e sottoinsiemi non in A sono chiamati *non qualificati*.

Uno schema di condivisione del segreto prevede un dealer che abbia un segreto, un insieme di n partecipanti ai quali distribuire gli share del segreto e una struttura d'accesso A. Un tale schema per A è un metodo secondo il quale il dealer distribuisce le shares ai partecipanti in modo tale che ogni sottoinsieme in A possa ricostruire il segreto dai suoi shares, mentre qualsiasi altro sottoinsieme non in A non possa ricavare alcuna informazione sul segreto.

Definizione 2.2.2. Uno schema di condivisione di un segreto $\boldsymbol{S}$ per una struttura d'accesso A consiste in una coppia di algoritmi (SHARE, RECON).
SHARE è una procedura probabilistica che prende in input un segreto s (da un dominio di segreti S tale che $|S| > 2$) e un numero n, e genera in output n shares $\Pi_1^{(s)}, \ldots, \Pi_n^{(s)}$.
RECON, invece, è una procedura deterministica che prende in input gli shares di un sottoinsieme B e ritorna una stringa.
I requisiti sono:

1. **Correttezza:** Per ogni segreto $s \in S$ ed ogni insieme qualificato $B \in A$, sappiamo che:
$$\mathbf{Pr}[RECON(\{\Pi_i^{(s)}\}_{i \in B}, B) = s] = 1$$
2. **Sicurezza:** Per ogni insieme non qualificato $B \notin A$ ed ogni coppia di due segreti differenti $s_1, s_2 \in S | s_1 \neq s_2$, sappiamo che le distribuzioni $(\{\Pi_i^{(s_1)}\}_{i \in B})$ e $(\{\Pi_i^{(s_2)}\}_{i \in B})$ sono identiche.

La *dimensione degli shares* (*share size*) di uno schema è il numero massimo di bit che ogni partecipante possiede nel caso peggiore su tutti i partecipanti e tutti i segreti. Una constatazione utile è che la concatenazione di azioni generate indipendentemente non danneggia la sicurezza. Cioè, per qualsiasi due segreti s_0, s_1 e qualsiasi due strutture di accesso A_0, A_1 , se un insieme di partecipanti non è qualificato in entrambe le strutture, allora non può ottenere alcuna informazione sui segreti. Affermiamo questo statement per due strutture di accesso e due segreti, ma naturalmente si può generalizzare a più strutture:

Claim 2.2.3. Fissate due strutture d'accesso A_0, A_1 con i loro corrispondenti schemi di condivisione del segreto, quattro segreti $s_0^{(0)}, s_1^{(0)}, s_0^{(1)}, s_1^{(1)}$, e un insieme di partecipanti $B \notin A_1, A_2$ non qualificato né in A_0, né in A_1, allora sappiamo

che le distribuzioni $(\{\Pi_{0,i}^{(s_0^{(0)})}, \Pi_{1,i}^{(s_1^{(0)})}\}_{i\in B})$ e $(\{\Pi_{0,i}^{(s_0^{(1)})}, \Pi_{1,i}^{(s_1^{(1)})}\}_{i\in B})$ sono identiche, dove $\Pi_{c,i}^{(s_b^{(a)})}$ per $a, b \in \{0,1\}$ e $i \in [n]$ è una variabile casuale per lo share dell'i-esimo partecipante quando condivide il segreto $s_b^{(a)}$ secondo la struttura d'accesso A_b.

Il precedentemente descritto schema di Shamir per una struttura d'accesso con un (k, n)-threshold soddisfa la seguente affermazione:

Claim 2.2.4([Sha79]): Per ogni $n \in \mathbb{N}$ e $1 \leq k \leq n$, c'è uno schema di condivisione del segreto per segreti di lunghezza m ed una struttura d'accesso con (k, n)-threshold nel quale la grandezza dello share è l, dove $l \geq max\{m, \log q\}$ e $q > n$ è un numero primo (o la potenza di un primo). Inoltre, se $k = 1$ o $k = n$, allora $l = m$.

2.3 Evolving secret sharing

Ricordiamo ora la definizione di struttura d'accesso evolutiva come definita dapprima in [KNY16] e poi confermata in [KP17]. In parole povere, queste definizioni catturano lo scenario in cui la struttura di accesso non è completamente nota alla procedura di condivisione in una sola volta, ma è piuttosto rivelata un po' per volta. Concretamente, i partecipanti arrivano uno per uno e, nel caso più generale, un sottoinsieme qualificato è rivelato solo quando tutti i partecipanti in quel sottoinsieme sono presenti (in casi speciali la struttura di accesso è nota all'inizio, ma non c'è un limite superiore al numero di partecipanti). Per dare un senso al condividere un segreto rispetto a una tale sequenza di strutture di accesso, richiediamo che i cambiamenti alla struttura di accesso siano monotoni, vale a dire che i partecipanti vengano solo aggiunti e che gli insiemi qualificati rimangano qualificati.

Definizione 2.2.5. (evolving access structure). Una struttura d'accesso evolutiva $A \subseteq 2^{\mathbb{N}}$ è una collezione (con possibilità che sia infinita) monotona di sottoinsiemi di numeri naturali tali che per ogni $t \in \mathbb{N}$, la collezione di sottoinsiemi $A_t \triangleq A \cap [t]$ è una struttura d'accesso (come definita in 2.2.1.).

Definizione 2.2.6. (Evolving threshold access structure). Per ogni $k \in \mathbb{N}$, sia la k-soglia evolutiva la struttura d'accesso evolutiva a soglia che contiene tutti i sottoinsiemi di $2^{\mathbb{N}}$ di grandezza almeno k.

Ora definiamo una generalizzazione della struttura d'accesso a soglia definita in precedenza (vedere 2.2.2.). Partiamo dicendo che in questa impostazione, in qualsiasi punto $t \in \mathbb{N}$ abbiamo una struttura d'accesso A_t che definisce se un sottoinsieme è o meno qualificato.

Definizione 2.2.7. (condivisione del segreto in una struttura d'accesso evolutiva). Sia $A = \{A_t\}_{t\in\mathbb{N}}$ una struttura d'accesso evolutiva. Sia S_D un dominio di segreti, con $S_D \geq 2$. Uno schema di condivisione del segreto S per A ed S_D consiste in una coppia di algoritmi (SHARE, RECON). Il primo consiste in una procedura di condivisione probabilistica, il secondo in una di ricostruzione deterministica, tali che soddisfino i seguenti requisiti:

1. $SHARE(s, \{\Pi_1^{(s)}, \ldots, \Pi_{t-1}^{(s)}\})$ prende in input un segreto $s \in S_D$ e gli shares del segreto dei partecipanti 1,. . . ,t-1. In output ritorna lo share per il t-esimo partecipante. Per $t \in \mathbb{N}$ e gli shares del segreto $\Pi_1^{(s)}, \ldots, \Pi_{t-1}^{(s)}$ generati per

i partecipanti 1,. . . ,t-1, costruiamo lo share segreto del t-esimo partecipante come:

$$\Pi_t^{(s)} \leftarrow SHARE(s, \{\Pi_1^{(s)}, \ldots, \Pi_{t-1}^{(s)}\})$$

Con $\Pi_t^{(s)}$ descriviamo la variabile casuale che corrisponde allo share del segreto del t-esimo partecipante, nonostante possa essere considerato un abuso di notazione.

2. **Correttezza:** Per ogni segreto $s \in S_D$ e per ogni $t \in \mathbb{N}$, ogni sottoinsieme qualificato in A_t può ricostruire il segreto. Quindi, per ogni $s \in S_D, t \in \mathbb{N}$, e $B \in A_t$, sappiamo che:

$$\mathbf{Pr}[RECON(\{\Pi_i^{(s)}\}_{i \in B}, B) = s] = 1$$

dove la probabilità riguarda la casualità della procedura di condivisione.

3. **Segretezza:** Per ogni $t \in \mathbb{N}$, ogni sottoinsieme non qualificato $B \notin A_t$, ed ogni due segreti $s_1, s_2 \in S_D$, la distribuzione degli shares del segreto dei partecipanti in B generati con il segreto s_1 e la distribuzione degli shares dei partecipanti in B generati con il segreto s_2 sono identici. Cioè, le distribuzioni $(\{\Pi_i^{(s_1)}\}_{i \in B})$ e $(\{\Pi_i^{(s_2)}\}_{i \in B})$ sono identiche.

Riguardo la scelta della struttura di accesso in modo adattivo. Si può anche considerare una definizione più forte in cui A_t è scelto al tempo t (piuttosto che in anticipo) finché la sequenza di strutture di accesso $A = \{A_1, \ldots, A_t\}$ è in evoluzione. In questa variante, le procedure SHARE e RECON ricevono la struttura di accesso A_t come parametro aggiuntivo. Un esempio illustrativo in cui A_t è noto in anticipo è la struttura di accesso con la k-threshold evolutiva menzionata sopra. (In questo caso k è fisso ed è indipendente da t.) Considereremo (nella Sezione 3) una generalizzazione naturale in cui c'è una sequenza di soglie crescenti $k_1 < k_2$... che dicono quante parti dovrebbero essere presenti in funzione degli indici delle parti presenti stesse. Non è necessario conoscere questa sequenza di soglie in anticipo.

Riguardo il dominio dei segreti. A meno che non venga esplicitamente espresso, assumiamo che il segreto sia un singolo bit (o uno 0 o un 1). Qualcuno può generalizzare un qualsiasi schema del genere per supportare segreti più lunghi condividendo ogni bit del segreto indipendentemente, considerando che la grandezza dello share aumenta di un fattore dipendente dalla lunghezza del segreto.

2.4 Warm-Up

Iniziamo introducendo due schemi, la connessione in un grafo non diretto da s -> t (Undirected st-connectivity), la cui struttura d'accesso può facilmente essere adattata ad un impostazione evolutiva, e i codici di rilevamento di manipolazione algebrica (Algebraic Manipulation Detection Codes), per descrivere degli schemi di condivisione del segreto evolutivi robusti.

2.4.1 Undirected st-connectivity

In questa struttura d'accesso, i partecipanti corrispondono agli archi di un grafo non diretto $G = (V, E)$. Fissiamo due vertici nel grafo, che chiamiamo rispettivamente **s** e **t**, con $s, t \in V$. Un insieme di partecipanti (cioè di archi) è qualificato se e solo se permette un percorso da s -> t.
Questa struttura d'accesso per la condivisione del segreto è stata standardizzata nel paper del 1988 di Benaloh and Rudich [BR88]. Il dealer, preso un segreto $s \in \{0, 1\}$, assegna per ogni vertice $v \in V$ un'etichetta. Quando $v = s$, l'etichetta sarà $\omega_s = s$, mentre quando $v = t$, avremo che $\omega_t = 0$; infine per i restanti vertici, l'etichetta viene scelta in maniera casuale ed indipendente in modo tale che $\omega_v \leftarrow \{0, 1\}$. Lo share del partecipante $e = (u, v) \in E$ è $\omega_u \oplus \omega_v$.
Per dimostrarne la correttezza va considerato un insieme di partecipanti che includa un percorso da s -> t tale che $s = v_1 v_2 \dots v_k = t$; per ricostruire il segreto basta che i partecipanti applichino lo XOR dei loro shares in modo da ottenere:

$$(\omega_{v_1} \oplus \omega_{v_2}) \oplus (\omega_{v_2} \oplus \omega_{v_3}) \oplus \cdots \oplus (\omega_{v_{k-1}} \oplus \omega_{v_k}) = \omega_{v_1} \oplus \omega_{v_k} = s$$

Per quanto riguarda la sicurezza, invece, si può mostrare che ogni sottoinsieme di partecipanti che non forma un percorso da s -> t, mantiene degli share indipendenti dal segreto.

2.4.2 Algebraic Manipulation Detection Codes

Questo tipo di schema è stato utilizzato per trasformare degli schemi di condivisione del segreto standard in schemi robusti.

Definizione 2.2.8. Un codice AMD-(S,G,δ) è una codifica probabilistica di un mapping E: S $\rightarrow$ G preso un insieme S di grandezza S_S, un gruppo G di grandezza S_G e una funzione di decodifica deterministica $D : \mathbb{Z}_G \rightarrow [S] \cup \{\perp\}$ tali che $D(E(s)) = s$ ha probabilità 1 per ogni $s \in [S]$. Inoltre, per ogni $s \in [S]$ e $\Delta \in \mathbb{Z}_G$ sappiamo che

$$\mathbf{Pr}[D(E(s) + \Delta) \notin \{s, \perp\}] \leq \delta$$

Il codice AMD è definito *sistematico* se S è un gruppo; in questo caso, la codifica è della forma $E : S \rightarrow S \times G_1 \times G_2$ ed E(s) è della forma $(s, x, f(x, s))$ per qualche funzione f ed $x \in_R G_1$. La funzione di decodifica di un codice AMD sistematico è dato da $D(s', x', \sigma')$ se $\sigma' = f(s', x')$, altrimenti è data da $\perp$.

Teorema 2.2.9. *Sia $\mathbb{F}$ un campo di grandezza q e caratteristica p, e sia d un intero tale che $d+2$ non sia divisibile per p; allora sappiamo che esiste una costruzione di un codice AMD sistematico $(q^d, q^{d+2}, (d+1)/q)$. La codifica della funzione mappa $\mathbb{F}^d$ in $\mathbb{F}^d \times \mathbb{F} \times \mathbb{F}$.*

Per avere un parametro d'errore pari a γ, preso un dominio di input S, costruiamo lo schema descritto sopra in modo tale che $G = \mathbb{F}_2^t, d = 1$, dove $t = \log S + \gamma + O(1)$. Ci riferiremo a questa costruzione scrivendo $AMD_{S,\gamma}$.

Capitolo 3

Strutture d'accesso evolutive generali: uno schema

Descriviamo qui la costruzione di uno schema di condivisione del segreto valida per ogni struttura d'accesso evolutiva.

Teorema 3.1. *Per ogni struttura d'accesso evolutiva, esiste sempre uno schema di condivisione del segreto per un segreto di 1 bit, nel quale per ogni $t \in \mathbb{N}$, la grandezza dello share del t-esimo partecipante è al più 2^{t-1}.*

Dimostrazione Teorema 3.1. Sia A una struttura d'accesso evolutiva. Sia $s \in \{0,1\}$ il segreto da condividere. Descriviamo ciò che viene salvato dal dealer e come si prepara alla condivisione all'arrivo di un partecipante.
Preso un istante t, il dealer ha 2^t bits denotati con s_A per ogni $A \subseteq [t]$. Inizialmente settiamo $s_\emptyset = s$.
Il valore di ogni s_A, con $A = \{i_1, \ldots, i_k, t\} \subseteq [t]$, viene definito come segue:

1. Se $A \notin A_t$, il dealer prende in maniera uniformemente casuale $r_A \leftarrow \{0,1\}$ e setta $s_A = s_{A\setminus\{t\}} \oplus r_A$. A questo punto dà al t-esimo partecipante il bit r_A.

2. Se $A \in A_t$ e $A\setminus\{t\} \notin A_{t-1}$, il dealer dà al t-esimo partecipante il bit $s_{A\setminus\{t\}}$.

Il t-esimo partecipante ha al più un singolo bit per ogni $A \subseteq [t]$, tale che $t \in A$. Ciò implica che la grandezza dello share del t-esimo partecipante sia di al più 2^{t-1} bit.

Correttezza e sicurezza. Discutiamo la correttezza della precedente procedura al tempo $t \in \mathbb{N}$, con una struttura d'accesso A_t. Sia $A = \{i_1, \ldots, i_k, t\}$ un insieme minimale e qualificato di partecipanti presente al tempo t. Dal momento che $A \in A_t$, sappiamo che per ogni $j \in [k]$, l'i_j-esimo partecipante ha il bit $r_{\{i_1,\ldots,i_j\}}$. Il partecipante t, per costruzione, mantiene il bit $s_A = r_{\{i_1\}} \oplus \ldots \oplus r_{\{i_1,\ldots,i_k\}} \oplus s$. Quindi la correttezza dello schema può subito essere derivata dal fatto che basta che i partecipanti di A applichino lo XOR su tutti gli shares per recuperare il segreto s.

Dimostriamo ora la sicurezza dello schema per induzione su t:

1. **Caso base:** per $t = 1$, lo schema è sicuro, dato che o l'insieme consiste di un partecipante qualificato, nel qual caso non essendoci informazioni pubbliche lo schema è sicuro, o il partecipante non è qualificato. In quest'ultimo scenario il partecipante ottiene un bit casuale $r_{\{1\}}$, indipendente dal segreto.

2. **Ipotesi induttiva:** Supponiamo che lo schema sia sicuro per tutte quelle strutture d'accesso fino a t-1 partecipanti.

3. **Passo induttivo:** Consideriamo ora $A = \{i_1, \ldots, i_k, t\}$, cioè un insieme non qualificato rispetto ad A_t.
   Consideriamo ora il dealer subito dopo aver condiviso lo share al primo partecipante. In questo momento il dealer ha due bit: $s_\emptyset$ ed $s_{\{1\}}$. Ora, osserviamo che il resto della procedura del dealer consiste in due ripetizioni dello schema per strutture d'accesso su t-1 partecipanti e due segreti. Infatti, il punto chiave consiste nel fatto che le condivisioni del segreto vengono svolte indipendentemente (e quindi la loro concatenazione è sicura). Per prima cosa, il dealer condivide il segreto $s_\emptyset$ tra i partecipanti $2, \ldots, t$ rispetto alla struttura d'accesso $A_t^0 = \{A \subseteq \{2, \ldots, t\} | A \in A_t\}$, la quale contiene tutti gli insiemi qualificati dei quali il primo partecipante non fa parte. Poi, il dealer condivide gli shares del segreto $s_{\{1\}}$ tra i partecipanti $2, \ldots, t$ rispetto alla struttura d'accesso $A_t^1 = \{A \subseteq \{2, \ldots, t\} | \{1\} \cup A \in A_t\}$ che contiene tutti gli insiemi qualificati dei quali il primo partecipante è un membro.
   Dal momento che l'insieme originale di partecipanti non è qualificato deve essere che:

   (1) il rimanente insieme di partecipanti (cioè $A \backslash \{1\}$) non è qualificato in A_t^0 e che (2) o $1 \notin A$, o il rimanente insieme di partecipanti non è qualificato in A_t^1. Possiamo inoltre applicare l'ipotesi induttiva sulla condivisione in accordo con A_t^0 e capire che il segreto è indipendente dallo share corrispondente.
   Per quanto riguarda la seconda parte abbiamo due casi possibili:
   (1) Se $1 \notin A$, allora il segreto condiviso ($s_{\{1\}}$) è indipendente da s, e quindi tutta la visione degli shares di A è indipendente da s.
   (2) Se $1 \in A$, ma $A \notin A_t$, allora l'insieme A conosce il mascheramento di $s_\emptyset$ che è un bit casuale $r_{\{1\}}$, tuttavia, l'ipotesi induttiva ci assicura che le azioni delle parti rimanenti siano indipendenti da $s_{\{1\}} = s_\emptyset \oplus r_{\{1\}}$, il che implica che sono indipendenti dal segreto.

   Ogni caso è indipendente dall'altro e in ogni caso le azioni risultanti sono indipendenti dal segreto. Sfruttando la constatazione 2.2.3., concludiamo che il segreto rimane perfettamente nascosto.

Capitolo 4

Condivisione del segreto evolutiva con 2-threshold

In questo capitolo descriviamo un'equivalenza tra i codici prefisso (o codici istantanei) per gli interi e la struttura d'accesso evolutiva con soglia di 2 per schemi di condivisione di segreti da 1 bit. Questa costruzione si basa sui codici prefisso di Elias [Eli75] ed è stata dimostrata ottimale in [KNY16].

Teorema 4.1. Sia $\phi : \mathbb{N} \to \mathbb{N}$, allora un codice prefisso per gli interi, nel quale la lunghezza della t-esima parola di codice è $\phi(t)$ esiste se e solo se uno schema di condivisione del segreto per struttura d'accesso evolutiva con soglia di 2 e un segreto di 1 bit ha una grandezza dello share del t-esimo partecipante di $\phi(t)$.

Corollario 4.2. Esiste uno schema di condivisione del segreto per struttura d'accesso evolutiva con soglia di 2 e un segreto di 1 bit nel quale la grandezza dello share del t-esimo partecipante è di $\log t + 2 \log \log t + 2$.

Corollario 4.3. Per qualsiasi costante $c, l \in \mathbb{N}$, non c'è alcuno schema di condivisione del segreto per una struttura d'accesso evolutiva con un 2-threshold nel quale per ogni $t \in \mathbb{N}$ la grandezza dello share del t-esimo partecipante è al più $\log t + \log \log t + \ldots + \log^{(l)} t + c$, dove $\log^{(i)}(t)$ è l'i-esimo logaritmo di t.

Inoltre, viene data una costruzione *diretta* di uno schema di condivisione del segreto per una struttura d'accesso evolutiva con una soglia di 2 che sia più efficiente nel condividere segreti più lunghi. Questo schema fungerà da warm-up per lo schema usato per le strutture d'accesso a condivisione del segreto per una soglia k qualsiasi, ed è lineare.

Teorema 4.4. Esiste uno schema di condivisione del segreto per struttura d'accesso evolutiva con soglia di 2 e un segreto di l bit nel quale la grandezza dello share del t-esimo partecipante è di $\log t + (l+1) \log \log t + 4l + 1$.

4.1 Codici prefisso

Un codice prefisso è un tipo di codice che si distingue in quanto ha la "proprietà prefisso", che richiede che non ci sia nessun codice nel sistema che sia un prefisso (segmento iniziale) di qualsiasi altro codice.

Per esempio, un codice con parole 9, 55 ha la proprietà prefisso; un codice composto da 9, 5, 59, 55 no, poiché "5" è prefisso di "59" e anche di "55".

Un codice prefisso è un codice univocamente decodificabile: data una sequenza completa e accurata, chi la riceve può identificare ogni parola senza richiedere un marcatore speciale tra le parole. Tuttavia, ci sono codici univocamente decodificabili che non sono codici prefisso; per esempio, l'inverso di un codice prefisso è ancora univocamente decodificabile (è un codice suffisso), ma non è necessariamente un codice prefisso. Usando i codici prefisso, un messaggio può essere trasmesso come una sequenza di codici concatenati, senza marcatori speciali tra le parole per inquadrare le parole nel messaggio. Il destinatario può decodificare il messaggio senza ambiguità, trovando e rimuovendo ripetutamente le sequenze che formano codici validi.

4.2 Connessione ai codici prefisso

Qui dimostriamo il teorema 4.1., e con esso la stretta connessione tra gli schemi di strutture d'accesso evolutive con soglia di 2 e codici prefisso.

Dimostrazione della parte "solo se" del Teorema 4.1. Sia $\Sigma : \mathbb{N} \to \{0,1\}*$ un codice prefisso per gli interi; cioè per qualsiasi $t_1, t_2 \in \mathbb{N}$ tali che $t_1 \neq t_2$, tenga che $\Sigma(t_1)$ non sia un prefisso di $\Sigma(t_2)$. Per $t \in \mathbb{N}$ denotiamo con $\sigma(t)$ la lunghezza della parola di codice $\Sigma(t)$. **Lo schema.** Sia $s \in \{0,1\}$ il segreto da condividere. Sia ω una stringa binaria casuale infinita. Il dealer genera la stringa nel seguente modo: al tempo $t \in \mathbb{N}$ il dealer ha il prefisso di lunghezza $\sigma(t)$ della stringa ω, che scriviamo come $\omega_{[\sigma(t)]}$. Lo share del t-esimo partecipante è una stringa u_t tale che:

1. Se $s = 0$, allora $u_t = \omega_{[\sigma(t)]}$.
2. Se $s = 1$, allora $u_t = \Sigma(t) \oplus \omega_{[\sigma(t)]}$ (XOR bit a bit).

La grandezza dello share del t-esimo giocatore in questo schema è di $\sigma(t)$.
Per ricostruire il segreto, due qualsiasi dei partecipanti t_1 e t_2 che hanno rispettivamente gli shares u_1 e u_2, con $|u_1| \leq |u_2|$, dovrebbero controllare se u_1 è un prefisso di u_2. Se lo è, allora mandano in output $s = 0$, altrimenti $s = 1$.

Correttezza e sicurezza. Se $s = 0$, allora dal momento che u_1 e u_2 sono entrambi prefissi della stessa stringa ω, sappiamo che u_1 è prefisso di u_2. D'altra parte, se $s = 1$, allora $u_1 = \Sigma(t_1) \oplus \omega_{[\sigma(t_1)]}$ e $u_2 = \Sigma(t_2) \oplus \omega_{[\sigma(t_2)]}$, dove $\omega_{[\sigma(t_1)]}$ è un prefisso di $\omega_{[\sigma(t_2)]}$. Dato che Σ è un codice istantaneo, $\Sigma(t_1)$ non è un prefisso di $\Sigma(t_2)$, quindi si deduce che u_1 non è un prefisso di u_2.
La sicurezza dell'implementazione segue dal fatto che sia per $s = 0$ che per $s = 1$ ogni partecipante ha una singola stringa u_t uniformemente distribuita in $\{0,1\}^{\sigma(t)}$. Nel caso in cui $s = 0$ ciò è vero per costruzione, mentre nel caso in cui $s = 1$ è vero dal momento che tutti i partecipanti vedono la parola di codice $\Sigma(t)$ in XOR con $\omega_{[\sigma(t)]}$, che è uniforme.

Dimostrazione del corollario 4.2. Descriviamo il codice prefisso di Elias come una soluzione a due passi (per semplicità, nel descriverlo ignoriamo eventuali problemi di arrotondamento). Il primo passo è quello di costruire uno schema di base, e il secondo è quello di utilizzare lo schema appena costruito ricorsivamente in modo tale da migliorare la grandezza della codifica.
Nello schema base, un numero $t \in \mathbb{N}$ viene codificato in due parti: la prima è formata da $\log t$ zeri, in modo da rappresentare in base 1 la lunghezza di t, e la seconda parte è la rappresentazione binaria di t. In totale la grandezza della codifica è $2 \log t + 1$.
Nel secondo step, codifichiamo la prima parte dello schema di base ricorsivamente

usando lo schema base stesso. In particolare codifichiamo la rappresentazione unaria della lunghezza usando $2\log\log t + 1$ bit. Ciò risulta in un messaggio criptato di grandezza $\log t + 2\log\log t + 2$ (si può applicare ancora ricorsivamente la precedente trasformazione per avere una codifica ancora più corta).

Dimostrazione della parte "se" del Teorema 4.1. Assumiamo che ci venga dato uno schema di condivisione del segreto per struttura d'accesso evolutiva con soglia di 2 e un segreto di 1 bit nel quale la grandezza dello share del t-esimo partecipante è di $\sigma(t)$. Una proprietà che la funzione $\sigma(\cdot)$ deve soddisfare è implicita nella dimostrazione del limite inferiore dimostrato da Kilian e Nisan [KN90] per quanto riguarda la grandezza di uno share negli schemi di condivisione del segreto di 2 su n. Ciò che è stato dimostrato è che le grandezze degli share devono soddisfare la disuguaglianza di Kraft.

Claim 4.5. Per ogni $n \in \mathbb{N}$, per tutti gli schemi a condivisione di segreto con un (2,n)-threshold, sappiamo che $\Sigma_{t=1}^{n} \frac{1}{2^{\sigma(t)}} \leq 1$, dove $\sigma(t)$ è la grandezza dello share del t-esimo partecipante.
Può essere dimostrato che la disuguaglianza di Kraft dà una condizione necessaria e sufficiente affinché esista un codice prefisso per un dato insieme di lunghezze di codici.

Claim 4.6. Per ogni codice prefisso su un alfabeto 0, 1, le lunghezze di codici $\sigma(1), \ldots, \sigma(n)$ deve soddisfare la disuguaglianza $\Sigma_{t=1}^{n} \frac{1}{2^{\sigma(t)}} \leq 1$. Viceversa, dato un insieme di lunghezze di codici che soddisfano questa disuguaglianza, esiste un codice prefisso con queste lunghezze di codici.

La dimostrazione della sufficienza nella doppia implicazione viene svolta per costruzione: data una collezione di lunghezze di codici, è possibile costruire il codice. Inoltre, non c'è bisogno di sapere la collezione di lunghezze in anticipo, cioè possiamo creare il codice al volo, l'importante è che $\Sigma_t \frac{1}{2^{\sigma(t)}} \leq 1$, dove $\sigma(t)$ è la grandezza della codifica del numero t, non superi mai 1.
Quindi, qualsiasi schema di condivisione del segreto per una struttura di accesso evolutiva con soglia 2 in cui la dimensione degli shares del t-esimo partecipante è $\sigma(t)$, implica l'esistenza di un codice prefisso la cui lunghezza del t-esimo codice è $\sigma(t)$ (in questo caso però, l'efficienza potrebbe non essere conservata).

Dimostrazione del corollario 4.3. Supponiamo per contraddizione che ci sia uno schema di condivisione del segreto per la struttura di accesso evolutiva con soglia 2 la cui dimensione dello share del t-esimo partecipante è al più $\sigma(t) = \log t + \log\log t + \ldots + \log^{(l)} t + c$, per delle costanti $c, l \in \mathbb{N}$. Possiamo usare questo schema per implementare uno schema di condivisione del segreto standard con soglia (2, n) nel quale la grandezza dello share del partecipante $t \in [n]$ sia $\sigma(t)$. Partendo dal Claim 4.5., sappiamo che:

$$1 \geq \sum_{t=1}^{n} \frac{1}{2^{\sigma(t)}} \geq \frac{1}{2^c} \cdot \sum_{t=1}^{n} \frac{1}{t \cdot \log t \cdot \ldots \cdot \log^{l-1} t}$$

Ma, se $n \to \infty$, la parte destra della disuguaglianza è almeno pari a
$\frac{1}{2^c} \cdot \int_1^{\infty} \frac{1}{\Pi_{i=0}^{l-1} \log^{(i)}(t)} dt = \log^{(l)} t$, che è strettamente maggiore di 1 per un t abbastanza grande. Questa contraddizione dimostra il nostro enunciato.

4.3 Una costruzione diretta

In questa sezione dimostriamo il Teorema 4.4., una costruzione diretta di uno schema di condivisione del segreto per la struttura di accesso evolutiva con una soglia di 2. La costruzione si basa su un lemma di "composizione ricorsiva".

Lemma 4.7. Presupponiamo esista uno schema di condivisione del segreto per la struttura di accesso evolutiva con soglia 2 ed un segreto ad l bit, nel quale la grandezza dello share del t-esimo partecipante sia $\sigma(t)$. Allora esiste uno schema di condivisione del segreto per la struttura di accesso evolutiva con soglia 2 in cui la grandezza dello share del t-esimo partecipante è di

$$max\{l, \log t\} + \sigma(\log t + 1)$$

Dimostrazione. Sia Π una costruzione di uno schema di condivisione del segreto evolutivo con soglia 2, nel quale la grandezza dello share del t-esimo partecipante quando si sta condividendo un segreto ad l bit è di $\sigma(t)$. Sia, poi, $s \in \{0,1\}^l$ il segreto da essere condiviso. Ogni partecipante, quando arriva, viene assegnato ad una generazione. Le generazioni sono di grandezza crescente: per $g = 0, 1, 2, \ldots$ la g-esima generazione inizia quando il 2^g-esimo partecipante arriva. Quindi, la grandezza della g-esima generazione, cioè il numero di partecipanti che fanno parte di questa generazione, è $SIZE(g) = 2^g$ e il t-esimo partecipante, con $t \in \mathbb{N}$ fa parte della generazione $g = \lfloor \log t \rfloor$.
Quando una generazione inizia, il dealer prepara degli shares per tutti i partecipanti che fanno parte di quella generazione. Consideriamo la g-esima generazione, e vediamo come procede il dealer:

1. Divide s utilizzando uno schema di condivisione del segreto con una soglia di (2, SIZE(g)). Chiama gli shares risultanti $u_1^{(g)}, \ldots, u_{SIZE(g)}^{(g)}$.

2. Genera uno share utilizzando lo schema a condivisione del segreto Π dato un segreto s e gli shares precedenti $\{v^{(i)}\}_{i \in \{0,\ldots,g-1\}}$. Indichiamo lo share risultante con $v^{(g)}$

3. Assegniamo al j-esimo partecipante, con $j \in [SIZE(g)]$, della g-esima generazione lo share segreto in modo tale che sia

$$(u_j^{(g)}, v^{(g)})$$

Correttezza e sicurezza. Siano $t_1, t_2 \in \mathbb{N}$ due partecipanti qualsiasi. Se t_1 e t_2 appartengono alla stessa generazione g, cioè se $\lfloor \log t_1 \rfloor = \lfloor \log t_2 \rfloor = g$, allora possono ricostruire il segreto s utilizzando la procedura di ricostruzione dello schema con soglia (2, SIZE(g)) utilizzando lo share $u^{(g)}$. Se invece sono in due generazioni $g_1 \neq g_2$, allora i partecipanti possono computare s utilizzando la procedura di ricostruzione dello schema con soglia 2 e i due shares $v^{(g_1)}$ e $v^{(g_2)}$. Ciò assicura quindi la correttezza.
Per quanto riguarda la sicurezza va considerato ogni partecipante $t \in \mathbb{N}$ dalla generazione g. Sfruttiamo il Claim 2.2.3. e l'assunzione che lo schema con soglia (2, SIZE(g)) e quello evolutivo con soglia 2 siano sicuri, insieme al fatto che la condivisione secondo entrambi è fatta indipendentemente, per ottenere che il segreto sia perfettamente nascosto e concludere la dimostrazione.

Analisi della grandezza di uno share. Fissato un partecipante $t \in \mathbb{N}$ e presupponiamo che sia il j-esimo partecipante della generazione $g = \lfloor \log t \rfloor$.
Lo share è composto da due parti:

1. $u_j^{(g)}$ - generato dalla condivisione del segreto s utilizzando uno schema per (2, size(g))-threshold. Poiché la $SIZE(g) = 2^g$ e sfruttando il claim 2.2.4 sappiamo che

$$|u_j^{(g)}| \leq max\{l, \log(SIZE(g))\} \leq max\{l, \log t\}$$

2. $v^{(g)}$ - generato creando uno share di uno schema di condivisione del segreto Π per lo schema evolutivo con 2-threshold. Ricordiamo che g shares sono state generate per le generazioni precedenti. Pertanto,

$$|v^{(g)}| = \sigma(g+1) = \sigma(\lfloor \log t \rfloor + 1)$$

Quindi, la grandezza totale degli share nello schema Π' è delimitato dal $max\{l, \log t\} + \sigma(\log t + 1)$.

Dimostrazione del Teorema 4.4. Iniziamo con lo schema descritto alla fine della sezione 3 utilizzato per la condivisione di segreti ad un bit per strutture d'accesso evolutive con una soglia di 2. Denotiamo con $\Pi^{(0)}$ questo schema quando verrà usato per condividere un segreto ad l bit (condividendo indipendentemente ogni bit). La grandezza dello share in questo schema è

$$\sigma^{(0)}(t) = lt$$

Utilizzando il Lemma 4.7. otteniamo uno schema $\Pi^{(1)}$, nel quale la grandezza dello share del t-esimo partecipante è:

$$\begin{aligned}\sigma^{(1)}(t) &= max\{l, \log t\} + \sigma^{(0)}(\log t + 1) \\ &= max\{l, \log t\} + l \log t + l \\ &\leq (l+1)\log t + 2l\end{aligned}$$

Applicando ancora il Lemma 4.7. otteniamo uno schema $\Pi^{(2)}$ con grandezza degli shares pari a:

$$\begin{aligned}\sigma^{(2)}(t) &= max\{l, \log t\} + \sigma^{(1)}(\log t + 1) \\ &\leq max\{l, \log t\} + (l+1)\log(\log t + 1) + 2l \\ &\leq \log t + (l+1)\log\log t + 4l + 1\end{aligned}$$

Notiamo che applicando il Lemma 4.7. altre volte si può diminuire la dipendenza moltiplicativa rispetto ad l in termini ancora minori.

Capitolo 5

Schema a condivisione del segreto evolutivo con k-threshold

In questo capitolo, diamo una costruzione di uno schema a condivisione del segreto per strutture d'accesso evolutive con una soglia di k, con $k \in \mathbb{N}$. Lo schema, descritto in [KNY16], è lineare.

Teorema 5.1. per ogni $k, l \in \mathbb{N}$, c'è uno schema di condivisione del segreto per strutture d'accesso evolutive con soglia k ed un segreto a l bit, per cui per ogni $t \in \mathbb{N}$ la grandezza dello share del t-esimo partecipante è:

$$(k-1) \cdot \log t + 6k^4 l \log\log t \cdot \log\log\log t + 7k^4 l \log k$$

L'approccio usato segue quello del Teorema 4.4.
Iniziamo con uno schema di base che abbia una buona dipendenza su k, ma non buona su t, e utilizziamo una tecnica di riduzione del dominio per ottenere una migliore dipendenza rispetto a t. Presentiamo ora una costruzione per strutture d'accesso evolutive con k-threshold la cui grandezza degli share del t-esimo partecipante ha una dipendenza quasi lineare in $k \cdot t$.

5.1 Schema base

Lemma 5.2. Per ogni $k, l \in \mathbb{N}$, c'è uno schema di condivisione del segreto per le strutture d'accesso evolutive con una soglia di k ed un segreto ad l bit nel quale per ogni $t \in \mathbb{N}$ la grandezza dello share del t-esimo partecipante è delimitato da $kt \cdot max\{l, \log(kt)\}$.

Dimostrazione. Il seguente schema può essere visto come una variante dello schema per strutture d'accesso generali descritto nella sezione 3, al quale viene aggiunto un meccanismo di generazioni. L'idea è quella di considerare ogni possibile combinazione di k partecipanti come prima, ma di raggrupparli in generazioni, ignorare l'identità di un partecipante singolo in una generazione, e spostare l'attenzione piuttosto sulla loro quantità.

Ad ogni partecipante, quando arriva, viene assegnata una generazione. Al t-esimo, con $t \in \mathbb{N}$ viene assegnata la generazione $g = \lfloor \log_k t \rfloor$. Le generazioni crescono in dimensione: la g-esima generazione inizia quando il k^g-esimo partecipante arriva. Perciò la grandezza della g-esima generazione, cioè il numero di partecipanti che ne

fanno parte, è $SIZE(g) = k^{g+1} - k^g = (k-1) \cdot k^g$.
Sia $s \in \{0,1\}^l$ il segreto. Quando una generazione g inizia, il dealer ricorda k^g stringhe s_A di l bit, per ogni $A = (c_0, \ldots, c_{g-1}) \in \{0, \ldots, k\}^g$. Se $g = 0$ viene ricordato solo il segreto.
Si può intuire che ogni s_A così fatta è una stringa di l bit che viene condivisa con i partecipanti della g-esima generazione, presupponendo che nella generazione $i \in \{0, \ldots, g-1\}$ sono arrivati c_i partecipanti. Spieghiamo come il dealer imposta il valore di s_A quando $A = (c_0, \ldots, c_g)$.
(Sia $s_{prev(A)} = s$ se $g = 0$ e $s_{prev(A)} = s_{(c_0,\ldots,c_{g-1})}$ altrimenti).

1. Se $c_g = 0$, settiamo $s_A = s_{prev(A)}$ e ci fermiamo (HALT).
2. Se $c_0 + \ldots + c_g < k$, allora il dealer:
   (a) prende un $r_A \leftarrow \{0,1\}^l$ in maniera uniforme e casuale
   (b) fissa $s_A = s_{prev(A)} \oplus r_A$
   (c) condivide r_A, stringa di l bit, tra i partecipanti della g-esima generazione utilizzando lo schema di condivisione del segreto di Shamir per un threshold di $(c_g, SIZE(g))$
3. Se $c_0 + \ldots + c_g = k$, allora il dealer condivide la stringa $s_{prev(A)}$ di l bit tra i partecipanti della g-esima generazione utilizzando lo schema di condivisione del segreto di Shamir per un threshold di $(c_g, SIZE(g))$

Correttezza e sicurezza. Per dimostrare la correttezza, consideriamo un qualsiasi insieme minimale e qualificato di partecipanti, che avrà cardinalità k. Sia g la generazione dell'ultimo partecipante e sia $A = (c_0, \ldots, c_g)$ una tupla di numeri che indichi quanti partecipanti arrivano da ogni generazione. Sappiamo che $\sum_{i=0}^{g} c_i = k$, e senza perdita di generalità che $c_g > 0$. Mostriamo che questi partecipanti possono recuperare s. Senza dubbio possiamo affermare che c_g partecipanti dalla generazione g possono ricostruire $s_{prev(A)}$, il quale è uguale a $s \oplus r_{(c_0)} \oplus r_{(c_0,c_1)} \oplus \ldots \oplus r_{(c_0,\ldots,c_{g-1})}$ se $g \neq 0$, altrimenti è esattamente s e possiamo concludere senza operazioni. Quindi i partecipanti c_0 dalla generazione $g = 0$ possono recuperare $r_{(c_0)}$, i partecipanti c_1 dalla generazione $g = 1$ possono recuperare $r_{(c_0,c_1)}$ e così via. Quindi tutti i partecipanti possono, insieme, ricostruire e recuperare s.

Per quanto riguarda la sicurezza procediamo per induzione in maniera simile alla dimostrazione effettuata per gli schemi per strutture d'accesso generali data nella sezione 3.

(a) **Caso base.** Il caso base segue immediatamente dalla sicurezza dello schema di Shamir.

(b) **Ipotesi induttiva.** Assumiamo ora come ipotesi induttiva che lo schema sia sicuro per g generazioni e per ogni soglia fino a k, e dimostriamo che lo è ancora per g + 1 generazioni e per ogni soglia.

(c) **Passo induttivo.** Dopo che il dealer ha generato degli shares per la generazione 0, ha k stringhe di l bit: $s_{(0)}, s_{(1)}, \ldots, s_{(k-1)}$, dove $s_0 = s$ è il secreto originario e sapendo che per $i \in [k-1]$, allora $s_{(i)} = s \oplus r_{(i)}$

per un $r_{(i)}$ casuale. Stiamo quindi dicendo che ogni $s_{(i)}$ è un one-time pad indipendente del segreto, dove ogni pad viene condiviso tra i partecipanti della 0-esima generazione in maniera tale che ogni i di loro possa recuperarlo (come da schema di Shamir). La rimanente procedura del dealer consiste nel condividere indipendentemente ogni $s_{(i)}$ utilizzando la struttura d'accesso evolutiva con un threshold di (k, i) tra le g generazioni. Dal momento che l'insieme controllato da un eventuale avversario non è qualificato, per ogni segreto $s_{(i)}$ o gli sarà impossibile recuperare il segreto criptato, o non riuscirà a ricavare il pad, rispettivamente grazie alla sicurezza dello schema di Shamir o da quella garantita dall'ipotesi induttiva. Quindi, dal momento che la condivisione tra diversi segreti avviene indipendentemente, possiamo concludere grazie al Claim 2.2.3. che il segreto è perfettamente nascosto, anche nel momento in cui un avversario riesca a vedere un insieme completo di shares.

Analisi della grandezza di uno share. Lo share del t-esimo partecipante, appartenente alla g-esima generazione, è composto da k^{g+1} shares generati attraverso lo schema con un threshold standard su SIZE(g) partecipanti. Perciò, in totale, la grandezza di uno share del t-esimo partecipante è delimitato da $k^{g+1} \cdot max\{l, \log(SIZE(g))\}$. Ricordiamo inoltre che $g = \lfloor \log_k t \rfloor$ e che $SIZE(g) = (k-1) \cdot k^g$.
Quindi possiamo concludere che la grandezza dello share è limitato da:

$$k \cdot t \cdot max\{l, \log((k-1) \cdot t)\} \leq kt \cdot max\{l, \log(kt)\}$$

5.2 Composizione ricorsiva

Lemma 5.3. Fissiamo $k, l \in \mathbb{N}$. Presupponiamo esista uno schema di condivisione del segreto per strutture d'accesso evolutive con una soglia di k e un segreto di l-bit nel quale la grandezza dello share del t-esimo partecipante è $\sigma(t)$. Allora, esiste uno schema di condivisione del segreto per strutture d'accesso evolutive con una soglia di k e un segreto di l-bit nel quale la grandezza dello share del t-esimo partecipante è al più $(k-1) \cdot \log t + k \cdot \sigma(\log t + k) + k^2 + l$.

Dimostrazione Lemma 5.3. Sia Π uno schema di condivisione del segreto per strutture d'accesso evolutive con una soglia di k la cui grandezza dello share del t-esimo partecipante è $\sigma(t)$. Sia $s \in \{0,1\}^l$ il segreto da condividere. Ogni partecipante è assegnato a una generazione. Le generazioni crescono di dimensione all'aumentare delle generazioni: per $g = 0, 1, 2, \ldots$, la g-esima generazione inizia quando il $2^{(k-1)\cdot g}$-esimo partecipante arriva. Quindi, il partecipante $t \in \mathbb{N}$ è parte della generazione $g = \lfloor (\log t)/(k-1) \rfloor$, mentre il numero di partecipante che fanno parte della g-esima generazione è:

$$SIZE(g) = 2^{(k-1)\cdot(g+1)} - 2^{(k-1)\cdot g} = 2^{(k-1)\cdot g} \cdot (2^{(k-1)} - 1) \leq t \cdot 2^{k-1}$$

Quando una generazione inizia, il dealer prepara gli shares per tutti i partecipante che sono membri di quella generazione. Ci concentreremo su ciò che avviene all'inizio della generazione g e descriveremo la procedura seguita dal dealer:

1. Dividiamo s usando uno schema di condivisione del segreto con soglia di (k, SIZE(g)). Siano gli shares risultanti $u_1^{(g)}, \ldots, u_{SIZE(g)}^{(g)}$.

2. Generiamo k-1 shares utilizzando lo schema Π dato il segreto s e gli shares precedenti $\{v_j^{(i)}\}_{i \in [g-1], j \in [k-1]}$. Siano gli shares risultanti $v_1^{(g)}, \ldots, v_{k-1}^{(g)}$.

3. Per $i \in [k-1]$, dividiamo $v_i^{(g)}$ utilizzando uno schema di condivisione del segreto con un threshold di (i, SIZE(g)).
   Siano gli shares risultanti $\{w_{i,1}^{(g)}, \ldots, w_{i,SIZE(g)}^{(g)}\}_{i \in [k-1]}$.

4. Settiamo la condivisione dello share del j-esimo partecipante nella g-esima generazione (con $j \in SIZE(g)$) in modo tale che

$$(u_j^{(g)}, w_{1,j}^{(g)}, \ldots, w_{k-1,j}^{(g)})$$

Correttezza e sicurezza. Mostriamo ora che k partecipanti possono recuperare il segreto. Se tutti i partecipanti provengono dalla stessa generazione g, allora possono usare i loro $u^{(g)}$ per la procedura di ricostruzione dello schema con soglia di (k, SIZE(g)) e recuperare s.
Mostriamo ora come k partecipanti che provengono da almeno due generazioni possano insieme recuperare k shares per lo schema Π, evolutivo, con una soglia di k. Per correttezza di Π, usando questi shares loro possono ricostruire s. Infatti, assumiamo che c_0 partecipanti provengano dalla generazione 0, mentre c_1 partecipanti dalla generazione 1 e così via, dove vi è una qualche generazione g tale che $\sum_{i=0}^{g} c_i = k$, e che per ogni i si sappia che $c_i \leq k-1$.

Claim 5.4. Qualsiasi $c \in [SIZE(g)]$ partecipanti da una generazione g può calcolare $v_c^{(g)}$.
Dimostrazione Claim 5.4. I c partecipanti hanno:
(1). c shares per lo schema con soglia di (1, SIZE(g)) che dà $v_1^{(g)}$,
(2). c shares per lo schema con soglia di (2, SIZE(g)) che dà $v_2^{(g)}$, e così via.

Partendo da questo claim, sappiamo che k partecipanti possono recuperare $\sum_{i=0}^{g} c_i = k$ per lo schema evolutivo con soglia di k, come richiesto.

Per quanto riguarda la sicurezza, consideriamo invece un qualsiasi insieme di k - 1 partecipanti B, dove c_i partecipanti vengono dalla generazione i e $\sum c_i = k-1$. Per prima cosa, gli u shares dello schema con threshold (k, SIZE(g)) tenuti dai partecipanti in B sono indipendenti dal segreto, dal momento che B ha meno di k partecipanti. Quindi, sfruttando il Fact 2.3., resta da mostrare che il segreto sia indipendente dagli shares tenuti da B relativi allo schema Π evolutivo con un threshold k. Consideriamo ciò che vedono i partecipanti in B che provengono dalla generazione g, denotata da $B|_g$: loro hanno gli shares $\{w_{1,j}^{(g),\ldots,w_{k-1,j}^{(g)}}\}_{j \in B_g}$, dove $w_i^{(g)}$ è uno share con soglia di (i, SIZE(g)) di $v_i^{(g)}$, che a sua volta è uno share dello schema evolutivo con threshold k. Sfruttando la sicurezza dello schema di Shamir, e dal momento che c_g partecipanti sono presenti dalla generazione g, i valori $v_{c+1}^{(g)}, \ldots, v_{k-1}^{(g)}$ sono indipendenti dal segreto dati tutti gli altri shares dei partecipanti in B, che quindi sono perfettamente nascosti. Ora, dal momento che $\sum c_i < k$, la concatenazione degli shares $v_{c_i}^{(i)}$ consiste in meno di k shares per lo schema evolutivo con soglia k, il che implica che l'intero insieme di shares è indipendente dal segreto.

Analisi della grandezza di uno share. Delimitiamo la grandezza di ogni componente nello share del partecipante t nello schema risultante. Lo share del t-esimo partecipante, che è il j-esimo partecipante della generazione $g = \lfloor (\log t)/(k-1) \rfloor$, è

composto di $u_j^{(g)}$ e $w_{1,j}^{(g)}, \ldots, w_{k-1,j}^{(g)}$, dove:

1. $u_j^{(g)}$ viene generato dalla condivisione del segreto s con uno schema con un threshold di (k, SIZE(g)). Sfruttando il Claim 2.4., sappiamo che:

$$|u_j^{(g)}| \leq max\{l, \log(SIZE(g))\} \leq max\{l, \log t + (k-1)\}$$

2. $w_{i,j}^{(g)}$ viene generato dalla condivisione del segreto $v_i^{(g)}$ usando uno schema con soglia di (k, SIZE(g)). Sfruttando il Claim 2.4., per $1 < i \leq k-1$ sappiamo che:

$$|w_{i,j}^{(g)}| \leq max\{\log(SIZE(g)).|v_i^{(g)}|\} \leq max\{\log t + (k-1), |v_i^{(g)}|\}$$

   e che per $i = 1$ vale che:

$$|w_{1,j}^{(g)}| = |v_i^{(g)}|$$

   - $v_i^{(g)}$, a sua volta, viene generato creando uno share dello schema evolutivo di condivisione Π con soglia k. Ricordiamo infatti che $g \cdot (k-1) \leq \log t + (k-1)$ shares sono state generate per generazioni precedenti a g. Quindi, per ogni $i \in [k-1]$:

$$|v_i^{(g)}| \leq \sigma(\log t + i) \leq \sigma(\log t + (k-1))$$

   Quindi, per $1 < i \leq k-1$:

$$|w_{i,j}^{(g)}| \leq max\{\log t + (k-1), \sigma(\log t + (k-1))\}$$

   mentre per $i = 1$:

$$|w_{1,j}^{(g)}| \leq \sigma(\log t + (k-1))$$

Possiamo concludere che la grandezza totale degli share in questo schema ha come limite superiore:

$$max\{l, \log t + (k-1)\} + \sigma(\log t + (k-1)) + (k-2) \cdot max\{\log t + (k-1), \sigma(\log t + (k-1))\}$$

$$\leq \log t + k + l + \sigma(\log t + k) + (k-2)(\log t + k + \sigma(\log t + k))$$

$$\leq (k-1)\log t + k \cdot \sigma(\log t + k) + k^2 + l$$

5.3 Dimostrazione del Teorema 5.1.

Usiamo lo schema evolutivo per una soglia di k, costruito nella sezione 5.1., nel quale la grandezza dello share del t-esimo partecipante è di $\sigma^{(0)}(t) = kt \cdot max\{l, \log(kt)\}$. Utilizzando il Lemma 5.3., otteniamo uno schema $\Pi^{(1)}$ evolutivo con threshold k, nel quale la grandezza degli share del t-esimo partecipante è di:

$$\sigma^{(1)}(t) = (k-1) \cdot \log t + k \cdot \sigma^{(0)}(\log t + k) + k^2 + l$$

Cerchiamo ora di limitare $\sigma^{(0)}(\log t + k)$.

Claim 5.5. $\sigma^{(0)}(\log t + k) \leq 4k \log t \cdot \log\log t + 4k^2 \cdot \log(2k) + 2k^2 l \log t$

Dimostrazione Claim 5.5. Se $k > \log t$, allora:

$$\sigma^{(0)}(\log t + k) \leq \sigma^{(0)}(2k) \leq 2k^2 \cdot max\{l, \log(2k^2)\} \leq 4k^2 \log(2k) + 2k^2 l$$

Se, invece, $k \leq \log t$, allora:

$$\sigma^{(0)}(\log t + k) \leq \sigma^{(0)}(2 \log t)$$

$$\leq k \cdot 2 \log t \cdot max\{l, \log(k \cdot 2 \log t)\}$$

$$\leq 2k \log t \cdot \log \log t + 2k \cdot \log t \cdot \log(2k) + 2kl \log t$$

$$\leq 4k \log t \cdot \log \log t + 4k^2 \log(2k) + 2k^2 l \log t$$

,

dove l'ultima disuguaglianza è vera poiché $2k \log t \cdot \log(2k) \leq 2k \log t \cdot \log \log t + 4k^2 \log(2k)$ e che $2kl \log t \leq 2k^2 l \log t$. Mettendole insieme otteniamo che:

$$\sigma^{(0)}(\log t + k) \leq max\{\sigma^{(0)}(2 \log t), \sigma^{(0)}(2k)\}$$

$$\leq 4k \log t \cdot \log \log t + 4k^2 \log(2k) + 2k^2 l \log t$$

Da questo otteniamo che:

$$\sigma^{(1)}(t) = (k-1) \cdot \log t + k \cdot \sigma^{(0)}(\log t + k) + k^2 + l$$

$$\leq (k-1) \cdot \log t + 4k^2 \log t + \log \log t + 4k^3 \log(2k) + 2k^3 l \log t + k^2 + l$$

$$\leq 5k^2 \log t \cdot \log \log t + 2k^3 l \log t + 5k^3 \log k + l$$

Utilizzando ancora il Lemma 5.3., otteniamo uno schema $\Pi^{(2)}$ nel quale la grandezza dello share del t-esimo partecipante è:

$$\sigma^{(2)}(t) = (k-1) \cdot \log t + k \cdot \sigma^{(1)}(\log t + k) + k^2 + l$$

Limitiamo ora $\sigma^{(1)}(\log t + k)$ come segue:

$$\sigma^{(0)}(\log t + k) \leq max\{\sigma^{(0)}(2 \log t), \sigma^{(0)}(2k)\}$$

$$\leq 5k^2 \cdot \log(2 \log t) \cdot \log \log(2 \log t) + 2k^3 l \log(2 \log t) +$$

$$5k^2 \cdot \log(2k) \cdot \log \log(2k) + 2k^3 l \log(2k) + 5k^3 \log k + l$$

$$\leq 6k^2 \cdot \log \log t \cdot \log \log \log t + 3k^3 l \log \log t + 6k^3 l \log k + l$$

Così otteniamo che:

$$sigma^{(2)}(t) = (k-1) \cdot \log t + k \cdot \sigma^{(1)}(\log t + k) + k^2$$

$$\leq (k-1) \cdot \log t + 6k^3 \log \log t \cdot \log \log \log t + 3k^4 l \log \log t + 6k^4 l \log(k) + 2lk^2$$

$$\leq (k-1) \cdot \log t + 6k^4 l \log \log t \cdot \log \log \log t + 7k^4 l \log k$$

Il lemma 5.3. può essere applicato ancora ed ancora in modo da diminuire la dipendenza rispetto a $\log \log t \cdot \log \log \log t$. Con questo metodo però non può essere migliorata la dipendenza su $\log t$.

Capitolo 6

Uno schema per il threshold dinamico

In questo capitolo ci riferiremo allo schema di condivisione del segreto per strutture d'accesso evolutive con un threshold dinamico come descritto in [KP17]. Questa struttura d'accesso ha come parametri una sequenza di valori soglia (valori di threshold) $k_1 \leq k_2 \ldots$, tali che al tempo t gli insiemi qualificati siano quelli di cardinalità almeno k_t. La condizione che $k_t \leq k_{t+1}$ è necessaria per garantire la monotonicità della sequenza delle strutture d'accesso, cioè per garantire che le strutture d'accesso siano effettivamente evolutive.

Definizione 6.1. (Threshold dinamico). Una struttura d'accesso con threshold dinamico ha come parametri una (anche infinita) sequenza di numeri $k_1 \leq k_2 \leq \ldots$; per ogni $t \in \mathbb{N}$, l'insieme A_t di insiemi qualificati, al tempo t, contiene tutti quegli insiemi di cardinalità almeno k_t.

Una struttura d'accesso con soglia dinamica di particolare interesse è quella in cui la soglia, in qualsiasi momento è una funzione fissata. Nello specifico, la funzione che considereremo è quella nella quale, al tempo t, gli insiemi qualificati sono quelli con una cardinalità di almeno $\gamma \cdot t$, per una $\gamma \in (0, 1)$ fissata.

Definizione 6.2. (Threshold dinamico con parametro γ). Preso un parametro $\gamma \in (0, 1)$, la struttura d'accesso con una soglia dinamica e un parametro γè la struttura ad accesso dinamico appena descritta, che ha come sequenza di numeri $\gamma \cdot 1, \gamma \cdot 2, \ldots$.
Quindi, i k partecipanti $i_1 < \cdots < i_k$ è qualificato se e solo se esiste un indice $j \in [j]$ tale che $|\{i_1, \ldots, i_j\}| \geq \gamma \cdot i_j$.

Teorema 6.3. Per qualsiasi sequenza di valori di soglia $\{k_t\}_{t\in\mathbb{N}}$ che definisce una struttura d'accesso con un threshold dinamico, esiste uno schema a condivisione del segreto, per un segreto di 1-bit la cui grandezza dello share del t-esimo partecipante è limitato superiormente da $O(t^4 \cdot \log t)$ bits.

Idea generale della dimostrazione. L'idea principale è quella di rappresentare la struttura d'accesso come un albero di decisione infinito, i cui nodi dell'i-esimo livello sono etichettati da x_i. Un albero di decisione infinito può essere trasformato in uno schema di condivisione del segreto sfruttando uno schema evolutivo per una connessione non diretta da s -> t, come descritta nella sezione 2.4.1.; come abbiamo visto, però, utilizzandola come descritta in [KNY16], la grandezza degli share è

proporzionale alla grandezza dell'albero, e ciò porterebbe, in uno schema a soglia dinamica, uno schema con grandezza degli share esponenziale.
Per avallare questo problema, Komargodski e Paskin-Cherniavsky, hanno osservato che l'albero di decisione può essere compresso in maniera tale che ogni livello sia etichettato da una sequenza di variabili $x_i, \ldots, x_j$, e non solo da x_i. Una simile sequenza viene definita *generazione*. Ora, dal momento che ogni libello è etichettato da una sequenza di variabili, definiamo ogni arco in modo che sia una funzione booleana monotona delle variabili nella generazione. Questa operazione riduce il numero di archi nell'albero. Se, inoltre, la funzione dovesse essere abbastanza semplice, cioè se dovesse esistere uno schema di condivisione del segreto abbastanza semplice, si riduce la grandezza degli shares della nostra costruzione: infatti possiamo a quel punto condividere il segreto in base al nuovo albero di decisione, avendo un insieme "virtuale" con molti meno partecipanti che in precedenza, per poi ricondividere questi shares attraverso uno schema di condivisione del segreto tra i partecipanti all'interno di una generazione.

Nel caso di una maggioranza dinamica, ogni arco tra due generazioni è etichettato dal numero di partecipanti nella generazione che è arrivata. Questa è l'unica informazione che per ogni generazione nella struttura va ricordata. Se a questo punto dovessero arrivare abbastanza partecipanti da ricostruire il segreto, deve esistere un percorso che porta ad un nodo accettante, e viceversa. Per fare ciò si può creare questa struttura d'accesso efficientemente utilizzando lo schema di Shamir.
Infine va descritto come imporre la grandezza di una generazione. Se dovesse essere troppo piccola, infatti non salveremmo molto spazio nella grandezza dell'albero, mentre se dovesse essere troppo grande, allora avremmo tanti partecipanti in ogni generazione, e il primo partecipante in questa generazione dovrebbe compiere troppi calcoli per essere una soluzione ottimale. La scelta esatta, per quanto riguarda il caso di un threshold dinamico, vede come impostazione ottimale una grandezza della generazione tale che aumenti con un ritmo polinomiale, cioè la grandezza dell'i-esima generazione è il quadrato della grandezza della (i-1)-esima generazione.

Dimostrazione Teorema 6.3. Assegniamo ad ogni partecipante $t \in \mathbb{N}$ una generazione **GenOf**(t). La dimensione della generazione i è doppiamente esponenziale, cioè $\mathbf{GenSz}(i) = 2^{2^i}$. Quindi, il t-esimo partecipante appartiene al più alla $\lceil \log \log t \rceil$-esima generazione, in modo da includere al più t^2 partecipanti. Il primo insieme di partecipanti nella g-esima generazione ha una grandezza di $\sum_{i=1}^{g} \mathbf{GenSz}(i) = \sum_{i=1}^{g} 2^{2^i}$. Ciò che il dealer ha in conclusione della g-esima generazione consiste in delle stringhe che denoteremo con s_A, sapendo che A varia su ogni tupla $(c_0, \ldots, c_g)$, dove $c_i \in [2^{2^i}]$. In altre parola, il dealer ha una stringa s_A per ogni $A = (c_0, \ldots, c_g) \in [\mathbf{GenSz}(0)] \times \ldots \times [\mathbf{GenSz}(g)]$, dove $\mathbf{GenSz}(i) = 2^{2^i}$. In un certo senso, si può affermare che c_i rappresenta il numero di partecipanti presenti nella generazione i.

Indichiamo l'indice in generale dell'i-esimo partecipante per la g-esima generazione all'inizio del tempo con **IdxOf(g, i)**. Sia s il segreto da condividere, e settiamo $s_0 = s$. Quando la (g+1)-esima generazione inizia, per ogni $(c_0, \ldots, c_g) \in [\mathbf{GenSz}(0)] \times \ldots \times [\mathbf{GenSz}(g)]$ il dealer esegue le seguenti operazioni:

1. Per ogni partecipante $i \in [\mathbf{GenSz}(g+1)]$:
   (a) Condivide il segreto $s_{(c_0,\ldots,c_g)}$ utilizzando uno schema con un threshold di $(k_{\mathbf{IdxOf}(g+1,i)} - \sum_{i=1}^{g} c_i, i)$ per ottenere degli shares $\Pi_1, \ldots, \Pi_i$.

(b) Per ogni $j \in [i]$, dà al j-esimo partecipante della generazione lo share Π_j.

2. Per ogni $c_{g+1} \in [\mathbf{GenSz}(g+1)]$:

   (a) Genera $r_{(c_0,\ldots,c_{g+1})} \leftarrow \{0,1\}$ in maniera uniformemente casuale.

   (b) Condivide $r_{(c_0,\ldots,c_{g+1})}$ utilizzando uno schema con threshold pari a $(c_g, \mathbf{GenSz}(g+1))$ tra i partecipanti della (g + 1)-esima generazione.

   (c) Pone $s_{(c_0,\ldots,c_{g+1})} = s_{(c_0,\ldots,c_g)} \oplus r_{(c_0,\ldots,c_{g+1})}$.

Correttezza e sicurezza. Per dimostrare la correttezza, osserviamo che se i partecipanti c_i arrivano dalla generazione i, per ogni $i \in [g+1]$, allora basta la correttezza dello schema di Shamir per permettere loro di recuperare $r_{(c_0)}, r_{(c_0,c_1)}$ fino a $r_{(c_0,\ldots,c_g)}$. Assumiamo che l'insieme attualmente presente sia qualificato, e che l'ultimo partecipante arrivato sia l'i-esimo della (g + 1)-esima generazione. In più, assumiamo che dalla (g + 1)-esima generazione ci siano l partecipanti presenti dai primi i partecipanti. Sapendo che l'insieme è qualificato, sappiamo che $\sum_{i=0}^{g} c_i + l \geq k_{\mathbf{IdxOf}(g+1,i)}$. Quindi, ancora grazie alla correttezza della schema di Shamir, l'insieme di partecipanti può recuperare $s_{(c_0,\ldots,c_g)}$. A questo punto, sapendo che $s_{(c_0,\ldots,c_g)} = s_{(c_0,\ldots,c_{g-1})} \oplus r_{(c_0,\ldots,c_g)}$, possiamo recuperare $s_{(c_0,\ldots,c_{g-1})}$ poiché conosciamo $r_{(c_0,\ldots,c_g)}$. Questo procedimento può essere svolto ricorsivamente, recuperando $s_{(c_0,\ldots,c_{g-2})}$, poi $s_{(c_0,\ldots,c_{g-3})}$, etc fino ad arrivare ad s_0, che per definizione è il segreto condiviso.

Per quanto riguarda la sicurezza, invece, va mostrato che un insieme non qualificato non può ottenere informazioni riguardo il segreto condiviso s. Possiamo dimostrarlo per induzione sul numero di generazioni:

1. **Caso base:** Il caso base deriva immediatamente dalla sicurezza dello schema di Shamir.

2. **Ipotesi induttiva:** Assumiamo che lo schema sia sicuro per i partecipanti provenienti da g generazioni e cerchiamo di mostrare che lo sia ancora per i partecipanti provenienti dalle prime g + 1 generazioni.

3. **Passo induttivo:** Lasciamo che il dealer condivida il segreto tra i partecipanti nella prima generazione. Ora, osserviamo che ciò che il dealer fa nella procedura di condivisione rimanente è condividere i segreti di $\mathbf{GenSz}(0)$ tra le rimanenti g generazioni con strutture di accesso leggermente modificate. Cioè, condivide il segreto s_i, per $i \in [\mathbf{GenSz}(0)]$ seguendo la sequenza di threshold dinamico $k_1 - i, k_2 - i, \ldots$. Dobbiamo dimostrare che le parti rimanenti soddisfino ancora la sicurezza in entrambi i due seguenti casi:

   (a) Sono non qualificati nella nuova struttura d'accesso, e quindi sono indipendenti da $s_{(i)}$.

   (b) Sono qualificati, e quindi possono scoprire $s_{(i)}$; ma nonostante ciò grazie alla sicurezza dello schema di Shamir non sono in grado di recuperare s.

   Una terza opzione che viene in mente sarebbe quella in cui non solo sono qualificati, ma possono anche recuperare s, un'opzione che non può occorrere dal momento che l'insieme dal principio non è qualificato.
   A questo punto, se applichiamo l'ipotesi induttiva, otteniamo che gli shares che un nemico potrebbe ottenere da questi schemi non bastano per recuperare il segreto, in quanto indipendenti da esso. Inoltre, la condivisione da parte

del dealer viene effettuata indipendentemente tra queste strutture d'accesso, rendendo qualsiasi combinazione di tutti questi shares indipendenti dal segreto.

Grandezza di uno share. Consideriamo ora la grandezza dello share di un partecipante appartenente alla generazione g: essa consiste di due parti, corrispondenti sopra descritte procedure di condivisione tramite lo schema di Shamir. In particolare, la prima parte sarà di grandezza al più:

$$\prod_{j=1}^{g} \mathbf{GenSz}(j) \cdot \log(\mathbf{GenSz}(g)) = \prod_{j=1}^{g} 2^{2^j} \cdot 2^g = 2^{\sum_{j=1}^{g} 2^j} \cdot 2^g \leq 2^{2^{g+1}} \cdot 2^g$$

La seconda parte, inoltre, anche ha una grandezza di al più:

$$\prod_{j=1}^{g} \mathbf{GenSz}(j) \cdot \log(\mathbf{GenSz}(g)) \leq 2^{2^{g+1}} \cdot 2^g$$

Si può quindi concludere banalmente che la grandezza di uno share è limitata superiormente da $2 \cdot (2^{2^{g+1}} \cdot 2^g)$, ed essendo il t-esimo partecipante nella generazione $g = \lceil \log \log t \rceil$, significa che la grandezza dello share è limitata da $4t^4 \cdot \log t$.

Condividere segreti lunghi. Lo schema descritto sopra può essere generalizzato in modo tale da condividere segreti più lunghi di un singolo bit in maniera più efficiente. Questo possiamo ricavarlo dal fatto che lo schema a soglia di Shamir può essere usato per condividere un segreto più lungo di 1 bit senza aumentare la dimensione degli share. In particolare, lo schema di Shamir permette di condividere un segreto di lunghezza l con azioni di dimensione $max\{l, \log q\}$ (dove $q > n$ è un numero primo ed n è il numero di partecipanti tra i quali condividiamo il segreto). Quindi, anche per i segreti lunghi, per indici $t \in \mathbb{N}$ dei partecipanti abbastanza grandi, applicheremo lo schema di Shamir su un insieme molto grande in modo tale da avere che $maxl, \log q = \log q$ e quindi validare l'analisi di cui sopra. Per i partecipanti con un indice basso, cioè per i quali $max\{l, \log q\} = l$ avremo una grandezza dello share proporzionale ad l.

6.1 Uno schema generale

In questa sezione descriveremo uno schema generalizzato per strutture d'accesso più generali in maniera tale che vengano soddisfatte tre proprietà speciali:

1. I partecipanti possono essere divisi in generazioni di grandezza crescente, nelle quali la grandezza della g-esima generazione viene indicata con **GenSz**(g);

2. In ogni generazione, bisogna cercare di dover ricordare meno informazioni possibili per il futuro;

3. Deve essere possibile mettere insieme tutte le informazioni da generazioni differenti e poter indicare se un insieme sia o meno qualificato.

La struttura d'accesso al tempo $t \in \mathbb{N}$, che indichiamo con A_t, è una funzione di "bit indicatori", i quali rappresentano se ogni partecipante sia o meno presente nel processo di ricostruzione. In pratica, possiamo pensare alla funzione $A_t(x_1, \ldots, x_t)$ come alla funzione indicatrice della struttura d'accesso, nella quale ogni argomento

x_i indica se l'i-esimo partecipante sia presente o meno. Sia X_g l'insieme di partecipanti della g-esima generazione. Associamo, ad ognuna di esse, delle funzioni monotone $\Psi_0^g, \ldots, \Psi_{l_g}^g : \{0,1\}^{|X_g|} \to \{0,1\}$ tali che prendano in input l'indicatore dei partecipanti della generazione, e manda in output un singolo bit; l_g è un parametro noto. In più, per ogni $(c_0, \ldots, c_{g-1}) \in \{0,1\}^{l_0} \times \ldots \times \{0,1\}^{l_{g-1}}$, associamo una funzione monotona $\Phi_{c_0,\ldots,c_{g-1}} : \{0,1\}^{X_g} \to \{0,1\}$ tale che l'indicatore di un insieme di partecipanti $x_1, \ldots, x_t$ (dove la generazione del t-esimo partecipante è g^*) sia qualificato nella struttura d'accesso A_t se e solo se:

$$A_t(x_1, \ldots, x_t) = 1 \iff$$

$$\exists c_0, \ldots, c_{g^*-1} \in [l_0] \times \ldots \times [l_{g^*-1}] : \Phi_{\Psi_{c_0^0}(X_0),\ldots,\Psi_{c_{g^*-1}^{g^*-1}}(X_{g^*-1})}(X_{g^*}) \tag{6.1}$$

Sappiamo che se settiamo ogni Ψ_i' in modo tale che sia la funzione identità che ritorna l'i-esimo bit (quindi che $l_g = \mathbf{GenSz}(g)$), e facendo sì che $\Phi_{c_0,\ldots,c_g}$ corrisponda ad A_t (per un valore appropriato di t), dove l'output di Ψ_i sia fissato per ogni $i \in [g-1]$ tranne che per l'ultima generazione, allora una simile funzione esiste sempre.
Ad esempio, in uno schema a threshold dinamico come quello considerato sopra, possiamo settare ogni Ψ_i in modo da contare quanti partecipanti provengano da quella generazione, quindi $l_i = \mathbf{GenSz}(i)$, e una funzione monotona $\Phi_{l_0,\ldots,l_g}$, che sia definita naturalmente su input $x_1, \ldots, x_{\mathbf{GenSz}(g)}$ in modo tale che controlli, per ogni $j \in [\mathbf{GenSz}(g)]$ se $\sum_{i=0}^{g} l_i + \sum_{i=1}^{j} x_i$ sia almeno tanto grande quanto la soglia desiderata.

Il mapping appena descritto ha senso poiché ora la struttura d'accesso A può essere vista come una composizione di più strutture d'accesso della forma $\Psi_{c_i}^g$ e $\Phi_{c_0,\ldots,c_g}$. Se scegliessimo le generazioni in modo tale che siano abbastanza grandi, ma mantenendo gli l_i il più piccoli possibile, ed in più utilizzando schemi efficienti per le strutture su descritte, possiamo descrivere uno schema efficiente.
Per farlo iniziamo dicendo che il dealer, dopo che la g-esima generazione finisce, ha un insieme di stringhe s_A, con A che varia su ogni tupla $(c_0, \ldots, c_g)$, sapendo che $c_i \in \{0,1\}^{l_i}$. Definiamo poi $s \in \{0,1\}$ il segreto da condividere, e imponiamo che $s_0 = s$. Quando la (g+1)-esima generazione inizia, il dealer esegue le seguenti azioni per ogni $(c_0, \ldots, c_g) \in [l_0] \times \ldots \times [l_g]$:

1. Sfrutta la funzione $\Phi_{c_0,\ldots,c_g}$ per condividere il segreto $s_{c_0,\ldots,c_g}$ tra i partecipanti della (g+1)-esima generazione
2. Per ogni $c_{g+1} \in [l_{g+1}]$
   (a) Genera un $r_{c_0,\ldots,c_{g+1}} \leftarrow \{0,1\}$ in maniera uniformemente casuale
   (b) Sfrutta $\Psi_{c_{g+1}}^{g+1}$ per condividere $r_{c_0,\ldots,c_{g+1}}$ tra i partecipanti della (g+1)-esima generazione
   (c) Poni $s_{c_0,\ldots,c_{g+1}} = s_{c_0,\ldots,c_g} \oplus r_{c_0,\ldots,c_{g+1}}$

Correttezza e sicurezza. La correttezza e la sicurezza dello schema segue, con una dimostrazione simile a quella su correttezza e sicurezza del threshold dinamico, dalla funzione (6.1).

Analisi della grandezza di uno share. Analizziamo ora la grandezza di uno share di un partecipante nella (g+1)-esima generazione. Essa consiste in due partecipanti, corrispondenti alle due procedure di condivisione Φ e Ψ. Presupponiamo che la grandezza dello share di ogni $\Phi_{c_0,\ldots,c_g}$ sia limitata superiormente da $\phi_{c_0,\ldots,c_g}$, e che la grandezza di uno share di ogni $\Psi^g_{c_g}$ sia limitato superiormente invece da $\psi^g_{c_g}$. La prima parte dello schema sopra descritto ha una grandezza massima di:

$$\prod_{c_0 \in [l_0]} \cdots \prod_{c_g \in [l_g]} \phi_{c_0,\ldots,c_g}$$

La seconda parte, invece, ha una grandezza massima al più pari a:

$$\prod_{c_0 [l_0]} \cdots \prod_{c_g \in [l_g]} \prod_{c_{g+1} \in [l_{g+1}]} \psi^g_{c_g}$$

In conclusione, possiamo affermare che la grandezza massima di uno share per il t-esimo partecipante appartenente alla generazione g è limitata superiormente dalla somma dei due termini appena descritti.

Perché generale? Ciò che abbiamo appena descritto cattura non solo lo schema a soglia dinamica descritto nel capitolo 6 e in [KP17], ma anche quello con un threshold pari a k, per valori costanti di k, come descritto nel capitolo 5 e in [KNY16]. Va però specificato che nei singoli casi la scelta della grandezza della generazione deve essere differente. Nel caso generale, infatti, le generazioni vanno prese di grandezza 1, poiché non può essere guadagnato nulla dalla compressione, essendo la struttura completamente arbitraria, mentre nel caso di un threshold pari a k, le generazioni crescono in grandezza con complessità lineare in k piuttosto che polinomiale in t come nel caso della soglia dinamica.

Capitolo 7

Condivisione del segreto evolutiva robusta

In questo capitolo, tratteremo la robustezza, cioè di come rendere un qualsiasi schema con una soglia di k tale che anche se alcuni partecipanti dovessero ricevere degli shares incorretti, potrebbero comunque recuperare il segreto. Questa strategia è stata descritta nella pubblicazione [KP17] da Komargodski e Paskin-Cherniavsky. Per formalizzare questo concetto, viene introdotto un nuovo algoritmo nella struttura d'accesso evolutiva a condivisione del segreto standard, chiamata **R-RECON**, il quale prende in input gli shares di un insieme di partecipanti A dai quali può essere recuperato il segreto.
Assumiamo che ad un avversario sia ammesso rovinare un qualsiasi insieme $B \subseteq A$ tale che $A \backslash B$ sia ancora un insieme qualificato. La struttura di ricostruzione ha successo in ogni caso, tranne con una probabilità di $2^{-\lambda}$, dove λ è un parametro fissato nella procedura di condivisione.

Definizione 7.1. (Condivisione del segreto evolutiva robusta). Uno schema di condivisione del segreto robusto viene definito con tre algoritmi: (**SHARE**, **RECON**, **R-RECON**). L'insieme formato dalle due procedure (**SHARE**, **RECON**) forma uno schema di condivisione del segreto evolutivo come visto nella sezione 2.2.2., con la differenza che l'algoritmo di **SHARE** prende in più in input 1^{λ}, dove λ è un parametro di sicurezza. La procedura **R-RECON** segue i seguenti requisiti:

Ricostruzione robusta (Robust-RECONstruction.): Partendo dal presupposto che il segreto sia stato condiviso utilizzando l'algoritmo di **SHARE**$(1^{\lambda}, s)$, prendiamo in considerazione l'ipotesi nella quale un nemico N scelga un istante di tempo t e due sottoinsiemi di partecipanti $A, B \subseteq [t]$ tali che:
(1) $B \subseteq A$,
(2) B sia non qualificato, e
(3)$A \backslash B$ sia qualificato.
Al nemico N vengono, a questo punto, dati gli shares dei partecipanti di B, che indicheremo con Π_B^s, e supponiamo li cambi arbitrariamente in modo da ottenere $\Pi_B^{s\prime}$. In output verrà mandato il valore di $s' =$ **R-RECON**$(1^{\lambda}, \Pi_A^s \cup \Pi_B^{s\prime})$.
Definiamo uno schema λ-robusto se per ogni nemico N si può affermare che:

$$\mathbf{Pr}[s' \neq s] \leq 2^{-\lambda}$$

Con il seguente Teorema viene definito un modo per ottenere uno schema robusto per una struttura d'accesso evolutiva con una soglia di k nel quale gli insiemi

qualificati sono quelli con grandezza di almeno k.

Teorema 7.2. Siano $k \in \mathbb{N}$ e $\lambda > 0$. Assumiamo esista una famiglia di schemi evolutivi e lineari con una soglia di k tali che preso un dominio di segreti S_D, sia ancora lineare rispetto al campo $\mathbb{F} = \mathbb{F}_2^t$, dove $t \geq \log |S_D|$. Sapendo ciò, dimostriamo che esiste uno schema λ-robusto di condivisione del segreto per strutture d'accesso evolutive con threshold pari a k. L'overhead nella condivisione per il partecipante t in relazione alla dimensione della condivisione dello schema originale è un fattore additivo di $O(\lambda + k - \log k)$ bit (per un dominio S sufficientemente grande, altrimenti l'overhead è moltiplicativo).

Idea generale della dimostrazione. Per dimostrare il Teorema 7.2. adattiamo il classico schema di condivisione del segreto robusto descritto in [CDF^+08] a delle impostazioni evolutive. A questo punto, sfruttiamo lo schema descritto precedentemente nel capitolo 5 per strutture d'accesso evolutive con soglia k e trasformiamolo in uno robusto; Ciò che abbiamo precedentemente descritto non lo è poiché si può osservare che un problema è che il campo su cui operano le varie istanze di Shamir cresce man mano che arrivano più parti. Usando campi di estensione, gli shares possono essere visti come un vettore di combinazioni lineari su un singolo campo $\mathbb{F}_2^t$ di dimensione adatta, e la dimostrazione si svolge in modo simile. L'idea ad alto livello della costruzione è quella secondo la quale, invece di condividere il segreto stesso, vengono condivisi gli shares della codifica AMD del segreto, come descritto nella definizione 2.2.8. Lo schema risultante è robusto proprio perché i codice AMD proteggono l'informazione contro attacchi aggiuntivi, e questo schema di condivisione del segreto è lineare.

Dimostrazione Teorema 7.2. Assumiamo di avere uno schema $\varepsilon = (\mathbf{SHARE}, \mathbf{RECON})$ per una struttura di accesso con un threshold k, e la trasformiamo in uno schema evolutivo robusto per quella stessa struttura. Prendiamo i segreti dal dominio S_D. Sfruttiamo, come già detto, la costruzione di tale struttura descritta nella pubblicazione [KNY16] e descritta nel capitolo 5, su uno spazio di segreti abbastanza grande. La grandezza di uno share per il t-esimo partecipante in questo tipo di schema è limitato superiormente da $\sigma(t) = kt \log kt$ bits. Fissiamo quindi un codice AMD con funzioni di codifica e decodifica (E, D) e con un parametro d'errore $\gamma' = (\lambda + k \log k)$ su un dominio di segreti con cardinalità $|S_D|$. Concretamente, utilizzeremo quindi il codice $AMD_{\sigma,\gamma'}$.

Descriviamo quindi il nuovo schema di condivisione del segreto robusto:

1. La nuova procedura di condivisione **SHARE'**$(1^\lambda, \Pi_1^s, \ldots, \Pi_{t-1}^s, s)$ prende in input un parametro di robustezza, 1^λ, gli shares dei partecipanti 1,...,t-1 e il segreto. Da qui, genera gli shares per il t-esimo partecipante come segue: all'inizio della procedura di condivisione, prima che arrivasse il primo partecipante, il dealer computa una codifica AMD di s, che chiameremo $\hat{s} = E(s)$, e la sfrutta in questo schema, in particolare lanciando l'algoritmo **SHARE**$(\Pi_1^s, \ldots, \Pi_{t-1}^s, \hat{s})$ e dando al t-esimo partecipante questo valore.

2. La procedura di ricostruzione **RECON'**(Π_B^s, B) che prende in input gli shares di un sottoinsieme di partecipanti B, e applica la procedura di ricostruzione originale dello schema, cioè **RECON**(Π_B^s, B) per ottenere una codifica AMD $\hat{s}$. Poi, produce la decodifica AMD di questo valore $s = D(\hat{s})$.

3. L'algoritmo di ricostruzione robusto **R-RECON**$(1^{\lambda,\Pi^s_B,B})$, che prende in input un parametro di robustezza 1^λ e gli shares di un insieme di partecipanti B funziona come segue: sia B' un insieme che denota i primi $min\{2k-1,|B|\}$ partecipanti di B. Si scorrano tutti i mintermini $T \subseteq B'$, degli insiemi di cardinalità pari a k, e si applichi la procedura di ricostruzione su ciascuno di essi ($\hat{s}_T = \textbf{RECON'}(1^\lambda, \Pi^s_T, T)$). Se ogni $\hat{s}$ è $\perp$, si ritorni $\perp$. Altrimenti, ritornare il primo valore diverso da $\perp$. Notiamo che dal momento che k è costante, la complessità computazionale di questo algoritmo è polinomiale rispetto alla grandezza dell'input.

Correttezza, sicurezza e robustezza. Per quanto riguarda i primi due, basta semplicemente notare che sono stati utilizzati lo schema evolutivo con soglia k, che abbiamo già dimostrato essere corretto e sicuro, e lo schema AMD, che ancora soddisfa sia perfetta correttezza che segretezza.
La robustezza dell'implementazione va dimostrata. Prima di tutto osserviamo che $|B'| \leq 2k-1$, e che deve contenere un sottoinsieme T' nel quale non abbiamo partecipanti malevoli. Inoltre, se $B' = B$, questo segue dalle possibili parti maligne che l'avversario è autorizzato a scegliere (altrimenti, l'avversario ha scelto un insieme qualificato, il che è illegale). Se invece $|B\prime| = 2k-1$, allora l'insieme di partecipanti non malevoli è un sottoinsieme di grandezza k, il che lo rende qualificato.
Ora, dimostriamo che l'algoritmo di ricostruzione robusto **R-RECON** ritorna il segreto condiviso s con una probabilità pari almeno a $1-2^{-\lambda}$. Sfruttando la correttezza degli schemi AMD, affermiamo che $\hat{s}'_{T'} = s$. Rimane quindi da mostrare che per ogni altro mintermine T, vale che $\hat{s}_T \in \{s, \perp\}$ con una probabilità di $1-2^{-\lambda}$. Per dimostrarlo, in [KP17] si sono basati su una dimostrazione simile vista in [CDF^+08].
Per ogni mintermine T, consideriamo un qualsiasi possibile shift Δ_T tra gli shares scelti dal nemico. Questo shift naturalmente ne corrisponde ad uno aggiuntivo sull'insieme totale degli shares utilizzati per la ricostruzione, e quindi sul valore condiviso (dal momento che lo schema evolutivo con k-soglia è lineare).
Data la sicurezza dello schema di condivisione del segreto, gli shares dei partecipanti che l'avversario controlla non dipendono da $\hat{s}$. Quindi la distribuzione degli shares shiftati è ancora indipendente dal segreto $\hat{s}$. Ora, data la sicurezza dei codici AMD, sappiamo che la probabilità che $\hat{s}_T \notin \{s, \perp\}$ è al più pari a $2^{-\lambda+k\cdot(\log k+1)}$. Dal momento che ci sono al più $\binom{|B'|}{k} \leq \binom{2k-1}{k}$ diversi possibili insiemi diversi di mintermini T, si può applicare un limite di unione per ottenere che la probabilità che ciò accada per qualche $\hat{s}_T$ è al più:

$$(2k)^k \cdot 2^{-\lambda+k\cdot(\log k+1)} = 2^{k\cdot(\log k+1)} \cdot 2^{-\lambda+k\cdot(\log k+1)} \leq 2^{-\lambda}$$

E qui si conclude la dimostrazione.

Capitolo 8

Implementazione Shamir's Secret Sharing ed Evolving Secret Sharing

Ho implementato in Python 3 due delle funzioni descritte nel paper precedentemente, in particolare:

1. Shamir's Secret Sharing
   [https://github.com/Harachili/Tesi-triennale-ESS/blob/main/shamirScheme.py]
2. Evolving Secret Sharing
   [https://github.com/Harachili/Tesi-triennale-ESS/blob/main/evolvingSS.py]

8.1 Shamir's Secret Sharing

```
#shamirScheme.py
from __future__ import division
import random
import functools
from timeit import timeit
from Crypto.Util.number import bytes_to_long, long_to_bytes
import ast


# (per questa applicazione vogliamo un numero PRIME noto).
# il piu' vicino possibile al nostro livello di sicurezza;
# ad es. con il livello di sicurezza desiderato di 128 bit:
# utilizziamo il 12-esimo PRIME di Mersenne
# Se prendessimo un PRIME troppo grande, tutto il testo cifrato sarebbe
    troppo grande.
# Se lo prendessimo troppo piccolo, invece, la sicurezza sarebbe
    compromessa)

PRIME = 2**127 - 1 # 12-esimo PRIME di Marsenne

# Il 13-esimo PRIME di Mersenne e' 2**521 - 1

interorandom = functools.partial(random.SystemRandom().randint, 0)
prod = lambda x, y: x * y
```

```
def eval_at(poly, x, prime):
    '''
    Calcola il polinomio in x
    '''
    # print(poly)
    accum = 0
    for coeff in reversed(poly):
        accum *= x
        accum += coeff
        accum %= prime
    return accum


def create_shares_from_secret(secret, minimum, nShares, prime=PRIME):
    '''
    Divide il segreto in "nShares" shares, con un threshold
    di "minimum" shares necessari per recuperare il segreto iniziale
    '''
    # poly = [secret]
    # print(poly)
    if minimum > nShares:
        raise ValueError("Il segreto sarebbe irrecuperabile se dovessero
            servire piu' shares di quanti ne esistano effettivamente.")
    poly = [secret] + [interorandom(prime) for i in range(minimum-1)]
    # poly = [secret] + poly
    points = [(i, eval_at(poly, i, prime)) for i in range(1, nShares + 1)]
    # print(poly[0], points)
    return points

# divisione di interi modulo p, significa trovare l'inverso del
    denominatore
# modulo p e poi moltiplicare il numeratore per il suo inverso
# ad esempio: l'inverso di A e' quel B tale che A*B % p == 1
# Per calcolarlo utilizzo l'algoritmo esteso di Euclide
# http://en.wikipedia.org/wiki/Modular_multiplicative_inverse#Computation
# Per l'implementazione mi sono ispirato a
    https://github.com/lapets/egcd/blob/main/egcd/egcd.py
def extended_gcd(a, b):
    x = 0
    last_x = 1
    y = 1
    last_y = 0
    while b != 0:
        quot = a // b
        a, b = b, a%b
        x, last_x = last_x - quot * x, x
        y, last_y = last_y - quot * y, y
    return last_x, last_y


def divmod(num, den, p):
    '''
    Calcola num / den modulo p numero PRIME, cioe'
    ritorno il valore tale che renda vera la seguente uguaglianza:
    den * divmod(num, den, p) % p == num
    '''
```

```
    inv, _ = extended_gcd(den, p)
    return num * inv


def lagrange_interpolation(x, x_s, y_s, p):
    '''
    Trovare il valore di y per la data x, dati n punti composti dalla
        coppia (x, y);
    k punti serviranno a definire un polinomio fino al k-esimo ordine
    '''
    k = len(x_s)
    assert k == len(set(x_s)), "I punti devono essere distinti"
    PI = lambda vals: functools.reduce(prod, vals, 1) # PI -- Prodotto
        elementi di una lista
    # def PI(vals): return functools.reduce(prod, vals, 1)
    nums = [] # per evitare divisioni non precise
    dens = []
    for i in range(k):
        others = list(x_s)
        cur = others.pop(i)
        nums.append(PI(x - o for o in others))
        dens.append(PI(cur - o for o in others))
    den = PI(dens)
    num = sum([divmod(nums[i] * den * y_s[i] % p, dens[i], p)
               for i in range(k)])
    return (divmod(num, den, p) + p) % p


def recover_secret(shares, prime=PRIME):
    '''
    Recupera il segreto dalle coppie di punti (x, y) giacenti sul polinomio
    '''
    if len(shares) < 2:
        raise ValueError("Sono necessari almeno due shares");
    x_s, y_s = zip(*shares);
    return lagrange_interpolation(0, x_s, y_s, prime);


def main():
    b = input("Cosa vuoi fare? [E]ncrypt, [D]ecrypt:\n")
    if b.lower() == "e":
        secretString = input("Inserire il segreto: ");
        secret = bytes_to_long(secretString.encode());
        # print(secret)
        nShares = int(input("Inserire il numero di shares che si vogliono:
            "));
        minimum = int(input("Inserire il numero minimo di shares per
            recuperare il segreto (almeno 2): "));
        shares = create_shares_from_secret(secret, minimum, nShares);
        time = timeit(lambda: create_shares_from_secret(secret, minimum,
            nShares), number=1);
        print(shares);
        print(f"Tempo totale per crittare: {time}")
        print(recover_secret(random.sample(shares, minimum)));
        assert recover_secret(random.sample(shares, minimum)) == secret,
            "No";
        assert recover_secret(shares) == secret, "Nope";
    elif b.lower() == "d":
```

```
        inpShares = input("Inserire una lista di almeno 'minimum' shares da
            decrittare:\n");
        shares = ast.literal_eval(inpShares);
        print("Se cio' che hai inserito e' corretto, il plaintext e': ");
        try:
            print(long_to_bytes(recover_secret(shares)).decode());
        except:
            print("Servono piu' shares per ricostruire il segreto!");


if __name__== "__main__":
    main()
```

8.2 Evolving Secret Sharing

```
#evolvingSecretSharing.py
from __future__ import division
import random
import functools
from timeit import timeit
from Crypto.Util.number import bytes_to_long, long_to_bytes
import ast


# (per questa applicazione vogliamo un numero PRIME noto).
# il piu' vicino possibile al nostro livello di sicurezza;
# ad es. con il livello di sicurezza desiderato di 128 bit:
# utilizziamo il 12-esimo PRIME di Mersenne
# Se prendessimo un PRIME troppo grande, tutto il testo cifrato sarebbe
    troppo grande.
# Se lo prendessimo troppo piccolo, invece, la sicurezza sarebbe
    compromessa)

PRIME = 2**127 - 1 # 12-esimo PRIME di Marsenne

# Il 13-esimo PRIME di Mersenne e' 2**521 - 1

interorandom = functools.partial(random.SystemRandom().randint, 0)
prod = lambda x, y: x * y


def eval_at(poly, x, prime):
    '''
    Calcola il polinomio in x
    '''
    # print(poly)
    accum = 0
    for coeff in reversed(poly):
        accum *= x
        accum += coeff
        accum %= prime
    return accum


def create_poly(secret, minimum):
```

```
    print(secret)
    return [secret] + [interorandom(PRIME) for i in range(minimum-1)]

def create_shares_from_secret(poly, nShares, prime=PRIME):
    '''
    Divide il segreto in "nShares" shares, con un threshold
    di "minimum" shares necessari per recuperare il segreto iniziale
    '''
    # poly = [secret]
    # print(poly)
    # if minimum > nShares:
    #   raise ValueError("Il segreto sarebbe irrecuperabile se dovessero
        servire piu' shares di quanti ne esistano effettivamente.")

    # poly = [secret] + poly
    points = [(i, eval_at(poly, i, prime)) for i in range(1, nShares + 1)]
    # print(poly[0], points)
    return points

# divisione di interi modulo p, significa trovare l'inverso del
    denominatore
# modulo p e poi moltiplicare il numeratore per il suo inverso
# ad esempio: l'inverso di A e' quel B tale che A*B % p == 1
# Per calcolarlo utilizzo l'algoritmo esteso di Euclide
# http://en.wikipedia.org/wiki/Modular_multiplicative_inverse#Computation
# Per l'implementazione mi sono ispirato a
    https://github.com/lapets/egcd/blob/main/egcd/egcd.py
def extended_gcd(a, b):
    x = 0
    last_x = 1
    y = 1
    last_y = 0
    while b != 0:
        quot = a // b
        a, b = b, a%b
        x, last_x = last_x - quot * x, x
        y, last_y = last_y - quot * y, y
    return last_x, last_y


def divmod(num, den, p):
    '''
    Calcola num / den modulo p numero PRIME, cioe'
    ritorno il valore tale che renda vera la seguente uguaglianza:
    den * divmod(num, den, p) % p == num
    '''
    inv, _ = extended_gcd(den, p)
    return num * inv


def lagrange_interpolation(x, x_s, y_s, p):
    '''
    Trovare il valore di y per la data x, dati n punti composti dalla
        coppia (x, y);
    k punti serviranno a definire un polinomio fino al k-esimo ordine
    '''
    k = len(x_s)
    assert k == len(set(x_s)), "I punti devono essere distinti"
```

```
    PI = lambda vals: functools.reduce(prod, vals, 1) # PI -- Prodotto
        elementi di una lista
    # def PI(vals): return functools.reduce(prod, vals, 1)
    nums = [] # per evitare divisioni non precise
    dens = []
    for i in range(k):
        others = list(x_s)
        cur = others.pop(i)
        nums.append(PI(x - o for o in others))
        dens.append(PI(cur - o for o in others))
    den = PI(dens)
    num = sum([divmod(nums[i] * den * y_s[i] % p, dens[i], p)
               for i in range(k)])
    return (divmod(num, den, p) + p) % p


def recover_secret(shares, prime=PRIME):
    '''
    Recupera il segreto dalle coppie di punti (x, y) giacenti sul polinomio
    '''
    if len(shares) < 2:
        raise ValueError("Sono necessari almeno due shares");
    x_s, y_s = zip(*shares);
    return lagrange_interpolation(0, x_s, y_s, prime);


def main():
    totTime = 0;
    d = dict();
    counter = 1
    toInsToDecr = []
    b = input("Cosa vuoi fare? [E]ncrypt, [D]ecrypt:\n")
    if b.lower() == "e":
        secretString = input("Inserire il segreto da crittare: ");
        secret = bytes_to_long(secretString.encode());
        minimum = int(input("Inserire qui la soglia minima di shares di cui
            si deve avere bisogno per ricostruire il segreto: "));
        poly = create_poly(secret, minimum);
        while True:
            print(f"Benvenuto partecipante n. {counter}");
            point = create_shares_from_secret(poly, counter)[-1];
            time = timeit(lambda: create_shares_from_secret(poly,
                counter)[-1], number=1);
            print(f"Lo share per il partecipante n. {counter} e': {point}");
            print(f"Il tempo impiegato per creare lo share, in
                microsecondi, e': {time}");
            totTime+=time;
            d[point[0]] = point[1];
            counter += 1;
            ans = input("Desideri ricevere un altro share? [Y]|[N]\n");

            if ans.lower() == "n":
                if counter < minimum: raise ValueError("Non hai richiesto
                    abbastanza shares, il segreto e' perso per sempre!")
                for i in range(1, counter):
                    toInsToDecr.append((i, d[i]))
                break;
        print(toInsToDecr)
```

```
        print(f"Tempo totale impiegato nel crittare: {totTime}") # 150
            encryptions, con soglia 15: 0.16363659693161026 secondi

    elif b.lower() == "d":
        inpShares = input("Inserire una lista con un numero di elementi
            almeno pari al numero minimo di shares necessari per
            decrittare:\n");
        shares = ast.literal_eval(inpShares);
        try:
            print("Se cio' che hai inserito e' corretto, il plaintext e':
                ");
            print(long_to_bytes(recover_secret(shares)).decode());
        except UnicodeDecodeError:
            print("Non hai inserito abbastanza shares per ricostruire il
                segreto!")


if __name__== "__main__":
    main()
```

Gli algoritmi da me implementati sono il secret sharing di Shamir (SSS), che sfrutta l'interpolazione di Lagrange come descritta nel capitolo 2.1.1 e l'evolving secret sharing (ESS) che, da mia implementazione, utilizza ancora lo schema di condivisione del segreto di Shamir ma, arrivando i partecipanti uno per volta, posso dividerli in generazioni, come descritto precedentemente e fare in modo che la grandezza degli shares vari in base al loro arrivo.

Testing			
chars and threshold	Secret Sharing time	Evolving Secret Sharing time	Rapporto ESS/SS
1 and (8, 3)	0.0001468900009058416	0.0003248950015404262	2.211825172148294
1 and (10, 4)	0.0001356089996988885	0.0003544329956639558	2.613639186565439
1 and (15, 7)	0.0003656070002762135	0.0011853360010718461	3.242104227151923
1 and (23, 13)	0.000396353003452532	0.003920939001545776	9.892542676330079
4 and (8, 3)	0.0001572479959577322	0.0004355350174591876	2.769733342587451
4 and (10, 4)	0.0001653709987294860	0.0003919660011888481	2.370222132056095
4 and (15, 7)	0.0003449729993008077	0.0011097449951193994	3.216903923984294
4 and (23, 13)	0.0005766430040239356	0.0037232520116958767	6.456771322489382
16 and (8, 3)	0.0001117869978770613	0.0003153010038658976	2.820551672857807
16 and (10, 4)	0.0001681630019447766	0.0004790980092366226	2.84900961386235
16 and (15, 7)	0.0003610919993661809	0.001227837999977055	3.400346731947147
16 and (23, 13)	0.00079077100235736	0.004249283017998096	5.373594890721332

Da questa serie di test, possiamo notare come non è la grandezza del segreto ad incidere sulla performance dell'algoritmo dell'evolving secret sharing, quanto invece il numero di shares:

Nell'algoritmo di Shamir il tempo di esecuzione incrementa logaritmicamente all'aumento del numero di share desiderati; Nella mia implementazione dell'evolving secret sharing, invece, l'incremento è lineare, dato che non potendo sapere a priori su quanti shares verrà eseguito l'algoritmo non è possibile ottimizzarlo.

In compenso, mentre nell'algoritmo di Shamir tutti gli shares creati hanno una lunghezza standard, nell'implementazione dell'ESS la dimensione varia in base alla grandezza dello share, in particolare, nonostante di default usi l'algoritmo di Shamir,

si può anche fare in modo di sfruttare i codici prefisso per arrivare ad avere una grandezza della codifica pari al $2 \log \log t$, con t=dim(share) [come da corollario 4.2] Questa implementazione (con i codici prefisso) non è stata attualmente implementata, poiché avendo lavorato in python ho avuto problemi nell'impossibilità della gestione della memoria; infatti un altro limite importante dell'implementazione è il segreto, che se più lungo di 16 caratteri genera un errore di overflow, che con una migliore gestione di memoria non sarebbe scaturito. Si intende, infatti, continuare a lavorare al codice sviluppandolo in linguaggi più memory-friendly, per migliorare performance e usabilità.

Infine vediamo come varia l'esecuzione in base alla grandezza del segreto:
Nel SS, così come nell'ESS, la grandezza del segreto non incide sulla complessità dell'algoritmo, essendone l'algoritmo di calcolo degli shares indipendente. Questo accade poiché il segreto viene trasformato in intero e posto come coefficiente della curva, e una volta impostato, la sua grandezza non incrementa la complessità dei calcoli.

Capitolo 9

Conclusione

Giunti a questo punto, il lettore dovrebbe essere ormai consapevole dei possibili molteplici utilizzi dell'evolving secret sharing, di come possa essere usato in vari ambiti sia pratici, tramite varie possibili implementazioni, che teoretici, prettamente legati quindi alla dimostrazione di sicurezza di un sistema.

L'obiettivo principale che si vuole conseguire ora, oltre al già citato miglioramento dell'algoritmo, è quello di dare una definizione di non-malleabilità nell'evolving secret sharing, necessario per impedire ad un attaccante di ricavare informazioni trasformando un testo cifrato di un messaggio m in un altro ciphertext f(m), per una funzione conosciuta f, senza necessariamente conoscere o imparare nulla su m.

Ritengo di aver svolto un lavoro pedissequo sul concetto di evolving secret sharing, studiando a fondo i paper già esistenti e mettendone insieme le informazioni completandole. Spero che ciò possa essere utile, seppur in minima parte, alla ricerca sull'argomento, che potrebbe contribuire ad un'innovazione tecnologica nell'ambito della sicurezza informatica.

Bibliografia

[1] I. Komargodski, M. Naor, and E. Yogev, "How to share a secret, infinitely," Cryptology ePrint Archive, Paper 2016/194, 2016, https://eprint.iacr.org/2016/194. [Online]. Available: https://eprint.iacr.org/2016/194

[2] I. Komargodski and A. Paskin-Cherniavsky, "Evolving secret sharing: Dynamic thresholds and robustness," Cryptology ePrint Archive, Paper 2017/948, 2017, https://eprint.iacr.org/2017/948. [Online]. Available: https://eprint.iacr.org/2017/948

[3] V. Goyal and A. Kumar, "Non-malleable secret sharing," Cryptology ePrint Archive, Paper 2018/316, 2018, https://eprint.iacr.org/2018/316. [Online]. Available: https://eprint.iacr.org/2018/316

[4] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[5] G. R. BLAKLEY, "Safeguarding cryptographic keys," in *1979 International Workshop on Managing Requirements Knowledge (MARK)*, 1979, pp. 313–318.

[6] Wikipedia, "Secret sharing — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Secret%20sharing&oldid=1096933559, 2022, [Online; accessed 30-August-2022].

[7] H. Krawczyk, "Secret sharing made short," in *Advances in Cryptology — CRYPTO' 93*, D. R. Stinson, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 136–146.

[8] Wikipedia, "Lagrange polynomial — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Lagrange%20polynomial&oldid=1096433409, 2022, [Online; accessed 30-August-2022].

[1] [2] [3] [4] [5] [6] [7] [8]