Lecture 9:

Access Control I:

Authentication

4th Year Course- CCSIT, UoA

Aim of Lecture

- To show the importance of including access control mechanisms in information systems
- To study most important system authentication techniques

Access Control

We'll use the term access control as an umbrella for any security issues related to access of system resources. Thus, there are two areas of primary interest:

1. **Authentication:** Are you who you say you are?
   - Determine whether access is allowed or not
   - Authenticate human to machine (or possibly machine to machine)
2. **Authorization:** Are you allowed to do that?
   - Once you have access, what can you do?
   - Enforces limits on actions

- *Note: "access control" sometimes used as synonym for authorization*

Authentication

- We are specifically here interested in system authentication as the process of determining whether a user (or other entity) should be allowed access to a system. This can be based on:

1. **Something you know** (For example, a password)
2. **Something you have** (For example, a smartcard)
3. **Something you are** (For example, your fingerprint)

- *Another type of authentication problem arises when the authentication information must pass through a network (This has been explored previously in the course).*

Something You Know

- Passwords: An ideal password is something that you know, something that a computer can verify that you know, and something nobody else can guess—even with access to unlimited computing resources. However, in practice it's difficult to even come close to this ideal
- Lots of things act as passwords!
  1. PIN
  2. Social security number
  3. Mother's maiden name
  4. Date of birth
  5. Name of your pet, etc.

Trouble with Passwords

- "Passwords are one of the biggest practical problems facing security engineers today."
- If left to their own devices, users tend to select bad passwords, which makes password cracking surprisingly easy
- From a security perspective, a solution to the password problem would be to instead use randomly generated cryptographic keys.
- However, humans are incapable of securely storing high-quality cryptographic keys
- Humans have unacceptable speed and accuracy when performing cryptographic operations.
- cryptographic keys are also large, expensive to maintain, and difficult to manage

Why Passwords?

- Despite their security problems, passwords are so popular. So, why is “something you know” more popular than “something you have” and “something you are”?

1. **Cost**: Passwords are free while smartcards and biometric devices cost money.

2. **Convenience**: Easier for system administrator to reset password than to issue a new thumb(!!!)

Crypto Keys vs. Passwords

Crypto keys

- For example, assume using key size of 64 bits
- Then the space is 2^{64} keys
- It is easy to choose key at random…
- Then attacker must try about 2^{63} keys before reaching a solution (brute force attack)

Passwords

- For example, assume using passwords of 8 characters with 256 possible choices for each character.
- Then the space is $256^8 = 2^{64}$ passwords
- Users do not select passwords at random
- Attacker has far less than 2^{63} passwords to try **(dictionary attack)**

Dictionary Attack 1

- users don't select passwords at random, because users must remember their passwords.
- As a result, a user is far more likely to choose an 8-character dictionary word such as “computer” than, say, “kf&Yw!a[“
- So, for example, a carefully selected dictionary of 2^{20} (about 1,000,000 passwords) would likely give attacker a reasonable probability of cracking a given password.
- On the other hand, if attacker attempted to find a randomly generated 64-bit key by trying only 2^{20} possible keys, the chance of success would be a mere $2^{20}/2^{64} = 1/2^{44}$, or less than 1 in 17 trillion.
- The bottom line is that the non-randomness of password selection is at the root of the problems with passwords.

Good and Bad Passwords Examples

Bad passwords

- frank
- Fido
- Password
- incorrect
- Pikachu
- 102560
- AustinStamp

Good Passwords?

- jflej,43j-EmmL+y
- 09864376537263
- P0kem0N
- FSa7Yago
- 0nceuP0nAt1m8
- PokeGCTall150

Password *passphrase*

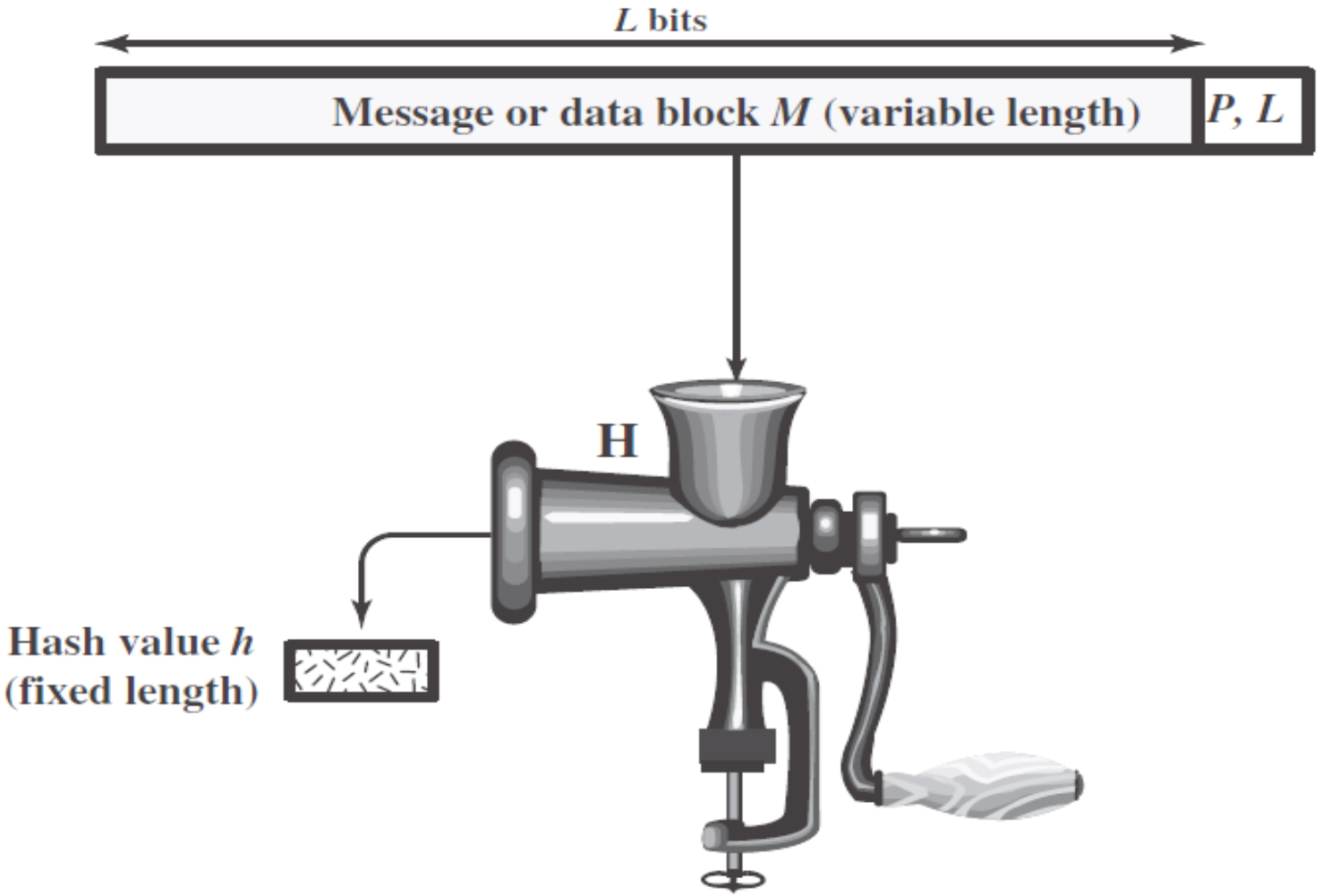
- The password *FSa7Yago* might appear to reside in the difficult to guess, but too difficult to remember category. However, there is a trick to help the user remember it—it's based on a passphrase.
- That is, *FSa7Yago* is derived from the phrase "*four score and seven years ago*."
- Consequently, this password should be relatively easy for Alice to remember, and yet relatively difficult for Trudy to guess.
- passphrases provide good option for password selection, since the resulting passwords are relatively difficult to crack yet easy to remember.

Password Retry

- Suppose system locks after 3 bad passwords. How long should it lock?
  1. 5 seconds?
  2. 5 minutes?
  3. Until administrator restores service?
- Five seconds might be insufficient to deter an automated attack. If it takes more than five seconds for attacker to make three password guesses for every user on the system, then he/she could simply cycle through all accounts, making three guesses on each
- On the other hand, five minutes might open the door to a denial of service attack, where attacker is able to lock accounts indefinitely by periodically making three password guesses on an account.
- The correct answer to this dilemma is not readily apparent.

Cryptographic Hash Function

- A **hash function *H*** accepts a variable-length block of data *M* as input and produces a fixed-size hash value *h = H(M).* A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed and apparently random.
- The kind of hash function needed for security applications is referred to as a **cryptographic hash function.**
- **A cryptographic hash function** is an algorithm for which it is computationally infeasible to find either:
  1. a data object that maps to a pre-specified hash result (the one-way property) or
  2. two data objects that map to the same hash result (the collision-free property).



P, L = padding plus length field

Cryptographic Hash Function; h = H(M)

[the figure is from the textbook of Stallings]

Password Verification (1)

- For a computer to determine the validity of a password, it must have something to compare against.
- It is a bad idea to store actual passwords in a file for verification
- It might be tempting to encrypt the password file with a symmetric key. However, to verify passwords, the file must be decrypted, so the decryption key must be as accessible as the file itself. Consequently, if Trudy can steal the password file, she can probably steal the key as well.
- Consequently, encryption is of little value here..

Password Verification (2)

- One possible solution is **Hashing** passwords. This solution is based on the one-way property of cryptographic hash functions to protect the passwords
- So, instead of storing raw passwords in a file, it's more secure to store hashed passwords, i.e.

$$y = H(password)$$

- Then when someone claiming to be Alice enters a password x, it is hashed and compared to y, and if y = H(x) then the entered password is assumed to be correct and the user is authenticated.

Password Verification (3)

- The advantage of hashing passwords is that if attacker obtains the password file, he/she does not obtain the actual passwords—instead he/she only has the hashed passwords.
- Of course, if attacker knows the hash value *y*, he/she can conduct a forward search attack by:
  1. guessing likely passwords *x*
  2. Check *y* until finding an *x* for which $y = H(x)$
  3. at which point he/she will have cracked the password.
- But at least attacker has work to do after he/she has obtained the password file.