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.

1.3 concetti chiave per questa ricerca

bisogna tener presente che ci sono 2 concetti che sono fondamentali su cui si basano tutti gli algoritmi di questa ricerca.

Ogni algoritmo visto nelle pagine seguenti implementa questi 2 concetti in modo diverso, ma il principio rimane lo stesso.

1.3.1 Confusione

Confusione si riferisce alla capacità di nascondere la relazione tra il testo cifrato e la chiave crittografica. L'obiettivo è rendere complicato per un attaccante dedurre la chiave o il testo in chiaro basandosi sul testo cifrato. Questo principio si realizza attraverso operazioni non lineari e sostituzioni complesse, come le tabelle S-Box utilizzate in algoritmi simmetrici come AES. Queste trasformazioni rendono non prevedibile la modifica del testo cifrato in risposta a cambiamenti nei dati o nella chiave.

1.3.2 Diffusione

Diffusione significa che una modifica in un singolo bit del testo in chiaro dovrebbe influenzare molti bit nel testo cifrato. Questo rende difficile per un attaccante capire quale parte del testo cifrato corrisponde a quale parte del testo in chiaro. La diffusione si ottiene tipicamente attraverso operazioni che mescolano i dati, come le trasformazioni lineari e le permutazioni. Per esempio, nel DES e nell'AES, operazioni come "MixColumns" e "ShiftRows" diffondono i dati su più byte, incrementando così la sicurezza.

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.

Capitolo 2

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 [9] e si chiama "password Safe" e fu introdotto come software utility per windows 95.

2.1 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

2.2 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.

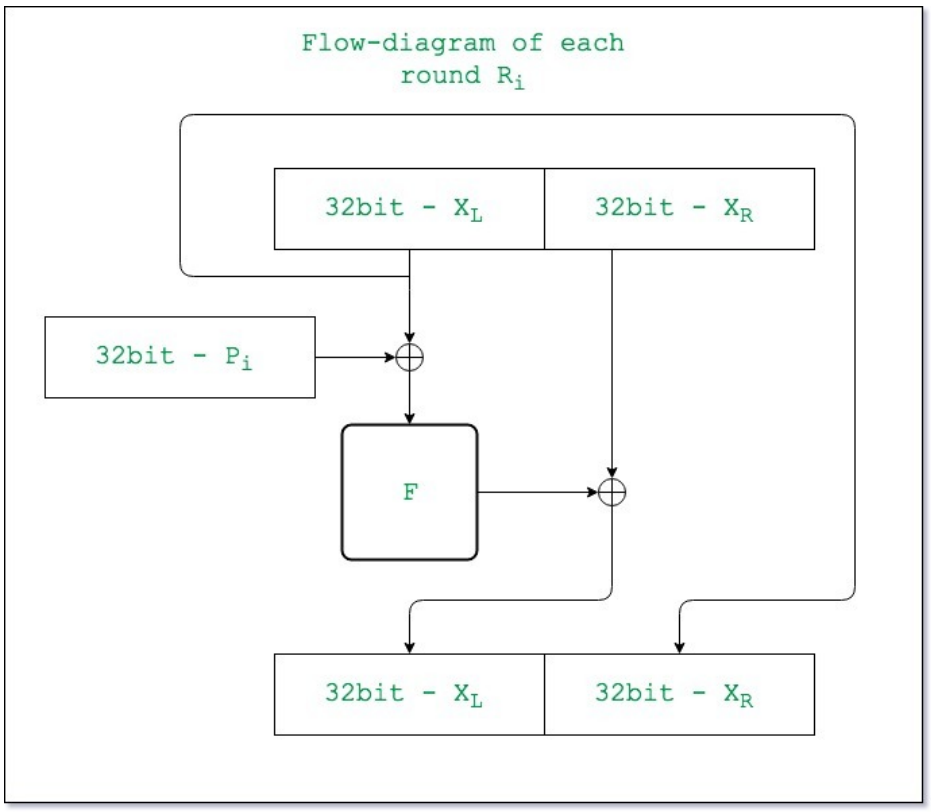


Figura 2.1: grafico che cattura il processo di crittografia dell'algoritmo [5]

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.

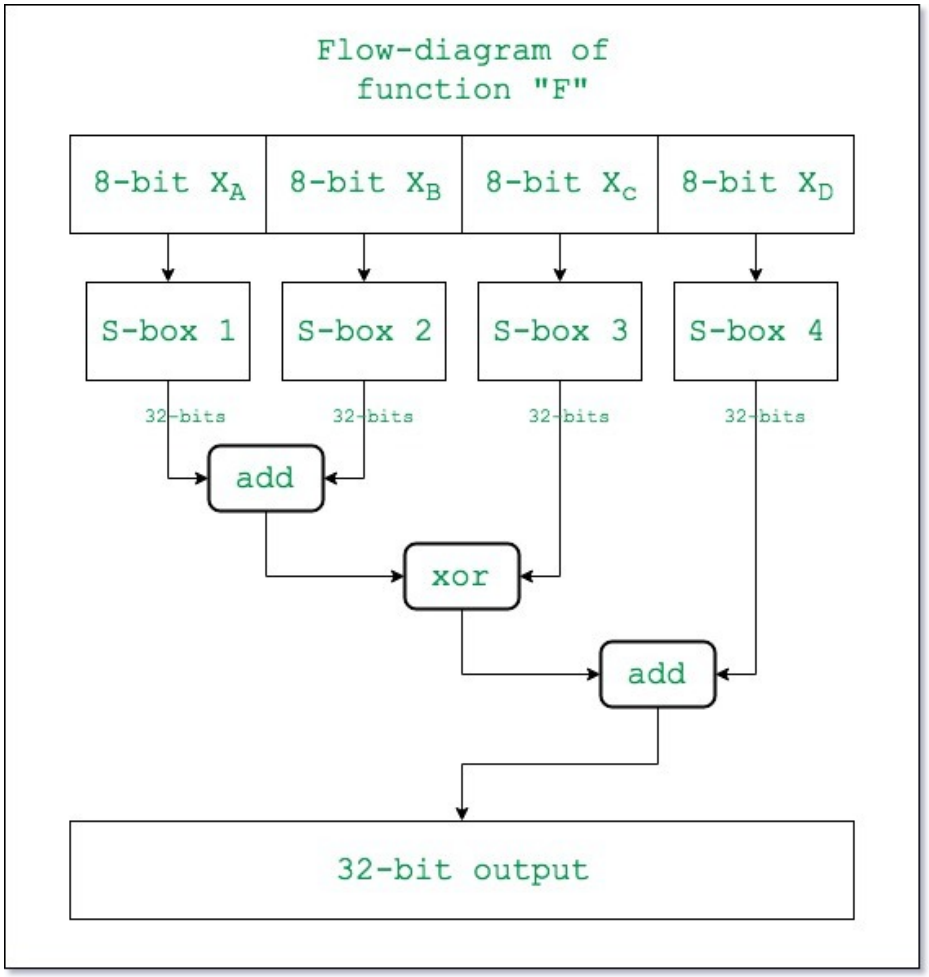


Figura 2.2: metodo f dell'algoritmo [5]

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. [6]

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.

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

[2]Per maggiori informazioni sugli attacchi TMTO, si può consultare il seguente link

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.

implementazione completa di Blowfish

utilizzando la funzione `F` e le S-box, possiamo scrivere un pseudocodice che rappresenta l'intero processo di crittografia e decrittografia di Blowfish.

```
function encrypt_block(block, key):
  initialize_S_boxes(key)
  left = block[0:32]
  right = block[32:64]
  for i = 1 to 16:
    temp = left
    left = right
    right = temp xor F(right) xor P[i]
  temp = left
  left = right
  right = temp
  return left || right
```

dove `P` è un array di 18 elementi che contiene i valori delle chiavi derivate dalla chiave principale dell'utente.

Capitolo 3

Bitwarden

3.1 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.

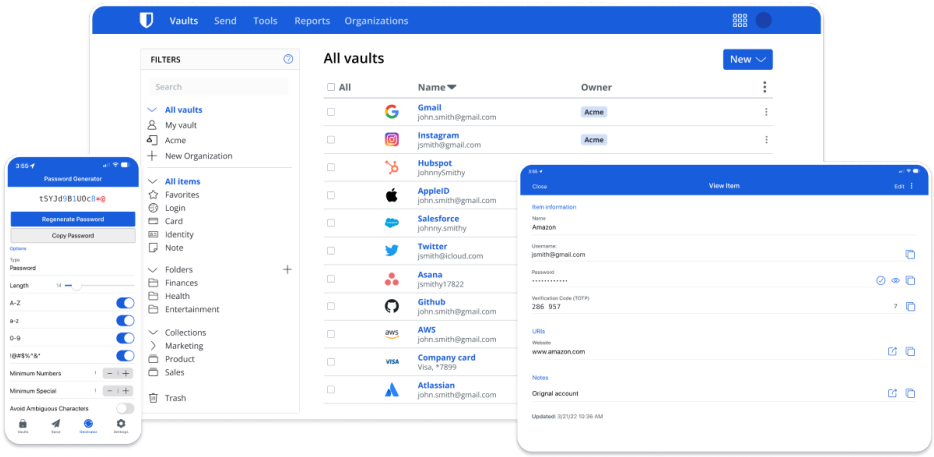


Figura 3.1: l'enviroment di Bitwarden

3.2 missione del progetto

Bitwarden 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 4

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.[2]
Le fasi più critiche e fondamentali verranno prese in esame inseguito.

4.1 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
```

4.1.1 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.[3]

Dopo il primo hashing, il valore ottenuto viene nuovamente salato e hashato molteplici volte. Il numero di iterazioni, *streaching-function*[8], 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]
```

4.1.2 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.[4]

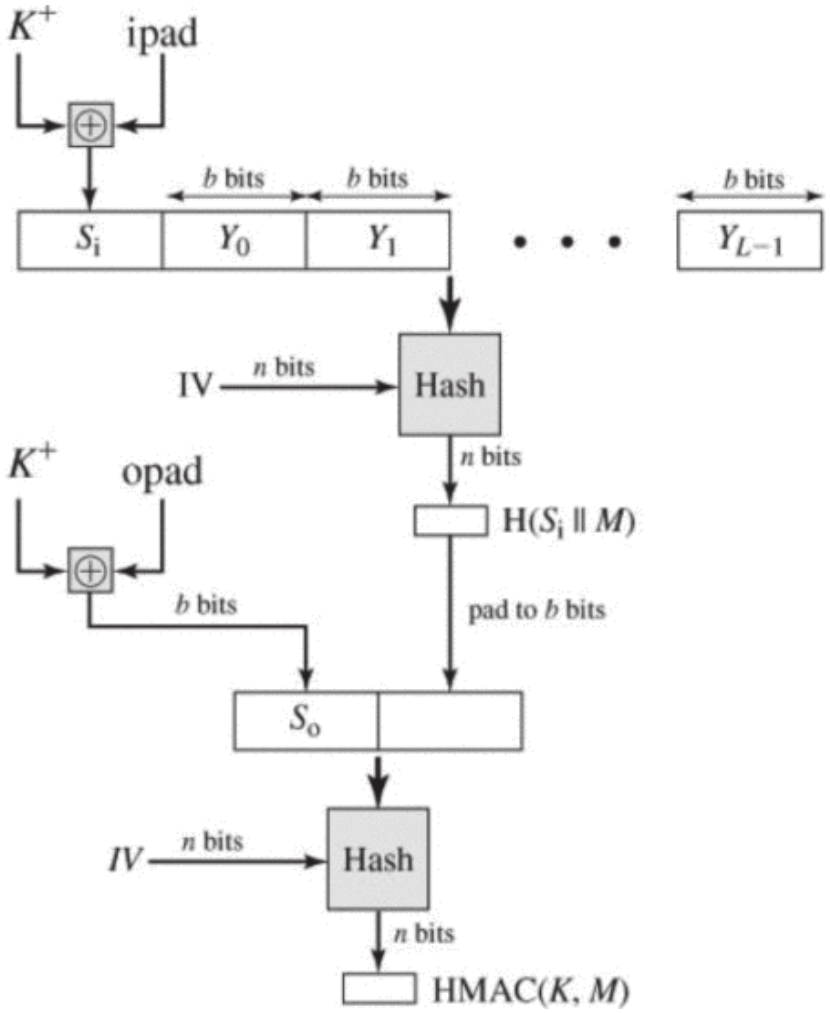


Figura 4.1: funzionamento di HMAC-SHA-256

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.

4.2 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
```

4.2.1 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[7].

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

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. [7]

$$T(0) = \emptyset$$

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

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

$$\dots$$

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

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

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.

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
```

4.2.2 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. Vediamo come si comporta e che differenze ha rispetto a PBKDF2.
La differenza fra PBKDF2 e Argon2 sta nel fatto che Argon2 sfrutta una memoria più grande e un tempo di esecuzione più lungo, grazie alla sua funzione principale di stretching.

fase di espanzione di Argon2

una volta presa una stringa da criptare, Argon2 si comporta nel seguente modo:

1. Viene generato un blocco di memoria di dimensione fissa, chiamato *lanes*, che contiene il materiale di chiave iniziale (IKM).

2. Il blocco di memoria viene diviso in segmenti di dimensione fissa, chiamati *lanes*.

3. Ogni segmento viene elaborato in parallelo, utilizzando una funzione di hash crittografico (ad esempio, SHA-256).

4. I risultati di ogni segmento vengono combinati per produrre un'unica chiave di output.

Funzione di compressione G e funzione di round P di Blake2b

La funzione di compressione **G** è costruita sulla funzione di round **P** di **Blake2b**. La funzione **P** opera su un input di 128 byte, che può essere visto come 8 registri di 16 byte:[1]

$$P(A_0, A_1, \ldots, A_7) = (B_0, B_1, \ldots, B_7).$$

La funzione di compressione **G(X, Y)** opera su due blocchi di 1024 byte, **X** e **Y**. Inizialmente, si calcola:

$$R = X \oplus Y.$$

Quindi, **R** è visto come una matrice 88 di registri di 16 byte. La funzione **P** viene applicata prima per riga e poi per colonna per ottenere **Z**:

$$(Q_0, Q_1, \ldots, Q_7) \leftarrow P(R_0, R_1, \ldots, R_7);$$

$$(Q_8, Q_9, \ldots, Q_{15}) \leftarrow P(R_8, R_9, \ldots, R_{15});$$

$$\ldots$$

$$(Q_{56}, Q_{57}, \ldots, Q_{63}) \leftarrow P(R_{56}, R_{57}, \ldots, R_{63});$$

$$(Z_0, Z_8, Z_{16}, \ldots, Z_{56}) \leftarrow P(Q_0, Q_8, Q_{16}, \ldots, Q_{56});$$

$$(Z_1, Z_9, Z_{17}, \ldots, Z_{57}) \leftarrow P(Q_1, Q_9, Q_{17}, \ldots, Q_{57});$$

$$\ldots$$

$$(Z_7, Z_{15}, Z_{23}, \ldots, Z_{63}) \leftarrow P(Q_7, Q_{15}, Q_{23}, \ldots, Q_{63}).$$

Infine, **G** produce l'output $\mathbf{Z} \oplus \mathbf{R}$:

$$G : (X, Y) \rightarrow R = X \oplus Y \rightarrow P \rightarrow Q \rightarrow P \rightarrow Z \rightarrow Z \oplus R.$$

Perché "Decompressione"?

Il termine "decompressione" si riferisce al processo attraverso il quale un'informazione sintetizzata o combinata (come nel caso di $R = X \oplus Y$) viene espansa in una rappresentazione più articolata attraverso vari passaggi di trasformazione. In particolare, la funzione P applicata row-wise e column-wise agisce in modo da rimescolare e ridistribuire l'informazione contenuta in R, aumentando così la sua entropia e rendendo l'output meno riconoscibile rispetto agli input originali.

Questo processo di "decompressione" assicura che, sebbene il risultato finale (output di G) sia complesso e diffuso, esso conserva una relazione matematica precisa con i dati di input.

4.3 utilizzo delle due Master Key

Insomma, tutta questa fatica per avere due chiavi fortemente sicure, ma a cosa servono?
Possiamo dire che siamo a questo punto dello schema crittografico di Bitwarden:

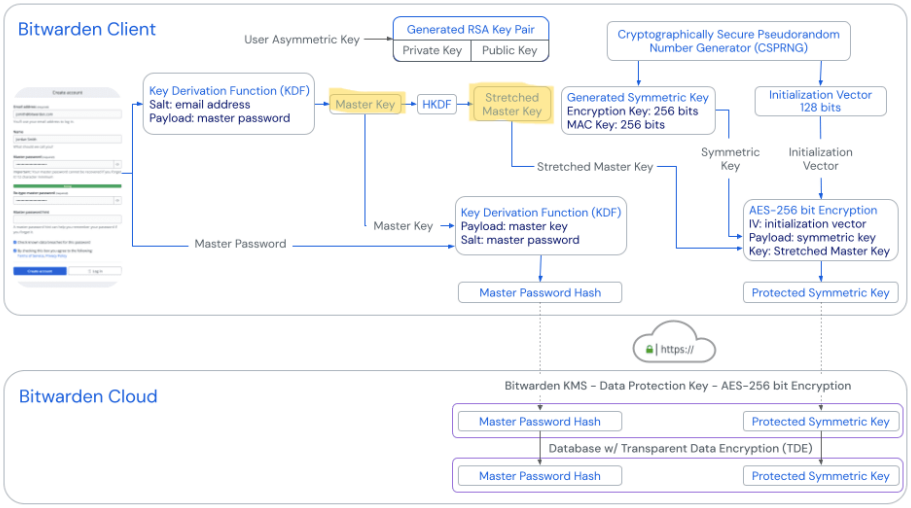


Figura 4.2: fasi di generazione delle chiavi

La Master Password quindi viene solo utilizzata per generare una complessa "danza" crittografica. In questo modo non viene mai salvata in chiaro se non nell'istante in cui il client la inserisce.

Ora vediamo come Bitwarden ha deciso come utilizzare queste due chiavi per poter riconoscere l'utente garantendone la massima sicurezza.

4.3.1 Master Password Hash

Viene generata utilizzando un algoritmo di derivazione della chiave, abbiamo già visto come funziona PBKDF2, una volta fatto il primo accesso puoi decidere quale algoritmo utilizzare fra quelli precedentemente presentati

4.3.2 Protected Symmetric Key

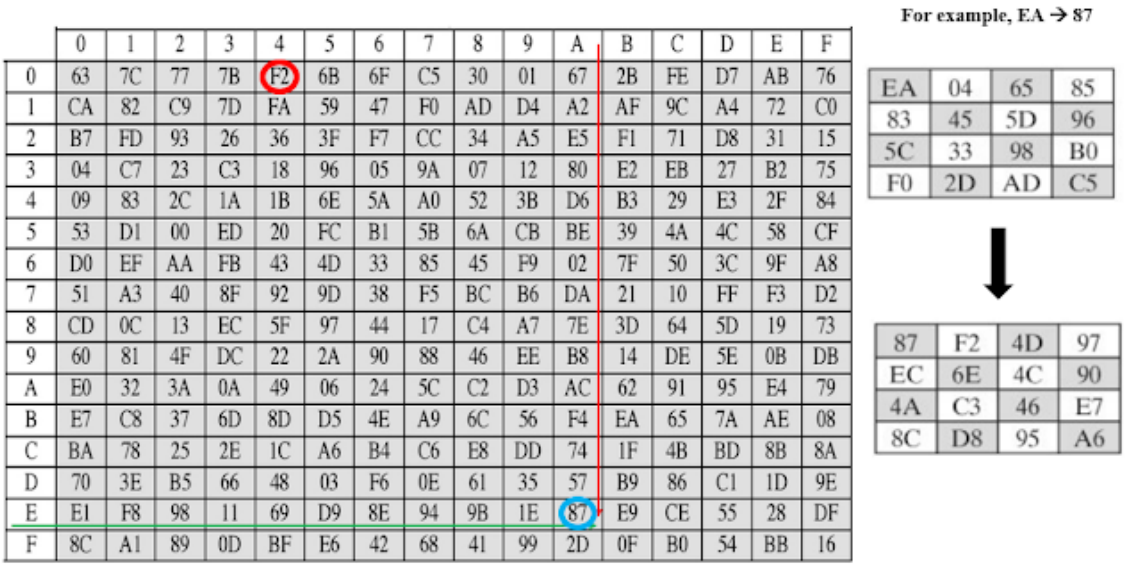
generata con un algoritmo AES-256 che ora spieghiamo nel dettaglio:
L'AES-256 (Advanced Encryption Standard con una chiave di 256 bit)
è un algoritmo di cifratura a blocchi che utilizza una chiave simmetrica per cifrare e decifrare i dati.

4.3.3 Come applica Confusine AES-256?

SubBytes e S-Box

- **S-Box (Substitution Box)**: L'S-Box è una tabella di sostituzione che mappa ogni byte del blocco di stato a un altro byte. Questa mappatura non è lineare, il che significa che piccoli cambiamenti nel testo in chiaro o nella chiave possono causare cambiamenti significativi nel testo cifrato. L'S-Box di AES è progettata per resistere ad attacchi di crittoanalisi differenziale e lineare.

- **SubBytes**: In questa operazione, ogni byte del blocco di stato (una matrice $4 \cdot 4$ di byte) viene sostituito con il byte corrispondente nell'S-Box. Questa operazione introduce confusione, in quanto la sostituzione non segue un modello prevedibile e aumenta la complessità dell'analisi del testo cifrato.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 4.3: spostamento sulla matrice $16 \cdot 15$ di S-Box, nelle varianti con meno bit di AES si utilizzano matrici più piccole

4.3.4 come applica Diffusione?

ShiftRows

- **ShiftRows** è un'operazione in cui i byte nelle righe della matrice di stato sono ciclicamente spostati a sinistra. La prima riga rimane inalterata, la seconda riga è spostata di una posizione, la terza di due posizioni e la quarta di tre posizioni. Questo spostamento crea una diffusione tra i byte nelle diverse colonne della matrice.

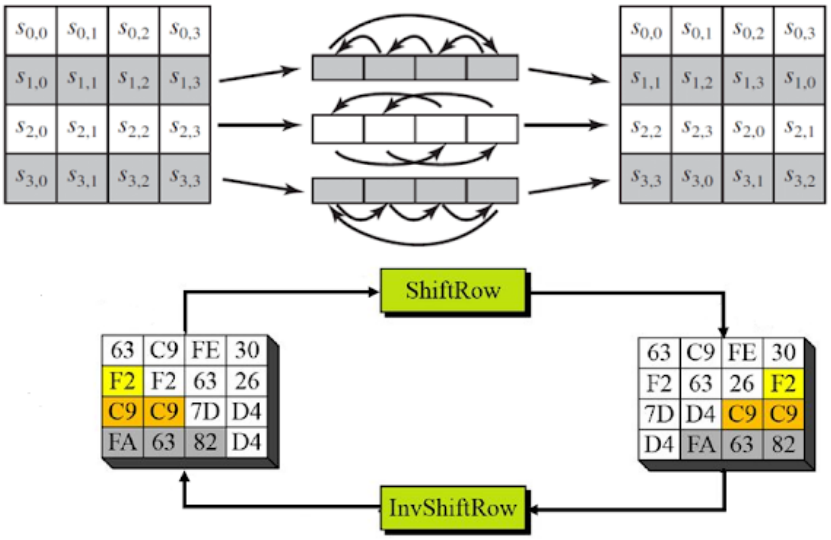


Figura 4.4: rappresentazione matriciale di SubBytes

MixColumns

- è un'operazione che applica una trasformazione lineare ai byte di ogni colonna della matrice di stato. Ogni colonna è trattata come un vettore di quattro byte e viene moltiplicata, in un campo di Galois (matematica finita), per una matrice fissa, che è invertibile. Questa operazione mescola i byte all'interno di ciascuna colonna, assicurando che i cambiamenti in un singolo byte si diffondano a tutti gli altri byte della colonna.

AddRoundKey

Anche se l'operazione di **AddRoundKey** non contribuisce direttamente alla confusione e alla diffusione, è essenziale per la sicurezza dell'algoritmo. In questa fase, una chiave di round derivata dalla chiave principale viene aggiunta al blocco di stato tramite l'operazione di XOR. Ogni round ha una chiave diversa, derivata attraverso un processo noto come "expansion key". Questa aggiunta introduce un nuovo livello di sicurezza, rendendo ogni round unico.

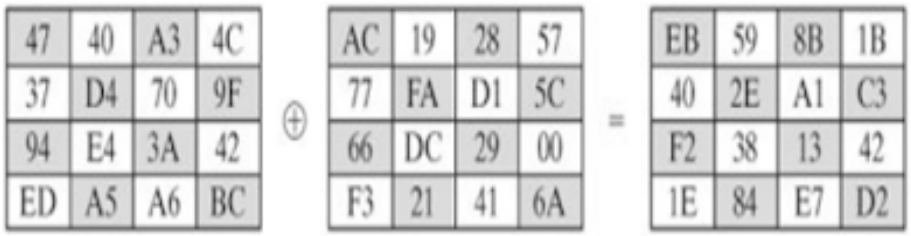


Figura 4.5: rappresentazione matriciale di round key

Capitolo 5

accesso di Bob

Contesto: Bob è un utente di Bitwarden e vuole accedere al suo account utilizzando la sua *email* e la sua *master password*.

Passaggi dell'accesso di Bob

1. **Inserimento delle Credenziali**:
   - *Email*: Bob inserisce il suo indirizzo email, ad esempio, `bob@example.com`.
   - *Master Password*: Bob inserisce la sua master password.
2. **Derivazione della Master Key**:
   - Localmente sul dispositivo di Bob, la master password viene combinata con la sua email (`bob@example.com`) come sale in un processo di derivazione della chiave utilizzando **PBKDF2** con un numero elevato di iterazioni (ad esempio, 600.000). Questo produce una chiave di 256 bit chiamata **Master Key**.
3. **Creazione dell'Hash della Master Password**:
   - La Master Key viene usata per generare un hash della master password (MPH). Questo hash è derivato usando **PBKDF-SHA256** con un payload della Master Key e un `salt` derivato dalla master password. Questo hash viene inviato ai server di Bitwarden per l'autenticazione.
4. **Autenticazione sul Server**:
   - I server di Bitwarden confrontano l'hash inviato con l'hash memorizzato corrispondente. Se gli hash corrispondono, l'autenticazione è considerata valida.

5. **Decriptazione della Protected Symmetric Key**:
    - Sul dispositivo di Bob, la **Stretched Master Key** viene derivata dalla Master Key utilizzando l'**HKDF** (HMAC-based Extract-and-Expand Key Derivation Function).
    - La Stretched Master Key viene quindi utilizzata per decriptare la **Protected Symmetric Key**, che è memorizzata sul server e criptata con la Stretched Master Key. La Protected Symmetric Key decriptata fornisce accesso alla **User Symmetric Key**, che è la chiave usata per decriptare i dati del vault.
6. **Accesso ai Dati**:
    - Con la User Symmetric Key, Bob può ora accedere ai dati criptati nel suo vault. Questo processo di decriptazione avviene interamente sul suo dispositivo.

Controesempio: Accesso Fallito

Immaginiamo che Bob, accidentalmente, inserisca una master password errata o un'email errata:

1. **Inserimento delle Credenziali Errate**:
    - Bob inserisce una master password o un'email sbagliata.
2. **Derivazione della Master Key Errata**:
    - L'errore nella master password o nell'email porta alla derivazione di una **Master Key errata**.
3. **Creazione di un Hash della Master Password Errato**:
    - La Master Key errata produce un hash della master password che non corrisponde a quello memorizzato sui server di Bitwarden.
4. **Autenticazione Fallita**:
    - Gli hash non corrispondono, quindi l'autenticazione fallisce. Bob non può accedere al suo account perché il server non autorizza l'accesso.
5. **Mancata Decriptazione**:

- Poiché l'autenticazione fallisce, Bob non riceve la **Protected Symmetric Key** corretta, e quindi non può decriptare i dati del suo vault. Anche se avesse ricevuto una chiave protetta, la Stretched Master Key errata non sarebbe in grado di decriptarla correttamente.

Questo processo garantisce che solo chi conosce la master password corretta possa accedere ai dati criptati, proteggendo l'utente da accessi non autorizzati.

Capitolo 6

Conclusioni

Personalmente, Dopo questa ricerca mi sento di dire che Bitwarden è un software molto sicuro e affidabile per la gestione delle password. Mi piace pensare che la crittografia sia utilizzata per scopi positivi e Bitwarden è la migliore offerta open source per la gestione delle proprie password.
Ho apprezzato talmente tanto la complessità e la capacità degli ingegneri nel saper architettare un meccanismo crittografico e talmente fine da non aver paura di ricervere attacchi nel server cloud, poichè anche se venissero rubati i dati, sarebbero inutilizzabili senza la master password.
Per questi motivi e per lo studio fornito sulla complessità crittografica che ho deciso di aprire un accounti Bitwarden per gestire, da qui in poi, i miei account personali

Bibliografia

[1] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: the memory-hard function for password hashing and other applications. https://www.password-hashing.net/argon2-specs.pdf.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] H. Krawczyk,P. Eronen. "hmac-based extract-and-expand key derivation function (hkdf)". Accessed: 3/08/2024 15:32.

[8] 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.

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