Il principio dell'autenticazione è che bisogna essere sicuri che l'utente è quello che dice di essere prima di poter usufruire di un servizio. Se non si è sicuri del fatto che la persona non è veramente ciò che dice di essere ci possono essere truffe e attacchi.

L'autenticazione avviene su 3 principi fondamentali:

- Qualcosa che l'utente **POSSIEDE** (cose fisiche o elettroniche, …).
- Qualcosa che l'utente **CONOSCE** (password, PIN, …).
- Qualcosa che l'utente **E'** (biometria, cioè misura di proprietà biologiche).

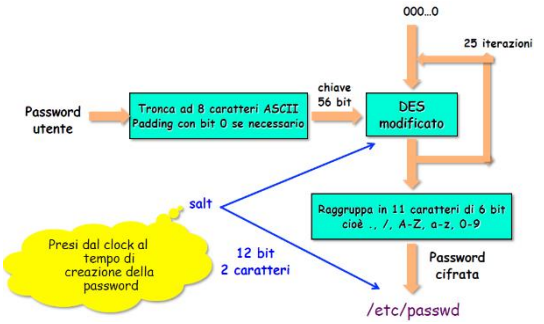
Il più diffuso sistema di autenticazione si basa sulle **password**. Assumiamo una situazione del genere:

Nella fase di registrazione in una piattaforma, Alice inserisce le proprie credenziali ad un database che salverà le informazioni riguardo ad Alice. Successivamente in fase di login, Alice inserirà i dati forniti al momento della registrazione e il database da l'accesso all'utente. Uno dei problemi è: il sistema deve memorizzare una rappresentazione della password. In che modo lo fa? Se si decidesse di memorizzare la password in chiaro in un file protetto si creerebbe il problema che l'attaccante può leggere il file e conoscere la password.

L'idea, allora, è che le password non devono essere memorizzate in chiaro, bensì in forma cifrata cioè che all'interno del database delle password è presente il testo cifrato della password di Alice. Nella fase di login quindi, Alice inserisce le sue credenziali e le invia al database delle password il quale dovrà usare un algoritmo di cifratura della password e verificare se il testo cifrato appena ottenuto e quello presente nel database coincidono.

Per applicare questa idea è necessario che durante la cifratura si utilizzi una chiave. Ma qual è questa chiave? È la password stessa. Per effettuare la cifratura, si può utilizzare una funzione di cifratura che è una variante del DES: evita che le stesse password abbiano la stessa cifratura in diversi sistemi ed è composto da 25 iterazioni. Con queste idee si aumenta la difesa per un attacco con dizionario. L'idea è la seguente: funzione di crypt()

La password dell'utente viene trasformata attraverso un processo "tronca ad 8 caratteri ASCII, padding con bit 0 se necessario". L'output di questo processo è una chiave a 56 bit che entra in input al DES modificato insieme ad una sequenza di 0. Questa operazione viene ripetuta per 25 volte (chiaramente ad una seconda iterazione la chiave rimane invariata, ma la sequenza di 0 sarà cambiata e cambierà ad ogni iterazione). L'output finale viene riscritto come "Raggruppa in 11 caratteri di 6 bit…" questo viene fatto perché l'output del DES modificato è una stringa binaria di caratteri (quindi possono esserci caratteri non leggibili), e mediante il processo descritto in precedenza, la password può essere scritta in un file /etc/passwd e tutti i caratteri diventano leggibili.



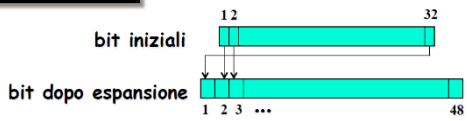
Il DES viene modificato nel seguente modo: c'è un valore casuale salt (12 bit, viene preso dal clock al tempo di creazione della password), a seconda di questi 12 bit si modifica il DES di volta in volta. Naturalmente questi valori casuali generali devono essere conosciuti e vengono quindi messi nel file /etc/passwd. Poiché il valore del salt è sempre casuale, per due password uguali è associato un cifrato differente. Questo è un aspetto importantissimo.

I 12 bit del salt vengono utilizzati in questo modo: sono associati a 12 coppie particolari (1,25), (2,26), (3,27) , … se bit salt=1 swap coppia corrispondente nei 48 bit output della tabella espansione E. Di conseguenza il DES cambia perché cambia la funzione di espansione E.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Espansione E
16 bit sono duplicati

12 bit salt
- associati a 12 coppie (1,25), (2,26), (3,27),…
- Se bit salt =1 swap coppia corrispondente nei 48 bit output della tabella espansione E



Anziché utilizzare una cifratura, si può pensare di memorizzare l'Hash della password di Alice all'interno del database. Come descritto in precedenza per la fase di registrazione e login, qui il ragionamento è analogo solo fatto con l'Hash.

Altra tecnica utilizzata è quella delle **Shadow Password**: all'interno del database la password cifrata non viene inserita immediatamente ma viene sostituita con una "x", che rappresenta un riferimento alla password cifrata nel file "passwd". Il file "shadow" contiene i cifrati ma è accessibile solo da root.

Gli attacchi che possono esserci alle password sono: spiare durante la digitazione, intercettazioni, tentare a caso o sistematicamente (se la password ha bassa entropia quindi password deboli o con attacchi con dizionario).

Se si pensasse ad un attacco di ricerca esaustiva la formula è la seguente $T= c^n * t * y$ dove c=numero di possibili caratteri, n= lunghezza delle password, t=numero di iterazione di Hash/cifratura, y= tempo richiesto per singola iterazione. Un esempio è posto al lato:

⇨ c / ⇩ n	26 (minuscole)	36 (minuscole alfanumerici)	62 (min. e maius. alfanumerici)	95 (caratteri tastiera)
5	0,67 ore	3,4 ore	51 ore	430 ore
6	17 ore	120 ore	130 giorni	4,7 anni
7	19 giorni	180 giorni	22 anni	442 anni
8	1,3 anni	18 anni	1385 anni	42.073 anni
9	34 anni	644 anni	85.852 anni	3.997.015 anni
10	895 anni	23.187 anni	5.322.801 anni	3.879.716.476 anni

L'attacco con dizionario si basa sulla possibilità che un utente utilizzi come password una parola di senso compiuto. Si tenta di "indovinare" una password utilizzando un file di parole (il dizionario). Il successo dipende dalla bontà del dizionario. Ci sono anche dei password crackers che usano dizionari ed un insieme di regole modificabili dall'utente (ad esempio parole duplicate o scritte all'inverso), richiede l'accesso al file delle passwd e il successo dipende dalla bontà del dizionario.

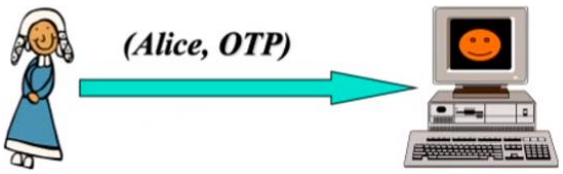
Per evitare cattive scelte come password, molte piattaforme obbligano l'utente a rispettare dei vincoli come lunghezza minima, caratteri non alfabetici etc…

Vulnerabilità delle password derivano da nomi comuni, parole comuni, specificità dell'utente. Le idee per scegliere le password sono di usare minuscole e maiuscole, usare numeri e lettere, effettuare sostituzioni sistematiche etc…

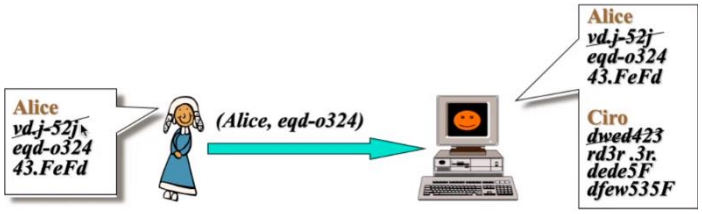
Altri sistemi impongono che il ciclo di vita delle password sia corto perché cambiare la password migliora la sicurezza. Esistono programmi che determinano la bontà della password.

Dopo un discorso informativo, adesso analizziamo il concetto di **One-time Password.**

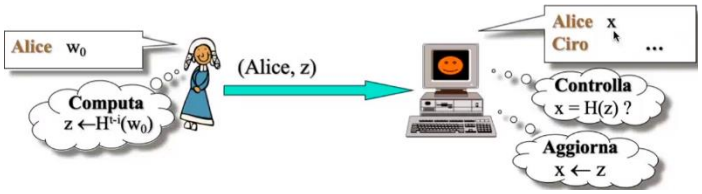
Il concetto è chiaro, ogni password è usata solo una volta. In un momento successivo di login la password è diversa. Vedremo quattro modi diversi di utilizzare la One-time Password.



La prima è la più semplice mediante utilizzo di una **lista condivisa** in cui ogni persona scrive in un blocco note tutte le sue password (le possiede anche il sistema ovviamente). Ogni volta che l'utente deve autenticarsi al sistema sceglie una password tra quelle segnate. Una volta utilizzata una determinata password questa non sarà più utilizzabile e verrà cancellata dalla lista condivisa. Un problema che può nascere è che se la lista termina bisogna trovare una nuova lista concordata col server. È una modalità valida ma non sicura poiché poco pratica.



La seconda possibilità è di creare un sistema di carattere informatico in cui l'utente conosce una sola password e da questa deriva tutte le successive. Questa idea è detta **Schema di Lamport**. Alice sceglie un valore iniziale w_0 e di questo valore ne calcola l'Hash per **i** volte (in pratica si fa l'Hash dell'Hash dell'Hash e così via… $H(H(…H(w_0)…))$). Il valore finale $x= H^i(w_0)$ lo possiede il sistema. Quando Alice deve autenticarsi la prima volta, deve inviare al sistema il valore **x**, questo può farlo ovviamente perché possiede w_0 e dunque esegue il calcolo per ottenere **x** come descritto sopra.

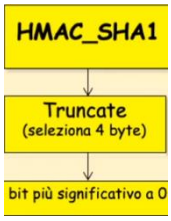


La seconda volta che Alice deve autenticarsi al Server non manderà nuovamente il valore **x**, bensì manderà il risultato dell'Hash questa volta iterato **i-1** volte. Il sistema per verificare che l'utente è corretto, riceve il valore di Alice che è l'Hash iterato **i-1** volte, lo itera un'altra volta e verifica se il valore **x** coincide con quello che possiede in partenza.

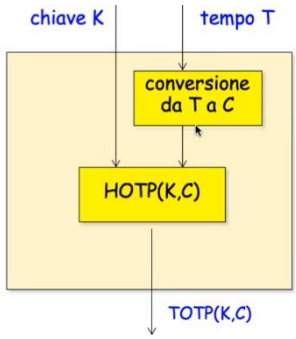
La terza volta che Alice deve autenticarsi al Server manderà ovviamente il risultato dell'Hash iterato **i-2** volte. Il sistema riceve questo valore, itera per due volte il calcolo dell'Hash, e verifica se coincide con **x**. Questo schema è sicuro perché la funzione Hash è una funzione non invertibile.

La terza possibilità, più utilizzata rispetto alle precedenti, si basa su **HMAC**. L'idea è che l'utente e il sistema conoscono entrambi una chiave segreta K. L'utente ogni volta che deve autenticarsi, utilizza un contatore C (8 byte) che indica "questa è la prima/seconda… volta che mi autentico". Utilizzando questo algoritmo HMAC(K,C) essenzialmente l'utente mette in input la chiave segreta e il contatore e questo algoritmo HMAC restituisce un **HOTP(K,C)** che è la One-Time Password.

In fase di login, Alice invia al sistema HOTP(K,C) e il sistema per essere sicuro che Alice le ha mandato un valore giusto gira lo stesso algoritmo usato da Alice e verifica se i due valori coincidono. L'idea dell'HMAC è la seguente: una volta calcolato l'HMAC c'è una fase di Truncate (permette di prendere il risultato binario di HMAC e di visualizzarlo in caratteri leggibili). Successivamente vengono presi i bit più significativi a 0.



Il quarto e ultimo algoritmo, più usato rispetto agli altri 3, è basato sul **tempo**. Di conseguenza in base al tempo in cui avviene l'autenticazione verrà restituito un TOTP(K,C) differente. L'idea è che Alice e il sistema possiedono solo la chiave K e a seconda del tempo T che viene convertito prima in un contatore C, viene eseguito l'HOTP(K,C) che genera la One-Time Password. Poiché Alice e il sistema posseggono le stesse informazioni la verifica sul valore finale può essere fatta agevolmente.



Il concetto di tempo deve essere approfondito in questo modo: se ad esempio un utente si autentica in un tempo T ad un sistema bancario internazionale, fino a che punto l'utente e il server Bancario hanno un tempo in comune? Ore, minuti, millisecondi? L'idea del tempo è che quindi non si deve essere precisi in maniera esagerata, ma bisogna invece avere un certo grado di approssimazione, cioè il tempo viene misurato ad intervalli. Un esempio di intervallo potrebbe essere un intervallo di 30 secondi, 1 minuto.

Per effettuare la conversione da tempo a contatore si esegue il seguente calcolo: C= (TEMPO CORRENTE – TEMPO INIZIO) / TIME STEP, dove Time step è di default 30 secondi. Alla fine si prende la parte intera del valore C calcolato. Si trova così un algoritmo che si basa sul tempo.

Abbiamo considerato un'idea in cui sia Alice che il Server hanno la stessa informazione T sul tempo. Può succedere che il sincronismo tra il tempo T di Alice e quello T' del server bancario siano differenti. Dunque $T \neq T'$ e di conseguenza la banca non permette l'autenticazione ad Alice. Come si risolve un problema di questo tipo? Il Server non deve controllare solamente l'intervallo di 30 secondi dove si trova, ma deve controllare anche i 30 secondi precedenti e i successivi. In pratica deve controllare diversi intervalli vicini a quello reale (concetto di **time windows**) in maniera tale che nonostante T e T' siano diversi, l'autenticazione avviene con successo.

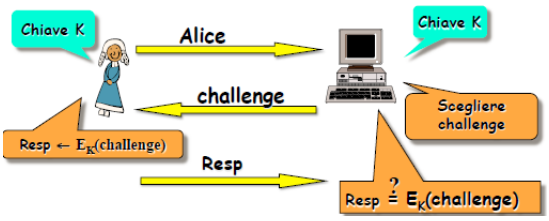
Questo concetto di time windows crea un possibile attacco: se l'attaccante legge l'informazione TOTP(K,T) che è in transito da Alice al Server potrebbe effettuare un attacco di replay cioè riutilizzare lo stesso TOTP(K,T) e dichiarare di essere Alice in un tempo successivo.

Per evitare un attacco di replay di questo tipo il Server accetta ogni password una sola volta, cioè se Alice manda un TOTP(K,T) il Server autentica la prima richiesta e rifiuta le successive. Il sistema quindi deve ricordare chi nel determinato momento di tempo T gli ha mandato la prima richiesta, lo memorizza e non accetta i successivi.

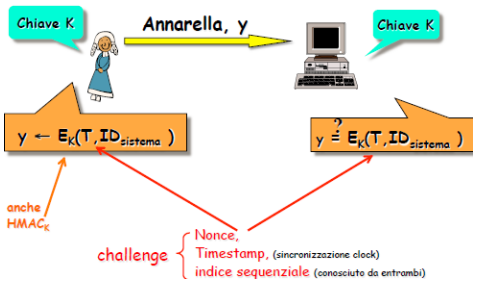
Questi sono i vari tipi di Token utilizzati che permettono di far girare l'algoritmo Time-Based OTP. Il risultato sono di solito 6 cifre decimali. L'utente prende queste 6 cifre decimali e le invia al sistema su cui si deve autenticare.

Anche l'arrivo su un SMS di un codice di cifre decimali è un tipo di Time-Based OTP.

Un altro sistema per l'autenticazione è basato sul concetto **Challenge – Response** in cui quando Alice si identifica al sistema, il sistema invia ad Alice una serie di domande segrete. Il protocollo si basa sul fatto che Alice e il Server posseggono una chiave privata K in comune. Quando Alice si autentica al sistema, naturalmente il sistema possedendo la stessa chiave privata K, manda alcune domande (Challenge) ad Alice, la quale ha la possibilità di rispondere correttamente poiché possiede la chiave K. Un altro utente che non possiede la chiave K chiaramente non avrà la possibilità di soddisfare la Challenge.



Si potrebbe pensare ad un attacco man in the middle in cui l'attaccante si mette al centro della comunicazione tra Alice e il sistema. L'attaccante rappresenta un server fasullo, che manda la Challenge ad Alice e Alice risponde mandando al server fasullo la sua chiave segreta K. Per evitare attacchi di questo tipo quando si manda un'informazione cifrata al server, non si invia solo il risultato della Challenge ma anche un ID di sistema ad esempio. L'attacco di prima in questo modo non funziona. Naturalmente quello dell'invio di un ID di sistema non è l'unica modalità possibile.



L'ultima possibilità di autenticazione che vedremo è quella basata sull'**Autenticazione a due fattori**: all'inizio di questo capitolo abbiamo detto che l'autenticazione si basa su 3 fattori (POSSIEDE, CONOSCE, E'). L'autenticazione a due fattori si basa sul fatto di soddisfare 2 fattori tra i 3.

Il tipo più vecchio di autenticazione a due fattori è l'Automated teller machine (ATM): per autenticarsi ad uno sportello bancomat, l'utente deve POSSEDERE la carta magnetica e deve CONOSCERE il Pin. Se ha i 2 fattori l'utente viene autenticato, altrimenti no. Con 2 fattori la sicurezza è maggiore.

Anche Google ha adottato l'autenticazione a due fattori, basata sulla password e sull'invio di un codice via sms.