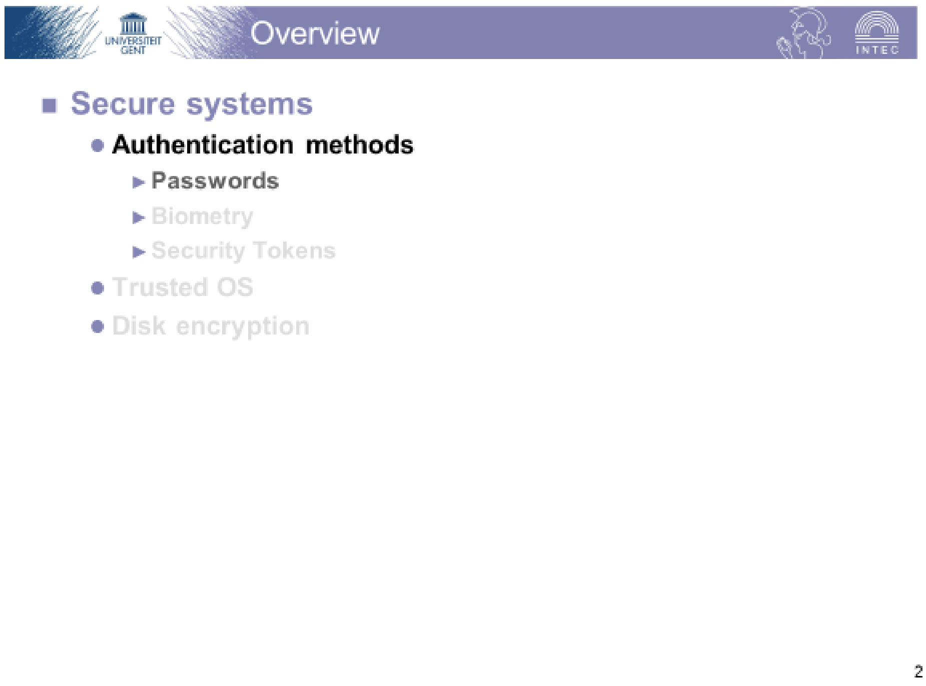
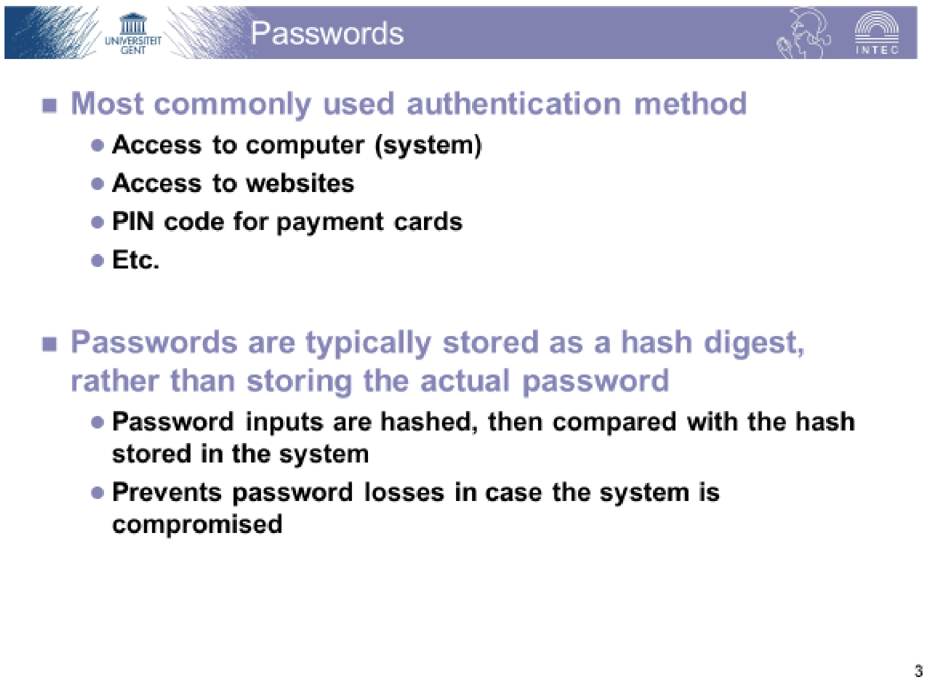


Overview

- Secure systems
  - Authentication methods
    - Passwords
    - Biometry
    - Security Tokens
  - Trusted OS
  - Disk encryption

2

Different methods exist to authenticating an entity requesting access to system. Typically, certificate based methods (Kerberos, X509) are not directly suitable for user logins into local systems.

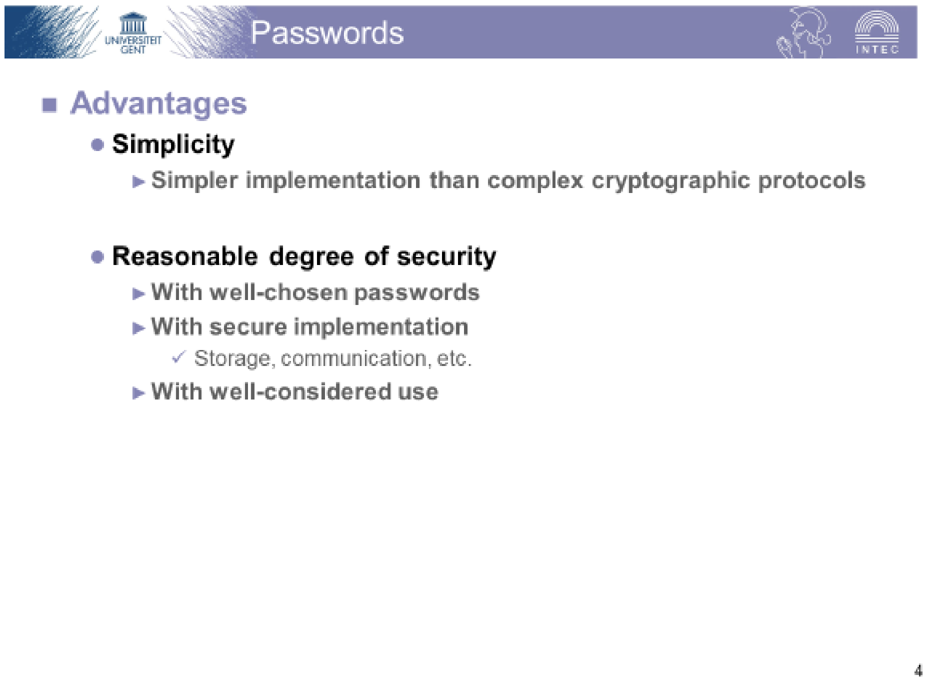


Passwords

- Most commonly used authentication method
  - Access to computer (system)
  - Access to websites
  - PIN code for payment cards
  - Etc.
- Passwords are typically stored as a hash digest, rather than storing the actual password
  - Password inputs are hashed, then compared with the hash stored in the system
  - Prevents password losses in case the system is compromised

3

Storing all user passwords as cleartext can result in a massive security breach if the password file is compromised. One way to reduce this danger is to only store the hash digest of each password. To authenticate a user, the password presented by the user is hashed and compared with the stored hash. Note that this approach prevents the original passwords from being retrieved if forgotten or lost, and they have to be replaced with new ones.

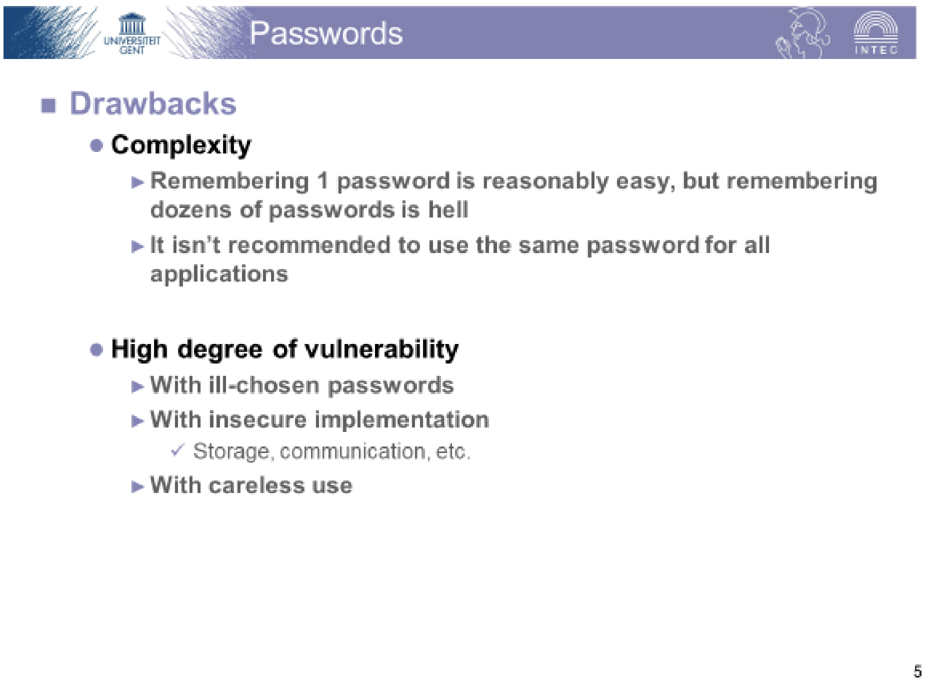


Passwords

- Advantages
  - Simplicity
    - Simpler implementation than complex cryptographic protocols
  - Reasonable degree of security
    - With well-chosen passwords
    - With secure implementation
      - ✓ Storage, communication, etc.
    - With well-considered use

4

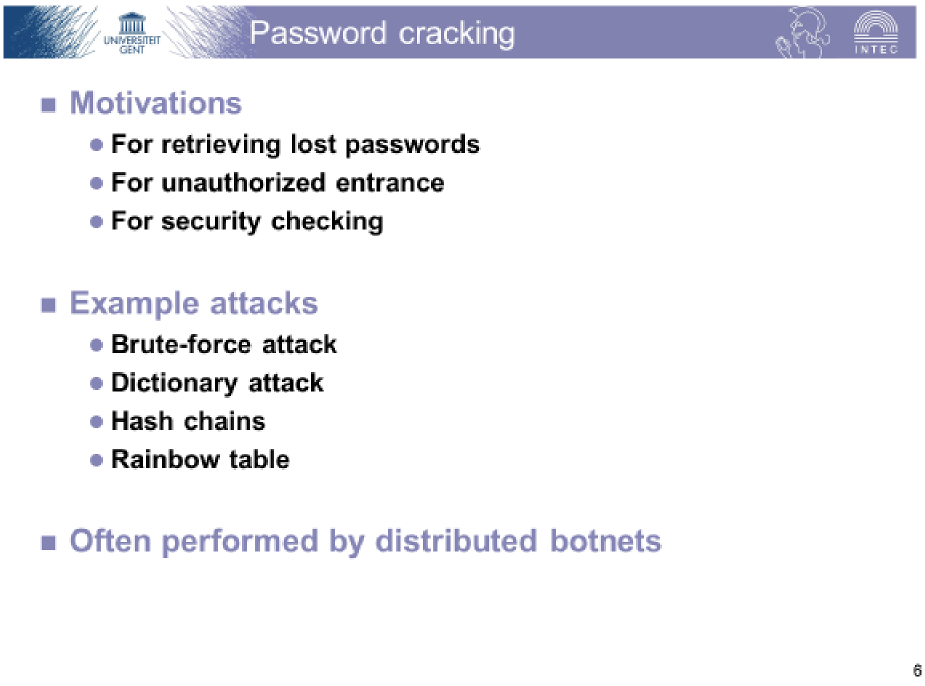
A good password shouldn’t be easy to guess for a potential attacker. A secure implementation implies e.g. that a password shouldn’t be stored or communicated in plaintext (certainly not over an ill-secured wireless connection). However, transmitting a password over an SSH or SSL/TLS connection is acceptable. A password shouldn’t be simply written down, although secure storage of passwords (e.g. using PGP) can be justified. When typing passwords one should also take care for possible “shoulder surfing” (think twice about this when you’re typing your password on a smartphone or tablet). Don’t use passwords for potentially unsecured or ill-secured applications (logging in at non-critical websites, unsecure POP-retrieval of e-mail, etc.) for applications where communication is secure (and reliable). The system on which a password is used should also be secured against other threats such as “keyloggers” (malware registering keyboard input), but this system security is also a requirement for many other security mechanisms.



Passwords

- Drawbacks
  - Complexity
    - Remembering 1 password is reasonably easy, but remembering dozens of passwords is hell
    - It isn't recommended to use the same password for all applications
  - High degree of vulnerability
    - With ill-chosen passwords
    - With insecure implementation
      - ✓ Storage, communication, etc.
    - With careless use

5

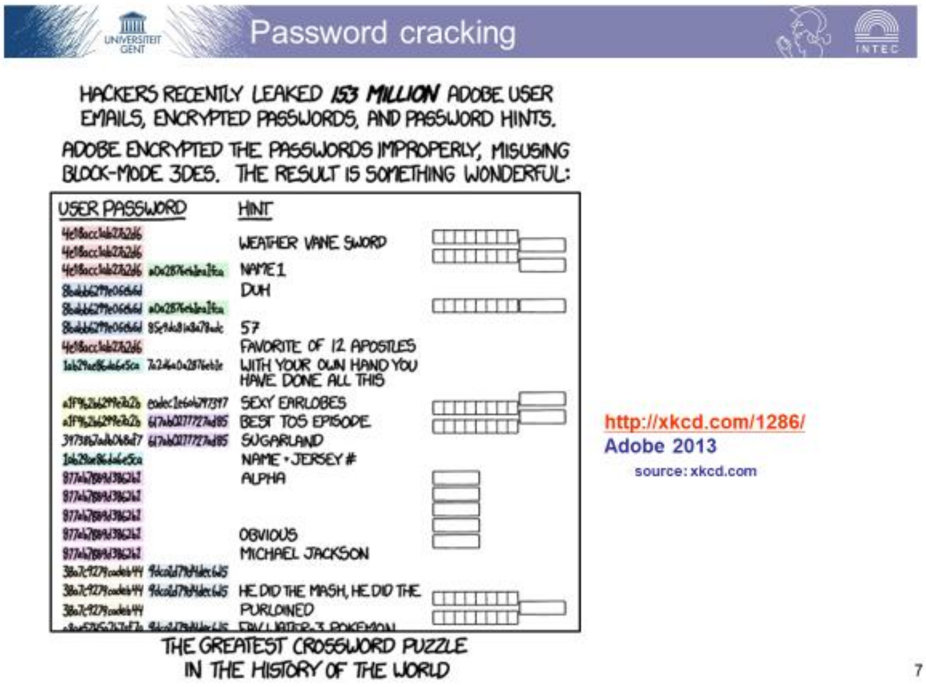


Password cracking

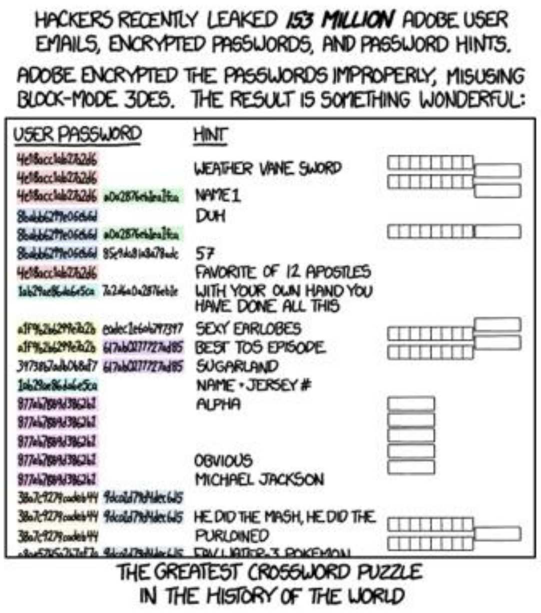
- Motivations
  - For retrieving lost passwords
  - For unauthorized entrance
  - For security checking
- Example attacks
  - Brute-force attack
  - Dictionary attack
  - Hash chains
  - Rainbow table
- Often performed by distributed botnets

6

In cryptanalysis and computer security, password cracking is the process of recovering passwords from data that have been stored in or transmitted by a computer system. The purpose of password cracking might be to help a user recover a forgotten password (installing an entirely new password is less of a security risk, but it involves System Administration privileges), to gain unauthorized access to a system, or as a preventive measure by System Administrators to check for easily crackable passwords.

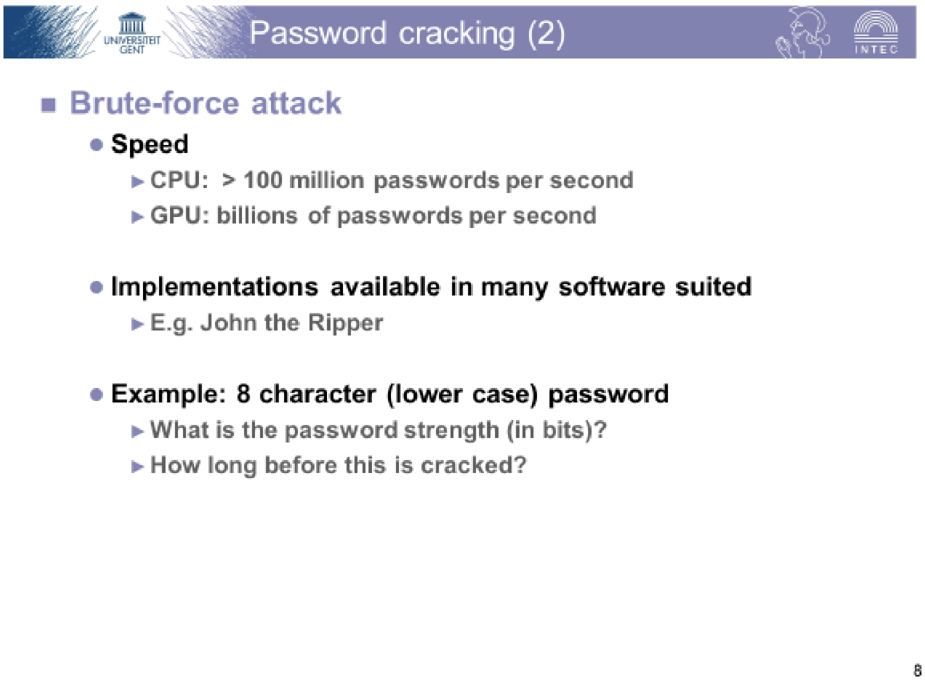


Password cracking


HACKERS RECENTLY LEAKED 153 MILLION ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS.
ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:
USER PASSWORD
HINT
WEATHER VANE SWORD
NAME1
DUH
57
FAVORITE OF 12 APOSTLES
WITH YOUR OWN HAND YOU HAVE DONE ALL THIS
SEXY EARLOBES
BEST TOS EPISODE
SUGARLAND
NAME + JERSEY #
ALPHA
OBVIOUS
MICHAEL JACKSON
HE DID THE MASH, HE DID THE
PURLOINED
THE GREATEST CROSSWORD PUZZLE IN THE HISTORY OF THE WORLD

http://xkcd.com/1286/
Adobe 2013
source: xkcd.com

7



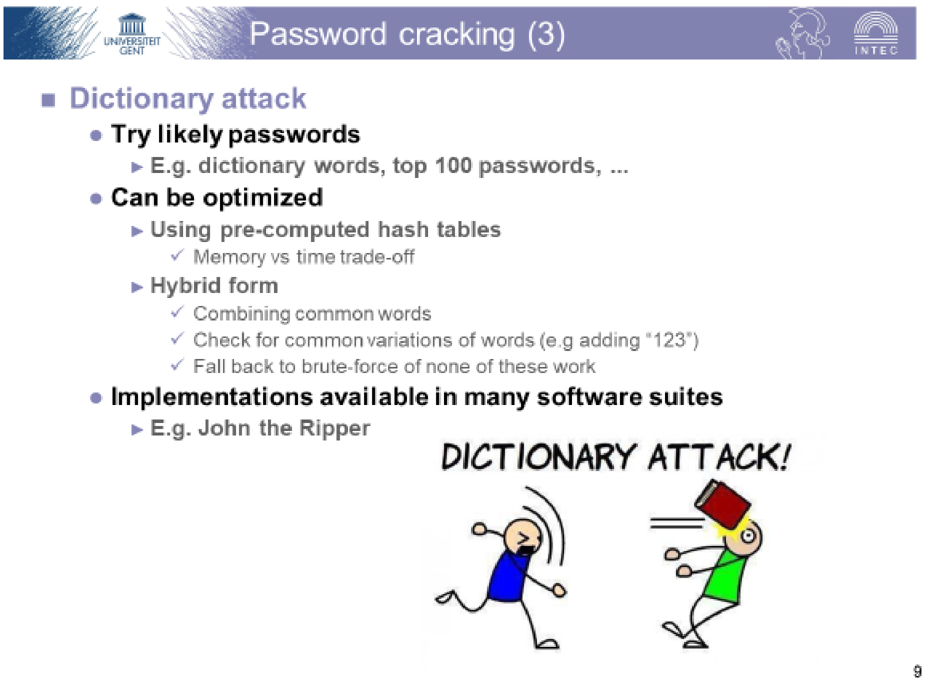
Password cracking (2)

- Brute-force attack
  - Speed
    - CPU: > 100 million passwords per second
    - GPU: billions of passwords per second
  - Implementations available in many software suited
    - E.g. John the Ripper
  - Example: 8 character (lower case) password
    - What is the password strength (in bits)?
    - How long before this is cracked?

8

Brute force attacks work by calculating every possible combination that could make up a password and testing it to see if it is the correct password. As the password's length increases, the amount of time, on average, to find the correct password increases exponentially. This means short passwords can usually be discovered quite quickly, but longer passwords may take decades.

For some kinds of password hashes, ordinary desktop computers can test over a hundred million passwords per second using password cracking tools running on a general purpose CPU and billions of passwords per second using GPU-based password cracking tools. The rate of password guessing depends heavily on the cryptographic function used by the system to generate password hashes. A suitable password hashing function, such as bcrypt, is many orders of magnitude better than a naive function like simple MD5 or SHA. A user-selected eight-character password with numbers, mixed case, and symbols, with commonly selected passwords and other dictionary matches filtered out, reaches an estimated 30-bit strength, according to NIST. 2^{30} is only one billion permutations and would be cracked in seconds if the hashing function is naive. When multiple ordinary desktop computers are combined in a cracking effort, as can be done with botnets, the capabilities of password cracking are considerably extended. In 2002, distributed.net successfully found a 64-bit RC5 key in four years, in an effort which included over 300,000 different computers at various times, and which generated an average of over 12 billion keys per second. Graphics processors can speed up password cracking by a factor of 50 to 100 over general purpose computers. As of 2011, available commercial products claim the ability to test up to 2,800,000,000 passwords a second on a standard desktop computer using a high-end graphics processor. Such a device can crack a 10 letter single-case password in one day. Note that the work can be distributed over many computers for an additional speedup proportional to the number of available computers with comparable GPUs.



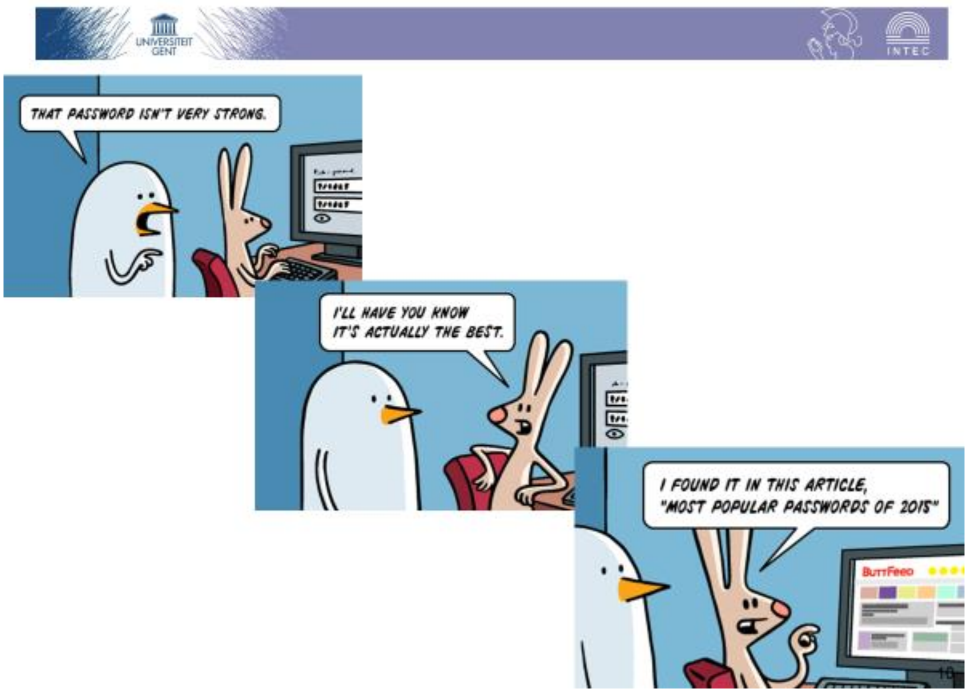
Password cracking (3)

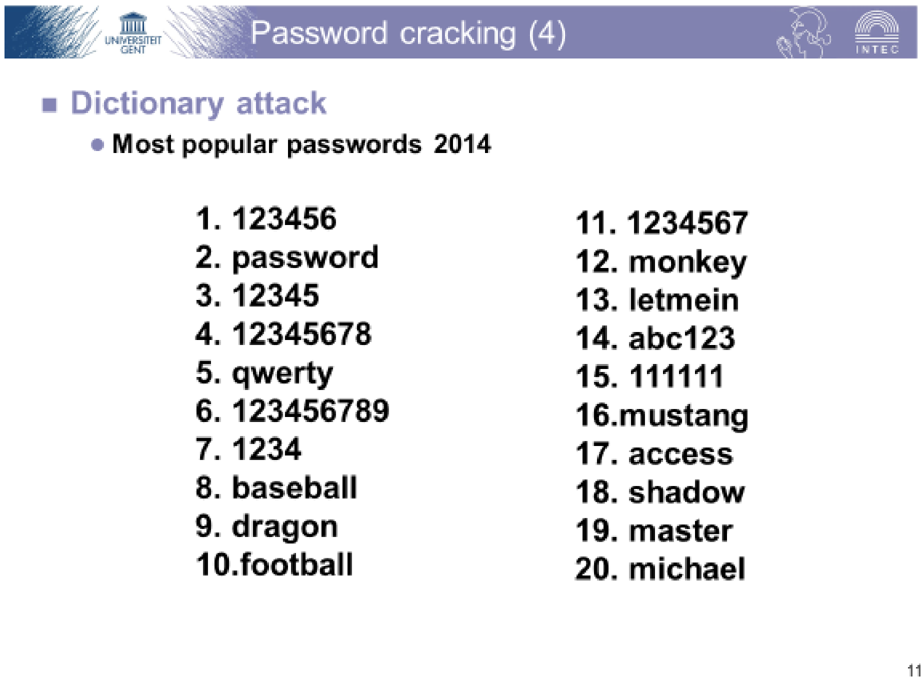
- Dictionary attack
  - Try likely passwords
    - E.g. dictionary words, top 100 passwords, ...
  - Can be optimized
    - Using pre-computed hash tables
      - ✓ Memory vs time trade-off
    - Hybrid form
      - ✓ Combining common words
      - ✓ Check for common variations of words (e.g adding "123")
      - ✓ Fall back to brute-force of none of these work
  - Implementations available in many software suites
    - E.g. John the Ripper

9

A dictionary attack is a technique for defeating a cipher or authentication mechanism by trying to determine its decryption key or passphrase by trying hundreds or sometimes millions of likely possibilities, such as words in a dictionary. A dictionary attack is based on trying all the strings in a pre-arranged listing, typically derived from a list of words such as in a dictionary (hence the phrase dictionary attack). In contrast to a brute force attack, where a large proportion of the key space is searched systematically, a dictionary attack tries only those possibilities which are deemed most likely to succeed. Dictionary attacks often succeed because many people have a tendency to choose short passwords that are ordinary words or common passwords, or simple variants obtained, for example, by appending a digit or punctuation character. Dictionary attacks are relatively easy to defeat, e.g. by choosing a password that is not a simple variant of a word found in any dictionary or listing of commonly used passwords. It is possible to achieve a time-space tradeoff by pre-computing a list of hashes of dictionary words, and storing these in a database using the hash as the key. This requires a considerable amount of preparation time, but allows the actual attack to be executed faster. The storage requirements for the pre-computed tables were once a major cost, but are less of an issue today because of the low cost of disk storage. Pre-computed dictionary attacks are particularly effective when a large number of passwords are to be cracked. The pre-computed dictionary need only be generated once, and when it is completed, password hashes can be looked up almost instantly at any time to find the corresponding password.

There are of course attacks which leverage both techniques in the interest of balancing the tradeoff. For example, if the attacker believes a user is likely to form a password by concatenating a dictionary word and then adding a number (which he increments each time he is required to change his password), then the guesses being executed may combine the word list and then append numbers (e.g., "mypassword2014" and then "mypassword2015"). Hybrids may also combine words in a brute force manner. Consider a requirement for a user to change his password every 90 days, he may form passwords like "mypasswordsummer" and then "mypasswordfall". The attacker then builds a hybrid attack which will take a dictionary word and then append other dictionary terms (potentially the same of different dictionaries) to make guesses.


THAT PASSWORD ISN'T VERY STRONG.
I'LL HAVE YOU KNOW IT'S ACTUALLY THE BEST.
I FOUND IT IN THIS ARTICLE, "MOST POPULAR PASSWORDS OF 2015"
BUTTFEED



Password cracking (4)

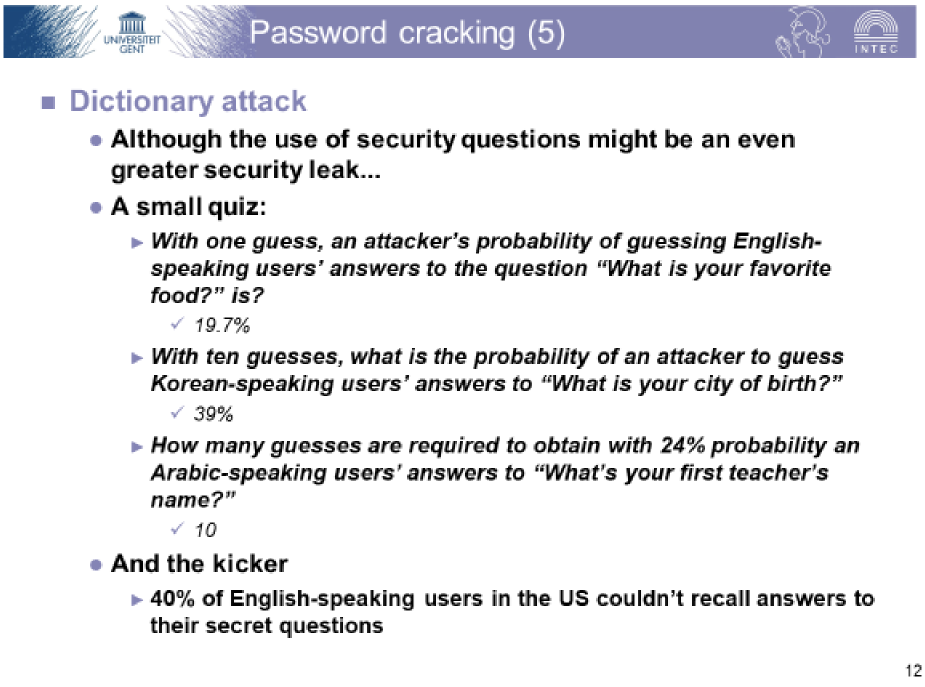
- Dictionary attack
  - Most popular passwords 2014

1. 123456	11. 1234567
2. password	12. monkey
3. 12345	13. letmein
4. 12345678	14. abc123
5. qwerty	15. 111111
6. 123456789	16.mustang
7. 1234	17. access
8. baseball	18. shadow
9. dragon	19. master
10.football	20. michael

11

From http://gizmodo.com/the-25-most-popular-passwords-of-2014-were-all-doomed-1680596951

Although the source does not mention which of these passwords are used for sensitive information, and which for forced website registrations…



Password cracking (5)

- Dictionary attack
  - Although the use of security questions might be an even greater security leak…
  - A small quiz:
    - *With one guess, an attacker's probability of guessing English-speaking users' answers to the question "What is your favorite food?" is?*
      - ✓ *19.7%*
    - *With ten guesses, what is the probability of an attacker to guess Korean-speaking users' answers to "What is your city of birth?"*
      - ✓ *39%*
    - *How many guesses are required to obtain with 24% probability an Arabic-speaking users' answers to "What's your first teacher's name?"*
      - ✓ *10*
  - And the kicker
    - 40% of English-speaking users in the US couldn't recall answers to their secret questions

12

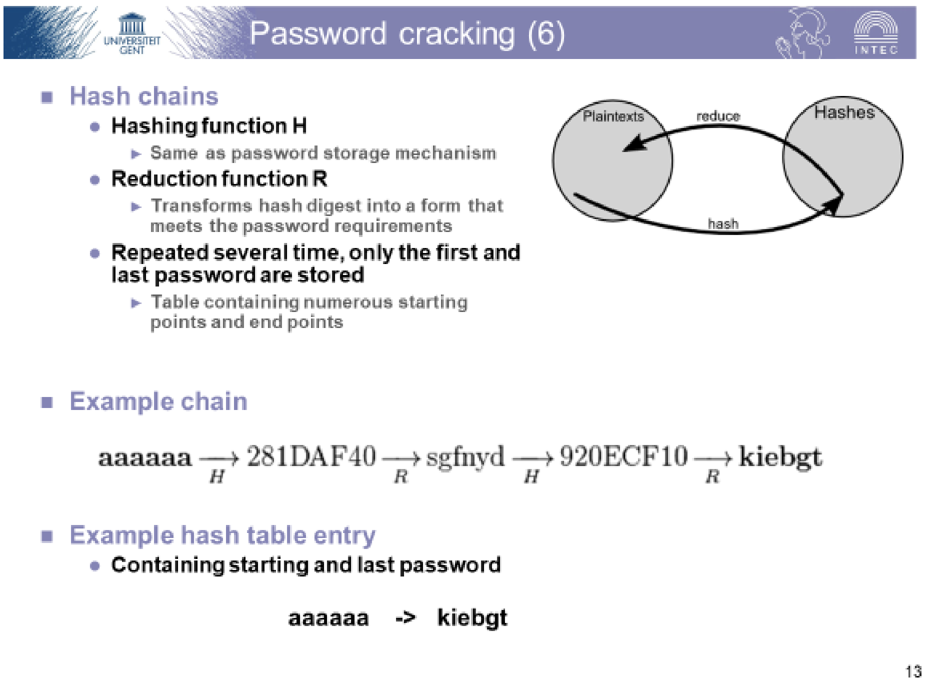
Many Secret Questions are less secure than the passwords they're intended to act as a fail-safe for because they don't operate with the same restrictions as passwords do:

- Can you use special characters?
- Is it case-sensitive?
- Is there a minimum of characters that must be used?
- Can you use at least 16 characters?
- Is there a limit on the number of unsuccessful attempts?

In many cases, the answer to these questions is "no." The last one in particular makes Secret Questions incredibly easy to brute-force; if someone's trying to reset your password, he or she oftentimes has an infinite number of attempts to answer your secret question before ultimately cracking it.

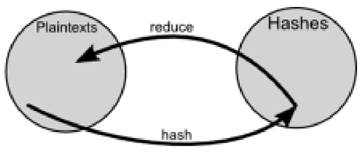
More examples at http://androidcommunity.com/google-analyzes-answers-to-common-security-questions-20150526/

Safer questions include: what is your frequent flyer number?, what is your library card number?, etc.



Password cracking (6)

- Hash chains
  - Hashing function H
    - Same as password storage mechanism
  - Reduction function R
    - Transforms hash digest into a form that meets the password requirements
  - Repeated several time, only the first and last password are stored
    - Table containing numerous starting points and end points



- Example chain

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

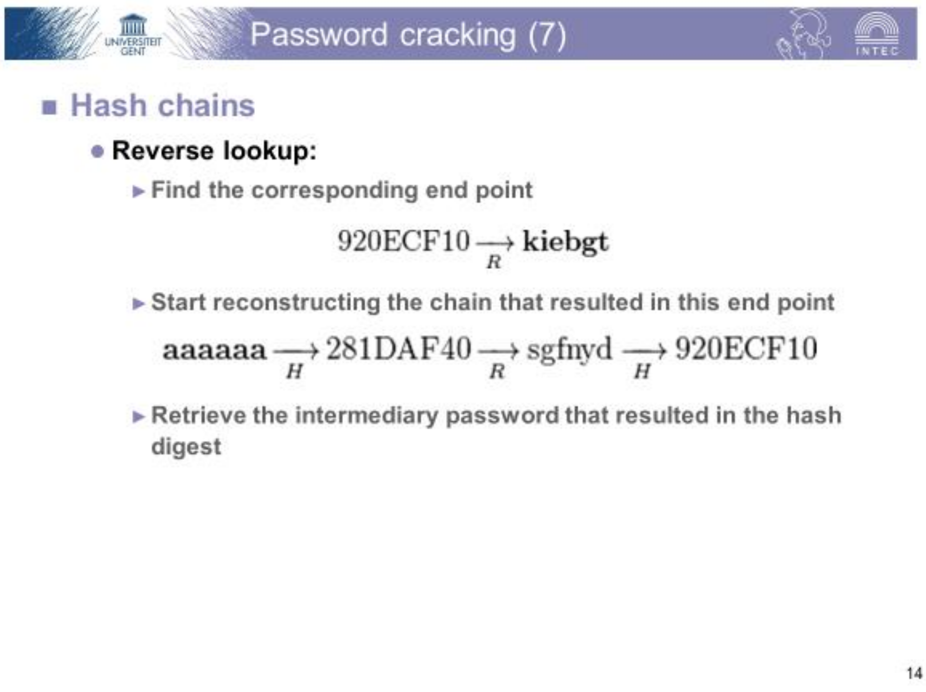
- Example hash table entry
  - Containing starting and last password

aaaaaa -> kiebgt

13

Hash tables are constructed by hashing each word in a password dictionary, or by hashing all possible passwords. The password-hash pairs are stored in a table, sorted by hash value. To use a hash table, simple take the hash and perform a binary search in the table to find the original password, if it's present. However, where hash tables require an infeasible amount of memory once the password size increases, a hash chain offers a compromise for the memory/time trade-off, using less computer processing time and more storage than a brute-force attack which calculates a hash on every attempt, but more processing time and less storage than a simple lookup table with one entry per hash.

A hash chain is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes. Constructing a hash chain requires two things: a hashing function and a reduction function. The hashing function must match the hashing function used by the system from which you want to recover a password. The reduction function transforms the obtained hash into something usable as a password. Note, however, that the reduction function is not actually an inverse of the hash function: the only requirement for the reduction function is to be able to return a "plain text" value in a specific size. A simple example reduction function is to Base64 encode the hash, then truncate it to a certain number of characters. By alternating the hash function with the reduction function, chains of alternating passwords and hash values are formed. To generate the table, we choose a random set of initial passwords from P, compute chains of some fixed length k for each one, and store only the first and last password in each chain. The first password is called the starting point and the last one is called the endpoint. None of the intermediary passwords or hash digests is stored.



Password cracking (7)

- Hash chains
  - **Reverse lookup:**
    - Find the corresponding end point

$$920\text{ECF}10 \xrightarrow[R]{} \textbf{kiebgt}$$

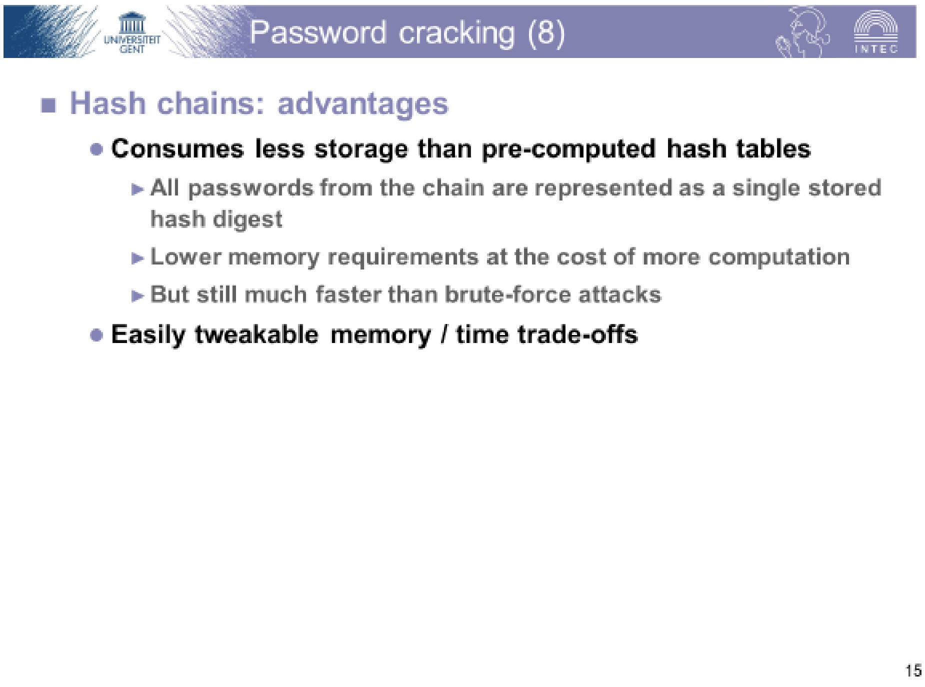
    - Start reconstructing the chain that resulted in this end point

$$\textbf{aaaaaa} \xrightarrow[H]{} 281\text{DAF}40 \xrightarrow[R]{} \text{sgfnyd} \xrightarrow[H]{} 920\text{ECF}10$$

    - Retrieve the intermediary password that resulted in the hash digest

14

Now, given a hash value h that we want to invert (find the corresponding password for), compute a chain starting with h by applying R, then H, then R, and so on. If at any point we observe a value matching one of the endpoints in the table, we get the corresponding starting point and use it to recreate the chain. There's a good chance that this chain will contain the value h, and if so, the immediately preceding value in the chain is the password p that we seek. 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. The password just before reaching this digest (“sgfnyd” in the example) is the one we are looking for.

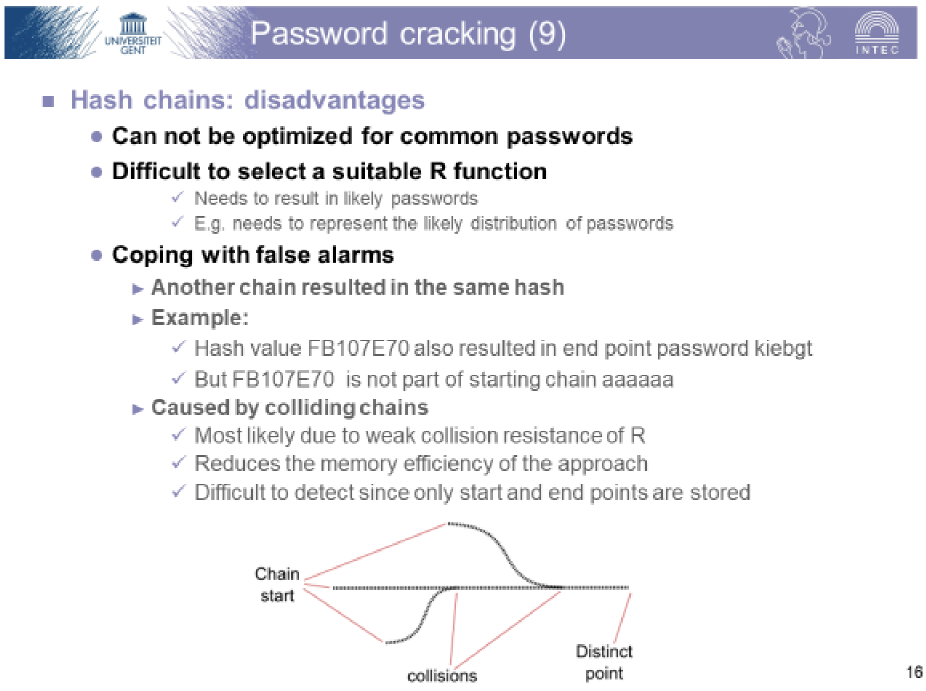


Password cracking (8)

- Hash chains: advantages
  - Consumes less storage than pre-computed hash tables
    - All passwords from the chain are represented as a single stored hash digest
    - Lower memory requirements at the cost of more computation
    - But still much faster than brute-force attacks
  - Easily tweakable memory / time trade-offs

15

The table content does not depend on the hash value to be inverted. It is created once and then repeatedly used for the lookups unmodified. The more links between the seed and the final value, the more passwords are captured. Increasing the length of the chain thereby decreasing the size of the table. However, it also increases the time required to perform lookups, and this is the time-memory trade-off of the table. In a simple case of one-item chains, the lookup is very fast, but the table is very big. Once chains get longer, the lookup slows down, but the table size goes down.



Password cracking (9)

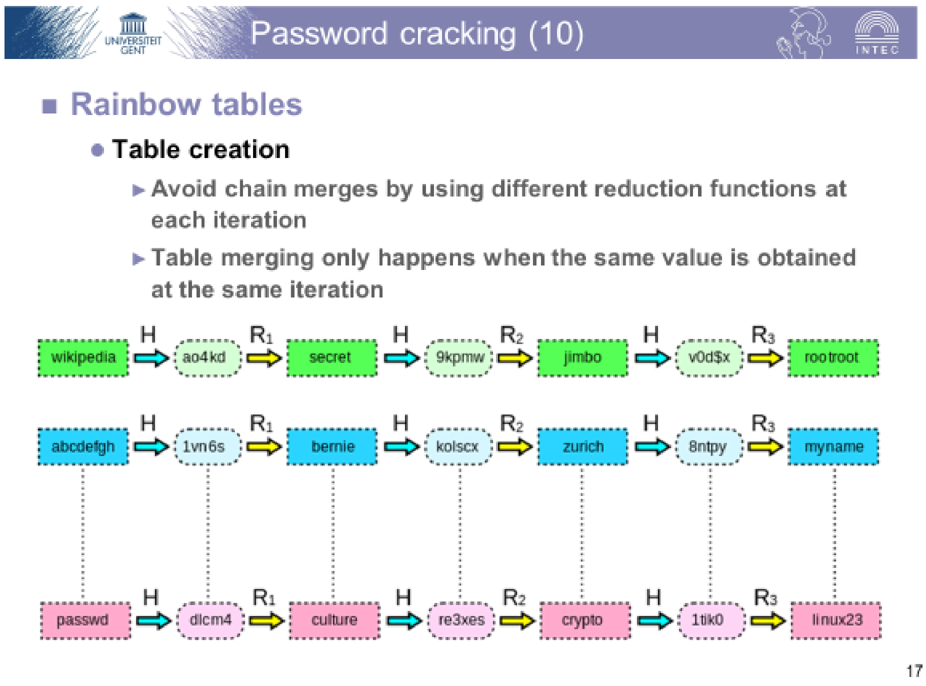
- Hash chains: disadvantages
  - Can not be optimized for common passwords
  - Difficult to select a suitable R function
    - ✓ Needs to result in likely passwords
    - ✓ E.g. needs to represent the likely distribution of passwords
  - Coping with false alarms
    - Another chain resulted in the same hash
    - Example:
      - ✓ Hash value FB107E70 also resulted in end point password kiebgt
      - ✓ But FB107E70 is not part of starting chain aaaaaa
    - Caused by colliding chains
      - ✓ Most likely due to weak collision resistance of R
      - ✓ Reduces the memory efficiency of the approach
      - ✓ Difficult to detect since only start and end points are stored

16

One first weakness is that the person building the chains doesn't choose the passwords they capture so the created tables can't be optimized for common passwords. Hash tables are good for common passwords (such as dictionary words), hash chains (or rainbow tables, see next slides) are good for tough passwords. The best approach is to recover as many passwords as possible using hash tables and/or conventional cracking with a dictionary of the top N passwords. For those that remain, use Rainbow Tables.

Also, it is important to choose a suitable function for R. Picking R to be the identity is little better than a brute force approach. Only when the attacker has a good idea of what the likely plaintexts will be he or she can choose a function R that makes sure time and space are only used for likely plaintexts, not the entire space of possible passwords. In effect R shepherds the results of prior hash calculations back to likely plaintexts but this benefit comes with drawback that R likely won't produce every possible plaintext in the class the attacker wishes to check denying certainty to the attacker that no passwords came from his chosen class. Also it can be difficult to design the function R to match the expected distribution of plaintexts.

Another problem is the occurrence of false alarms. The chain does not always contain the hash value h; it may so happen that the chain starting at h merges with the chain starting at the starting point at some point after h. For example, we may be given a hash value FB107E70, and when we follow its chain, we get the end point kiebgt. But FB107E70 is not in the chain starting at "aaaaaa". If at any point two chains collide (produce the same value), they will partly merge and consequently the table will not cover as many passwords despite having paid the same computational cost to generate. In this case, we ignore the match and continue to extend the chain of h looking for another match. If the chain of h gets extended to length k with no good matches, then the password was never produced in any of the chains. Because previous chains are not stored in their entirety, this is impossible to detect efficiently. For example, if the third value in chain 3 matches the second value in chain 7, the two chains will cover almost the same sequence of values, but their final values will not be the same. The hash function H is unlikely to produce collisions as it is usually considered an important security feature not to do so, but the reduction function R, because of its need to correctly cover the likely plaintexts, can not be collision resistant.

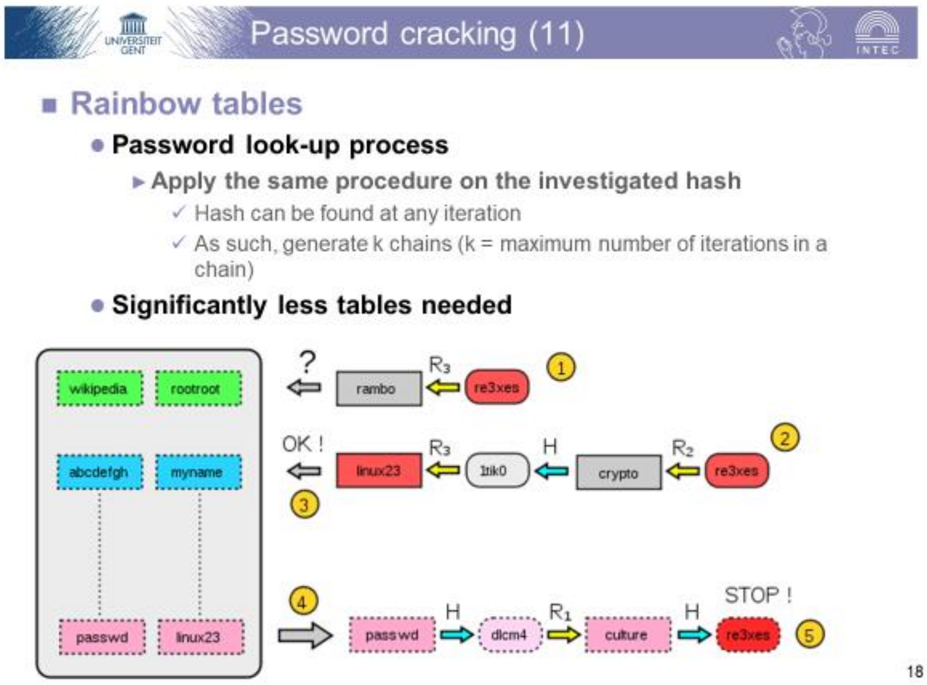


Password cracking (10)

- Rainbow tables
  - Table creation
    - Avoid chain merges by using different reduction functions at each iteration
    - Table merging only happens when the same value is obtained at the same iteration

UNIVERSITEIT GENT
INTEC
wikipedia H ao4kd R1 secret H 9kpmw R2 jimbo H v0d$x R3 rootroot
abcdefgh H 1vn6s R1 bernie H kolscx R2 zurich H 8ntpy R3 myname
passwd H dlcm4 R1 culture H re3xes R2 crypto H 1tik0 R3 linux23
17

Rainbow tables effectively solve the problem of collisions with ordinary hash chains by replacing the single reduction function R with a sequence of related reduction functions R1 through Rk. In this way, for two chains to collide and merge they must hit the same value on the same iteration. Consequently, the final values in each chain will be identical. A final postprocessing pass can sort the chains in the table and remove any "duplicate" chains that have the same final value as other chains. New chains are then generated to fill out the table. These chains are not collision-free (they may overlap briefly) but they will not merge, drastically reducing the overall number of collisions.



Password cracking (11)

- Rainbow tables
  - **Password look-up process**
    - **Apply the same procedure on the investigated hash**
      - ✓ Hash can be found at any iteration
      - ✓ As such, generate k chains (k = maximum number of iterations in a chain)
  - **Significantly less tables needed**

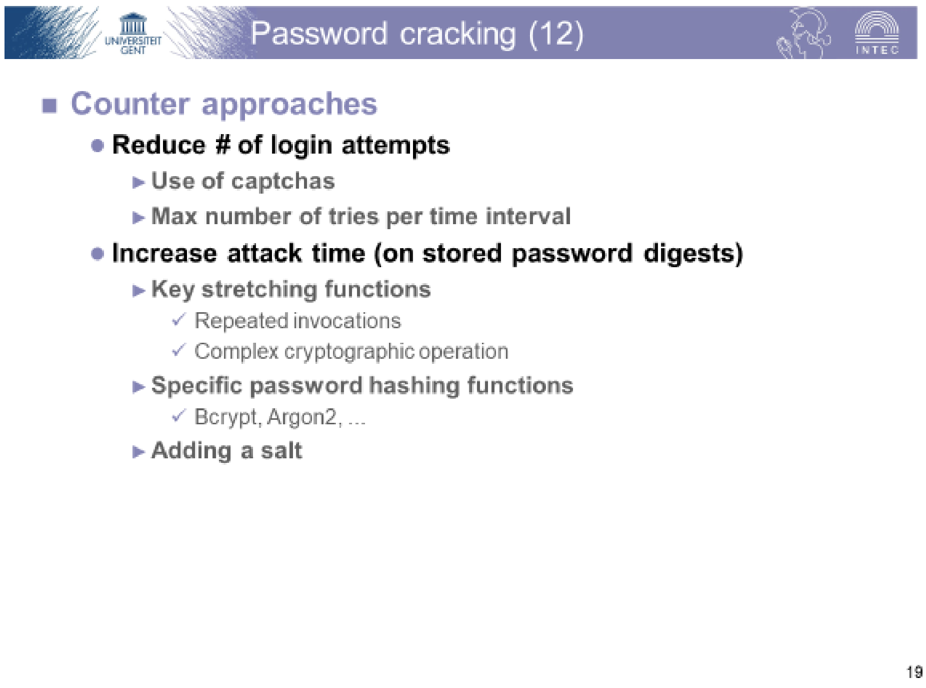
wikipedia
rootroot
abcdefgh
myname
passwd
linux23
?
rambo
R3
re3xes
1
OK !
linux23
R3
1tik0
H
crypto
R2
re3xes
2
3
4
passwd
H
dlcm4
R1
culture
H
STOP !
re3xes
5
18

Now, given a hash value h that we want to invert (find the corresponding password for), we again compute a chain starting with h by applying R, then H, then R, and so on. However, using sequences of reduction functions changes how the lookup procedure is performed: because the hash value of interest may be found at any location in the chain, it's necessary to generate *k* different chains. The first chain assumes the hash value is in the last hash position and just applies R_k; the next chain assumes the hash value is in the second-to-last hash position and applies R_{k-1}, then H, then R_k; and so on until the last chain, which applies all the reduction functions, alternating with H. This creates a new way of producing a false alarm: if we "guess" the position of the hash value wrong, we may needlessly evaluate a chain.

The look-up process is illustrated above

- Starting from the hash ("re3xes"), one computes the last reduction used in the table and checks whether the password appears in the last column of the table (step 1).
- If the test fails (rambo doesn't appear in the table), one computes a chain with the two last reductions (these two reductions are represented at step 2).
- If this new test fails again, one continues with 3 reductions, 4 reductions, etc. until the password is found. If no chain contains the password, then the attack has failed.
- However, if this test is positive (step 3, linux23 appears at the end of the chain and in the table), the password is retrieved at the beginning of the chain that produces linux23. Here we find passwd at the beginning of the corresponding chain stored in the table.
- At this point (step 4), one generates a chain and compares at each iteration the hash with the target hash. The test is valid and we find the hash re3xes in the chain. The current password (culture) is the one that produced the whole chain: the attack is successful.

Although rainbow tables have to follow more chains, they make up for this by having fewer tables: simple hash chain tables cannot grow beyond a certain size without rapidly becoming inefficient due to merging chains. To deal with this, they maintain multiple tables, and each lookup must search through each table. Rainbow tables can achieve similar performance with tables that are k times larger, allowing them to perform a factor of k fewer lookups. Other advantages and disadvantages of this approach are the same ones as for the hash chains. Although creating a rainbow table is computationally expensive, the table can be reused during a long period, making it increasingly beneficial the longer the table is used.



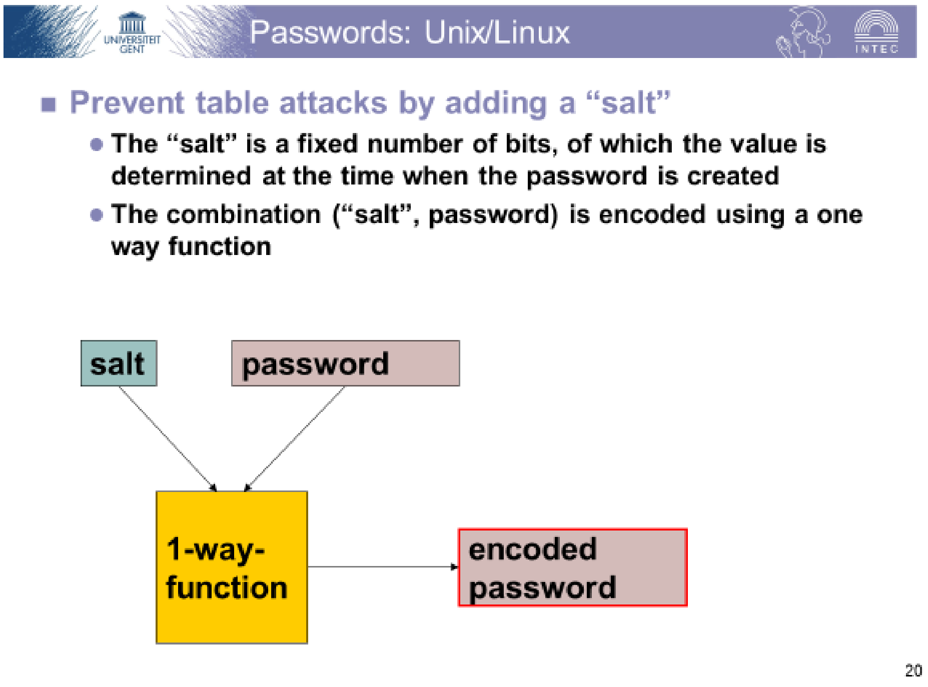
Password cracking (12)

- Counter approaches
  - **Reduce # of login attempts**
    - Use of captchas
    - Max number of tries per time interval
  - **Increase attack time (on stored password digests)**
    - Key stretching functions
      - ✓ Repeated invocations
      - ✓ Complex cryptographic operation
    - Specific password hashing functions
      - ✓ Bcrypt, Argon2, …
    - Adding a salt

19

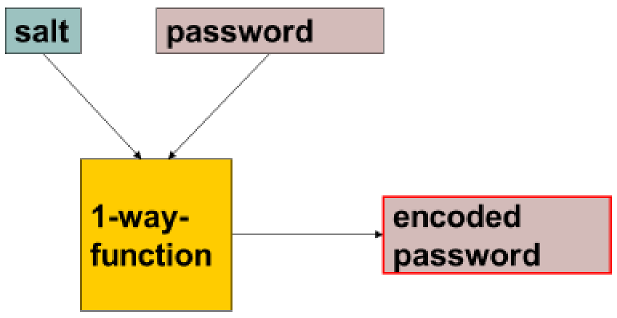
Key stretching functions, such as PBKDF2, Bcrypt or Scrypt, typically use repeated invocations of a cryptographic hash to increase the time required to perform brute force attacks on stored password digests.

A suitable password hashing function, such as bcrypt, is many orders of magnitude better than a naive function like simple MD5 or SHA. In 2013 a "Password Hashing Competition" was announced to choose a new, standard algorithm for password hashing to avoid a repeat of previous password breaches involving weak or no hashing, such as the ones involving RockYou (2009), JIRA (2010), Gawker (2010), PlayStation Network outage (2011), EHarmony (2012), 2012 LinkedIn hack, Battlefield Heroes (2011), Adobe (2012), Evernote 2013, ASUS (2012), South Carolina Department of Revenue (2012), Ubuntu Forums (2013), etc. An open competition was organized to select one or more password hash functions that can be recognized as a recommended standard. In the wake of allegations that NSA forced NIST to standardize a backdoored algorithm (Dual EC DRBG), the competition was being run by an independent panel of cryptographers and security practitioners independent of NIST, in order to avoid even the appearance of a backdoored algorithm. On 20th July 2015 it selected Argon2 as a basis for the final PHC winner and gave special recognition to four other schemes: Catena, Lyra2, yescrypt and Makwa.



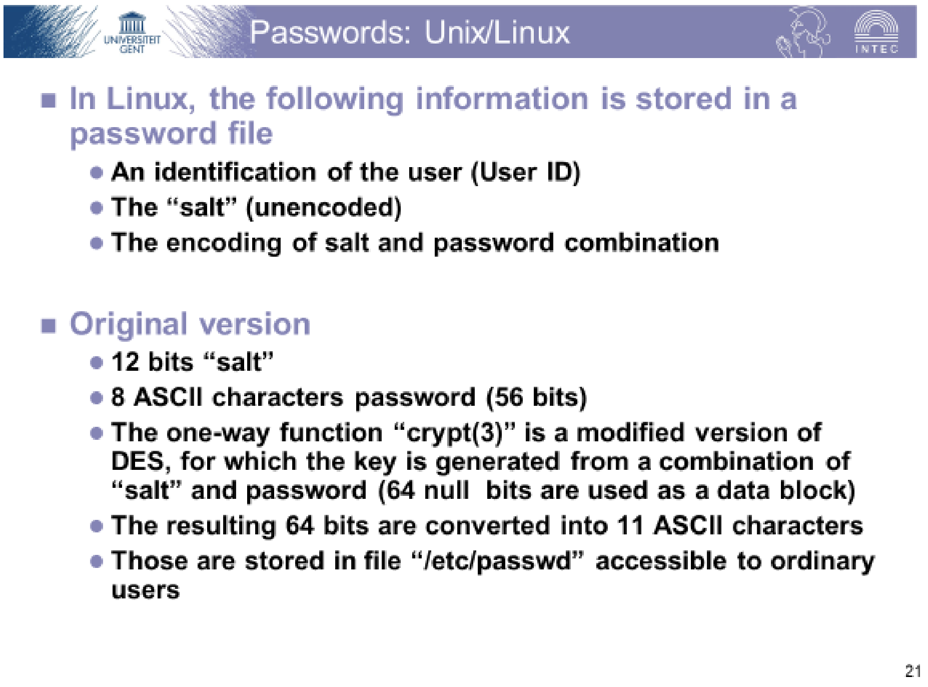
Passwords: Unix/Linux

- **Prevent table attacks by adding a "salt"**
  - **The "salt" is a fixed number of bits, of which the value is determined at the time when the password is created**
  - **The combination ("salt", password) is encoded using a one way function**



20

The goal of this "salt" is that possibly identical passwords result in different encoded passwords, so that passwords must be tested individually. This is important, not only for systems with many users, but also for widely used systems. Without a salt the password storage system becomes vulnerable to techniques using pre-computed encoded passwords (e.g. rainbow tables) and birthday attacks. A rainbow table is ineffective against one-way hashes that include large salts.



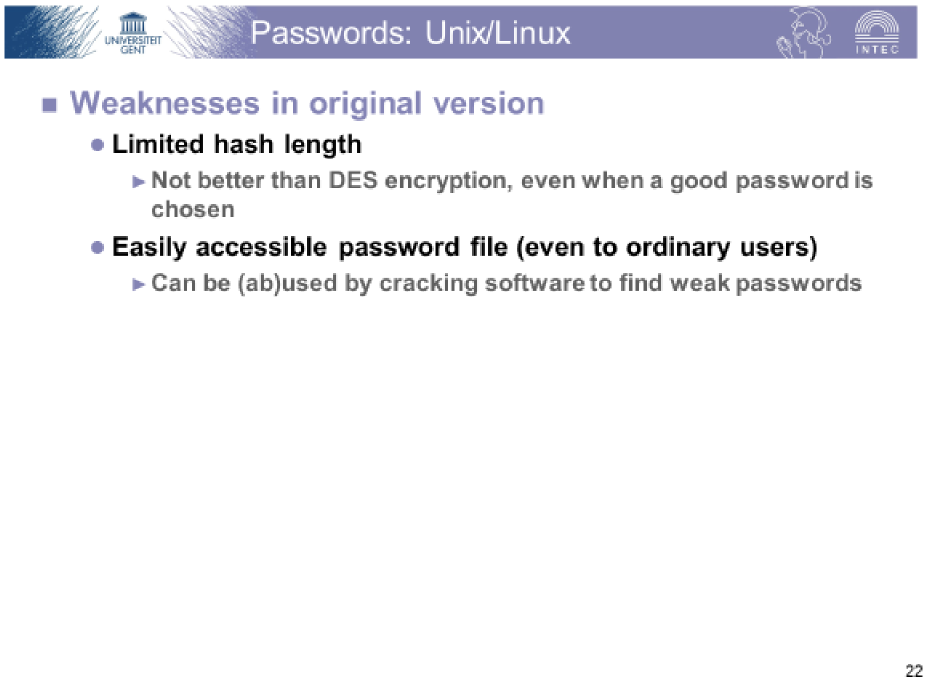
Passwords: Unix/Linux

- **In Linux, the following information is stored in a password file**
  - **An identification of the user (User ID)**
  - **The "salt" (unencoded)**
  - **The encoding of salt and password combination**
- **Original version**
  - **12 bits "salt"**
  - **8 ASCII characters password (56 bits)**
  - **The one-way function "crypt(3)" is a modified version of DES, for which the key is generated from a combination of "salt" and password (64 null bits are used as a data block)**
  - **The resulting 64 bits are converted into 11 ASCII characters**
  - **Those are stored in file "/etc/passwd" accessible to ordinary users**

21

The modification to the DES algorithm makes it impossible to use (faster) hardware DES implementations. The (modified) DES encryption is applied 25 times consecutively. This doesn't fundamentally improve the security of DES, but it slows down the algorithm compared to classical DES, making the hacker's job somewhat harder.

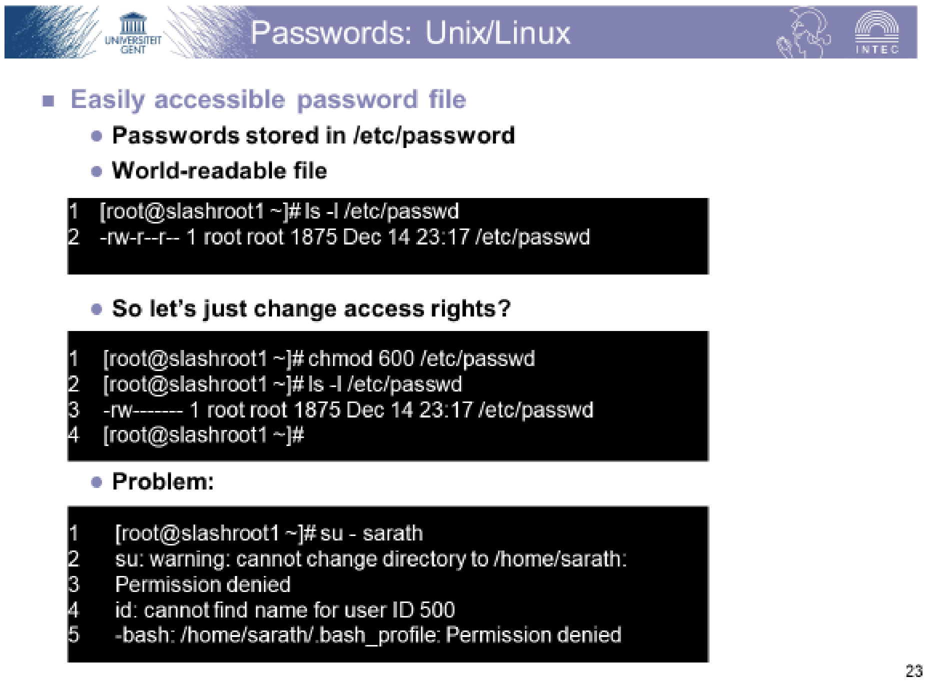
Although a salt is used, this does not result in a large security improvement as it can be found unencoded in the password file.



Passwords: Unix/Linux

- Weaknesses in original version
  - **Limited hash length**
    - Not better than DES encryption, even when a good password is chosen
  - **Easily accessible password file (even to ordinary users)**
    - Can be (ab)used by cracking software to find weak passwords

22



Passwords: Unix/Linux

- Easily accessible password file
  - **Passwords stored in /etc/password**
  - **World-readable file**

```
[root@slashroot1 ~]# ls -l /etc/passwd
-rw-r--r-- 1 root root 1875 Dec 14 23:17 /etc/passwd
```

  - **So let's just change access rights?**

```
[root@slashroot1 ~]# chmod 600 /etc/passwd
[root@slashroot1 ~]# ls -l /etc/passwd
-rw------- 1 root root 1875 Dec 14 23:17 /etc/passwd
[root@slashroot1 ~]#
```

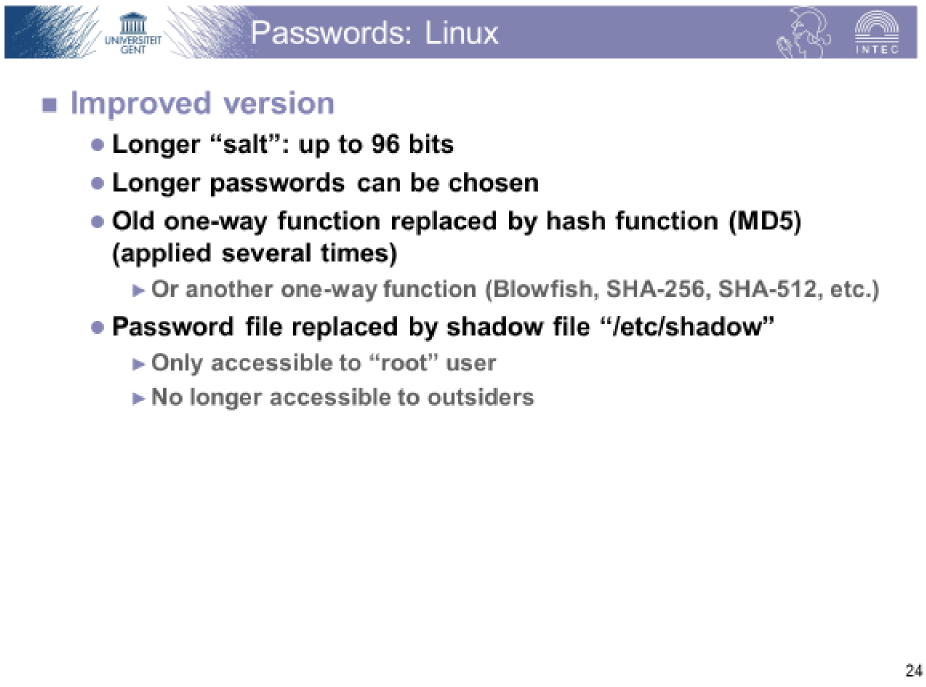
  - **Problem:**

```
[root@slashroot1 ~]# su - sarath
su: warning: cannot change directory to /home/sarath:
Permission denied
id: cannot find name for user ID 500
-bash: /home/sarath/.bash_profile: Permission denied
```

23

You can see that there is an "r" (Which stands for read), in all the three fields for that file. Let's try to change this permission, so that only "root" can read the file. Then let's see what's the effect of this change on the system...Let's try to become a normal user, and see what happens.

You can't even change the user. Even if you change the user, you will not get the user name, home directory, custom shell, etc. (because all these details are stored in/etc/passwd). Due to this reason the file /etc/passwd, needs to be kept world readable. But we cannot keep passwords in a file that's world readable (because of the risk involved, even though its encoded in a one way hash algorithm). Hence there arises a need to separate passwords from this file and keep it in a file, that's only accessible by root. The solution to this problem is implemented in the form of a package in Linux called "shadow-utils".

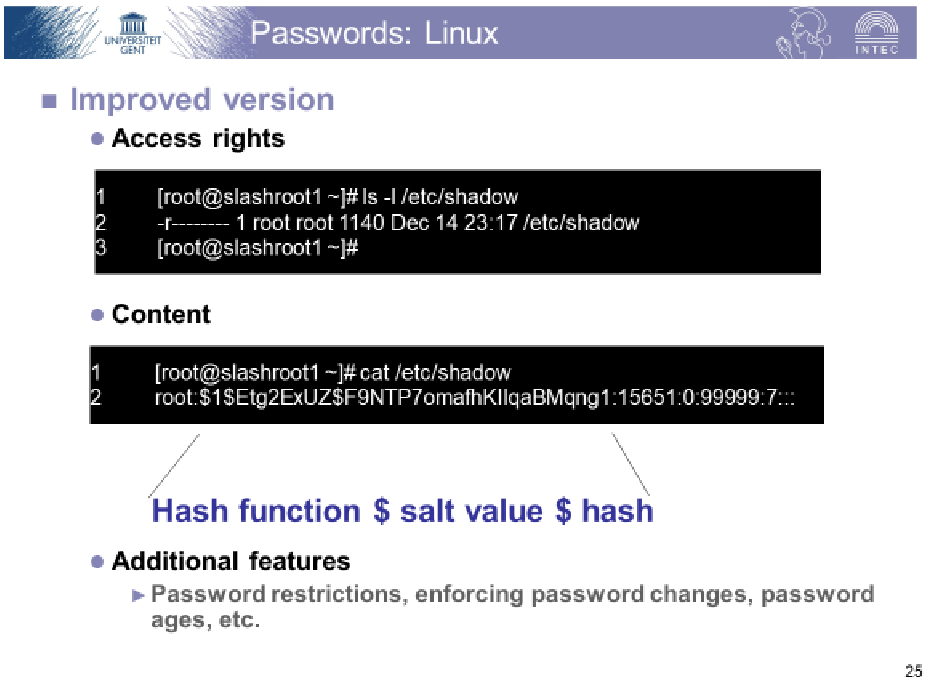


Passwords: Linux

- Improved version
  - Longer "salt": up to 96 bits
  - Longer passwords can be chosen
  - Old one-way function replaced by hash function (MD5) (applied several times)
    - Or another one-way function (Blowfish, SHA-256, SHA-512, etc.)
  - Password file replaced by shadow file "/etc/shadow"
    - Only accessible to "root" user
    - No longer accessible to outsiders

24

The much longer "salt" makes the system also suitable for large numbers of users. The most important drawback of a freely accessible password file is that almost anyone could use this file to test passwords "off-line" using existing cracking software (such as "John The Ripper", "Crack", or rainbow tables). The advantage of a password file inaccessible to ordinary users is that the only way for an attacker to test passwords is attempting to log in (through SSH or FTP), or obtaining access to the system by some other means (e.g. through a security breach or physical access to the system). This is orders of magnitude less practical than running cracking software locally. Notice the system administrator still has access to the password file, meaning an internal attack using cracking software is still possible.



Passwords: Linux

- Improved version
  - Access rights

```
[root@slashroot1 ~]# ls -l /etc/shadow
-r-------- 1 root root 1140 Dec 14 23:17 /etc/shadow
[root@slashroot1 ~]#
```

  - Content

```
[root@slashroot1 ~]# cat /etc/shadow
root:$1$Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:15651:0:99999:7:::
```

Hash function $ salt value $ hash

  - Additional features
    - Password restrictions, enforcing password changes, password ages, etc.

25

The file itself contains several fields, separated by :

1. The first field is self explanatory, its the USERNAME

2. The second field is the encoded password describing the hashing algorithm, the salt value and the one way hash digest, all separated by dollar signs. The first entry describes the hashing algorithm:

$1 = MD5 hashing algorithm.

$2 =Blowfish Algorithm is in use.

$2a=eksblowfish Algorithm

$5 =SHA-256 Algorithm

$6 =SHA-512 Algorithm

3. The third field is the day's since the UNIX time that password was changed.

4. This field specifies the number of days, that are required between password changes.

5.No of days after which it is necessary to change the password.

6.This is the number of days before the required password change that the user gets a warning

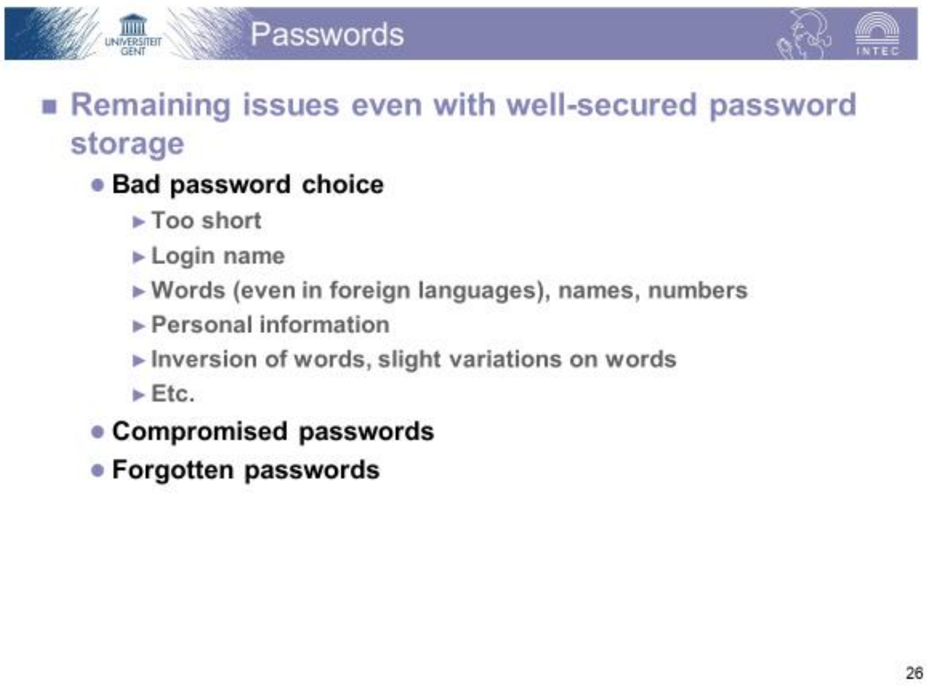
7.If the password has expired, after this number of days the account will be disabled

8.No of days from the Unix Time that the account is disabled

9. This field is not used yet...

Additional fields are available because shadow-util's package provides more advanced feature's along with storing encoded passwords in /etc/shadow. The above mentioned fields of /etc/shadow, file allows to check and enforce e.g.

- The age of the passwords and its expiry
- Default parameters for user account creation (/etc/login.defs)
- Tools to modify user accounts and groups
- Enforcing strict password selection
- ...

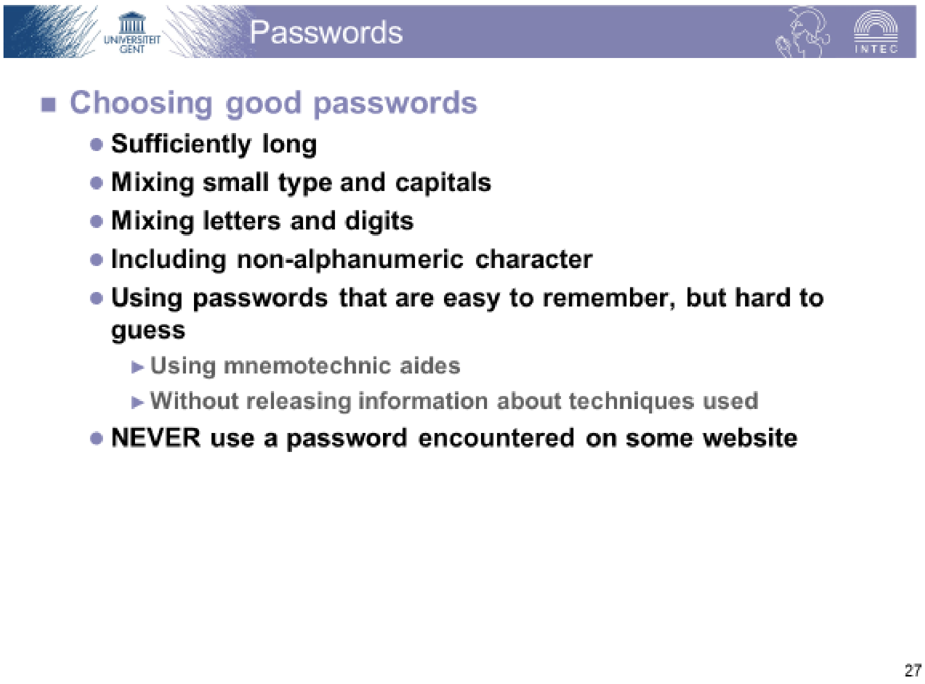


Passwords

- Remaining issues even with well-secured password storage
  - **Bad password choice**
    - Too short
    - Login name
    - Words (even in foreign languages), names, numbers
    - Personal information
    - Inversion of words, slight variations on words
    - Etc.
  - **Compromised passwords**
  - **Forgotten passwords**

26

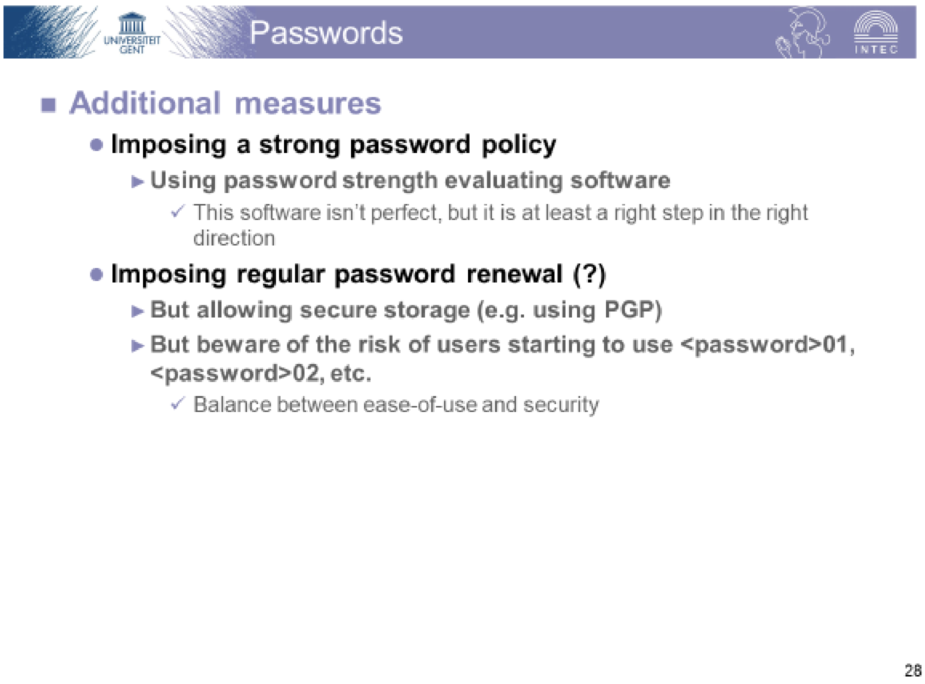
Making it possible to use passwords of 15 characters and more doesn't mean that users will grasp the opportunity of choosing long passwords. Passwords of 2 or 3 characters also occur: the eternal opposition between ease of use and security. Furthermore, a password can be compromised if it is used in an ill-secured or insecure context. It is therefore recommended to use different passwords for different categories of applications. For passwords granting system administrator access, it is even recommended to use a different password for each application.



Passwords

- Choosing good passwords
  - Sufficiently long
  - Mixing small type and capitals
  - Mixing letters and digits
  - Including non-alphanumeric character
  - Using passwords that are easy to remember, but hard to guess
    - Using mnemotechnic aides
    - Without releasing information about techniques used
  - NEVER use a password encountered on some website

27

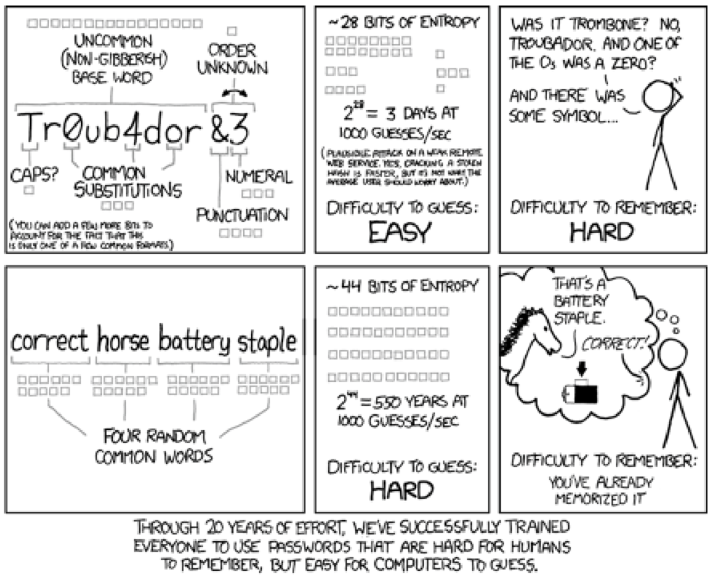


Passwords

- Additional measures
  - Imposing a strong password policy
    - Using password strength evaluating software
      - ✓ This software isn't perfect, but it is at least a right step in the right direction
  - Imposing regular password renewal (?)
    - But allowing secure storage (e.g. using PGP)
    - But beware of the risk of users starting to use <password>01, <password>02, etc.
      - ✓ Balance between ease-of-use and security

28

Passwords

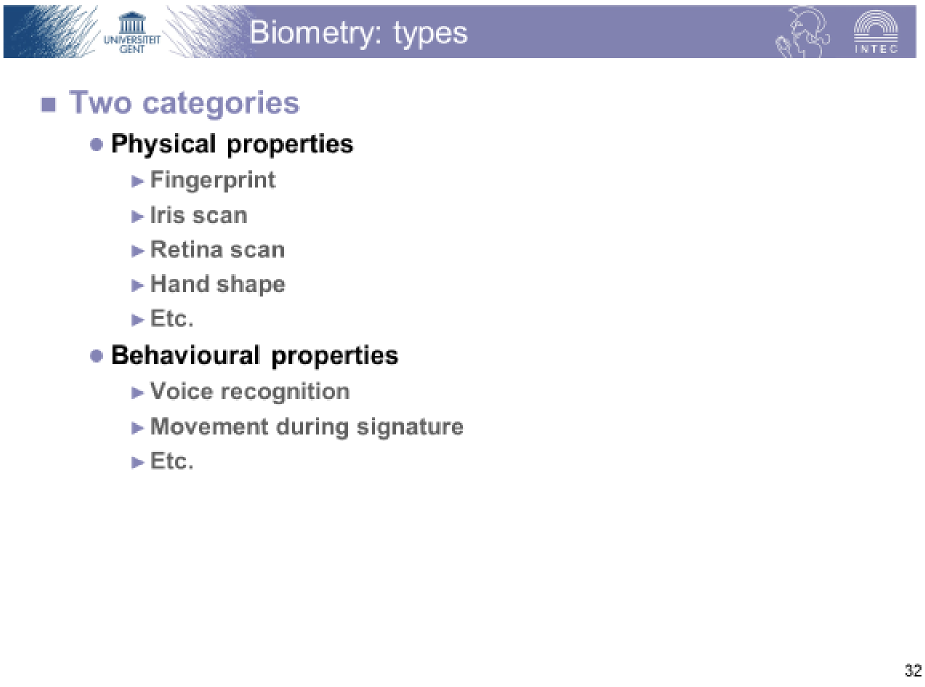

UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

29

Overview

- Secure systems
  - **Authentication methods**
    - Passwords
    - Biometry
    - Security Tokens
  - Trusted OS
  - Disk encryption

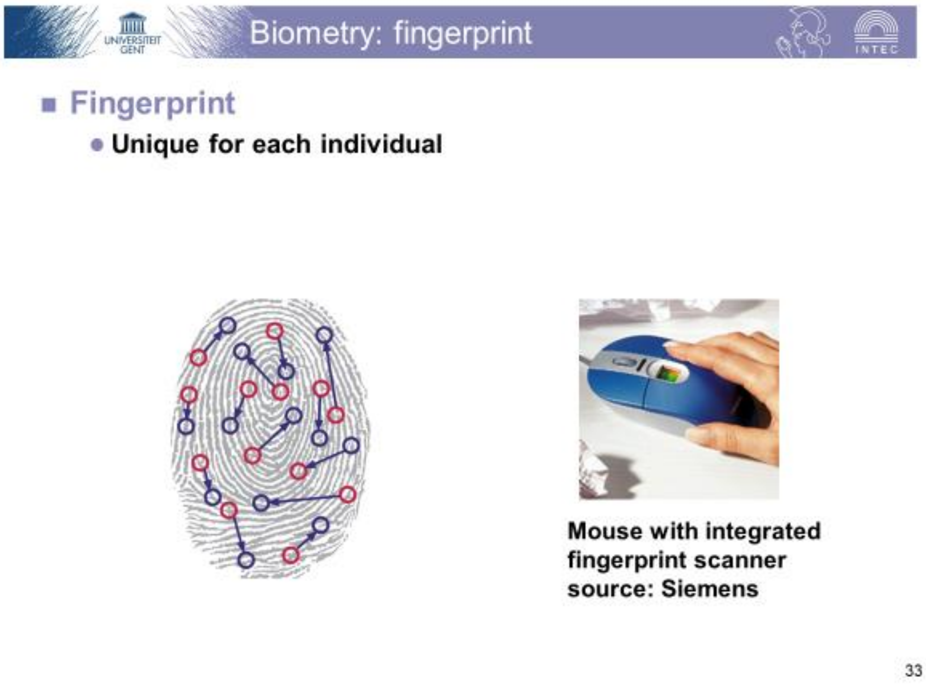
30



Biometry: types

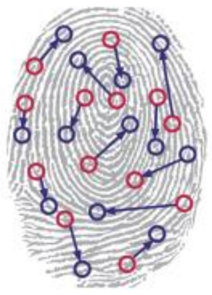
- Two categories
  - Physical properties
    - Fingerprint
    - Iris scan
    - Retina scan
    - Hand shape
    - Etc.
  - Behavioural properties
    - Voice recognition
    - Movement during signature
    - Etc.

32



Biometry: fingerprint

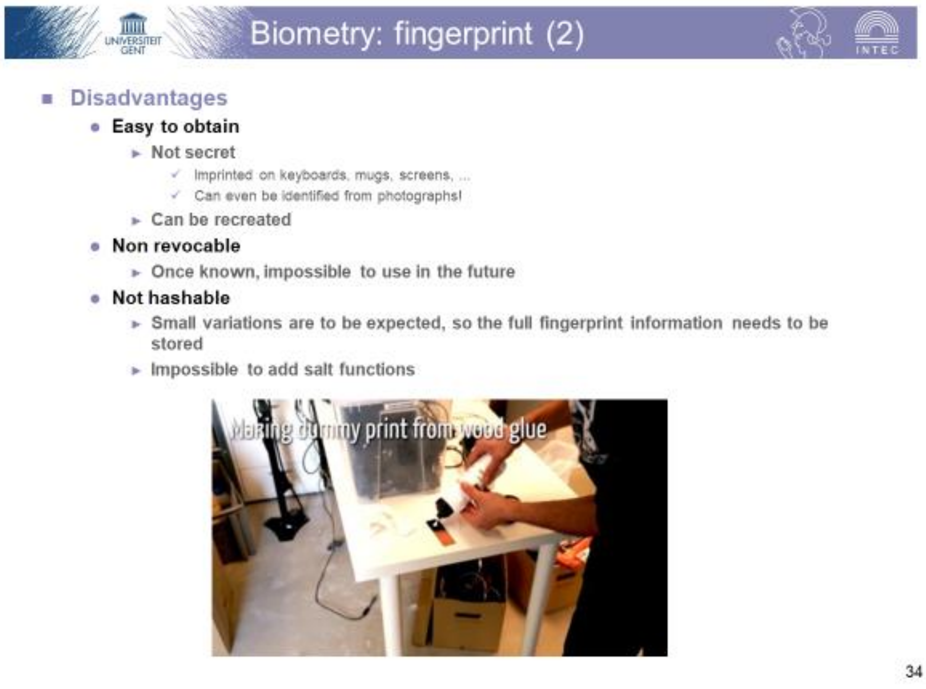
- Fingerprint
  - Unique for each individual



Mouse with integrated fingerprint scanner
source: Siemens

33

The fine details of ridges, valleys, and swirls that define our fingerprints are influenced by random stresses experienced in the womb. Even a slightly different umbilical cord length changes your fingerprint. As such, even identical twins have different prints.



Biometry: fingerprint (2)

- Disadvantages
  - **Easy to obtain**
    - Not secret
      - ✓ Imprinted on keyboards, mugs, screens, ...
      - ✓ Can even be identified from photographs!
    - Can be recreated
  - **Non revocable**
    - Once known, impossible to use in the future
  - **Not hashable**
    - Small variations are to be expected, so the full fingerprint information needs to be stored
    - Impossible to add salt functions

34

Obtaining and reproducing fingerprints poses no challenge. An example technique of fingerprint recreation etches a copy of a fingerprint into copper (as if making a PCB), coating the etching in graphite spray, and finally topping it all off with a layer of wood glue or latex. Where the copper is etched away, the glue-and-graphite finger mold is deeper, simulating the ridges on your finger. The graphite spray is used to give the model the right bulk capacitance. As a result, back when the iPhone 5's touchID system was just announced, it was hacked after only 2 days. A video describing the methods used can be found at http://arstechnica.com/security/2013/09/touchid-hack-was-no-challenge-at-all-hacker-tells-ars/

Biometry: iris

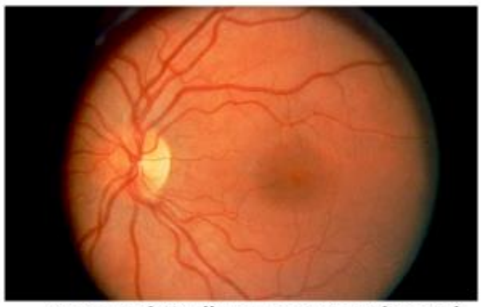
- Iris around pupil
  - Exhibits complex pattern, unique for each individual
    - Isn't altered by time

source: CNN 2002-03-27

35

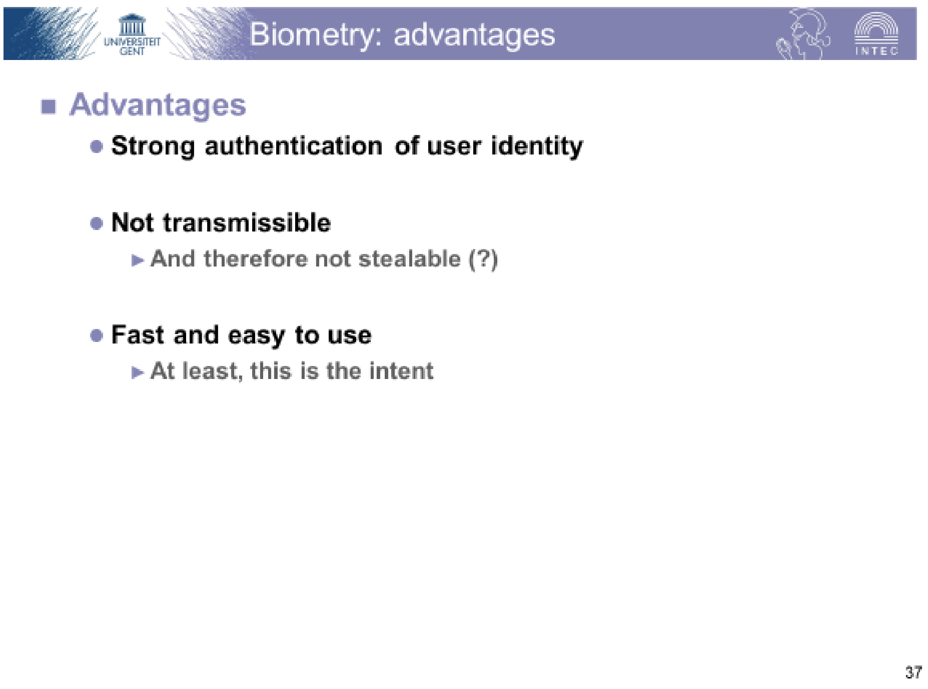
Biometry: retina

- Retina
  - Unique pattern of blood vessels
    - May however be altered by some ocular affections



source: http://www.eyesearch.com/

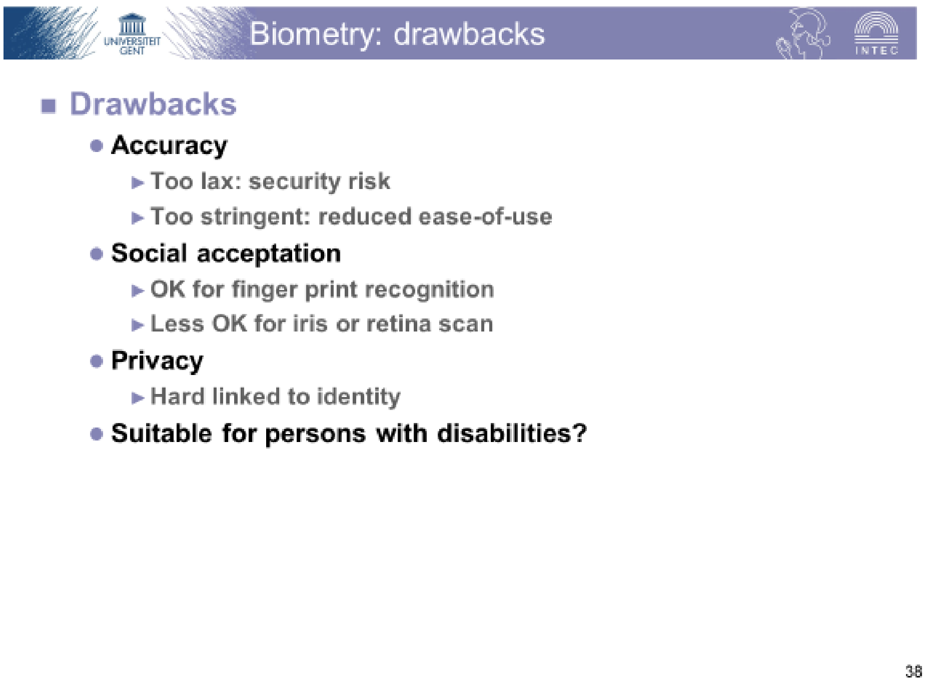
36



Biometry: advantages

- Advantages
  - Strong authentication of user identity
  - Not transmissible
    - And therefore not stealable (?)
  - Fast and easy to use
    - At least, this is the intent

37

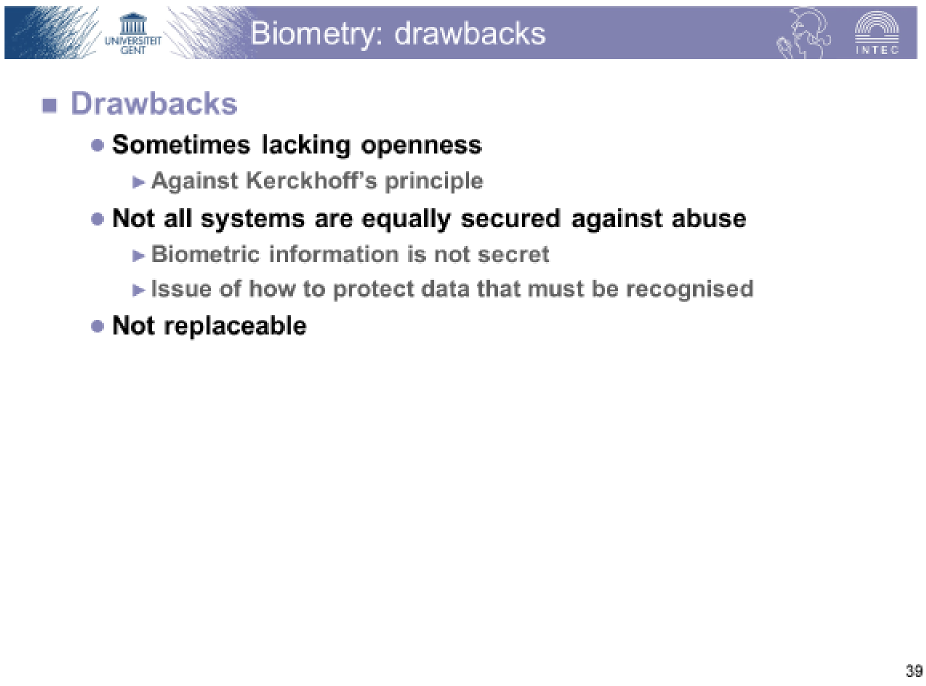


Biometry: drawbacks

- Drawbacks
  - Accuracy
    - Too lax: security risk
    - Too stringent: reduced ease-of-use
  - Social acceptation
    - OK for finger print recognition
    - Less OK for iris or retina scan
  - Privacy
    - Hard linked to identity
  - Suitable for persons with disabilities?

38

Biometric characteristics such as a fingerprint are never measured twice in a row in exactly the same way (orientation of a finger may differ, etc.). This means that some error tolerance must be accepted for the recognition function. If the error tolerance is too large, users will wrongly obtain access (which implies a security risk). If the error tolerance is too small, legitimate users will wrongly be denied access, which can generate non-negligible irritation if this happens too often. These accuracy issues often strongly depend on the method: iris recognition is much less problematic than fingerprint recognition. The larger the number of legitimate users, the larger the risk of wrongly recognising a non-legitimate user.

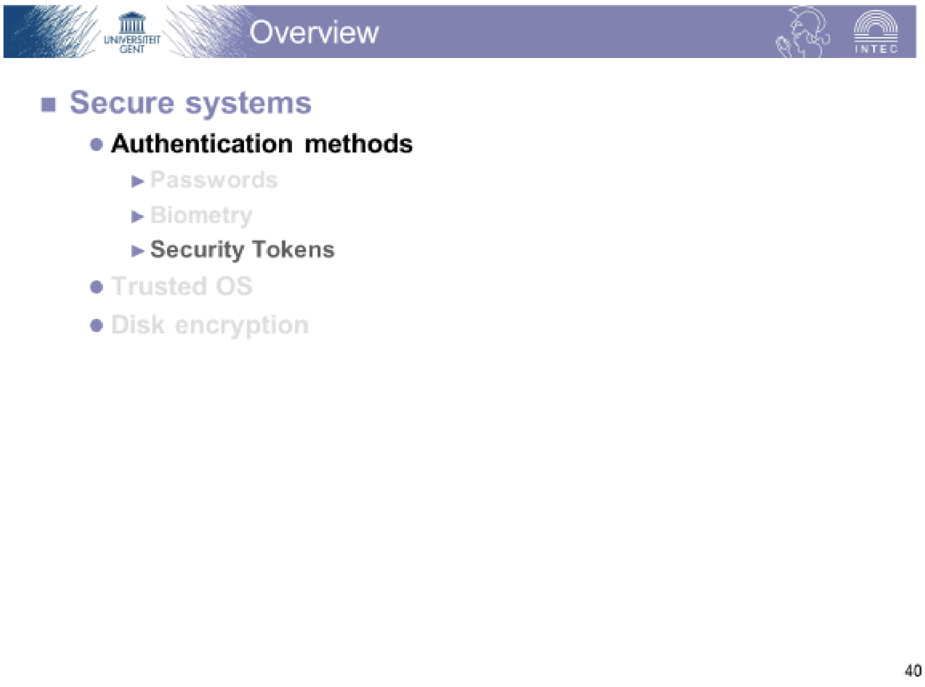


Biometry: drawbacks

- Drawbacks
  - Sometimes lacking openness
    - Against Kerckhoff's principle
  - Not all systems are equally secured against abuse
    - Biometric information is not secret
    - Issue of how to protect data that must be recognised
  - Not replaceable

39

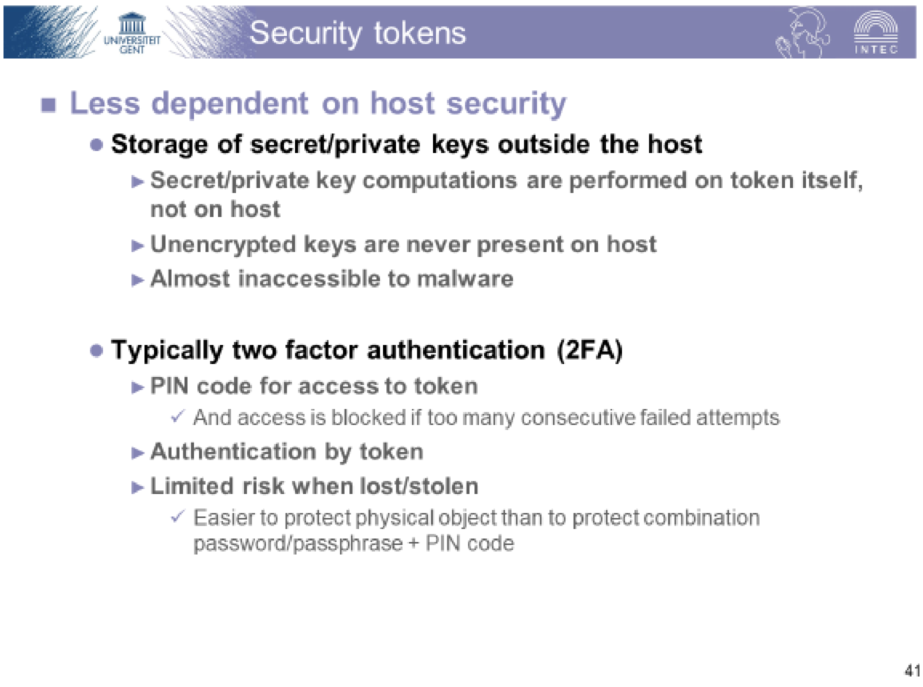
Biometric systems are often much less unbreakable than producers claim (fingerprints can be forged). Simply replacing the login/password system isn't obvious. A normal password must be stored **encoded** on the system and at login the input password in encoded and compared to the stored encoded password. Because of the error margin which must be accepted for biometric data, a comparable approach isn't possible. This means we need special security for the stored biometric data. Physical protection of the input device is necessary, so that the recognition can take place on the input device itself (and not on the PC).



Overview

- Secure systems
  - Authentication methods
    - Passwords
    - Biometry
    - Security Tokens
  - Trusted OS
  - Disk encryption

40



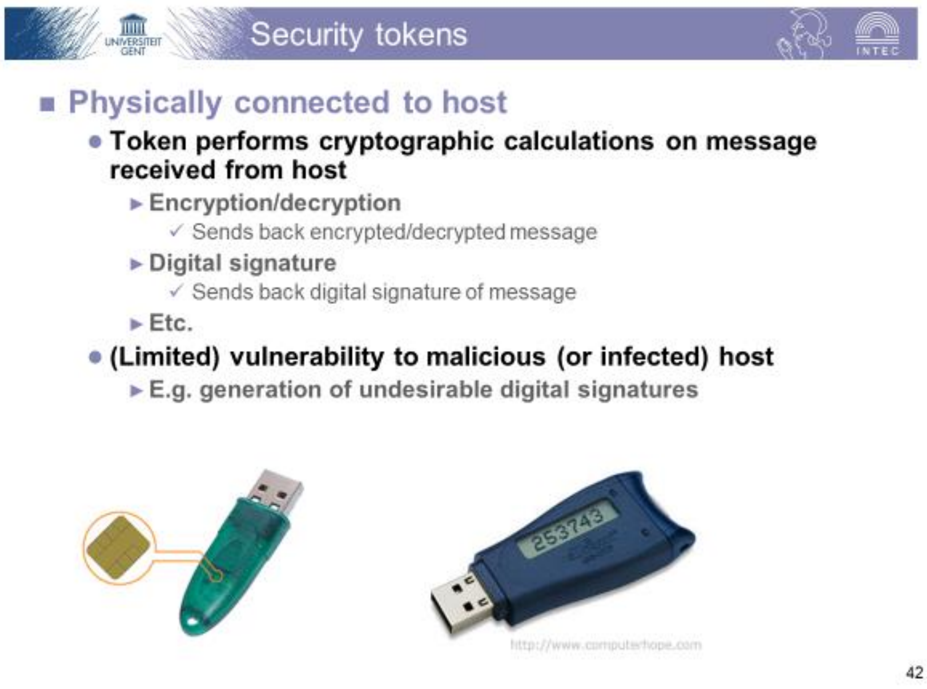
Security tokens

- Less dependent on host security
  - Storage of secret/private keys outside the host
    - Secret/private key computations are performed on token itself, not on host
    - Unencrypted keys are never present on host
    - Almost inaccessible to malware
  - Typically two factor authentication (2FA)
    - PIN code for access to token
      - ✓ And access is blocked if too many consecutive failed attempts
    - Authentication by token
    - Limited risk when lost/stolen
      - ✓ Easier to protect physical object than to protect combination password/passphrase + PIN code

41

Even when using secure storage solutions, at some times the secret or private key must be available: during the symmetric encryption or decryption process, during the asymmetric decryption process, when applying a digital signature, when computing a message authentication code, etc. At these moments, the unencrypted key will be present in the memory of the end system. Well-designed security programs will take care that this part of memory cannot be swapped to disk by the operating system ("locking" in memory) and that the memory used will be erased before being released, so that another program cannot misuse the memory contents to retrieve confidential information. However, sufficiently clever malware will still be able to bypass these security measures, e.g. by logging the password that was input at the keyboard.

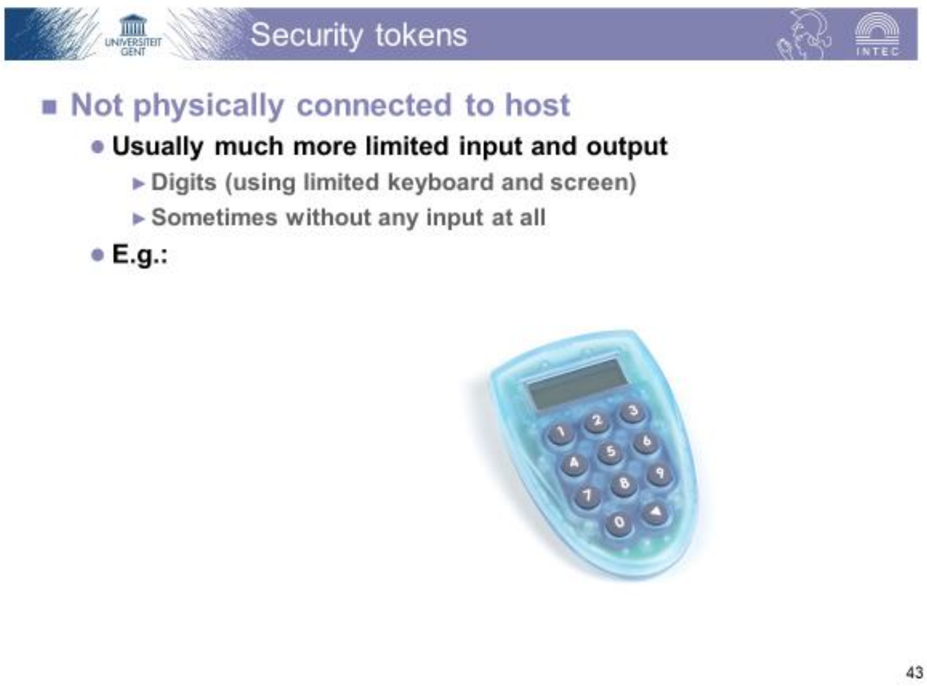
Two factor authentication enhances the provided security by requiring multiple access methods of different types. In the case of security tokens, access is only granted based on what you know (e.g. the passphrase) and what you own (the token). Other two factor authentication methods use e.g. other combinations of information you know (passsword, PIN, etc.), possess (SIM card, token, etc,) or characteristics you exhibit (voice, iris patterns, etc.)



Security tokens

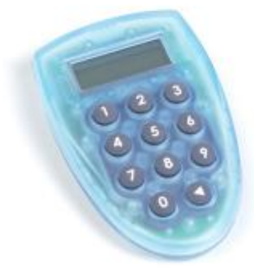
- Physically connected to host
  - Token performs cryptographic calculations on message received from host
    - Encryption/decryption
      - ✓ Sends back encrypted/decrypted message
    - Digital signature
      - ✓ Sends back digital signature of message
    - Etc.
  - (Limited) vulnerability to malicious (or infected) host
    - E.g. generation of undesirable digital signatures

42

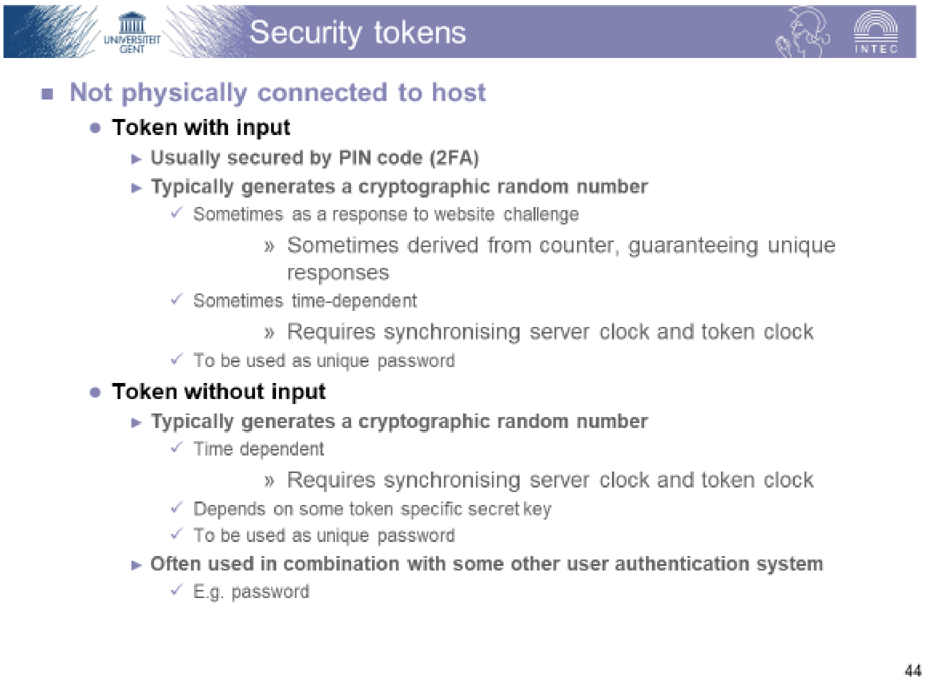


Security tokens

- Not physically connected to host
  - Usually much more limited input and output
    - Digits (using limited keyboard and screen)
    - Sometimes without any input at all
  - E.g.:



43

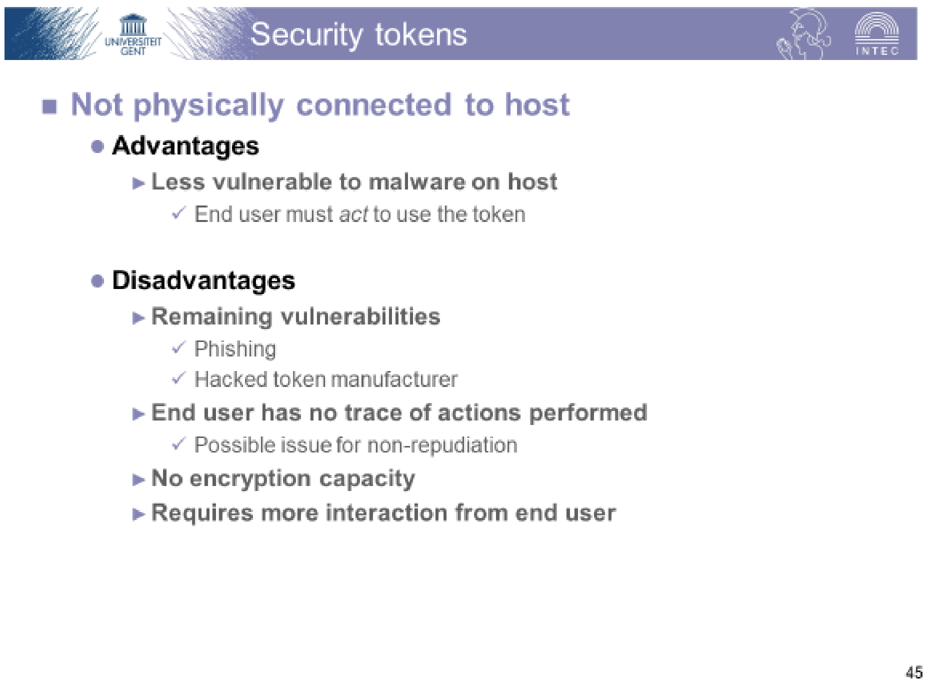


Security tokens

- Not physically connected to host
  - Token with input
    - Usually secured by PIN code (2FA)
    - Typically generates a cryptographic random number
      - Sometimes as a response to website challenge
        - Sometimes derived from counter, guaranteeing unique responses
      - Sometimes time-dependent
        - Requires synchronising server clock and token clock
      - To be used as unique password
  - Token without input
    - Typically generates a cryptographic random number
      - Time dependent
        - Requires synchronising server clock and token clock
      - Depends on some token specific secret key
      - To be used as unique password
    - Often used in combination with some other user authentication system
      - E.g. password

44

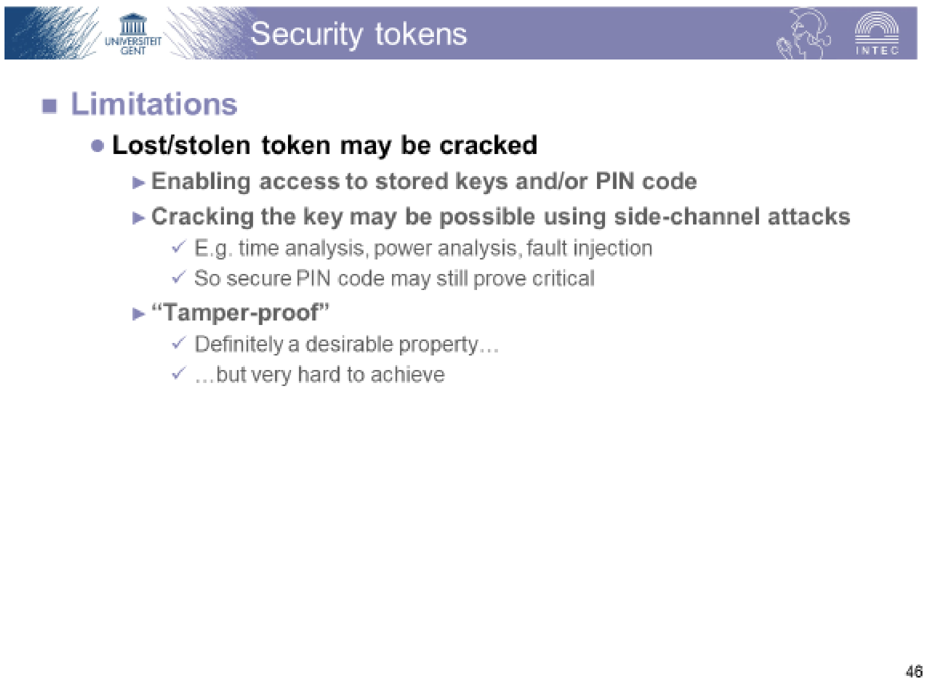
Note that with a well-chosen challenge, replay attacks are implausible. The only risk is that the "responses" of your token to all possible "challenges" are known by the attacker. As the number of "challenges" is limited (e.g. 10^6), it cannot be fully ruled out. The situation can be improved by requiring to input more than 1 "challenge" (at the cost of diminished user friendliness). An alternative is to make the "response" time dependent or to include a counter mechanism.



Security tokens

- Not physically connected to host
  - **Advantages**
    - **Less vulnerable to malware on host**
      - ✓ End user must *act* to use the token
  - **Disadvantages**
    - **Remaining vulnerabilities**
      - ✓ Phishing
      - ✓ Hacked token manufacturer
    - **End user has no trace of actions performed**
      - ✓ Possible issue for non-repudiation
    - **No encryption capacity**
    - **Requires more interaction from end user**

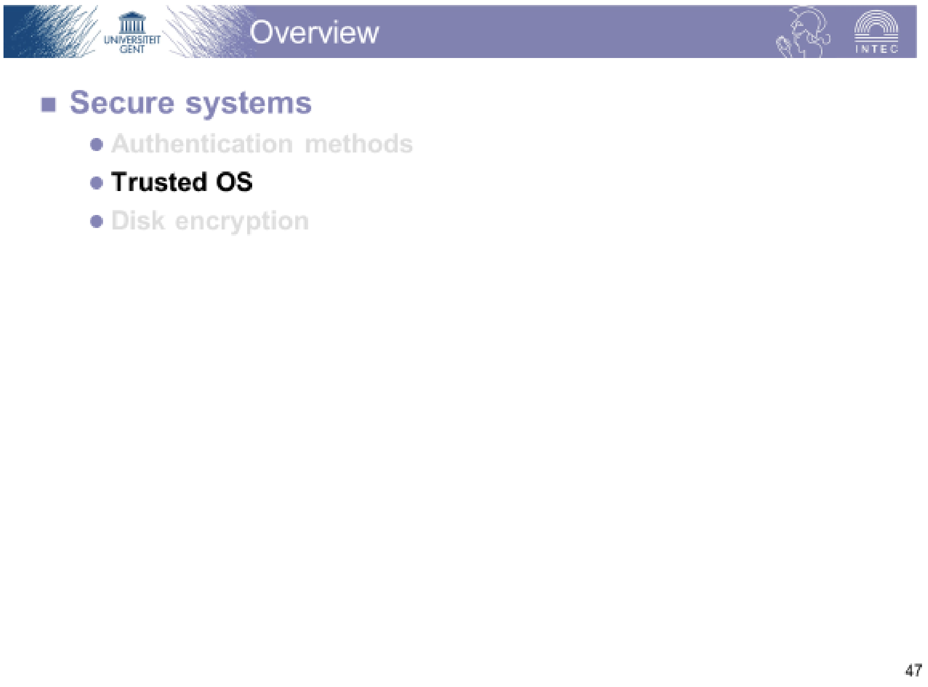
45



Security tokens

- Limitations
  - **Lost/stolen token may be cracked**
    - **Enabling access to stored keys and/or PIN code**
    - **Cracking the key may be possible using side-channel attacks**
      - ✓ E.g. time analysis, power analysis, fault injection
      - ✓ So secure PIN code may still prove critical
    - **"Tamper-proof"**
      - ✓ Definitely a desirable property…
      - ✓ …but very hard to achieve

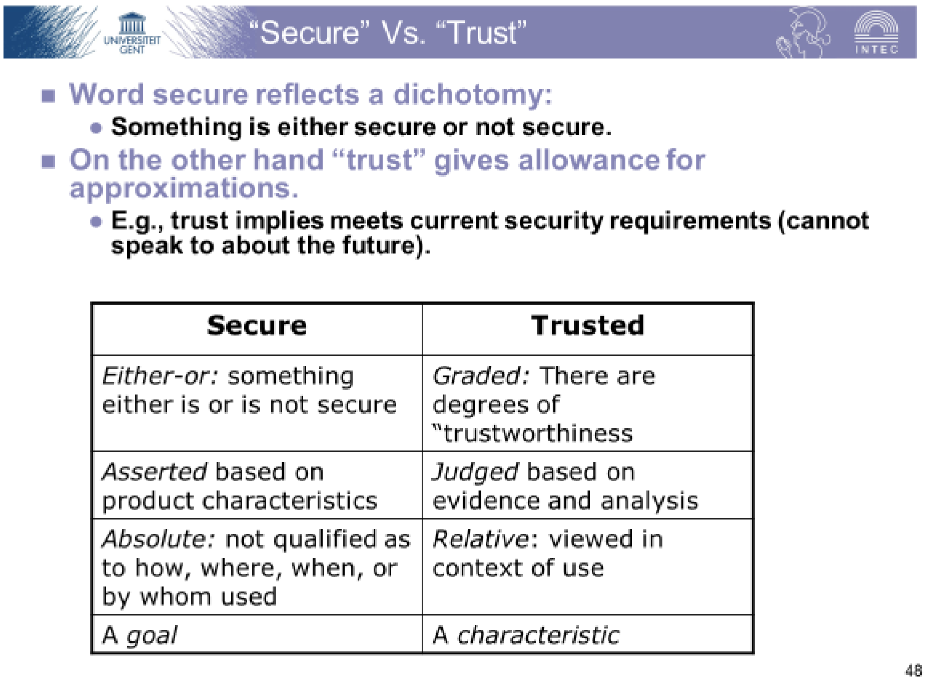
46



Overview

- Secure systems
  - Authentication methods
  - **Trusted OS**
  - Disk encryption

47

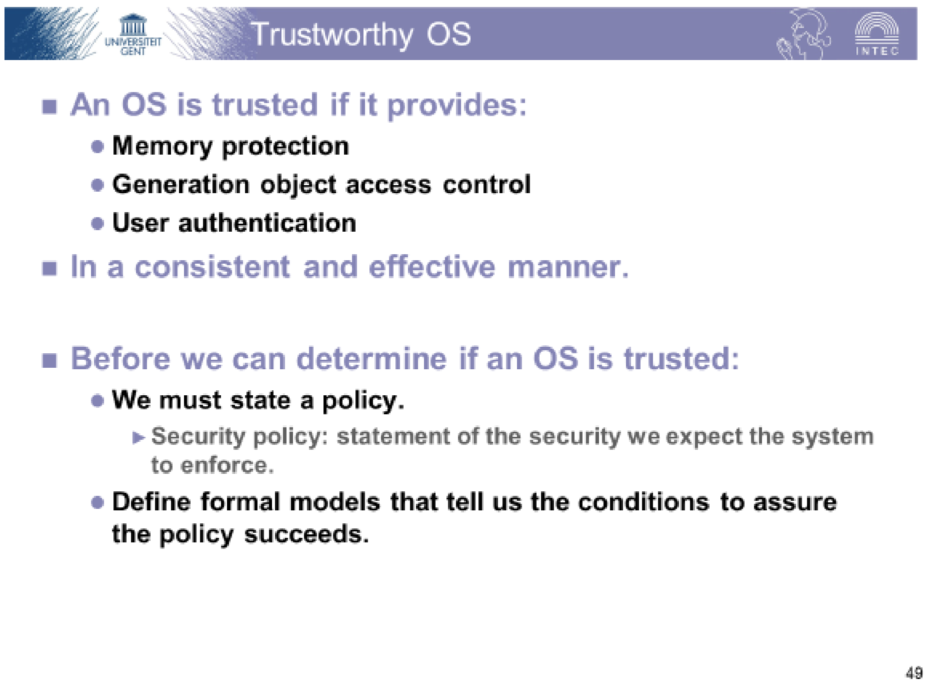


“Secure” Vs. “Trust”

- Word secure reflects a dichotomy:
  - **Something is either secure or not secure.**
- On the other hand “trust” gives allowance for approximations.
  - **E.g., trust implies meets current security requirements (cannot speak to about the future).**

Secure	**Trusted**
Either-or: something either is or is not secure	*Graded:* There are degrees of “trustworthiness
Asserted based on product characteristics	*Judged* based on evidence and analysis
Absolute: not qualified as to how, where, when, or by whom used	*Relative*: viewed in context of use
A *goal*	A *characteristic*

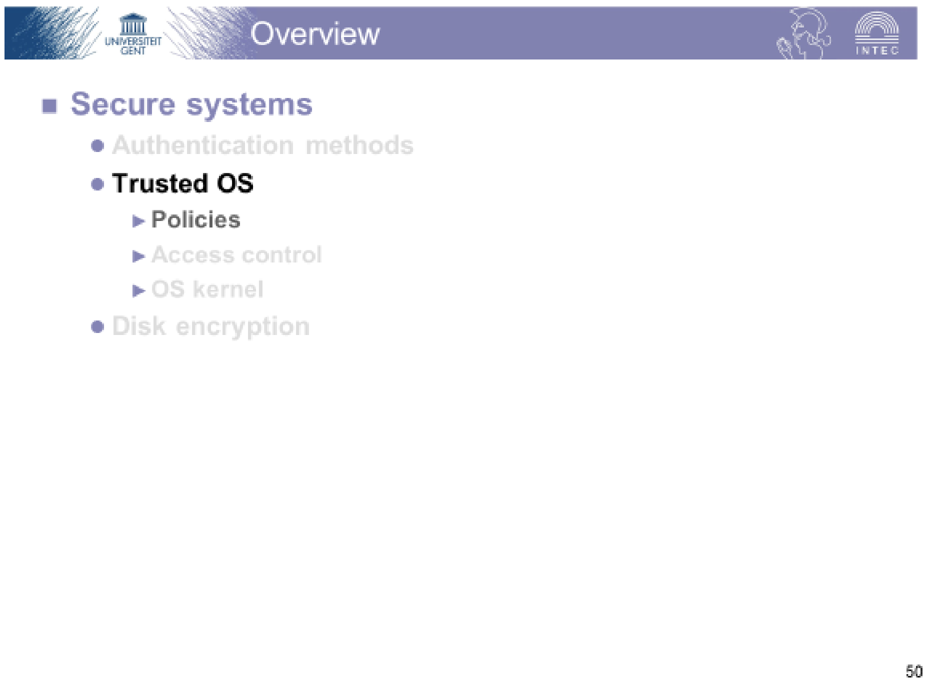
48



Trustworthy OS

- An OS is trusted if it provides:
  - Memory protection
  - Generation object access control
  - User authentication
- In a consistent and effective manner.
- Before we can determine if an OS is trusted:
  - We must state a policy.
    - Security policy: statement of the security we expect the system to enforce.
  - Define formal models that tell us the conditions to assure the policy succeeds.

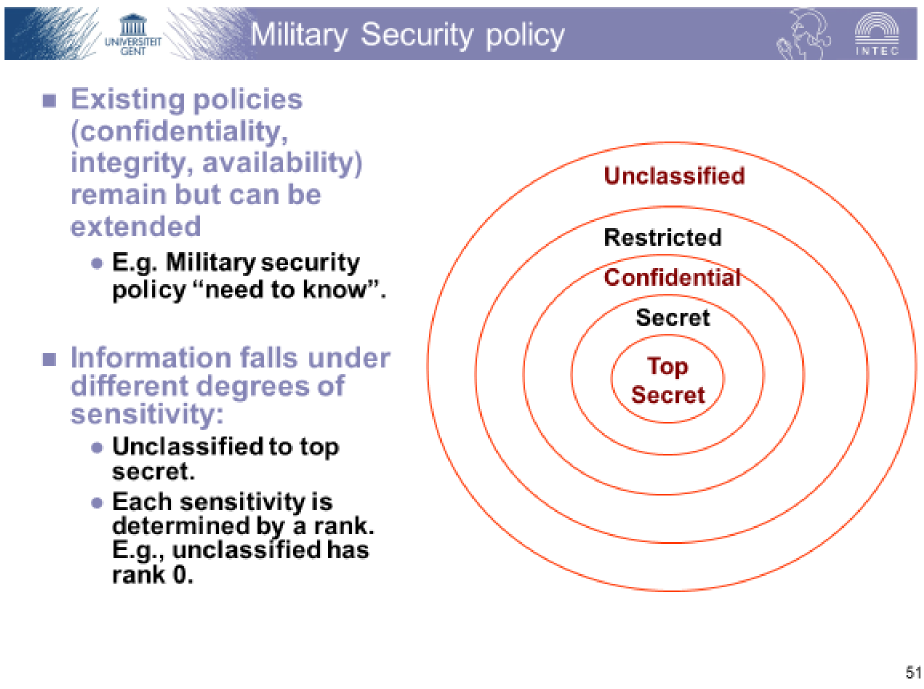
49



Overview

- Secure systems
  - Authentication methods
  - Trusted OS
    - Policies
    - Access control
    - OS kernel
  - Disk encryption

50



Military Security policy

- Existing policies (confidentiality, integrity, availability) remain but can be extended
  - E.g. Military security policy "need to know".
- Information falls under different degrees of sensitivity:
  - Unclassified to top secret.
  - Each sensitivity is determined by a rank. E.g., unclassified has rank 0.

Unclassified
Restricted
Confidential
Secret
Top Secret

51

A security clearance is a status granted to individuals allowing them access to classified information (state or organizational secrets) or to restricted areas, after completion of a thorough background check. The term "security clearance" is also sometimes used in private organizations that have a formal process to vet employees for access to sensitive information.

Top Secret is applied to information that reasonably could be expected to cause exceptionally grave damage to the national security if disclosed to unauthorized sources.

This level needs to be reinvestigated every 5 years.*

Secret is applied to information that reasonably could be expected to cause serious damage to the national security if disclosed to unauthorized sources.

This level is reinvestigated every 10 years.*

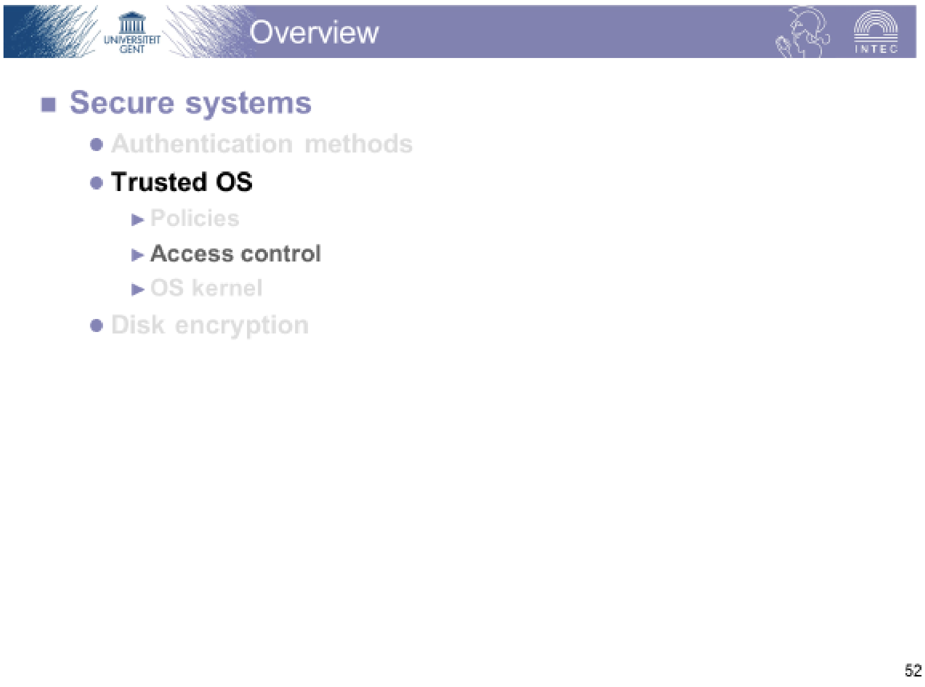
Confidential is applied to information that reasonably could be expected to cause damage to the national security if disclosed to unauthorized sources. The vast majority of military personnel are given this very basic level of clearance.

This level needs to be reinvestigated every 15 years.*

Material that is classified as **Restricted** (or "For Official Use Only (U//FOUO))" is considered between Unclassified and Confidential and may deal with employee data.

Unclassified is a valid security description, especially when indicating unclassified information within a document classified at a higher level. For example, the title of a Secret report is often unclassified, and must be marked as such.

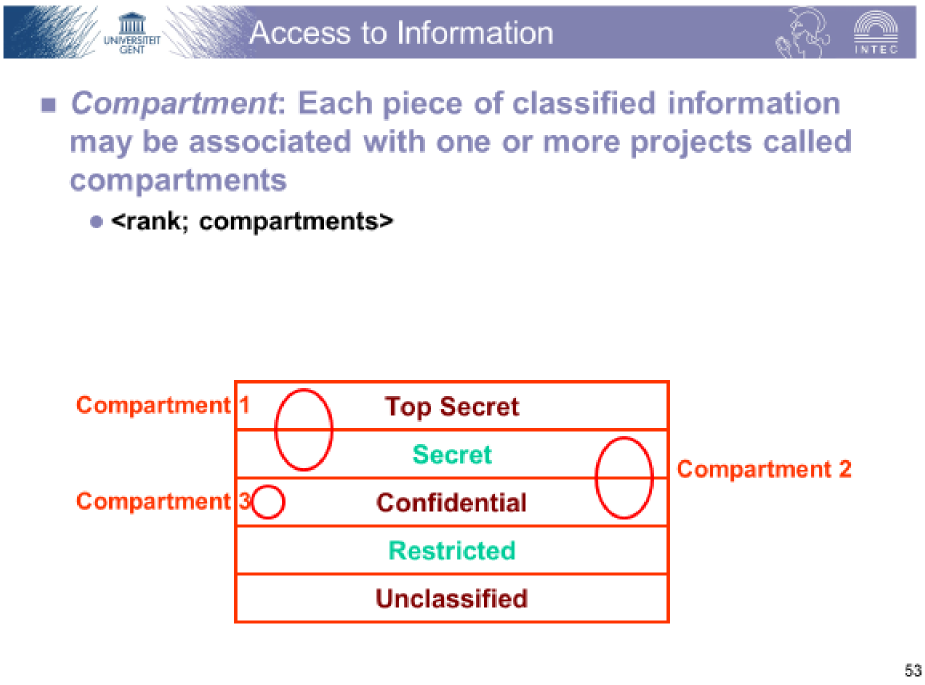
* Reinvestigations are more important than the original investigation because those individuals who have held clearances longer are more likely to be working with increasingly critical information.



Overview

- Secure systems
  - Authentication methods
  - **Trusted OS**
    - Policies
    - Access control
    - OS kernel
  - Disk encryption

52

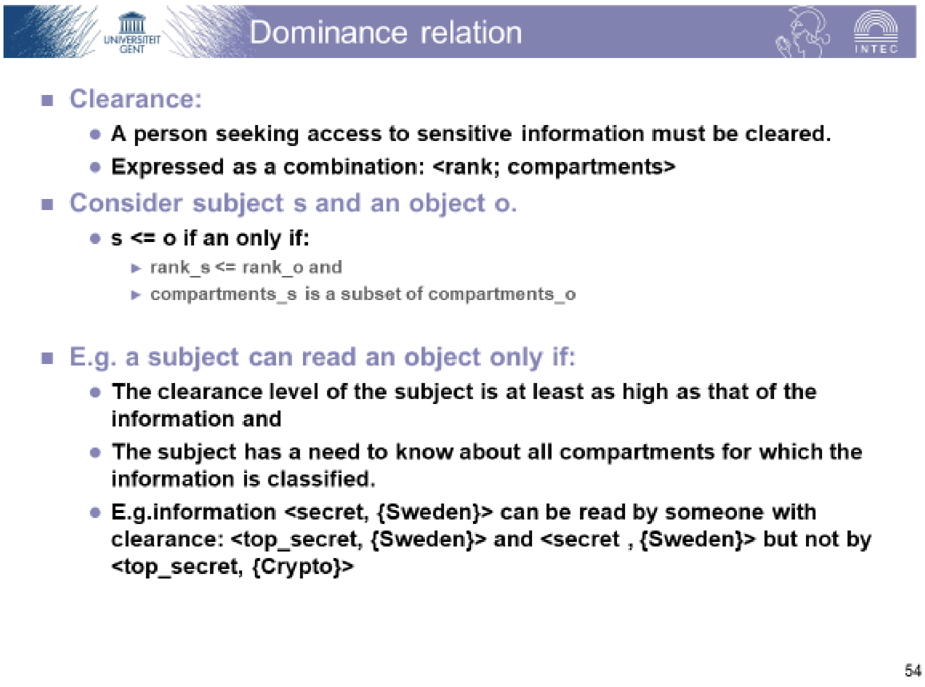


Access to Information

- *Compartment*: Each piece of classified information may be associated with one or more projects called compartments
  - **<rank; compartments>**

53

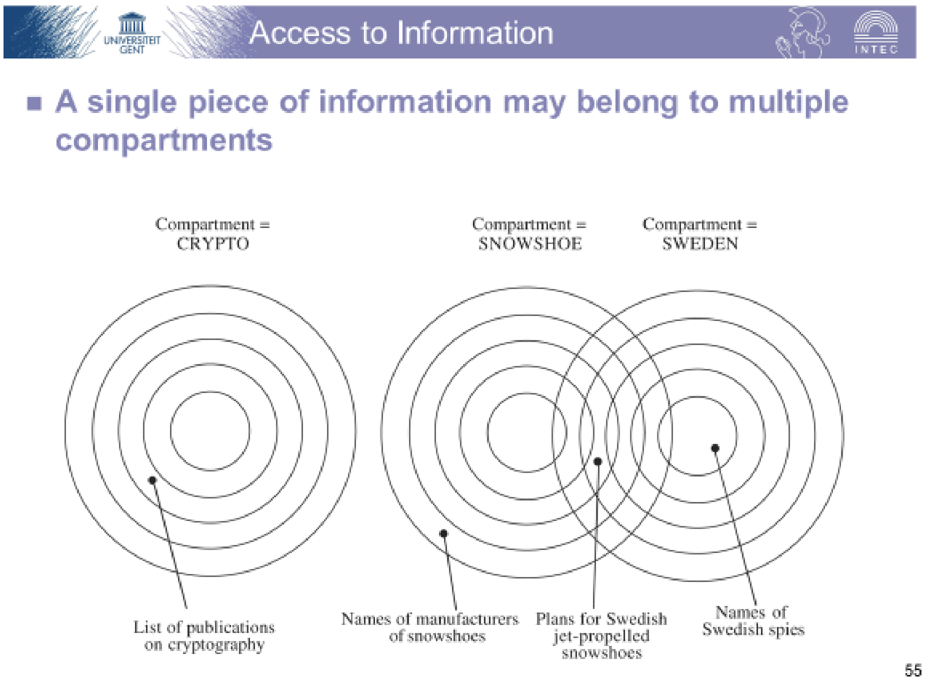
A clearance by itself is normally not sufficient to gain access; the organization must also determine that the cleared individual needs to know specific information ("need to known base"). No one is supposed to be granted automatic access to classified information solely because of rank, position, or a security clearance.



Dominance relation

- Clearance:
  - A person seeking access to sensitive information must be cleared.
  - Expressed as a combination: <rank; compartments>
- Consider subject s and an object o.
  - s <= o if an only if:
    - rank_s <= rank_o and
    - compartments_s is a subset of compartments_o
- E.g. a subject can read an object only if:
  - The clearance level of the subject is at least as high as that of the information and
  - The subject has a need to know about all compartments for which the information is classified.
  - E.g.information <secret, {Sweden}> can be read by someone with clearance: <top_secret, {Sweden}> and <secret , {Sweden}> but not by <top_secret, {Crypto}>

54



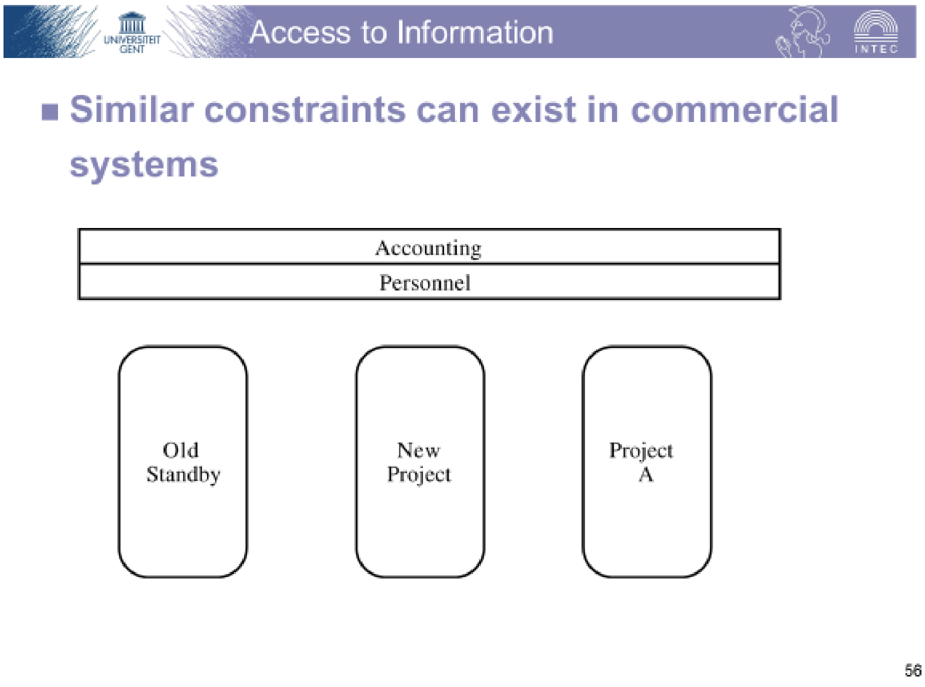
Access to Information

- A single piece of information may belong to multiple compartments

Compartment = CRYPTO
Compartment = SNOWSHOE
Compartment = SWEDEN
List of publications on cryptography
Names of manufacturers of snowshoes
Plans for Swedish jet-propelled snowshoes
Names of Swedish spies

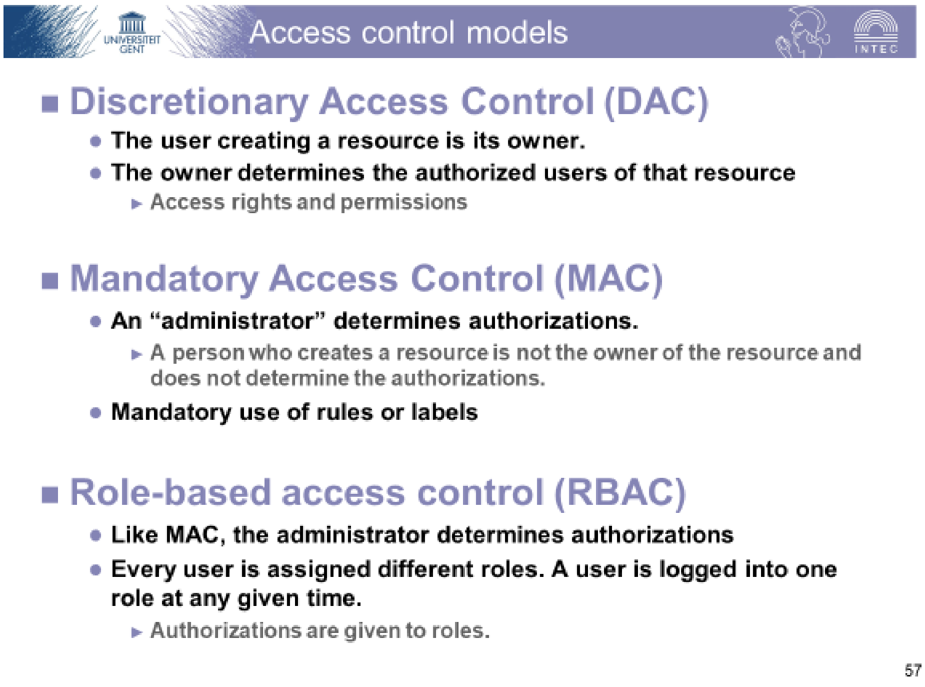
55

A particular project may need to use information which is both top secret and secret. . In this case, a solution is the creation of a compartment to cover the information in both. This compartment may include information across multiple sensitivity levels.



Access to Information

- Similar constraints can exist in commercial systems



Access control models

- Discretionary Access Control (DAC)
  - The user creating a resource is its owner.
  - The owner determines the authorized users of that resource
    - Access rights and permissions
- Mandatory Access Control (MAC)
  - An "administrator" determines authorizations.
    - A person who creates a resource is not the owner of the resource and does not determine the authorizations.
  - Mandatory use of rules or labels
- Role-based access control (RBAC)
  - Like MAC, the administrator determines authorizations
  - Every user is assigned different roles. A user is logged into one role at any given time.
    - Authorizations are given to roles.

57

Access control models are sometimes categorized as either discretionary or non-discretionary. The three most widely recognized models are Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC).

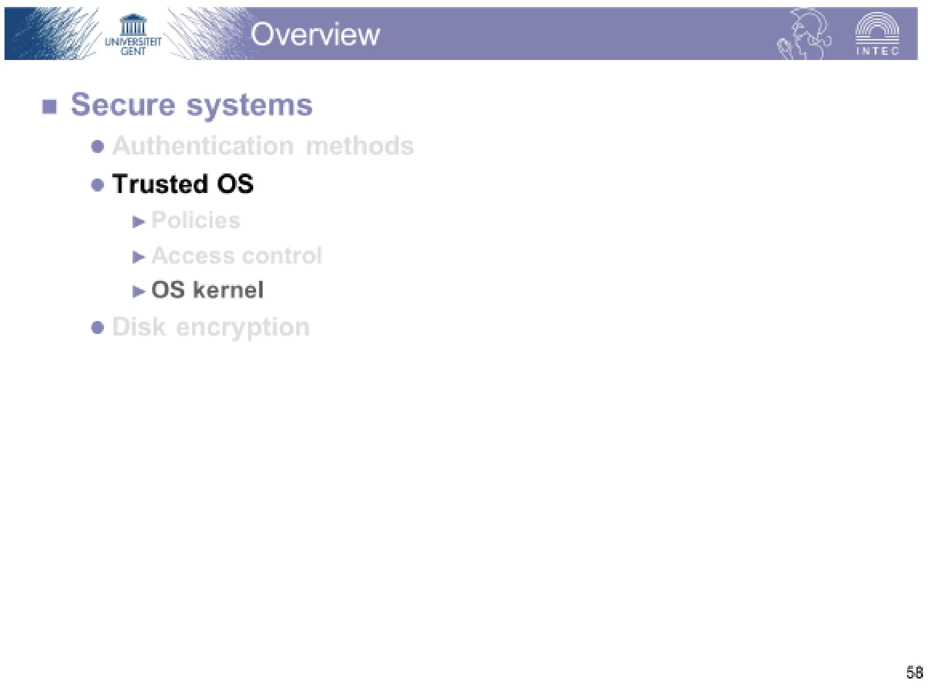
Discretionary access control (DAC) is a policy determined by the owner of an object. The owner decides who is allowed to access the object, and what privileges they have.

Two important concepts in DAC are

- File and data ownership: Every object in the system has an owner. In most DAC systems, each object's initial owner is the subject that caused it to be created. The access policy for an object is determined by its owner.
- Access rights and permissions: These are the controls that an owner can assign to other subjects for specific resources.

Mandatory access control (MAC) refers to allowing access to a resource if and only if rules exist that allow a given user to access the resource. The term mandatory has historically implied a very high degree of robustness that assures that the control mechanisms resist subversion, thereby enabling them to enforce an access control policy that is mandated by some regulation that must be absolutely enforced. It is more difficult to manage, but its use is usually justified when used to protect highly sensitive information. Examples include certain government and military information. Management is often simplified (over what can be required) if the information can be protected using hierarchical access control, or by implementing sensitivity labels. In such a system subjects and objects must have labels assigned to them. A subject's sensitivity label specifies its level of trust. An object's sensitivity label specifies the level of trust required for access. In order to access a given object, the subject must have a sensitivity level equal to or higher than the requested object. Controlling the import of information from other systems and export to other systems (including printers) is a critical function of these systems, which must ensure that sensitivity labels are properly maintained and implemented so that sensitive information is appropriately protected at all times.

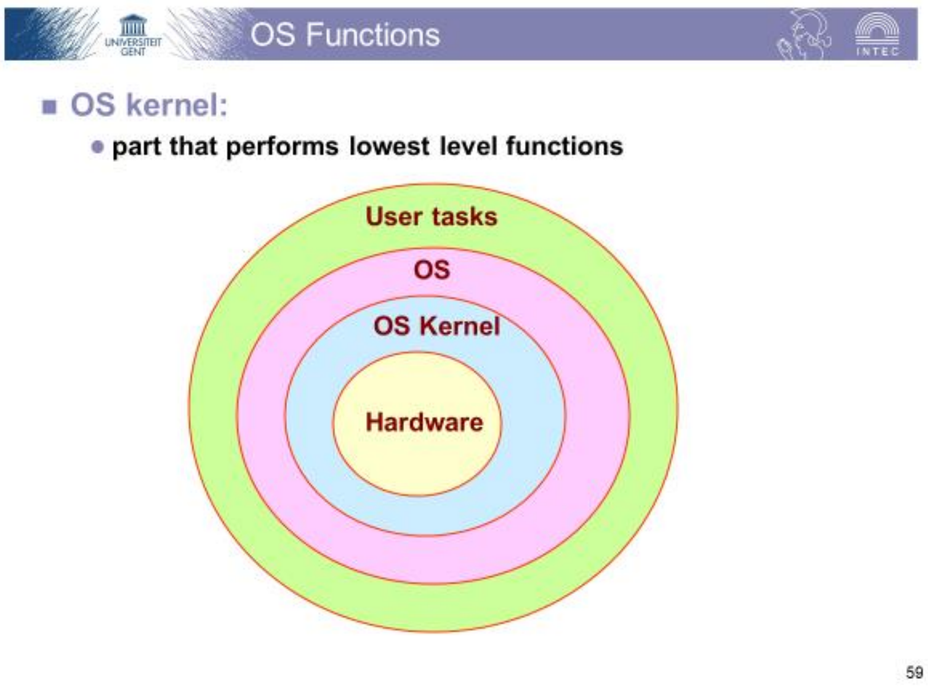
Role-based access control (RBAC) is also an access policy determined by the system, not by the owner. RBAC is used in commercial applications and also in military systems, where multi-level security requirements may also exist. It can be distinguished from MAC primarily in the way permissions are handled. MAC controls read and write permissions based on a user's clearance level and additional labels. RBAC controls collections of permissions that may include complex operations such as an e-commerce transaction, or may be as simple as read or write. Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members or staff (or other system users) are assigned particular roles, and through those role assignments acquire the computer permissions to perform particular computer-system functions. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user's account; this simplifies common operations, such as adding a user, or changing a user's department. A role in RBAC can be viewed as a set of permissions.



Overview

- Secure systems
  - Authentication methods
  - Trusted OS
    - Policies
    - Access control
    - OS kernel
  - Disk encryption

58

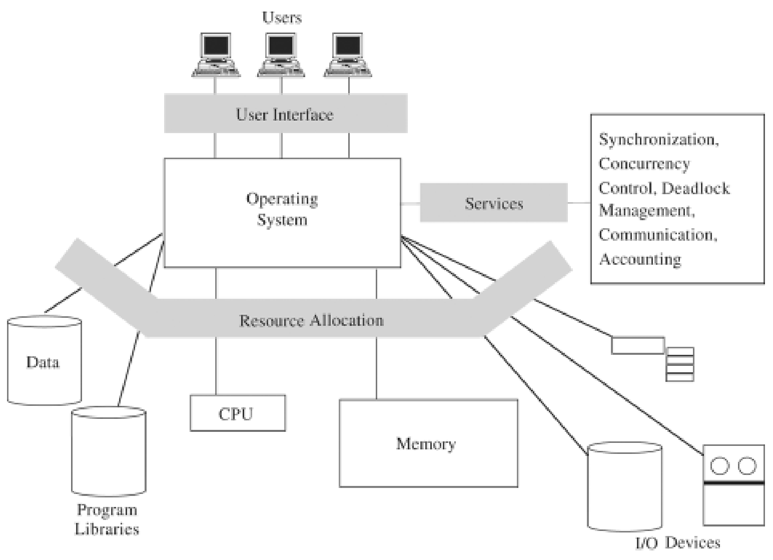


OS Functions

- OS kernel:
  - part that performs lowest level functions

59

OS Functions



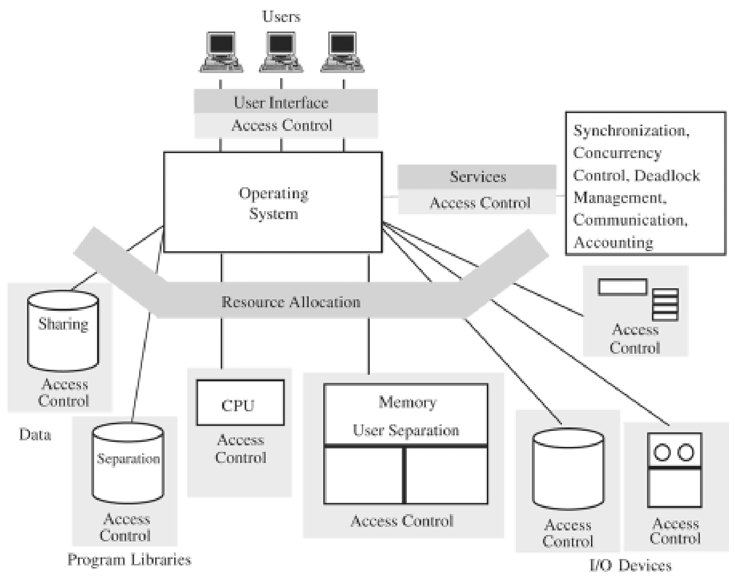
60

Security features in ordinary OS

- Authentication of users
  - password comparison
- Protection of memory
  - user space, paging, segmentations
- File and I/O device access control
  - access control matrix
- Allocation & access control to general objects
  - table lookup
- Enforcement of sharing
  - integrity, consistency
- Fair service
  - no starvation
- Inter process communication & synchronization
  - table lookup
- Protection of OS protection data
  - encryption, hardware control, isolation

61

Trusted OS Functions



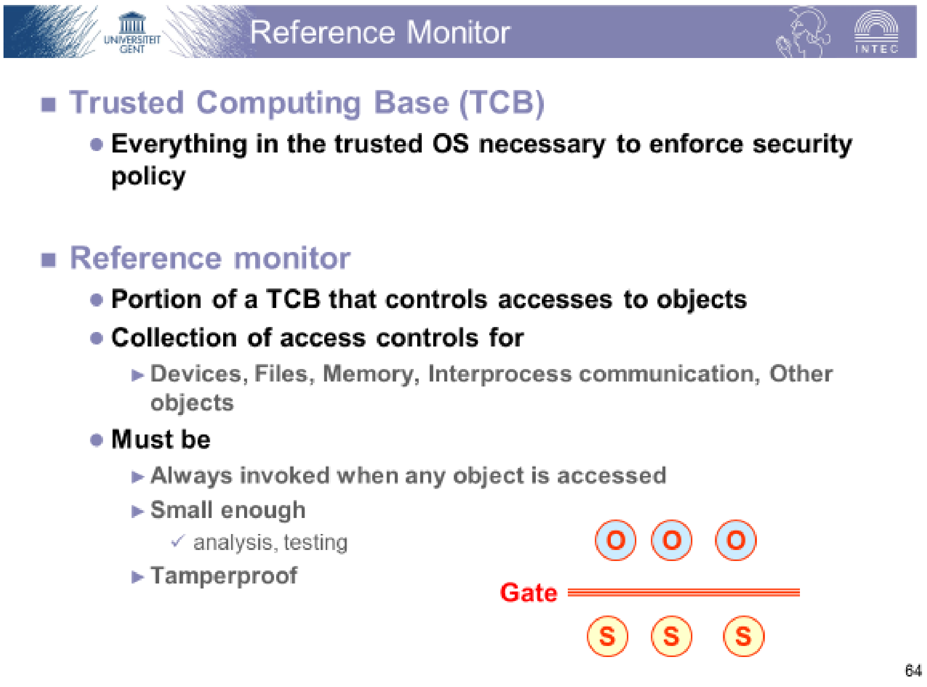
62

Security features of Trusted OS

Design features

- Coverage
  - ensure that every access is checked
- Separation
  - security mechanisms are isolated from the rest of OS and from user space → easier to protect
- Unity
  - all security mechanisms are performed by a single set of code → easier to trace problems
- Modifiability
  - security mechanism changes are easier to make and test
- Compactness
  - relatively small
- Verifiability
  - formal methods, all situations are covered

63



Reference Monitor

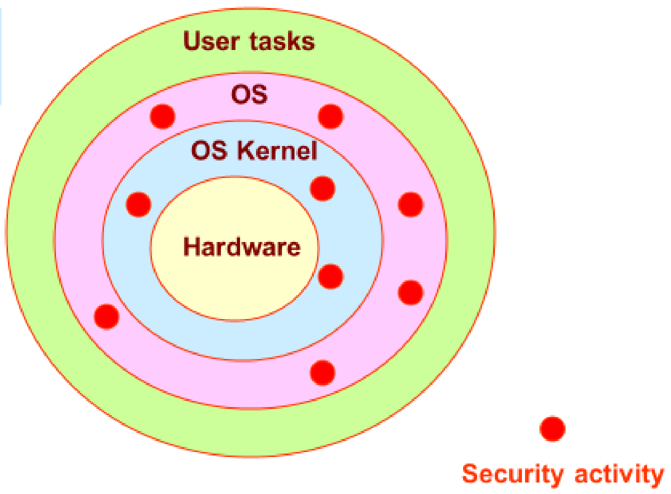
- **Trusted Computing Base (TCB)**
  - **Everything in the trusted OS necessary to enforce security policy**
- **Reference monitor**
  - **Portion of a TCB that controls accesses to objects**
  - **Collection of access controls for**
    - Devices, Files, Memory, Interprocess communication, Other objects
  - **Must be**
    - Always invoked when any object is accessed
    - Small enough
      - ✓ analysis, testing
    - Tamperproof

64

The trusted computing base (TCB) of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system. By contrast, parts of a computer system outside the TCB must not be able to misbehave in a way that would leak any more privileges than are granted to them in accordance to the security policy. The Orange Book, a classic computer security literature reference, describes the TCB of a computer system, as "the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy". In other words, a given piece of hardware or software is a part of the TCB if and only if it has been designed to be a part of the mechanism that provides its security to the computer system. In operating systems, this typically consists of the kernel (or microkernel) and a select set of system utilities (for example, setuid programs and daemons in UNIX systems)

Combined Security Kernel / OS System

OS Kernel:
- HW interactions
- Access control

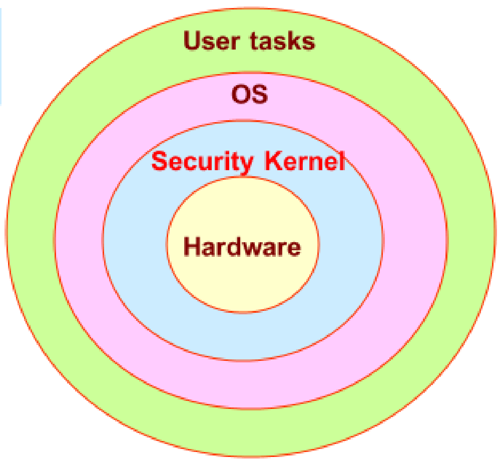


OS:
- Resource allocation
- Sharing
- Access control
- Authentication functions

65

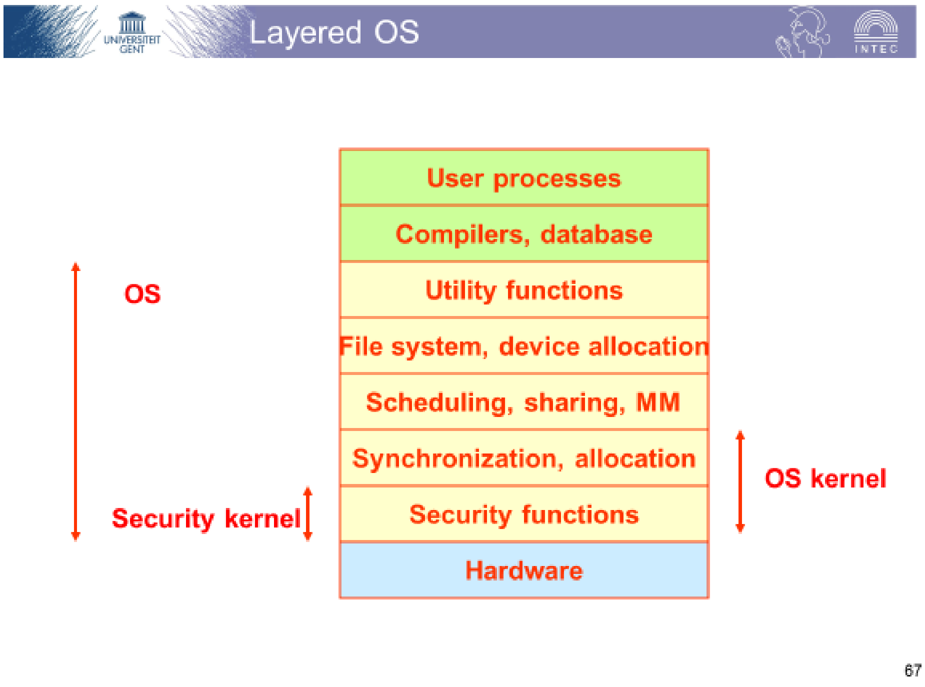
Separate Security Kernel

OS Kernel:
- HW interactions
- Access control

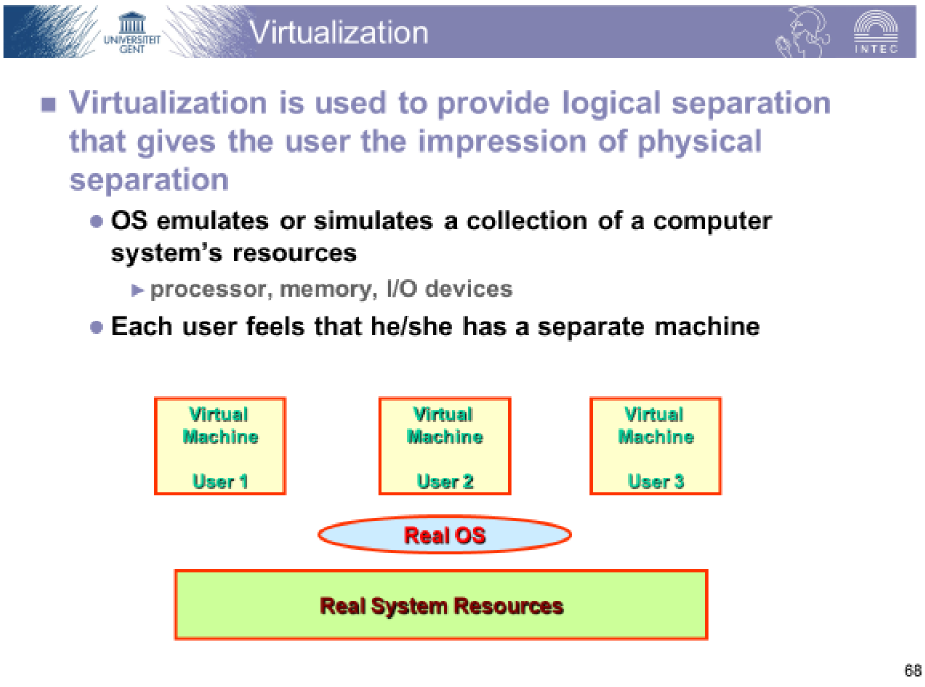


OS:
- Resource allocation
- Sharing
- Access control
- Authentication functions

66



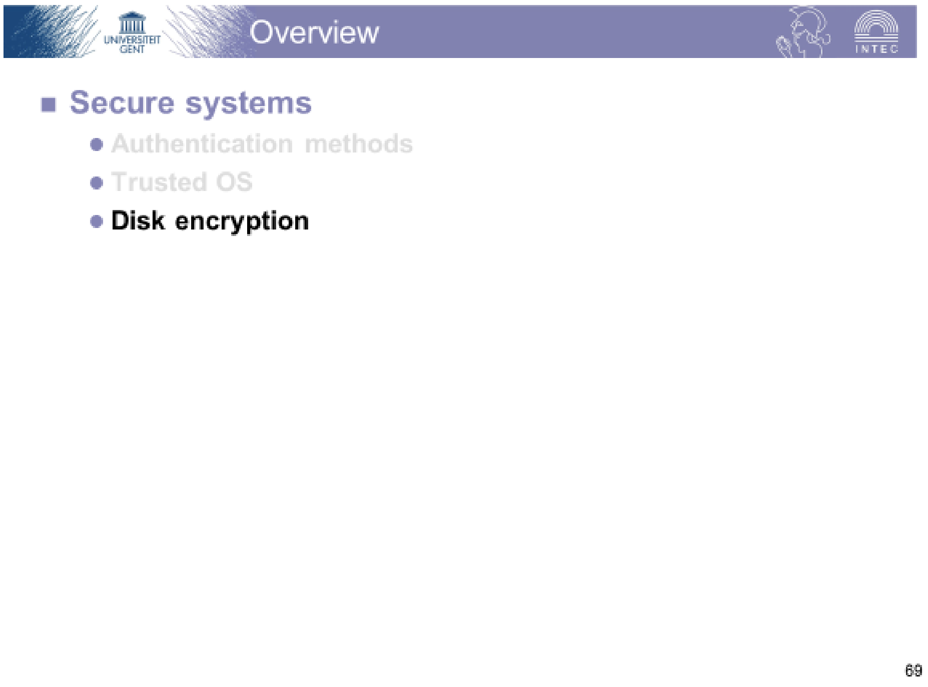
Layered OS



Virtualization

- Virtualization is used to provide logical separation that gives the user the impression of physical separation
  - OS emulates or simulates a collection of a computer system's resources
    - processor, memory, I/O devices
  - Each user feels that he/she has a separate machine

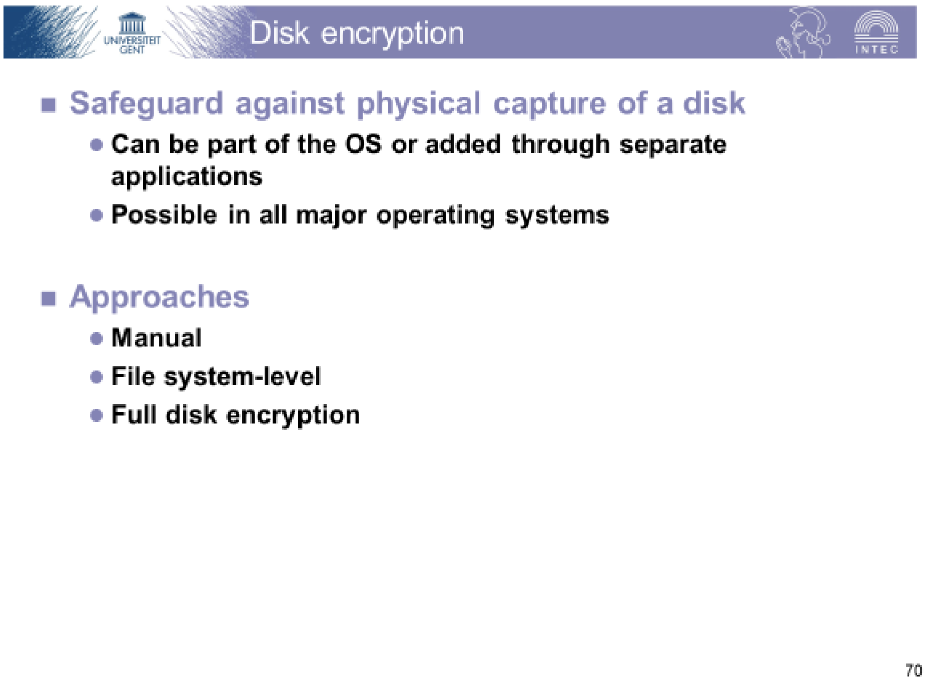
Virtual Machine User 1
Virtual Machine User 2
Virtual Machine User 3
Real OS
Real System Resources
68



Overview

- Secure systems
  - Authentication methods
  - Trusted OS
  - Disk encryption

69

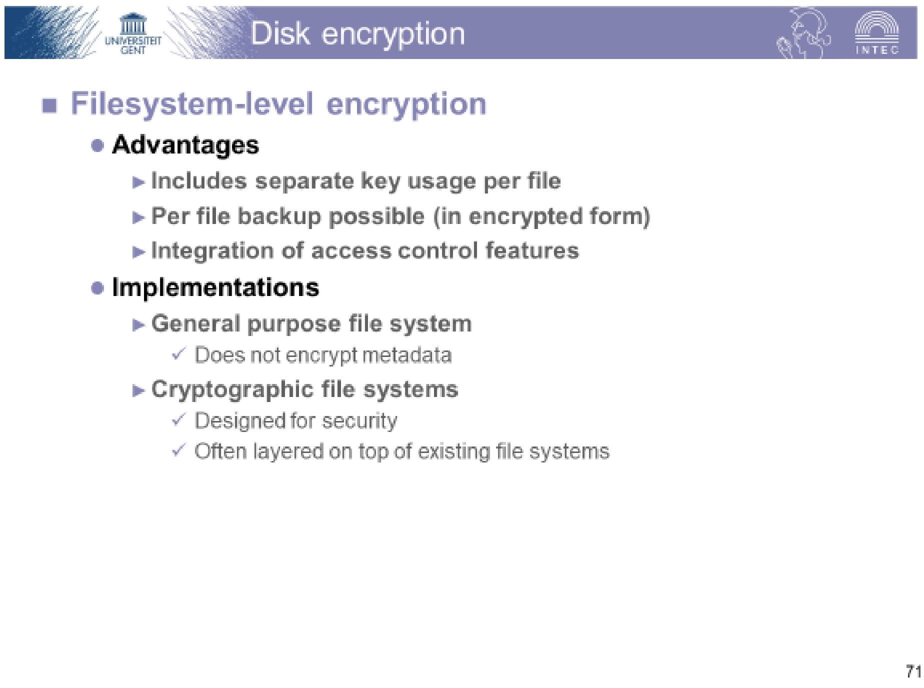


Disk encryption

- Safeguard against physical capture of a disk
  - Can be part of the OS or added through separate applications
  - Possible in all major operating systems
- Approaches
  - Manual
  - File system-level
  - Full disk encryption

70

Manuel encryption requires an active choice from the user to encrypt or decrypt specific files or directories. In contrast, in filesystem-system level encryption, Individual files or directories are encrypted & decrypted by the file system itself, which is often performed as of the OS. Finally, full disk encryption approaches encrypt the full system (sometimes including boot sector). These last approaches are typically implemented below OS level.



Disk encryption

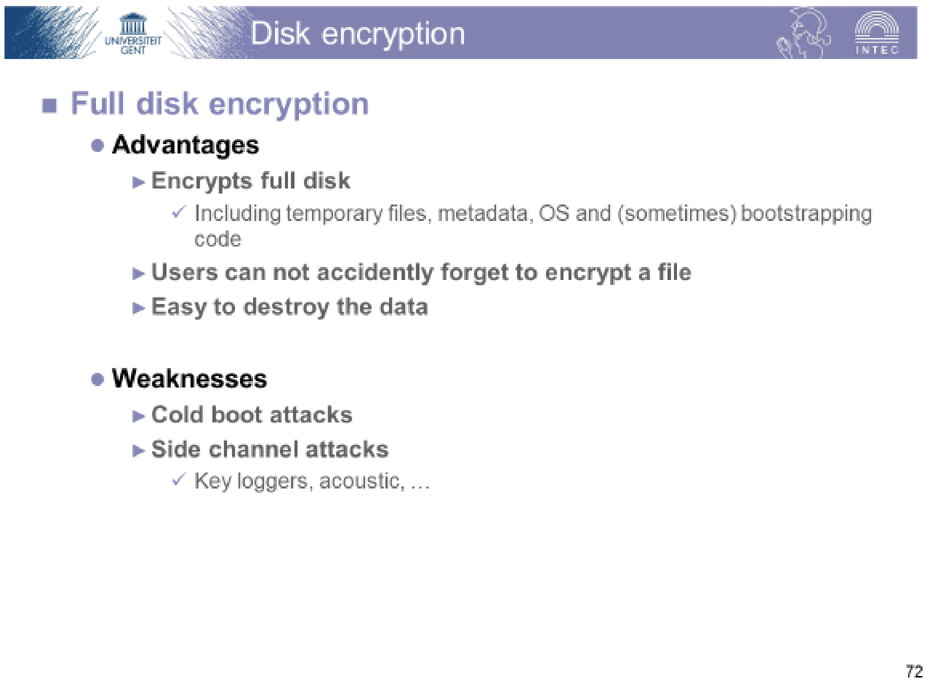
- Filesystem-level encryption
  - Advantages
    - Includes separate key usage per file
    - Per file backup possible (in encrypted form)
    - Integration of access control features
  - Implementations
    - General purpose file system
      - ✓ Does not encrypt metadata
    - Cryptographic file systems
      - ✓ Designed for security
      - ✓ Often layered on top of existing file systems

71

In contrast to full disk encryption, filesystem-level encryption, often called file/folder encryption, is a form of disk encryption where individual files or directories are encrypted by the file system itself.

General-purpose file systems that include filesystem-level encryption do not typically encrypt file system metadata, such as the directory structure, file names, sizes or modification timestamps. This can be problematic if the metadata itself needs to be kept confidential. In other words, if files are stored with identifying file names, anyone who has access to the physical disk can know which documents are stored on the disk, although not the contents of the documents. One exception to this is the encryption support being added to the ZFS filesystem. Filesystem metadata such as filenames, ownership, ACLs, extended attributes are all stored encrypted on disk. The ZFS metadata relating to the storage pool is stored in plaintext, so it is possible to determine how many filesystems (datasets) are available in the pool, including which ones are encrypted. The content of the stored files and directories remain encrypted.

Cryptographic file systems are specialized (not general-purpose) file systems that are specifically designed with encryption and security in mind. They usually encrypt all the data they contain – including metadata. Instead of implementing an on-disk format and their own block allocation, these file systems are often layered on top of existing file systems e.g. residing in a directory on a host file system. Many such file systems also offer advanced features, such as deniable encryption, cryptographically secure read-only file system permissions and different views of the directory structure depending on the key or user. One use for a cryptographic file system is when part of an existing file system is synchronized with 'cloud storage'. In such cases the cryptographic file system could be 'stacked' on top, to help protect data confidentiality.



Disk encryption

- Full disk encryption
  - Advantages
    - Encrypts full disk
      - ✓ Including temporary files, metadata, OS and (sometimes) bootstrapping code
    - Users can not accidently forget to encrypt a file
    - Easy to destroy the data
  - Weaknesses
    - Cold boot attacks
    - Side channel attacks
      - ✓ Key loggers, acoustic, …

72

Full disk encryption has several benefits compared to regular file or folder encryption, or encrypted vaults. The following are some benefits of disk encryption.

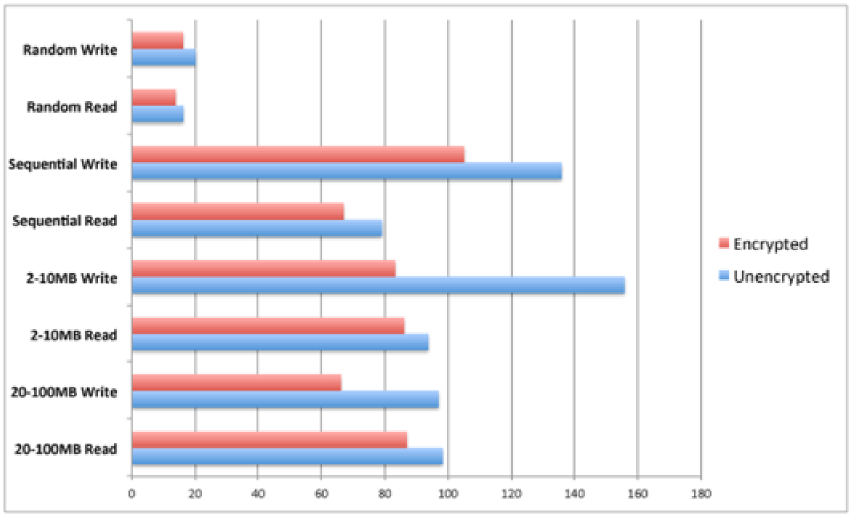
- Nearly everything including the swap space and the temporary files is encrypted. Encrypting these files is important, as they can reveal important confidential data. With a software implementation, the bootstrapping code cannot be encrypted however. (For example, BitLocker Drive Encryption leaves an unencrypted volume to boot from, while the volume containing the operating system is fully encrypted.)
- With full disk encryption, the decision of which individual files to encrypt is not left up to users' discretion. This is important for situations in which users might not want or might forget to encrypt sensitive files.
- Immediate data destruction, such as simply destroying the cryptographic keys, renders the contained data useless.

Weaknesses

- Most full disk encryption schemes are vulnerable to a cold boot attack, whereby encryption keys can be stolen by cold-booting a machine already running an operating system, then dumping the contents of memory before the data disappears. The attack relies on the data remanence property of computer memory, whereby data bits can take up to several minutes to degrade after power has been removed. Even a Trusted Platform Module (TPM) is not effective against the attack, as the operating system needs to hold the decryption keys in memory in order to access the disk.
- All software-based encryption systems are vulnerable to various side channel attacks such as acoustic cryptanalysis and hardware keyloggers. In contrast, self-encrypting drives are not vulnerable to these attacks since the hardware encryption key never leaves the disk controller.

Disk encryption

- Full disk encryption
  - MAC OS X 10.7 filevault performance impact



73

From http://www.maclive.net/macosx107lionfilevaultwholediskencryptionbenchmarkcomparison/

Disk encryption

- Possible features
  - Plausible denial
  - Hidden containers
  - Pre-boot authentication
  - Single sign-on
  - Custom authentication
  - Multiple keys
  - Passphrase strengthening
  - Hardware acceleration
  - Trusted Platform Module
  - Filesystems
  - Two-factor authentication
  - Boot sector encryption

74

Plausibly deniability: describes encryption techniques where the existence of an encrypted file or message is deniable in the sense that an adversary cannot prove that the plaintext data exists.

Hidden containers: Includes hidden containers (an encrypted container (A) within another encrypted container (B) so the existence of container A can not be established) that are created for deniable encryption.

Pre-boot authentication: Whether authentication can be required before booting the computer, thus allowing one to encrypt the boot disk. One issue to address in full disk encryption is that the blocks where the operating system is stored must be decrypted before the OS can boot, meaning that the key has to be available before there is a user interface to ask for a password. Most Full Disk Encryption solutions utilize Pre-Boot Authentication by loading a small, highly secure operating system which is strictly locked down and hashed versus system variables to check for the integrity of the Pre-Boot kernel.

Single sign-on: Whether credentials provided during pre-boot authentication will automatically log the user into the host operating system, thus preventing password fatigue and reducing the need to remember multiple passwords.

Custom authentication: Whether custom authentication mechanisms can be implemented with third-party applications.

Multiple keys: Whether an encrypted volume can have more than one active key. Sometimes, one key is used to destroy all data on the disk when used, or to boot in such a way that only containers which do not contain sensitive information are shown.

Passphrase strengthening: Whether key strengthening is used with plain text passwords to frustrate dictionary attacks, usually using PBKDF2.

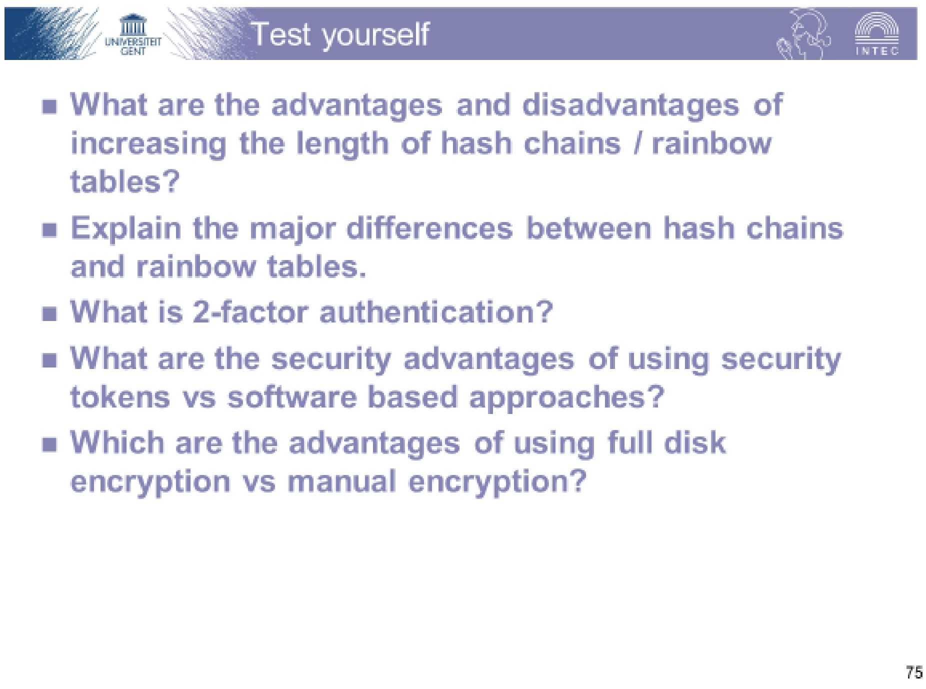
Hardware acceleration: Whether dedicated cryptographic accelerator expansion cards can be taken advantage of.

Trusted Platform Module: Trusted Platform Module (TPM) is a secure cryptoprocessor embedded in the motherboard that can be used to authenticate a hardware device. Since each TPM chip is unique to a particular device, it is capable of performing platform authentication. It can be used to verify that the system seeking the access is the expected system. A limited number of disk encryption solutions have support for TPM. These implementations can wrap the decryption key using the TPM, thus tying the hard disk drive (HDD) to a particular device. If the HDD is removed from that particular device and placed in another, the decryption process will fail. Recovery is possible with the decryption password or token.

Filesystems: what filesystems are supported.

Two-factor authentication: Whether optional security tokens (hardware security modules, such as Aladdin eToken and smart cards) are supported (for example using PKCS#11)

Boot sector encryption: Whole disk encryption often signify that everything on disk is encrypted – including the programs that can encrypt bootable operating system partitions – when part of the disk is necessarily not encrypted. On systems that use a master boot record (MBR), that part of the disk remains non encrypted. Some hardware-based full disk encryption systems can truly encrypt an entire boot disk, including the MBR.



Test yourself

- What are the advantages and disadvantages of increasing the length of hash chains / rainbow tables?
- Explain the major differences between hash chains and rainbow tables.
- What is 2-factor authentication?
- What are the security advantages of using security tokens vs software based approaches?
- Which are the advantages of using full disk encryption vs manual encryption?

75