Systems 3

OS Summary

Marcel Waldvogel

(Handout)

Department of Computer and Information Science
University of Konstanz

Winter 2019/2020

Review of some of the goals

- What are OS goals, structure? How does the OS boot process work?
- What are the abstraction and isolation mechanisms and how do they work?
- What are the goals of the I/O subsystem and how is it structure?
- Why does caching work? (Locality of reference and the forms of caching, speed trade-offs.)
- Why is virtual memory important? How does it work? How do page tables and TLBs work?
- What are the goals of file systems? How are file types determined? What are the security implications?
- How do file systems work? What are (some of) their options?
- How to keep data secure? How do RAID and backup work? What are their goals and differences? What is the speed impact (live, backup, restore)?

Review of some of the goals (cont'd)

- Processes: The goals and benefits of multiprogramming (as processes and/or threads)?
- How do interrupts work? Why are they needed?
- Why and how to schedule?
- When is mutual exclusion needed? What are the mechanisms? How do they work and what benefits do they have? How do typical mutual exclusion paradigms work?
- What are the necessary conditions for race conditions? How can race conditions be avoided?
- Security: What are the three pillars? How do the three factors work? How to design for security? What can your opponent do?
- How do encryption mechanisms work? How is the identity of your peer proven?
- How (not) to store passwords?

What does an Operating System do?

Traditional goals

Hardware abstraction

1. Simplify application development
2. Portability
3. ...

Resource management

1. Share CPU among multiple processes
2. Share network cards among multiple services
3. ...

What does an Operating System do?

Today: Communications gateway

Enable communications

1. Between processes
2. Between systems

Restrict/select communications

1. Not everyone should be able to interfere with everything
2. Only when a process is ready
3. Only what a process expects

Hardware abstraction, resource management, and communications are independent dimensions (more or less).

What should be part of the Kernel?

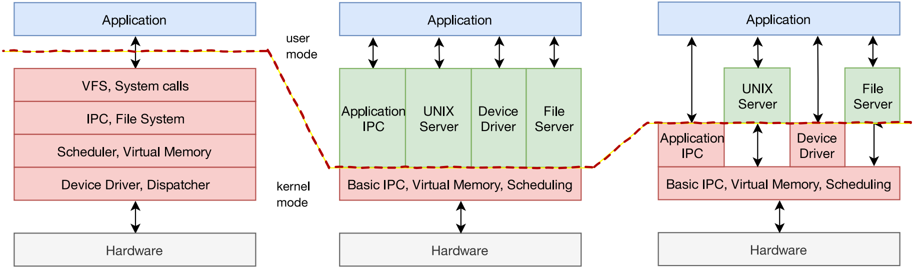
Criteria for inclusion

1. Direct hardware manipulation (I/O, CPU configuration)
2. Direct triggering by interrupt
3. Communication between processes

Criteria for exclusion

1. Complex operation (error-prone)
2. Debuggable
3. Updateable
4. Restartable

Kernel Overview



The main steps of the boot process

Basic overview of the Linux boot process.

Actor	Actions
BIOS/UEFI	test hardware, execute Master Boot Record
MBR	load and execute Grand Unified Bootloader
GRUB	load kernel and initrd
Kernel	mount root file system and start init
Init	determine run level and start
Runlevel	start various services

Partition tables

Master Boot Record (MBR)

- pretty old (1983)
- disk size limitation; originally physical (CHS) addressing
- only support for 4 primary partitions

GUID Partition Tables (GPT)

- globally unique identifier
- no size limitation
- unlimited number of partitions
- multiple copies
- CRC

Init RAM Disk (initrd)

- Grub allows to specify an initrd file
- Compressed archive containing a few kernel modules and scripts
- Work only with a specific kernel version

```
$ ls -l /boot/
total 230483
...
lrwxrwxrwx 1 root root       27 Jan  7 12:36 initrd.img -> initrd.img-5.3.0-26-gene
-rw-r--r-- 1 root root 80844710 Jan  7 12:36 initrd.img-5.3.0-26-generic
...
lrwxrwxrwx 1 root root       24 Jan  7 12:36 vmlinuz -> vmlinuz-5.3.0-26-generic
-rw------- 1 root root 11399928 Dez 18 06:27 vmlinuz-5.3.0-26-generic
...
```

Practical hint

Use mkinitramfs(8) to generate a new initrd file.

Systemd targets

Target	Description
poweroff.target	Halts the system and turns the power off
emergency.target	Single user mode. No services are running; …
rescue.target	Base system
multi-user.target	All services with CLI only
graphical.target	Multi-user with GUI

Abstraction and Isolation

Mechanism	Goal
Functions	Group variables and code
Object files	Larger groups, separate compilation
Process	+ isolate memory and execution
Accounts	+ isolate files, other resources
`chroot`(2), `jail`	+ separate file system
Containers	+ isolate system (processes, libraries, ...)
Virtualization	+ separate OS, believes to run on hardware[1]
Emulation	+ different (instruction set) architecture
Machine	+ real hardware

[1] Including optimizations to reduce expensive traps (paravirtualization, direct hardware access) and necessary hacks if the CPU does not distinguish clearly between privileged and unprivileged instructions (binary translation).

I/O software in the OS: Goals

1. device independence
2. uniform naming
3. error handling
4. synchronous vs. asynchronous
5. buffering

I/O Software layers

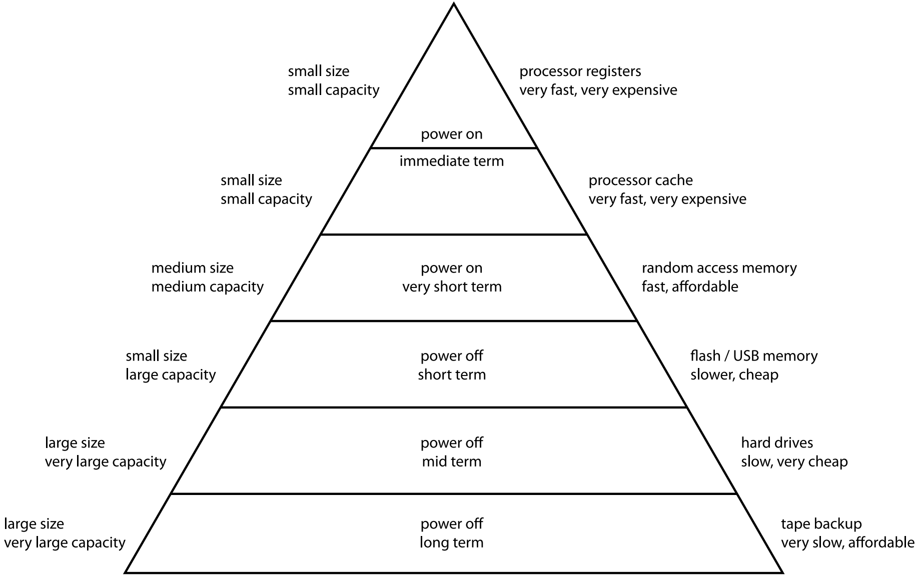
I/O Software is often organized in four layers:

Hardware

1. Interrupt handlers
2. Device drivers
3. Device-independent OS software
4. User-level I/O software

User

Memory Hierarchy



Working set[2]

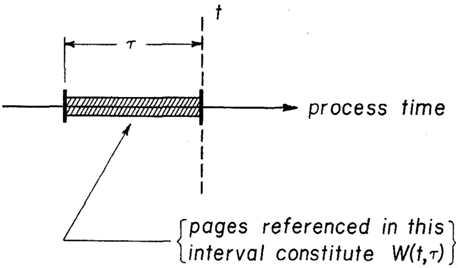


FIG. 2. Definition of $W(t, \tau)$

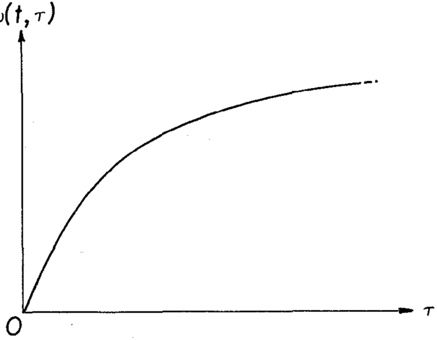


FIG. 3. Behavior of $\omega(t, \tau)$

This locality results in a (slowly) growing working set. Memory areas, which have not been recently used are less likely to be accessed again soon and could be moved to slower memory.

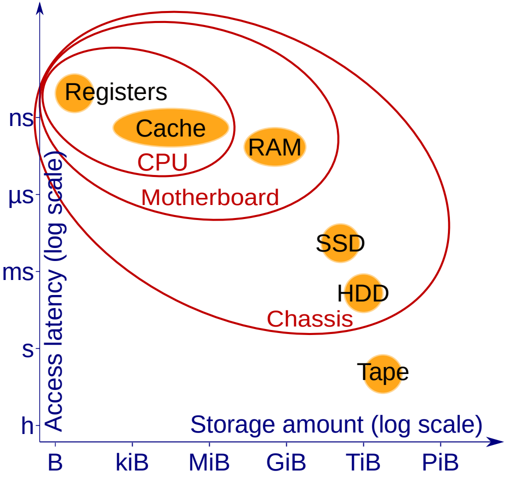
[2]Denning, Peter J. (1968). "The working set model for program behavior". Communications of the ACM. 11(5):323–333

Locality of Reference, speed/size/cost tradeoff

Data structures and program code (loops, shared functions, ...) frequently exhibit Locality of Reference:

- Temporal locality (accessing the same address again soon) → cache
- Spatial locality (accesing nearby addresses) → cache lines, pages

Advantage of combining big/slow and small/fast memories in a storage hierarchy.



Address translation

Page #			Offset											
0	1	0	0	0	1	0	1	1	0	1	0	0	0	0

(Index)	Frame #	Present?
7	— (0)	0
6	— (0)	0
5	3	1
4	4	1
3	0	1
2	6	1
1	1	1
0	2	1

1:1

1	1	0	0	0	1	0	1	1	0	1	0	0	0	0
Frame #			Offset											

Multilevel Page Tables

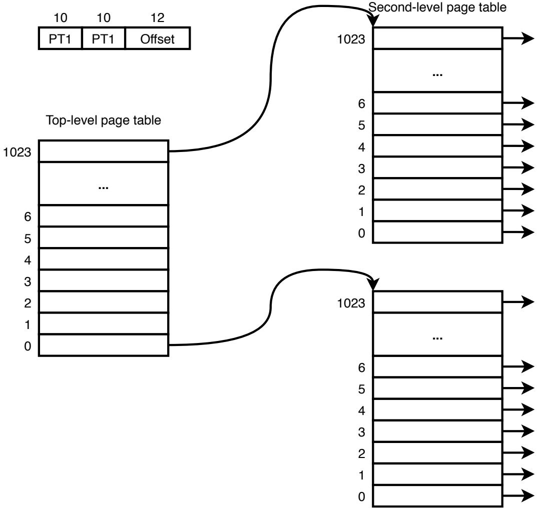


Figure: Two-level page tables with a 32-bit address.

Page Table Entry

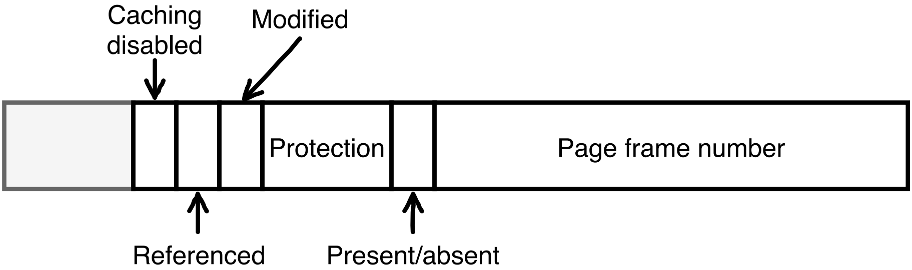


Figure: A typical page table entry[3].

RW(X) for whom?

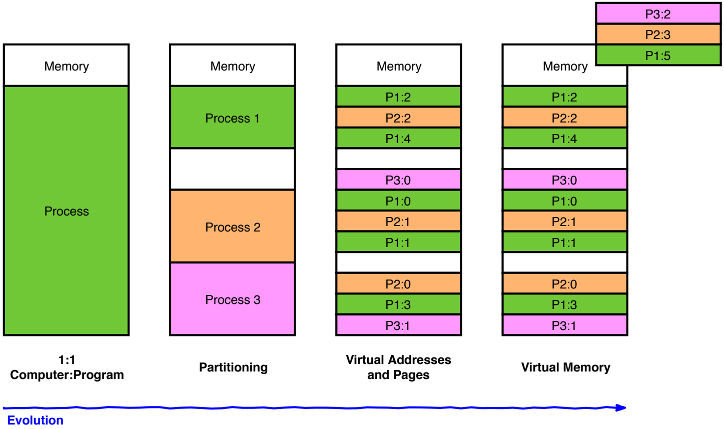
[3] modified bit = dirty bit

Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	$RW-$	31
1	20	0	$R-X$	38
1	130	1	$RW-$	29
1	129	1	$RW-$	62
1	19	0	$R-X$	50
1	21	0	$R-X$	45
1	860	1	$RW-$	14
1	861	1	$RW-$	75

Table: A TLB to speed up paging

Memory development



Swapping Entire process image to disk (*Partitioning*, ...).

(Demand) Paging Pages not yet/recently needed on disk.

Present Bit

Index	Present	Modified	Frame / Info
0	1	1	1234
1	1	0	2600
2	0	-	File #123, block 883
3	0	-	File #123, block 884
4	1	0	1536
5	0	-	Really invalid
…	…	…	…

Three views of a file system

- **Hardware view**
  Sectors, tracks, blocks, access time, disk scheduling ✓
- **Data view**
  Data structures, layout, properties
- **Application view**
  System calls, protection, programs

Storing/Retrieving information

Essential requirements for persistent ("long-term") information storage:

1. It must be possible to store a very large amount of information.
2. The information must survive the termination of the process using it.
3. Multiple processes must be able to access the information concurrently.

How to determine file type?

- **File extension** (.exe, .jpeg, .txt)
  untrustworthy (extension can be changed by user)
- **File format** (e.g. magic numbers)
  untrustworthy (PHP image attack[4])

Question: Is the filename stored inside the file? Why not?

[4] "Programs do not care about file names. They do not even care whether the file has a name, or finally is it a file or something else!"
— "PHP: Running *.jpg as *.php or How to Prevent Execution of User Uploaded Files", Igor Data, Medium.com, 2017-01-24 (accessed 2019-12-05).

General file system layout

1. Boot control block (per volume)
2. Volume control block / superblock (per volume)
3. Directory structure
4. File control block (per file)

Allocation Methods

1. Contiguous Allocation
2. Linked allocation
3. Indexed allocation (linked, array-based, tree-based)

RAID: The Idea

- Reliable, fast disks are expensive
- (Cheap) disks are unreliable, slow
- "Redundant Array of [Inexpensive;Independent] Disks"
- Use multiple (cheap) disks to get/exceed the reliability/speed of expensive disks

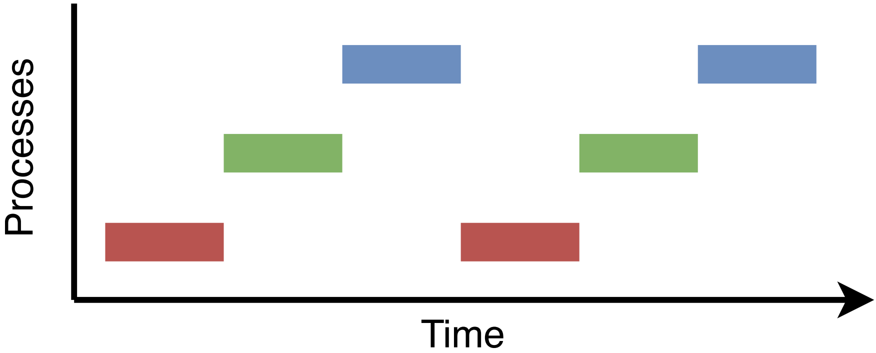
See e.g. https://en.wikipedia.org/wiki/RAID

Modern File Systems

- Large directory support (B-Trees etc.)
- Consistency (write ordering)
- Journaling
- Snapshots (Copy-on-Write)[5]
- Deduplication (Retroactive Copy-on-Write)

[5] Addresses **some** backup issues, not all!

Multiprogramming



Interrupts

Interrupt handling on the lowest level:

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector
3. Assembly language procedure saves registers
4. Assembly language procedure sets up new stack
5. C interrupt service runs
6. Scheduler marks waiting task as ready
7. Scheduler decides which process is to run next
8. C procedure returns to the assembly code
9. Assembly language procedure starts up new current process

Processes vs. Threads

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

In PCB or CPU

In process memory

Process Hierarchies

```
klaus    16437  2306  0 Nov07 ?        01:11:58  \_ /usr/bin/tilix
klaus    28498 16437  0 Nov11 pts/2    00:00:00  |   \_ /bin/bash
klaus     2113 16437  0 Nov11 pts/4    00:00:00  |   \_ /bin/bash
klaus     2323 16437  0 Nov11 pts/5    00:00:00  |   \_ /bin/bash
klaus     5978 16437  0 Nov11 pts/6    00:00:00  |   \_ /bin/bash
klaus    25024 16437  0 Nov12 pts/8    00:00:00  |   \_ /bin/bash
klaus    12380 16437  0 Nov14 pts/3    00:00:00  |   \_ /bin/bash
klaus     5420 12380  0 13:23 pts/3    00:00:00  |   |   \_ ps -ef --forest
```

Different process behavior

- **compute-bound**
  spend most of their time computing
- **I/O-bound**
  spend most of their time waiting for I/O

When to Schedule

When scheduling is absolutely required:

1. When a process exits.
2. When a process blocks on I/O or a mutual exclusion mechanism.

When scheduling usually done (though not absolutely required)

1. When a new process is created.
2. When an I/O interrupt occurs.
3. When a clock interrupt occurs.

Why? When?

Goals of scheduling algorithms

- All systems
  - Fairness
  - Policy enforcement
  - Balance
- Batch systems
  - Throughput
  - Turnaround time
  - CPU utilization
- Interactive systems
  - Response time
  - Proportionality
  - User happiness
- Real-time systems
  - Avoiding event loss
  - Avoiding data loss
  - Predictability

When to Schedule

- First-Come First-Serve
- Shortest Job First
- Shortest Remaining Time Next
- Three Level
- Round-Robin
- Priority
- Dynamic Priorities

What is used?

System	Goals	Scheduler
Real-time	React to events in time	Strict Priority
Server	Fast reaction to many requests	Dynamic priority
HPC	Finish simulations fast	Don't care/Admission
Desktop	Fast reaction to user inputs	Dynamic priority

Critical Sections: Avoiding Race Conditions

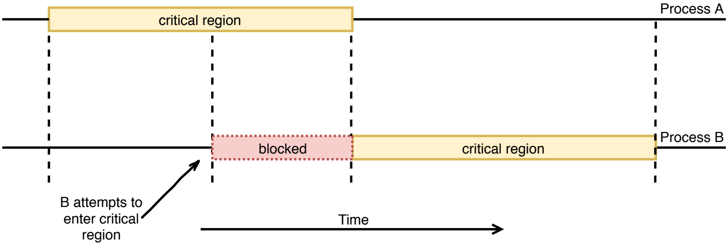
Basic assumptions necessary:

1. No two processes may be simultaneously inside their critical regions.
2. No assumptions may be made about speeds or the number of CPUs.[6]
3. No process running outside its critical region may block other processes.
4. No process should have to wait forever to enter its critical region.

$\Rightarrow$ Mutual exclusion

[6] One(!) or more. Are CPUs always the same speed?

Mutual exclusion



Deadlock

"A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause."

Conditions for Deadlock

- Mutual exclusion
- Hold and wait
- No (lock) preemption
- Circular wait

Process coordination

- Mutex
- Semaphore
- Monitor
- Message passing

Producer-Consumer

Bank Teller

- Any number of customers
- Customers can come whenever they want
- Any number of tellers
- Tellers work in parallel

Holiday card writers

- Any number of card writers
- Put finished cards on the shared desk
- Any number of envelope packagers
- Pick up cards for packaging

Readers and Writers

Example

- Given a shared data structure (hash, linked list, tree, database, ...)
- Some threads want to modify the data structure (read-write access[7])
- Some threads only want to look up information (read-only access[8])

1. Mutual exclusion among readers is wasteful.
2. Only required among writers or between readers and writers.

Problem

How to allow concurrent readers? How to still lock out writers?

1. Only first reader in locks the database; last reader out unlocks.
2. Writers always lock/unlock.

[7] aka "writer"

[8] aka "reader"

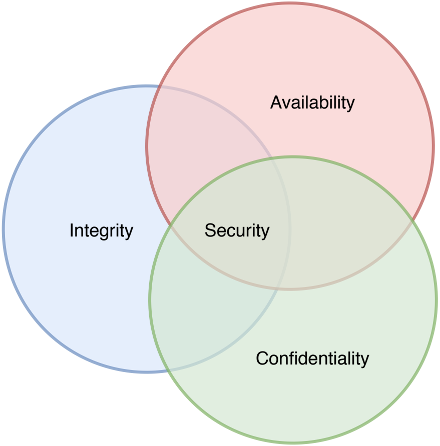
Filesystem Hierarchy Standard (FHS)[9]

The Filesystem Hierarchy Standard (FHS) defines the directory structure and directory contents in Linux distributions.

/bin	binaries for all users
/boot	boot loader files
/dev	device files
/etc	System-wide configuration files
/lib	Libraries for binaries in /bin and /sbin
/media	Mount point for removable media
/srv	Data served by the system
/tmp	Temporary files
/usr	User System Resources
/var	Variable data

[9] https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html (yes, all necessary!)

Security



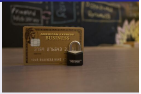
Identification: Three factors

Something you know

- Password
- PIN

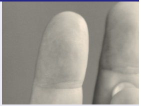
Something you have

- Bank card
- Hardware token
- Phone



Something you are

- Biometrics:
  Fingerprint, face, retina, speech, typing pattern, gait, …



Security Design

Criteria	Goal
Kerckhoffs's principle	Open, inspectable system
Principle of least privilege	Contain results of misbehavior
Secure by default	Laziness should not cause problems
Secure by design	Not as an afterthought
Economy of mechanism	KISS means less can go wrong
Privacy by design	Data can be toxic
Psychological acceptability	Keep users on our side
Fail securely	If something fails, avoid the epic

Attacks

- Passive
  - Wiretapping
  - Keystroke logging
  - Data harvesting
- Active
  - Denial-of-service
  - Spoofing
  - Man-in-the-middle
  - Ping flood
  - Code injection
- Malware
  - Virus
  - Ransomware
  - Trojan horse
  - Worm
- User stupidity

How to protect yourself

- Software updates
- One service, one password
- Lock your screen!
- Do not access USB "drives"
- Use full disk encryption
- Create a backup
- Prepare for security incident

- Never trust user input!

Meltdown & Spectre

By Raimond Spekking (CC BY-SA 4.0)

CPU bugs → OS/app mitigation

By Alan Light (CC BY-SA 3.0)

Speculative Execution + Resource Sharing

Buffer overflows

- Stack Canaries
- Data Execution Prevention
- Return-oriented programming
- Address-Space Layout Randomization

Also remember `printf`(3) is dangerous and do not construct code strings.

Exploiting race conditions

Exploitable code:

```
int fd;

/* Using real UID */
if (access("./.well-known/CylmEesyudneyd1", W_OK) != 0) {
exit(1);
}

/* Malicious program could remove file and create a symbolic link */

/* Using effective UID */
fd = open("./.well-known/CylmEesyudneyd1", O_WRONLY);
write(fd, someInput, sizeof(someInput));
```

From access(2): the use of this system call should be avoided.

Check always your return value!

Responsible Disclosure

1. Create a report
2. Contact company
   - security@, abuse@, noc@ (RFC2142[10])
   - bug bounty program
   - security.txt[11]
3. Wait for response and fix
4. Publish details
   - Google's disclosure policy[12]
   - Responsible Vulnerability Disclosure Process[13]

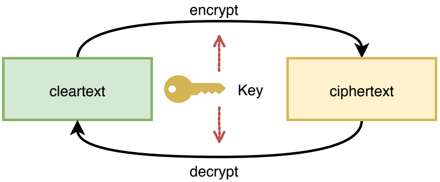
[10] https://www.ietf.org/rfc/rfc2142.html

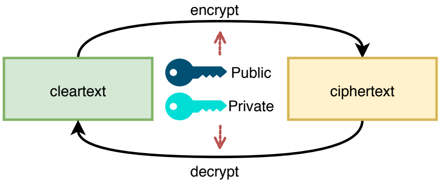
[11] https://securitytxt.org/

[12] https://googleprojectzero.blogspot.com/2015/02/feedback-and-data-driven-updates-to.html

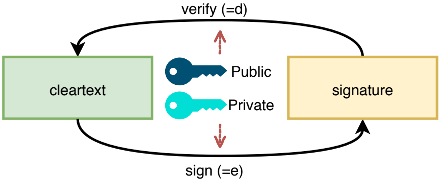
[13] https://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00

Symmetric vs. Asymmetric Encryption





- Symmetric is (much) faster
- Asymmetric is more versatile
- For efficiency, hashing is often used (especially with signatures)
- Asymmetric can be used “the wrong way around”:



Hybrid operation

Hybrid encryption

- Generate random secret key k
- Encrypt message M symmetrically ($E_k(M)$)
- Encrypt k asymmetrically with recipient key(s)

Hybrid signature

- Hash message (symmetrical, $H(M)$)
- Sign hash (asymmetric)

Benefits

- No secret key exchange
- Only a few bytes for the (slow) asymmetric operation

Certificates

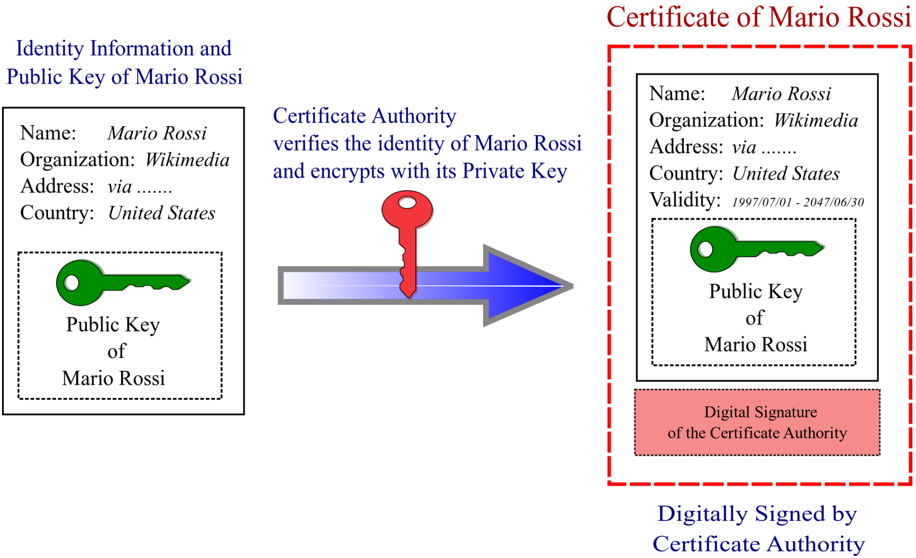


Image credit: Wikpedia user I, Giaros; CC BY-SA 3.0.

Password storage

Never in plain text!

Hashing

Makes reversing hard.
Attack: Lookup tables.

Salted Hashing

Defeats lookup tables.
Attack: Cloud infrastructure

Iterated Hashing

Defeats cheap computing power through higher computation costs.
Common algorithms are PBKDF2, scrypt and bcrypt.