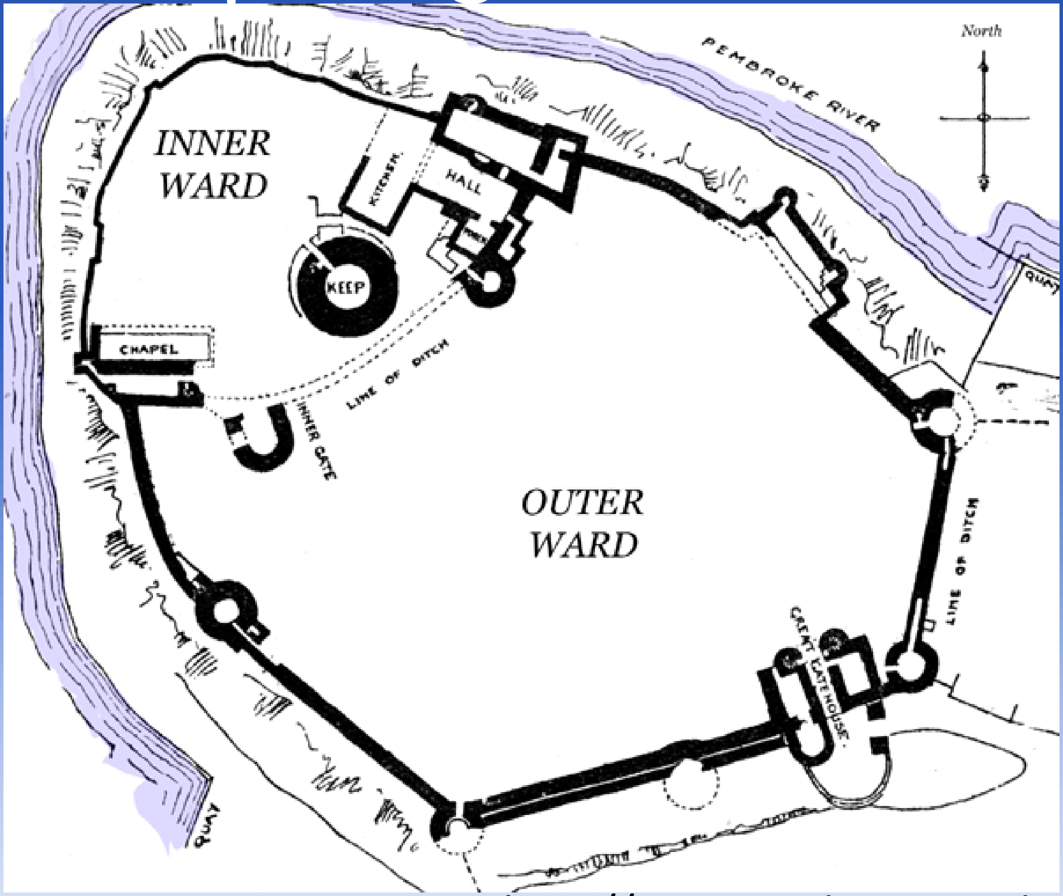
CS 493
Secure Software Systems
Trusted Computing



http://www.medart.pitt.edu

Dr. Williams Central Connecticut State University

Computer Security Models

Two fundamental computer security facts:

All complex software systems have eventually revealed flaws or bugs that need to be fixed

It is extraordinarily difficult to build computer hardware/software not vulnerable to security attacks

- Problems involved both design and implementation
- Led to development of formal security models
- Need for trusted platforms

Operating system security

OS and Security

- OSs are large, complex programs
  - Many bugs in any such program
  - We have seen that bugs can be security threats
- Here we are concerned with security provided by OS
  - Not concerned with threat of bad OS software
- Concerned with OS as security **enforcer**
- In this section we only scratch the surface

OS Security Challenges

- Modern OS is **multi-user** and **multi-tasking**
- OS must deal with
  - Memory
  - I/O devices (disk, printer, etc.)
  - Programs, threads
  - Network issues
  - Data, etc.
- OS must protect processes from other processes and users from other users
  - Whether accidental or malicious

OS Security Functions

- Memory protection
  - Protect memory from users/processes
- File protection
  - Protect user and system resources
- Authentication
  - Determines and enforce authentication results
- Authorization
  - Determine and enforces access control

Memory Protection

- Fundamental problem
  - How to keep users/processes separate?
- Separation
  - Physical separation — separate devices
  - Temporal separation — one at a time
  - Logical separation — sandboxing, etc.
  - Cryptographic separation — make information unintelligible to outsider
  - Or any combination of the above

Memory Protection

- **Fence** —— users cannot cross a specified address
  - Static fence —— fixed size OS
  - Dynamic fence —— fence register

- Base/bounds register —— lower and upper address limit
- Assumes contiguous space

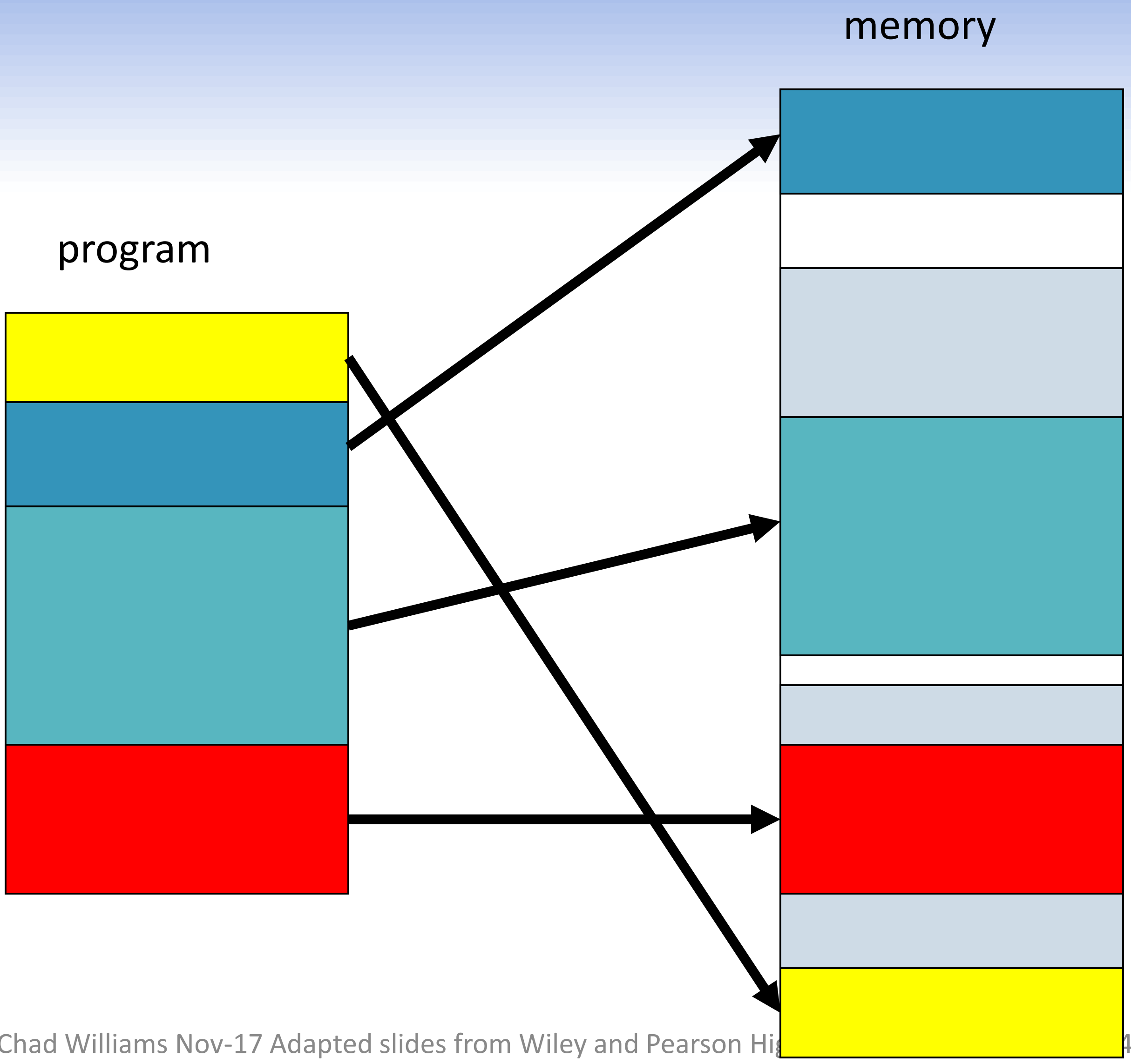
Memory Protection

- Tagging — specify protection of each address
  - + Extremely fine-grained protection
  - - High overhead — can be reduced by tagging sections instead of individual addresses
  - - Compatibility
- More common is segmentation and/or paging
  - Protection is not as flexible
  - But much more efficient

Segmentation

- Divide memory into logical units, such as
  - Single procedure
  - Data in one array, etc.
- Can enforce different access restrictions on different segments
- Any segment can be placed in any memory location (if location is large enough)
- OS keeps track of actual locations

Segmentation



Segmentation

- OS can place segments anywhere
- OS keeps track of segment locations as <segment,offset>
- Segments can be moved in memory
- Segments can move out of memory
- All address references go thru OS

Segmentation Advantages

- Every address reference can be checked
  - Possible to achieve **complete mediation**
- Different protection can be applied to different segments
- Users can share access to segments
- Specific users can be restricted to specific segments

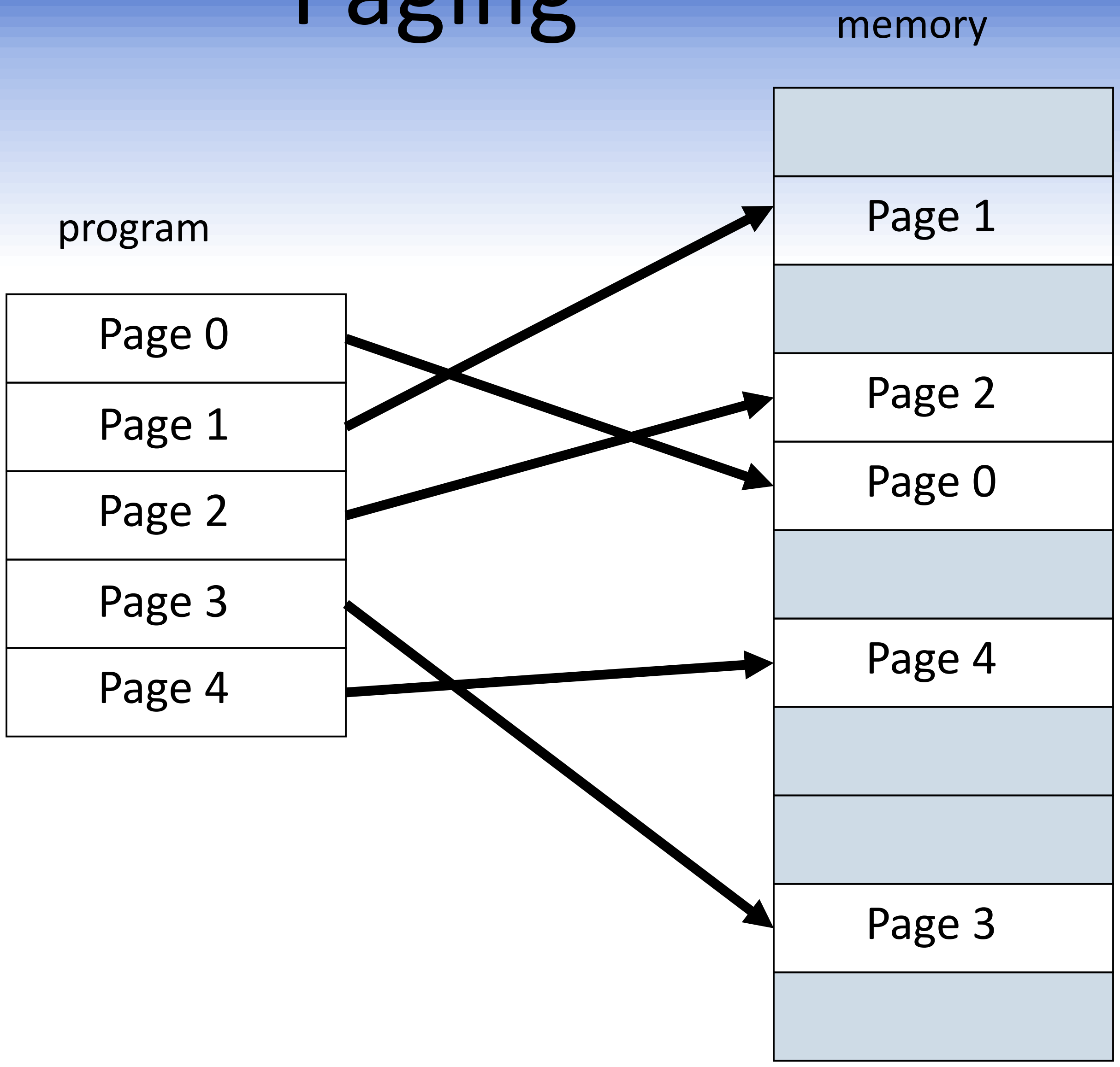
Segmentation Disadvantages

- How to reference <segment,offset> ?
  - OS must know segment **size** to verify access is within segment
  - But some segments can grow during execution (for example, dynamic memory allocation)
  - OS must keep track of **variable** segment sizes
- Memory fragmentation is also a problem
  - Compacting memory changes tables
- A lot of work for the OS
- More complex $\Rightarrow$ more chance for mistakes

Paging

- Like segmentation, but fixed-size segments
- Access via <page,offset>
- Plusses and minuses
  - + Avoids fragmentation, improved efficiency
  - + OS need not keep track of variable segment sizes
  - - No logical unity to pages
  - - What protection to apply to a given page?

Paging


memory
program
Page 0
Page 1
Page 2
Page 3
Page 4
Page 1
Page 2
Page 0
Page 4
Page 3

Other OS Security Functions

- OS must enforce access control
- Authentication
  - Passwords, biometrics
  - Single sign-on, etc.
- Authorization
  - ACL
  - Capabilities
- These topics discussed previously
- OS is an attractive target for attack!

Trusted Operating System

Trusted Operating System

- An OS is **trusted** if we rely on it for
  - Memory protection
  - File protection
  - Authentication
  - Authorization
- Every OS does these things
- But if a trusted OS fails to provide these, our security fails

Trust vs Security

- **Trust** implies reliance
- Trust is binary
- Ideally, only trust secure systems
- All trust relationships should be explicit

- **Security** is a judgment of effectiveness
- Judge based on specified policy
- Security depends on trust relationships

- Note: Some authors use different terminology!

Trusted Systems

- Trust implies reliance
- A trusted system is relied on for security
- An untrusted system is not relied on for security
- If all untrusted systems are compromised, your security is unaffected
- Ironically, **only a trusted system can break your security!**

Trusted OS

- OS mediates interactions between subjects (users) and objects (resources)
- Trusted OS must decide
  - Which objects to protect and how
  - Which subjects are allowed to do what

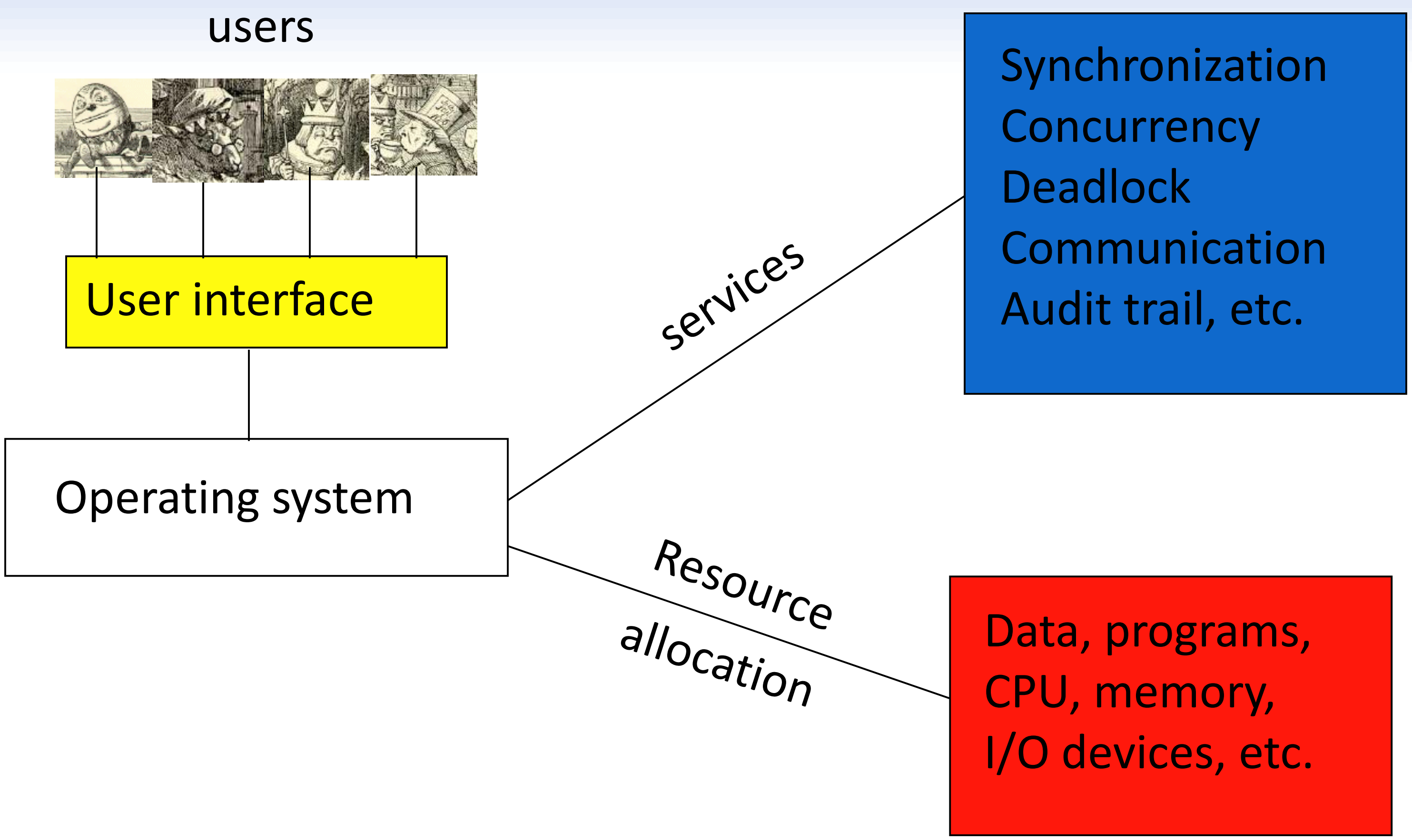
General Security Principles

- Least privilege — like “low watermark”
- Simplicity
- Open design (Kerchoffs Principle)
- Complete mediation
- White listing (preferable to black listing)
- Separation
- Ease of use
- But commercial OSs emphasize **features**
  - Results in complexity and poor security

OS Security

- Any OS must provide some degree of
  - Authentication
  - Authorization (users, devices and data)
  - Memory protection
  - Sharing
  - Fairness
  - Inter-process communication/synchronization
  - OS protection

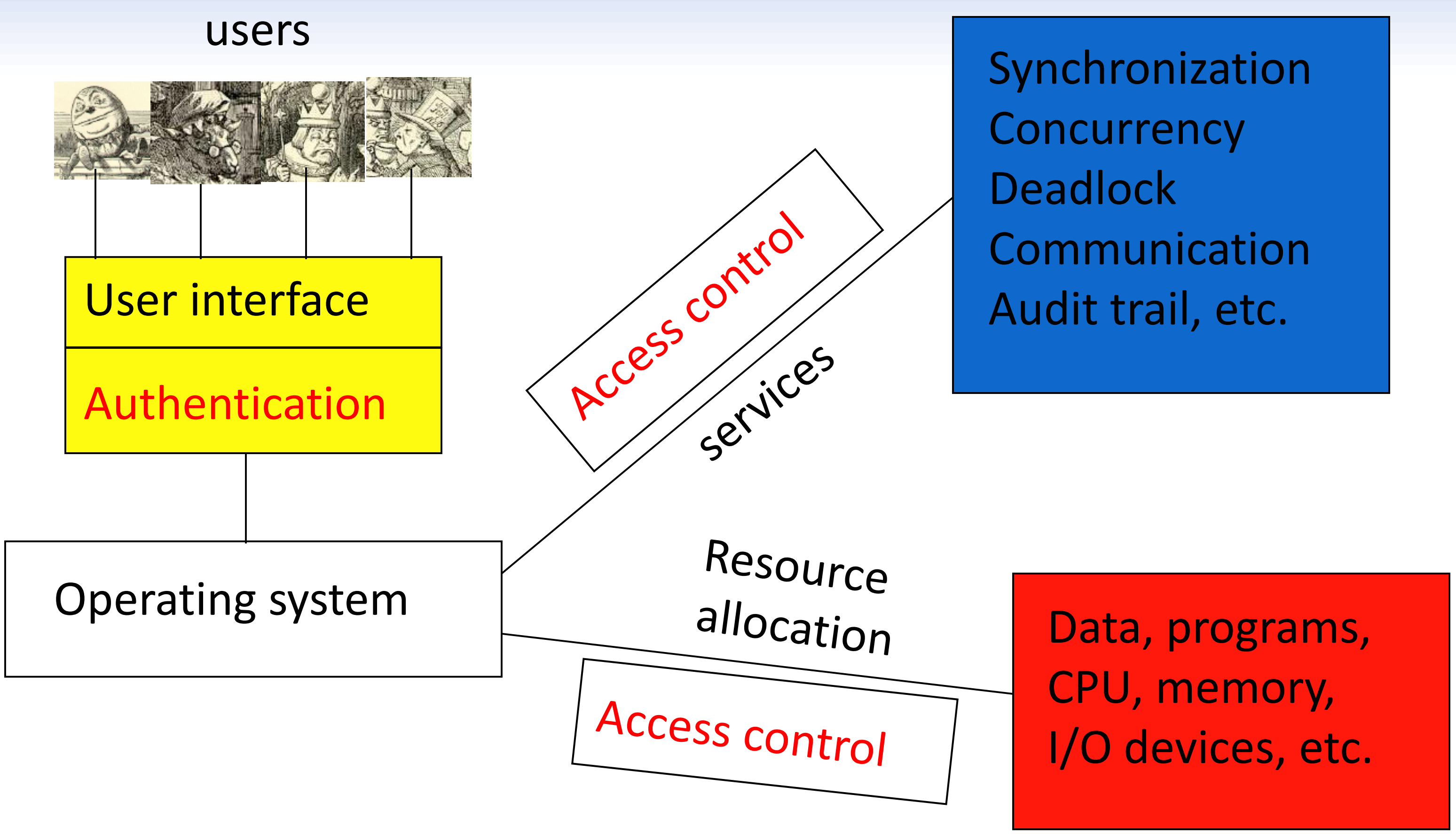
OS Services


users
User interface
Operating system
services
Resource allocation
Synchronization
Concurrency
Deadlock
Communication
Audit trail, etc.
Data, programs,
CPU, memory,
I/O devices, etc.

Trusted OS

- A trusted OS also provides some or all of
  - User authentication/authorization
  - Mandatory access control (**MAC**)
  - Discretionary access control (**DAC**)
  - Object reuse protection
  - Complete mediation — access control
  - Trusted path
  - Audit/logs

Trusted OS Services



MAC and DAC

- Mandatory Access Control (MAC)
  - Access not controlled by owner of object
  - Example: User does not decide who holds a **TOP SECRET** clearance
- Discretionary Access Control (DAC)
  - Owner of object determines access
  - Example: UNIX/Windows file protection
- If DAC and MAC both apply, MAC wins

Object Reuse Protection

- OS must prevent leaking of info
- Example
  - User creates a file
  - Space allocated on disk
  - But same space previously used
  - “Leftover” bits could leak information
  - Magnetic remanence is a related issue

Trusted Path

- Suppose you type in your password
  - What happens to the password?
- Depends on the software!
- How can you be sure software is not evil?
- Trusted path problem:

  "I don't know how to to be confident even of a digital signature I make on my own PC, and I've worked in security for over fifteen years. Checking all of the software in the critical path between the display and the signature software is way beyond my patience. "

  —— Ross Anderson

Audit

- System should log security-related events
- Necessary for postmortem
- What to log?
  - Everything? Who (or what) will look at it?
  - Don’t want to overwhelm administrator
  - Needle in haystack problem
- Should we log incorrect passwords?
  - “Almost” passwords in log file?
- Logging is not a trivial matter

Security Kernel

- **Kernel** is the lowest-level part of the OS
- Kernel is responsible for
  - Synchronization
  - Inter-process communication
  - Message passing
  - Interrupt handling
- The **security kernel** is the part of the kernel that deals with security
- Security kernel contained within the kernel

Security Kernel

- Why have a security kernel?
- All accesses go thru kernel
  - Ideal place for access control
- Security-critical functions in one location
  - Easier to analyze and test
  - Easier to modify
- More difficult for attacker to get in “below” security functions

Food for thought…

Two fundamental computer security facts:

All complex software systems have eventually revealed flaws or bugs that need to be fixed

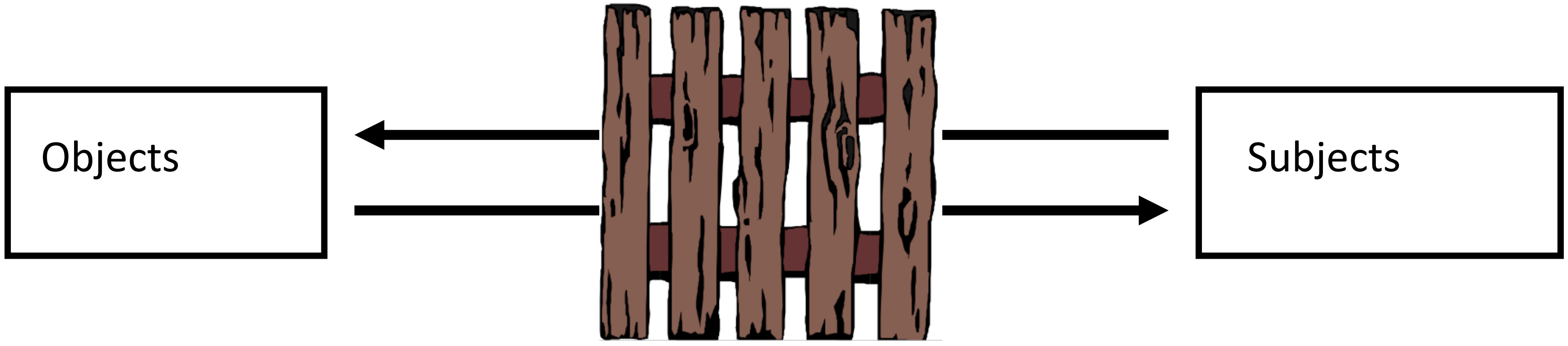
It is extraordinarily difficult to build computer hardware/software not vulnerable to security attacks

- Problems involved both design and implementation

The average number of changes accepted into the Linux kernel per hour over the last year (2017) was 8.5. That's not a typo. That's 8.5 code changes per hour or 204 changes every day.

Reference Monitor

- The part of the security kernel that deals with access control
  - Mediates access of subjects to objects
  - Tamper-resistant
  - Analyzable (small, simple, etc.)



Reference monitor

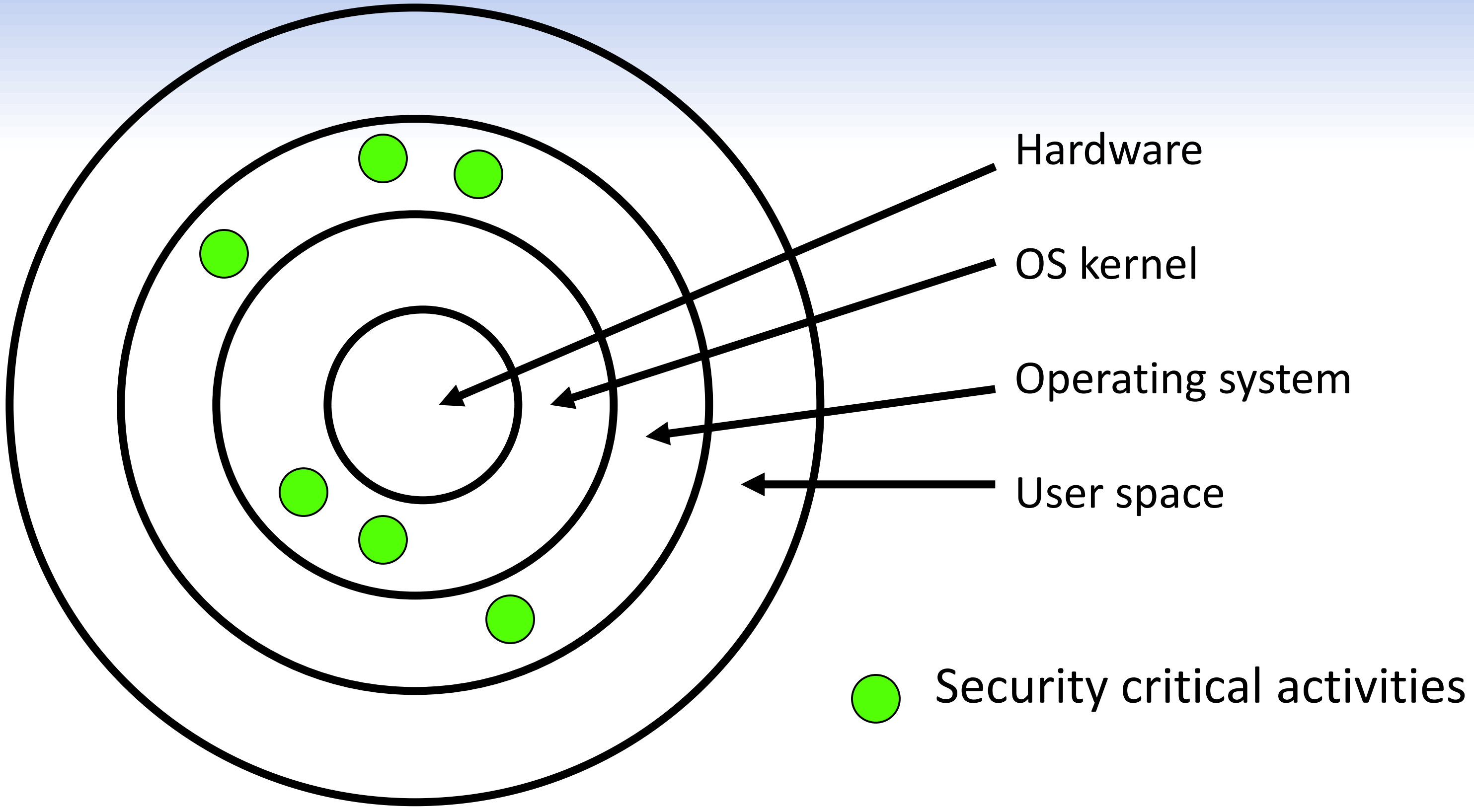
Trusted Computing Base

- TCB — everything in the OS that we rely on to enforce security
- If everything outside TCB is subverted, trusted OS would still be trusted
- TCB protects users from each other
  - Context switching between users
  - Shared processes
  - Memory protection for users
  - I/O operations, etc.

TCB Implementation

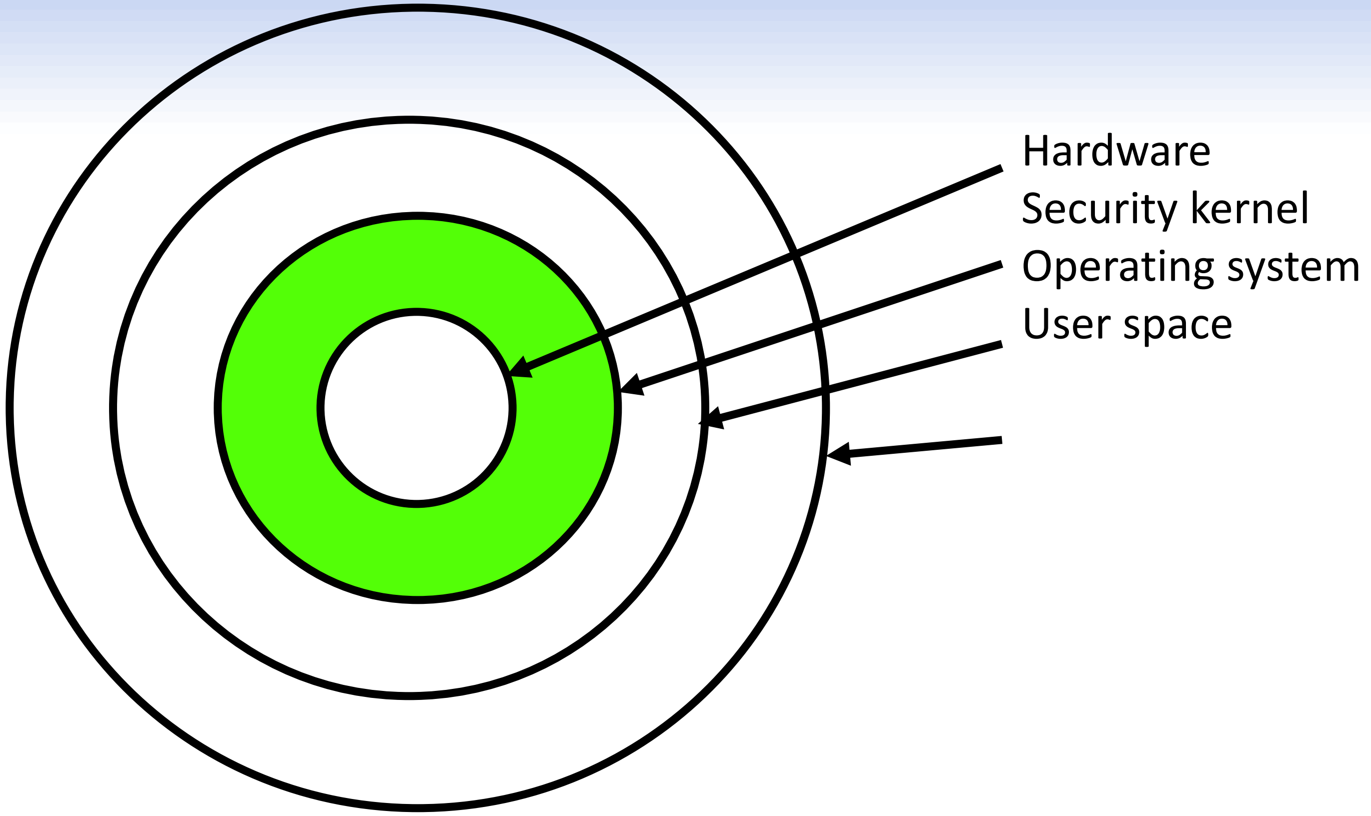
- Security may occur many places within OS
- Ideally, design security kernel first, and build the OS around it
  - Reality is usually the other way around
- Example of a trusted OS: **SCOMP**
  - Developed by Honeywell
  - Less than 10,000 LOC in SCOMP security kernel
  - Win XP has 40,000,000 lines of code!

Poor TCB Design



Problem: No clear security layer

Better TCB Design



Security kernel is **the** security layer

Trusted OS Summary

- Trust implies reliance
- TCB (trusted computing base) is everything in OS we rely on for security
- If everything outside TCB is subverted, we still have trusted system
- If TCB subverted, security is broken

$$\begin{tikzcd}
\text{OS} \arrow[d] \\
\textbf{OS Kernel} \arrow[d] \\
\textbf{Security Kernel}
\end{tikzcd}$$

Trusted Platform Module (TPM)

- Concept from Trusted Computing Group
- Hardware module at heart of hardware/software approach to trusted computing (TC)
- Uses a TPM chip
  - Motherboard, smart card, processor
  - Working with approved hardware/software
  - Generating and using crypto keys

Has 3 basic services:

- Authenticated boot
- Certification
- Encryption

Authenticated Boot Service

- Responsible for booting entire OS in stages and ensuring each is valid and approved for use
  - At each stage digital signature associated with code is verified
  - TPM keeps a tamper-evident log of the loading process
- Log records versions of all code running
  - Can then expand trust boundary to include additional hardware and application and utility software
  - Confirms component is on the approved list, is digitally signed, and that serial number hasn’t been revoked.
- Result is a configuration that is well-defined with approved components

Certification Service

- Once a configuration is achieved and logged the TPM can certify configuration to others
  - Can produce a digital certificate
- Confidence that configuration is unaltered because:
  - TPM is considered trustworthy
  - Only the TPM possesses this TPM's private key
- Include challenge value in certificate to also ensure it is timely
- Provides a hierarchical certification approach
  - Hardware/OS configuration
  - OS certifies application programs
  - User has confidence in application configuration

Encryption Service

- Encrypts data so that it can only be decrypted by a machine with a certain configuration
- TPM maintains a master secret key unique to machine
  - Used to generate secret encryption key for every possible configuration of that machine
- Can extend scheme upward
  - Provide encryption key to application so that decryption can only be done by desired version of application running on desired version of the desired OS
  - Encrypted data can be stored locally or transmitted to a peer application on a remote machine

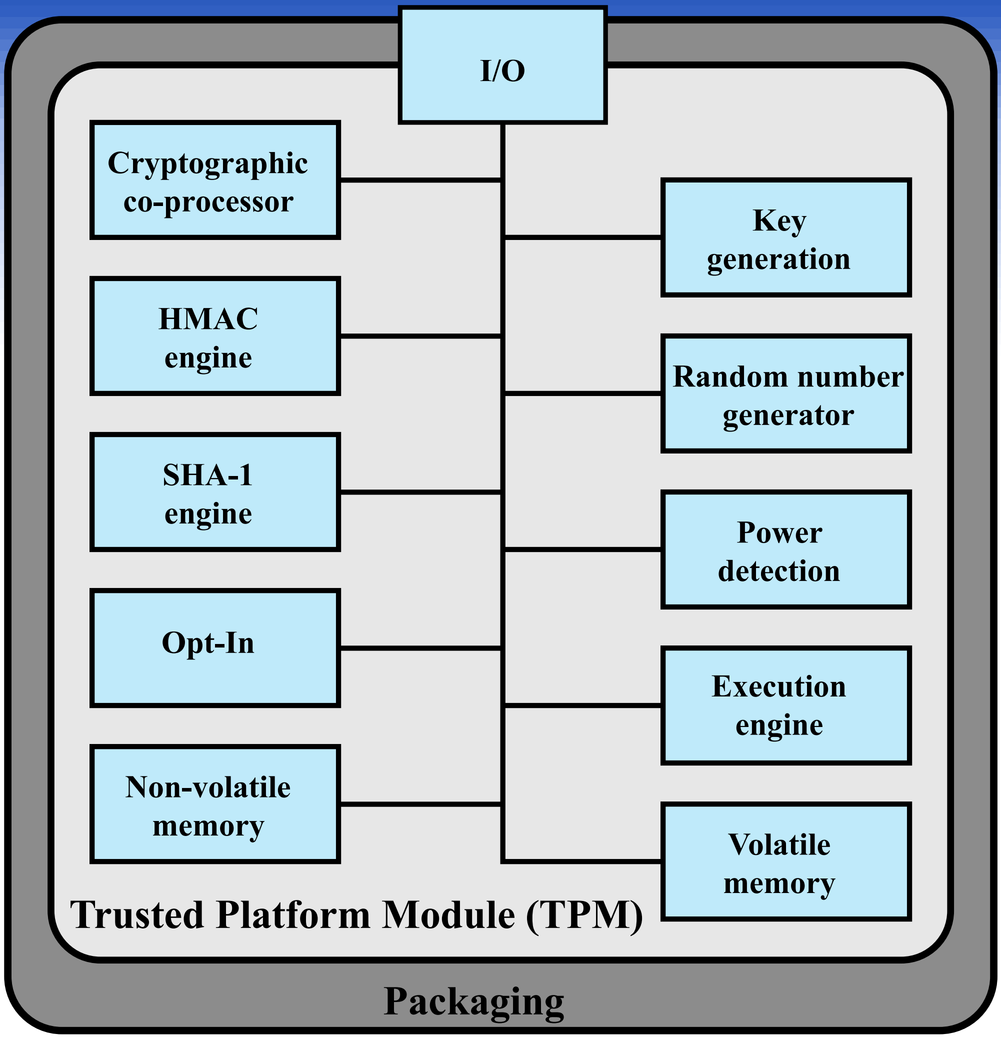


Figure 13.11 TPM Component Architecture

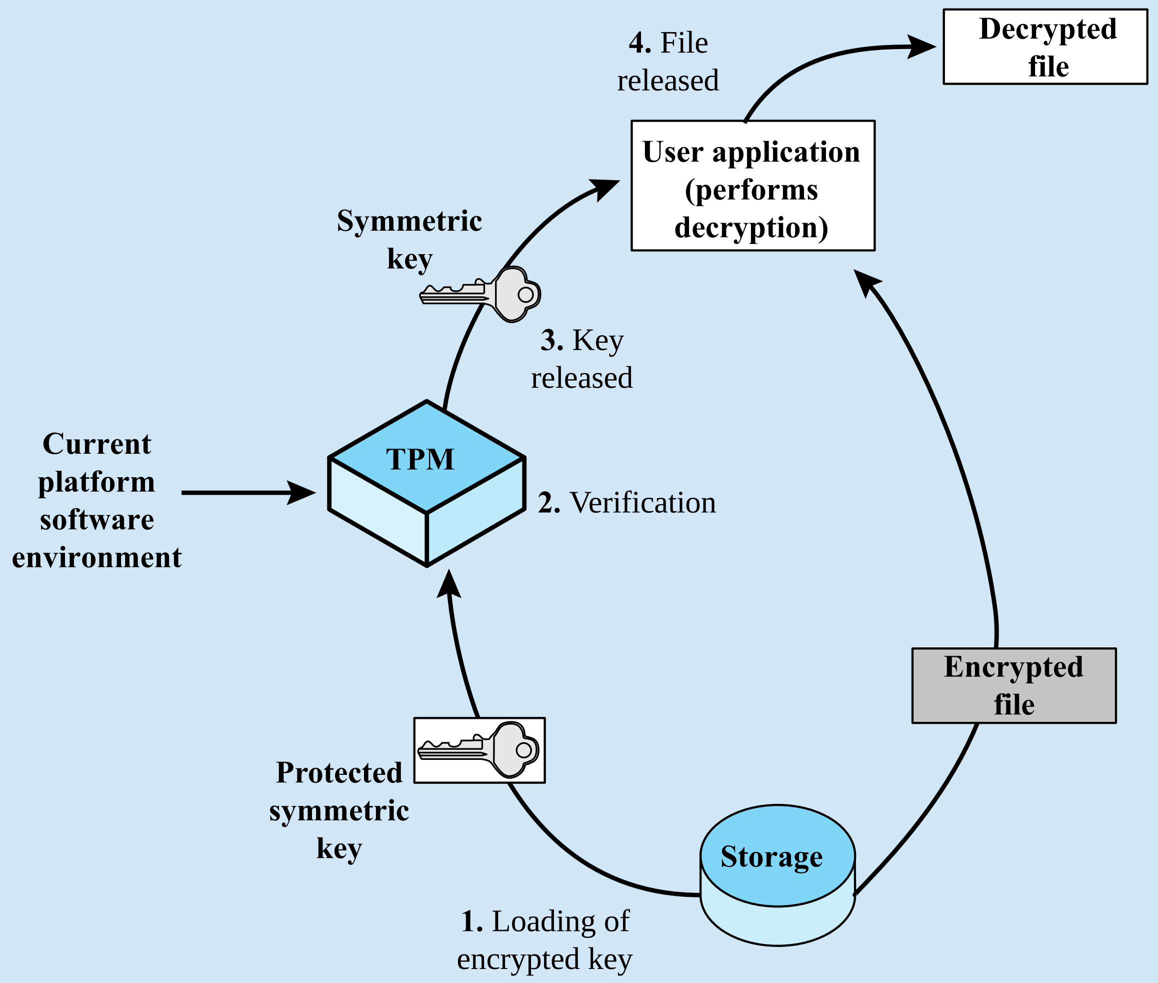


Figure 13.12 Decrypting a File Using a Protected Key