O/S & Access Control

Aggelos Kiayias

One system Many users

- Objects that require protection
  - Memory
  - I/O devices (disks, printers)
  - programs and subprocedures
  - networks
  - data

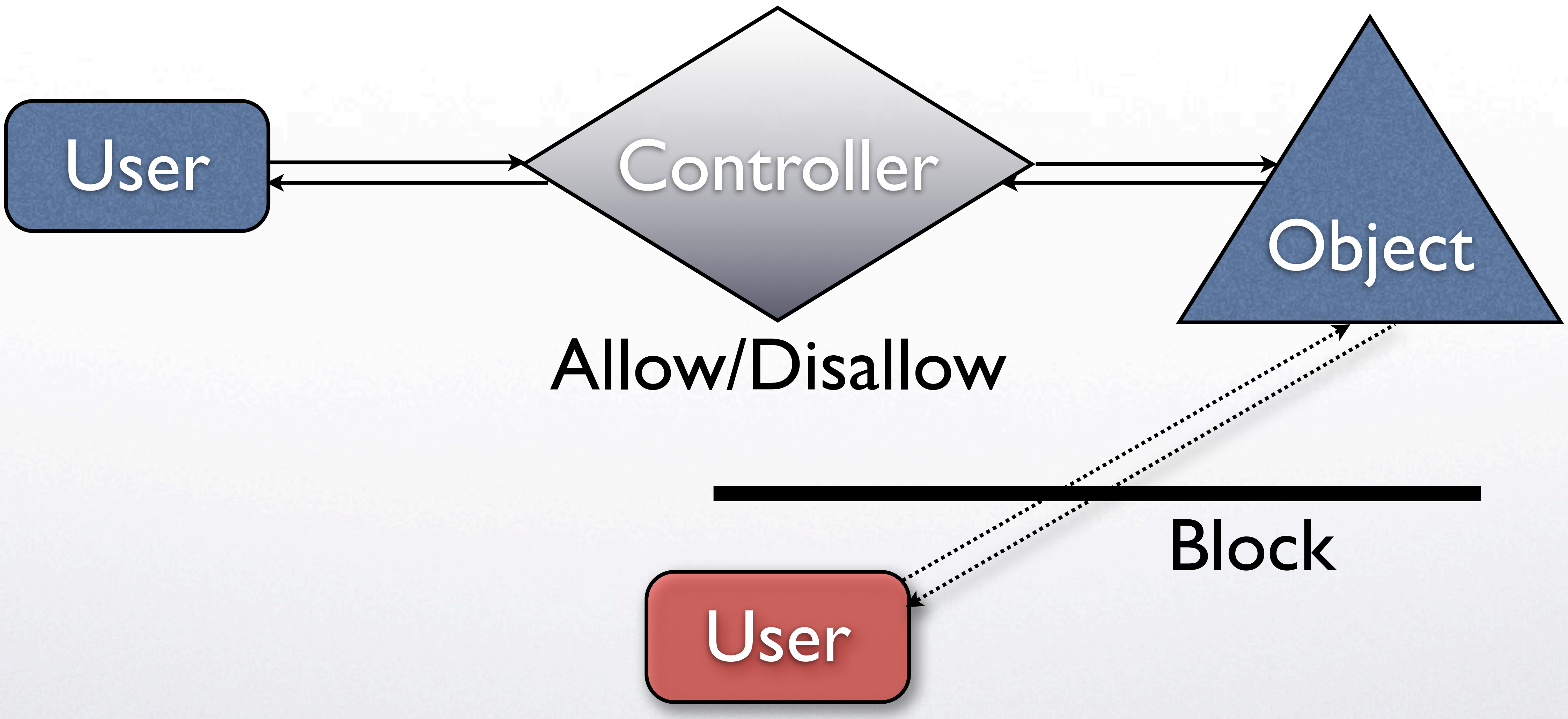
Separation

- keeping one user's objects separate from others.
  - Physically
  - Temporally
  - Logically
  - Cryptographically

Sharing is inevitable

- Users still need to share resources:
  - memory, CPU time, disks
  - Two extremes:
    Monolithic: single user owns all approach.
    Isolation: multiple virtual personality disorder.

Access Control



Focus: file system

- A paradigm for access control.
- Many objects can be thought as files (unix).

Access Control List

- Each object has an ACL.

Directory

File1	o
File2	o
File3	o

pointer

File1 ACL

User A	ORW
User B	RW
Other	R

In Unix

- Processes make requests.
- Each process has a **uid**.
- Each file has an ACL that contains a triple of **rwx rwx rwx**
  user group other
- The ACL contains user and group info.
- **x** is execute for files and access for dirs.

Temp Acquired Permission: suid bit

- How is it possible to allow a certain uid to peep into a higher access role through an executable?
- When an executable has the suid bit set, the file when it is executed it inherits the uid of its owner rather than the uid of the caller. E.g.,

```
-rwsr-xr-x  1 root root 32680 2005-10-11 12:13 passwd
```

Windows NTFS (5+)

- ACL is stored with every file.
- Contains users and groups and corresponding permissions for each.
  - Folder permissions: Read, Write, List, Read & Execute, Modify, Full Control.

Processes

- Objects are accessed by processes.
- How does a process acquire a user id?
  - It is created by a parent process and inherits the user id.
  - But how does a user access a system?

Login Process (console)

- Prompt process: *invites user* (cf., getty)
- Login process: *challenges user to authenticate*.
  - if login is unsuccessful restart the prompt
  - if successful an interface process is spawned that inherits the uid/gid of the authenticated user.

Separation?

- in a multi-user environment
    - Access-control as described so far offers a logical separation; is this foolproof?
    - What would a cryptographic separation offer?

User Authentication

- Can be based on
  - *Something the user knows.*
  - *Something the user has.*
  - *Something the user is.*

Password-based Auth

- Authentication based on what user *knows*.
- O/S must keep a database of username/ password pairs.
- Where to store it?
- What to store?

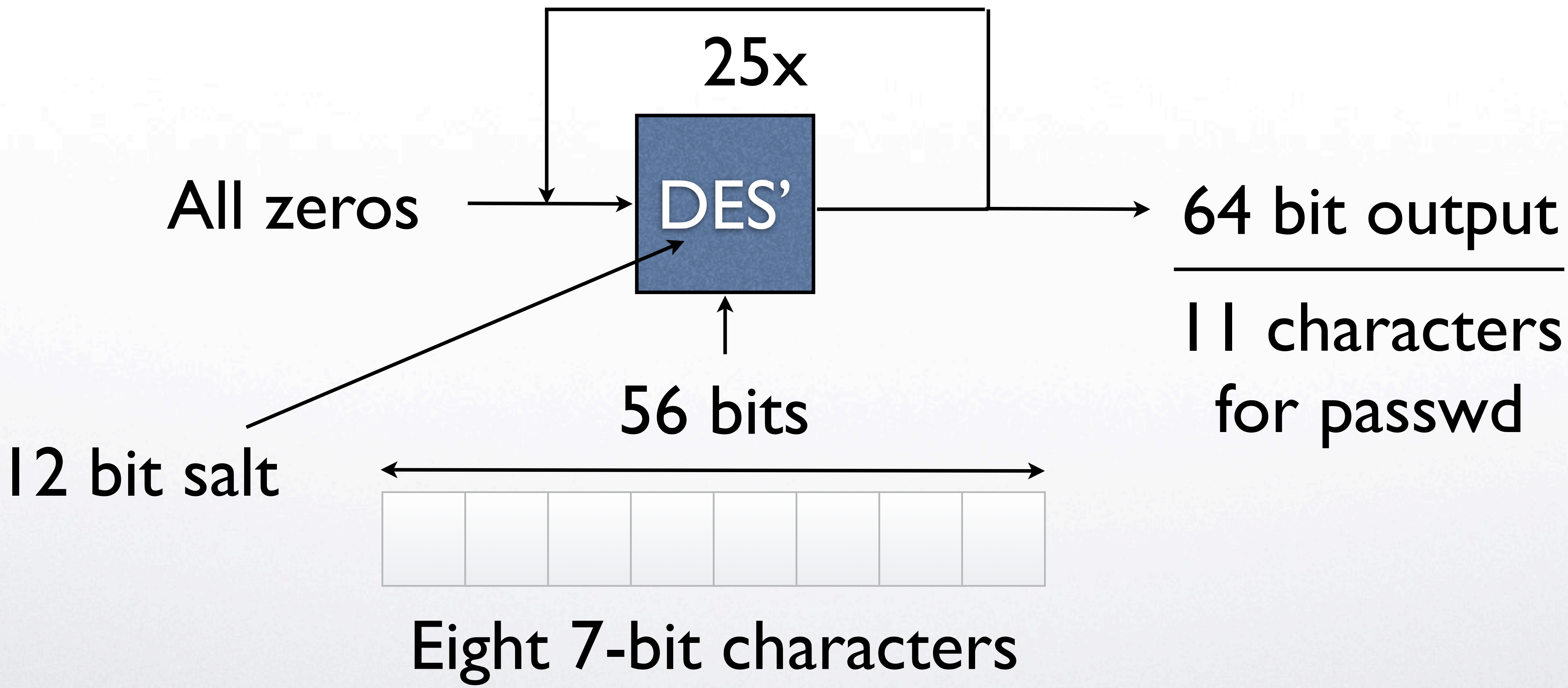
The /etc/passwd file in UNIX

- FORMAT
  Name:Password: UserID:PrincipleGroup:Gecos: HomeDirectory:Shell
- guest:AvCSyg9e75YZM:200:0::/home/guest:/usr/bin/sh
- One line for each user.
- The file is publicly readable.
- In current deployments the password file is *shadowed* in another location (e.g., /etc/shadow) -- this file is not publicly readable.

Storing passwords

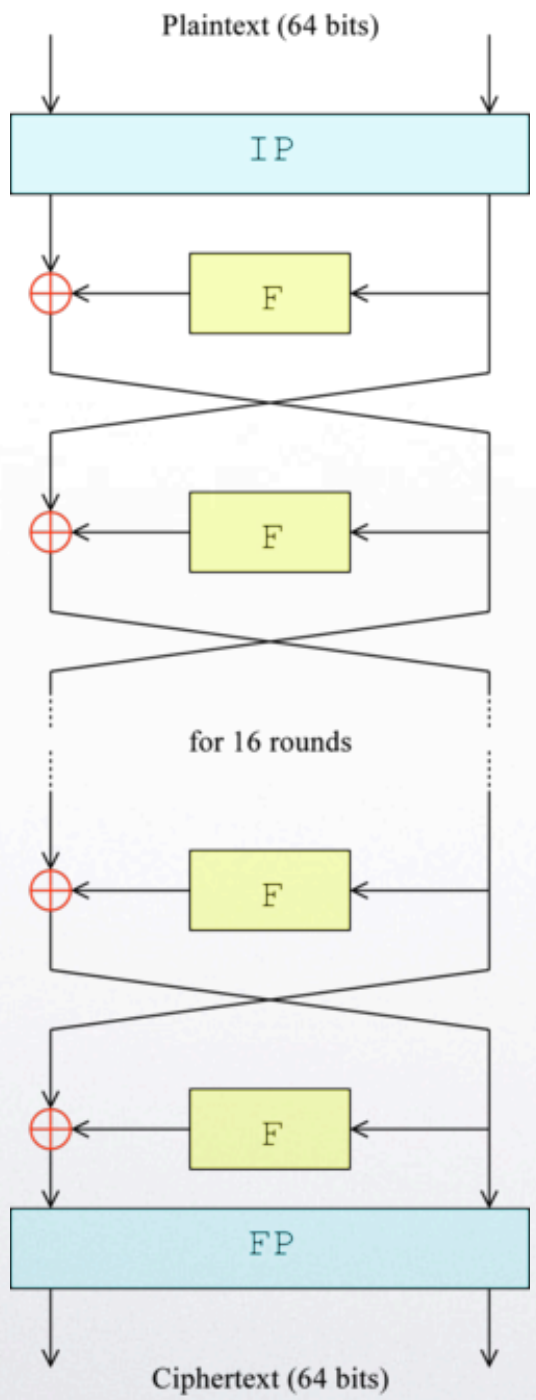
- Should the passwords be stored into the passwd?
  - Use one-way transformations.

The crypt() function



DES

Feistel Network



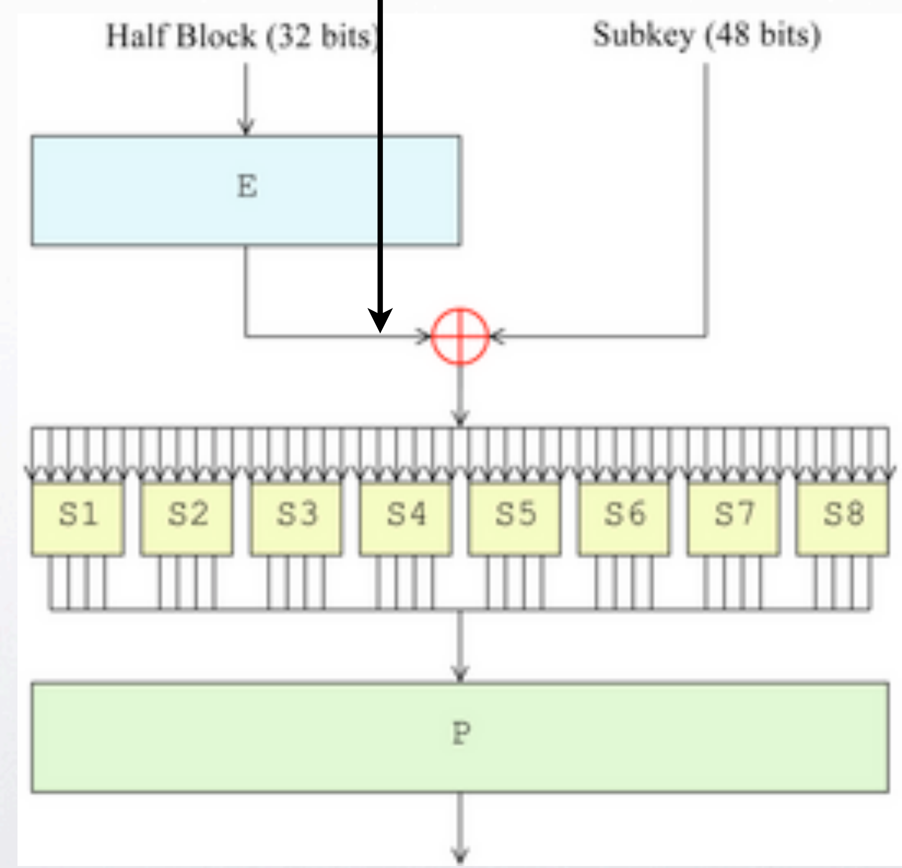
<u>salt-based permutation</u>

If (Bit 1) then swap(1,25)

If (Bit 2) then swap(2,26)

...

F-function:



Examples

- crypt(”password”,”Ee”) =EeAJqAJ0sluG.
- crypt(”password”,”4!”) = 4!wpbYhg6W8qM
- crypt(”password is what some people choose but I chose a passphrase!”,”4!”) =4!wpbYhg6W8qM

The glibc2 extension

- If salt starts with 1 followed by at most 8 characters, terminated by $.
- MD5 based algorithm with 22 char output from [a-zA-Z0-9./].
- entire password is significant.

Examples

- crypt("password","1GoodSalt") =

  1GoodSalt$czxN1PirYBY5pqE1Q98el.

- crypt("password is what people choose but I chose a passphrase","1GoodSalt") = 1GoodSalt$Obp/S5k35O0r1ymT0v9t./
  - to test the command: perl -e 'print(crypt("test","\$1\$abcd\$")."\n");'

In Windows?

- Security Accounts Management Database (SAM) stored in the registry.
- It stores hashed copies of user passwords.
- The database itself is encrypted with a locally stored system key.
- It is possible to store this key elsewhere.
    - Attack against NT4.0, 2000 if SAM was deleted one can gets a free login.

Dictionary Attacks

- Given dictionary of possible passwords:
  - For each dictionary entry & salt:
    - Apply the one-way transformation.
    - search the password file for a match.
- *Online cost high. Salting is useless against this attack.*

check John the Ripper password cracker http://www.openwall.com/john/

Codebook Dictionary

- Produce "codebook dictionary"
  - Apply one-way transformation to each candidate pwd.
- Sort according to transformation.

Off-line!

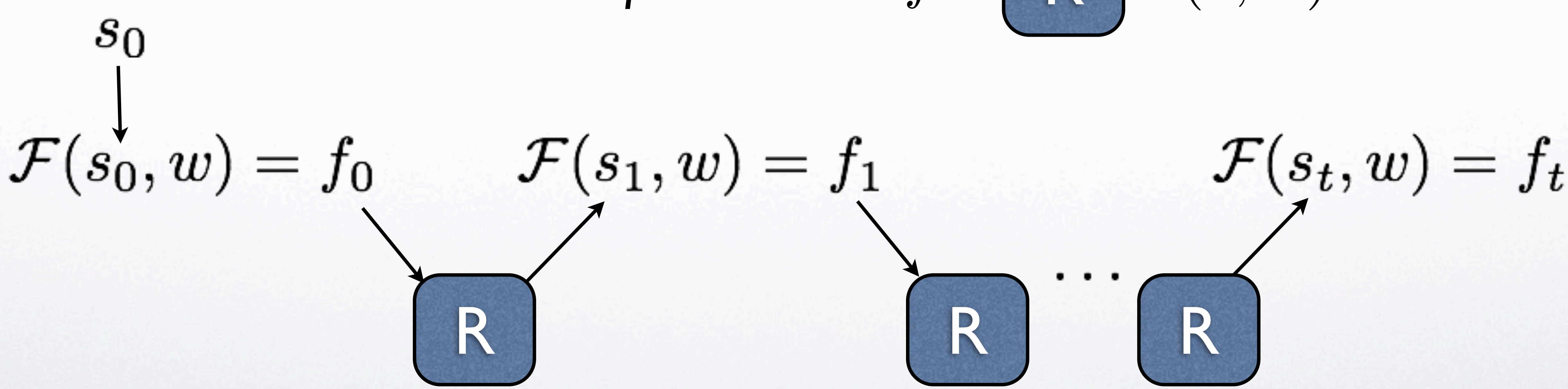
- For each entry of the password file search the codebook dictionary.
- *online: super-fast!* <u>*but salting can really make a difference against this attack.*</u>

logarithmic

Time-Memory Tradeoff

- Can we do something in between the previous two approaches?

Consider a function R: $f \xrightarrow{\text{R}} (s, w)$



Storage reduction: $\underline{w,} \langle s_0, f_0 \rangle, \langle s_1, f_1 \rangle, \ldots, \langle s_t, \underline{f_t} \rangle$

Time-Memory Tradeoff, II

Sort codebook dictionary according to end of chain

Given $f = f[0]$ calculate $f[1], \ldots, f[t]$ applying R

Perform binary search in the codebook dictionary
for each of $f[1], \ldots, f[t]$
every chain hit gives a candidate password

Tradeoff: Dictionary size has been reduced by size ~ t
searching time has been multiplied by ~ t

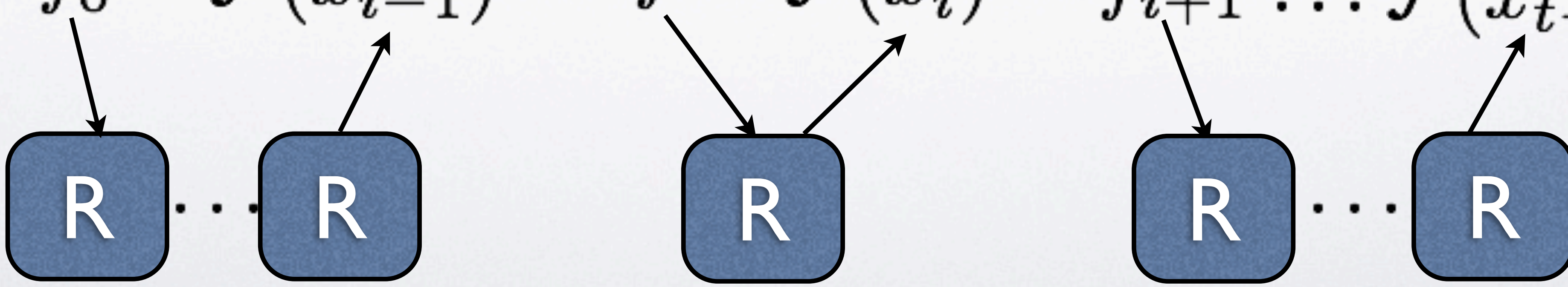
Time-Memory Tradeoff, V

- How to recover the password after you hit end of chain?

- Start from the beginning of chain.

$$x_0$$

$$\mathcal{F}(x_0) = f_0 \cdots \mathcal{F}(x_{i-1}) \stackrel{?}{=} f \quad \mathcal{F}(x_i) = f_{i+1} \ldots \mathcal{F}(x_{t-1}) = f_t$$



Time-Memory Tradeoff, III

- Tight tradeoff is contingent on a good choice of R
- Too few/short chains may not cover the full dictionary.
- Too many/long chains will overlap and waste space/time.

Time-Memory Tradeoff, IV

- Even possible to model $\boxed{R}$ to produce "human" passwords, i.e., consider those chains for which it holds that *w* follows a certain distribution

Time-Memory Tradeoff, VI

- A very powerful technique.
- If applied against unsalted hashes can break any **strong** human memorizable password.
- Implementations: *Ophcrack*, *RainbowCrack*.
- ***Random*** Windows NT Lan Manager passwords can be broken in 13 seconds with 1.4 GB tables. [*Oeschlin* CRYPTO ‘03]

Choosing a Dictionary

- Without salting one is totally vulnerable (even with **random** *but of human-memorizable length* passwords).
- With salting things get better. But still a good starting dictionary can lead to a devastating attack [we can leverage salting with a good time-memory trade-off].

Language Entropy

http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf

Language Entropy

- Shannon : *entropy* of English language is about 4.6 --1.5 bits per character for 8-char long.

http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf

Language Entropy

- Shannon : *entropy* of English language is about 4.6 --1.5 bits per character for 8-char long.
- How much entropy do Human memorizable passwords have?

http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf

Language Entropy

- Shannon : *entropy* of English language is about 4.6 --1.5 bits per character for 8-char long.
- How much entropy do Human memorizable passwords have?
- NIST: Using an allowed 94 character alphabet.

http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf

Language Entropy

- Shannon : *entropy* of English language is about 4.6 --1.5 bits per character for 8-char long.
- How much entropy do Human memorizable passwords have?
- NIST: Using an allowed 94 character alphabet.
  - 8 chars: only 18 bits! vs. 52 bits for random

http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf

Language Entropy

- Shannon : *entropy* of English language is about 4.6 --1.5 bits per character for 8-char long.
- How much entropy do Human memorizable passwords have?
- NIST: Using an allowed 94 character alphabet.
  - 8 chars: only 18 bits! vs. 52 bits for random
  - 40 chars gave 56 bits.

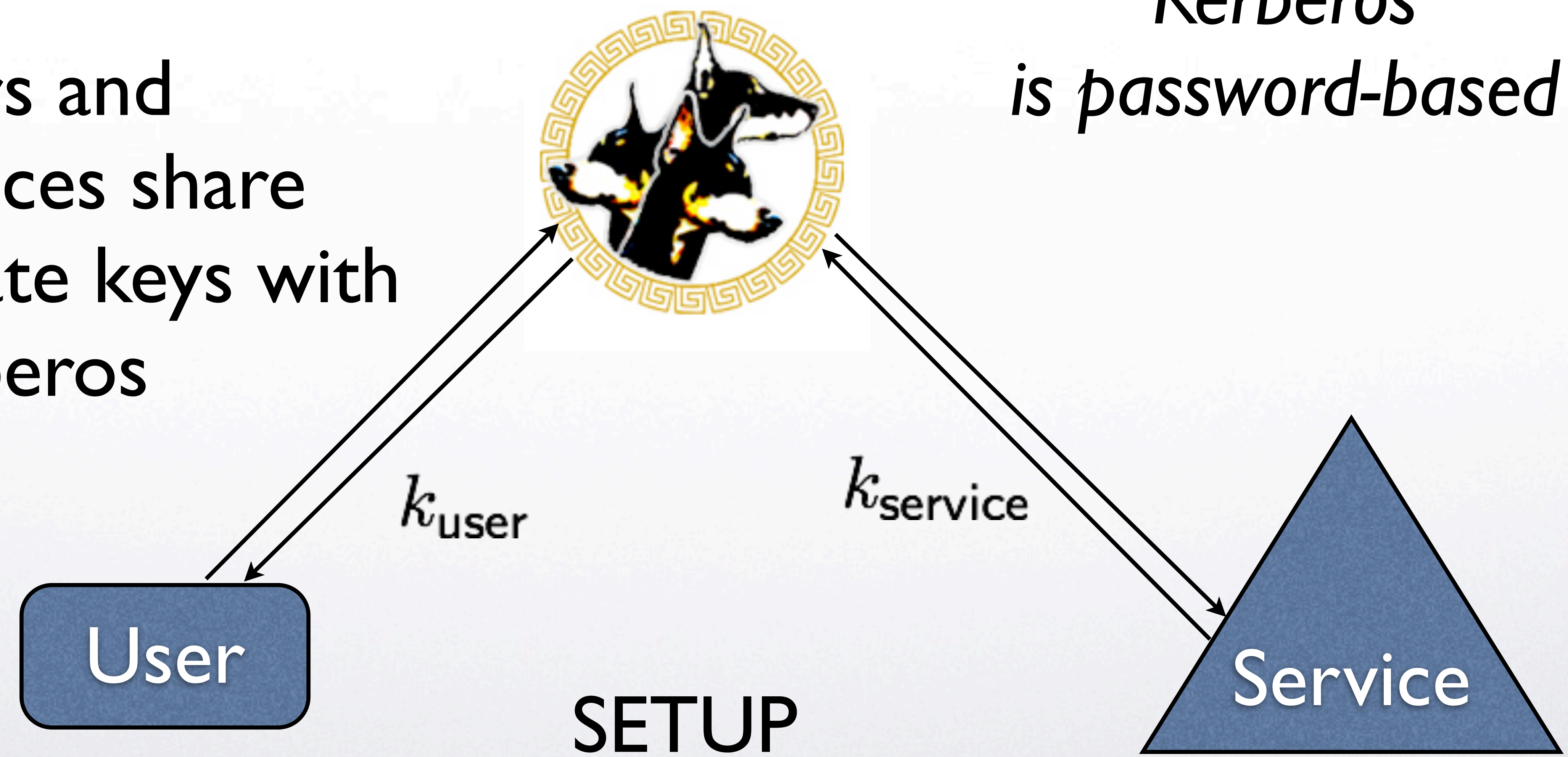
http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf

Kerberos

The Kerberos Approach

Users and services share private keys with Kerberos

Kerberos is password-based



Kerberos, II

K_{session}
Random key
selected on the fly

User: I want *service*

$\{\text{service}, K_{\text{session}}\}_{K_{\text{user}}}$

$\{\text{user}, \text{service}, \text{time}, \text{life}, K_{\text{session}}\}_{K_{\text{service}}}$

ticket

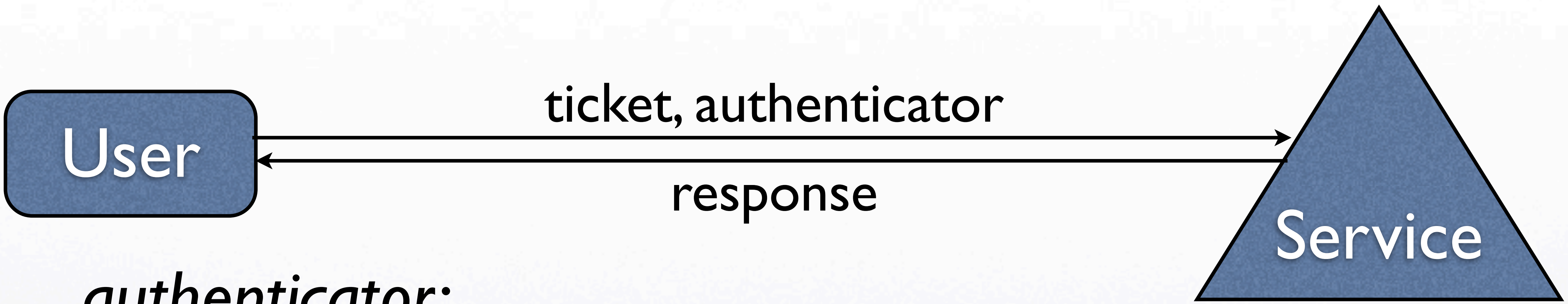
User

Can be used multiple times

Kerberos, III

ticket:

$\{\text{user, service, time, life}, K_{\text{session}}\}_{K_{\text{service}}}$



authenticator:

$\{\text{user, time}\}_{K_{\text{session}}}$

Kerberos, IV

- Above description too stressful for Kerberos.
- Easing Kerberos task:
  - Kerberos will recognize only one service, the *Ticket Granting Service*.
  - Instead of giving tickets for every service it will give tickets only for using the TGS.

Kerberos, V

can be reused

User: I want *TGS*

$\{\text{user}, K_{\text{TGSsession}}\}_{K_{\text{user}}}$

$\{\text{user}, \text{TGS}, \text{time}, \text{life}, K_{\text{TGSsession}}\}_{K_{\text{TGS}}}$

TGS-ticket

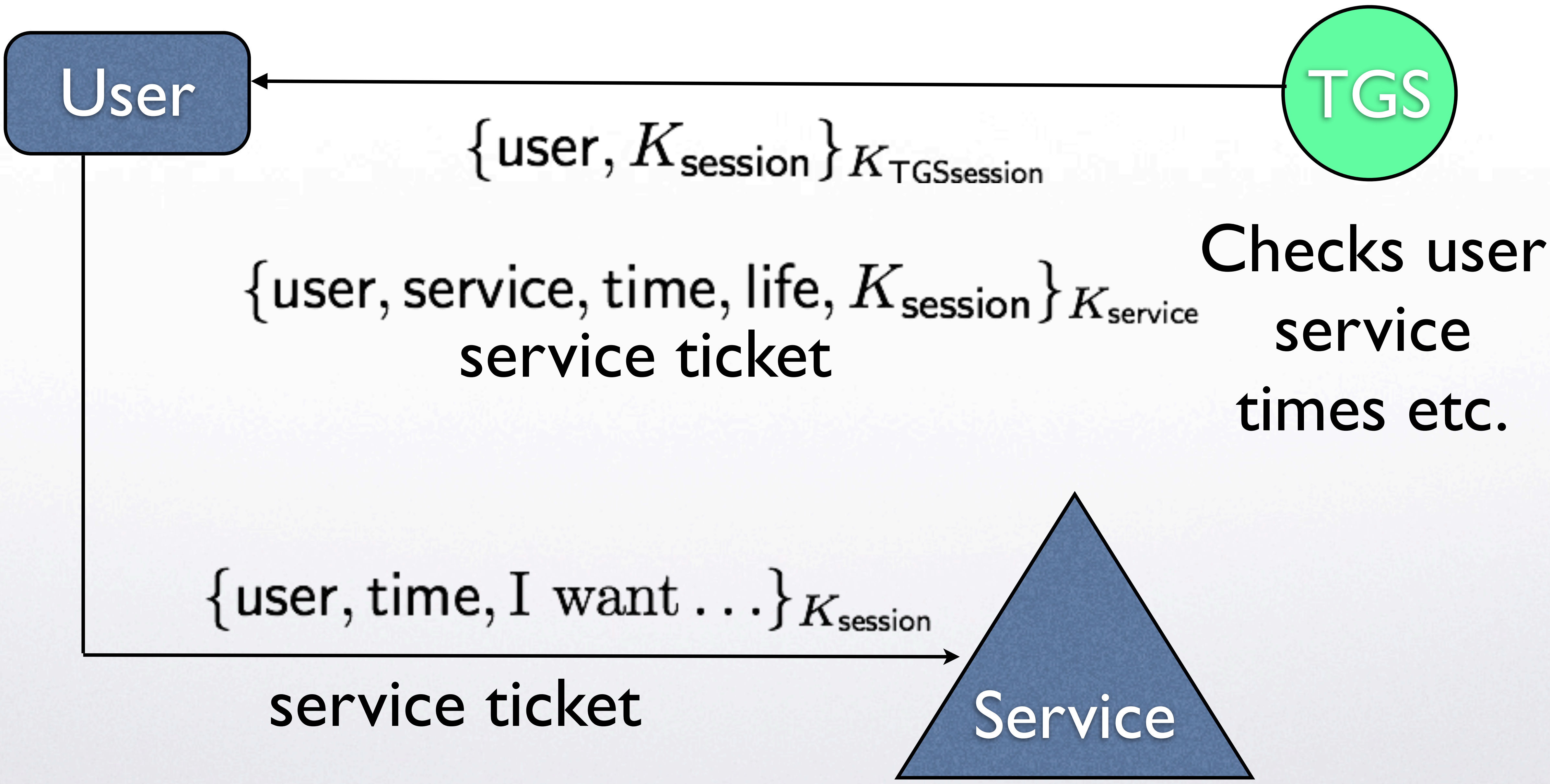
User

TGS

$\{\text{user}, \text{time}, \text{ I want service}\}_{K_{\text{TGSsession}}}$

TGS-ticket

Kerberos, VI


User
TGS
$\{\text{user}, K_{\text{session}}\}_{K_{\text{TGSsession}}}$
Checks user
service
times etc.
$\{\text{user}, \text{service}, \text{time}, \text{life}, K_{\text{session}}\}_{K_{\text{service}}}$
service ticket
$\{\text{user}, \text{time}, \text{I want} \ldots\}_{K_{\text{session}}}$
service ticket
Service

Kerberos VII

- Kerberos server knows all user keys and the TGS key. It handles user authentication.
- Ticket granting server knows service keys. It handles user requests to access services.
- Kerberos does not need to know about system services. TGS does not need to worry about authenticating users.

Kerberos VIII

- Where do keys come from?
  - user keys are derived from human passwords.
  - service keys are random and stored locally. Assumed to be stored securely.

Kerberos IX

- Kerberos advantages:
  - Human passwords are never communicated. Only on the fly usage by local "login" challenge.
  - Mutual authentication between users and services.
- Kerberos disadvantages:
  - monolithic

Kerberos X

- Windows (all the way since 2000) uses Kerberos for authentication services.
- Possible to install for Linux, Unix.
- Mac-OS X has built-in Kerberos support.