CSCI262 : System Security

Week 2: User Authentication

Schedule

- Authentication basics
- Password-based authentication
- Token-based authentication
- Biometric authentication

Authentication

- Authentication is the binding of an identity to a subject (a user or an entity).
- A user or entity is often required to authenticate itself to a computer system.
  - When (if) you use Internet banking you need to be authenticated by the bank site.
  - When using email you need to provide a password linked to your username or account name, that is to your identity, before you access your email.
  - If you are wondering why you do this … think what would happen if you weren't authenticated!

A Digital User Authentication Model

- NIST SP 800-63-3 defines a general model for user authentication.
- It involves two steps:
  - Initial step: user registration with the system
  - Second step: authentication

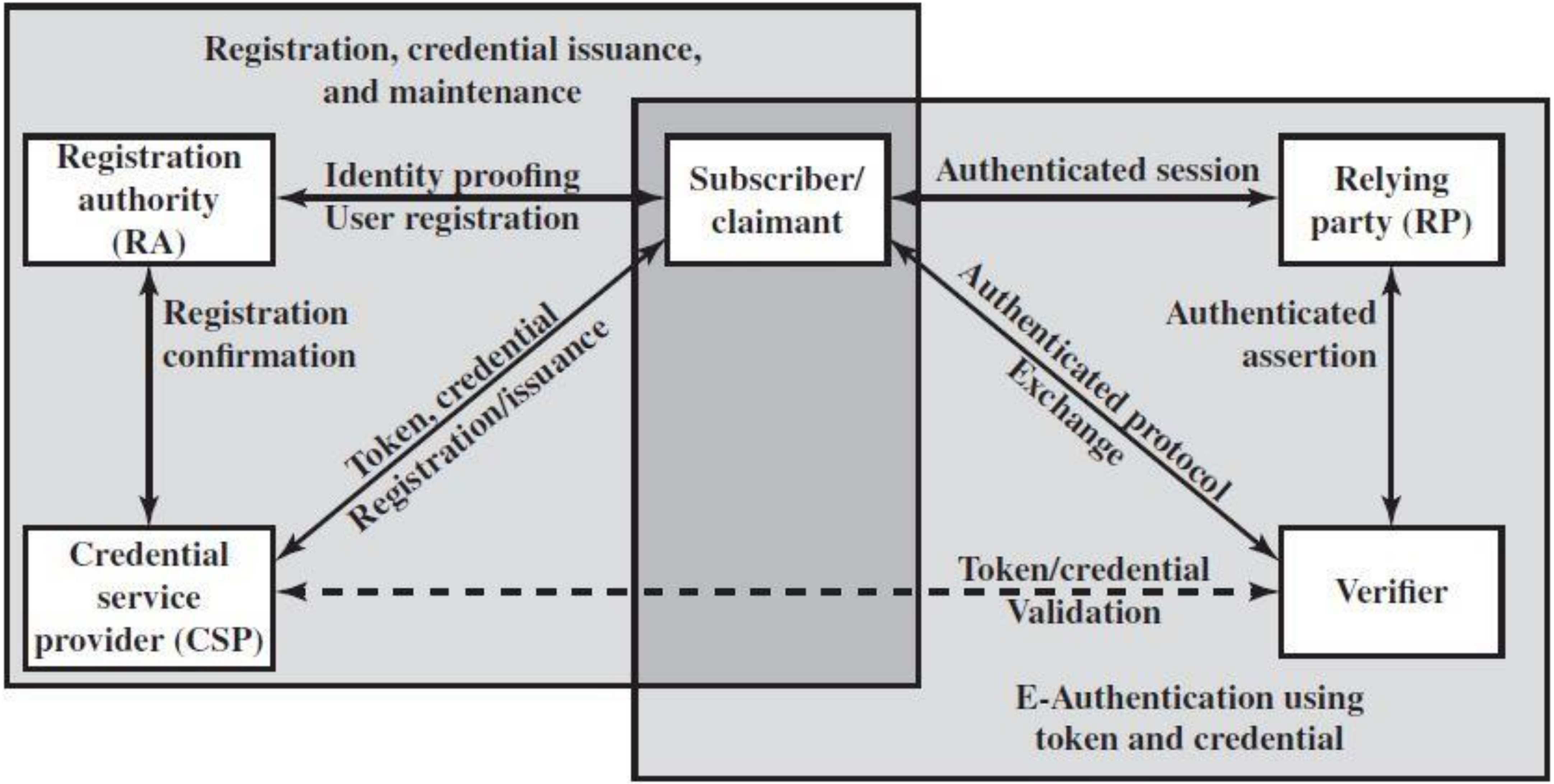


Figure 3.1 The NIST SP 800-63-3 E-Authentication Architectural Model [SB18, page 88]

Means of Authentication

- A subject, (a user or an entity), must provide information to enable the computer system to confirm its identity.
- This "information" could be one, or a combination, of the following:
  - **Something the individual knows**: a password, a personal identification number (PIN), or answers to a prearranged set of questions
  - **Something the individual possesses (token):** electronic keycards, smart cards, and physical keys.
  - **Something the individual is (static biometrics):** recognition by fingerprint, retina, and face.
  - **Something the individual does (dynamic biometrics):** recognition by voice pattern, handwriting characteristics, and typing rhythm.

Multifactor authentication

- Use more than one of the authentication means in the preceding list
- The strength of the system is determined by the number of factors incorporated by the system
    - Two factors are considered to be stronger than one, three factors are stronger than two . . .

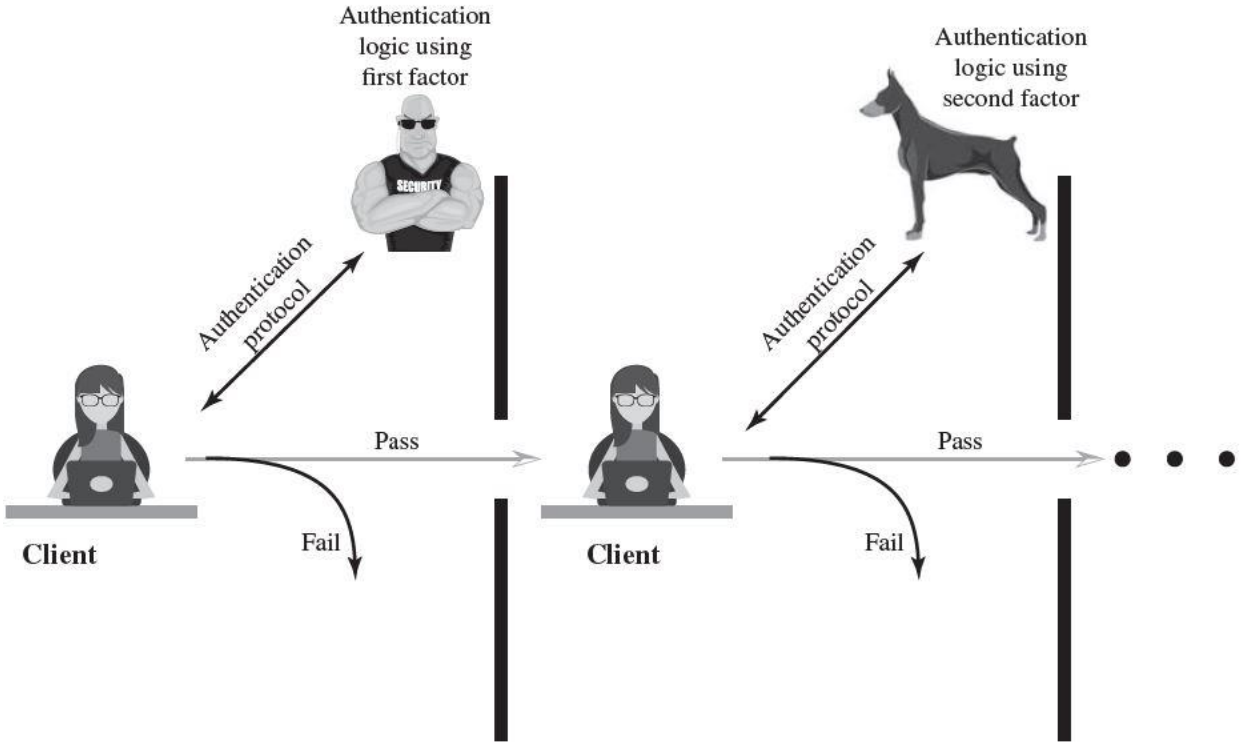


Figure 3.2 **Multifactor Authentication**

Password-based Authentication

- One simple and common method of user authentication is the password-based method.
  - Despite the many security vulnerabilities of passwords, they are the most commonly used user authentication technique.
- A **password** is information associated with an entity that confirms the entity's identity.

How do password systems work?

- Account registration:
  - Could be allocated/set by an administrator.
    - Passwords should be changed in this case.
  - Double entry of passwords.
  - Confirmation through an email.
- Authentication.
- Reset/Update/Recover.

Password Authentication

- The user supplies an identity.
- The user supplies a password.
  - If the authentication is remote this isn't necessarily what is sent to the server, that data may be some function of the password.
    - Local and remote authentication differ.
- The server checks the supplied information.
- If the password information matches with the user, the user's identity is authenticated; otherwise, the password is rejected.
- What happens without the identity?

False positives and negatives

- One concern in any situation where we attempt to distinguish between authorised and unauthorised entities, is the likelihood of getting a result which is wrong.
- Such errors are relevant in the context of intrusion detection systems, malware detection, spam, and also in the context of authentication.
- So in authentication we can make mistakes in checking.

- A **false positive** is when we make a match but “shouldn’t have”.
  - Sometimes when installing a game it will be flagged as a key logger, since some of the behaviour is consistent with a key logger.
- We have to be a little careful thinking about this for authentication.
  - There is the **false acceptance rate (FAR).**
  - This will be the proportion of authentication attempts resulting in false acceptances.
- Notice that the false positive effects are quite different in the context of “raising an alarm” from the context of “making a match”.

- A **false negative** is when we don’t make a match but “should have”.
  - We install a game we have downloaded, scan it and find nothing.
    - But actually it has a virus in it.
- Again, we have this reverse thinking for authentication.
  - There is the **false rejection rate** (FRR).
  - This will be the proportion of authentication attempts resulting in false rejections.

FAR and FRR

- How is it possible to have a FAR and a FRR?
  - In some authentication mechanisms, typically biometrics, there needs to be tolerance in matching.
- At least initially, we have no tolerance in matching, it’s bit sensitive, so these rates default to zero.
- FAR might make sense once we get to storing transformed passwords rather than plain visible passwords.
  - The use of cryptographic hash function
  - We will be back to this soon

Threats against password systems

- Password guessing.
- Password exposure.
- Login Trojan programs.
- Poor passwords.
- Common attacks:
    - Dictionary attacks.
    - Brute force attacks.
    - Hybrid attacks.
- On-line or off-line:
    - Compromise of the password file.

Password guessing

- It is pretty much always possible to *attempt to guess* a password online.
- You could, but shouldn't, try and guess the password of a user on **Capa** simply by trying to login as that user.
    - If you get logged in you have something acceptable as a password.
    - Depending on the authentication mechanism it may not actually be the password. More about this later.

Password exposure

- An “eavesdropper” may see the password when it is typed.
  - Typing very slowly isn’t a good move.
- Some users write their passwords down, even next to their computer.
  - This is often a bad idea. Writing your password down and physically securing it may be helpful.
- Some users pass their password to others.
  - Even if you appropriately protect your password, the other person may not.
  - Trust assumptions are critical in security.

Login Trojan Horses

- These are programs that produce an apparently genuine login screen.
- The user logs in, but the program captures the password and stores in along with the username for the malicious owner of the Login Trojan Horse.
- The program can subsequently pass the information to the genuine login program so the user doesn’t realize something is wrong.
  - The protection against this lies in not installing it in the first place.
- More on Trojan Horses under malware.

Poor password

- Passwords must be remembered by its owner, so users often choose simple passwords.
  - Often as simple as possible.
  - Many systems enforce significant restrictions on the passwords allowed.

Poor passwords → Dictionary attacks

- The requirements could be only a length restriction, so even if passwords meet the requirements they may be a dictionary word.
  - A dictionary attack exploits this.
- Dictionaries of common words can be used as sets of passwords to try.
- The attacker steps through the words in a dictionary and tries them as passwords.
- This dictionary attack may not succeed but is quite fast.

Tailored dictionary attacks

- The dictionary isn’t necessarily a complete English dictionary, or from another language.
  - A dictionary doesn’t have to just be real English words.
- For example, users may like cars or motorbikes, and a suitable dictionary could be a list of car or motorbike brands.
  - Or sports teams or players names or …
- Users may use even more personal information for passwords:
  - Birthdates, family names, pet names.

Brute force

- All password systems are vulnerable to somebody guessing the correct password.
- A brute force attack involves trying every possible password, and more generally every possible solution.

- Unlike a dictionary attack, brute force always works
  - The important factor is that this guessing is unlikely within the lifetime of the password.
  - Changing passwords makes a moving target that is “harder to hit”.
- With a brute force attack, you start with the letter a, then try aa, ab, ac, and so on until zz; then you try aaa, aab, aac, and so on.

Choosing secure passwords

- Time in seconds to definitely test the correct password is N/R, where:
  - N: Size of the set of possible passwords.
  - R: Number of passwords that can be tested in a second.
- The expected time, or expectation value of the associated random variable, is ½ of N/R.
- Consider a randomly generated password of length 8, from a character set of the lowercase letters a-z.
  - The size of the password space is 26^8=208827064576 = ~2*1011.
  - If we can test 1000 passwords a second, testing the whole space will take ~6.6 years.
- A password chosen as above is secure against that kind of attack, but the password is fairly hard to remember.
- And going to an arbitrary password over the 10 digits and upper and lower case → factor of 1000 increase.

Password entropy

- Entropy is to do with the information content, and randomness, and uncertainty.
  - Here it's probably helpful to think of the uncertainty of someone else regarding your password!
- Entropy is often measured in bits.
- If there are two equally likely options we have 1 bit of entropy, four equally likely options we have 2 bits of entropy and so on…
- The entropy for N equally likely options is $\log_2 N$.

Example: compute password's entropy

- Consider the following password of length 8 generated as follows:
  - First 2 characters are randomly chosen from the lower case letters from a to z.
  - Next 4 characters are random numbers from 0 to 9
  - Last 2 characters are randomly chosen from the upper case letters from A to Z
- What is the password's entropy?

Example: compute password's entropy

- First, compute the password space, i.e., the number N of all possible passwords.
  - There are 26 characters a-z: first 2 characters have 26^2 choices
  - There are 10 numbers from 0 to 9: next 4 character have 10^4 choices
  - There are 26 characters A-Z: last 2 characters have 26^2 choices.
- The total number of possible passwords is:

$$N = 26^2 \text{ x } 10^4 \text{ x } 26^2 = 4{,}569{,}760{,}000$$

- Hence the entropy is $\log_2 N \approx 32.08$, or 33 bit.

Trying to improve passwords

- Using pronounceable passwords makes remembering passwords easier. But …
  - … this reduces the number of possible passwords, since the number of vowels is likely to be fairly high. Every third character could be a vowel.
- Using a "pass-phrase" is another alternative.
- Dictionaries can be used for either of these scenarios so …
  - … we could use pass-phrases with intentional misspellings, odd capitalizations and symbol replacements. Or insert some numbers.

Hybrid attacks

- These modified methods are vulnerable to …
- … hybrid attacks.
- Basically, we use a dictionary as a basis but take variants on each of the words tested.
  - We might replace each lowercase “L” with 1, or “O” with 0, and so on.
- The hybrid attack falls between the dictionary and the brute force attack, in the time consumed, the number of passwords tried, and, therefore, in the (naïve) chance of success.

Personal phrase based: Helping memory?

- Humans are better at remembering things with structure.
- So … you can choose a phrase and take the first letter from each word as your password.
- Choosing a well-known phrase is not such as good idea → phrase dictionary attack.
- A rolling stone gathers no moss.

  Arsgnm
- My cat Boris has a long tail and 16 teeth.

  McBhalta16t

Protective mechanisms

- Keep track of incorrect password attempts:
  - Limit the number of account/passwords guesses per connect attempt.
  - Or lock the account when a threshold is exceeded.
    - Although the attacker can use this to perform a DOS attack. ☹
  - Or raise an alarm and try to trace the intruder.
- Slowly process passwords, it doesn’t make much difference to a legitimate user, but it makes a lot to the processing speed of an attacker.
  - An attacker will attempt to use many passwords.

“Online” versus “Offline” guessing

- Online vs offline:
  - Online (“live”) guessing will usually face restrictions on the number of attempts.
  - Offline attacks do not usually face this problem, and may take place without the awareness of the password owner or system administrator.
- Offline attacks can occur if an intruder accesses the password file on a computer.
- Or if the transmission of a password is intercepted, say in logging into a website on the Internet.
  - The interception may capture the password directly, which means the communication wasn’t adequately protected, or it may be some function of the password, possibly an encrypted or a hashed version, or a password protected file.
  - Communication security and cryptography is looked at more in CSCI361 and CSCI368.

- The distinction between “online” and “offline” isn’t really that important anymore.
- What is important?
  - The issue that really matters is whether the number of “guesses” is restricted or not.
  - The distinction completely changes the way in which attackers are likely to operate.
- If you can guess without restriction it is probably worthwhile trying a dictionary attack.
  - If you cannot guess without restriction then another approach, probably some form of social engineering, is probably more useful.

Rules for password systems

- These are examples of rules, the application and details should be context dependent.
  - Change passwords every 45 days.
  - Minimum length of eight (or higher) characters.
  - Must contain at least one alpha, one number, and one special character.
  - Alphabetical, numerical, and special characters must be mixed up.
    - For example, fg#g3s^hs5gw is good, abdheus#7 is not.
  - Cannot contain dictionary words.
  - Cannot reuse any of the previous five passwords.
  - After five failed logon attempts, password is locked for several hours.

UOW password rules …

- A password must contain 8-31 characters.
- A password must only contain printable characters.
- A password should have a combination of alphabetic, numeric or punctuation characters.
- A password is case sensitive, e.g. "a" is not the same as "A".
- A password cannot be re-used.
- A password cannot be based on your username (e.g. abc123) or your real name (e.g. jciti01) or any other personal information.
- A password must differ from your old password by at least 3 characters.
- Passwords will not be accepted if they are found to be on a national list of compromised passwords.
- See: https://www.uow.edu.au/its/accounts-passwords/

Protecting passwords

- So far we have focused on the individual user and their password.
- But there are locations where collections of passwords are stored, and such password repositories must be well protected.
- In UNIX, users passwords are not stored.
  - Hashes of the passwords are stored.
  - Such information is vulnerable to offline password guessing, so this file must be protected.
    - With appropriately hashed information it is computationally infeasible to find the associated password.

What is hashing?

- Hashing is a procedure often used in providing integrity for messages.
  - Integrity is about providing checks that a message hasn’t changed, in transmission or storage.
  - Hashing is also used in for indexing, in compilers and elsewhere, and for other purposes, but we are going to focus on cryptographic hash functions.
- One common integrity mechanism is the digital signature (see CSCI361), a function of the message signed and the signer.
  - Hashing is used to reduce the computational overhead of digital signatures.
- Informally, the hash of a message is a “fixed length fingerprint” of the block of data.

Hash Functions

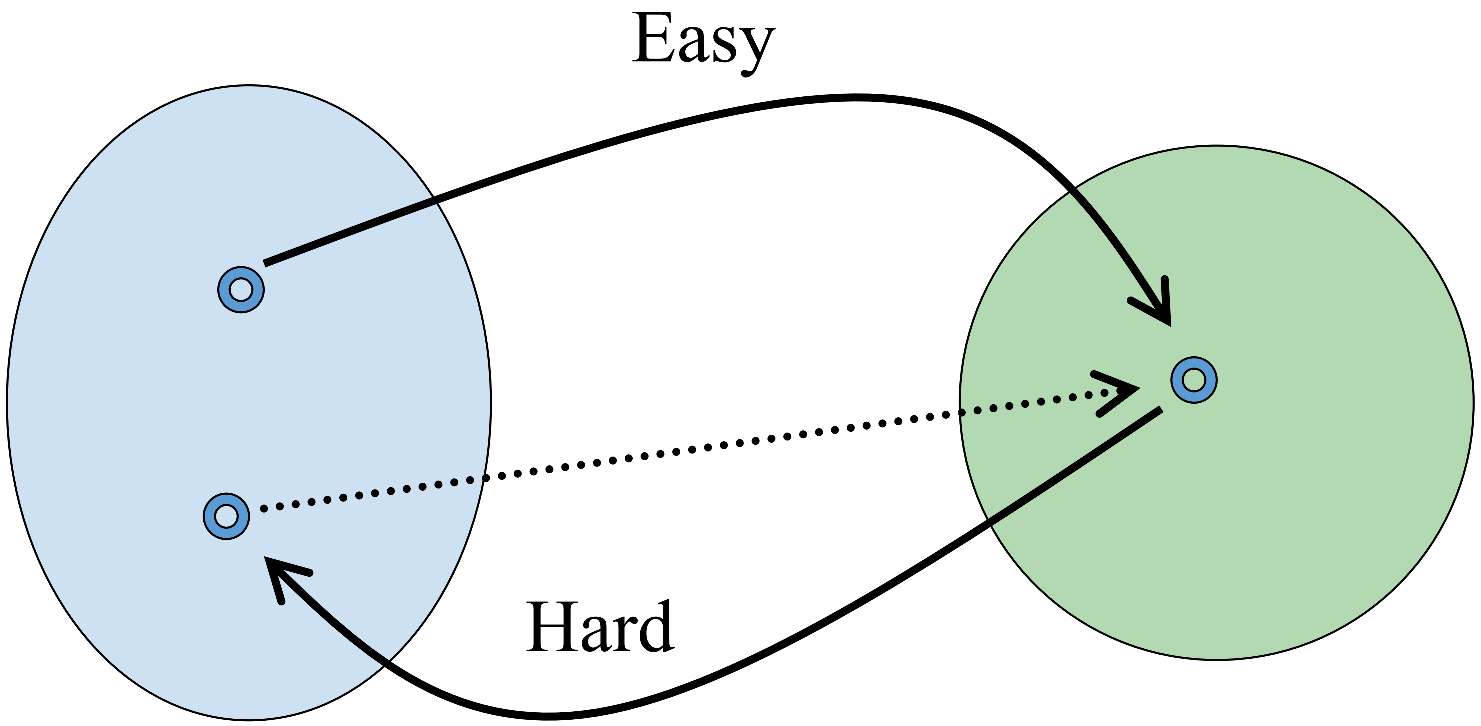
- A **hash function** transforms data from an arbitrary length into a fixed short length, pretty much anyway.
- Let H() be a hash function and X be a message.
  - The hash value of X can be efficiently computed as h = H(X).
  - The level of efficiency required depends on the application.
  - The output is referred to as the **hash value** or **message digest**.

- Two examples:
  1. H is a modulo function, so calculates x mod 8 for example.
  2. H takes the last 128 bits of a binary representation of X, unless the message is less than 128 bits, where it takes the whole message followed by enough bits to make the message digest 128 bits in length.
- These are little use as hash functions for cryptography, we need some additional properties.

Cryptographic Hash Functions

- We already have the first properties:
  - Our hash function can be applied to input of any size and produces a fixed size output.
- There are other properties, we will look at two here...
- **One-way or (vs) pre-image resistant**:
  - It is computationally infeasible that for a given message digest Y, we can find an X such that $H(X) = Y$.
- **Collision-resistant**:
  - A hash function H is called collision-resistant if it is computationally infeasible to find messages X and X', with $X \neq X'$, such that $H(X) = H(X')$.
- The examples on the previous page don’t satisfy these additional properties. (Why?)

Cryptographic Hashing



One-way vs pre-image resistant:
Collision-resistant:

MD5 and SHA-1

- MD5 is the most well-known and one of the most popular hash algorithms …
  - It produces a 128-bit message digest.
- Secure Hash Algorithm (SHA-1): A hash algorithm proposed and adopted by NIST, and to be used with DSA standard. It produces a 160 bit message digest and uses the design approach used in MD5.
- **Both are broken**, at least with respect to collisions.
- They are still both used.
- You don’t always need collision resistance and it is (a lot) harder to be collision resistant than pre-image resistant.
  - More on this in CSCI361.

More sources

- https://en.wikipedia.org/wiki/Hash_function_security_summary
- https://csrc.nist.gov/Projects/Hash-Functions/NIST-Policy-on-Hash-Functions
- https://www.streetdirectory.com/etoday/-ejcluw.html
- https://www.freecodecamp.org/news/md5-vs-sha-1-vs-sha-2-which-is-the-most-secure-encryption-hash-and-how-to-check-them/
- https://blog.jscrambler.com/hashing-algorithms

Back to the password file

- You might recall that we mentioned earlier that passwords are not directly stored in UNIX, instead the hash of them is stored.
- The password file is encrypted using a password known by the system.
- The password is inputted when the system is booted.
- Actually it is a little more complicated, we wouldn't typically directly store the hash either. ☺
  - At least not anymore.
  - We use something called **salting**.

Password Salting

- The “salt” is a value used to distinguish between users.
  - It’s typically random, or pseudo-random.
- The hash of the combination of the salt and the password, is stored.
- The salt is stored somewhere too.

user ID	salt value	password hash
Alice	3487	hash(3487||password_Alice)
Bob	8254	hash(8254||password_Bob)
Oscar	1098	hash(1098||password_Oscar)

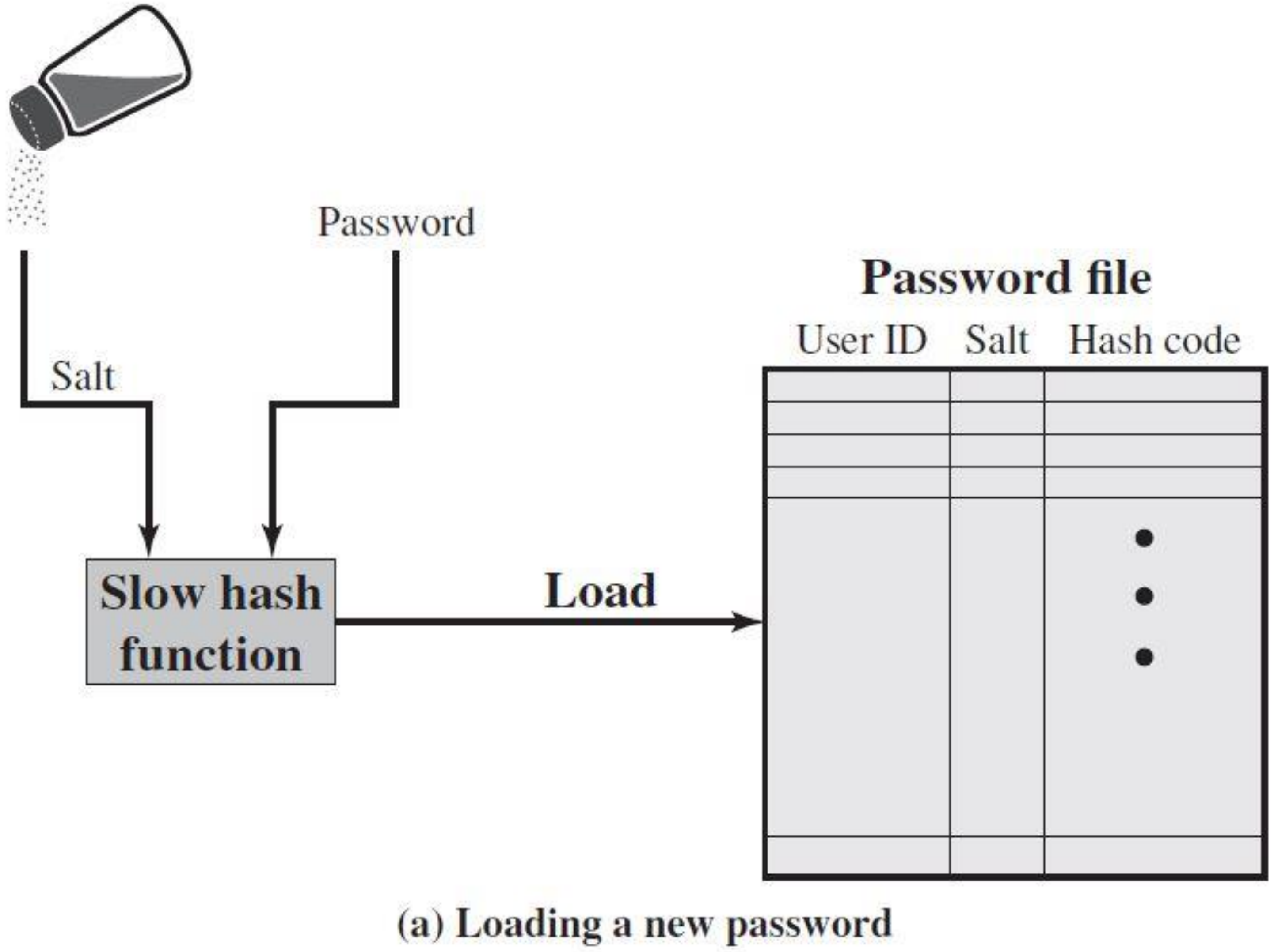


Figure 3.3(a) in [SB18]

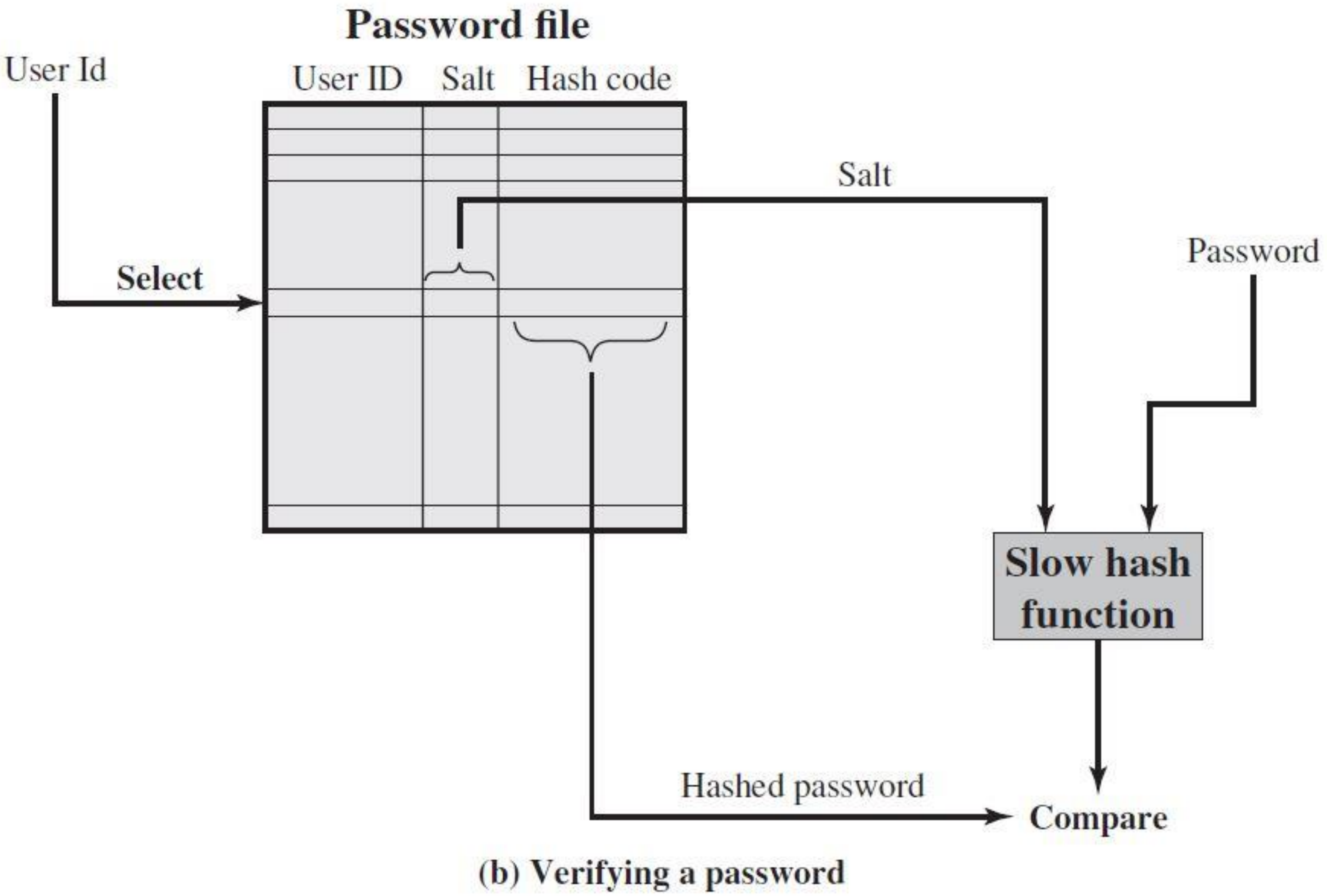


Figure 3.3(b) in [SB18]

The Salt serves three purposes:

- It prevents duplicate passwords from being visible in the password file.
    - Even if two users choose the same password, those passwords will be assigned different salt values. Hence, the hashed passwords of the two users will differ.
- It greatly increases the difficulty of offline dictionary attacks.
    - For a salt of length b bits, the number of possible passwords is increased by a factor of 2^b, increasing the difficulty of guessing a password in a dictionary attack.
- It becomes nearly impossible to find out whether a person with passwords on two or more systems has used the same password on all of them.

- With these hashed values someone might enter the wrong password but have it accepted ☺
  - Due to collisions in the hash function.
- This is a false positive and will contribute to the false acceptance rate.
  - It’s very unlikely though…

Password storage in UNIX: Where?

- Early versions of UNIX contained a file /etc/passwd, which stored all of the user IDs and protected/transformed passwords in the same file. This file was a text file and contained the user ID, encrypted password, home directory, and default shell. The following is a sample passwd file:

- ```
  root:6T1E6qZ2Q3QQ2:0:1:Super-User:/:/sbin/sh
  john:.D532YrN12G8c:1002:10::/usr/john:/bin/sh
  mike:WD.ADWz99Cjjc:1003:10::/usr/mike:/bin/sh
  . . .
   cathy:BYQpdSZZv3gOo:1010:10::/usr/cathy:/bin/sh
  frank:bY5CQKumRmv2g:1011:10::/usr/frank:/bin/sh
  tom:zYrxJGVGJzQL.:1012:10::/usr/tom:/bin/sh
  karen:OZFGkH258h8yg:1013:10::/usr/karen:/bin/sh
  ```
  ```

The general format for the passwd file

- **Username:passwd:UID:GID:full_name:home directory:shell**
- **Username**: Stores the username of whom the account belongs to.
- **Passwd**: Stores the user's transformed password.
  - If shadow files are used, an x appears in this location.
- **UID**: The user ID or the user identification number, generally chosen by the system.
- **GID**: The group ID or group identification number, which reflects the native group (base group of membership).
- **Full name**: This field usually contains the user's full names but is not mandatory.
- **Home Directory**: Stores the location of the user's home directory.
- **Shell**: Stores the user's default shell, which is what runs when the user first logs onto the system.

Shadow Files

- A solution to the readability problem.
- UNIX does, and has for a long time, split the passwd file information into two files.
- The passwd file still exists and contains everything except the protected passwords.
- A second file, shadow, was created.
  - This contains the transformed password and is only accessible to the root user.
- This information is stored centrally.
  - /etc/passwd
  - /etc/shadow

- Using "shadow passwords" is the preferred way of storing password hashes.
- You shouldn’t have any system that still stores password hashes in /etc/passwd.
- Consider the following pair of /etc/passwd and /etc/shadow files:

```
root:x:0:1:Super-User:/:/sbin/sh               root:6T1E6qZ2Q3QQ2:6445::::::
eric:x:1001:10::/usr/eric:/bin/sh              eric:T9ZsVMlmal6eA:::::::
John:x:1002:10::/usr/john:/bin/sh      John:.D532YrN12G8c:::::::
mike:x:1003:10::/usr/john:/bin/sh      mike:WD.ADWz99Cjjc:::::::
. . .                                  …
tim:x:1009:10::/usr/tim:/bin/sh                tim:sXu5NbSPLNEAI:::::::
cathy:x:1010:10::/usr/cathy:/bin/sh    cathy:BYQpdSZZv3gOo:::::::
frank:x:1011:10::/usr/frank:/bin/sh    frank:bY5CQKumRmv2g:::::::
tom:x:1012:10::/usr/tom:/bin/sh                tom:zYrxJGVGJzQL.:::::::
karen:x:1013:10::/usr/karen:/bin/sh    karen:OZFGkH258h8yg:::::::
```

Shadow files: The fields

username:passwd:last:min:max:warning:expire:disable

- **username**: The user's name of the account. There should be a corresponding line in the passwd file with the same username.
- **passwd**: Contains the transformed password.
- Only the first two fields are manditory.
- **last**: Contains the date of the last password change.
- **min**: The minimum number of days until the password can be changed.
- **max**: The maximum number of days until the password must be changed.
- **warning**: The number of days that the user is warned that the password must change.
- **expire**: The number of days in which the password expires and the account is disabled.
- **disable**: The number of days since the account has been disabled.

How safe are shadowed systems?

- Using shadow files is safer than before but …
- To break into a system using a password, an attacker needs a valid user ID and a password.
  - Valid user ID’s can be obtained from /etc/passwd.
  - A password guessing attack could then be launched.
  - It is (slightly) more difficult to detect single attempts against multiple users than multiple attempts against a single user.
- Although shadow files require root access, there were attacks that can be used to acquire a copy of the shadow file without obtaining root access directly.
  - For example imapd (a mail related server) and telnet were, at one time, both guilty of dumping core on occasion complete with the shadow file in the core where it was user-readable.
  - It is possible to recover information from the core.

Rainbow tables …

- Sometimes it is worthwhile, for an attacker, to do some pre-computation to speed up an attack.
    - **Rainbow tables** are an example.
- The basic idea is to have a lookup table of passwords, maybe salted, and the corresponding hash value...
- Having obtained a particular hash value we look it up in the rainbow table.
- Extended versions carry out more pre-computation to give a smaller lookup table.
    - This is done using hashing and reduction functions.

- **Reduction functions** (R) map from the hash output space back into whatever password space is considered appropriate.
- **Table construction**: Take an original password and hash it, then use a reduction function to take you to another password, which you then hash, then reduce, then hash and so on.
  - These sequences of H R H R ... are sometimes referred to as hash chains.
  - The saving comes in only storing the first (aaaaaa) and last elements (kiebgt) in a particular sequence.

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

Example from Wikipedia: https://en.wikipedia.org/wiki/Rainbow_table

- **Table lookup**: If you have a hash value you can check it in the table.
  - If it isn't in the table you reduce and hash it until it does appear in the table.
  - Having found the value in the table you know the password to start from, effectively a row of the table.
  - You can hash, then reduce and hash, until your original hash value appears.
  - The password immediately preceding that is the one you want.
  - Previous example continued: if you are given the hash 920ECF10, then we compute its chain by first applying R to get

$$\texttt{920ECF10} \xrightarrow[R]{} \texttt{kiebgt}$$

  - Since "kiebgt" is one of the endpoints in our table, we then take the corresponding starting password "aaaaaa" and follow its chain until 920ECF10 is reached:

$$\texttt{aaaaaa} \xrightarrow[H]{} \texttt{281DAF40} \xrightarrow[R]{} \texttt{sgfnyd} \xrightarrow[H]{} \texttt{920ECF10}$$

  - Then the password is "sgfnyd" (or a different password that has the same hash value).

More on rainbow table

- https://en.wikipedia.org/wiki/Rainbow_table
- https://paperzz.com/doc/7842274/rainbow-tables
- https://www.youtube.com/watch?v=rv06bwwAQqM
- https://crypto.stackexchange.com/questions/5900/example-rainbow-table-generation

One-time passwords

- With a one-time password system the user and the system have an ordered collection, or list, of valid passwords such that each one is valid only once.
- Provided the passwords are not obviously correlated, this system is immune to eavesdropping.
- An observed password leaks no information about the other passwords.
- **Problems**: The number of passwords to be shared (established) and stored.

Why are these problems?

- The passwords need to be established requiring significant initial costs.
  - More in CSCI368 on key establishment and cryptographic authentication.
- From the point of view of the server they need to store more information.
- From the point of view of the user they are more likely to write down passwords and be less careful in choosing them.
  - Users cannot rely on repeated usage to reinforce their memory of a single password.

Lamport's One-time Password

- **A**lice, the user, remembers a password.
- **B**ob, the server (computer), has a database in which it stores, for each user, ...
  - The username U_i.
  - A counter n that decrements each time Bob authenticates the user.
  - The hash value $x_n=h^n(\text{password})$, for some specified hash function h.
    - $h^i(X)=h(h^{i-1}(X))$ and $h^0(X)=X$.
- The setup phase is establishing this information.

How does it work?

- **A**lice has a workstation, and **B**ob is the server.
- To authenticate we use the following protocol:

  Workstation $\rightarrow$ **B**ob : **A**lice

  Bob $\rightarrow$ Workstation : n

  Workstation $\rightarrow$ **B**ob : h^{n-1}(password)
- **B**ob checks if

$$h(\, h^{n-1}(\text{password})) = h^{n}(\text{password})$$

- If it does then **B**ob accepts the communicating party as **A**lice. If it doesn't **B**ob rejects the communication.

- Once Alice has been authenticated, the server needs to update it's information.
- We replace $x_n=h^n$(password) with the one-time password sent by Alice's workstation … $x_{n-1}=h^{n-1}$(password).
- The value n is replaced by n-1.
- When n reaches 0 we will have run out of passwords in the hash chain and will have to run a new setup process, with a new base password.

Alice and Bob?

- Alice and Bob are the names typically used to represent the participants in two-party cryptographic protocols.
- There are various other cryptographic identities, so of whom you might meet in next year or so.
- Eve the eavesdropper, Mel the malicious entity, Oscar the opponent, Peggy the prover, Victor the verifier, and so on ...

Token-based Authentication

- Objects that a user possesses for the purpose of user authentication are called **tokens**.
- We consider two widely used types of tokens:
    - memory cards
    - smart cards.

Memory cards

- Memory cards can store but not process data
- Examples:
  - bank card with magnetic stripe on the back
  - hotel room card
- A magnetic stripe can store only a simple security code
  - which can be read by an inexpensive card reader
- **Authentication:** user provides both the memory card and some form of password or personal identification number (PIN)
- Adversary needs to gain physical possession of the card (or be able to duplicate it) plus must gain knowledge of PIN

Memory cards - drawbacks

- Requires special reader: increase cost of using the token and maintain the security of the reader's hardware and software
- Token loss: A lost token temporarily prevents its owner from gaining system access → require administrative cost in replacing the lost token
- User dissatisfaction.

Smart cards

- A wide variety of devices qualify as smart tokens.
- Categories:
  - **Physical characteristics**: Smart tokens include an embedded microprocessor (e.g., bank card)
  - **User interface**: Manual interfaces include a keypad and display for human/token interaction.
  - **Electronic interface**: A smart card or other token requires an electronic interface to communicate with a compatible reader/writer.
    - Contact or contactless
  - **Authentication protocol**: smart token to provide a mean for user authentication
    - Static or dynamic password generator or challenge-response

- A smart card contains a microprocessor, including processor, memory and I/O ports.
- Include 3 types of memory:
  - Read-only memory (ROM)
  - Electrically erasable programmable ROM (EEPROM)
  - Random access memory (RAM)

- Typical interaction between a smart card and a reader or computer system.

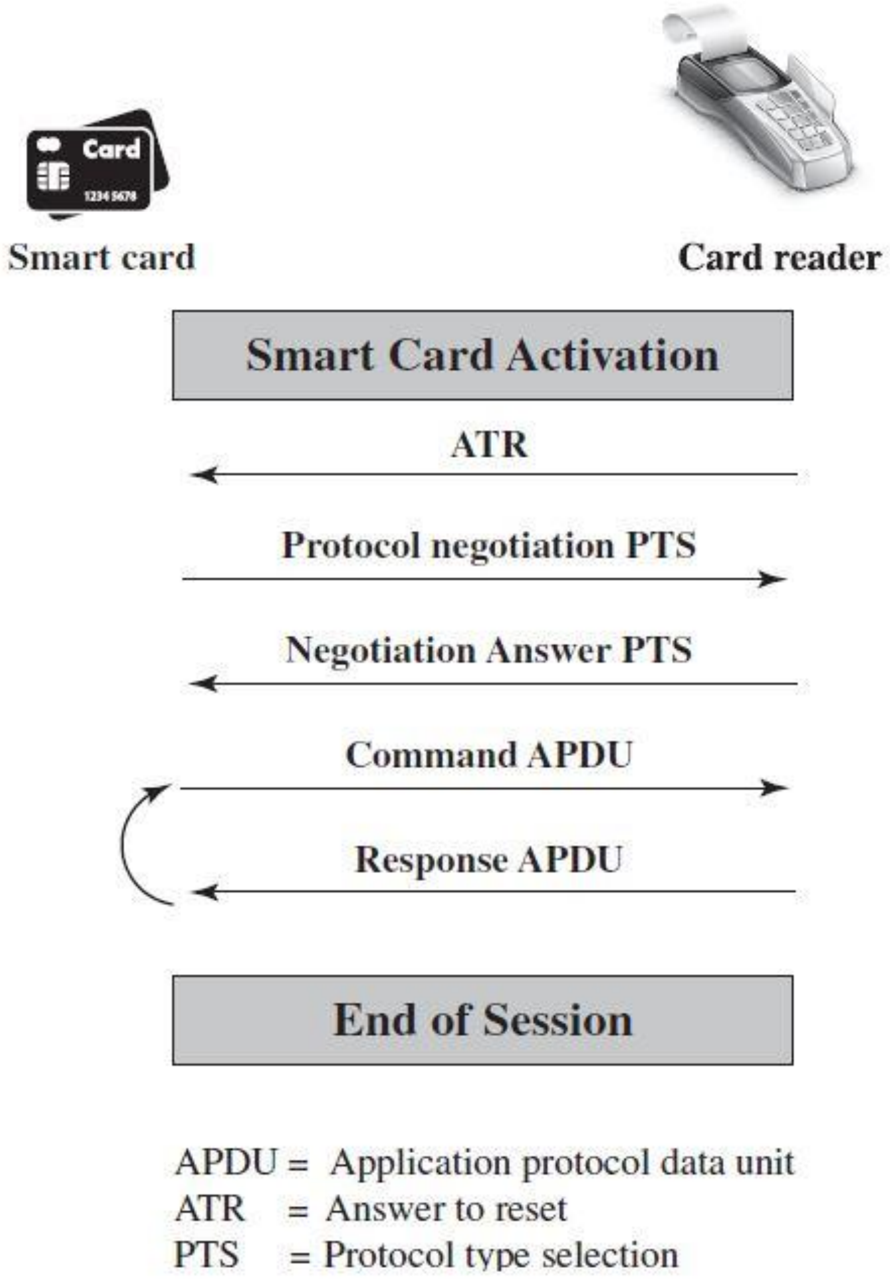


Figure 3.6 **Smart Card/Reader Exchange**

Biometric Authentication

- Biometrics for authentication should only ever be used as a component of a multi-factor authentication system.
- Why?
  - They are not private!
- Biometrics are generally used to make attacks by outsiders more difficult, and probably more expensive.
- Biometrics tend to most reliable where the use is supervised, by a guard perhaps
- Beware the Jelly Baby trick.
  - https://www.neowin.net/news/jelly-babies-dupe-fingerprint-security/

Types ...

- Face recognition.
- Handwriting.
- Fingerprints.
  - The most widely used requiring significant technology.
- Iris codes.
  - Probably the best hope for a robust biometric system, although it still has some problems.
- Voice recognition.
- DNA.
- Signature

Figure 3.8 (below) in [SB18] gives a rough indication of the relative cost and accuracy of these biometric measures.

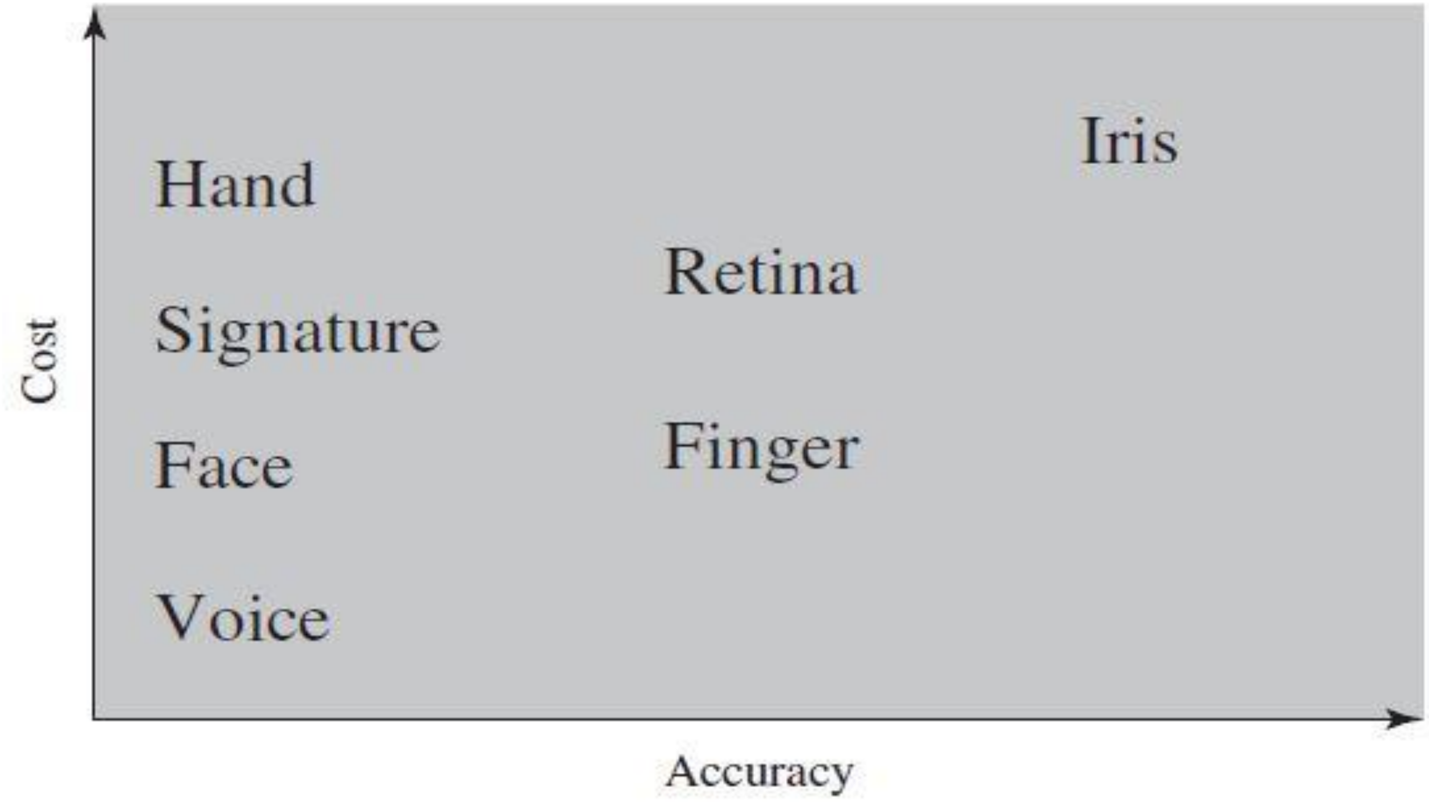


Figure 3.8 Cost Versus Accuracy of Various Biometric Characteristics in User Authentication Schemes

UOW Learning Co-Op – uow.info/learningcoop

ACADEMIC SUPPORT

FREE Academic Consultations and Study Support Seminars

- Assessment Writing: Understanding and writing assessments, and interpreting feedback
- English Language Skills
- Maths and Stats Skills
- Thesis and Journal Writing
- Book or request a consultation via the Academic Consultations page
- Register for a seminar via the Study Support Seminars page
- Email for more information: learning-development@uow.edu.au

Self Help Resources

- Writing, maths and stats, and study guides
- Assignment (planning) calculator
- Recorded seminars (go to the Study Support Seminars page)

Studiosity 24/7 support – through the link in Moodle subject pages

- Short question chats
- Written feedback on drafts

"Excellent feedback, very knowledgeable, very easy to talk to and encouraging"

"I really appreciate the opportunity for the consultations, it has really benefited me this semester. The consultation is always extremely easy to understand and so helpful"

- **Print - 15% discount has been added**

  https://www.pearson.com/store/p/computer-security-principles-and-practice-global-edition/P200000005478/9781292220611

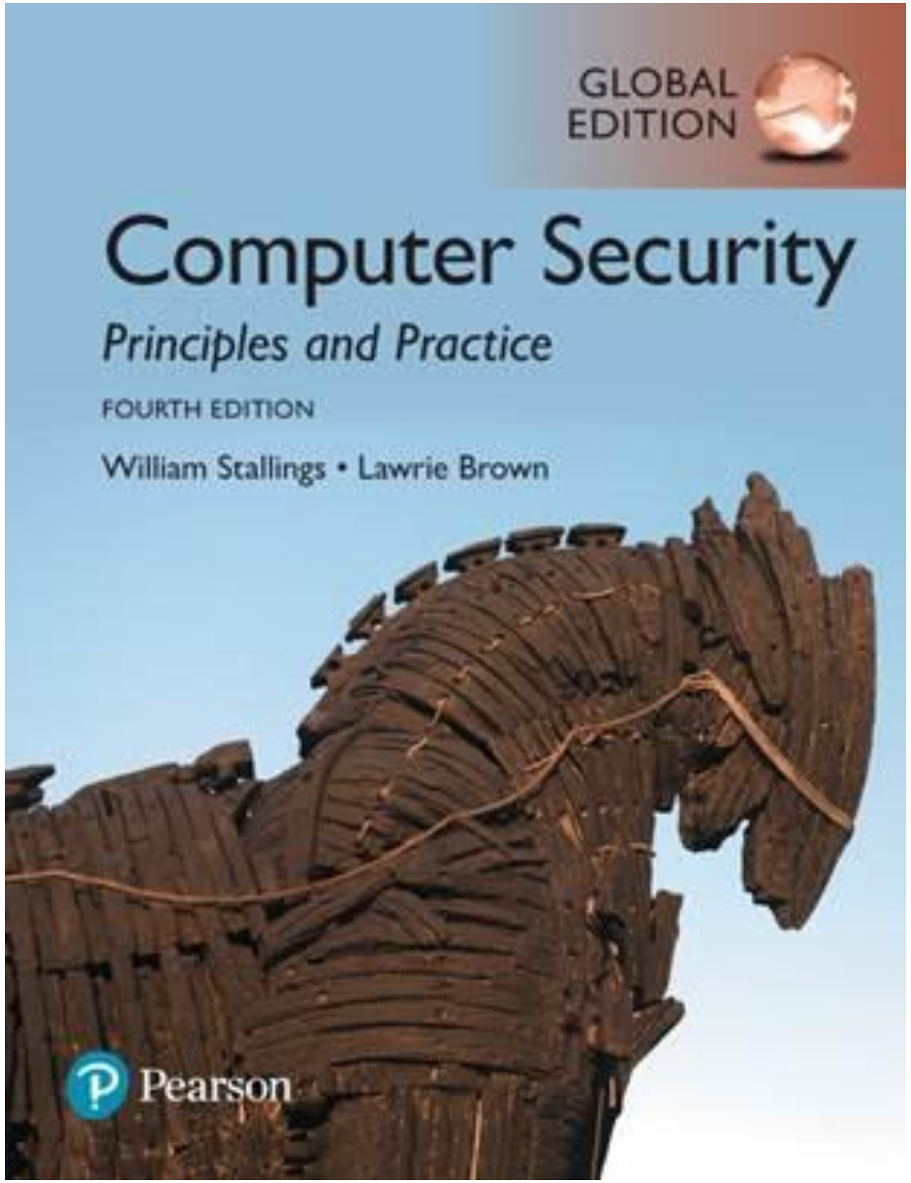
- **eText - 10% discount has been added**

  https://www.pearson.com/store/p/computer-security-principles-and-practice-global-edition/P200000005478/9781292220635

- Students can use the following promo code to receive an **additional 10% discount -**

  22S2-UOW

  (expires on the 21st of August 2022).



- **Print - 15% discount has been added** https://www.pearson.com/store/p/computer-security-art-and-science/P200000000134/9780321712332

- **eText - 10% discount has been added** https://www.pearson.com/store/p/computer-security-art-and-science/P200000000134/9780134097176

- Students can use the following promo code to receive an **additional 10% discount -**

22S2-UOW

(expires on the 21st of August 2022).