# Sistemi di autenticazione

**Antonio Lioy**
**< lioy @ polito.it >**

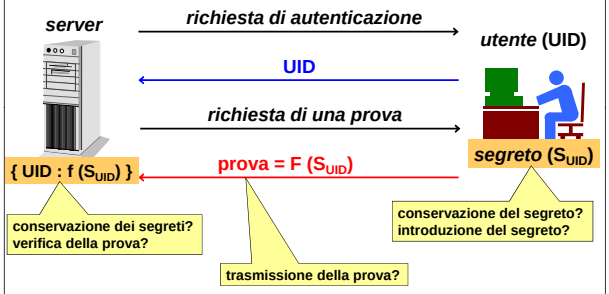
***Politecnico di Torino***
***Dip. Automatica e Informatica***

## Metodologie di autenticazione

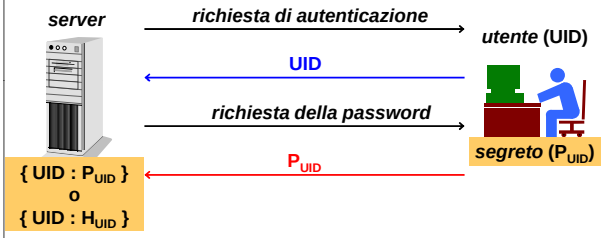
- **basate su meccanismi diversi ( 1/2/3-factors authentication ):**
  - qualcosa che so (es. una password)
  - qualcosa che posseggo (es. una carta magnetica)
  - qualcosa che sono (es. impronta digitale)


- **possibilità di combinare meccanismi diversi per passare da semplice autenticazione ad identificazione**

## Autenticazione degli utenti




## Password (ripetibili)




## Password ripetibili

- **segreto = la password dell'utente**
- **(client) creazione e trasmissione prova :**
  - F = I (la funzione identità)
  - ossia prova = password (in chiaro!)
- **(server) memorizzazione e validazione della prova**
  - caso #1: f = I (la funzione identità)
    - il server conserva le password in chiaro (!), $P_{UID}$
    - controllo d'accesso: prova = $P_{UID}$?
  - caso #2: f = one-way hash (ossia digest)
    - il server conserva i digest delle password, $H_{UID}$
    - controllo d'accesso: f(prova) = $H_{UID}$ ?

## Password ripetibili

- **vantaggi:**
  - semplice per l'utente
  - ... a condizione che ne abbia una sola
- **svantaggi:**
  - conservazione della password lato utente (post-it!)
  - password indovinabile (il nome di mio figlio!)
  - password leggibile durante la trasmissione
  - conservazione/validazione della password lato server – il server deve conoscere in chiaro la password o un suo digest non protetto (dictionary attack)

## Password

- **suggerimenti per renderle meno pericolose:**
  - caratteri alfabetici (maiuscoli + minuscoli) + cifre + caratteri speciali
  - lunghe (almeno 8 caratteri)
  - parole non presenti in dizionario
  - cambiate frequentemente (ma non troppo!)
  - non usarle :-)
- **uso di almeno una password (o PIN o codice di accesso o ...) inevitabile a meno di usare sistemi biometrici**

## Memorizzazione delle password

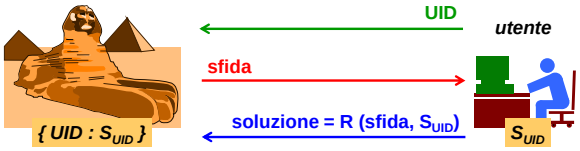
- **MAI in chiaro!**
- **password cifrata? il server deve però conoscere la chiave in chiaro …**
- **memorizzare un digest della password**
- **… ma si presta ad attacchi dizionario:**
  - sia HP l'hash di una password
  - for (each Word in Dictionary) do
    if ( hash(Word) = HP) then write ("trovata!", Word)
  - tecnica velocizzabile tramite "rainbow table"
- **occorre quindi introdurre una variazione imprevedibile chiamata "salt"**

## Uso del salt per memorizzare password

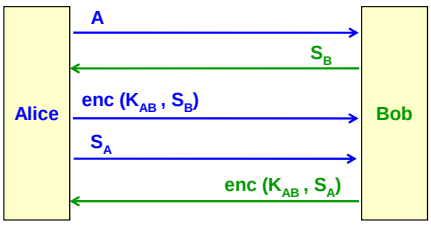
- **per ogni utente UID:**
  - scegliere / chiedere la pwd
  - generare un salt random (meglio se contiene caratteri poco usati o di controllo)
  - calcolare HP = hash ( pwd || salt )
- **memorizzare le triple { UID, HP, $salt_{UID}$ }**
- **evita di avere HP uguale per utenti diversi ma con stessa pwd**
- **rende praticamente impossibili gli attacchi dizionario (inclusi quelli basati su rainbow table)**

## Sistemi a sfida (simmetrici)

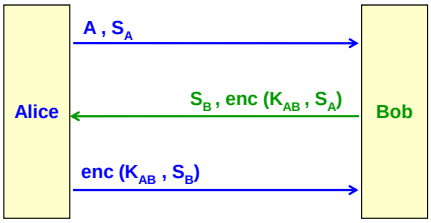
- **una sfida (tipicamente un numero random) viene inviato all'utente ...**
- **... che risponde con la soluzione effettuando un calcolo che coinvolge il segreto e la sfida**
- **il server deve conoscere in chiaro il segreto**
- **spesso la funzione R è una funzione di hash**




## Mutua autenticazione con protocolli a sfida simmetrici (v1)

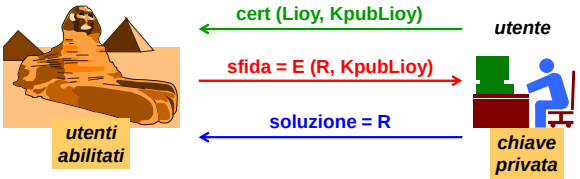



## Mutua autenticazione con protocolli a sfida simmetrici (v2)



## Sistemi a sfida (asimmetrici)

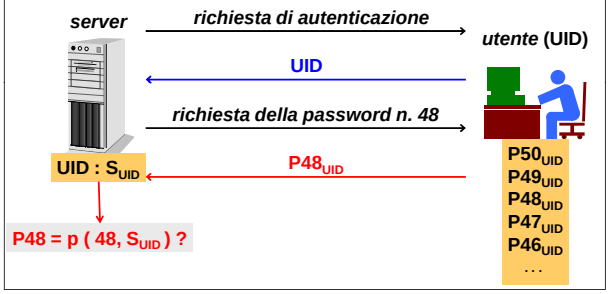
- **un numero random R viene cifrato con la chiave pubblica dell'autenticando ...**
- **... il quale risponde inviando R in chiaro grazie alla sua conoscenza della chiave privata**




## Pericoli delle sfide asimmetriche

- **fiducia nella CA che ha emesso il certificato**
- **verifica del name costraint sulle CA accettate**
- **possibilità di far deporre una firma RSA inconsapevolemente:**
  - se R=digest(documento) ...
  - e lo sfidante invia R in chiaro e lo richiede cifrato con la chiave privata ...
  - allora lo sfidato ha inconsapevolmente firmato il documento!!!

## Password (usa-e-getta)




## Password usa-e-getta

- **OTP (One-Time Password)**
- **idea originale:**
  - Bell Labs
  - sistema S/KEY
  - implementazione di pubblico dominio
- **implementazioni commerciali con generatori automatici hardware (autenticatore)**

## Come fornire password OTP agli utenti?

- **per uso su postazioni "stupide" o insicure:**
  - password pre-calcolate e scritte su un foglio
  - autenticatori hardware (criptocalcolatrici)
- **per uso su postazioni sicure e intelligenti:**
  - programmi di calcolo
  - eventualmente integrati col sw (telnet, ftp, ...) o hw (modem) di comunicazione

## Il sistema S/KEY (I)

- **RFC-1760**
- **l'utente genera un segreto S (il seme)**
- **l'utente calcola N one-time password:**
  - $P_1 = h(S)$
  - $P_2 = h(P_1) = h(h(S))$
  - . . .
- **l'utente inizializza il server di autenticazione con l'ultima password (es. $P_{100}$)**

## Il sistema S/KEY (II)

- **il server chiede le password in ordine inverso:**
  - P99? X
  - h (X) = P100 ?
  - memorizza X
- **in questo modo:**
  - il server non deve conoscere il segreto del client
  - solo il client conosce tutte le password
- **RFC-1760 usa MD4 (possibili altre scelte)**
- **implementazione public-domain per Unix, MS-DOS, Windows, MacOs**

## S/KEY – generazione della password

- **l'utente inserisce una pass phrase (PP) da almeno 8 caratteri**
- **alla PP viene concatenato un seme (seed) inviato dal server**
- **dal risultato viene estratto un hash MD4 su 64 bit (effettuando un EX-OR tra primo gruppo di 32 bit e terzo, e tra secondo e quarto gruppo)**

## S/KEY – password

- **64 bit di lunghezza della password sono un compromesso**
- **non troppo lunga né troppo insicura**
- **possibile inserirla come sequenza di 6 parole inglesi corte scelte da un dizionario di 2048 (es: "A", "ABE", "ACE", "ACT", "AD", "ADA")**
- **client e server devono avere lo stesso dizionario**

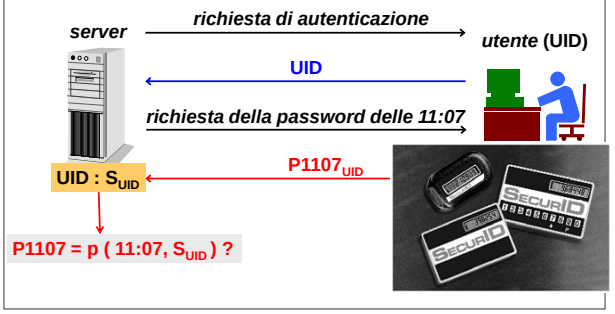
## Problemi delle OTP

- **scomode da usare in assoluto**
- **scomode da usare per accesso a servizi multipli basati su password (es. POP con check periodico della posta)**
- **costose se basate su autenticatori hardware**
- **non possono essere usate da un processo ma solo da un umano**

## Problemi degli autenticatori hardware

- **denial-of-service:**
  - tentativi falliti appositamente per negare l'accesso
- **social engineering:**
  - telefonata per denunciare smarrimento e chiedere l'inizializzazione di una nuova carta

## Password (usa-e-getta con autenticatore)



## RSA SecurID

- **inventato e brevettato da Security Dynamics**
- **meccanismo OTP di tipo sincrono:**

$$P_{UID}(t) = h(S_{UID}, t)$$

- **codice di accesso (*token-code*):**
  - 8 cifre decimali
  - casuale, non ripetibile
  - cambia ogni 60 s
- **deriva massima di 15 s / anno**
- **validità massima 4 anni**
- **basato su algoritmo di hash proprietario**

## SecurID: architettura

- **il client invia al server in chiaro *user*, PIN, *token-code* (seed, time)**
- **in base a *user* e PIN il server verifica contro tre possibili token-code:**
  **$TC_{-1}$, $TC_0$, $TC_{+1}$**
- ***duress code*: PIN che fà scattare un allarme (utile sotto minaccia)**
- **limite al numero di tentativi di autenticazione errati (default: 10)**
- **possibile avere tre diverse chiavi di accesso (per tre diversi sistemi)**

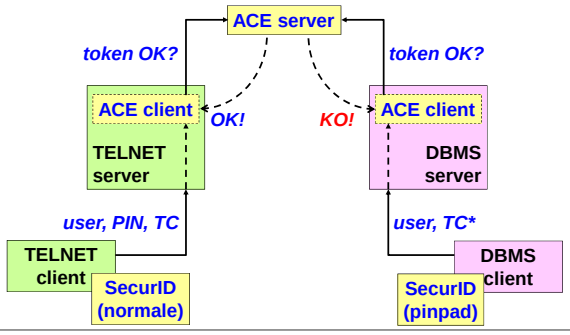
## SecurID: hardware

- **SecurID Card: carta classica**
- **SecurID PinPad: introduzione del PIN ed invio solo di *user* e *token-code**
- **SecurID Key Fob: formato portachiavi**
- **SecurID modem: modem PCMCIA-II V.34 contenente un token attivabile via sw tramite l'introduzione del PIN**


## SecurID: software

- **SoftID**
  - come un PinPad ma sw
  - trasmissione automatica o manuale del risultato
  - problema: sincronizzazione dei clock


## SecurID: architettura




## SecurID: client

- **ACE/client**
  - gestisce il dialogo con l'ACE/server
  - canale crittografato
  - sd_ftp per FTP sicuro
- **disponibile per:**
  - Unix
  - Win32
  - Netware
  - Macintosh
  - TACACS

## SecurID: server

- **ACE/server:**
  - autenticazione con SecurID
  - monitor, audit e report
  - interfaccia di gestione GUI
  - authentication API
  - interfaccia SQL per accesso ad un DBMS avente i dati degli utenti
  - vasto supporto commerciale sia per prodotti di sicurezza (es. firewall) sia di comunicazione (es. comm. server)
  - disponibile per Solaris, AIX, HP-UX, NT, 2000, XP

## CRYPTOCard

- **meccanismo a sfida**
- **usa DES-CBC**
- **unico prodotto: RB-1 card**
- **display LCD a 8 cifre (hex, dec)**
- **batterie sostituibili dall'utente (autonomia di 3-4 anni)**
- **per evitare di inserire la challenge, si può memorizzare l'ultima e calcolare automaticamente la prossima**
- **server per Unix e Windows (Radius, Tacacs+)**

## CRYPTOcard: hardware


## Autenticazione di esseri umani

- **come essere certi di stare interagendo con un essere umano e non con un programma (es. che invia una password memorizzata in un file)?**
- **due soluzioni:**
  - tecniche CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)
    - es. immagini di caratteri distorti
  - tecniche biometriche
    - es. impronte digitali

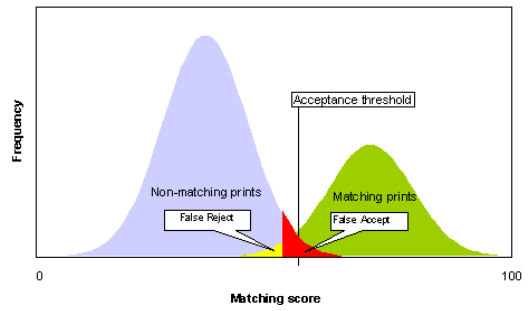
## Sistemi biometrici

- **misurano una caratteristica biologica dell'utente**
- **principali caratteristiche usate:**
  - impronte digitali
  - voce
  - scansione della retina
  - scansione della pupilla
- **utili per sostituire *localmente* un PIN o una password**

## Problemi dei sistemi biometrici

- **FAR = False Acceptance Rate**
- **FRR = False Rejection Rate**
- **FAR e FRR sono in parte aggiustabili ma dipendono moltissimo dal costo del dispositivo**
- **caratteristiche fisiche variabili:**
  - ferita su un dito
  - voce tremante per l'emozione
  - vasi oculari dilatati per alcool o droga

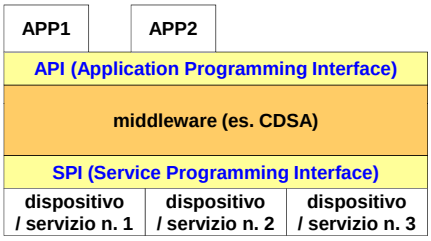
## FAR / FRR




## Problemi dei sistemi biometrici

- **accettazione psicologica:**
  - timore di essere schedati
  - paura di danneggiamento
- **mancanza di API / SPI standard:**
  - alti costi di sviluppo
  - dipendenza da un solo fornitore
- **in corso di sviluppo una API / SPI**
  - standard ed unitaria
  - basata su CDSA

## API? SPI? middleware!



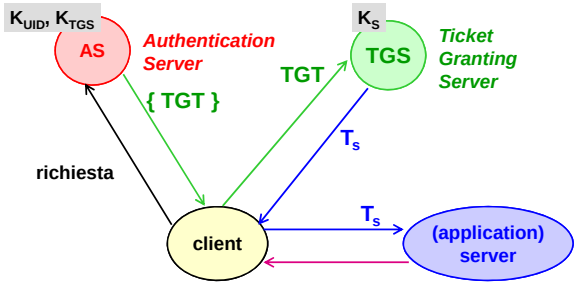

## Kerberos

- **sistema di autenticazione basato su una terza parte fidata (TTP = Trusted Third Party)**
- **sviluppato nel progetto Athena al MIT**
- **password mai trasmessa ma solo usata localmente come chiave di crittografia (simmetrica)**
- ***realm*** **= dominio di Kerberos, ossia insieme di sistemi che usano Kerberos come sistema di autenticazione**
- ***credenziale*** **= user.instance@realm**

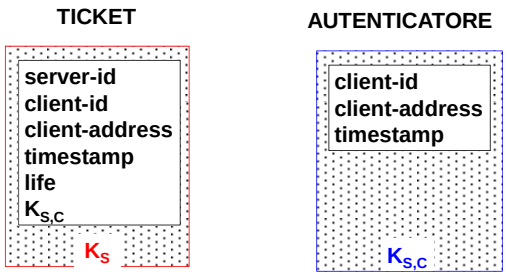
## Kerberos

- **ticket**
  - struttura dati che autentica un client nei confronti di un server
  - durata variabile
    (V4: max 21 ore = 5' x 255)
    (V5: illimitata)
  - crittografato con la chiave DES del server a cui è destinato
  - legato all'indirizzo IP del client
  - legato ad una sola credenziale
- **autenticazione semplice o mutua**

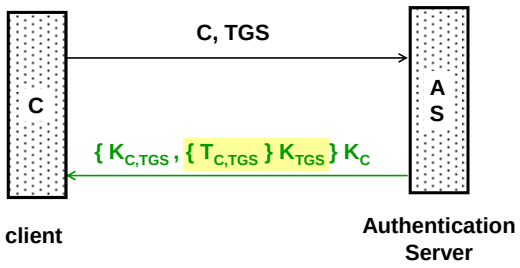
## Organizzazione di Kerberos



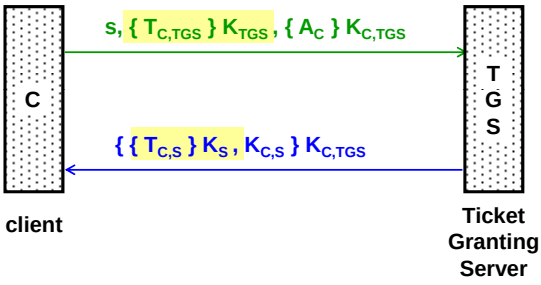
## Kerberos: formato dei dati (v4)



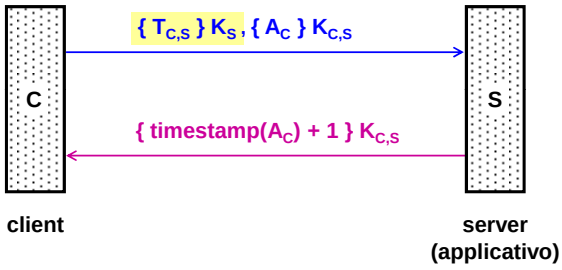

## Richiesta del TGT




## Richiesta del Ticket




## Uso del Ticket




## Versioni di Kerberos

- **MIT V4 (originale)**
- **MIT V5 (RFC-1510)**
  - non solo DES
  - durata maggiorata (inizio-fine)
  - inter-realm authentication
  - forwardable ticket
  - ticket estesi
- **OSF-DCE**
  - basata su MIT V5
  - implementato come RPC invece che come protocollo a messaggi

## Problemi di Kerberos

- **richiede la sincronizzazione dei clock**
  - all'interno di una LAN è una cosa comunque utile
  - in WAN crea problemi
  - Kryptoknight (alias IBM NetSP) ha rimosso questa limitazione
- **accesso remoto richiede password in chiaro**
  - canale cifrato oppure integrazione con OTP, sfida, o chiave pubblica
  - modem in dial-up Kerberizzati

## Vantaggi di Kerberos

- **login unico per tutti i servizi kerberizzati**
  - K-POP, K-NFS, K-LPD
  - K-telnet, K-ftp
  - K-dbms
- **il meccanismo dei ticket è ideale per connessioni intermittenti**
  - computer portatili
  - ISDN, WiFi
- **crescente supporto commerciale (MS usa Kerberos* a partire da Windows-2000)**

## SSO (Single Sign-On)

- **fornire all'utente un'unica "credenziale" con cui autenticarsi per svolgere tutte le operazioni su qualunque sistema**
- **SSO fittizio:**
  - client per sincronizzazione/gestione automatica pwd (alias "password wallet")
  - specifico per alcune applicazioni
- **vero SSO:**
  - tecniche di autenticazione multiapplicazione (es. sistemi a sfida asimmetrici, Kerberos)
  - quasi sempre richiede modifica delle applicazioni

## Interoperabilità

- **OATH (www.openauthentication.org)**
- **interoperabilità dei sistemi basati su OTP, sfida simmetrica e asimmetrica**
- **in corso di definizione gli standard per il protocollo client-server e per il formato dati sul client (draft RFC):**
  - HOTP (HMAC OTP, RFC-4226)
  - challenge-response protocol
  - bulk provisioning
  - symmetric key container