UnitelmaSapienza
Università degli Studi di Roma

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

# Identification and Authentication 1

Prof. F. Parisi Presicce

# Why Authentication?

- Common policy requirement: restrict the behavior of a user
  To permit different users to do different things, we need a way to identify or distinguish between users
  - Identification mechanisms to indicate/provide identity
  - Authentication mechanisms to validate identity
- Authentication is a mutual process which may use different mechanisms (and therefore have different levels of assurance):
  - Users must prove their identity to the computer.
  - Computers must prove their identity to the user.
    (This also applies to processes and other computers … any subject wishing to interact)

# Identification & Authentication

- When logging on to a computer you enter
  - user name and
  - password
- The first step is called identification:
  - You announce who you are.
- The second step is called authentication;
  - You prove that you are who you claim to be.
- To distinguish this type of ‘authentication’ from other interpretations, we refer here to user authentication: the process of verifying a claimed user identity.
- Authentication by password is widely accepted and not too difficult to implement.

# Authentication System

(*A*, *C*, *F*, *L*, *S*)

- *A* set of authentication information used by entities to prove identity
- *C* complementary information stored on computer and used by system to **validate** authentication information
- *F* complementation functions $f : A \rightarrow C$ *to generate c=f(a)*
- *L* functions that prove identity *l* (*a*,*c*)= T/F
- *S* functions enabling entity to create or alter information in *A* or *C*

# (bad) Example

Password system, with passwords stored on line in clear text

- *A* set of strings over fixed alphabet to construct the passwords
- *C* = *A*
- *F* singleton set of identity function { *I* }
- *L* single equality test function { *eq* }
- *S* functions to set/change password

# User Authentication

- Common mechanisms for “proving” user identity
- where the user is
  - access to the keyboard or IP address
- what the user knows
  - passwords, personal information
- what the user possesses
  - a physical key, a ticket, a passport, a token, a smart card, a badge
- what the user is (biometrics)
  - fingerprints, voiceprint, signature dynamics
- … or some combination of these

# “Where You Are”

- Some operating systems grant access only if you log on from a certain terminal.
  - A system manager may only log on from an operator console but not from an arbitrary user terminal.
  - Users may be only allowed to log on from a workstation in their office.
- Decisions of this kind will be even more frequent in mobile and distributed computing.
- Global Positioning System (GPS) might be used to established the precise geographical location of a user during authentication.

# “Something you have”

- User presents a physical token to be authenticated (keys, cards or identity tags, smart cards).
- It needs
  - an object which may or may not be unique, but to which the access is limited to “authorized” users or other subjects
  - a way to present this object to the entity which requires the subject to provide proof
  - a way to determine if the object as presented is the one which was expected
- but
  - physical tokens can be lost or stolen.
  - anybody in possession of the token has same rights as legitimate owner.

# Smart Cards

- A portable device with a CPU, I/O ports, and some nonvolatile memory (currently few thousand bytes) that is accessible only through its CPU
- It can carry out the computations required (for example by public key algorithms) and transmit results directly to the host
- Since devices are subject to theft, some devices require a PIN (something you know)
- PIN used by the device to authenticate the user
- Some use biometrics data about the user instead of the PIN

# “Something you know”

- Sequence of characters (digits, letters, etc )
  - Generated randomly, by user, by computer with user input, …
- a word (password)
- an algorithm (challenge-response, one-time passwords, …)
- a phrase (pass-phrase)
- a picture (pass-picture?)
- a combination or sequence of the above

Authentication

- Allows an entity (a user or a system) to prove its identity to another entity
- Typically, the entity whose identity is verified *reveals knowledge* of some secret S to the verifier
- Strong Authentication: The entity reveals knowledge of S to the verifier *without* revealing S to the verifier

# How well does this work?

**Ideal Policy:** only a certain set of individuals are allowed into the system.

**Stated Policy:** only users having a valid password are allowed into the systems.

**Actual Policy:** permit users who

- Are issued a valid password (authenticator)
- Can obtain a valid authenticator
- Can bypass the authentication process

# To get a valid authenticator...

- **Social engineering**
- **Guessing**: most break-ins occur because of bad passwords.
  - Do not use Your name (first, last, account name), Spouse, SO, pet, children, ..., even with a single digit, Any word in any language, even with standard replacement (1=i, 0=o, ...)
- **Known/standard account** and password pairs
  Many systems have certain accounts set up with certain default passwords (either well known or easy to guess). UNIX provides the guest account, with password often GUEST! VAX/VMS used to come with FIELD/SERVICE.
- **Known algorithms** for assigning passwords
  - use some/part of SSN, birthday, name, student/employee id, account name, phone extension

# Problem: pswd Storage

- Store as cleartext
  - If password file compromised, *all* passwords revealed
- Encipher file
  - Need to have decipherment, encipherment keys in memory
  - Reduces to previous problem (where is the key?)
- Store one-way hash of password
  - If the file is read, attacker must still guess passwords or invert the hash (but where is the hash?)

# Example

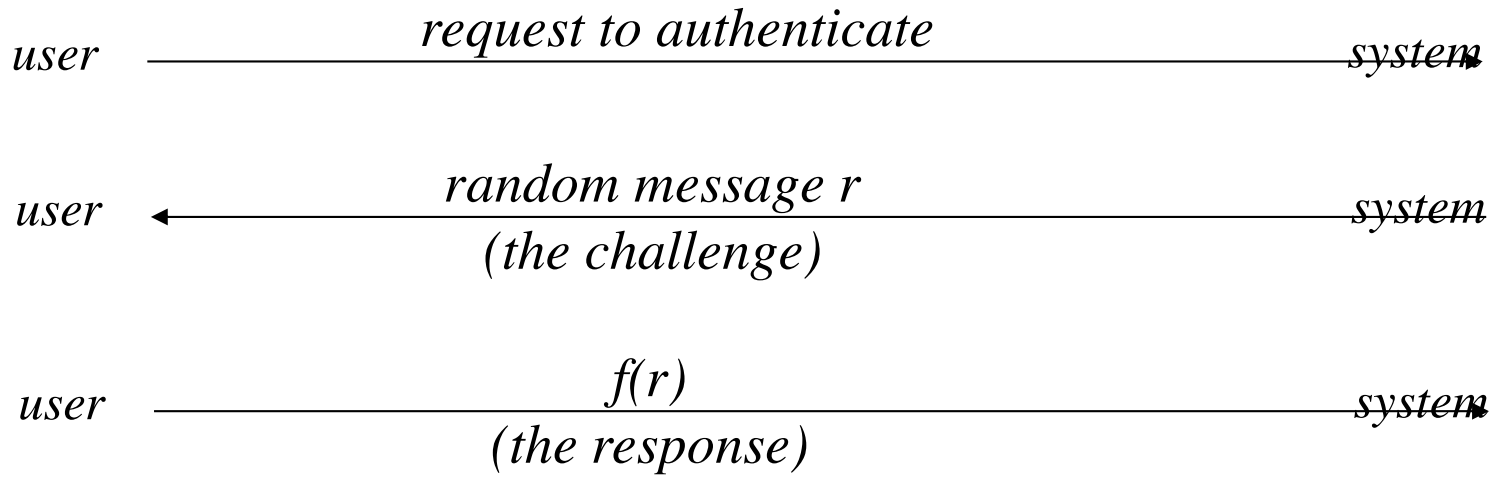
UNIX system standard hash function

- Hashes password into 11 printable char string using one of 4096 hash functions

- As authentication system:
  - *A* = { strings of 8 chars or less }
  - *C* = { 2 char hash id || 11 char hash }
  - *F* = { 4096 versions of modified DES }
  - *L* = { *login*, *su*, ... }
  - *S* = { *passwd*, *nispasswd*, *passwd+*, ... }

# Challenge-Response

User and system share a secret function f (in practice, f is a known function with unknown parameters, such as a cryptographic key)



# Pass Algorithms

Challenge-response with the function *f* itself a secret

- Example:
  - Challenge is a random string of characters such as “abcdefg”, “ageksido”
  - Response is some function of that string such as “bdf”, “gkio”
- Can alter algorithm based on ancillary information
  - Network connection is as above, dial-up might require “aceg”, “aesd”
- Usually used in conjunction with fixed, reusable password

# What is the advantage over passwords?

- Avoids “replay” attacks
- One-time password
  - authentication information *a* changes after each use
  - Why is this challenge-response?

- Attack
  - Attacker knows (space of) encryption function
  - Captures challenge and response
  - Learns encryption function / key
  - Can now properly respond to new challenge
- Solution: encrypt challenge
  - Use shared key to share session key
  - Session key encrypts challenge
  - Challenge thus indistinguishable from random data

# Dictionary Attacks

Trial-and-error from a list of potential passwords

- *Off-line*: attacker knows A, *f* and *c*'s, and repeatedly tries different guesses $g \in A$ until the list is done or passwords guessed
  - Examples: *crack*, *john-the-ripper*
- *On-line*: have access to functions in *L* and try guesses *g* until some *l* (*g*) succeeds
  - Examples: trying to log in by guessing a password

# Using Time to counter guessing

Anderson's formula:

- *P* probability of guessing a password in specified period of time
- *G* number of guesses tested in 1 time unit
- *T* number of time units
- *N* number of possible passwords (|*A*|)
- Then $P \geq TG/N$

# Example

- Goal
  - Passwords drawn from a 96-char alphabet
  - Can test $10^4$ guesses per second
  - Probability of a success to be 0.5 over a 365 day period
  - What is minimum password length?
- Solution
  - $N \geq TG/P = (365 \times 24 \times 60 \times 60) \times 10^4 / 0.5 = 6.31 \times 10^{11}$
  - Choose $s$ such that $\Sigma^s_{j=0}\, 96^j \geq N$
  - So $s \geq 6$, meaning passwords must be at least 6 chars long

# First UNIX Password Scheme

- [Wilkes68] (recall DES was 1976)
- Encryption based on M-209 cipher machine (US Army WWII)
- Easy to invert unknown plaintext and known key, used password as key:
  - Instead of $E_K$ (password) used hash function $E_{Password}$ (0)
- PDP-11 could check all 5 or less letter lower-case passwords in 4 hours!

# Making Brute Force Attacks Harder

- Use a slower encryption (hashing) algorithm
  - Switched to DES: $H(p) = \mathrm{DES}_p(0)$
- Even slower: run DES lots of times
  - UNIX uses $\mathrm{DES}_p{}^{25}(0)$
  - $\ldots \mathrm{DES}_p(\mathrm{DES}_p(\mathrm{DES}_p(\mathrm{DES}_p(0))))$
- Require longer passwords
  - DES key is only 56 bits: only uses first 7.5 characters (ASCII)
  - 95 printable characters, $95^8 = 6.6 * 10^{15}$

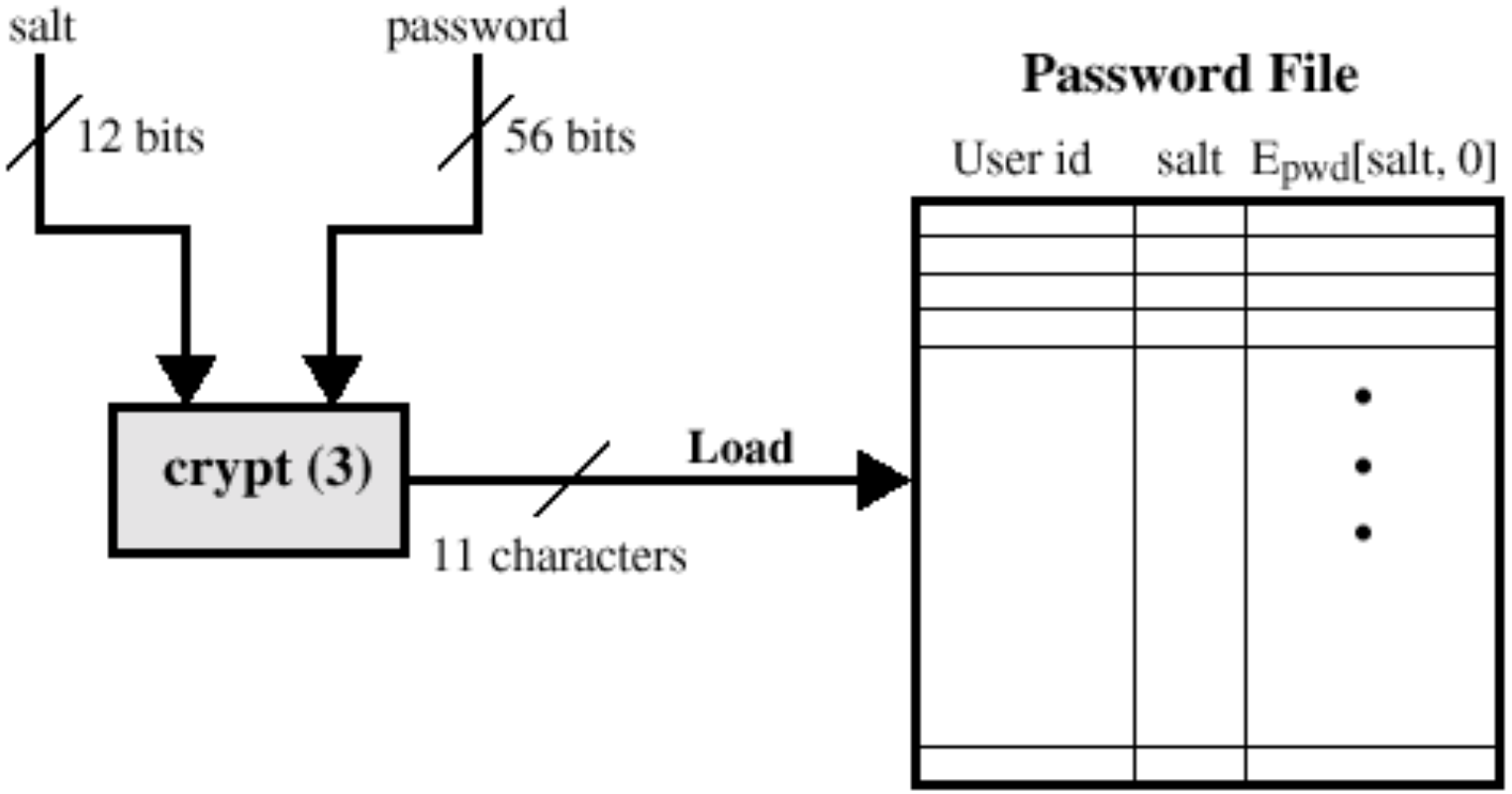
# UNIX Passwords

- UNIX passwords were kept in a publicly readable file, etc/passwords.
- Now they are often kept in a “shadow” directory and only visible by “root”.
- The salt serves three purposes:
  - Prevents duplicate passwords.
  - Effectively increases the length of the password.
  - Prevents the use of hardware implementations of DES

# UNIX Password salt

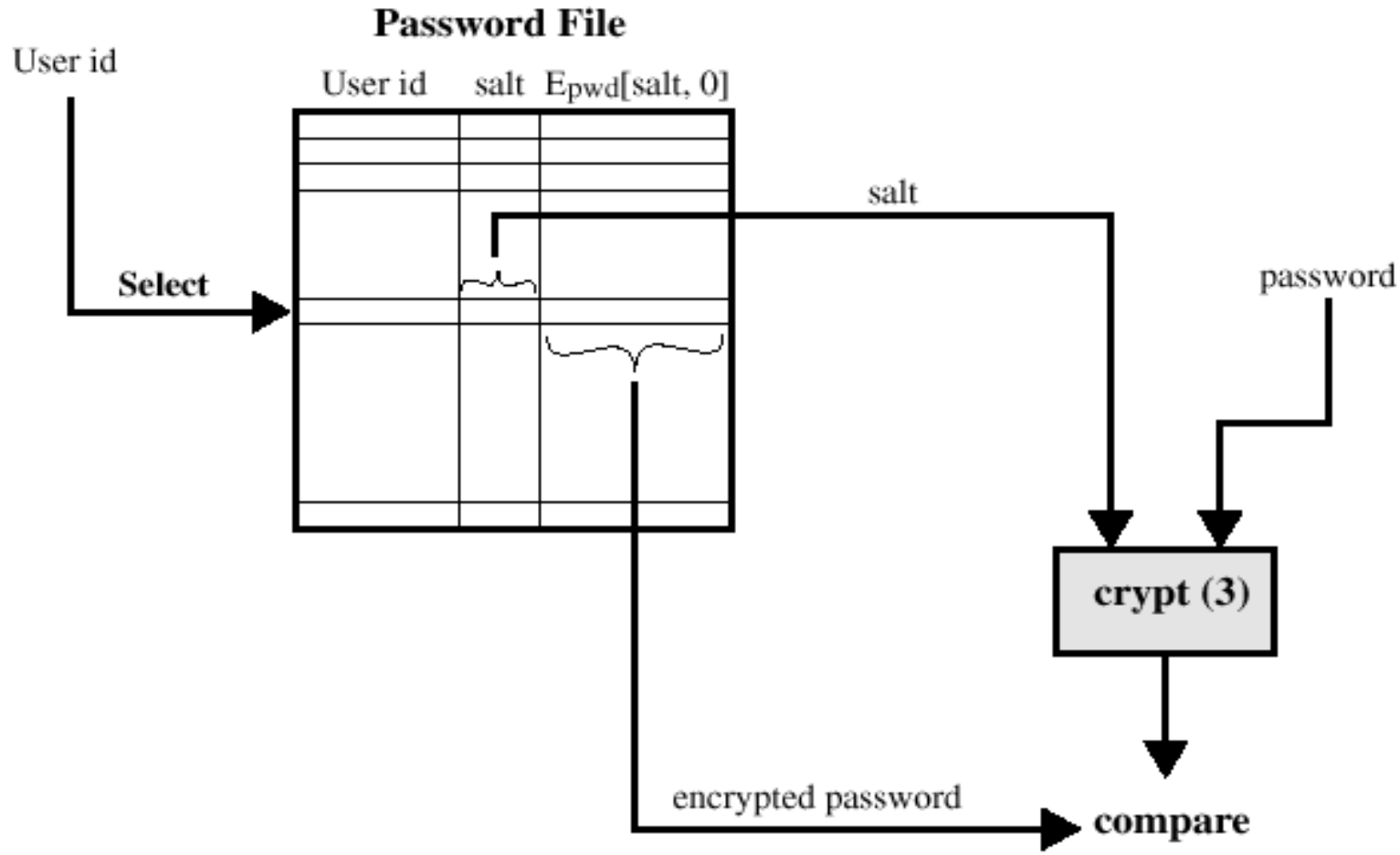
- It is used to make the dictionary attack a bit more difficult
- Salt is a 12 bit number between 0 and 4095
- It is derived from the system clock and the process identifier
- Rather than computing F(pwd), F(pwd + salt) is computed; both salt and F(pwd + salt) are stored in the password table
- When a user supplies the password, system fetches the salt for the user and computes F(pwd + salt) to check for a match
- Notice that with salt, the same password is computed in 4096 different ways

# UNIX Password Scheme




Loading a password

# UNIX Password Scheme




Verifying a password

# Dictionary Attacks on Passwords

- Attack 1:
  - Create a dictionary of common words and names and their simple transformations and use them to guess the password
- Attack 2:
  - Usually F is public  and so is the password file
    - In Unix, F is crypt(3) and /etc/passwd may be world readable
  - Compute F(word) for each word in the dictionary
  - A match gives the password
- Attack 3:
  - To speed up search, pre-compute F(dictionary)
  - A simple look up gives the password

These attacks work only with weak passwords

# Password Management Policy and Procedure

- Educate users to make better choices
  - Does not work if the user population is large or novice
- Define rules for good password selection and ask users to follow them
  - Rules serve as guideline for attackers
- Ask or force users to change their passwords periodically
- Force users to use machine generated passwords
  - Random password difficult to memorize; also password generator may become known to attacker through analysis
- Actively attempt to break users' passwords; force users to change those that are broken
  - Attacker may have better dictionary
- Screen password choices; if a choice is weak, force users to make a different choice

# Single Sign-on

- Having to remember many passwords for different services is a nuisance; with a single sign-on service, enter your password only once.
- A simplistic single-sign on service could store your password and do the job for you whenever you have to authenticate yourself.
  - Such a service adds to your convenience but it also raises new security concerns.
- System designers have to balance convenience and security; ease-of-use is important factor in making IT systems really useful, but many practices which are convenient also introduce new vulnerabilities.

# One-Time Passwords

- Password that can be used exactly *once*
  - After use, it is immediately invalidated
- Challenge-response mechanism
  - Challenge is number of authentications; response is password for that particular number
- Problems
  - Synchronization of user, system
  - Generation of good random passwords
  - Password distribution problem

# Lamport's Scheme

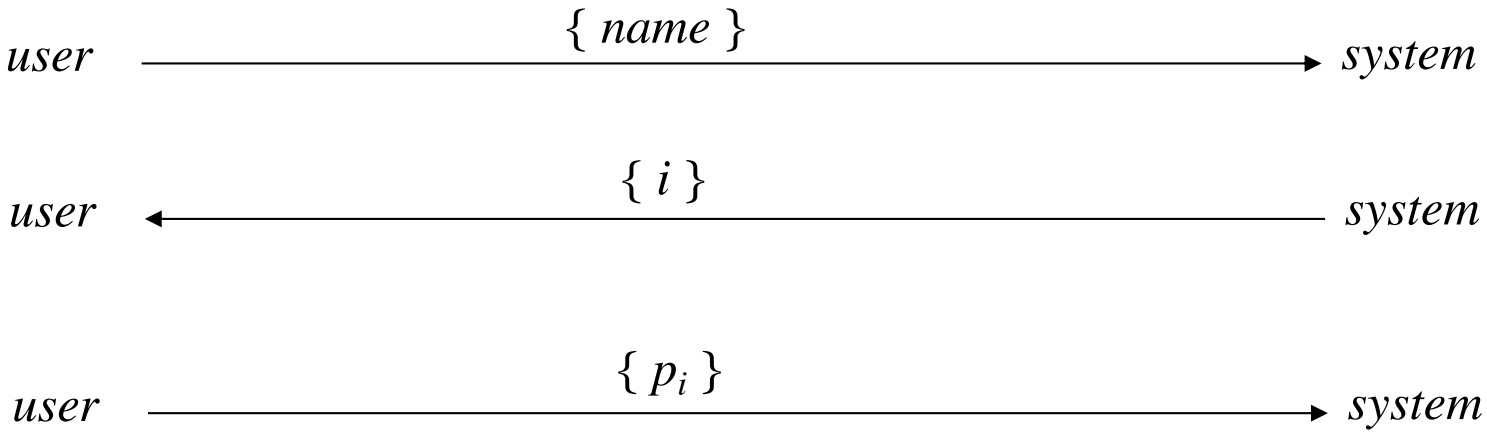
- Does not require any special hardware
- User selects x and computes F(x), $F^2$(x), ..., $F^{100}$(x) (This will allow 100 logins before a seed change)
- System stores (User name, $F^{100}$(x)) (need not know x)
- User supplies y = $F^{99}$(x) the first time
- System computes F(y) and compares it with $F^{100}$(x)
- If they match, the login is correct and the system replaces $F^{100}$(x) by $F^{99}$(x)
- User supplies $F^{98}$(x) the next time, and so on
- Knowing (intercepting) y does not reveal the next password ($F^{-1}$(y)) if F is a one-way function
- User calculates $F^n$(x) using a hand-held calculator, a trusted workstation, or a portable computer
- In Bellcore's implementation of this scheme, called S/Key, user calculates the sequence on a secure machine, encodes it as a sequence of short words, and prints it

# S/Key

- One-time password scheme based on idea of Lamport
- *h* one-way hash function (MD5 or SHA-1, for example)
- User chooses initial seed *k*
- System calculates:
  - $h(k) = k_1, h(k_1) = k_2, \ldots, h(k_{n-1}) = k_n$
- Passwords are reverse order:
  - $p_1 = k_n, p_2 = k_{n-1}, \ldots, p_{n-1} = k_2, p_n = k_1$

# S/Key Protocol

- System stores maximum number of authentications *n*, number of next authentication *I*, last correct supplied password $p_{i-1}$




- System computes h($p_{i-1}$)= h($k_{n-i+1}$) = $k_{n-i}$ = $p_{i-1}$ If this matches with stored, system replaces $p_{i-1}$ with $p_i$ and increments *i*.

# Key Points

- Authentication is not (precise as) cryptography
  - You have to consider system components
- Passwords provide a basis for most forms of authentication
  - Must be chosen well to resist attacks and not shared
- Protocols used are important
  - They can make masquerading harder
- Authentication methods can be combined
  - Bancomat & pin, biometrics & password