ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria e Scienze Informatiche

RUOLO DELLE FUNZIONI HASH NELLA SICUREZZA DELLE PASSWORD

Relatore:

Prof. Luciano Margara

Presentata da:

Alex Mazzotti

III SESSIONE

Anno Accademico 2022-2023

Fai della tua vita un sogno, e di un sogno una realtà...
Antoine de Saint-Exupery

Indice

1 Introduzione **4**
1.1 Contesto e Necessità 4
1.2 L'importanza delle funzioni Hash 5

2 Fondamenti di sicurezza delle password **7**
2.1 Rischi Legati alle Password 7

3 Concetti di Funzioni Hash **10**
3.1 Crittografia e contesto 10
3.2 Definizione e proprietà 12
3.3 Sicurezza delle funzioni Hash 13
3.3.1 Paradosso del Compleanno 14
3.3.2 Random Oracle Model 15

4 Archiviazione e gestione delle password **17**
4.1 Hashing delle password 17
4.2 Salt 18
4.3 Registrazione di un utente 20
4.4 Login di utente 21

5 Funzioni Hash **22**
5.1 Famiglia SHA 22
5.2 Famiglia RIPEMD 23
5.3 Famiglia MD 24

6 Analisi degli algoritmi delle funzioni Hash **25**
6.1 Costruzione iterativa 25
6.2 Costruzione di Merkle-Damgård 27
6.3 Funzioni one-way 28
6.4 Costruzione Sponge 29
6.5 Implementazione di SHA-3 32

7 Funzioni Hash avanzate **34**
7.1 Esempio di Crypt 34
7.2 Bcrypt 35
7.3 Analisi Bcrypt 35
7.4 Argon2 39
7.5 Analisi Argon2 40

8 Best Practice per sicurezza delle password **44**
8.1 Complessità delle password 44
8.2 Cambio periodico delle password 44
8.3 Controlli automatici sulle password 45
8.4 Autenticazione a due fattori - 2FA 45

9 Attacchi alle funzioni Hash - crittoanalisi **47**
9.1 Violare una funzione Hash 47
9.2 Attacco di forza bruta 48
9.2.1 Utilizzo di GPU negli attacchi di forza bruta 48
9.3 Attacco di Rainbow Table 48
9.4 Attacchi di dizionario 49
9.5 Attacchi Side-Channel 50
9.6 Raccomandazioni per soluzioni sicure 51

10 Sfide e sviluppi futuri **52**
10.1 Tendenze e sviluppi recenti 52
10.2 Blockchain e funzioni Hash 52
10.3 Computer quantistici e funzioni Hash 53
10.4 Intelligenza Artificiale applicata alle funzioni Hash 53

11 Conclusioni **55**

Bibliografia **57**

Capitolo 1

Introduzione

Nell'era digitale, l'incremento esponenziale dell'utilizzo di servizi online ha portato a una vasta circolazione di dati sensibili, che spaziano dalle informazioni personali a dati aziendali e finanziari. Questa rapida evoluzione verso un mondo sempre più digitalizzato ha generato una crescita di interconnessioni tra persone, ma ha esposto una serie di vulnerabilità che richiedono attenzione e soluzioni sofisticate. In questo scenario, la sicurezza informatica e la protezione delle password giocano un ruolo fondamentale, più nello specifico per prevenire accessi non autorizzati e violazioni della privacy.

La tesi si propone di esplorare in dettaglio il ruolo delle funzioni Hash nella sicurezza delle password, analizzando il loro funzionamento, le tipologie di funzioni Hash e un'analisi su quali utilizzare in un contesto di sicurezza delle password. Inoltre è affrontato il tema sulle best practise nell'uso delle password, le sfide attuali e possibili sviluppi futuri.

1.1 Contesto e Necessità

Le password sono ampiamente utilizzate in vari aspetti della nostra vita digitale, che vanno dall'accesso a e-mail e social network fino alle operazioni bancarie online e all'accesso a informazioni aziendali. Tuttavia, la debolezza delle password è spesso sfruttata da malintenzionati, in quanto gli utenti tendono a creare password deboli o a conservarle in modo non sicuro. Di conseguenza, i criminali informatici cercano costantemente nuovi metodi per violare gli account, utilizzando attacchi di forza bruta, attacchi di dizionario e altre tecniche di hacking.

1.2 L'importanza delle funzioni Hash

Le funzioni Hash svolgono un ruolo cruciale nella gestione delle password e nell'integrità dei dati. Di seguito le principali applicazioni in cui sono utilizzate:

1. **Protezione delle password:**

   Le funzioni Hash consentono di proteggere le password degli utenti. Quando un utente crea o cambia una password, questa viene passata attraverso una funzione Hash che la trasforma in una stringa di caratteri pseudocasuale. Questa stringa Hash è ciò che viene effettivamente memorizzato nei database. La password originale non è mai memorizzata in forma leggibile. Ciò rende molto difficile per gli aggressori che riescono ad accedere ai dati del database ottenere le password reali.

2. **Sicurezza durante la trasmissione:**

   Le funzioni Hash sono utilizzate anche per garantire la sicurezza durante la trasmissione dei dati. Ad esempio, quando accedi a un sito web sicuro (HTTPS), le informazioni scambiate tra il tuo browser e il server vengono crittografate e verificate tramite funzioni Hash. Questo garantisce che i dati non possano essere intercettati o alterati durante la trasmissione.

3. **Integrità dei dati:**

   Le funzioni Hash sono utilizzate per verificare l'integrità dei dati, cioè per verificare se un file è stato modificato o meno.
   Ad esempio, i file scaricati da Internet spesso vengono accompagnati da un valore Hash. Dopo il download, puoi calcolare il valore Hash del file e confrontarlo con quello fornito. Se i valori Hash corrispondono, puoi essere sicuro che il file non è stato alterato durante il download.

4. **Confronto delle credenziali:**

   Durante l'autenticazione, le funzioni Hash sono utilizzate per confrontare le credenziali inserite dall'utente con quelle memorizzate nei database. L'Hash della password inserita viene confrontato con l'Hash memorizzato per quella specifica password nell'archivio. Se corrispondono, l'utente viene autenticato.

5. **Firma digitale:**
   Nell'ambito della firma digitale le funzioni Hash svolgono un ruolo fondamentale, in quanto consentono di verificare se un messaggio è stato alterato o meno creando una vera e propria firma digitale.
   Il processo inizia con l'esecuzione dell'Hash del messaggio da parte del mittente. Questo Hash viene crittografato utilizzando la chiave privata del mittente, ottenendo così la firma.

Successivamente, sia il messaggio originale che la firma vengono inviati al destinatario.
Il destinatario utilizza la chiave pubblica del mittente per decrittografare la firma e ottenere l'Hash del messaggio originale. Dopodiché, utilizza la stessa funzione di Hash per calcolare l'Hash del messaggio ricevuto.
Infine, viene confrontato l'Hash del messaggio originale con quello ottenuto dalla decrittografia della firma. Se i due Hash coincidono, significa che il messaggio non è stato alterato e la firma è considerata valida.

In breve, le funzioni Hash sono una componente fondamentale della sicurezza informatica, contribuendo in modo significativo alla protezione delle password e dei dati sensibili. Tuttavia, è importante notare che la sicurezza non si basa solo sulle funzioni Hash, ma anche su altre pratiche come l'uso di password complesse, la gestione sicura delle chiavi e la protezione dei dati.

Capitolo 2

Fondamenti di sicurezza delle password

In questo primo capitolo affronteremo il tema delle password e dei rischi dovuti ad uso e una conservazione impropria di queste. Le parola "password", traducibile dall'inglese come "parola d'accesso" consiste in una sequenza di caratteri alfanumerici. Questa sequenza è associata a un nome utente univoco o a un indirizzo email, e consente all'utente corrispondente di accedere a un sistema informatico.
L'importanza delle password è di fondamentale rilievo perché sono il principale mezzo di difesa contro l'accesso non autorizzato e la violazione di privacy.

2.1 Rischi Legati alle Password

Una scelta di una password debole, la poca attenzione nel nasconderla e un uso improprio comporta una serie di rischi significativi. Di seguito le principali cause che aumentano il rischio di una violazione dei dati personali:

1. **Password deboli**
   Per password deboli si intendono quelle password composte da pochi caratteri, o password semplici e facile da indovinare. Per esempio una password debole è "qwerty", "123456" oppure "password".
   In questo caso, gli aggressori fanno uso di dizionari, strumenti con i quali cercano di accedere ai sistemi inserendo password comunemente utilizzate.

2. **Password riciclate**
   Le password riciclate sono password che vengono riutilizzate dalla stessa persona su più sistemi o servizi. Questo comportamento, sebbene possa sembrare conveniente, espone gli utenti a rischi considerevoli.

Se un aggressore viola una singola password, avrà accesso a tutti i sistemi o account in cui quella stessa password è utilizzata. Questa pratica rende gli utenti estremamente vulnerabili, poiché una singola violazione può mettere a repentaglio una serie di risorse digitali e dati sensibili.

3. **Password non conservate/gestite in modo non sicuro**
   Si intende una poca attenzione da parte dell'utente nel conservare le proprie password. Per esempio la scrittura su note fisiche o fogli di carta, i quali possono essere smarriti e letti da chiunque.

4. **Mancanza di cambio periodico delle password**
   Per incrementare la sicurezza dei propri dati è necessario e consigliato cambiare le proprie password periodicamente.

La violazione delle password costituisce una minaccia significativa per la sicurezza delle informazioni personali. Quando le password vengono compromesse, si aprono diverse porte agli attacchi informatici, esponendo utenti e organizzazioni a vari rischi.
Di seguito esempi di rischi dovuti a violazioni delle password.

1. **Accesso non autorizzato:** La violazione delle password consente agli attaccanti di ottenere accesso non autorizzato a sistemi, account email, social media e altri servizi online. Una volta penetrati, gli aggressori possono esplorare e manipolare dati sensibili, compromettendo la privacy degli utenti.

2. **Furto di identità:** Le password, spesso, costituiscono il principale mezzo di autenticazione per l'identità digitale di un individuo. Se compromesse, le informazioni personali possono essere sfruttate per compiere atti fraudolenti, come transazioni finanziarie non autorizzate o l'apertura di nuovi account a nome della vittima.

3. **Perdita di dati aziendali sensibili:** Nel contesto aziendale, la violazione delle password può portare alla compromissione di dati cruciali, quali informazioni strategiche, piani di sviluppo, segreti commerciali e dati finanziari. La perdita di tali informazioni può avere conseguenze gravi sull'integrità e sulla reputazione dell'azienda.

4. **Diffusione di malware:** Le violazioni delle password spesso coincidono con l'iniezione di malware nei sistemi compromessi. Questo software dannoso può essere utilizzato per monitorare attività, rubare ulteriori credenziali o danneggiare i sistemi operativi. La diffusione del malware può propagare la minaccia a nuovi utenti e sistemi.

5. **Esposizione delle password in chiaro:** In alcuni casi, le password possono essere memorizzate in modo inadeguato o essere oggetto di perdite di dati. Se le

password vengono esposte in modo non cifrato, diventano accessibili a chiunque abbia accesso a tali informazioni, facilitando l'accesso non autorizzato ai servizi correlati.

Per mitigare tali rischi, è fondamentale adottare pratiche di sicurezza informatica avanzate. La consapevolezza degli utenti e una vigilanza costante sono altrettanto essenziali per mantenere un ambiente digitale sicuro e resistere alle minacce.

Capitolo 3

Concetti di Funzioni Hash

In questo capitolo, introdurremo inizialmente la materia della crittografia, esplorando le sue fondamenta. Successivamente, approfondiremo le funzioni hash, illustrandone i concetti di base e i suoi principi di sicurezza.

3.1 Crittografia e contesto

La crittografia (dal greco ó [kryptós], "nascosto", e [graphía], "scrittura") è la disciplina che si occupa dello studio dei metodi per rendere un messaggio non comprensibile/intelligibile a persone non autorizzate a leggerlo; per garantire così il requisito di riservatezza tipico della sicurezza informatica. Tale messaggio si chiama crittogramma e i metodi utilizzati tecniche di cifratura.

La crittografia si basa su due principali approcci, noti come crittografia a chiave simmetrica e quella a chiave pubblica, ciascuno dei quali presenta caratteristiche specifiche e si adatta a diverse situazioni.

La crittografia a chiave simmetrica coinvolge l'utilizzo di una singola chiave segreta condivisa tra mittente e destinatario. Questa chiave viene impiegata sia per cifrare che per decifrare i dati. L'efficacia di questo approccio si basa sulla sicurezza della chiave condivisa, che richiede meccanismi sicuri per la sua distribuzione. La crittografia a chiave simmetrica è conosciuta per la sua efficienza e velocità, rendendola ideale per cifrare grandi quantità di dati. Tuttavia, la principale sfida risiede nella gestione delle chiavi, poiché la compromissione della chiave segreta metterebbe a rischio la sicurezza dei dati cifrati.

Dall'altro lato, la crittografia a chiave pubblica è un approccio basato su una coppia di chiavi: una chiave pubblica e una chiave privata. La chiave pubblica è diffusa aperta-

mente, mentre la chiave privata è mantenuta segreta. Qualsiasi mittente può utilizzare la chiave pubblica del destinatario per cifrare i dati, ma solo il destinatario, possedendo la chiave privata corrispondente, è in grado di decifrarli. Questo approccio elimina la necessità di una condivisione sicura delle chiavi, ma è generalmente più lento rispetto alla crittografia a chiave simmetrica a causa della complessità matematica coinvolta.

Spesso, in pratica, si utilizza una combinazione dei due metodi per massimizzare la sicurezza e l'efficienza.
Ad esempio, durante una comunicazione, la crittografia a chiave simmetrica è spesso utilizzata per cifrare i dati effettivi, mentre la chiave simmetrica stessa è scambiata in modo sicuro utilizzando la crittografia a chiave pubblica. In questo modo, si sfruttano le qualità di entrambi i metodi per una soluzione ottimale. Si utilizza, quindi, l'efficienza della crittografia a chiave simmetrica e la sicurezza della crittografia a chiave pubblica.

Per migliorare la sicurezza di questi algoritmi, lo studio della crittografia a portato a sviluppare metodi sempre più sofisticati, come per esempio l'uso delle curve ellittiche negli algoritmi a chiave pubblica oppure il protocollo Diffie-Hellman, con il quale mittente e destinatario collaborano alla creazione di una chiave privata.

Abbiamo esaminato come proteggere una comunicazione tra mittente e destinatario. Tuttavia, nel caso in cui un aggressore tenti di alterare il messaggio, come possiamo verificare se è stato manipolato o meno? Questa è definita come integrità del messaggio, ovvero la capacità di verificare se i dati sono stati alterati o manomessi durante la trasmissione. Per raggiungere questo obiettivo, facciamo ricorso alle funzioni Hash crittografiche, queste svolgono un ruolo cruciale nell'assicurare l'integrità dei dati, poiché consentono di rilevare modifiche non autorizzate, sia nei messaggi cifrati che in quelli in chiaro.

3.2 Definizione e proprietà

Una funzione Hash è una funzione che prende in input una sequenza di bit di lunghezza arbitraria e restituisce in output una sequenza di bit di lunghezza fissa, chiamata "digest" o "impronta".

Definizione:
Una $Hash\ family$ è una quadrupla (X, Y, K, H), in cui sono soddisfatte le seguenti condizioni:

1. X è un insieme di possibili input.
2. Y è un insieme finito di possibili digest.
3. K è un insieme finito di possibili chiavi.
4. Per ogni $k \in K$, esiste una funzione di Hash $h_k \in H$. Ogni $h_K : X \to Y$.

Nella definizione sopra indicata, X potrebbe essere un insieme finito o infinito, mentre Y è sempre un insieme finito.

Se X è un insieme finito e $X > Y$, la funzione è chiamata anche "funzione di compressione".

3.3 Sicurezza delle funzioni Hash

Una funzione Hash, in crittografia, per essere considerata sicura deve essere difficile risolvere questi tre problemi.

Pre-immagine (one-way)

Data una funzione Hash $h : X \rightarrow Y$ e un elemento $y \in Y$.
trovare $x \in X$ *t.c.* $h(x) = y$.

Nel problema della pre-immagine deve essere computazionalmente difficile avendo un digest y risalire al suo valore iniziale x.

Seconda immagine

Data una funzione Hash $h : X \rightarrow Y$ e un elemento $x \in X$.
trovare $x_0 \in X$ *t.c.* $x_0 \neq x$, $h(x_0) = h(x)$.

Nel problema della seconda immagine deve essere computazionalmente difficile avendo un valore x trovare x_0 che abbia stessa immagine di x.

Collisione (claw-free)

Data una funzione Hash $h : X \rightarrow Y$
trovare x_0 *t.c.* $x_0 \neq x$, $h(x_0) = h(x)$.

Nel problema delle collisioni deve essere computazionalmente difficile trovare due valori che abbiano una stessa immagine in Y.

Problema computazionalmente difficile:
Un problema computazionalmente difficile è un problema per il quale non esiste un algoritmo efficiente (cioè con complessità polinomiale) in grado di risolverlo in un tempo ragionevole. In altre parole, risolvere questi problemi richiede un tempo di calcolo che cresce in modo significativo all'aumentare della dimensione del problema. Tali problemi possono richiedere tempo esponenziale ed essere intrattabili da risolvere in modo pratico.

3.3.1 Paradosso del Compleanno

Il paradosso del Compleanno, è un problema di teoria della Probabilità definito da Richard von Mises nel 1939.
Nelle funzioni Hash, questo concetto è utile per spiegare la resistenza alle collisioni. Si fa uso del paradosso del compleanno per stimare il numero di volte che dobbiamo applicare la funzione Hash su input diversi prima di trovare una collisione.

Definizione:
Il paradosso del compleanno afferma che la probabilità che almeno due persone compiano gli anni lo stesso giorno è ampiamente superiore a quanto potrebbe dire l'intuito. Infatti in un gruppo di 23 persone la probabilità che due persone compiano gli anni lo stesso giorno è circa del 51%, con 30 persone circa il 70%, e con 50 circa il 97%.

Descrizione:
Assumiamo che tutti gli anni siano composti da 365 giorni, non consideriamo gli anni bisestili.
Utilizziamo la probabilità condizionata per calcolare: dato p numero di persone, quanto è probabile che ci sia almeno una coppia di persone che abbiano lo stesso giorno di compleanno.

$$P(365, p) = 1 - \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-p+1}{365} = 1 - \left(\frac{365!}{365^p \cdot (365-p)!}\right)$$

In una forma più generale:

$$P(n, p) = 1 - \left(\frac{n!}{n^p \cdot (n-p)!}\right)$$

n è il numero dei valori che possono assumere le p persone.

Facendo un'analogia con le funzione Hash, possiamo considerare il numero di giorni in un anno 365 come la cardinalità dell'insieme dei digest prodotti da una funzione Hash. Inoltre considerare le persone p come il numero di input diversi inseriti.

Dopo uno studio di probabilità matematica, si è trovato che con funzioni Hash che producono N bit dobbiamo avere $b = 2^{\frac{N}{2}}$ input differenti, poiché la probabilità che tra di essi ci siano delle collisioni sia all'incirca del 50%. Questo risultato non ci deve assolutamente allarmare, poiché le funzioni Hash utilizzano insiemi di digest molto grandi.

Per esempio SHA-256 produce digest della lunghezza di 256 bit, n è uguale a 2^{256}, e per avere una buona probabilità di trovare collisione dobbiamo eseguire la funzione Hash 2^{128} volte (b= 2^{128}) , numero estremamente alto.

$P(2^{256}, 2^{128}) \cong 50\%$ di trovare collisioni con 2^{128} tentativi.

3.3.2 Random Oracle Model

Il Random Oracle, è una macchina a oracolo random cioè possiamo considerarla come una scatola nera che risponde agli input con output veramente casuali su un dominio predefinito.

La teoria del Random Oracle Model, in crittografia, ci dà la possibilità di creare un modello matematico per una funzione di Hash "ideale". Questo modello è stato introdotto da Mihir Bellare e Phillip Rogaway e il teorema fornisce una base teorica per l'analisi di sicurezza delle funzioni di Hash.

Definizione:
Nel modello dell'oracolo casuale, si parte dall'assunzione che esiste un oracolo casuale che funge da funzione di Hash ideale. Questo oracolo è "perfetto" e restituisce valori casuali per ogni input, indipendentemente da qualsiasi altro input. Da questa assunzione possiamo dire che molte delle proprietà desiderabili possono essere dimostrate per i protocolli crittografici basati su questa funzione di Hash ideale.

Le proprietà chiave che possono essere dimostrate nel modello dell'oracolo casuale includono:

1. **Indipendenza delle pre-immagini**: La probabilità di trovare un input dato un valore di Hash è uniformemente distribuita, poiché ogni possibile valore di Hash è indipendente dagli altri.
2. **Indipendenza delle seconde immagini**: La probabilità di trovare un secondo input con lo stesso valore di Hash di un input noto è uniformemente distribuita
3. **Indipendenza delle collisioni**: La probabilità che due input diversi abbiano lo stesso valore di Hash è estremamente bassa, poiché ogni possibile valore di Hash è come una scelta casuale indipendente.

Dimostrazione di 1 e 2:
Consideriamo una funzione Hash $h : X \to Y$ e due elementi $x_1, x_2 \in X, x12$. La probabilità che $h(x_1)$ sia uguale a un valore specifico, ad esempio v, è la stessa della probabilità che $h(x_2)$ sia uguale a v. Quindi:

$P[h(x) = v] = P[h(y) = v] = 1/|Q|$

Questo dimostra l'indipendenza delle pre-immagini e l'indipendenza delle seconde immagini nel modello dell'oracolo casuale.

Dimostrazione di 3:
Per dimostrare questa proprietà, consideriamo due input diversi, x e y, e calcoliamo la probabilità che h(x) sia uguale a h(y). Poiché h(x) è come un valore casuale indipendente nel modello dell'oracolo casuale, la probabilità di collisione è la probabilità che due valori casuali siano uguali.

$P[h(x) = h(y)] = 1/|Q|$

Poiché il numero di possibili valori di Hash è molto grande, la probabilità di collisione è estremamente bassa nel modello dell'oracolo casuale.

Queste proprietà sono importanti perché riflettono le caratteristiche desiderabili di una funzione di Hash "ideale". Tuttavia, è importante notare che il modello dell'oracolo casuale è una costruzione teorica e non riflette la realtà delle funzioni di Hash nel mondo reale. La sua utilità sta nel fornire uno standard per la valutazione della sicurezza e dell'efficacia dei protocolli crittografici che utilizzano funzioni di Hash.

Capitolo 4

Archiviazione e gestione delle password

Questo capitolo si focalizzerà sulle tecniche di archiviazione e gestione delle password con l'obiettivo principale di garantire la massima sicurezza da parte di malintenzionati. La sicurezza delle password è di vitale importanza in un mondo digitale in cui le minacce informatiche sono sempre più sofisticate e diffuse.

In linea di principio, è possibile memorizzare una password insieme all'utente corrispondente direttamente nel database. Tuttavia, è importante notare che questa pratica presenta gravi rischi per la sicurezza dei dati e la privacy degli utenti. Poiché il database può essere accessibile a vari utenti, se le password sono memorizzate in chiaro, chiunque abbia accesso al database può leggere e potenzialmente abusare delle password degli utenti, compromettendo così la loro privacy.
Per evitare i rischi legati alla memorizzazione delle password nei database, è fondamentale introdurre il concetto di Salt e di Hashing delle password.

4.1 Hashing delle password

Uno degli usi delle funzioni Hash, in crittografia, è sicuramente l'Hashing delle password. Questa tecnica consiste nel trasformare le password in una forma crittografica che le rende indecifrabili per chiunque abbia accesso al database. L'Hashing delle password, in pratica, implica che le password degli utenti non vengano memorizzate come testo in chiaro nel database, ma vengano convertite in una sequenza di caratteri alfanumerici unica, detta Hash.

Esempio pratico: Un utente vuole registrarsi e inserisce come password: "password123", nel database è salvato l'Hash di "password123" come segue nel esempio.

$password123 \rightarrow h(password123)$
$\rightarrow EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F$

Ogni volta che l'utente vuole effettuare l'accesso, tenta d'inserire la password, questa viene Hashata e confrontata con l'Hssh salvato del database, se i due Hash corrispondono è eseguito il login.

$password1234 \rightarrow h(password1234)$
$\rightarrow B9C950640E1B3740E98ACB93E669C65766F6670DD1609BA91FF41052BA48C6F3$
$\neq EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F$

Con la tecnica appena descritta abbiamo aggiunto un livello di sicurezza, rendendo le password indecifrabili essendo delle stringhe casuali, ma questo approccio presenta ancora alcune criticità.
Per esempio se due utenti inseriscono una stessa password, nel database entrambi avranno la stessa stringa Hash. Utilizzando tecniche di crittoanalisi si può trovare la password comune. Nel seguente paragrafo sono descritte queste tecniche e come la crittografia ha trovato modi risolvere queste criticità.

4.2 Salt

Quando in una database si trovano due Hash uguali, significa che due utenti hanno utilizzato la stessa password. Questa password utilizzata è una password debole ed è semplice da trovare utilizzando tecniche di rainbow table.
La tecnica del rainbow table, è una tecnica in cui gli aggressori utilizzano tabelle in cui sono presenti coppie Hash-password. Queste tabelle sono pregenerate e possono includere molte righe con le password più comuni o possibili.
Gli aggressori utilizzano le rainbow tables per cercare di corrispondere l'Hash delle password crittografate, che stanno cercando di decifrare, con le password presenti nella tabella. Se l'Hash è presente nella tabella, possono risalire alla password corrispondente in modo relativamente rapido (vedi capitolo 9.3).

Per far fronte a questa vulnerabilità lo studio della crittografia ha introdotto l'uso del Salt nell'Hashing delle password.
Il Salt, semplicemente, è una stringa casuale di caratteri. Questa è generata e aggiunta a una password prima di applicare una funzione di Hash. Questo processo è noto come "Salting."
Quando si utilizza un Salt, anche se due utenti hanno la stessa password, i loro Hash salvati nel database saranno diversi a causa del Salt unico aggiunto a ciascuna password.

Ciò rende molto più difficile per un attaccante pre-calcolare gli Hash delle password, poiché la funzione Hash non è più applicata alla sola password ma alla concatenazione $Salt|password$.

Esempio pratico: come prima l'utente vuole registrarsi con la password: password123, nel momento in cui inserisce la password è generato un Salt, questo è concatenato alla password, ed è applicata la funzione Hash alla concatenazione delle due stringhe.

Salt: 8B4E2A1F6D3C9A5E7B0D8F6C4E9A1D6F
password: password123

$h(8B4E2A1F6D3C9A5E7B0D8F6C4E9A1D6F|password123) =$
$= 383E9F038A38238CCE3B0E5B68FFDECCB8D14F5B6A39E5891D866F220DD6B51B$

Nel database è memorizzato oltre l'Hash anche il Salt, necessario per risalire alla password.
Ogni volta che l'utente vuole effettuare l'accesso, tenta d'inserire la password, viene preso il Salt dal database e concatenato alla password inserita. Dopodichè è Hashata la concatenazione, questa è confrontata con l'Hash salvato del database, se i due Hash corrispondono è eseguito il login.

password inserita: password1234
$h(8B4E2A1F6D3C9A5E7B0D8F6C4E9A1D6F|password1234) =$
$= 070F97DF8C7D22096C0D1FC8BF6EB4FF170B2F1B82CAFF4537A48EA1DF13B0D8$
$\neq 383E9F038A38238CCE3B0E5B68FFDECCB8D14F5B6A39E5891D866F220DD6B51B$

Nella seguente tabella è rappresentato ciò che è memorizzato nella tabella User che utilizza l'Hashing e il Salt.

User	**Hash**	**Salt**
marcorossi	383E9F038A38238CCE3B0E5B68FF...	8B4E2A1F6D3C9A...
...	...	...

Tabella 4.1: Memorizzazione dell'Hash e del Salt

4.3 Registrazione di un utente

In questa sezione sono indicate le istruzioni di come eseguire una registrazione di un utente utilizzando l'Hashing della password combinato al Salt. Questo, come già detto, ci permette di garantire privacy e sicurezza per le password degli utenti.

Passaggi per eseguire la registrazione:

1. Generazione di un Salt, ovvero una stringa alfanumerica pseudocasuale
2. Concatenazione della password con il Salt $Salt|password$
3. Applicazione delle funzioni Hash $h(Salt|password)$
4. Memorizzazione nel database dell'Hash ottenuto e del Salt

Di seguito lo pseudocodice per eseguire la registrazione in modo sicuro di un utente.

```
Function Registrazione(utente, password)
    Salt ← generaSalt()
    Hash ← h(Salt|password)
    queryInsert(utente, Salt, Hash)
```

Per `queryInsert` si intende l'inserimento di un utente all'interno del database.

4.4 Login di utente

In questa sezione invece indichiamo le linee guida di come effettuare il login.

Passaggi per eseguire il login:

1. Dal database si prende Salt e Hash del corrispondente utente
2. Concatenazione della password inserita con il Salt $Salt|passwordInserita$
3. Applicazione delle funzioni Hash $h(Salt|passwordInserita)$
4. Se l'Hash memorizzato nel database è identico al nuovo Hash avviene il login altrimenti non è concesso il login

Di seguito lo pseudocodice per eseguire il login di un utente.

```
Function Login(utente, password):Boolean
    Salt ← queryGetSalt(utente)
    HashUtente ← queryGetHash(utente)
    nuovoHash ← h(Salt|password)
    if HashUtente == nuovoHash then
        return true
    return false
```

Capitolo 5

Funzioni Hash

Nel seguente capitolo, forniremo una descrizione delle funzioni Hash più conosciute e utilizzate. Queste non sono specifiche per i sistemi di sicurezza delle password, ma possono essere impiegate in vari contesti, come l'integrità dei dati o la firma digitale.

Le funzioni Hash di solito fanno parte di famiglie di funzioni Hash, il che significa che all'interno di una famiglia ci sono funzioni Hash simili, ognuna delle quali rappresenta un'evoluzione rispetto all'altra.
Le famiglie più note sono SHA, MD, RIPEMD, di seguito una panoramica su queste tre famiglie.

5.1 Famiglia SHA

SHA, acronimo dell'inglese "Secure Hash Algorithm", indica una famiglia di funzioni Hash, sviluppate a partire dagli anni 90'. Le funzioni della famiglia SHA, acronimo di "Secure Hash Algorithm", sono state sviluppate dagli istituti di sicurezza degli Stati Uniti e ufficialmente pubblicate dal NIST, il National Institute of Standards and Technology, un'agenzia governativa degli Stati Uniti dedicata alla standardizzazione e alla definizione di norme tecniche. Questi algoritmi crittografici, introdotti a partire dagli anni '90, sono diventati ampiamente adottati a livello globale e fungono da standard di riferimento per garantire la sicurezza e l'integrità dei dati in diverse applicazioni crittografiche e di sicurezza informatica. Di seguito una descrizione delle funzioni Hash SHA.

SHA-0

SHA-0 noto anche come SHA-Original, fu il primo algoritmo sviluppato della famiglia SHA. Tuttavia, non fu mai pubblicato ufficialmente a causa della sua poca sicurezza.

SHA-1

SHA-1 è stato introdotto come miglioramento dell'SHA-0 e pubblicato nel 1993. Produce una stringa di 160 bit ed è stato utilizzato ampiamente per scopi crittografici. Il suo principale problema è la vulnerabilità alle collisioni. Infatti nel 2005, è stato dimostrato che era possibile trovare 2 input che producevano lo stesso Hash, infrangendo così la sicurezza della funzione SHA-1.

SHA-2

A causa delle vulnerabilità dell'SHA-1, il NIST ha pubblicato la famiglia SHA-2. La famiglia di algoritmi SHA-2 è stata introdotta nel 2002 ed è riconosciuta per la sua sicurezza e la sua resistenza alle collisioni. Questa famiglia include le funzioni SHA-224, SHA-256, SHA-384 e SHA-512, i cui nomi indicano la lunghezza in bit specifica del loro digest.

SHA-3

SHA-3 è l'ultimo membro della famiglia SHA ed è stato selezionato dal NIST a seguito di una competizione, vinta da un team di analisti italiani e belgi. È stato pubblicato nel 2015 e noto per essere altamente sicuro. A differenza di SHA-1 e SHA-2 è basato su principi crittografici diversi, utilizzo di una Sponge Construction, che vedremo nei paragrafi successivi.

SHAKE-n

Le funzioni SHAKE-n sono una variante del costrutto SHA-3. Queste funzioni Hash sono progettate per fornire una lunghezza variabile dell'output. Le funzioni SHAKE più note sono SHAKE128 e SHAKE256.

Riassumendo, l'uso di SHA-1 è fortemente sconsigliato, perché ormai obsoleto e poco sicuro, visto che sono dimostrate tecniche per contrastare la sua sicurezza. Mentre attualmente SHA-2 e SHA-3 sono considerate sicure e valide per le attuali applicazioni crittografiche.

5.2 Famiglia RIPEMD

RIPEMD, acronimo in inglese di "Race Integrity Primitives Evaluation Message Digest", nacque come un alternativa europea alle famiglie MD e SHA di provenienza americana. Questa comprende le funzioni RIPEMD-160, RIPEMD-256, RIPEMD-320. La più conosciuta e utilizzata è RIPEMD-160. Il primo a essere sviluppato fu RIPEMD-128,

che non fu capace di resistere a un attacco di forza bruta di un super computer degli anni '90. Quindi fu sviluppato RIPEMD-160. Anni dopo furono sviluppati RIPEMD-256 evoluzione di RIPEMD-128 e RIPEMD-320 evoluzione di RIPEMD-160, questi non aggiunsero maggiori livelli di sicurezza, ma presentarono solo una possibilità di avere collisioni minore, grazie al fatto che producevano un Hash il doppio più lungo. Con il passare del tempo, questa famiglia di algoritmi non fu più sviluppata e aggiornata e divenne obsoleta.

5.3 Famiglia MD

La famiglia MD, dall' inglese Message Digest, è stata una delle prime a essere ideate, sviluppata da Ronald Rivest, noto crittografo coautore del sistema RSA. Il primo algoritmo di MD non venne mai pubblicato; il primo a essere pubblicato fu MD2, poi seguito da MD4 e MD5. Sia MD2 che MD4 avevano grossi problemi di vulnerabilità.

MD5

La funzione Hash MD5 è la più nota della famiglia Message Digest, realizzata nel 1991, e ampiamente utilizzata negli anni 90' e 2000. La sua popolarità era dovuta alla sua semplicità e alla sua efficienza nell'eseguire calcoli di Hash.
Questa produce stringhe di 128 bit e ha il vantaggio di eseguire l'Hash in maniera veloce richiedendo inoltre poche risorse.
Come le funzioni Hash di quei anni furono trovate collisioni e vulnerabilità che resero l'algiritmo non più utilizzabile.

Capitolo 6

Analisi degli algoritmi delle funzioni Hash

Nel seguente capitolo forniamo una descrizione di come sono stati sviluppati gli algoritmi delle principali funzioni Hash.

6.1 Costruzione iterativa

Gli algoritmi delle più comuni e utilizzate funzioni Hash sono implementati attraverso un costrutto iterativo.
Una costruzione iterativa è un processo che consente di creare una funzione in cui l'input può essere di qualsiasi lunghezza, e attraverso una serie di trasformazioni produce un output di lunghezza n.

Questo metodo utilizza funzioni di compressione, ovvero funzioni che prendono in input una stringa di $m + t$ bit, con $t >= 1$, e restituiscono una stringa di m bit. Queste funzioni sono applicate in modo iterativo, cioè una dopo l'altra.
Di seguito è spiegato il metodo, il quale è composto da 3 fasi:
pre-processing, processing, finalizzazione.

Supponiamo funzione di compressione *compress*: $\{0, 1\}^{m+t} \rightarrow \{0, 1\}^m$ con $t >= 1$

Fase di pre-processing

Dato x come input iniziale, e $|x| > m + t + 1$
Dalla stringa x costruiamo una stringa y t.c. $|y| = 0(\text{mod } t)$, cioè la sua dimensione è un multiplo di t
Possiamo scrivere $y = y_1||y_2||...||y_r$ dove $|y_i| = t$ per $i = 1...r$

Fase di processing

Consideriamo IV valore arbitrario di lunghezza m
Calcoliamo:

$z_0 \leftarrow \mathrm{IV}$
$z_1 \leftarrow \mathrm{compress}(z_0 \| y_1)$
$z_2 \leftarrow \mathrm{compress}(z_1 \| y_2)$
...
...
$z_r \leftarrow \mathrm{compress}(z_{r-1} \| y_r)$

In questo modo l'output della compress è utilizzato come input della compress successiva, creando cosi un processo iterativo.

Finalizzazione

Definiamo quindi $h(x) = z_r$

Di solito nella fase di pre-processing si aggiunge un padding: $y = x||pad(x)$
Questo per ottenere $|y| = 0(\mathrm{mod}\ t)$

Nella seguente figura è descritta come avviene la fase di processing.

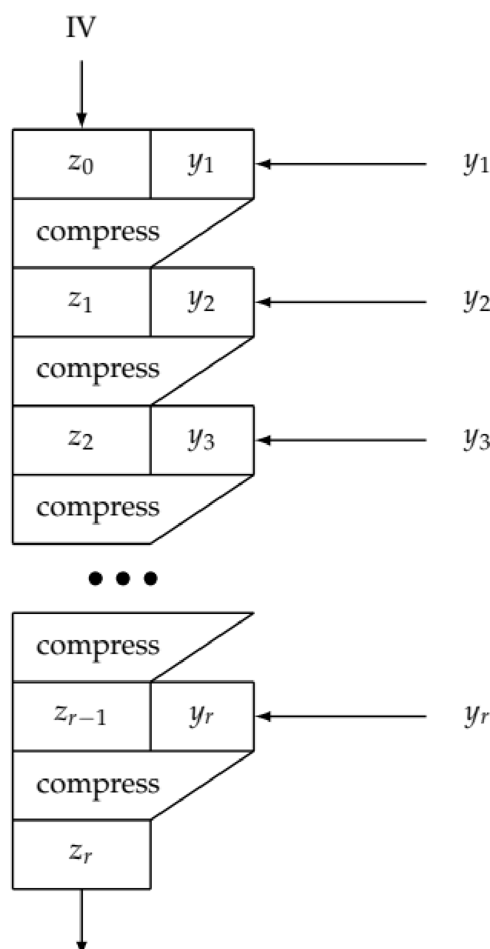

Figura 6.1: Fase di processing

6.2 Costruzione di Merkle-Damgård

Il metodo appena visto descrive una costruzione che attraverso l'utilizzo di funzioni di compressione che lavorano con stringhe di lunghezza fissa si ottiene una funzione Hash che riceve in input stringhe di lunghezza arbitraria e restituisce un output di stringhe di lunghezza n.
La costruzione Merkle-Damgård è un particolare metodo iterativo che soddisfa le proprietà della resistenza alle collisioni.
Questo metodo utilizza funzioni di compressione a senso unico (one-way), cioè funzioni

computazionalmente difficili da invertire.
Grazie a queste i due matematici Ralph Merkle e Ivan Damgård dimostrarono che se nella nostra funzione Hash utilizziamo funzioni di compressione a senso unico otteniamo una funzione Hash con caratteristica di resistenza alle collisioni, quindi ideali per scopi crittografici.
Funzioni Hash come sha-1 sha-2 sono implementate seguendo questo costrutto iterativo.

6.3 Funzioni one-way

Nel paragrafo precedente abbiamo visto come costruire una funzione di Merkle-Damgard attraverso un processo iterativo utilizzando funzioni di compressione one-way. Ma come sono realizzare queste funzioni one-way?

Possiamo definifire una funzione one-way come una funzione in cui è facile calcolare la sua immagine, ma risulta un problema computazionalmente difficile risalire al valore originale.

$f(x) = y$ è computazionalmente facile
$f^{-1}(y) = x$ è computazionalmente difficile

Il principio per cui una funzione di compressione può essere one-way è descritto nella seguente spiegazione:

La complessità matematica sottostante alla difficoltà di invertire una funzione di Hash crittografica è legata all'idea che, sebbene l'input possa essere di lunghezza variabile e potenzialmente molto grande, l'output ha una lunghezza fissa e costante. In altre parole, stiamo cercando di comprimere molte possibili voci in una singola "casella" (l'Hash), il che significa che ci sono molte più possibili voci dell'input rispetto agli Hash risultanti. Questo crea una situazione in cui più input potrebbero essere mappati a uno stesso Hash, ma è computazionalmente difficile determinare quale input specifico sia stato utilizzato per generare un Hash specifico, a meno che non si conosca l'input in anticipo.

6.4 Costruzione Sponge

Le funzioni Hash più recenti sono implementate attraverso un costrutto a "spugna", per esempio SHA-3 utilizza questa costruzione.

Questo tipo di costrutto invece di utilizzare una funzione di compressione, utilizza una funzione f che mappa le sequenze di bit di lunghezza fissa in sequenze di bit della stessa lunghezza.
Inoltre, opera in due fasi principali, fase di assorbimento e una fase di spremitura.
Breve descrizione del processo di Sponge:

Supponiamo $f : \{0,1\}^b \rightarrow \{0,1\}^b$

Denotiamo: b come dimensione, bit in cui lavora f
dove b=r+c in cui r è il bitrate, mentre c è la capacità.

Padding iniziale

M messaggio in input, è riempito da un padding affinché M è un multiplo di r:
$M = M \| pad$ t.c. $|M| = 0$ mod(r)
M è suddiviso in $m_0, m_1, ..., m_n$ t.c $|m_i| = r$ per ogni $i = 0...n$

Fase di assorbimento

Supponiamo state s, stringa di lunghezza b: $|s| = b$, inizializzata a tutti 0

$s_0 \leftarrow$ (m_0 XOR i primi r bit dello state s) $\|$ rimanenti bit dello state s
$s_0 \leftarrow f(s_0)$ poi applicata f

$s_1 \leftarrow$ (m_1 XOR i primi r bit dello state s_0) $\|$ rimanenti bit dello state s_0
$s_1 \leftarrow f(s_1)$

...
...

$s_n \leftarrow$ (m_n XOR i primi r bit dello state s_{n-1}) $\|$ rimanenti bit dello state s_{n-1}
$s_n \leftarrow f(s_n)$

Fase di spremitura

l definita come lunghezza in bit dell'output desiderato
Z è l'output

$Z \leftarrow$ primi r bit di s_n
se $l > r$ applichiamo f: $s_n \leftarrow f(s_n)$

$Z \leftarrow Z \parallel$ primi r bit di s_n
se $l > |Z|$ applichiamo f: $s_n \leftarrow f(s_n)$

...
...

Finché $l <= |Z|$

Nella seguente figura è rappresentato in modo schematico le due fasi della costruzione Sponge.

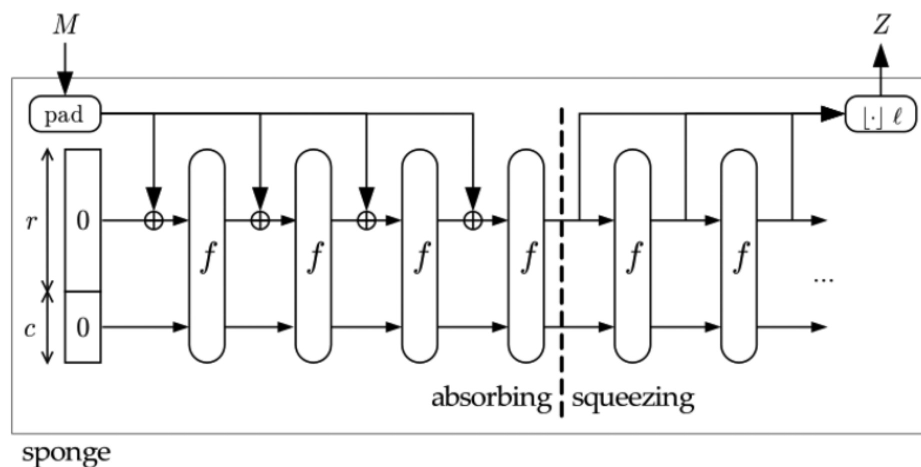

Figura 6.2: Fase di assorbimento e spremitura

Osservazione
La scelta del rate r e della capacità c influisce nell'efficienza e sicurezza della funzione:

1. un rate maggiore incrementa l'efficienza e la velocità di elaborazione dell'algoritmo, poiché si elaborano più informazioni alla volta di conseguenza ci sono meno iterazioni.

2. una capacità maggiore incrementa la sicurezza e riduce e la resistenza alle collisioni. Questo perché si ottiene una maggiore ampiezza per l'assorbimento e la diffusione dei dati.

È importante sottolineare che questi due parametri sono uno l'antagonista dell'altro, poiché aumentando uno diminuisce l'altro, e viceversa (infatti b=r+c).

6.5 Implementazione di SHA-3

SHA-3 utilizza come detto precedentemente il modello di costruzione Sponge.
SHA-3 è costituito da quattro funzioni di Hash, le quali sono SHA3-224, SHA3-256, SHA3-384 e SHA3-512. I suffissi indicano le lunghezze dei digest prodotti.

La funzione f è una funzione biunivoca che opera su uno stato che è una sequenza di bit di lunghezza 1600. Essa è composta da 24 round, ognuno dei quali è composto da cinque semplici passaggi. Sono quindi eseguite operazioni bit a bit, trasformando senza comprimere la stringa (a differenza della compress del costrutto iterativo).
La width, il bitrate e la capacità di queste funzioni sono riassunti nella tabella 6.1.

Nel SHA-3 sono incluse anche due funzioni aggiuntive, chiamate SHAKE128 e SHAKE256, che sono funzioni di output estensibili, ovvero funzioni nel quale il digest è di lunghezza variabile. Esso utilizza la stessa costruzione a spugna, ma può impiegare applicazioni aggiuntive di f nella fase di "spremitura" per generare digest di messaggio più lunghi. Tuttavia, è importante notare che quando si generano digest di messaggio più lunghi, la sicurezza dipende dal valore della capacità c.

Nella tabella 6.1 oltre al bitrate e alla capacità di ciascuna funzione, sono elencati anche i livelli di sicurezza di queste funzioni contro gli attacchi più noti, un valore maggiore indica una maggiore resistenza.

Hash Function	b	r	c	Resist.x alle collisioni	Resist. attacchi di pre-immagine
SHA3-224	1600	1152	448	112	224
SHA3-256	1600	1088	512	128	256
SHA3-384	1600	832	768	192	384
SHA3-512	1600	576	1024	256	512
SHAKE128	1600	1344	256	$\min\left(\frac{d}{2}, 128\right)$	$\min(d, 128)$
SHAKE256	1600	1088	512	$\min\left(\frac{d}{2}, 256\right)$	$\min(d, 256)$

Tabella 6.1: Sicurezza di SHA-3 e SHAKE

d sono i bit del digest prodotto

Osservazione 1:
Si può osservare dalla tabella, che come detto in precedenza, aumentando c migliora anche la sicurezza: aumenta sia la resistenza alle collisioni, sia la resistenza agli attacchi di prima immagine.

Osservazione 2:
Si può notare che tutte e quattro le funzioni di Hash in SHA-3 producono digest di

messaggio di lunghezza inferiore a r bit, si deduce quindi che nella fase di ”spremitura” non si applica la funzione f, mentre è utilizzata nelle funzioni Shake poiché l'output è un parametro arbitrario.

Capitolo 7

Funzioni Hash avanzate

In questo capitolo studieremo funzioni Hash sviluppate per essere utilizzate nell'Hashing delle password. Queste sono funzioni più avanzate e progettate appositivamente per difendere le password Hashate da possibili attacchi di varia natura.

7.1 Esempio di Crypt

Crypt fu una delle prime funzioni usate per criptare le password in ambiente Linux. Tuttavia, nel corso degli anni, con l'avanzare della tecnologia e l'aumento della potenza di calcolo, le debolezze di Crypt sono emerse, in particolare la sua velocità di Hashing relativamente bassa.

Infatti possiamo considerare Crypt come un esempio di mancato adattamento ai cambiamenti tecnologici. Secondo USENIX, nel 1976, Crypt poteva eseguire l'Hashing di meno di 4 password al secondo. Poiché gli aggressori devono trovare la pre-immagine di un Hash per poterlo invertire, questo ha fatto sì che il team UNIX si sentisse molto tranquillo sulla forza di Crypt. Tuttavia, 20 anni dopo, un computer veloce con software e hardware ottimizzati era in grado di eseguire l'Hashing di 200.000 password al secondo utilizzando quella funzione. Di conseguenza, un aggressore poteva eseguire un attacco completo a dizionario con estrema efficienza. Pertanto, per ostacolare i vantaggi in termini di velocità che gli aggressori potevano ottenere dall'hardware, era necessaria una crittografia che fosse esponenzialmente più difficile da violare man mano che l'hardware diventava più veloce.

7.2 Bcrypt

Bcrtypt è una funzione Hash sviluppata negli anni 90' da Niels Provos e David Mazières e pubblicata nel 1999.
Questa è ideata appositamente per proteggere le password poiché per generare l'Hash ha bisogno in input non solo della password, ma anche di un Salt; queste saranno combinate per creare un Hash.

Bcrypt porta anche un'altra novità rispetto alle precedenti funzioni Hash, questa richiede in input un parametro di costo c, in base a questo eseguirà 2^c iterazione nel calcolare l'Hash. Utilizzare un parametro c permette d'incrementare il costo computazionale, quindi rende il calcolo dell'Hash più lento, rendendo così più arduo per gli aggressori sfruttare hardware avanzato e il continuo aumento della potenza di calcolo.

Quindi Bcrypt è ottimo per resistere ad attacchi di rainbow table, inoltre è semplice incrementare la sua complessità aumentando il parametro costo.

7.3 Analisi Bcrypt

Bcrypt ha in input:

1. versione dell'algoritmo
2. fattore di costo
3. Salt
4. password

Mentre restituisce in output una stringa composta da:

1. versione dell'algoritmo
2. fattore di costo
3. Salt
4. Hash

I primi tre parametri sono intervallati da $ come possiamo notare dalla figura 5.1.

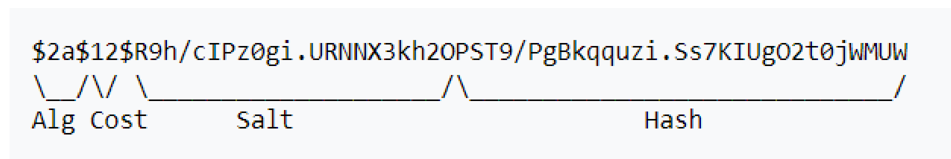

Figura 7.1: Hash di Bcrypt

Bcrypt è basato sul crittogramma Blowfish (da qui prende il nome la prima lettera di Bcrypt)
Infatti per produrre l'Hash è applicata una serie d'iterazioni della funzione Blowfish alla combinazione del Salt e della password. Il numero d'iterazioni, come già detto, è un parametro configurabile che influisce sulla complessità computazionale..

Blowfish

Blowfish è un algoritmo di crittografia a chiave simmetrica a blocchi ideato nel 1993 da Bruce Schneier. Questo utilizza una struttura a rete di Feistel operando su blocchi di dati da 64 bit, supporta chiavi/input di lunghezza variabile da 32 a 448 bit.
Di seguito è riportato il funzionamento del metodo crittografico.

L'algoritmo utilizza una rete di Feistel a 16 cicli, dividendo il blocco di dati in due metà. Ogni ciclo utilizza la funzione F, che coinvolge S-Box e operazioni XOR. Di seguito è mostrato come avviene un iterazione composta da 16 cicli.

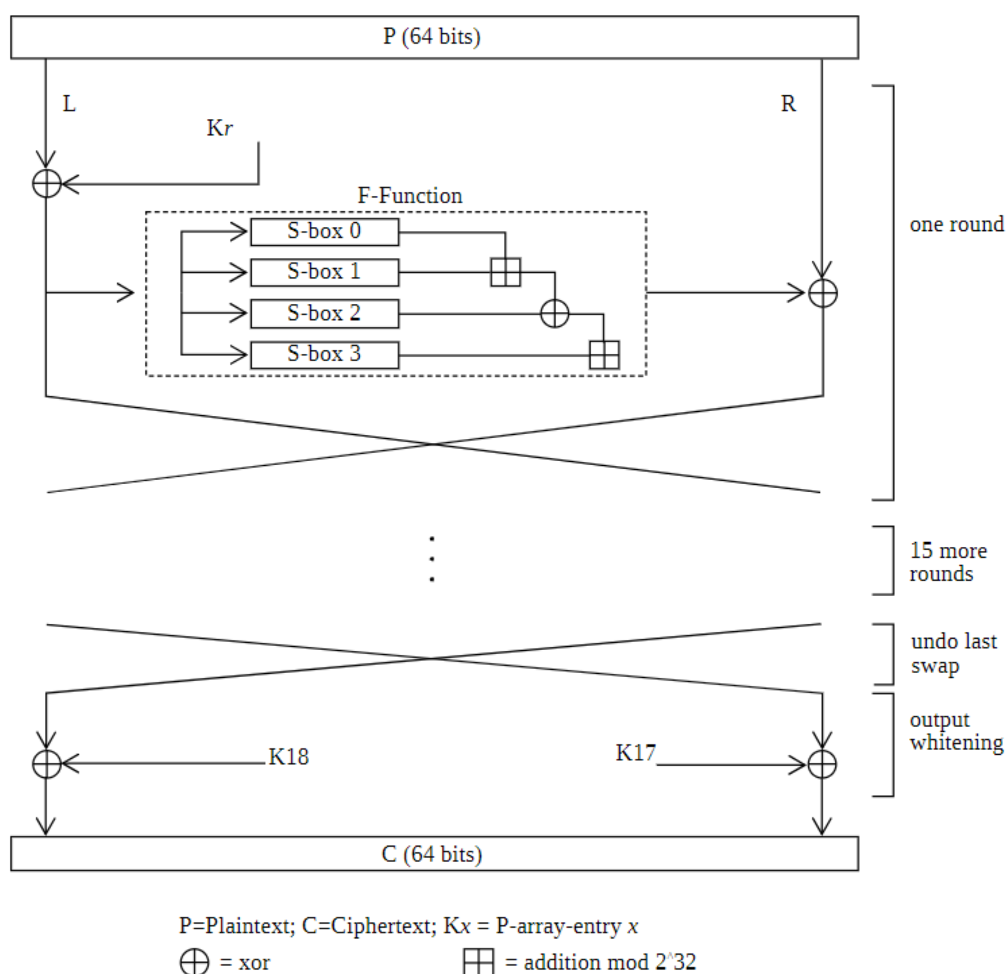

Figura 7.2: Diagramma di Blowfish

Di seguito una descrizione di come avviene un iterazione con il modello Blowfish.

Setup

Nella fase iniziale di setup vengono configurati diversi elementi chiave:

- Il blocco P, che rappresenta una porzione della suddivisione della password in input da crittografare.
- Le S-box: S_0, S_1, S_2, e S_3, ciascuna contenente 256 elementi da 32 bit.
- Il P-Array, composto da 18 elementi da 32 bit: P_1, P_2, P_3, ..., P_{18} (indicati in Figura 5.2 come K_x).

S-box e P-Array vengono inizializzati utilizzando parti della rappresentazione esadecimale di $\pi(pigreco)$.

Round

Il primo passo dell'algoritmo Blowfish coinvolge la suddivisione del blocco P in due parti, L e R, entrambe di 32 bit. L viene successivamente combinato tramite XOR con il primo elemento del P-Array, fungendo da input per la funzione F.

Nella funzione F, l'input L viene suddiviso in quattro parti da 8 bit ciascuna. Queste quattro parti agiscono come indici per le quattro S-box corrispondenti (ogni parte da 8 bit rappresenta un indice da 0 a 255). Per esempio, se la prima parte è un valore di 187, si accederà a $S_0[187]$. Gli elementi estratti dalle quattro S-box vengono quindi combinati tramite operazioni XOR e somma modulo 2^{32}, come mostrato nella figura 5.2.

Infine, l'output di F viene combinato tramite XOR con R, e le posizioni di L e R vengono scambiate.

Questo processo costituisce un round, e se ne eseguiranno altri 15. Al termine di questi, L ed R vengono XORati una volta finale con P_{18} e P_{17}, prima di essere ricongiunti per formare l'output crittografato, composto da 64 bit.

7.4 Argon2

Argon2 è una funzione di Hash crittografica progettata per l'Hashing di password e la derivazione di chiavi. È stato selezionato come vincitore del Password Hashing Competition (PHC) nel 2015 ed è considerato uno dei metodi più sicuri per proteggere le password.

Argon2 a livello di funzionalità è un'evoluzione di Bcrypt. A differenza di Bcrypt che prende come parametro in input solo il costo, Argon2 accetta come parametri: costo, memoria e parallelismo permettendo così ai programmatori di personalizzare l'uso della funzione in base alle esigenze e alle disponibilità hardware.

La progettazione di Argon2 si basa su principi avanzati di sicurezza, focalizzandosi su due aspetti cruciali: resistenza all'attacco e memoria intensiva. Questi aspetti sono stati attentamente studiati per rendere più complessa la decifrazione delle password crittografate e migliorare la sicurezza del sistema.

Resistenza all'attacco:

Argon2 è stato progettato per resistere efficacemente agli attacchi di tipo brute force e Rainbow Table (vedi paragrafi 9.2, 9.3)
Argon2 complica questo processo attraverso l'uso di una funzione di Hash che richiede un notevole sforzo computazionale per ogni tentativo di prova. Inoltre, la complessità intrinseca dell'algoritmo rende difficile l'uso di tabelle rainbow, un metodo di attacco che precalcola Hash e le associa alle password corrispondenti.

Operazioni di memoria intensiva:

Un aspetto distintivo di Argon2 è la sua intensità di utilizzo della memoria. L'algoritmo coinvolge una grande quantità di memoria durante il processo di Hash, rendendo più difficile per un attaccante parallelizzare l'esecuzione su molte risorse contemporaneamente. Questa caratteristica è cruciale per difendersi contro attacchi con hardware specializzato, come le GPU, che possono altrimenti velocizzare notevolmente gli attacchi di tipo "brute force". L'obbligo di utilizzare una quantità significativa di memoria rallenta il processo di Hashing, rendendo gli attacchi più costosi e, di conseguenza, meno praticabili.

In sintesi, l'approccio di Argon2 è quello di rendere estremamente costoso e complicato per gli attaccanti decifrare le password, fornendo un efficace strato aggiuntivo di sicurezza nella gestione delle credenziali utente.
Questa progettazione avanzata è stata riconosciuta e premiata nel contesto del Password Hashing Competition, contribuendo alla sua adozione come standard in molte implementazioni di sicurezza informatica.

Esistono tre versioni di Argon2:

1. **Argon2d**
   Argon2d utilizza un accesso alla memoria dipendente dai dati, dove le operazioni di lettura e scrittura nella memoria sono direttamente influenzate dai dati di input come le password o il Salt. Questo tipo di accesso alla memoria rende l'algoritmo resistente ad attacchi che utilizzano la parallelizzazione della GPU, ma lo rende più suscettibile ad attacchi side-channel, nei quali un attaccante cerca di estrarre informazioni sulla password analizzando comportamenti correlati, come il tempo di esecuzione o il consumo di energia (vedi capitolo 9.5).

2. **Argon2i**
   Argon2i, al contrario, utilizza un accesso alla memoria indipendente dai dati. In questo caso, le operazioni di lettura e scrittura nella memoria non sono influenzate direttamente dai dati di input. Ciò rende più difficile per gli aggressori ottenere informazioni da misurazioni di tempo o altri comportamenti.
   Argon2i è comunemente utilizzato per l'Hashing delle password.

3. **Argon2id**
   Argon2id, è una versione ibrida delle versioni Argon2d e Argon2i. Questa inizia eseguendo la fase di Argon2i per resistere agli attacchi di tipo side-channel, rendendo più difficile per un attaccante ottenere informazioni sulla password attraverso misurazioni di tempo o altri comportamenti. Successivamente, esegue la fase di Argon2d, che utilizza un accesso alla memoria dipendente dai dati, rendendo l'algoritmo più resistente agli attacchi basati sulla parallelizzazione della GPU.

7.5 Analisi Argon2

In questa sezione analizzeremo nei dettagli il funzionamento di Argon2.

Possiamo distinguere due tipologie di input: primari e secondari.

Gli input primari sono quelli obbligatori, abbiamo:

- Message P, o meglio la password
- Nonce S, nel nostro caso ha significato di Salt

Invece quelli secondari sono:

1. Grado di parallelismo p

2. Lunghezza del tag τ
3. Dimensione della memoria m
4. Numero di iterazioni t
5. Versione v
6. Chiave K, di solito non si utilizzano chiavi
7. Dati associati X
8. Tipo y, 0 per Argon2d, 1 per Argon2i, 2 per Argon2id

Argon2 utilizza una funzione di compressione G che ha in input 2048 bit e restituisce 1024 bit, e una funzione Hash Black2B interna H. G si basa su H.

La modalità di funzionamento di Argon2 è piuttosto semplice quando non viene utilizzato il parallelismo: la funzione G viene iterata m volte. Al passo i, viene preso un blocco con indice $\varphi(i) < i$ dalla memoria (Figura 5.2), $\varphi(i)$ è determinato dal blocco precedente in Argon2d, o è un valore fisso in Argon2i.

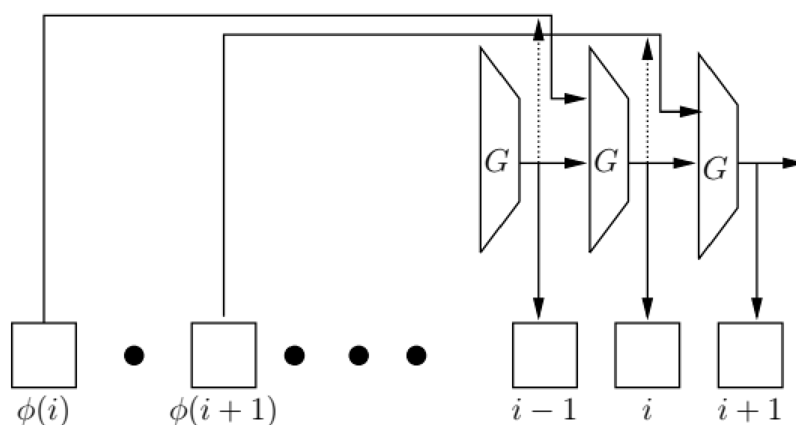

Figura 7.3: Argon2 senza parallelismo

Descrizione algoritmo:

Inizialmente eseguiamo una prima operazione di Hash, utilizzando in input tutti i parametri inseriti in questo modo:

$$H_0 = H(p, \tau, m, t, v, y, |P|, P, |S|, S, |K|, K, |X|, X)$$

I parametri $p, \tau, m, t, v, y, |P|, |S|, |K|, |X|$ sono trattati come interi a 32 bit.

La memoria è suddivisa in $m' = 4p[\frac{m}{4p}]$ blocchi da 1024 bit, dove m è il parametro di dimensione e p è il grado di parallelismo. Possiamo considerare la memoria suddivisa in una matrice $B[i][j]$, composta da p righe e q colonne dove $q = \frac{m'}{p}$.

I blocchi sono elaborati nel seguente modo:

$$B^1[i][0] = H'(H_0 || \underbrace{0}_{4 \text{ bytes}} || \underbrace{i}_{4 \text{ bytes}}), \quad 0 \leq i < p;$$
$$B^1[i][1] = H'(H_0 || \underbrace{1}_{4 \text{ bytes}} || \underbrace{i}_{4 \text{ bytes}}), \quad 0 \leq i < p;$$
$$B^1[i][j] = G(B^1[i][j-1], B^1[i'][j']), \quad 0 \leq i < p,\ 2 \leq j < q.$$

I primi due blocchi di ogni riga sono elaborati con H', con input H_0 concatenato con un padding di 8 byte. H' è una variante di H.
I blocchi seguenti invece sono elaborati utilizzando la funzione di compressione G, con input i blocchi B calcolati nelle colonne precedenti.
$B^1[i'][j']$ varia in base a Argon2d o a Ardon2i.

Se $t > 1$, in cui t è il numero di iterazioni, ripeteremo la procedura, però con la differenza che applicheremo al nuovo blocco uno XOR con il blocco dell'iterazione precedente (quello t-1), in questo modo:

$$B^t[i][0] = G(B^{t-1}[i][q-1], B[i'][j']) \oplus B^{t-1}[i][0];$$
$$B^t[i][j] = G(B^t[i][j-1], B[i'][j']) \oplus B^{t-1}[i][j].$$

Notiamo che per il primo blocco si applica G utilizzando l'ultimo blocco dell'iterazione precedente.

Una volta eseguite tutte le iterazioni per t, applichiamo uno XOR a tutti i blocchi dell'ultima colonna.

$$B_{\text{final}} = B^T[0][q-1] \oplus B^T[1][q-1] \oplus \cdots \oplus B^T[p-1][q-1].$$

Infine applichamo H' a B_{final}.

$$\text{Tag} \leftarrow H'(B_{\text{final}}).$$

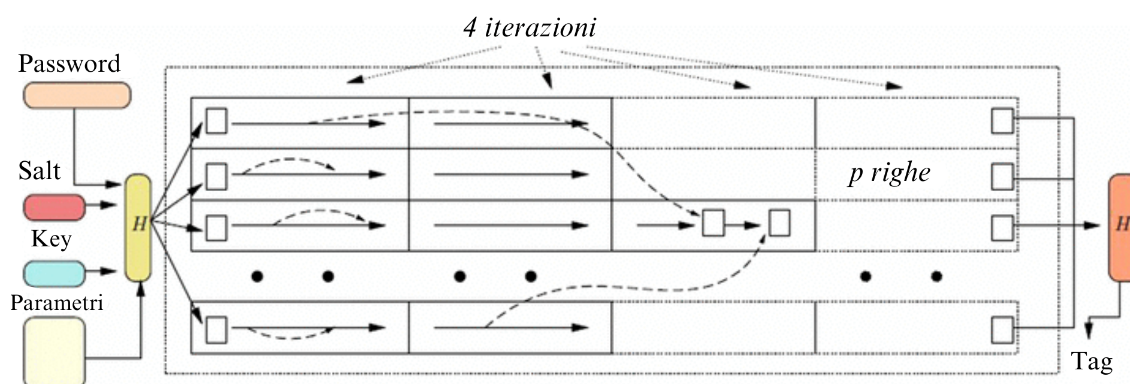

Figura 7.4: Funzionamento parallelismo Argon2

La figura 7.4 rappresenta uno schema che riassume come opera Argon2. Nella figura in esempio abbiamo 4 iterazioni, quindi per ogni blocco G è svolto 4 volte, si noti che G ha in input il blocco che lo precede e un blocco che varia. Come abbiamo detto in precedenza la scelta del blocco dipende dai dati se utilizziamo Argon2d, altrimenti è predefinita se utilizziamo Argon2i
Si possono osservare inoltre le p righe, queste sono esecuzioni parallele.

Capitolo 8

Best Practice per sicurezza delle password

Fino a ora abbiamo visto come da parte della crittografia c'è un ampio studio nella memorizzazione e nell'archiviazione delle password. In questo capitolo esploreremo le migliori pratiche per garantire robustezza delle password e ridurre il rischio di accessi non autorizzati.

8.1 Complessità delle password

Una password sicura deve essere sufficientemente lunga e complessa. Si raccomanda quindi di utilizzare una combinazione di lettere maiuscole e minuscole, numeri e caratteri speciali. La lunghezza consigliata è di almeno 12 caratteri ed è consigliato di evitare l'uso d'informazioni personali come nomi, date di nascita o numeri di telefono. Le password non dovrebbero essere facilmente deducibili da informazioni pubbliche.

8.2 Cambio periodico delle password

È consigliabile introdurre regole che richiedano agli utenti di cambiare le loro password a intervalli regolari. Tuttavia, è importante fare attenzione a non impostare scadenze così ravvicinate da spingere gli utenti a scegliere password deboli o facili da indovinare.
È utile fornire istruzioni chiare e semplici per il cambio delle password. Gli utenti devono essere informati quando è il momento di cambiare la password e guidati attraverso il processo in modo rapido ed efficiente.

8.3 Controlli automatici sulle password

Negli ultimi anni, sono stati introdotti sistemi avanzati che eseguono controlli sulle password scelte dagli utenti al fine di garantire che queste siano robuste e non suscettibili a vulnerabilità di sicurezza. Questi controlli mirano a impedire l'utilizzo di password deboli o facilmente prevedibili, contribuendo così a rafforzare la sicurezza degli account utente.
Di seguito una lista dei controlli che sono applicati per rendere le password più sicure:

1. **Complessità**, la password deve rispette una lunghezza minima, di solito di 8 caratteri, oltre che a contenere caratteri maiuscoli, minuscoli, numeri, e caratteri speciali.
   Possono essere inseriti check che verificano la ripetitività della password.

2. **Check sulle informazioni personali**, la password non deve contenere informazioni personali come il nome, l'email, anno di nascita etc...
   Il sistema viene a conoscenza di queste informazioni quando un utente si registra.

3. **Password più utilizzate**, password semplici come "password" o "12345678", rappresentano vulnerabilità significative per la sicurezza degli account. Per mitigare questo rischio, i sistemi di sicurezza implementano controlli attraverso una blacklist contenente le password più frequentemente utilizzate dagli utenti. Questa lista include le password che sono spesso le prime scelte in attacchi di tipo Rainbow Table o attacchi di Dizionario (vedi capitolo 9.3, 9.4).

8.4 Autenticazione a due fattori - 2FA

L'autenticazione a due fattori (2FA) è una metodologia di sicurezza adottata sempre più frequentemente dai sistemi moderni. Questa tecnica come si intuisce dal nome prevede due diversi elementi di verifica per concedere l'accesso a un sistema, account o servizio. 2FA migliora la sicurezza rispetto all'autenticazione a singolo fattore (come l'inserimento di una sola password).

I due fattori che 2FA utilizza appartengono a queste categorie:

1. **Qualcosa che l'utente sa - Conoscenza**
   Questo è il fattore più comune ed è rappresentato generalmente dalla tradizionale password.

2. **Qualcosa che l'utente possiede - Possesso**
   Questo fattore coinvolge un oggetto fisico in possesso dell'utente, come uno smartphone. Dopo aver inserito correttamente la password, l'utente deve fornire una

seconda verifica tramite il possesso fisico di un dispositivo specifico. Per esempio è inviato un codice numerico su un app o per messaggio, questo deve essere confermato.

3. **Qualcosa che l'utente è - Biometrico**
   Questo fattore utilizza tratti fisici unici dell'utente, tra le più comuni le impronte digitali o il riconoscimento facciale, ma anche la scansione dell'iride o la voce.

L'utilizzo dell'autenticazione a due fattori (2FA) è sempre più diffuso, specialmente nei settori bancario, finanziario e sanitario, dove la protezione dei dati personali è d'importanza cruciale. Questa pratica si sta rapidamente affermando come standard di sicurezza, offrendo un ulteriore strato di protezione per garantire la sicurezza e la riservatezza delle informazioni sensibili degli utenti. La necessità di salvaguardare dati delicati, come informazioni finanziarie e sanitarie, ha reso la 2FA una componente essenziale nella strategia di difesa contro possibili minacce alla sicurezza.

Capitolo 9

Attacchi alle funzioni Hash - crittoanalisi

Il presente capitolo mira a esplorare approfonditamente gli attacchi alle funzioni Hash nel contesto della crittoanalisi. Abbiamo precedentemente esaminato le strategie attraverso cui le funzioni Hash proteggono i dati sensibili, come le informazioni personali e le password, preservando così la privacy degli utenti. Tuttavia, è altrettanto cruciale comprendere i principi alla base degli attacchi crittografici che potrebbero mettere in discussione questa sicurezza.
Questa disciplina, nota come crittoanalisi, si concentra sull'analisi e sulla comprensione dei metodi che gli aggressori mettono in atto per violare i sistemi crittografici.

9.1 Violare una funzione Hash

In questa sezione, esamineremo i criteri e gli scenari che indicano quando una funzione Hash può essere considerata vulnerabile o addirittura violata. L'analisi sarà incentrata sui concetti di collisione e inversione, fondamentali per valutare l'integrità e la sicurezza delle funzioni Hash.
Facendo riferimento alla defizione di sicurezza data nel capitolo 3.3 possiamo dire che una funzione Hsash è vulnerabile quando:

1. **Facile inversione**
   Data un impronta è facile risalire all'elemento che lo ha generato.

2. **Facile collisione**
   Trovare due valori che producano uno stessa impronta.

Se uno di questi due criteri è violato possiamo considerare la funzione non più sicura.
Nelle seguenti sezioni esamineremo alcuni dei principali metodi che sono usati per cercare vulnerabilità nelle funzioni Hash.

9.2 Attacco di forza bruta

L'Attacco di Forza Bruta rappresenta uno degli approcci più diretti per violare una funzione Hash. Consiste nell'esplorare tutte le possibili combinazioni di input fino a trovare quello che corrisponde al valore di Hash desiderato. La forza bruta sfrutta la potenza computazionale per eseguire una vasta quantità di tentativi, rendendo eventualmente possibile trovare un input che produce lo stesso Hash cercato.

Questa tipologia di attacco sfrutta la teoria del paradosso del compleanno (paragrafo 3.3.1). Secondo questa teoria, in un insieme di N elementi, è probabile trovare collisioni con probabilità significativa quando il numero di elementi testati supera $2^{N/2}$. Applicando questo principio alla ricerca di collisioni, un attaccante potrebbe trovare una collisione più rapidamente di quanto ci si aspetterebbe intuitivamente.

9.2.1 Utilizzo di GPU negli attacchi di forza bruta

Il passaggio all'utilizzo di unità di elaborazione grafica (GPU) negli attacchi di forza bruta ha rappresentato un notevole salto in termini di potenza computazionale. Le GPU sono progettate per gestire simultaneamente numerosi calcoli, rendendole estremamente efficienti nel testare molte combinazioni di input in parallelo. Questa caratteristica rende gli attacchi di forza bruta più veloci e accessibili anche per individui o gruppi con risorse di calcolo limitate.

9.3 Attacco di Rainbow Table

L'attacco basato su Rainbow Table è una tecnica che si basa sull'uso di tabelle precalcolate per invertire le funzioni Hash. Questo tipo di attacco sfrutta l'idea che molte persone utilizzano password deboli o facili da indovinare, come nomi comuni o sequenze numeriche.

Le Rainbow Table sono essenzialmente tabelle che contengono coppie di input e output già calcolate. Avendo a disposizione un valore di Hash (output), è possibile cercarlo all'interno di queste tabelle per risalire alla password originale (input).

Questo tipo di attacco è comunemente impiegato per recuperare le password in cui è stato applicato l'Hashing. Risulta particolarmente efficace in sistemi in cui le password sono Hashate senza l'utilizzo di Salt. In questo contesto, è sufficiente consultare una Rainbow Table e, se la password non è estremamente complessa, sarà possibile risalire alla password utilizzata.

Descrizione di come avviene un attacco con Rainbow Table:

1. **Intercettazione dell'Hash di un utente**
   L'attaccante in qualche modo a intercetta l'Hash di un utente. Ciò può avvenire attraverso l'accesso non autorizzato a un database contenente le password degli utenti o tramite l'intercettazione del traffico di rete.

2. **Creazione o acquisizione di una Rainbow Table**
   È fondamentale possedere una Rainbow Table estremamente grande e specifica per l'attacco in corso. Questa tabella viene creata in anticipo e contiene coppie di Hash e password corrispondenti. Una dimensione e una specificità adeguate aumentano le probabilità di trovare l'Hash desiderato nella tabella.

3. **Ricerca dell'Hash nella Rainbow Table**
   Una volta ottenuta la tabella arcobaleno, l'attaccante cerca l'Hash intercettato al passo 1. all'interno di questa tabella. Se trova una corrispondenza, può recuperare la password corrispondente a quell'Hash.

9.4 Attacchi di dizionario

Anche questa tipologia di attacchi è particolarmente utilizzata per recuperare password da un Hash noto. A differenza di attacchi con Rainbow Table gli attacchi di dizionario non precalcolano alcun Hash, ma lo calcolano in una seconda fase.
Gli attacchi di dizionario quindi utilizzano liste predefinite di parole comuni o frasi, note come "dizionari", alle quali viene aggiunto il Salt. Successivamente, l'Hash viene calcolato, cercando di ottenere corrispondenze.

Descrizione di come avviene un attacco si dizionario:

1. **Intercettazione dell'Hash e del Salt di un utente**
   L'attaccante in qualche modo a intercetta l'Hash e il Salt corrispondente di un utente. Ciò può avvenire attraverso l'accesso non autorizzato a un database contenente le password degli utenti o tramite l'intercettazione del traffico di rete

2. **Selezione del dizionario**
   L'attaccante sceglie o crea un dizionario contenente parole comuni, frasi e altre sequenze che potrebbero essere utilizzate come password.

3. **Aggiunta del Salt e calcolo dell'Hash**
   Per ciascuna parola o frase del dizionario, l'attaccante aggiunge il Salt alle stringhe e calcola l'Hash.

4. **Ricerca dell'Hash** L'attaccante confronta l'Hash noto con gli Hash calcolati di ciascuna riga, se trova una corrispondenza, può risalire semplicemente alla password dell'utente.

Questo metodo è più oneroso rispetto a un attacco che utilizza una Rainbow Table, ma offre maggiore flessibilità e adattabilità. Infatti, mentre gli attacchi con Rainbow Table richiedono una preparazione intensiva delle tabelle in anticipo, gli attacchi di dizionario calcolano gli Hash in modo dinamico durante l'attacco, consentendo una maggiore adattabilità alle caratteristiche specifiche del sistema target.
Inoltre, a differenza del metodo Rainbow Table, gli attacchi a dizionatio riescono a operare su sistemi che utilizzano il Salt.

9.5 Attacchi Side-Channel

Gli attacchi Side-Channel (canali secondari) sono strategie di attacco che si basano sull'analisi d'informazioni collaterali, come il tempo di esecuzione o il consumo di potenza, per ottenere informazioni sensibili su un sistema crittografico o su un'altra operazione segreta.
Questi attacchi sfruttano il fatto che durante l'esecuzione di un algoritmo, il sistema può involontariamente rilasciare informazioni attraverso canali secondari noti come Side Channels. I principali Side Channels che vengono analizzati includono:

1. Tempo di esecuzione
2. Consumo elettrico
3. Emissioni elettriche
4. Emissioni acustiche

Analizzando uno o una combinazione di questi effetti collaterali, esistono algoritmi che possono trovare più facilmente collisioni o invertire l'Hash.

Gli attacchi Side-Channel rappresentano una minaccia significativa per la sicurezza dei sistemi crittografici, poiché consentono agli attaccanti di estrarre informazioni sensibili in modo non convenzionale, sfruttando le vulnerabilità nei canali secondari di un sistema. Pertanto, è essenziale implementare misure di difesa contro tali attacchi, come l'uso di algoritmi resistente agli attacchi Side-Channel o l'adozione di contromisure hardware per ridurre la quantità d'informazioni rilasciate attraverso questi canali.

9.6 Raccomandazioni per soluzioni sicure

Di seguito, le principali soluzioni per rafforzare la sicurezza dei sistemi ed evitare gli attacchi appena descritti.

1. **Utilizzo di funzioni Hash più resistenti**
   Per esempio è consigliato utilizzare SHA-256 o SHA-3 per proteggersi da attacchi più resistenti.

2. **Utilizzo di Salting**
   Prima di calcolare l'Hash di un input a questo viene aggiunto un Salt, cioè un valore pseudocasuale. L'aggiunta del Salt, incrementa la sicurezza, e riduce per esempio l'efficacia di tentativi di attacchi di forza bruta. (vedi capitolo 4).

3. **Concatenazione di più funzioni Hash**
   Si intende l'applicazione di più funzioni Hash una dopo l'altra, aumentando così la sicurezza.

Capitolo 10

Sfide e sviluppi futuri

In questo capitolo, esamineremo le sfide attuali e le possibili direzioni di sviluppo future per quanto riguarda le funzioni Hash.

10.1 Tendenze e sviluppi recenti

Una delle tendenze principali riguarda l'adozione di funzioni Hash resistenti a collisioni sempre più robuste e innovative. Con l'aumento della potenza di calcolo è fondamentale garantire che le funzioni Hash siano in grado di resistere a tentativi di attacco sempre più evolute. Inoltre, è importante che funzioni Hash, come SHA-2 e SHA-3, siano sempre più utilizzate per garantire l'integrità dei dati e la sicurezza nelle applicazioni.

10.2 Blockchain e funzioni Hash

Un altro sviluppo interessante riguarda l'uso di funzioni Hash in applicazioni blockchain e cripto valute, dove sono utilizzate per crittografare i dati contenuti in ciascun blocco, garantendo che le informazioni non possano essere facilmente manipolate o alterate. Inoltre, le funzioni Hash vengono utilizzate per collegare i blocchi in modo sequenziale, creando una catena di blocchi immutabile. Questo garantisce l'integrità della blockchain e la sua resistenza a modifiche non autorizzate. In questo contesto, si stanno esplorando nuove funzioni Hash e algoritmi di consenso per migliorare l'efficienza e la scalabilità delle blockchain.

10.3 Computer quantistici e funzioni Hash

L'introduzione dei computer quantistici rappresenta una svolta significativa nel campo della computazione, poiché sfrutta i principi della meccanica quantistica per eseguire calcoli a una velocità e con una capacità di elaborazione potenzialmente molto superiori rispetto ai computer attuali. Questa avanzata tecnologia potrebbe avere un impatto significativo sulla crittografia e sulla sicurezza dei dati personali.

Le funzioni Hash, che attualmente forniscono un livello elevato di sicurezza nei sistemi crittografici, potrebbero essere influenzate dalla capacità di calcolo dei computer quantistici. I computer quantistici, sfruttando algoritmi specifici come l'algoritmo di Shor, potrebbero teoricamente risolvere in modo efficiente, per esempio, problemi di fattorizzazione, che sono alla base di molti algoritmi crittografici attuali. Nelle funzioni Hash, per esempio potrebbero effettuare un attacco di forza bruta anche su funzioni computazionalmente lente e adatte a resistere a questo tipo di attacco.

Di conseguenza, è fondamentale considerare come le funzioni Hash debbano evolversi per mantenere la sicurezza in un'era in cui i computer quantistici potrebbero potenzialmente superare gli algoritmi crittografici. Tuttavia, questo campo è ancora in fase di sviluppo e richiederà un costante adattamento e innovazione per far fronte alle sfide emergenti nella sicurezza informatica.

10.4 Intelligenza Artificiale applicata alle funzioni Hash

Negli ultimi tempi, si parla sempre di più d'intelligenza artificiale, una tecnologia in grado, dopo un periodo di addestramento, di apprendere, ragionare e creare come una mente umana. Questa avanzata tecnologica sta vivendo una diffusione esponenziale, suscitando un crescente interesse in molteplici settori.
In questo contesto, ci possiamo chiedere se possiamo utilizzare l'I.A. per migliorare le funzioni Hash. Infatti, l'intelligenza artificiale, con la sua capacità di apprendimento e adattamento, potrebbe rappresentare una risorsa innovativa nel perfezionamento delle funzioni Hash, contribuendo a ottimizzare processi e garantire un livello più elevato di sicurezza nei sistemi che le impiegano.
Di seguito, analizziamo come si potrebbe impiegare l'IA nell'ambito delle funzioni Hash.

Apprendimento

Inizialmente, utilizziamo tecniche di apprendimento automatico, in particolare il machine learning, per identificare schemi e relazioni nei dati che possono essere sfruttati per

migliorare le prestazioni delle funzioni Hash. Ad esempio, un algoritmo di apprendimento automatico potrebbe analizzare grandi dataset di input e output di una funzione Hash esistente per identificare modelli non evidenti agli occhi umani.

Ottimizzazione dei parametri

L'IA può anche essere utilizzata per ottimizzare automaticamente i parametri delle funzioni Hash. Attraverso algoritmi di ottimizzazione, l'IA può esplorare lo spazio dei parametri in modo efficiente, cercando combinazioni ottimali che massimizzino la resistenza alle collisioni e minimizzino il rischio di attacchi crittografici.

Generazione e modifica di Funzioni Hash

L'Intelligenza Artificiale può esaminare attentamente le potenziali vulnerabilità presenti nelle funzioni Hash esistenti. Attraverso un'analisi dettagliata dei dati di input e output, è in grado d'individuare schemi che potrebbero essere sfruttati da un potenziale attaccante per compromettere la sicurezza delle Hash.
Dopo questa fase iniziale di analisi, è possibile procedere con la creazione di funzioni Hash più robuste e sicure o alla modifica di funzioni già esistenti. È importante sottolineare che il processo di progettazione può beneficiare della collaborazione umana, che può apportare correzioni e miglioramenti al lavoro dell'IA.

Capitolo 11

Conclusioni

La presente tesi si è concentrata sull'importanza delle funzioni Hash nel contesto della sicurezza delle password, analizzando i rischi associati alle pratiche di archiviazione e gestione delle password. Dai fondamenti della crittografia e delle funzioni Hash, passando per l'analisi di algoritmi come SHA, RIPEMD e MD, fino alle nuove frontiere rappresentate da algoritmi avanzati come Bcrypt e Argon2, è emerso un quadro completo delle sfide e delle best practices nel garantire la sicurezza delle password.

L'analisi dettagliata degli algoritmi Hash, comprese le implementazioni come SHA-3, ha fornito una panoramica approfondita delle caratteristiche e delle vulnerabilità potenziali. Le funzioni di hash avanzate, come Bcrypt e Argon2, hanno dimostrato di fornire soluzioni più robuste nel campo della sicurezza delle password. In particolare, queste funzioni sono progettate per offrire un livello superiore di protezione rispetto agli algoritmi di hash più tradizionali.

L'utilizzo di Bcrypt e Argon2 è particolarmente vantaggioso per la sicurezza delle password, poiché sono progettati per rallentare deliberatamente il processo di hashing. Ciò rende più difficile per gli attaccanti eseguire attacchi di forza bruta o di dizionario, in cui tentano di indovinare la password attraverso l'esplorazione di molte possibili combinazioni..

Le best practices per la sicurezza delle password, tra cui la complessità delle password, il cambio periodico delle stesse, i controlli automatici e l'autenticazione a due fattori, sono state esaminate come strumenti essenziali per mitigare i rischi associati alle password.

Nel contesto degli attacchi alle funzioni Hash, la tesi ha esaminato le diverse strategie utilizzate dagli attaccanti, come l'attacco di rainbow table, gli attacchi di dizionario e gli attacchi Side-Channel. Le raccomandazioni fornite offrono soluzioni pratiche per prevenire tali attacchi e migliorare la sicurezza complessiva del sistema.

Infine, la tesi ha esplorato le sfide e gli sviluppi futuri, tra cui le tendenze recenti, l'impatto potenziale dei computer quantistici sulle funzioni Hash e una possibile applicazione dell'intelligenza artificiale.

In sintesi, la sicurezza delle password è una componente critica nella protezione delle informazioni sensibili. Le funzioni Hash e le pratiche di gestione delle password delineate in questa tesi forniscono una base solida per affrontare le sfide attuali e future nel campo della sicurezza informatica.

Bibliografia

- Cryptography, Theory and Practise, Douglas R. Stinson, Maura B. Peterson
- Cryptography and Network Security, Principles and Practice, William Stallings
- Argon2: new generation of memory-hard functions for password Hashing and other applications, Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich
- https://www.cs.unibo.it/margara/page2/page6/page25/assets/biondiHash.pdf
- https://it.wikipedia.org/wiki/Crittografia
- https://it.wikipedia.org/wiki/MD5
- https://it.wikipedia.org/wiki/Secure_Hash_Algorithm
- https://it.wikipedia.org/wiki/RIPEMD
- https://en.wikipedia.org/wiki/Argon2
- https://en.wikipedia.org/wiki/Blowfish_(cipher)
- https://it.wikipedia.org/wiki/Bcrypt
- https://it.wikipedia.org/wiki/Costruzione_di_Merkle-Damg