Course: Crittografia

Crittografia applicata alla sicurezza di tutti i giorni: Bitwarden

Author: Edoardo Desiderio

Instructor: **Prof. Luciano Margara**

Indice

Capitolo 1

Introduzione

1.1 Scopo del Documento

L'idea della ricerca nasce poichè confrontandomi con amici e colleghi, ho notato che molti di loro studiavano il campo della crittografia da un punto di vista dei malware e dei ransomware, ma non da un punto di vista positivo. Questo documento ha lo scopo di fornire una panoramica generale della crittografia e delle sue applicazioni positive, con particolare attenzione ai password manager.

1.2 L'Uso Positivo della Crittografia

La crittografia, un campo di studio che si occupa della protezione delle informazioni attraverso l'uso di codici, ha un ruolo fondamentale nel mondo digitale di oggi. Attraverso l'utilizzo di complessi algoritmi matematici, la crittografia protegge i dati sensibili, garantisce la riservatezza delle comunicazioni, assicura l'integrità dei dati e favorisce un commercio elettronico sicuro. È uno strumento cruciale per proteggere la nostra privacy e preservare la sicurezza dei nostri dati. La crittografia è un elemento fondamentale per la cybersecurity, capace di assicurare una protezione efficace e duratura nel tempo dei sistemi e dei servizi a cui viene applicata.

Introduzione ai Password Manager

Un password manager è un sistema di sicurezza informatica, un programma che permette di creare password uniche per ogni singolo account, conservarle in un luogo sicuro e accedere ad esse attraverso un'estensione del browser o una app, sia da un computer che da un dispositivo mobile come tablet

o smartphone. Questi strumenti consentono agli utenti di sincronizzare le password tra vari dispositivi, e possono o meno conservare le password e i dati anche sul dispositivo. Il concetto principe di un password manager è quello di accedere ad una password unica, detta master password, che impastata rappresenta la chiave privata del mio storage di password.

1.3 Storia

La storia dei password manager è intrinsecamente legata all'evoluzione della sicurezza informatica. Le password, come metodo di autenticazione, hanno radici antiche, risalenti all'antica Grecia e utilizzate per proteggere segreti militari durante la Seconda Guerra Mondiale. Con l'avvento dei computer negli anni '60, le password hanno iniziato a diventare parte della vita quotidiana.

Il primo password manager della storia è stato sviluppato nel 1990 da Mark Thompson [7] e si chiama "password Safe" e fu introdotto come software utility per windows 95.

1.4 Passwword Save e l'algoritmo di Blowfish

Password Safe, nella sua versione originale, utilizzava l'algoritmo di crittografia **Blowfish** per proteggere le password memorizzate. Blowfish è un algoritmo di crittografia a blocchi simmetrico sviluppato da Bruce Schneier nel 1993.
All'avvio, l'applicazione chiedeva all'utente di creare un nuovo archivio di password, che poteva essere salvato in qualsiasi posizione desiderata dall'utente. Dopo aver creato l'archivio, l'applicazione chiedeva all'utente di impostare una password principale. Questa password veniva utilizzata per bloccare l'accesso all'archivio delle password. Era l'unica password che l'utente doveva ricordare.
Una volta impostata la password principale, l'utente poteva iniziare a memorizzare le proprie password e altre credenziali di accesso nell'archivio. Quando l'utente aveva bisogno di accedere alle sue password, doveva aprire l'applicazione Password Safe, inserire la password principale e quindi avrebbe avuto accesso all'archivio delle password.
In questo modo, Password Safe offriva un modo sicuro per memorizzare tutte le password in un unico luogo, proteggendole con una sola password principale. Questo metodo di gestione delle password è ancora utilizzato nelle versioni più recenti di Password Safe e in molti altri gestori di password

Punti chiave

- **Crittografia Simmetrica:** Usa la stessa chiave[1] per crittografare e decrittografare i dati.
- **Lunghezza della Chiave Variabile:** Supporta chiavi di lunghezza variabile, da 32 a 448 bit, rendendolo flessibile in base alle esigenze di sicurezza.
- **Dimensione del Blocco:** Opera su blocchi di dati di 64 bit.
- **S-Box:** Utilizza strutture interne note come S-box per realizzare la crittografia.

[1] la main password dell'utente

1.5 Funzionamento di Blowfish

Blowfish utilizza un insieme di operazioni come sostituzioni e permutazioni, gestite attraverso S-box per trasformare il testo in chiaro in testo cifrato. Ogni blocco di dati viene elaborato in 16 round di crittografia.

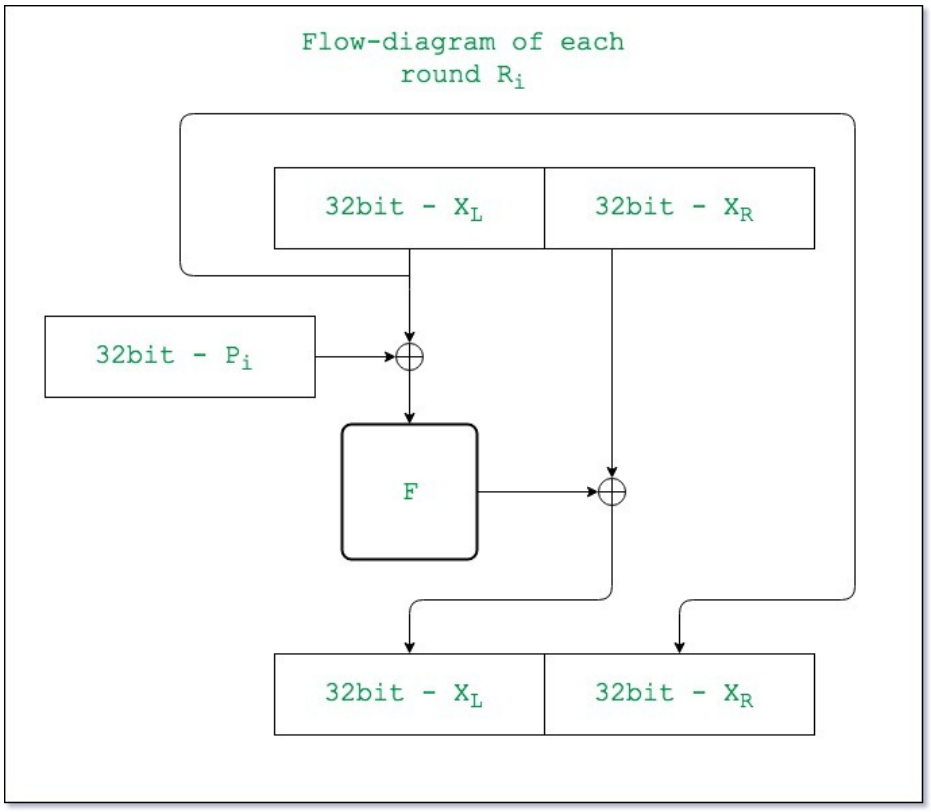

Figura 1.1: grafico che cattura il processo di crittografia dell'algoritmo [4]

Processo di Crittografia e Decrittografia

il processo principale di crittografia e decrittografia di Blowfish sta tutto nella funzione `f` che utilizza le S-box per creare una funzione non lineare che contribuisce alla sicurezza dell'algoritmo.

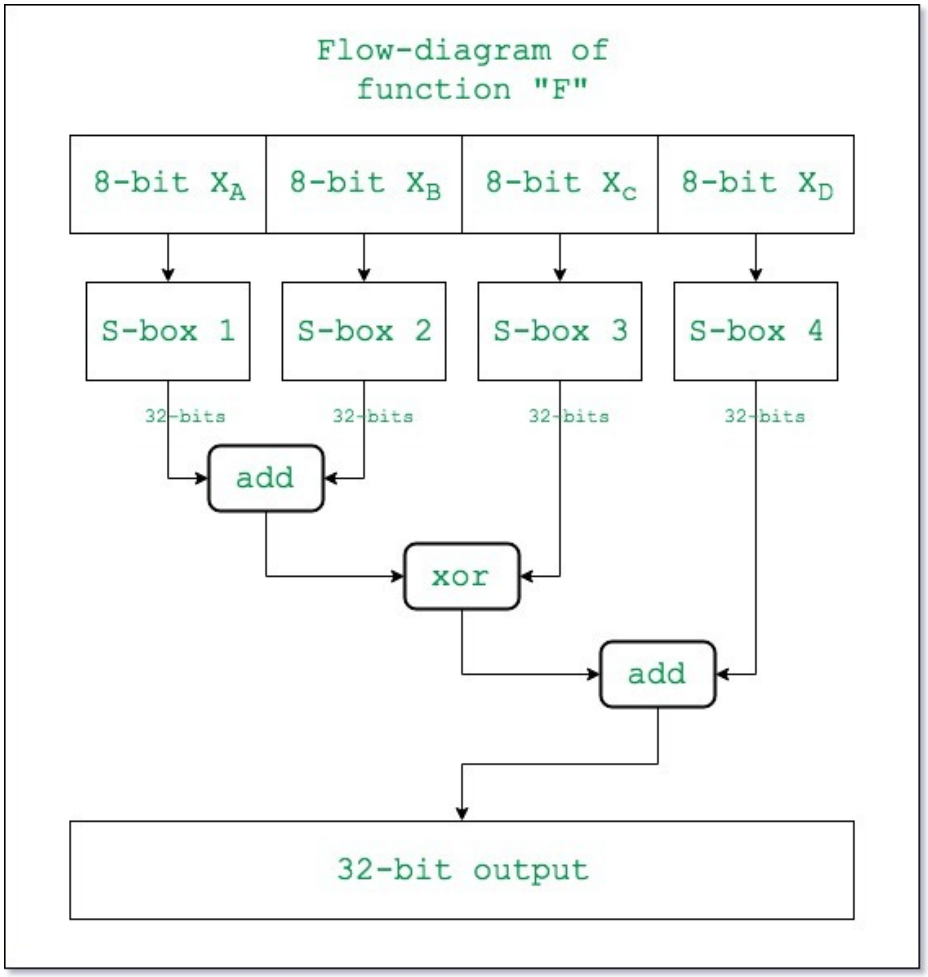

Figura 1.2: metodo f dell'algoritmo [4]

Funzione F dell'algoritmo Blowfish

La funzione F prende in input un valore di 32 bit x e restituisce un valore di 32 bit. Essa viene definita come segue:

$$F(x) = ((S_1[a] + S_2[b] \bmod 2^{32}) \oplus S_3[c]) + S_4[d] \bmod 2^{32}$$

- S_1, S_2, S_3, S_4 sono le S-box dell'algoritmo.
- a byte più significativo di x.
- b = secondo byte più significativo di x

- c = secondo byte meno significativo di x
- d = byte meno significativo di x

```
function F(x):
  result = ((S1[a] + S2[b] mod 2^32) xor S3[c])
     + S4[d] mod 2^32

  return result
```

conclusioni sul funzionamento di Blowfish

in pratica l'algoritmo di crittografia Blowfish si differenzia principalmente dal DES visto a lezione per la sua chiave variabile fino a 448 bit. [5]

L'obsolescenza di Blowfish nella prima versione di *Password Safe*

Uno dei principali punti deboli di Blowfish risiede nella sua progettazione con una rete di Feistel di 16 round descritta nei paragrafi precedenti. Sebbene non vi siano attacchi pratici noti che possano rompere Blowfish con meno di 16 round, la struttura fissa e il numero limitato di round non garantiscono la stessa sicurezza di algoritmi più moderni come l'AES (Advanced Encryption Standard), che offre una configurazione più robusta e una gestione più sicura delle chiavi. Inoltre, la progettazione di Blowfish non è ottimizzata per le implementazioni hardware moderne, risultando vulnerabile a tecniche come gli attacchi *time-memory trade-off* (TMTO) [2] e l'utilizzo di tabelle precomputate.

Con il tempo, la crittografia è diventata un campo in continua evoluzione,

con attacchi sempre più sofisticati come quelli differenziali e lineari. Mentre Blowfish non è stato completamente compromesso da tali attacchi, la sua mancanza di aggiornamenti e adattamenti alle nuove minacce lo rende meno preferibile rispetto ad altri algoritmi che hanno subito una revisione continua e miglioramenti.

Un'altra considerazione critica riguarda la derivazione

delle chiavi. La prima versione di *Password Safe* potrebbe non aver implementato meccanismi di derivazione delle chiavi robusti, come PBKDF2,

[2] Per maggiori informazioni sugli attacchi TMTO, si può consultare il seguente link: `https://en.wikipedia.org/wiki/Time/Memory/Data_Tradeoff_attack`

bcrypt o Argon2, che sono progettati per resistere agli attacchi a forza bruta implementando salting e iterazioni multiple. Questo rende particolarmente problematico l'uso di Blowfish in contesti moderni, dove la sicurezza a lungo termine è essenziale.

Inoltre, il supporto per chiavi di lunghezza fino a 448 bit, sebbene teoricamente sufficiente, non offre le stesse garanzie di sicurezza di AES con chiavi di lunghezza 256 bit, che è considerato lo standard di sicurezza per molte applicazioni critiche. La differenza non è solo nella lunghezza delle chiavi, ma anche nella resistenza agli attacchi, nella velocità di cifratura e nella versatilità in diversi ambienti hardware.e l'utilizzo di tabelle precomputate.

Un'altra considerazione critica riguarda la derivazione delle chiavi. La prima versione di *Password Safe* potrebbe non aver implementato meccanismi di derivazione delle chiavi robusti, come PBKDF2, bcrypt o Argon2, che sono progettati per resistere agli attacchi a forza bruta implementando salting e iterazioni multiple. Questo rende particolarmente problematico l'uso di Blowfish in contesti moderni, dove la sicurezza a lungo termine è essenziale. Inoltre, il supporto per chiavi di lunghezza fino a 448 bit, sebbene teoricamente sufficiente, non offre le stesse garanzie di sicurezza di AES con chiavi di lunghezza 256 bit, che è considerato lo standard di sicurezza per molte applicazioni critiche. La differenza non è solo nella lunghezza delle chiavi, ma anche nella resistenza agli attacchi, nella velocità di cifratura e nella versatilità in diversi ambienti hardware.

Capitolo 2

Bitwarden

2.1 Introduzione

Bitwarden è un password manager open source che offre una soluzione sicura per memorizzare e gestire le password. È disponibile su diverse piattaforme, tra cui desktop, web, mobile e browser. Bitwarden offre funzionalità di sincronizzazione, condivisione e generazione di password, nonché un'interfaccia intuitiva e facile da usare.

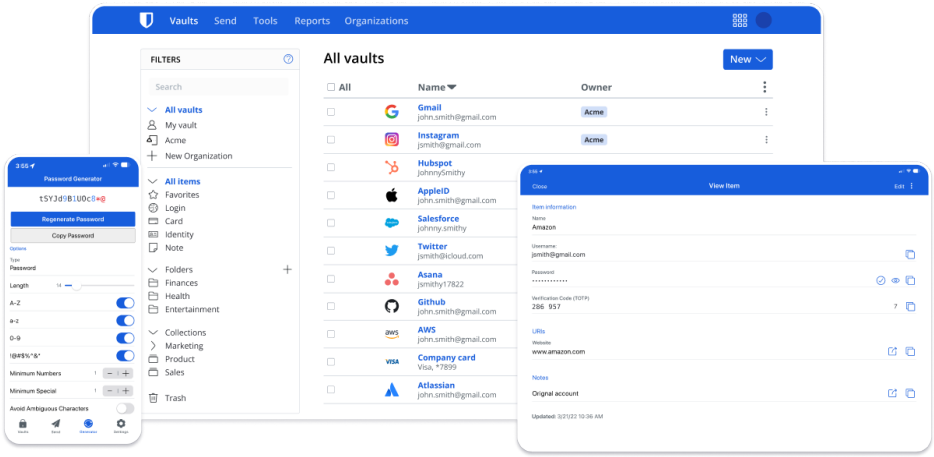

Figura 2.1: l'enviroment di Bitwarden

2.2 missione del progetto

Bitwarden è un progetto open source che si pone l'obiettivo di fornire una soluzione sicura e affidabile per la gestione delle password.

La missione di Bitwarden è quella di proteggere la privacy e la sicurezza degli utenti, offrendo un servizio gratuito e open source che garantisce la riservatezza dei dati e la sicurezza delle informazioni personali. Rendendo il suo codice sorgente pubblico nel 2016, Bitwarden ha dimostrato il suo impegno per la trasparenza e la sicurezza, consentendo agli utenti di verificare la qualità del software e la sicurezza delle loro password.

Come *password safe*, Bitwarden utilizza la master password dell'utente per crittografare e proteggere le password memorizzate, garantendo che solo l'utente possa accedere ai propri dati. Ma la domanda a cui vogliamo rispondere con questa ricerca è: come un software moderno opera in un mondo informatico multi device garantendo la massima sicurezza quindi sia in locale ma anche in cloud?

Capitolo 3

Registrazione di un utente su Bitwarden

Il processo di hashing della master password in un sistema di gestione delle password come Bitwarden segue diversi step che servono a generare una chiave derivata sicura per proteggere le password memorizzate. Questo processo permette di non rendere noto il valore della master password nemmeno al server storage di Bitwarden.[1]
Le fasi più critiche e fondamentali verranno prese in esame inseguito.

3.1 generazione della Master Key

La master password dell'utente viene utilizzata per generare una chiave segreta, chiamata *master key*, che verrà utilizzata per cifrare e decifrare i dati memorizzati. La master key viene derivata dalla combinazione della password principale dell'utente e dell'indirizzo email tramite la funzione di derivazione delle chiavi PBKDF2. Questo processo utilizza 600.000 iterazioni di HMAC-SHA256, producendo una chiave di 256 bit.

```
1       function generate_master_key(password, email):
2         derived_key = PBKDF2(password, email,
             600000, 256)
3         master_key = derived_key[0:32]
4
5         return master_key
```

3.1.1 la funzione PBKDF2

PBKDF2 (Password-Based Key Derivation Function 2) è un meccanismo di derivazione di chiavi basato su password ampiamente utilizzato per aumentare la sicurezza delle password memorizzate. Il processo inizia con l'aggiunta di un "sale", tipicamente una stringa casuale, alla password dell'utente. Questo aiuta a proteggere contro gli attacchi di dizionario e rainbow table, rendendo ogni hash unico anche se la password originale è comune. Successivamente, il valore combinato di password e sale viene passato attraverso un algoritmo di hash crittografico, in questo caso *HMAC-SHA-256*. HMAC sta per Hash-based Message Authentication Code, che è un tipo di funzione di hash che fornisce sia l'integrità dei dati che l'autenticazione del messaggio. SHA-256 è parte della famiglia di algoritmi Secure Hash Algorithm e produce un hash di lunghezza fissa di 256 bit.[2]

Dopo il primo hashing, il valore ottenuto viene nuovamente salato e hashato molteplici volte. Il numero di iterazioni, *streaching-function*[6], è configurabile e serve a rendere il processo di derivazione della chiave intenzionalmente lento. Questo rallentamento è critico per la sicurezza, poiché rende gli attacchi di forza bruta e di ricerca esaustiva molto più dispendiosi in termini di tempo e risorse computazionali. Con ogni iterazione, l'hash diventa più resistente agli attacchi, aumentando così la sicurezza della chiave derivata.

La chiave finale ottenuta dopo l'ultima iterazione è la chiave principale, che non è altro che un altro hash che sarà utilizzato per ulteriori operazioni crittografiche, come l'hash della password principale. Quest'ultimo è il valore che viene effettivamente utilizzato per verificare l'identità dell'utente durante il processo di autenticazione. Ogni volta che l'utente inserisce la sua password, il sistema ripete il processo di derivazione della chiave e confronta l'hash risultante con quello memorizzato. Se corrispondono, l'accesso è concesso.

```
1  function PBKDF2(password, salt, iterations,
       key_length):
2    key = password
3    for i = 1 to iterations:
4      key = HMAC-SHA256(key, salt)
5    return key[0:key_length]
```

3.1.2 HMAC-SHA-256

mentre a lezione abbiamo studiato e visto il funzionamento di SHA-256, mi sento di approfondire il funzionamento di HMAC-SHA-256.
HMAC sta per Hash-based Message Authentication Code, in questo caso specifico la funzione di hash scelta è quella vista a lezione, ora vediamo come la opera per rendere sicura l'informazione.[3]

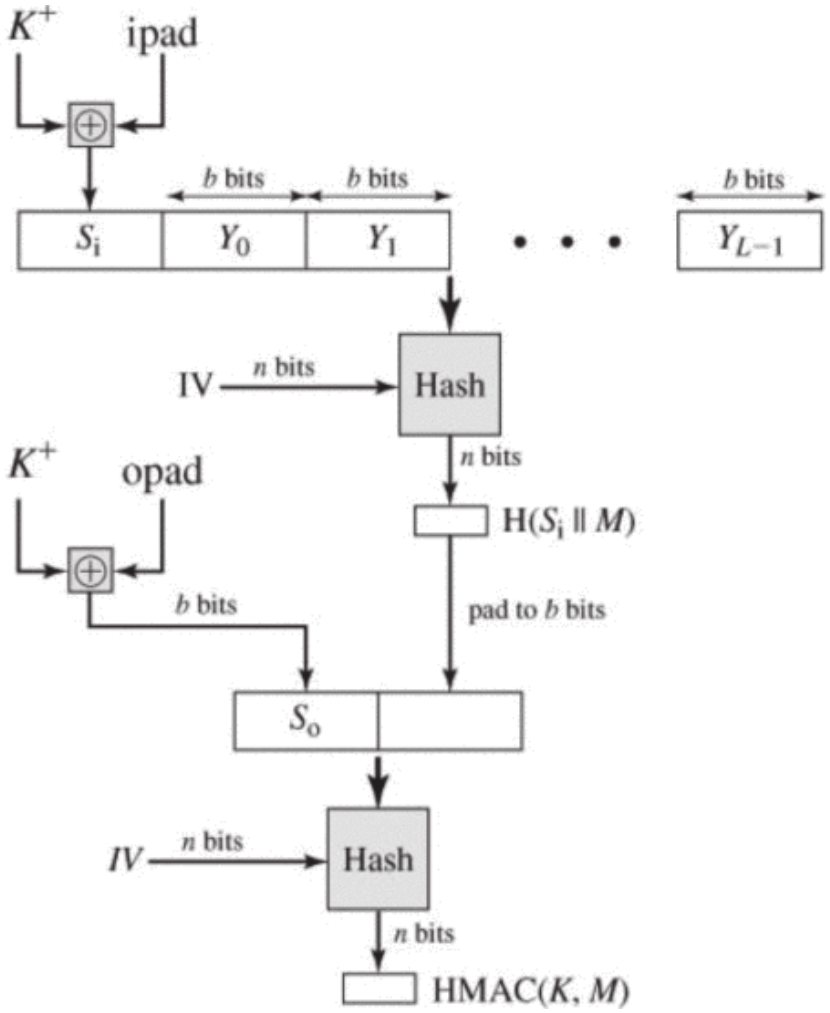

Figura 3.1: funzionamento di HMAC-SHA-256

1. Aggiungere zeri all'inizio di K per creare una stringa di bit b chiamata K^+.

2. Fare lo XOR tra K^+ e *ipad* per produrre il blocco di bit b S_i.

3. Aggiungere M a S_i.
4. Applicare l'algoritmo di hash H al flusso generato nel passaggio 3.
5. Fare lo XOR tra K^+ e *opad* per produrre il blocco di bit b S_0.
6. Aggiungere il risultato dell'hash del passaggio 4 a S_0.
7. Applicare l'algoritmo di hash H al flusso generato nel passaggio 6 e produrre il risultato finale.

3.2 Stretched Master Key

La master key risultante viene ulteriormente elaborata tramite l'algoritmo HKDF con SHA-256, che include una fase di espansione delle chiavi, per produrre una chiave estesa di 512 bit. Questa chiave estesa è utilizzata per cifrare i dati memorizzati nella cassaforte digitale dell'utente, compresi i vari segreti e le chiavi di cifratura.

```
function stretch_master_key(master_key):
  stretched_key = HKDF(master_key, 512)

  return stretched_key
```

3.2.1 HKDF

HKDF (HMAC-based Key Derivation Function) è un algoritmo di derivazione delle chiavi basato su HMAC che consente di generare chiavi di lunghezza variabile a partire da una chiave segreta.

HKDF è un meccanismo per derivare chiavi crittografiche sicure da un materiale di chiave iniziale (Initial Key Material, IKM). L'algoritmo è suddiviso in due fasi principali: estrazione ed espansione.

Fase 1: Estrazione

Scopo: Estrarre una chiave pseudocasuale (PRK) dal materiale di chiave iniziale.
Procedura: Si utilizza una funzione HMAC con un hash crittografico (ad esempio, SHA-256). Il sale ('salt') è un valore opzionale; se non fornito, si usa una stringa di zeri della lunghezza dell'hash.

$$\text{PRK} = \text{HMAC}_{\text{hash}}(\text{salt}, \text{IKM})$$

Fase 2: Espansione

Scopo: Derivare una o più chiavi di output (Output Key Material, OKM) dalla PRK.
Procedura: Utilizzando nuovamente HMAC, la PRK è impiegata come chiave per l'HMAC e un contesto ('info') come input. L'output è una concatenazione di più blocchi HMAC.

$$T(0) = \emptyset$$

$$T(1) = \text{HMAC}_{\text{hash}}(\text{PRK}, T(0) \| \text{info} \| 0x01)$$

$$T(2) = \text{HMAC}_{\text{hash}}(\text{PRK}, T(1) \| \text{info} \| 0x02)$$

$$\dots$$

$$T(n) = \text{HMAC}_{\text{hash}}(\text{PRK}, T(n-1) \| \text{info} \| n)$$

$$\text{OKM} = T(1) \| T(2) \| \dots \| T(n)$$

Dove:

- $\|$ denota la concatenazione.
- n è il numero di blocchi necessari per ottenere la lunghezza desiderata dell'OKM.

Caratteristiche Principali

- **Sicurezza**: HKDF è progettato per essere resistente a vari attacchi, mantenendo segrete le chiavi derivate anche se una di esse viene compromessa.
- **Modularità**: Può utilizzare diverse funzioni hash, adattandosi a diverse esigenze di sicurezza.
- **Efficienza**: È efficiente in termini computazionali e di memoria.

1. Generazione del Salt

Un valore casuale, chiamato *salt*, viene generato per ogni utente:

$$\text{salt} = \text{random\_bytes}(n)$$

dove n è la lunghezza del salt in byte.

2. Concatenazione della Password e del Salt

La master password dell'utente P viene concatenata con il salt:

$$\text{input} = P \| \text{salt}$$

dove $\|$ rappresenta l'operazione di concatenazione.

3. Hashing

L'input concatenato viene passato attraverso una funzione di hashing sicura, come SHA-256:

$$\text{hash} = \text{SHA-256}(\text{input})$$

4. Iterazione (PBKDF2)

Per aumentare la sicurezza, l'hash viene iterato utilizzando PBKDF2 (Password-Based Key Derivation Function 2):

$$\text{derived\_key} = \text{PBKDF2}(\text{hash}, \text{salt}, \text{iterations}, \text{key\_length})$$

dove *iterations* è il numero di iterazioni e *key_length* è la lunghezza della chiave derivata.

5. Memorizzazione

Il risultato finale, *derived_key*, insieme al salt, viene memorizzato nel database:

$$\text{stored\_value} = \{\text{salt}, \text{derived\_key}\}$$

Questo processo garantisce che anche se due utenti hanno la stessa master password, i valori hash memorizzati saranno diversi grazie all'uso del salt. Inoltre, l'uso di PBKDF2 rende l'attacco di forza bruta molto più difficile. ecco lo pseudocode completo dell'operazione:

```
function hash_master_password(password, salt):
  input = password || salt
  hash = SHA-256(input)
  derived_key = PBKDF2(hash, salt, iterations,
      key_length)
  stored_value = {salt, derived_key}

  return stored_value
```

PBKDF2 o Argon2?

Dalla versione 2023.2.0 di Bitwarden, è stata inserita la possibilità di scegliere tra PBKDF2 e Argon2 come algoritmo di derivazione delle chiavi per l'hashing della master password. Argon2 è un algoritmo di derivazione delle chiavi. Vincitore del concorso Password Hashing Competition (PHC) nel 2015, Argon2 è considerato uno degli algoritmi

Bibliografia

[1] Bitwarden. Bitwarden security principles. https://bitwarden.com/help/bitwarden-security-white-paper/#user-data-protection:~:text=When%20the%20Create,to%20Bitwarden%20servers. Accessed: 2/08/2024 18:00.

[2] Bitwarden. Pbkdf2. https://bitwarden.com/help/kdf-algorithms/#:~:text=PBKDF2%2C%20as%20implemented%20by%20Bitwarden%2C%20works%20by%20salting%20your%20master-,password%20with%20your%20username%20and%20running%20the%20resultant%20value%20through%20a,password%20hash%20used%20to%20authenticate%20that%20user%20whenever%20they%20log%20in,-(learn%20more). Accessed: 2/08/2024 21:30.

[3] Dilli Ravilla. Implementation of hmac-sha256 algorithm for hybrid routing protocols in manets. https://ieeexplore.ieee.org/abstract/document/7060558?casa_token=1YWQga6nPxEAAAAA:a-bb9cwvSG0TBPPO1JXc5L5mXQ8w-sy_BOxTW-WD99oUVftEkr6kmOdY8d4dZTfCz42jsoVZ4Q. Accessed: 2/08/2024 23:13.

[4] GeeksforGeeks contributors. Blowfish algorithm. https://www.geeksforgeeks.org/blowfish-algorithm-with-examples/. Accessed: 2/07/2024 15:00.

[5] GeeksforGeeks contributors. Blowfish algorithm. https://www.baeldung.com/cs/des-vs-3des-vs-blowfish-vs-aes, note = Accessed: 2/07/2024 15:30.

[6] Levent Ertaul, Manpreet Kaur, Venkata Arun Kumar R Gudise. Implementation and performance analysis of pbkdf2, bcrypt, scrypt algorithms. http://mcs.csueastbay.edu/~lertaul/PBKDFBCRYPTCAMREADYICWN16.pdf. Accessed: 2/08/2024 22:30.

[7] Wikipedia contributors. Password manager. https://en.wikipedia.org/wiki/Password_manager. Accessed: 2/07/2024 12:00.

Riferimenti bibliografici