Password Cracking (cont.)

Example: Brute-Force Password Search Space

Compare two systems in which the maximum password length is 16, and passwords may contain any printable ASCII characters.

System A allows passwords to be any length (1-16)

System B requires passwords to be at least 8 characters long (8-16).

Calculate the search space for these two systems.

A=94

System A:

$$S_{1\text{-}16} = \sum_{i=1}^{16} A^i = \frac{A^{16+1}-1}{A-1} = 3.75 \cdot 10^{31}$$

System B:

$$S_{8\text{-}16} = \sum_{i=8}^{16} A^i = \sum_{i=1}^{16} A^i - \sum_{i=1}^{7} A^i =$$

$$\approx 3.75 \cdot 10^{31}$$

Almost the same!!!

Password Cracking (cont.)

- **Biased Attack** ◈ **the search space is further reduced by focusing on most likely combinations of words and/or numbers …**

Example: Biased Attack on 4-Digit Pins

Assume a system requires that access passwords be comprised of 4 digits.

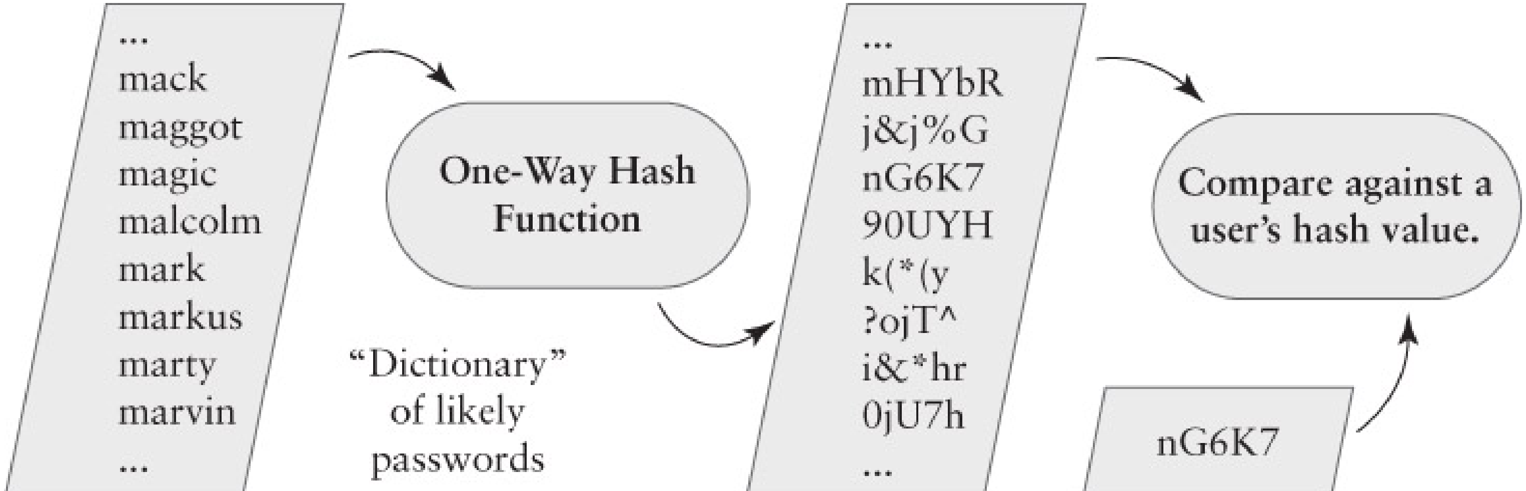
Total unbiased search space: any number between 0 – 9999 (10,000)

Many people use some important personal dates to generate 4-digit passwords.

Biased search space: **only 366 possible combinations!**

Password Cracking (cont.)

- **Dictionary Attack**
  - ◈ **users often create passwords using common dictionary words**
    - instead of trying every password, dictionary attack probes only common dictionary words
    - faster than brute force, as it uses smaller (more likely) search space
    - still might take considerable time, and might fail in the end



Password Cracking (cont.)

Example: Dictionary Attacks in Real World

Many studies on effectiveness of dictionary attack have been conducted.

Not 100% effective, but enough passwords were cracked to make the use of this attack worthwhile.

Research or Incident	% Guessed
Morris worm, estimated success (1988)	~50%
Klein's Study (1990)	24.2%
Spafford's Study (1992)	20%
CERT Incident 1998-03	25.6%
Cambridge study by Yan, et al. (2000)	35%
Lulz and Anonymous, estimated success (2011)	30%

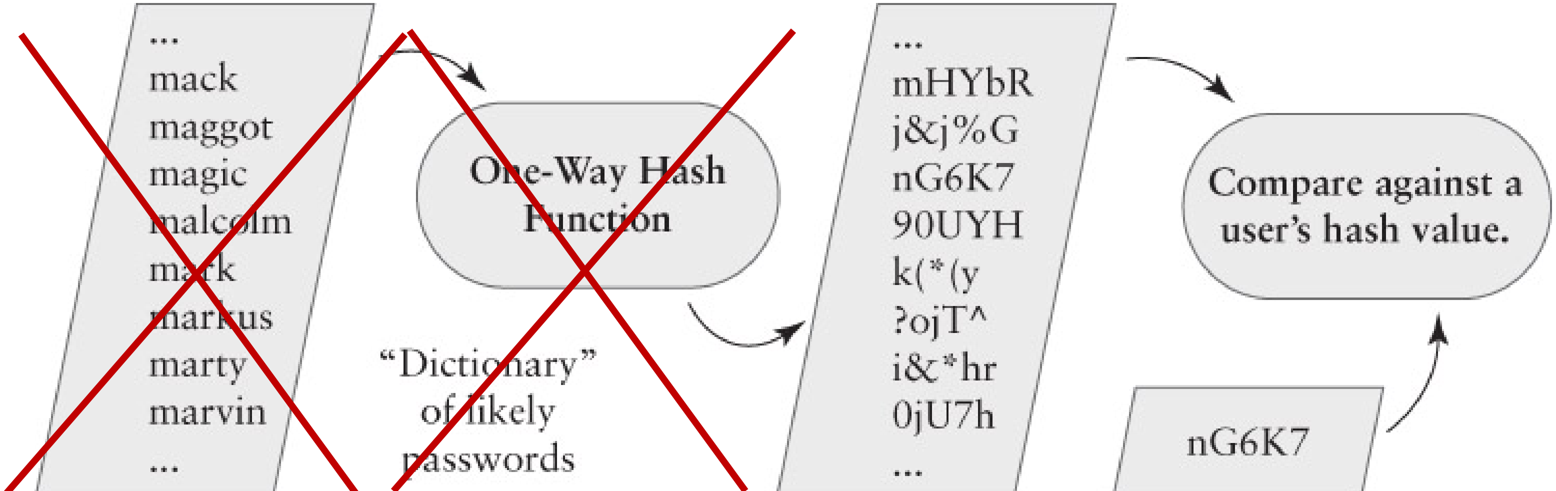
Elementary Information Security, R. E. Smith, pp. 253

Password Cracking (cont.)

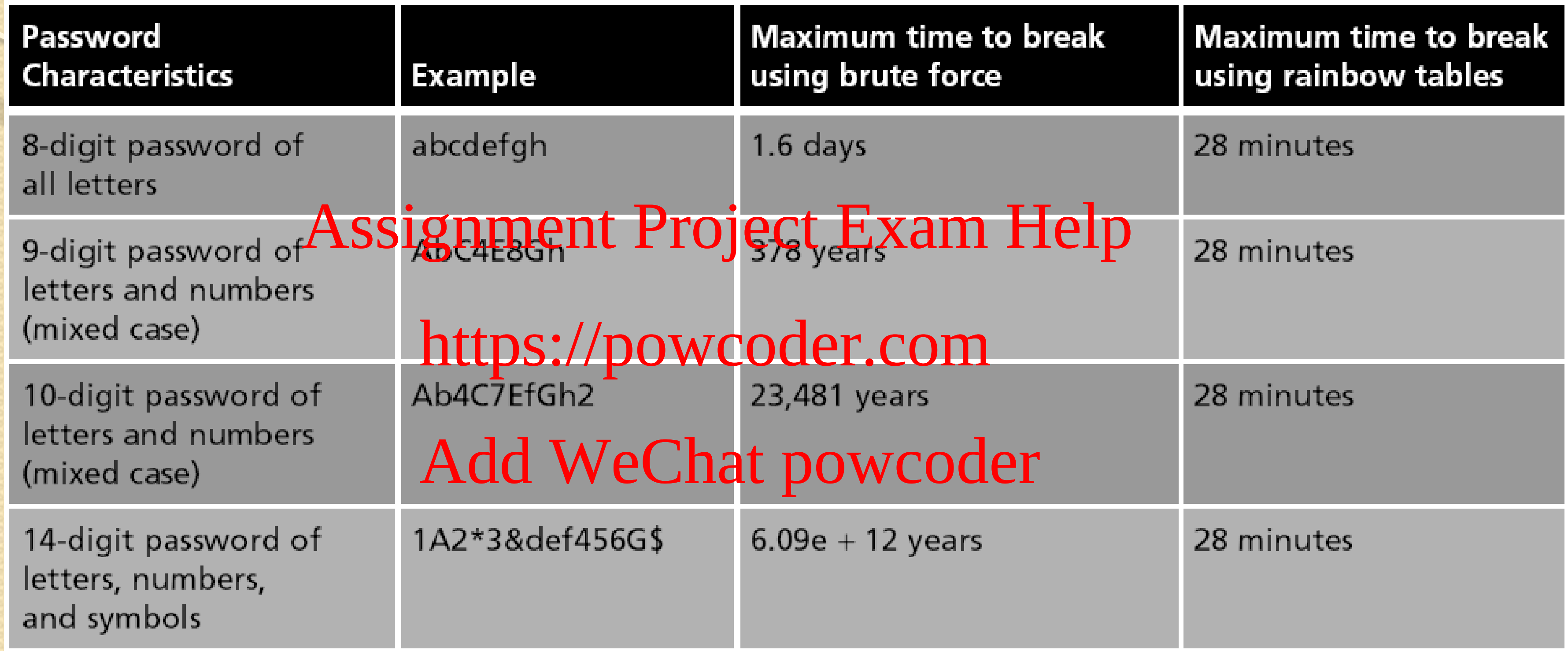
- **Pre-Computed Dictionary Attacks**

◈ **achieves <u>TIME-SPACE tradeoff</u> by pre-computing a list of hashes of dictionary words**

- pre-computed hashes are compared against those in a stolen password file
- **<u>rainbow tables</u>**
  1) **pregenerated sets/lists of hashes – n*Gbyte size!!** 👎
  2) **allow extremely rapid searching** 👍



Password Cracking (cont.)



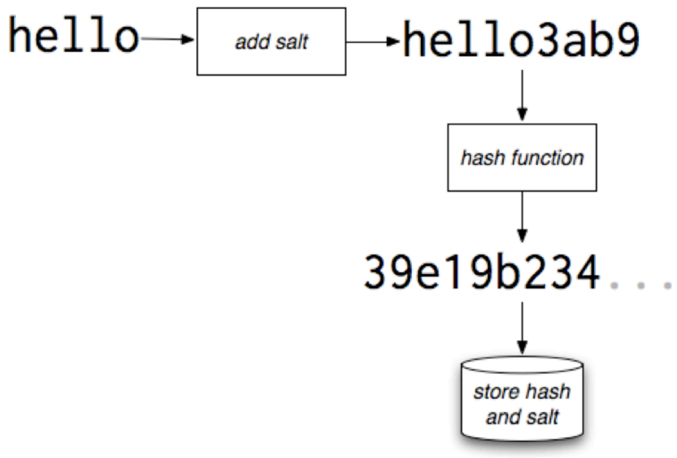
Password Characteristics	Example	Maximum time to break using brute force	Maximum time to break using rainbow tables
8-digit password of all letters	abcdefgh	1.6 days	28 minutes
9-digit password of letters and numbers (mixed case)	AbC4E8Gh	378 years	28 minutes
10-digit password of letters and numbers (mixed case)	Ab4C7EfGh2	23,481 years	28 minutes
14-digit password of letters, numbers, and symbols	1A2*3&def456G$	6.09e + 12 years	28 minutes

Table 7-5 Times to break a hash

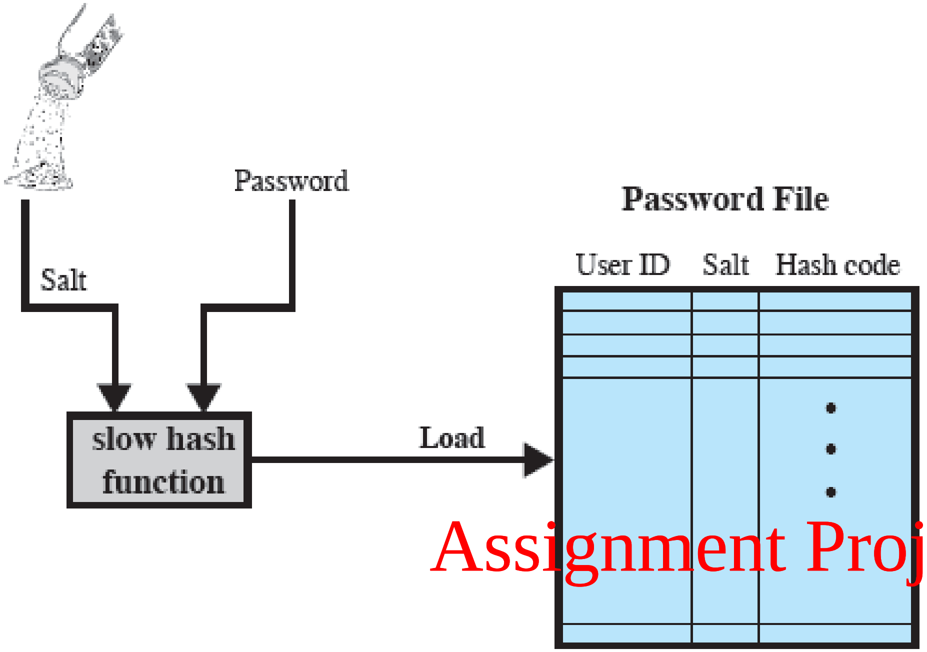
Rainbow Table: time/space tradeoff!!!

Password Salting

- **Password Salting** – **adding a unique random value to each password before hashing**
  - both the hash and salt are stored
  - does not fully prevent against password cracking, but makes it harder / more time consuming



https://spideroak.com/privacypost/cloud-security/byod-the-problems-with-plaintext-password-storage/



(a) Loading a new password

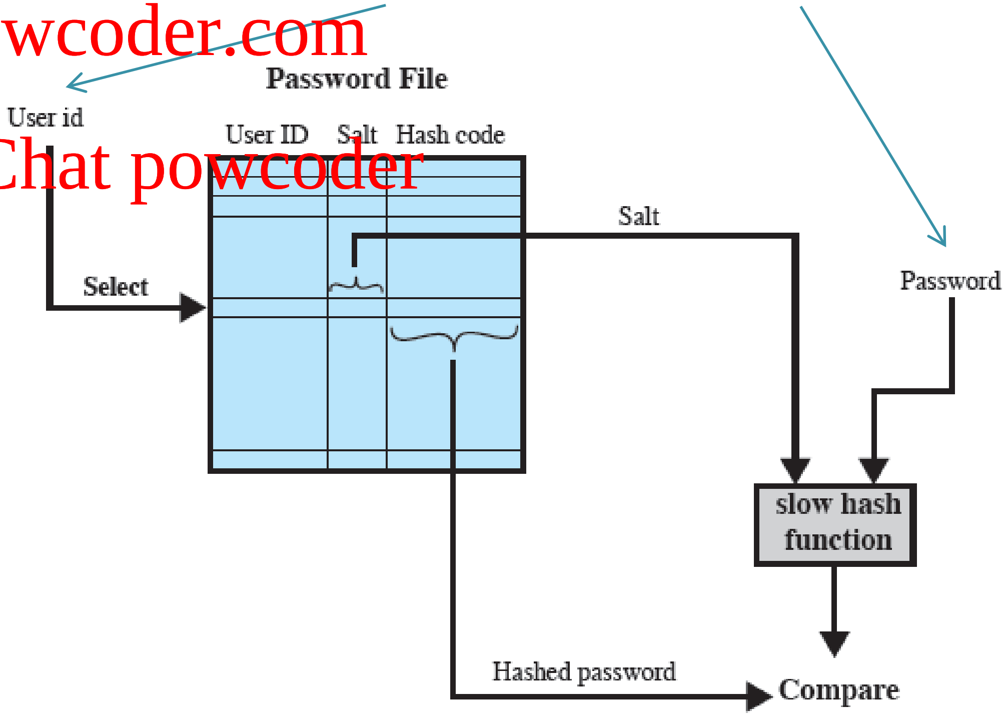
account creation stage

storing hash & salt instead of password

logging into an existing account:

testing a password against stored hash

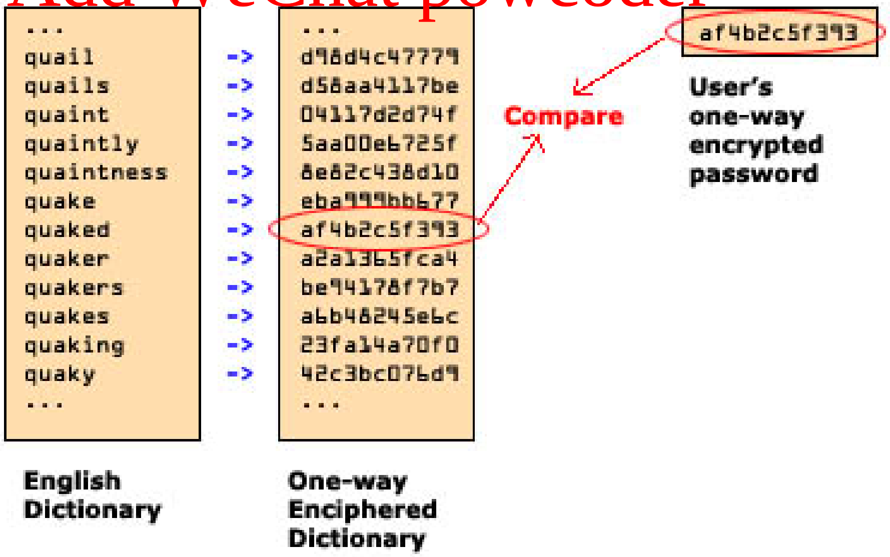
User id + Password



(b) Verifying a password

Password Salting (cont.)

- **Password Salting Benefits** – in case of a compromised Password File
  - dictionary and rainbow attacks impossible to perform
  - prevents duplicate passwords from being visible in password file
  - becomes impossible to find out whether a person has used the same password on multiple systems



Password Example

Example: Password policies – which one is better?!

Company A requires that its employees pick 6-character passwords made up of combinations of lowercase letters, uppercase letters, and digits. No other characters are allowed, and a given user's password must not use any character twice.

Example: ab98CD

Company B requires that its employees pick 12-character passwords, where each of the 12 can be any of 100 possible characters. Unlike for Company A, Company B's employees can reuse characters in their passwords. However, Company B finds that users often make mistakes with these long passwords, so if an authentication attempt fails, the login server helps the user by telling them how many of the initial letters were correct. For example, if a password entered was 'abcdefgij' and the server replies "*Wrong, but the first 4 letters were correct*", then 'abcd' are correct, 'e' is wrong, and nothing is revealed about the correctness of the letters after 'e'.

Password Example (cont.)

Example: Password policies – which one is better?!

Suppose an attacker is trying to guess/crack the password of user U1 at Company A, and user U2 at Company B. Both usernames are valid at the respective companies, and the users have chosen passwords that conform with the policy.

a) Write down an expression for the # of attempts the attacker needs for guessing the password of user U1 at Company A.

Solution:

Example: ab98CD

Total # of allowed characters = 26 + 26 + 10 = 62

Total # of possible passwords = 62*61*60*59*58*57 = $= 4.4 * 10^{10}$

Password Example (cont.)

Example: Password policies – which one is better?!

b) Write down an expression for the # of attempts the attacker needs for guessing the password of user U2 at Company B.

Solution:

The key for this part of the problem is that the attacker can use feedback provided by the login process to speed up the ‘cracking’ process.

To start, the attacker can try 100 passwords that each differ in their first character. One of these must succeed. In addition, when it succeeds, in the worst case the attacker is told that the second character in the attempted password is incorrect. Therefore, once the attacker learns that the first character is correct, they also can eliminate 1 of the possibilities for the second character.

Password Example (cont.)

Example: Password policies – which one is better?!

Password: bszi1289AMLK

1st round of guesses: aa, ba, ca, da, …

2nd round of guesses bba, bca, bda, bea, …, bsa, bta, …

At this point, they make another 100 − 1 = 99 guesses, each of which uses the first character learned in the previous step, and tries a different second character (excluding the character that the attacker has already learned is not correct for the second position).

This process continues until they try candidates for all 12 positions, requiring at worst a total of:

# of possible passwords = 100+99+99+ …
= $100 + 99 \cdot 11 = 1189$

You are not required to study the remaining slides!

They are provided only for your reference!

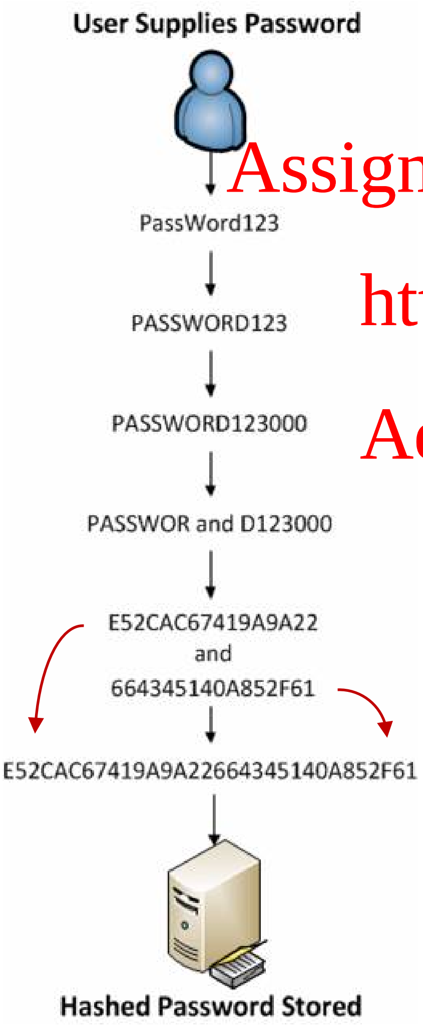
Password Hashing in Windows

- **Password Hashing in Windows** – **Windows-based computers utilize two hashing methods**
  - **LAN Manager (LM)**
    - used in earlier versions – up until Windows 2000, XP, Vista, and 7
  - **NT LAN Manager (NTLM)**
    - much stronger and harder to crack than LM hash
    - used in Windows 2000, XP, Vista, and 7
    - Windows 2000 and XP are also backward compatible – hash with both, to be able operate with older clients/servers*

* feature that should be disabled if not necessary

Password Hashing in Windows (cont.)

- **LM Password Hashing** – **not really a hash, but a a cryptographic value**



User Supplies Password

PassWord123

PASSWORD123

PASSWORD123000

PASSWOR and D123000

E52CAC67419A9A22
and
664345140A852F61

E52CAC67419A9A22664345140A852F61

Hashed Password Stored

1) user password is converted to all uppercase

2) password has null characters added to it until it equals 14 characters

3) new password is split into two 7 char. halves

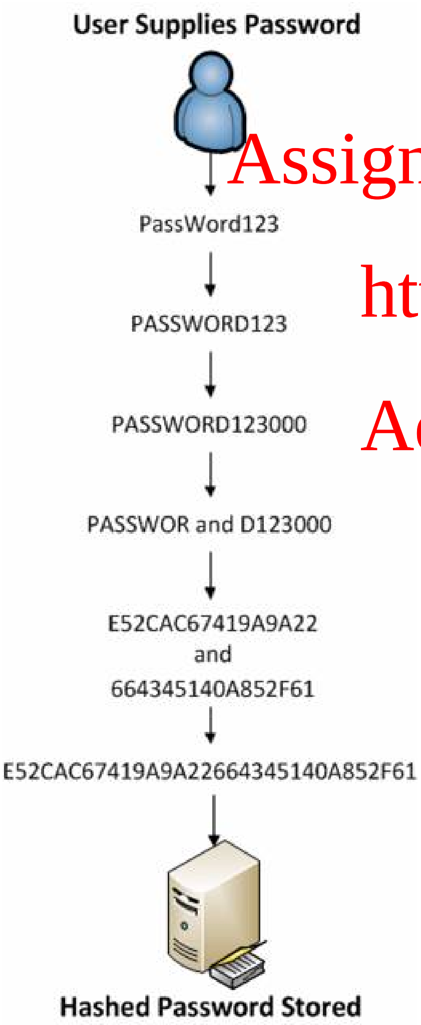
4) two 7-byte halves are used to create two 64-bit (8-byte) long DES encryption keys, by inserting a null bit after every seven bits

5) each key is used to DES-encrypt the constant ASCII string "KGS!@#$%", resulting in two 16-byte long ciphertext values

6) finally, two 16 byte hashes are concatenated to form the 32-byte long hash

Password Hashing in Windows (cont.)

- **LM Password Hashes (cont.)**



User Supplies Password

PassWord123

PASSWORD123

PASSWORD123000

PASSWOR and D123000

E52CAC67419A9A22 and 664345140A852F61

E52CAC67419A9A22664345140A852F61

Hashed Password Stored

- drawbacks:

1) case insensitive – significantly reduces character set that attacker must use (A – from 95 down to 69)

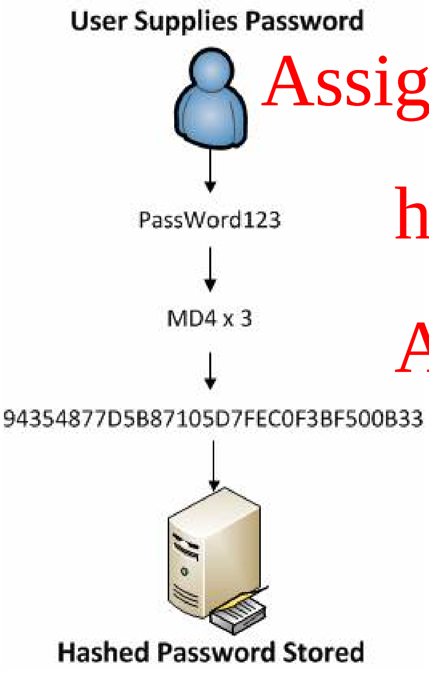
2) 14-character long passwords split into two 7-character long halves – search space dramatically reduced (from A^{14} to $2*A^{7}$)

total reduction in search space: from 95^{14} to $2*69^{7}$

3) DES encryption not considered safe anymore

Password Hashing in Windows (cont.)

- **NTLM Hashing** – much simpler in terms of OS operations than LM
  - ◈ applies MD4 hash algorithm 3 times
  - ◈ advantages: much 'stronger' than LN
    - ➢ allows for distinction between upper and lower case
    - ➢ does not split password into smaller, easier to crack, chunks
  - ◈ disadvantages: does not use 'salting' like in UNIX and Linux
    - ➢ salt – random combination of 0 & 1 added to a password
    - ➢ every bit of salt => 2X password-cracking demands on storage and/or computation


User Supplies Password
PassWord123
MD4 x 3
94354877D5B87105D7FEC0F3BF500B33
Hashed Password Stored