

Autenticazione Utente: PAP, CHAP

1 Introduzione all'Autenticazione Utente

L'obiettivo primario dell'autenticazione utente (*User Authentication*) è la verifica di una pretesa di identità. In termini formali, il sistema deve ottenere la garanzia che un soggetto sia effettivamente chi dichiara di essere prima di concedere l'accesso a una risorsa.

1.1 Distinzione Terminologica: Identificazione vs. Autenticazione

È fondamentale non confondere due fasi distinte del processo di controllo degli accessi, evidenziando la differenza tra il “mostrare” (*show*) e il “provare” (*prove*):

Identificazione (L'asserzione)

È l'atto di presentare la propria “identità digitale” al sistema.

- **Azione:** L'utente *mostra* un identificativo pubblico o noto al sistema (es. Indirizzo email, User ID, Codice Fiscale).
- **Natura:** È una dichiarazione (“Io sono l'utente X”), ma da sola non garantisce la veridicità dell'affermazione.

Autenticazione (La verifica)

È l'atto di convalidare l'identità digitale presentata.

- **Azione:** L'utente deve *provare* in modo inconfondibile che l'identità digitale dichiarata è sotto il suo legittimo controllo.
- **Meccanismo:** Avviene solitamente tramite la dimostrazione della conoscenza di un segreto (es. una password) che solo il legittimo proprietario dovrebbe possedere.

1.2 La Natura dell'Identità e la Continuità del Soggetto

Esiste una precisazione critica riguardante la relazione tra l'utente fisico e l'account digitale:

- **Disaccoppiamento dall'Identità Reale:** Non sussiste un requisito tecnico mandatorio che imponga il collegamento tra l'identificativo digitale e l'identità civile (*real-life identity*) dell'utente.
- **Il Principio di Continuità:** Il criterio di successo per un'autenticazione non è necessariamente identificare la persona fisica, ma stabilire una continuità temporale.

L'autenticazione ha successo quando il sistema è ragionevolmente certo che il soggetto che sta accedendo al servizio in questo momento è la **medesima entità** che vi ha acceduto in passato.

1.3 Definizione Normativa (Standard NIST SP 800-63-3)

Per formalizzare il concetto, si fa riferimento al *National Institute of Standards and Technology* (NIST). L'autenticazione utente digitale è definita come:

“Il processo di **stabilire fiducia** (*establishing confidence*) nelle identità utente che vengono presentate elettronicamente a un sistema informativo.”

L'uso del termine “stabilire fiducia” implica che l'autenticazione non è un semplice controllo binario, ma un processo di gestione del rischio volto a raggiungere un livello di sicurezza (*assurance*) adeguato.

2 Categorie di Autenticazione

L'autenticazione si basa sulla verifica di uno o più fattori distinti, classificati in base alla natura della ”prova” fornita dall'utente. La letteratura tecnica (e lo standard NIST) identifica quattro categorie principali, oltre a fattori contestuali emergenti.

2.1 Fattore di Conoscenza (Something you know)

Questo è il metodo più tradizionale e diffuso. Si basa su un segreto condiviso tra l'utente e il sistema.

- **Elementi:** Password, PIN, chiavi crittografiche segrete, risposte a domande di sicurezza pre-concordate.
- **Principio di Sicurezza:** L'efficacia risiede nel fatto che l'informazione sia nota *esclusivamente* all'utente (“and only you know”). Se il segreto viene condiviso o intercettato, l'autenticazione è compromessa.

2.2 Fattore di Possesso (Something you have)

L'autenticazione dipende dal possesso fisico di un oggetto specifico.

- **Elementi:** Smart card, token fisici (es. generatori OTP), dispositivi mobili.
- **Hardware Uniqueness e PUF:** Un'area di particolare interesse citata nella slide è l'autenticazione basata sull'unicità dell'hardware, specificamente le **PUF (Physically Unclonable Functions)**.

Le PUF sfruttano le variazioni microscopiche e involontarie introdotte durante il processo di fabbricazione dei semiconduttori per creare un'impronta digitale hardware unica e impossibile da clonare, rendendo il dispositivo stesso la prova di autenticità.

2.3 Fattore di Inerenza Statica (Something you are)

Riguarda le caratteristiche biologiche e fisiche dell'individuo. Si parla di biometria *statica* perché analizza tratti fisiologici che non cambiano (o cambiano molto lentamente) nel tempo.

- **Elementi:** Impronta digitale, scansione della retina o dell'iride, riconoscimento facciale.
- **Criticità (Privacy ed Etica):** La slide solleva interrogativi importanti sulla privacy. A differenza di una password, un dato biometrico non può essere ”cambiato” se viene compromesso. Inoltre, si fa riferimento a scenari futuristici (citando il film *Gattaca*) riguardo l'uso del DNA come mezzo di autenticazione estremo.

2.4 Fattore di Inerenza Dinamica (Something you do)

Nota anche come **biometria comportamentale** (*behavioral authentication*), questa categoria analizza i pattern di comportamento unici dell'utente.

- **Elementi:**

- *Riconoscimento vocale*: Tono e cadenza della voce.
- *Grafia*: Pressione e tratto della firma.
- *Keystroke Dynamics*: Ritmo e velocità di digitazione sulla tastiera.
- *Interazione uomo-macchina*: La slide evidenzia persino i **movimenti del mouse** come parametro distintivo per identificare un utente.

Altri Fattori Contestuali Esistono mezzi più difficili da classificare nelle categorie standard ("harder to classify"), come l'**autenticazione basata sulla posizione** (*location-based authentication*), che verifica dove si trova l'utente (es. GPS, indirizzo IP) o l'orario di accesso per rilevare anomalie.

3 Prova della Conoscenza di un Segreto

Dimostrare di conoscere una password non significa necessariamente mostrarla in chiaro. Esistono vari protocolli con livelli diversi di esposizione:

- **PAP (Password Authentication Protocol)**: la password viene inviata in chiaro al server. È il metodo più semplice, ma altamente insicuro in presenza di intercettazioni.
- **CHAP (Challenge Handshake Authentication Protocol)**: non si trasmette mai la password in chiaro. Il server invia una sfida (challenge), il client risponde con un valore calcolato in base alla password e alla sfida, tipicamente usando una funzione hash.
- **Zero-Knowledge Proof (ZKP)**: protocolli avanzati che permettono di dimostrare la conoscenza del segreto senza rivelarlo minimamente.

4 PAP: Vantaggi e Limitazioni

Il Password Authentication Protocol (PAP) rappresenta la forma più basile di autenticazione utente. Come illustrato nello schema di riferimento, il principio di funzionamento si basa su un modello a due vie (two-way handshake) che non prevede l'uso di crittografia per la protezione delle credenziali durante il transito.

4.1 Meccanismo di Funzionamento

Il protocollo opera secondo il paradigma “Dimostra di conoscere la password mostrandola”. Il flusso logico, visualizzato nell'esempio con l'utente *GB* e la password *pippo*, avviene come segue:

1. **Invio delle Credenziali:** Il client (User) invia al server (Authenticator) una coppia di dati contenente l'identificativo utente e la password:
$$\{\text{User_ID}, \text{Password}\} \rightarrow \{\text{GB}, \text{pippo}\}$$
2. **Verifica Lato Server:** L'Authenticator riceve la coppia e consulta il proprio *User Database*. Confronta la password ricevuta con quella memorizzata per l'utente corrispondente.
3. **Esito:** Se le stringhe coincidono, il server invia un messaggio di conferma (ACK) garantendo l'accesso (“OK, you're in!”); in caso contrario, la connessione viene terminata o rifiutata.

4.2 Analisi delle Vulnerabilità

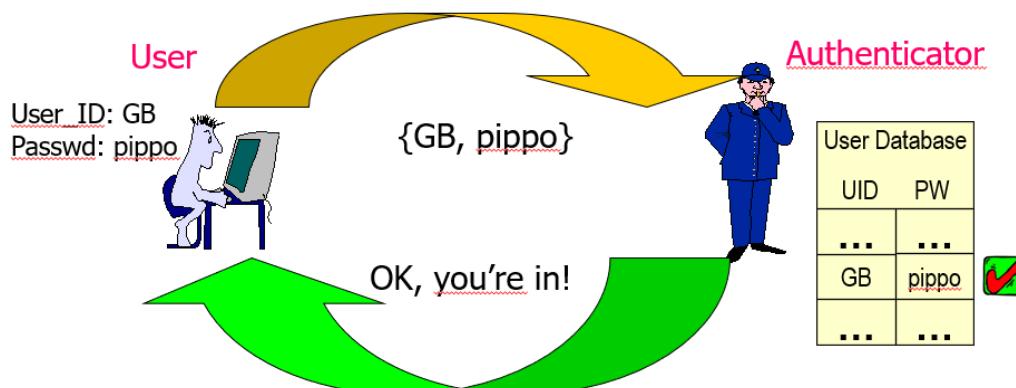
La semplicità del PAP introduce gravi rischi di sicurezza, rendendolo obsoleto per reti non fidate.

- **Trasmissione in Chiaro (Cleartext Transmission):** La vulnerabilità critica evidenziata è l'invio della password in chiaro (plain text). Chiunque sia in grado di intercettare il traffico di rete (tramite tecniche di *sniffing* su reti Wi-Fi pubbliche o hub condivisi) può leggere direttamente le credenziali senza doverle decifrare.
- **Suscettibilità ai Replay Attack:** Il protocollo è statico. Un attaccante che intercetta il pacchetto $\{\text{GB}, \text{pippo}\}$ può registrarla e rinviarla al server in un secondo momento. Poiché il PAP non utilizza *nonces* (numeri casuali usati una sola volta) o timestamp per garantire la freschezza della richiesta, il server accetterà la replica come autentica.

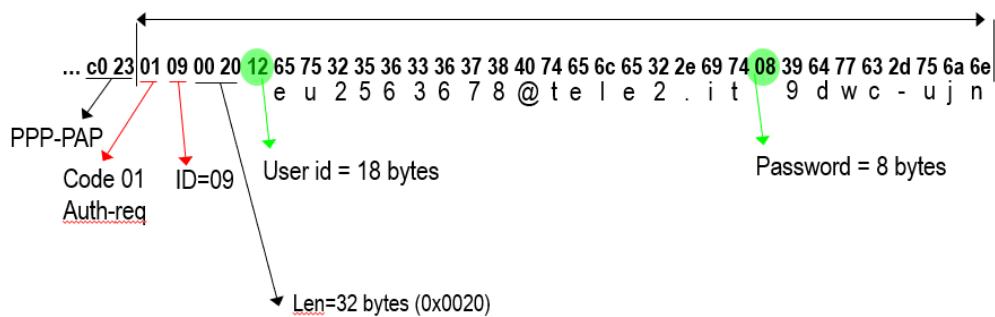
- **Mancanza di Autenticazione Mutua:** Il processo è unilaterale: l'utente dimostra la sua identità al server, ma il server non dimostra la propria all'utente. Questo espone l'utente al rischio di collegarsi a server malevoli (Rogue Access Points) che simulano il servizio legittimo per rubare le credenziali.
- **Attacchi Attivi:** Non essendoci protezione crittografica, un attaccante attivo (Man-in-the-Middle) può facilmente modificare i pacchetti o tentare attacchi di forza bruta direttamente sul canale di comunicazione, testando combinazioni di password fino a trovare quella corretta senza restrizioni crittografiche lato client.

4.3 Conclusioni

In sintesi, il PAP affida interamente la sicurezza alla segretezza del canale fisico. Se il canale non è sicuro, l'autenticazione è compromessa. Oggi è utilizzato solo in situazioni legacy o quando il tunnel di comunicazione è già cifrato a un livello inferiore (es. all'interno di un tunnel VPN IPSec già stabilito).



Esempio:



5 Challenge-Handshake Authentication Protocol (CHAP)

Il protocollo CHAP rappresenta un’evoluzione fondamentale rispetto al PAP, progettata per risolvere il problema della trasmissione della password in chiaro. Si basa sul paradigma *Challenge-Response* (Sfida-Risposta), che permette all’utente di provare la propria identità dimostrando la conoscenza di un segreto, senza mai inviarlo sulla rete.

5.1 Meccanismo di Funzionamento (3-Way Handshake)

Il processo, illustrato nello schema, si articola in tre fasi distinte che coinvolgono il *Remote User* (Client) e l’*Authenticator* (Server):

1. **La Sfida (Challenge):** Il Server inizia la procedura inviando al Client un valore casuale e imprevedibile, detto *nonce* (Number used ONCE).

$$\text{Server} \xrightarrow{\text{Challenge}} \text{Client}$$

2. **L’Elaborazione Sicura (Response):** Il Client non invia la password. Invece, utilizza una “funzione sicura” (solitamente una funzione di Hash crittografico come MD5 o SHA) per combinare la *Challenge* ricevuta con la propria *Password*.

$$\text{RES} = \text{SECURE_FUNCTION}(\text{Challenge}, \text{Password})$$

Il Client invia quindi al Server il proprio identificativo (UID) e il risultato calcolato (RES).

3. **La Verifica (Verification):** Il Server riceve l’UID, cerca la password corrispondente nel suo database e ripete localmente lo stesso identico calcolo effettuato dal Client per ottenere il risultato atteso (*Expected Response* o XRES):

$$\text{XRES} = \text{SECURE_FUNCTION}(\text{Challenge}, \text{Password}_{\text{stored}})$$

Infine, il Server confronta i due valori:

$$\text{RES} \stackrel{?}{=} \text{XRES}$$

Se coincidono, l’accesso è garantito (Success); altrimenti è negato (Failure).

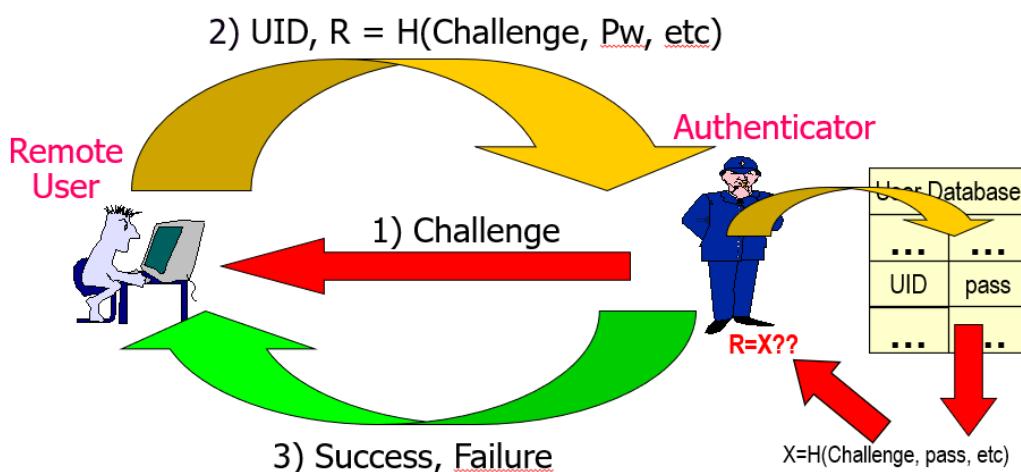
5.2 Dettagli Tecnici e Sicurezza

La slide evidenzia alcuni aspetti critici che differenziano il CHAP da altri protocolli:

- **Protezione dal Replay Attack:** Poiché la *Challenge* cambia ad ogni tentativo di autenticazione (è un numero casuale diverso ogni volta), intercettare la risposta (*RES*) passata non serve a nulla a un attaccante. La stessa password genera una risposta diversa per ogni sessione.
- **Natura della Funzione Sicura:** Sebbene l'hashing sia l'approccio più comune (es. $H(\text{Challenge} \parallel \text{Password})$), la slide specifica che **non è l'unico metodo**.

È possibile utilizzare anche algoritmi di cifratura (es. cifrare la challenge usando la password come chiave), ma questo richiede precauzioni maggiori.

- **Il Paradosso della Sicurezza Backend:** Un punto debole architettonale del CHAP risiede nel lato server. Affinché il server possa calcolare $XRES$, deve conoscere la password originale dell'utente (o una sua versione reversibile).
 - A differenza dei sistemi moderni che salvano solo l'hash della password (rendendo inutile il furto del database), un server CHAP deve conservare le password in chiaro (o cifrate in modo reversibile). Se il database del server viene compromesso, tutte le password degli utenti sono esposte.



6 Richiami di Teoria: Password vs. Chiave Segreta

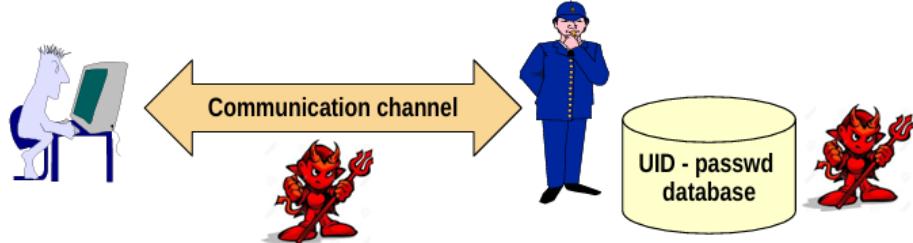
L'autenticazione è il processo attraverso il quale un'entità (utente) dimostra di conoscere un segreto (password o chiave) per verificare la propria identità. Sebbene concettualmente una *password* e una *chiave segreta* siano simili (entrambe segreti), nella pratica esiste una differenza sostanziale in termini di entropia:

- **Chiave segreta:** Solitamente una stringa casuale di N bit. La probabilità di indovinarla è 1 su 2^N .
- **Password:** Spesso è una stringa a **bassa entropia** scelta dall'uomo. La probabilità di indovinarla è molto più alta rispetto a 1 su 2^N (suscettibile ad attacchi a dizionario).

Ora ci chiediamo: chi è meglio tra CHAP e PAP? La risposta dipende interamente dal **modello di minaccia** (Threat Model) che si assume.

7 Modelli di Attacco

Per valutare la sicurezza di un protocollo di autenticazione, è necessario definire il modello di attacco, ovvero dove si posiziona l'avversario. Esistono due vettori principali:

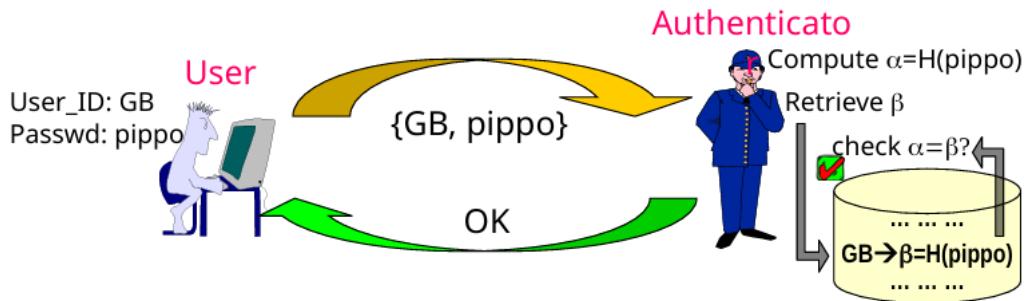


1. **Attacco al canale di comunicazione:** L'attaccante intercetta il traffico tra utente e server (Eavesdropping, Man-In-The-Middle, Replay Attack).
 - *Conseguenza:* Se si trasmettono password in chiaro (come nel protocollo PAP semplice), il canale **deve** essere protetto (es. tramite HTTPS/TLS).
2. **Attacco al backend (Database):** L'attaccante penetra nel server e ruba l'intero database contenente le coppie UserID-Password.

8 Password Authentication Protocol (PAP) e Hashing

Nel protocollo PAP, se ci concentriamo sulla difesa del backend, la memorizzazione delle password in chiaro è inaccettabile.

8.1 Memorizzazione tramite Hash



L'approccio standard prevede di non memorizzare la password P , bensì il suo hash $H(P)$.

$$\text{Database Entry} = \{\text{UserID}, H(P)\}$$

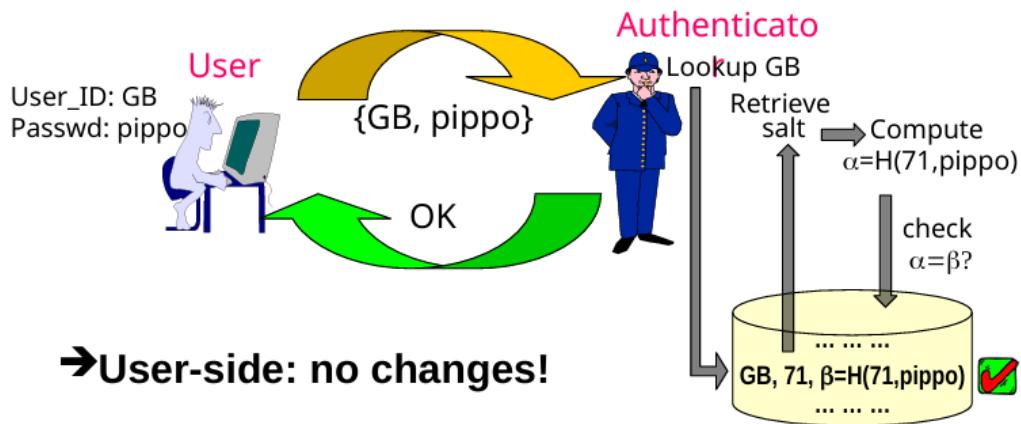
In fase di verifica, il server calcola l'hash della password ricevuta e lo confronta con quello memorizzato. Poiché H è una funzione *one-way* (monodirezionale), se l'attaccante ruba il DB, deve comunque investire risorse per invertire l'hash (brute-force o dizionario). La sicurezza dipende quindi dalla "forza" della password scelta dall'utente.

8.2 Limitazioni dell'Hash semplice

L'uso di un hash semplice (es. SHA256) presenta vulnerabilità:

- **Rainbow Tables:** È possibile pre-calcolare enormi tabelle di hash per password comuni.
- **Collisioni visibili:** Se due utenti (Giuseppe e Letizia) hanno la stessa password, avranno lo stesso hash nel DB. Un attaccante può dedurre questa informazione.

9 L'uso del Salt (Salting)



Per mitigare i problemi dell'hash semplice, si introduce il concetto di **Salt**. Il Salt è un valore casuale pubblico associato a ciascun utente.

$$\text{Hashed Password} = H(\text{Salt}, P)$$

Il database memorizzerà: {UserID, Salt, $H(\text{Salt}, P)$ }.

9.1 Vantaggi del Salt

1. **Difesa contro le Rainbow Tables:** L'attaccante dovrebbe generare una tabella diversa per ogni possibile valore di Salt (es. con un salt a 12 bit, servirebbero 2^{12} tabelle).
2. **Unicità dell'Hash:** Anche se due utenti usano la stessa password, avendo Salt diversi, i loro hash saranno completamente diversi. Questo nasconde la riutilizzazione delle password all'interno del DB o tra DB diversi.

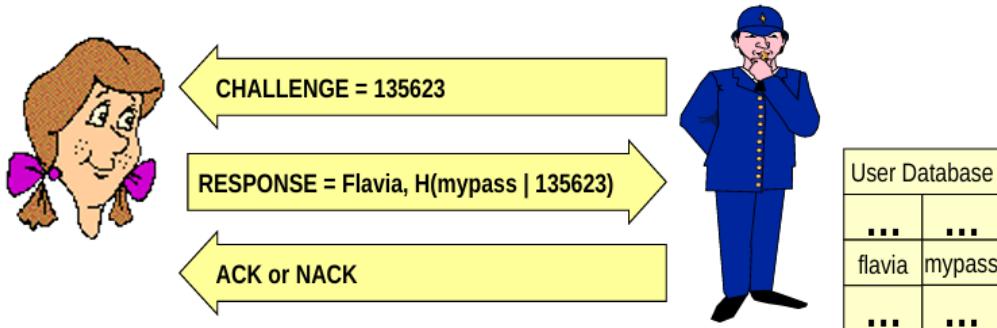
9.2 Flusso di Autenticazione con Salt

1. L'utente invia il proprio ID.
2. Il server recupera il Salt associato a quell'ID.
3. Il server calcola $\alpha = H(\text{Salt}, P_{\text{ricevuta}})$.
4. Il server confronta α con l'hash memorizzato β .

Nota: Lato utente non cambia nulla, il processo è trasparente.

10 Challenge-Handshake Authentication Protocol (CHAP)

CHAP è progettato per proteggere contro gli attacchi di replay e intercettazione sul canale, senza inviare la password in chiaro.



10.1 Funzionamento

1. Il server invia una **Challenge** (un numero casuale, es. 135623) all'utente.
2. L'utente calcola una **Response**: $R = H(\text{password} \mid \text{Challenge})$.
3. L'utente invia R al server.
4. Il server esegue lo stesso calcolo e verifica se il risultato coincide.

10.2 La vulnerabilità del Backend in CHAP

Per poter verificare la risposta dell'utente, il server deve essere in grado di calcolare $H(\text{password} \mid \text{Challenge})$. Ciò implica che il server deve conoscere la password in chiaro (o in una forma reversibile). **Conseguenza:** Se l'attaccante ruba il database di un server CHAP standard, ottiene le password in chiaro. In questo scenario, CHAP è meno sicuro di PAP con hashing.

11 Confronto Finale: Qual è l'algoritmo migliore?

Non esiste un algoritmo "migliore" in assoluto; la scelta dipende interamente dal **Threat Model** (Modello di Minaccia):

- **Se l'attaccante è un Eavesdropper (ascolta il canale):** CHAP è superiore, poiché la password non viaggia mai sulla rete.

- **Se l'attaccante punta al Backend (furto DB):** PAP (con Hash e Salt) è superiore, poiché il database non contiene password in chiaro.

12 Mitigazione in CHAP

È possibile migliorare CHAP per mitigare il rischio lato backend utilizzando un "Salt esplicito". Invece di memorizzare la password in chiaro, il server memorizza $H(\text{Salt}, P)$. Durante l'autenticazione:

$$\text{Response} = H(H(\text{Salt}, P) \mid \text{Challenge})$$

In questo modo, se il DB viene compromesso, l'attaccante ottiene solo degli hash "saltati" e non le password in chiaro, impedendo il riutilizzo immediato della password, sebbene rimanga la vulnerabilità agli attacchi a dizionario sulla singola entry.

13 La scelta della Funzione di Hash: Velocità vs Sicurezza

Una volta stabilito che la memorizzazione delle password richiede l'uso di funzioni di hash (eventualmente con salt), sorge il problema di quale algoritmo utilizzare. La scelta intuitiva di algoritmi crittografici standard moderni, come SHA256, si rivela in realtà pericolosa nel contesto specifico delle password.

13.1 Il paradosso di SHA256 e l'hardware specializzato

SHA256 è considerato uno standard sicuro per firme digitali e integrità dei dati. Tuttavia, presenta due caratteristiche che lo rendono inadatto per l'hashing delle password:

1. **Progettato per la velocità:** Gli algoritmi come SHA256 sono ottimizzati per essere calcolati rapidamente. In un attacco di brute-force, una computazione veloce favorisce l'attaccante, permettendogli di testare più candidati al secondo.
2. **L'impatto del Bitcoin Mining:** L'uso di SHA256 nel protocollo Bitcoin ha incentivato lo sviluppo di hardware specializzato a bassissimo costo e altissime prestazioni (ASIC - Application Specific Integrated Circuit).

Le prestazioni di cracking attuali evidenziano una disparità enorme tra hardware commodity e hardware specializzato:

- Una CPU standard (es. Core i7) calcola circa 66.6 MH/s (Milioni di hash al secondo).
- Un hardware di mining (es. Bitmain AntMiner S19j XP) raggiunge i 151 TH/s (Mila miliardi di hash al secondo).

Conseguenze pratiche: Con tale potenza di calcolo, una password di 8 caratteri casuali (charset base64) viene forzata in media in soli **0.93 secondi**. Una password di 8 caratteri che utilizza l'intera tastiera richiede circa 22 secondi.

13.2 La Soluzione: Slow Hashes e Bcrypt

Per contrastare l'aumento della potenza di calcolo, la difesa consiste nell'utilizzare funzioni di hash *intenzionalmente* lente. L'algoritmo di riferimento è ****Bcrypt****.

13.2.1 Caratteristiche di Bcrypt

Progettato nel 1999 per sistemi OpenBSD, Bcrypt nasce specificamente per l'hashing delle password. Le sue proprietà fondamentali sono:

- **Inclusione del Salt:** Gestisce nativamente il salting.
- **Fattore di Costo (Work Factor):** Introduce un parametro configurabile "cost" (x) che determina il numero di iterazioni necessarie per calcolare l'hash, secondo la formula:

$$\text{Iterazioni} = 2^x$$

Ad esempio:

- Costo = 10 → circa 1000 iterazioni ($2^{10} = 1024$).
- Costo = 13 → circa 10.000 iterazioni ($2^{13} = 8192$).

13.2.2 Vantaggi Prestazionali difensivi

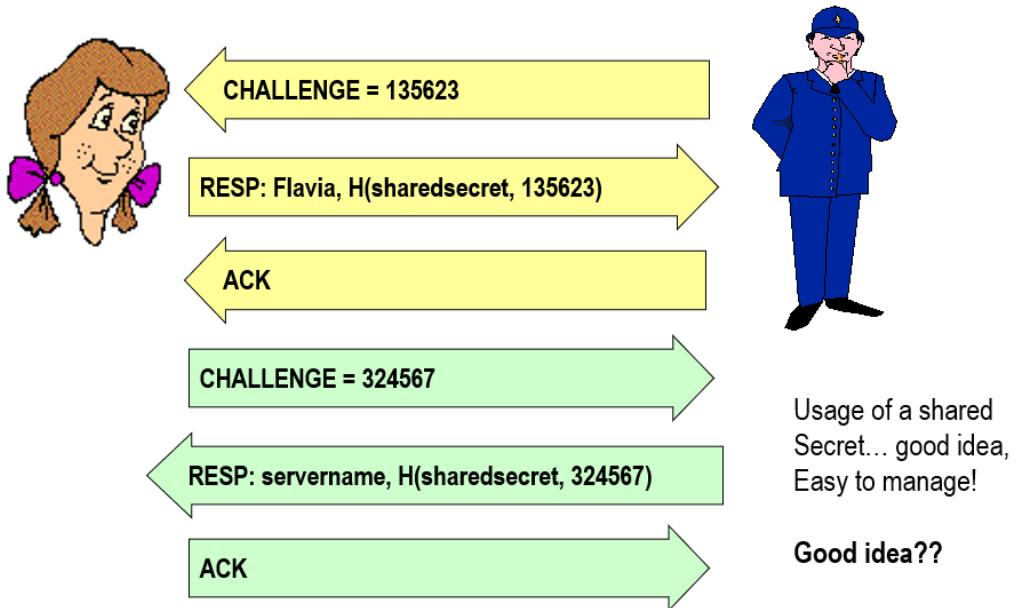
L'obiettivo di Bcrypt è rendere l'operazione di hashing computazionalmente onerosa ("expensive") per ogni singolo tentativo. Analizzando i benchmark su diversi dispositivi (dalle FPGA alle CPU moderne), il tasso di verifica scende drasticamente.

Mentre SHA256 permette 10^{14} tentativi al secondo su hardware specializzato, Bcrypt (con un costo adeguato) limita i tentativi a poche centinaia o migliaia al secondo (c/s), anche su hardware potente. Questo fattore di scala rende impraticabili gli attacchi di brute-force su larga scala.

14 Autenticazione Mutua con CHAP

In molti contesti operativi, non è sufficiente che il Server verifichi l'identità del Client; è altrettanto cruciale che il Client verifichi di comunicare con il Server legittimo (e non con un impostore o un *Rogue Access Point*). Questo processo prende il nome di **Autenticazione Mutua**.

14.1 Approccio Naive: Doppio CHAP



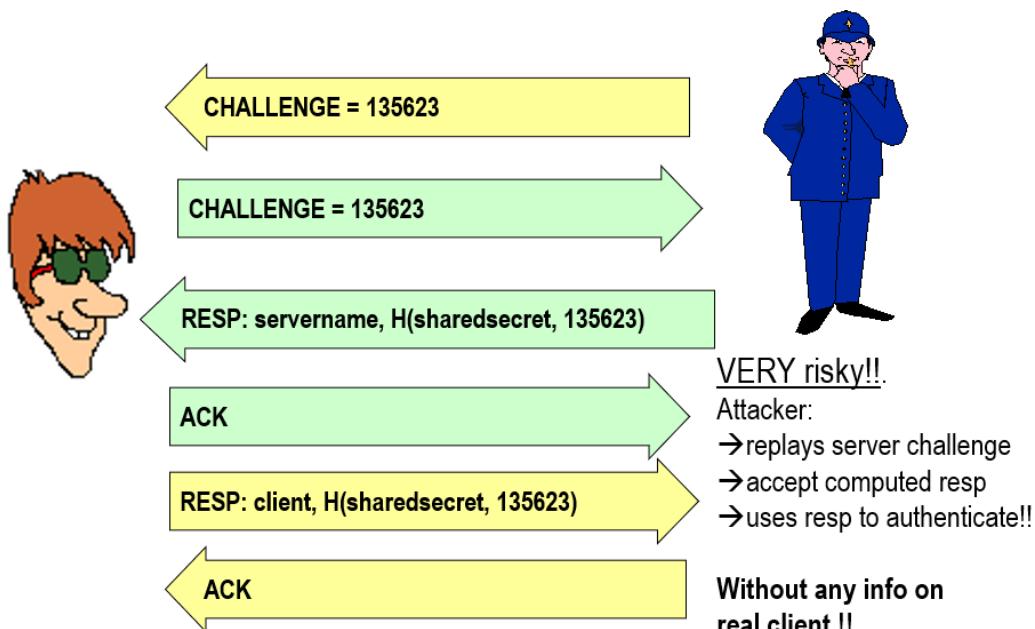
Come illustrato nello schema “CHAP and mutual authentication /1”, un approccio intuitivo ma rischioso consiste nel duplicare semplicemente il protocollo CHAP:

1. Il Server sfida il Client.
2. Il Client risponde e viene autenticato.
3. Successivamente, il Client sfida il Server (inviando una nuova *nonce*).

- Il Server calcola la risposta utilizzando lo stesso segreto condiviso e la invia al Client.

Sebbene sembri logico (“Usage of a shared Secret... good idea, Easy to manage!”), questa implementazione nasconde una vulnerabilità fatale se il protocollo non vincola strettamente l’ordine dei messaggi o non differenzia le chiavi per direzione.

14.2 Analisi del Reflection Attack



L’immagine sopra “Reflection attack” dimostra come un attaccante possa autenticarsi fraudolentemente al Server senza conoscere la password, sfruttando proprio il meccanismo di autenticazione mutua. L’attacco si basa sul principio di utilizzare il Server stesso come “oracolo” per risolvere la sfida.

L’attacco si svolge in tre fasi logiche (sfruttando il fatto che l’ordine dei messaggi in protocolli come PPP potrebbe non essere vincolante):

- Ricezione della Sfida (Sessione 1):** L’Attaccante si connette al Server. Il Server invia la prima sfida:

Server $\xrightarrow{\text{CHALLENGE}=135623}$ Attaccante

L’attaccante non conosce il segreto, quindi non può calcolare la risposta $H(\text{secret}, 135623)$.

2. **Riflessione della Sfida (Sessione 2 o Fase Mutua):** L'attaccante, invece di rispondere, apre una nuova sessione (o avvia la fase in cui è lui a dover sfidare il server) e invia al Server **la stessa identica sfida** appena ricevuta:

$$\text{Attaccante} \xrightarrow{\text{CHALLENGE}=135623} \text{Server}$$

3. **Ottenimento della Risposta:** Il Server, credendo di dover provare la propria identità, calcola onestamente l'hash corretto utilizzando il segreto che possiede e invia la risposta all'attaccante:

$$\text{Server} \xrightarrow{\text{RESP}=H(\text{secret},135623)} \text{Attaccante}$$

Ora l'attaccante possiede la risposta valida per la sfida originale.

4. **Chiusura dell'Attacco:** L'attaccante prende la risposta fornita dal Server e la invia indietro per chiudere la prima sessione:

$$\text{Attaccante} \xrightarrow{\text{RESP}=H(\text{secret},135623)} \text{Server}$$

Il Server verifica la risposta, la trova corretta e autentica l'attaccante.

14.3 Conclusioni

L'errore fondamentale è l'uso improprio delle primitive crittografiche:

Principale insegnamento: Non utilizzare algoritmi crittografici per uno scopo diverso da quello originale (anche se leggermente diverso) o con assunzioni differenti.

Per mitigare il *Reflection Attack* nell'autenticazione mutua, è necessario rompere la simmetria tra le due parti. Le soluzioni includono:

- **Chiavi diverse:** Utilizzare due segreti diversi per le due direzioni di autenticazione ($K_{\text{client} \rightarrow \text{server}} \neq K_{\text{server} \rightarrow \text{client}}$).
- **Format dei messaggi:** Includere l'identità del mittente o un identificativo di direzione all'interno dell'hash, in modo che $H(\text{sfida, secret, "Client"})$ sia matematicamente diverso da $H(\text{sfida, secret, "Server"})$.

15 Analisi dei Protocolli di Autenticazione Mutua e Vulnerabilità

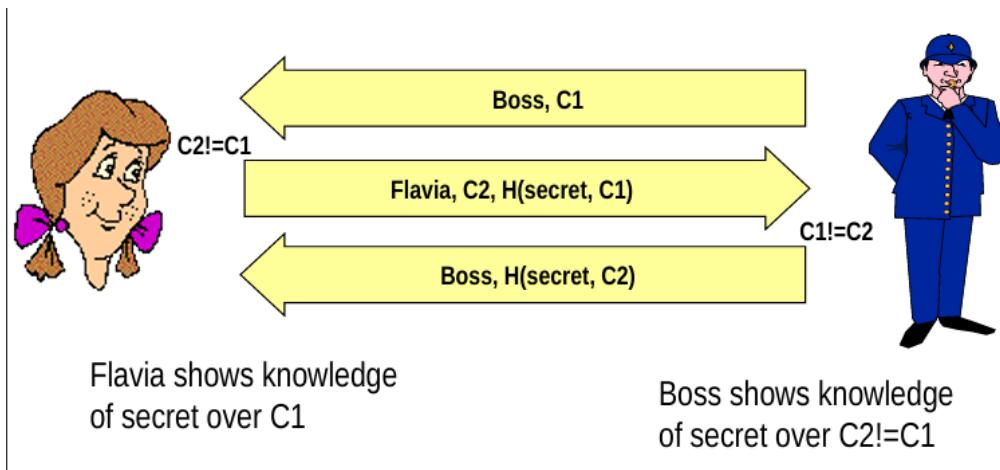
Questa sezione analizza l'evoluzione di un protocollo di autenticazione mutua basato su meccanismi *Challenge-Response*, evidenziando le criticità derivanti dall'unione ingenua di due protocolli di autenticazione unilaterale e le relative soluzioni.

15.1 Motivazione e Definizione del Problema

Spesso, nel tentativo di implementare un'autenticazione mutua (dove entrambe le parti, Initiator e Responder, devono provare la propria identità), si commette l'errore di combinare semplicemente due protocolli di autenticazione unilaterale. L'obiettivo è combinare le sfide in un unico scambio (handshake) per efficienza, ma se non progettato correttamente, questo approccio introduce gravi vulnerabilità, in particolare legate alla simmetria dei messaggi.

15.2 Il Protocollo Vulnerabile ("Pippo's Protocol")

Consideriamo un protocollo ipotetico (definito nelle slide come "Pippo's mutual authentication protocol") tra due entità: **Boss** (Initiator) e **Flavia** (Responder), che condividono un segreto S .



La sequenza dei messaggi è la seguente:

1. **Boss → Flavia**: Invia una challenge (nonce) C_1 .

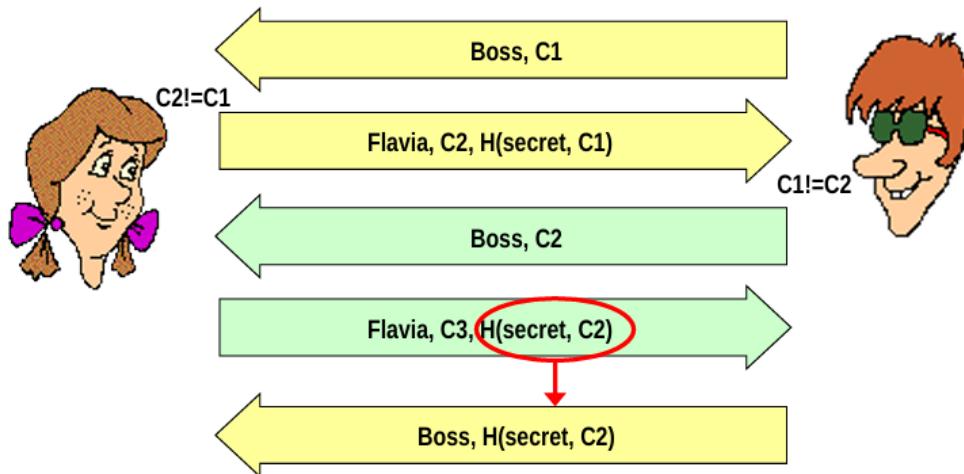
2. **Flavia → Boss:** Invia la propria challenge C_2 e la risposta alla prima challenge, calcolata come $H(S, C_1)$.
3. **Boss → Flavia:** Invia la risposta alla seconda challenge, calcolata come $H(S, C_2)$.

Nota: Sebbene vi sia una condizione $C_1 \neq C_2$, la struttura dei messaggi crittografici è identica in entrambe le direzioni: $H(S, \text{nonce})$. Questa simmetria è la radice del difetto di sicurezza.

15.3 Analisi delle Vulnerabilità

15.3.1 L'Attacco di Riflessione (Reflection Attack)

La simmetria del protocollo permette a un attaccante (Mallory) di impersonare il Boss senza conoscere il segreto S , sfruttando Flavia come "oracolo" per risolvere le sfide che lei stessa ha generato.



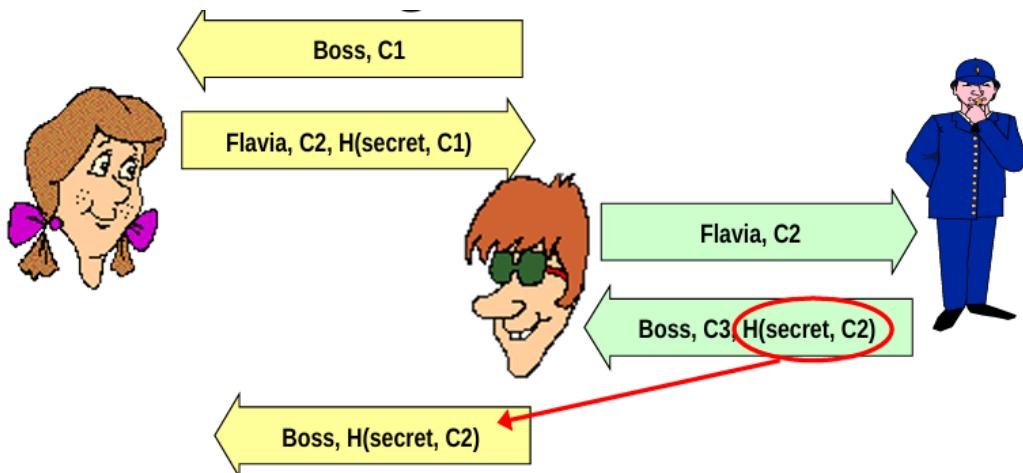
L'attacco procede come segue:

1. **Mallory (come Boss) → Flavia:** Invia C_1 .
2. **Flavia → Mallory:** Risponde con $C_2, H(S, C_1)$. Ora Mallory deve fornire $H(S, C_2)$ per completare l'autenticazione, ma non conosce S .
3. **Mallory (come Boss) → Flavia (Nuova Sessione):** Mallory apre una *seconda* sessione simultanea con Flavia, inviandole proprio C_2 come challenge (riflette la challenge di Flavia su se stessa).

4. **Flavia → Mallory**: Flavia, credendo sia una nuova sessione legittima, calcola e invia la risposta $H(S, C_2)$.
5. **Mallory → Flavia (Prima Sessione)**: Mallory prende $H(S, C_2)$ ottenuto dalla seconda sessione e lo usa per completare la prima sessione.

In questo modo, Flavia accetta Mallory come autenticato.

15.3.2 L'Attacco "Intertwining"



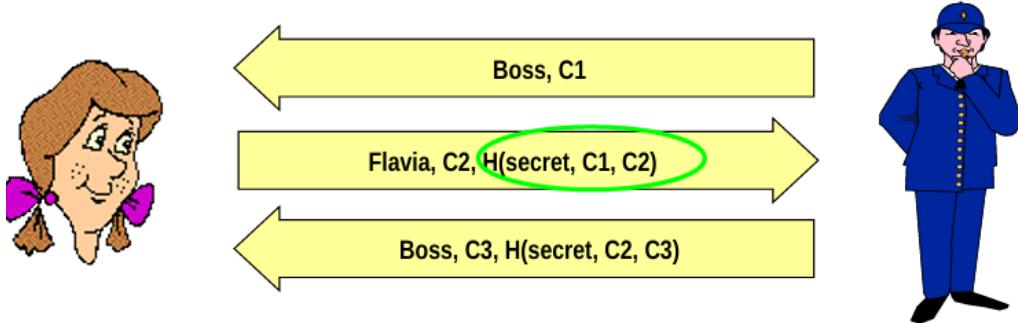
Anche se il protocollo avesse meccanismi per impedire la riflessione diretta (es. controllando lo stato del protocollo), l'attaccante può utilizzare un approccio *intertwining* (intrecciato). Se l'attaccante si pone come Man-in-the-Middle o impersona un'altra entità fidata, può:

1. Intercettare la challenge C_2 inviata da Flavia al Boss legittimo.
2. Aprire una sessione parallela con un'altra istanza di Flavia (o un altro server che condivide lo stesso segreto) inviando C_2 .
3. Ottenere la risposta valida $H(S, C_2)$ e inoltrarla al Boss originale.

Il problema di fondo rimane la non distinguibilità tra il ruolo dell'Initiator e quello del Responder nei messaggi crittografati.

15.4 Tentativi di Correzione e Soluzione Ottimale

15.4.1 Tentativo Fallito: Chaining delle Challenge



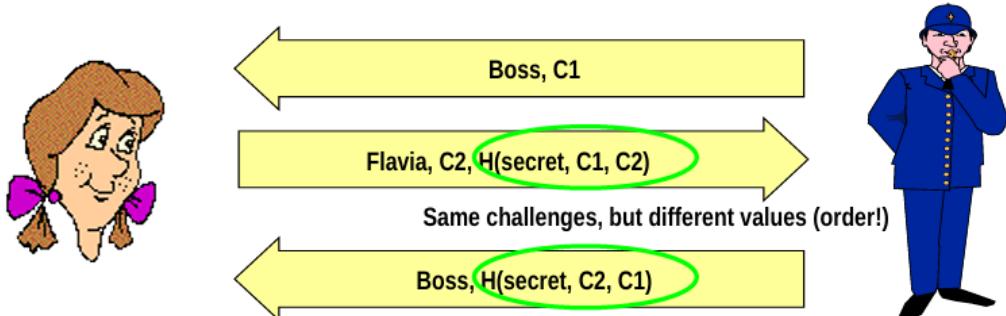
Chaining challenges! Add dependency
between challenges in same handshake

Un primo tentativo di correzione prevede il "chaining" (concatenazione) delle challenge per creare dipendenza tra i messaggi nello stesso handshake:

- Flavia invia $H(S, C_1, C_2)$.
- Boss risponde con $H(S, C_2, C_3)$.

Questo approccio, mostrato nelle slide come "Does not work", introduce complessità inutile (un terzo nonce C_3) e non risolve elegantemente il problema della simmetria se non gestito correttamente, portando a un eccesso di nonce ("Too many nonces!!").

15.4.2 Soluzione Ottimale: Minimizzazione dei Nonce e Ordine



No reflection possible anymore

La soluzione corretta ed efficiente consiste nel rompere la simmetria utilizzando l'**ordine** delle challenge all'interno della funzione crittografica, senza introdurre nuovi nonce.

Il protocollo sicuro diventa:

1. **Boss** → **Flavia**: C_1 .
2. **Flavia** → **Boss**: $C_2, H(S, C_1, C_2)$.
3. **Boss** → **Flavia**: $H(S, C_2, C_1)$.

Perché funziona?

- La risposta di Flavia include (C_1, C_2) nell'hash.
- La risposta del Boss include (C_2, C_1) nell'hash.
- Poiché $H(S, C_1, C_2) \neq H(S, C_2, C_1)$, i messaggi sono crittograficamente distinguibili.

Un attaccante non può più usare una sessione parallela per farsi firmare la risposta corretta, poiché Flavia, nel ruolo di Responder, genererà sempre hash nella forma $(C_{initiator}, C_{responder})$, mentre per chiudere la prima sessione l'attaccante avrebbe bisogno di un hash nella forma $(C_{responder}, C_{initiator})$, che solo il Boss (che conosce S) può generare legittimamente come conferma finale.