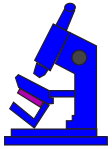
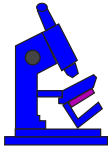
Introduction to Computer Security
CS3235

Hugh Anderson

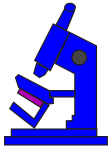


Contact information

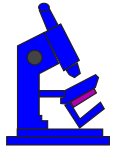


Room	S15 #06-12
Telephone	6874-6903
E-mail	hugh@comp.nus.edu.sg

...and Spinellis...

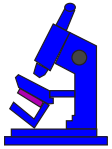


People

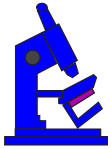


Dr Robert Deng, Institute for Infocomm Research

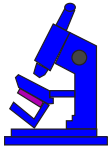
http://www.i2r.a-star.edu.sg/icsd/staff/Robert/



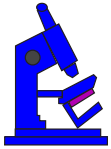
Official SOC description



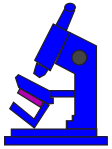
With the widespread use of computers and Internet as well as electronic commerce, computer security becomes more and more important. The objective of this module is to give students basic knowledge of computer security. This module covers the following topics: threats to computer systems, network security fundamentals, security in a layered protocol architecture, authentication in computer systems, access control, intrusion detection, security architecture and frameworks, lower layers security protocols, upper layer security protocols, electronic mail and EDI security, directory systems security, Unix systems security, security evaluation criteria.



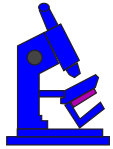
Assessment



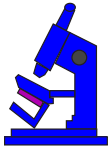
Assessment	Weighting	Grade
Assignments		35%
Tutorials		5%
Mid-term	Closed book	10%
Final Exam	Open Book	50%
Total marks		**100%**



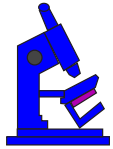
Resources



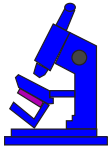
- ✔ Computer Security: Art and Science, Matt Bishop
- ✔ The notes are expanded versions of the overheads
- ✔ Directed readings - all available on the Internet.
- ✔ IVLE at http://ivle.nus.edu.sg/
- ✔ Web site at http://www.comp.nus.edu.sg/~cs3235



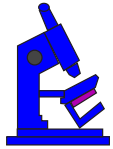
Comparison



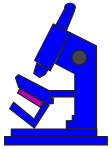
- ✔ Different focus
- ✔ More introductory and practical material
- ✔ Less material duplicated



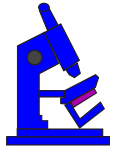
Topics - general



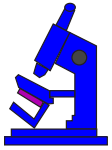
- ✔ History and background,
- ✔ Preliminaries
- ✔ Encoding and decoding
- ✔ Protocols used for security.



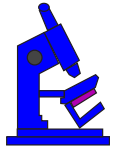
Topics - detail



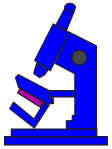
- Mathematical, physical, legal (2 lectures)
- Security models (1 lecture)
- Secrecy (1 lecture)
- Insecurity (2 lectures)
- Safety/control hardware/software (2 lectures)
- Assurance (1 lecture)
- Protocols (1 lecture)
- + Case studies



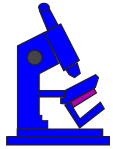
Tutorials



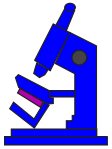
- ✔ Start in 3rd week
- ✔ More details next week



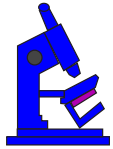
My expectation...



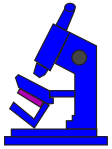
- ✔ Attend classes and tutorials
- ✔ Ask if you don’t know
- ✔ Read notes, book, and the readings...
- ✔ Get interested in the subject



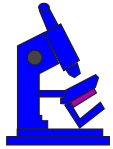
Chapter 1



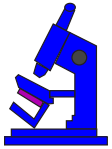
Lecture 1 - Introduction



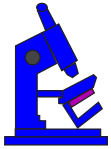
Jump-about-introduction



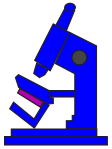
...sorry sorry...



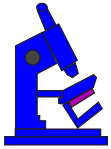
The History of Herodotus



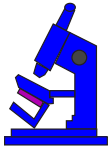
For Histiæus, when he was anxious to give Aristagoras orders to revolt, could find but one safe way, as the roads were guarded, of making his wishes known; which was by taking the trustiest of his slaves, shaving all the hair from off his head, and then pricking letters upon the skin, and waiting till the hair grew again. Thus accordingly he did; and as soon as ever the hair was grown, he despatched the man to Miletus, giving him no other message than this- "When thou art come to Miletus, bid Aristagoras shave thy head, and look thereon." Now the marks on the head, as I have already mentioned, were a command to revolt...



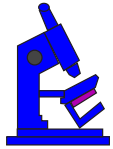
The History of Herodotus



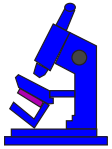
- ✔ Histiæus ensured *confidentiality*
- ✔ Used again by Germany in the 1914-1918 war
- ✔ This is now called steganography



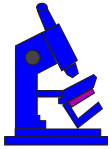
More history



- ✔ Cæsar encoded messages - cryptography
- ✔ Agreed protocols to ensure correct conduct of a war
- ✔ Examples taken from the world of warfare



Aspects to “computer security”



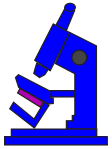
Security problems in society reoccur in computers

✔ Confidentiality = locks/encoding.

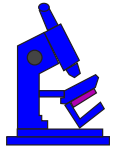
✔ Integrity = handshakes/signatures

Computer versions much faster.

In this course, security includes wider aspects.

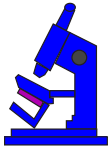


Terms: Services

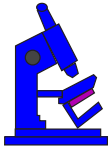


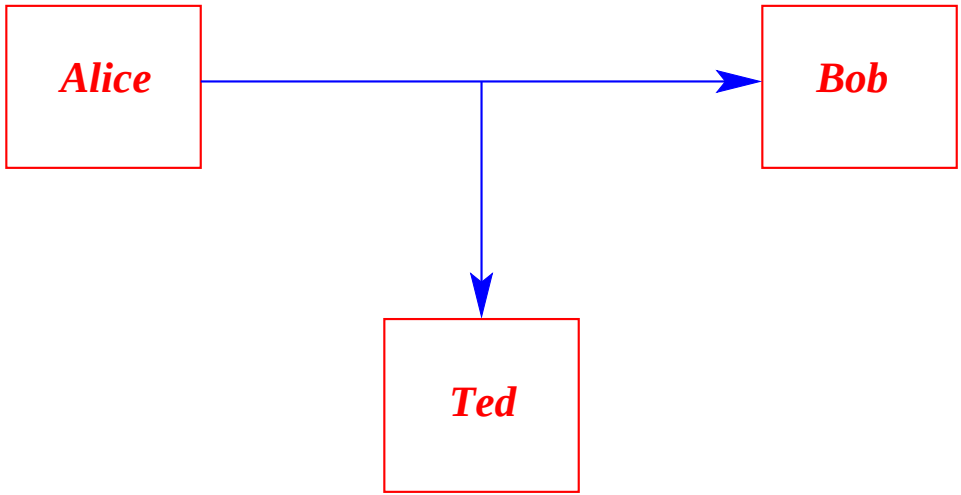
Three aspects of security *services*:

- **confidentiality**: concealing information - resources;
- **integrity**: trustworthiness of data - resources;
- **availability**: preventing denial-of-service.

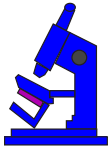


Terms: Threats

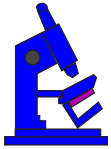


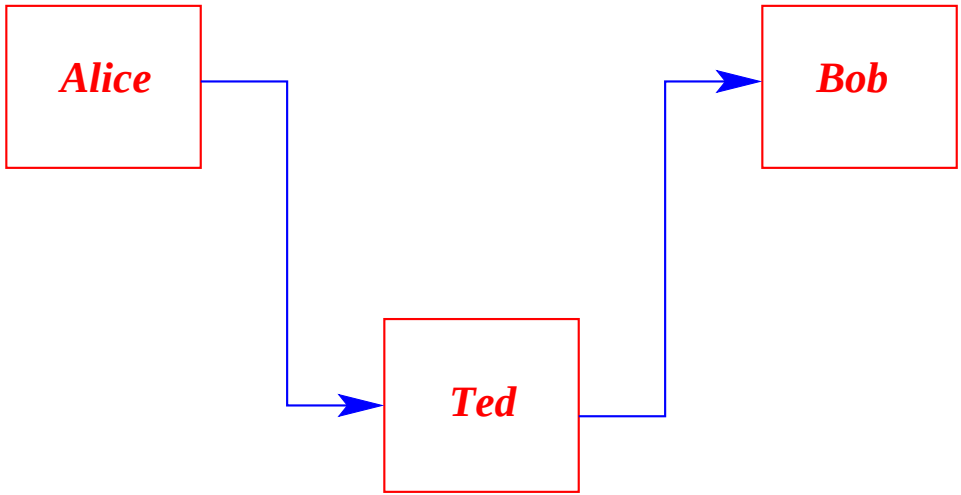


Snooping

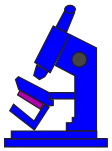


Terms: Threats

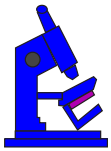


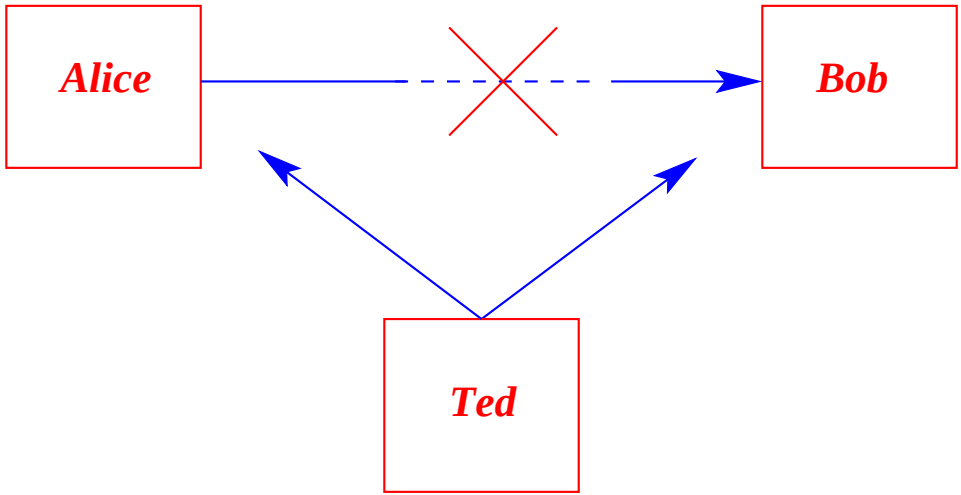


Man in the middle

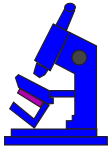


Terms: Threats

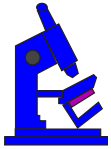


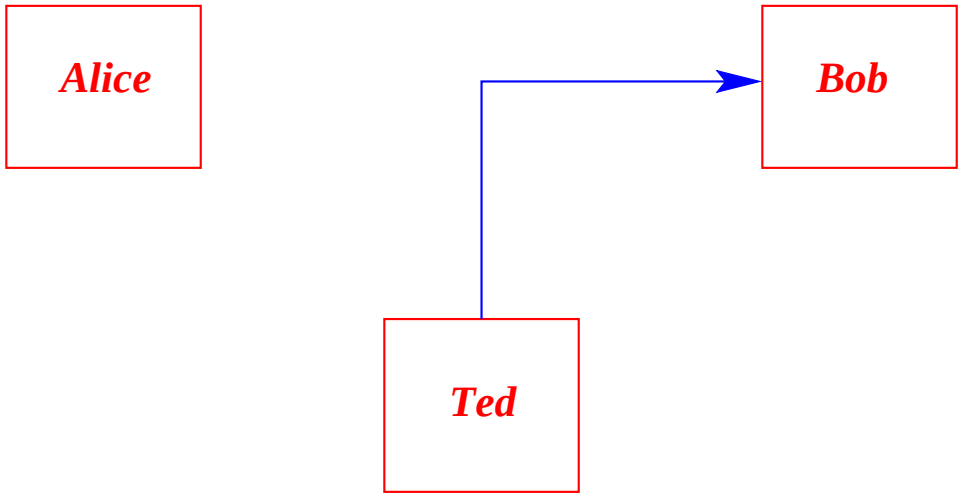


Denial of service

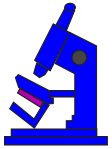


Terms: Threats

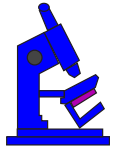




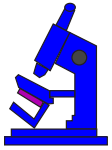
Spoofing



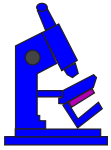
Terms: Threats



- **disclosure**: unauthorized access (snooping);
- **deception**: accept false data (man-in-the-middle);
- **disruption**: prevent correct operation (denial-of-service);
- **usurpation**: unauthorized control (spoofing).

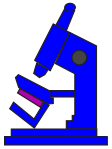


Terms: Policy and mechanism

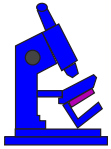


We differentiate between a security *policy* and a security *mechanism*:

- **policy**: what is allowed/disallowed;
- **mechanism**: ways of enforcing a policy

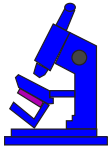


NUS IT policy

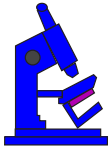


For example, at NUS, we have an IT policy which includes a range of clauses regarding security concerns, such as:

> ***4.2 Undermining System Integrity***
>
> *Users must not undermine the security of the IT Resources, for example, by cracking passwords or to modify or attempt to modify the files of other Users or software components of the IT Resources.*

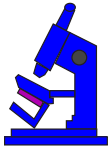


NUS mechanisms

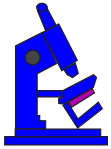


6.3 Use Of Security Scanning Systems

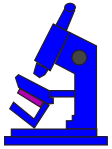
Users consent to the University's use of scanning programs for security purposes at system level for computers and systems that are connected to the University's network. This is to ensure that any computers or systems attached to the network will not become a launching pad for security attack and jeopardise the IT Resources. System level scanning includes scanning for security vulnerabilities and virus detection on email attachments. Users' files and data are excluded from the scanning.



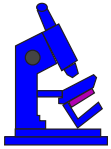
Topic: Preliminaries



- ✔ Review some mathematical concepts. XOR, modulo, primes
- ✔ The textbook, and my notes should be enough.
- ✔ *Physical* laws and procedures. Information and Entropy



Topic: Security models

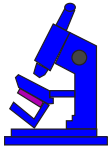


These models provide formal ways of looking at computer security in an abstract manner.

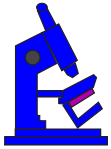
1. Define a model, and

2. prove it secure

3. Ensure system complies with model



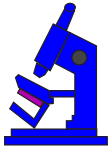
Topic: Security models



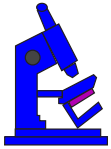
- The Bell-LaPadula model (no read-up, no write-down) provides a military viewpoint to assure *confidentiality* services.

- The Biba and Clark-Wilson models attempt to model the trustworthiness of data and programs, providing assurance for *integrity* services.

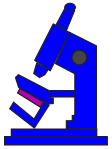
... Read ahead ...



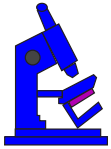
Topic: Security models



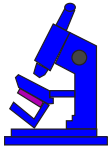
- ✔ Determine properties of the model, and
- ✔ Verify that implementations are valid.
- ✔ Basis of trusted operating systems
- ✔ Modelling for availability is tricky



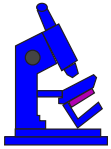
Topic: Secrecy



- ✔ Commerce relies on secure transfer of information, and
- ✔ Often just want things to be secret
- ✔ Distance between you and an attacker is shrinking
- ✔ Criminals have an access point into your living room



2000 years ago...



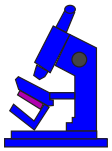
Replace each Roman letter in a message, with another Roman letter, obtained by rotating the alphabet some number of characters:

I C L A V D I V S

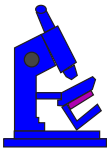
A B C D E F G H I K L M N O P Q R S T V X Y Z
V X Y Z A B C D E F G H I K L M N O P Q R S T

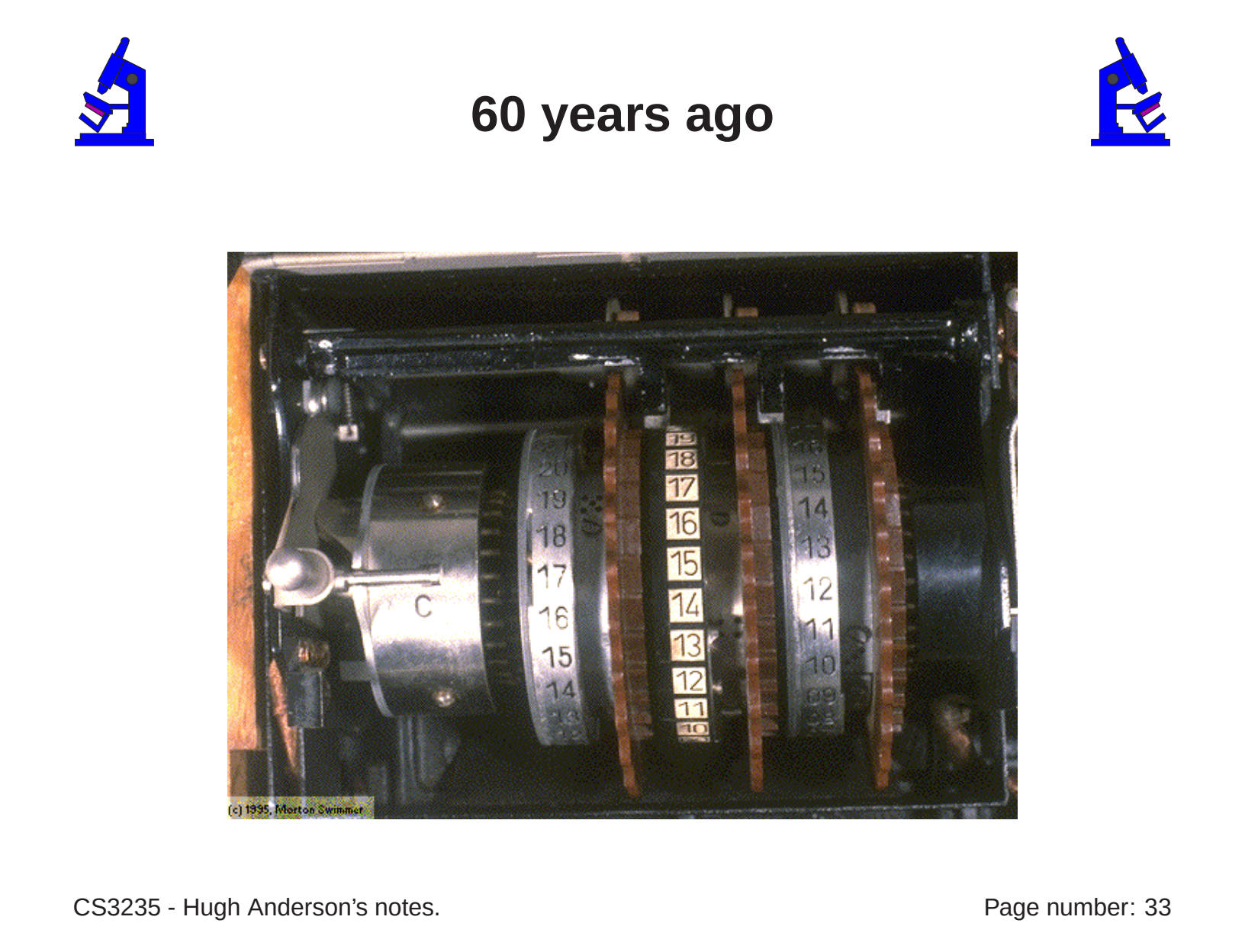
E Y G V Q Z E Q O

We can specify a Cæsar cipher by just noting the number of characters that the alphabet is rotated.



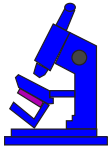
60 years ago...



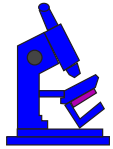


60 years ago

C
20
19
18
17
16
15
14
19
18
17
16
15
14
13
12
11
10
15
14
13
12
11
10
09
(c) 1995, Morton Swimmer



Enigma machines



- ✔ Commercial device
- ✔ Used by the German military
- ✔ Belief that could not be decoded.

Enigma machines

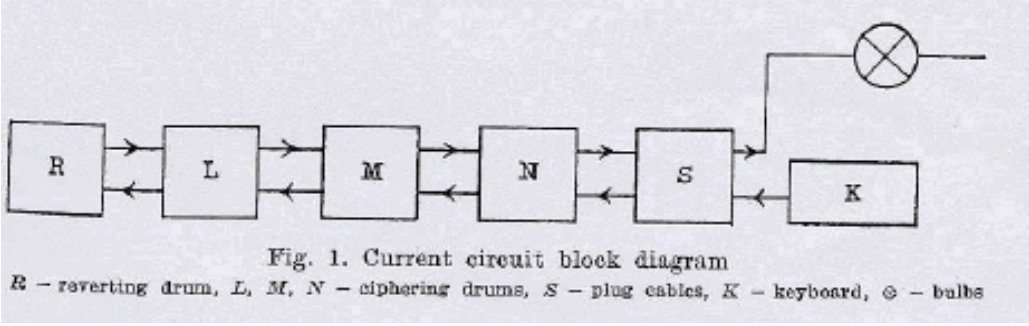
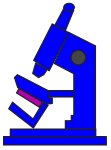
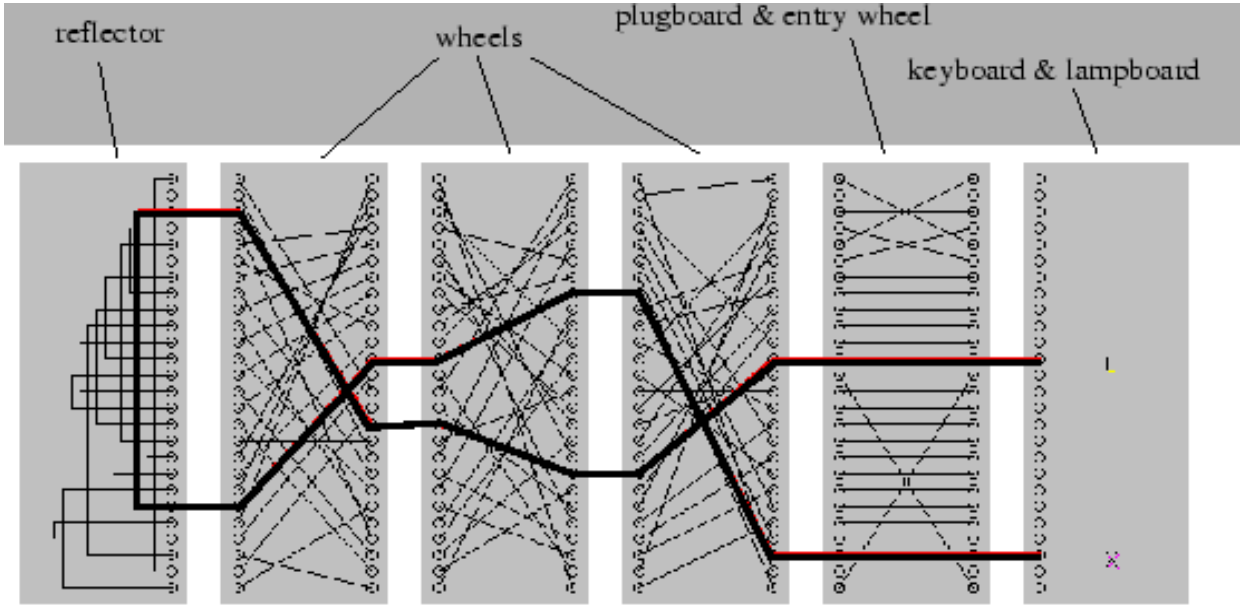


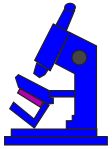
Fig. 1. Current circuit block diagram

R – reverting drum, *L*, *M*, *N* – ciphering drums, *S* – plug cables, *K* – keyboard, ⊗ – bulbs

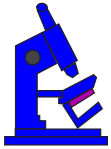


Enigma machines

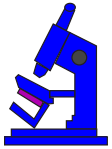




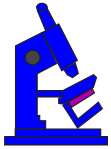
Hacking Enigma



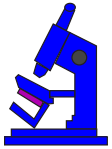
- ✘ Americans captured a German submarine?
- ✘ Alan Turing did it all?
- ✘ Hard workers at Bletchley Park?
- ✘ My dad?



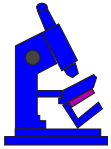
Hacking Enigma



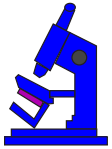
- ✔ 1928: Poles intercepted a machine
- ✔ 1928: Maths Dept at University of Poznan: Marian Rejewski, Jerzy Rozycki, Henryk Zygalski.
- ✔ Decoded some messages
- ✔ German army using an extra level of encoding
- ✔ French spies uncovered the extra encoding



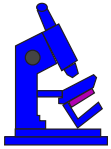
Hacking Enigma



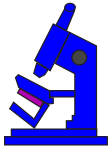
- ✔ 1933-1939: the Polish Ciphers Office was able to decode messages, although slowly.
- ✔ July 1939: Poland gave Enigma copies to English
- ✔ Bletchley Park
- ✔ May 1941: English captured the U-110 submarine, complete with a genuine Enigma machine, and code books.



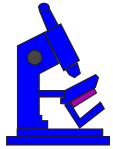
Hacking Enigma



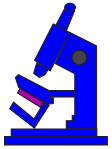
- ✔ 1941-45: English could decode most German military transmissions.
- ✔ 1941-45: developed a hardware system
- ✔ Precursor to modern-day computers



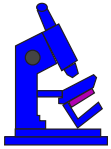
Today...sssshhhh



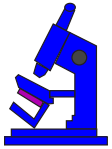
- ✔ Secure encrypted communications between
  - ✔ two untrusted hosts
  - ✔ over an insecure network.
- ✔ Other connections can also be forwarded
- ✔ Users must prove their identity to the remote machine



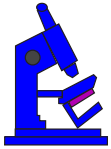
Secure-shell



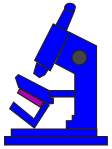
- ✔ Based on public-key cryptography:
  - ✔ Encryption and decryption use separate keys
  - ✔ not possible to derive one from other
  - ✔ RSA is one such system.
- ✔ Encodings believed to be difficult to decode, and
- ✔ protocols of message exchange that are believed to be secure.



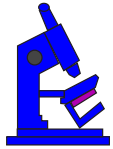
Topic: Insecurity



- ✔ Systems dangerously easy to subvert
- ✔ Adversary gains control over your system
- ✔ You sign a contract, and other party doesn’t.
- ✔ Investigate hacking and reducing risk

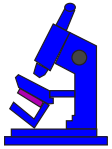


Topic: Insecurity

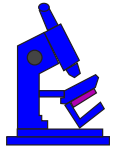


A locked air-conditioned room with file server:

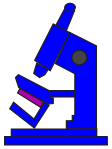
- The lock can be picked, or the door kicked in.
- The console of the server computer may be password protected, but
  - it may be rebooted with a different disk.



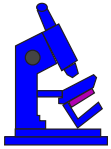
Topic: Insecurity



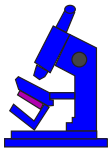
- The reboot process may be (BIOS) password protected, but
  - the case of the computer may be opened and the disk removed.
- And so on...



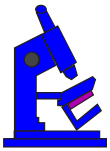
Topic: Insecurity

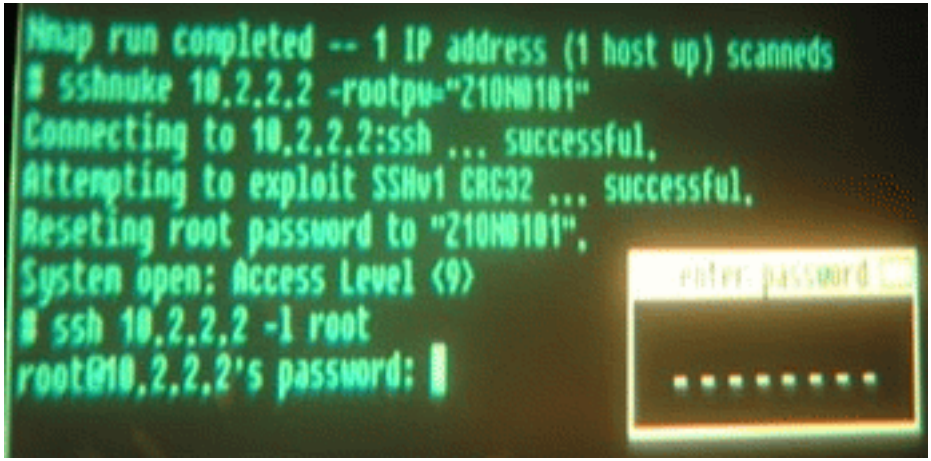


- ✔ Tempest - computer screen monitoring
- ✔ Paper
  - ✔ http://jya.com/emr.pdf
- ✔ Overcoming
  - ✔ http://www.cs.rice.edu/~dwallach/courses/comp527_s2000/ih98-tempest.pdf
- ✔ Monitor screens at a distance of 1km for $15.

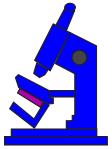


Topic: Insecurity

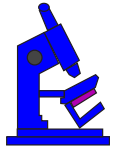




Kick in doors without even using your feet

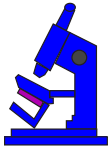


Topic: Insecurity

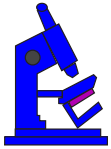


Non-repudiation for e-commerce:

- the buyer cannot order an item and then deny the order took place;
- the seller cannot accept money or an order and then later deny that this took place.

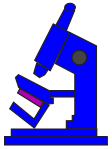


Topic: Insecurity

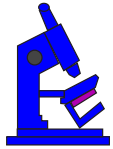


- ✔ Intrusive hacking is common on the Internet.
- ✔ Farms of subservient machines:

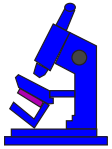
  At first, it looked as if some students at the Flint Hill School, a prep academy in Oakton, Va., had found a lucrative alternative to an after-school job...



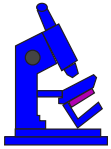
Topic: Insecurity



- ✔ Virusses: boot-sector hide their code in the boot sector of a disk.
  - ✔ the stoned virus for DOS, written by a student from New Zealand!
- ✔ A virus contains code that replicates, attaching itself to a program, boot sector or document. Some viruses do damage as well.

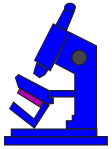


Topic: Insecurity

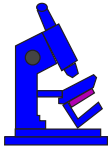


Worm is a program that makes copies of itself, transferring itself around. The Morris worm in 1988:

> *On the evening of 2 November 1988, someone infected the Internet with a worm program. That program exploited flaws in utility programs in systems based on BSD-derived versions of UNIX. The flaws allowed the program to break into those machines and copy itself, thus infecting those systems.*



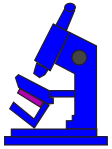
The Morris Worm



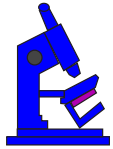
This program eventually spread to thousands of machines, and disrupted normal activities and Internet connectivity for many days.

ftp://ftp.cs.purdue.edu/pub/reports/TR823.PS.Z

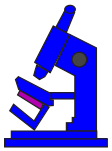
The author of the worm, Robert Morris, was convicted and fined $10,050 in 1990, and is currently a professor in the Parallel and Distributed Operating Systems group at MIT, lecturing in distributed systems areas.



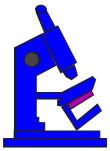
Topic: Protocols

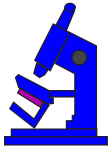


Some aspects of security are determined by the way in which we do things (the protocol), rather than what is actually done.

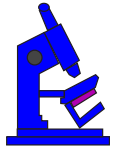


Topic: Protocols



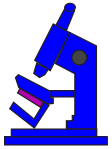


Summary of topics

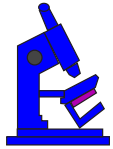


In this section, we introduced the following topics:

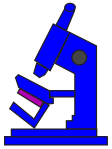
- An introduction to computer security
- Some definitions



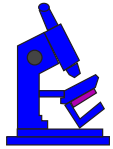
Further study

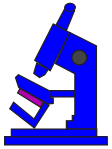


- Textbook Chapter 1
- Monitoring computer screens
  http://jya.com/emr.pdf
- Overcoming Tempest monitoring
  http://www.cs.rice.edu/~dwallach/courses/comp527_s2000/ih98-tempest.pdf
- The Morris worm
  ftp://ftp.cs.purdue.edu/pub/reports/TR823.PS.Z
- Military mathematical modelling of security
  http://80-ieeexplore.ieee.org.libproxy1.nus.edu.sg/xpl/tocresult.jsp?isNumber=13172

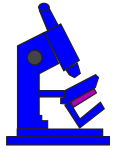


Done!

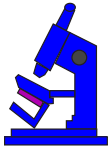




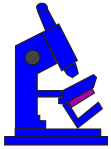
Chapter 2



Lecture 2 - Preliminaries

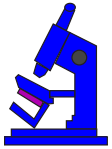


Note: CORS

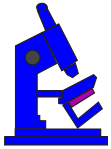


You should be getting your tutorial sessions sorted out using CORS!

http://www.cors.nus.edu.sg/

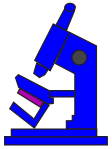


Overheads and notes

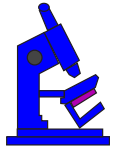


You can find all sorts of stuff looking in

http://www.comp.nus.edu.sg/~cs3235/2003-semesterI/



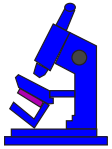
Question box



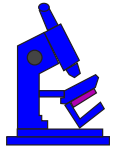
If you have any questions, feel free to place them in the question box...

Or stick your hand up...

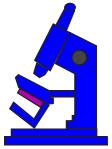
Or...



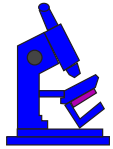
Last session



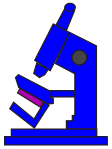
- ✔ Introduction, setting context
- ✔ Definitions
- ✔ Cæsar cipher, Enigma, Secure shell
- ✔ Insecurity



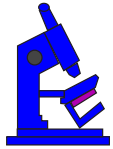
This session



- Finish context
- Math preliminaries
  - XOR
  - Logarithms
  - Fields and groups



This session



- Finish context
- Math preliminaries
  - XOR
  - Logarithms
  - Fields and groups

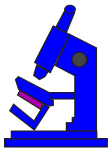
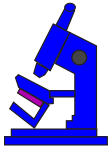
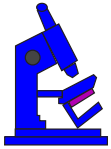


Diagram for BAG



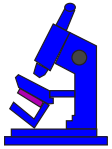
Safety/control software



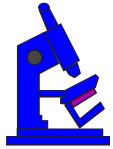
A naive approach to security might involve attempting to ensure that all programs that run on a computer are *safe*, and that all users of computer systems are *trustworthy*.

Checking even one program is a non-trivial task.

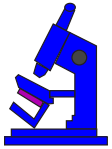
The computer *operating system* normally provides some level of software and hardware security for computer systems, combined with some level of user authorization.



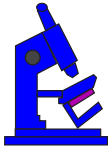
Safety/control software



- ✔ **User authorization** means passwords!
- ✔ Systems have grown in complexity over the years.
- ✔ An article shows the changes in the UNIX mechanism

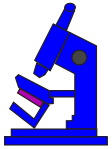


Hardware security

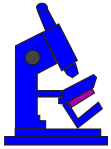


Hardware security in operating systems has been studied in CS2106 (Operating Systems) and other courses. The Kernel/Supervisor bit, processor ring0, memory protection/mapping hardware and so on are all examples of hardware security systems intended to co-operate with the OS to enhance system security.

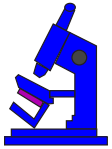
Software security in operating systems takes many forms. The forms range from ad-hoc changes to operating systems to fix security loopholes as they are found, through to operating systems built from the ground up to be secure.



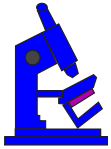
Example: network security



- ✔ TCP wrappers:
  - ✔ Attacks through poorly controlled TCP or UDP ports.
  - ✔ Wrapper provides single point of control
  - ✔ Default installation disables *all* access
  - ✔ Re-enable on a case-by-case basis.



OS security

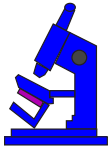


NSA have a security-enhanced Linux system:

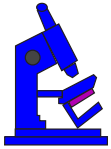
> *This version of Linux has a strong, flexible mandatory access control architecture incorporated into the major subsystems of the kernel. The system provides a mechanism to enforce the separation of information based on confidentiality and integrity requirements.*

You can read about SELinux at

http://www.nsa.gov/selinux/index.html

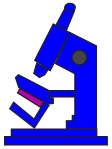


OS security

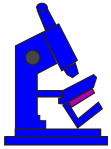


- ✔ Java virtual machine has built-in security model
- ✔ Microsoft point out that the Linux security model is weak...

  Every member of the Windows NT family since Windows NT 3.5 has been evaluated at either a C2 level under the U.S. Government's evaluation process or at a C2-equivalent level under the British Government's ITSEC process. In contrast, no Linux products are listed on the U.S. Government's evaluated product list.

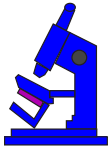


Topic: Assurance

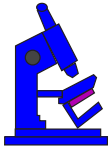


How can we convince ourselves (or our employer) that the computer system is to be trusted?

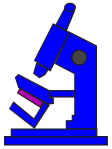
Building assurance is best done by adopting formal methods to confirm, specify and verify the behaviour of systems.



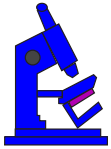
ITSEC and CC



- ✔ UK, Germany, France, Netherlands produced Information Technology Security Evaluation Criteria (ITSEC).
- ✔ IT Security Evaluation Manual (ITSEM) specifies methodology for evaluation.
- ✔ Common Criteria for Information Technology Security Evaluation is ITSEC, CTCPEC (Canadian Criteria) and US Federal Criteria
- ✔ Accepted by the ISO (ISO15408).

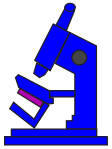


ITSEC

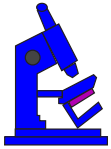


In an article, elements of the first certification of a smart-card system under the European ITSEC level 6 certification are outlined.

This process involved verification of the specification with independent systems, and a formal process for the implementation, deriving it from the specification using the *refinement* process.

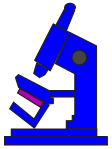


Math preliminaries

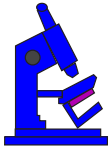


This chapter and the following chapter are copied verbatim from the "The Laws of Cryptography with Java Code", with permission from Prof Neal Wagner. The book is well worth reading and contains a lot of information that is relevant to this course. You can find the book at

http://www.cs.utsa.edu/~wagner/lawsbookcolor/laws.pdf

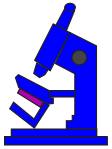


Exclusive-Or

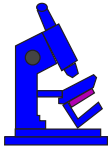


Law XOR-1:
The cryptographer's favorite function is *Exclusive-Or*.

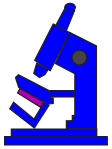
- ✔ Exclusive-Or comes up constantly in cryptography.
- ✔ Same as *addition mod 2*



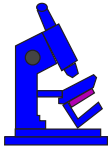
Exclusive-Or



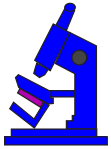
- ✔ Also as **`xor`** or a plus sign in a circle, $\oplus$.
- ✔ The expression $a \oplus b$ means either a or b *but not both*.
- ✔ Ordinary *inclusive-or* in mathematics means either one or the other *or both*.
- ✔ The exclusive-or function in C / C++ / Java for bit strings as a hat character: **^**.



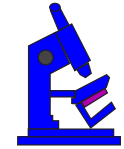
Exclusive-Or for 1-bit



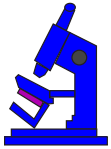
Exclusive-Or		
a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0



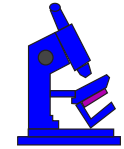
Exclusive-Or



Message	A	B	C
m	0 1 0 0 0 0 0 1	0 1 0 0 0 0 1 0	0 1 0 0 0 0 1 1 . . .
Key$= k$	0 0 0 1 0 0 1 1	0 1 1 0 0 1 0 1	0 0 1 1 1 0 0 1 . . .
$K(m) = m \oplus k$	0 1 0 1 0 0 1 0	0 0 1 0 0 1 1 1	0 1 1 1 1 0 1 0 . . .
$K(m)$	R	’	z

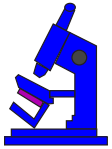


Exclusive-Or

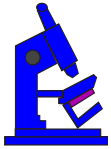


$K(m)$	R	’	z
	0 1 0 1 0 0 1 0	0 0 1 0 0 1 1 1	0 1 1 1 1 0 1 0 . . .
Key$= k$	0 0 0 1 0 0 1 1	0 1 1 0 0 1 0 1	0 0 1 1 1 0 0 1 . . .
$m = K(m) \oplus k$	0 1 0 0 0 0 0 1	0 1 0 0 0 0 1 0	0 1 0 0 0 0 1 1 . . .
Message	A	B	C

If the bit-stream is random, and not known to an eavesdropper, then this is the most secure system. It is known as a one-time-pad.



Properties of XOR



$a \oplus a = 0$

$a \oplus 0 = a$

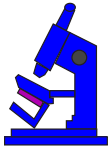
$a \oplus 1 = \sim a$, where $\sim$ is bit complement.

$a \oplus b = b \oplus a$ (commutativity)

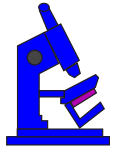
$a \oplus (b \oplus c) = (a \oplus b) \oplus c$ (associativity)

$a \oplus a \oplus a = a$

if $a \oplus b = c$, then $c \oplus b = a$ and $c \oplus a = b$.

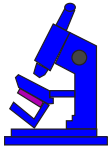


Reminder

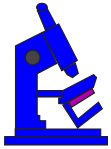


Exchange the values in two variables **a** and **b**

```
temp = a;
a = b;
b = temp;
```

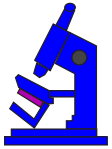


Exchange using XOR

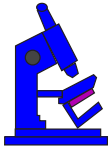


```
a = a xor b;
b = a xor b;
a = a xor b;
```

$$
\begin{aligned}
a' &= a \oplus b \\
b' &= (a \oplus b) \oplus b = a \\
a'' &= (a \oplus b) \oplus a = b
\end{aligned}
$$

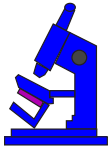


Logarithms

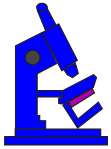


Law LOG-1:
The cryptographer's favorite logarithm is *log base 2*.

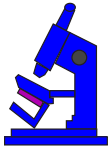
- ✔ $y = \log_b x$ is the same as $b^y = x$
- ✔ $b^{(\log_b x)} = x$
- ✔ Logarithm is inverse of exponential.



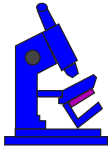
Logarithms



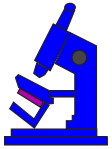
- ✔ Use logs base 2 in cryptography.
- ✔ $y = \log_2 x$ is the same as $2^y = x$
- ✔ $2^{10} = 1024$ is the same as $\log_2 1024 = 10$.
- ✔ $2^y > 0$ for all y, and
- ✔ $\log_2 x$ is not defined for $x \leq 0$.



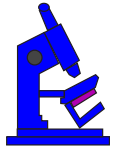
Properties of logs



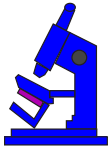
$$\log_2(ab) = \log_2 a + \log_2 b, \text{ for all } a, b > 0$$
$$\log_2(a/b) = \log_2 a - \log_2 b, \text{ for all } a, b > 0$$
$$\log_2(1/a) = \log_2(a^{-1}) = -\log_2 a, \text{ for all } a > 0$$
$$\log_2(a^r) = r \log_2 a, \text{ for all } a > 0, r$$
$$\log_2(a + b) = \text{(Oops! No simple formula for this.)}$$



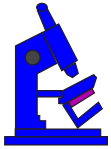
Examples



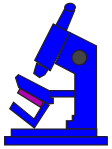
Logarithms base 2	
$x = 2^y = 2^{\log_2 x}$	$y = \log_2 x$
$1,073,741,824$	30
$1,048,576$	20
$1,024$	10
8	3
4	2
2	1
1	0



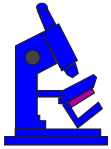
Examples



Logarithms base 2	
$x = 2^y = 2^{\log_2 x}$	$y = \log_2 x$
1	0
$1/2$	-1
$1/4$	-2
$1/8$	-3
$1/1,024$	-10
0	$-\infty$
< 0	undefined



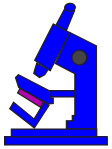
Natural logs



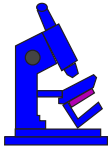
- A log base 2 is just a fixed constant times a natural log:

$$\begin{aligned} \log_2 x &= \log_e x / \log_e 2, \text{ (mathematics)} \\ &= \texttt{Math.log(x)/Math.log(2.0);} \text{ (Java).} \end{aligned}$$

- The magic constant is:
  - $\log_e 2 = 0.69314\ 71805\ 59945\ 30941\ 72321$, or
  - $1/\log_e 2 = 1.44269\ 50408\ 88963\ 40735\ 99246$.



Proof of formula



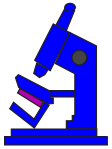
$2^y = x$, or $y = \log_2 x$ (then take $\log_e$ of each side)

$\log_e(2^y) = \log_e x$ (then use properties of logarithms)

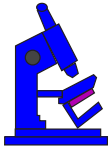
$y \log_e 2 = \log_e x$ (then solve for y)

$y = \log_e x / \log_e 2$ (then substitute $\log_2 x$ for y)

$\log_2 x = \log_e x / \log_e 2.$



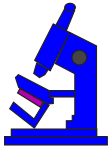
Bits to represent



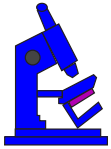
Law LOG-2:
The log base 2 of an integer x tells how many bits it takes to represent x in binary.

Thus $\log_2 10000 = 13.28771238$, so it takes 14 bits to represent 10000 in binary. (In fact, $10000_{10} = 10011100010000_2$.) Exact powers of 2 are a special case: $\log_2 1024 = 10$, but it takes 11 bits to represent 1024 in binary, as 10000000000_2.

Similarly, $\log_{10}(x)$ gives the number of decimal digits needed to represent x.

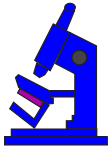


Groups

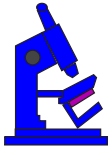


- A *group* is
  - a set of *group elements* with
  - a *binary operation* f

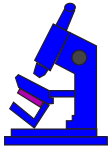
If one denotes the group operation by $\#$, then the above says that for any group elements a and b, $a\#b$ is defined and is also a group element.



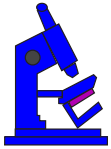
Groups



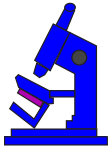
- ✔ Groups
  - ✔ are *associative*, meaning that $a\#(b\#c) = (a\#b)\#c$
  - ✔ have an *identity element* e satisfying $a\#e = e\#a = a$ for any group element a.
  - ✔ have an *inverse* a' any element a satisfying $a\#a' = a'\#a = e$.



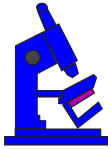
Groups



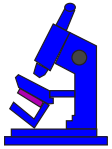
- If $a\#b = b\#a$ for all group elements a and b, the group is *commutative*.
- Otherwise it is *non-commutative*. Notice that even in a non-commutative group, $a\#b = b\#a$ might sometimes be true — for example if a or b is the identity.
- A group with only finitely many elements is called *finite*; otherwise it is *infinite*.



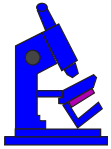
Examples



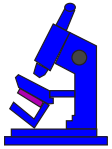
- The *integers* (all whole numbers, including 0 and negative numbers) form a group using addition. The identity is 0 and the inverse of a is $-a$.
  - This is an infinite commutative group.
- The *positive rationals* (all positive fractions, including all positive integers) form a group if ordinary multiplication is the operation. The identity is 1 and the inverse of r is $1/r = r^{-1}$.
  - This is another infinite commutative group.



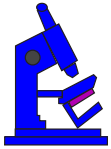
Examples



- The *integers mod n* form a group for any integer $n > 0$. This group is often denoted Z_n. Here the elements are 0, 1, 2, $\ldots$, $n-1$ and the operation is addition followed by remainder on division by n. The identity is 0 and the inverse of a is $n-a$ (except for 0 which is its own inverse).
  - This is a finite commutative group.



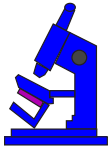
Non-commutative Group



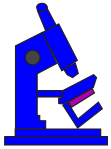
Consider 2-by-2 non-singular matrices of real numbers (or rationals), where the operation is matrix multiplication: $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$. Here a, b, c, and d are real numbers (or rationals) and $ad - bc$ must be non-zero. Inverse is

$$\frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

and the identity is $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. This is an infinite non-commutative group.

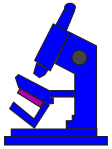


Groups

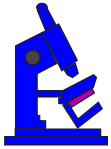


Law GROUP-1:
The cryptographer's favorite group is the *integers mod n*, Z_n.

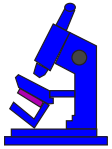
In the special case of $n = 10$, the operation of addition in Z_{10} can be defined by $(x + y) \bmod 10$, that is, divide by 10 and take the remainder.



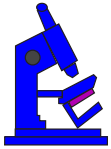
Integers modulo 10



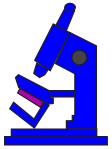
+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8



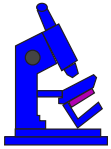
Fields



- A field has two operations
  - $+$, with elements of the field forming a commutative group. Identity is 0 and inverse of a is $-a$.
  - $*$, with elements of the field except 0 forming another commutative group, identity denoted by 1 and inverse of a denoted by a^{-1}.



Fields

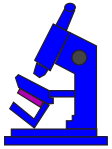


- There is also the *distributive identity*, linking $+$ and $*$:

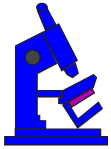
$$a * (b + c) = (a * b) + (a * c)$$

- Exclude *divisors of zero*, that is, non-zero elements whose product is zero.

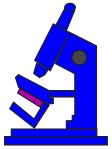
- Equivalent to the following cancellation property: if c is not zero and $a * c = b * c$, then $a = b$.



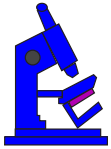
Examples



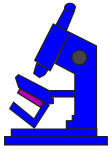
- ✔ The *rational numbers* (fractions) Q, or the *real numbers* $\mathbb{R}$, or the *complex numbers* $\mathbb{C}$, using ordinary addition and multiplication (extended in the last case to the complex numbers).
  - ✔ These are all infinite fields.



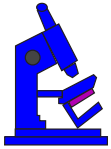
Example: integers mod p



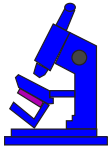
- The *integers mod p*, denoted Z_p, where p is a prime number ($2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots$).
  - A group using $+$.
  - Elements without 0 form a group under $*$.
  - The identity is clearly 1, but
  - the inverse of a non-zero element a is not obvious.



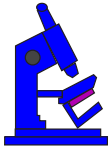
Integers mod p inverse



- In Java, inverse must be x satisfying $(x * a)\%p == 1$.
- Find x using the *extended Euclidean algorithm*:
  - p is prime and a is non-zero, the greatest common divisor of p and a is 1.
  - The extended Euclidean algorithm gives x and y satisfying $x * a + y * p = 1$, or $x * a = 1 - y * p$,
  - and x is the inverse of a.

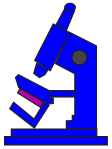


Field

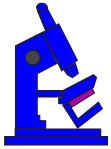


Law FIELD-1:
The cryptographer's favorite field is the *integers mod p*, denoted Z_p, where *p* is a prime number.

The above field is the only one with p elements. In other words, the field is unique up to renaming its elements, meaning that one can always use a different set of symbols to represent the elements of the field, but it will still be essentially the same.

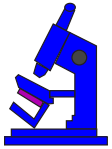


Another Field

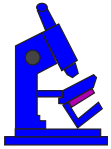


Law FIELD-2:
The cryptographer's *other* favorite field is *GF(2^n)*.

- ✔ A finite field with p^n elements for any integer $n > 1$, denoted $GF(p^n)$.
- ✔ Useful in cryptography with $p = 2$, that is, with 2^n elements for $n > 1$.
- ✔ The case $2^8 = 256$ is used, for example, in the new U.S. Advanced Encryption Standard (AES).

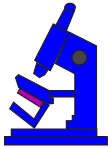


Fermat's Theorem

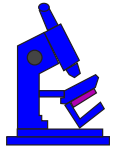


Law FERMAT-1:
The cryptographer's favorite theorem is Fermat's Theorem.

- ✔ In cryptography, one often wants to raise a number to a power, modulo another number.
- ✔ For the integers mod p where p is a prime (denoted Z_p), there is a result know as Fermat's Theorem, discovered by the 17th century French mathematician Pierre de Fermat, 1601-1665.

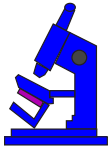


Fermat's theorem

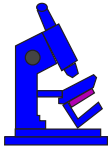


Theorem (Fermat): If p is a prime and a is any non-zero number less than p, then

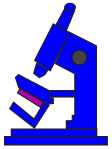
$$a^{p-1} \bmod p = 1$$



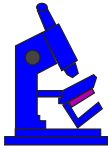
Fermat's theorem



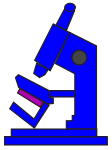
p	a	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}
13	2	2	4	8	3	6	12	11	9	5	10	7	1
13	3	3	9	1	3	9	1	3	9	1	3	9	1
13	4	4	3	12	9	10	1	4	3	12	9	10	1
13	5	5	12	8	1	5	12	8	1	5	12	8	1
13	6	6	10	8	9	2	12	7	3	5	4	11	1
13	7	7	10	5	9	11	12	6	3	8	4	2	1
13	8	8	12	5	1	8	12	5	1	8	12	5	1
13	9	9	3	1	9	3	1	9	3	1	9	3	1
13	10	10	9	12	3	4	1	10	9	12	3	4	1
13	11	11	4	5	3	7	12	2	9	8	10	6	1
13	12	12	1	12	1	12	1	12	1	12	1	12	1



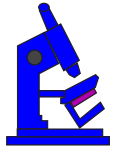
Fermat's theorem



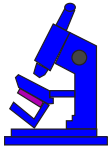
- ✔ For $p = 13$ the value is always 1 by the time the power gets to 12
- ✔ Sometimes the value gets to 1 earlier
- ✔ Lengths of runs are always numbers that divide evenly into 12
- ✔ A value of a for which the whole row is needed is called a *generator*. 2, 6, 7, and 11 are generators.



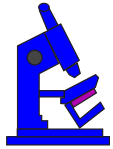
Summary of topics



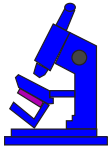
In this section, we introduced “Cryptographers favorites”

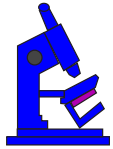


Further study



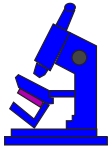
- The Laws of Cryptography with Java Code
  http://www.cs.utsa.edu/~wagner/lawsbookcolor/laws.pdf



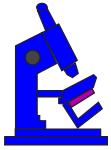


Chapter 3

Lecture 3 - Preliminaries

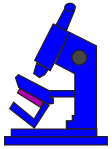


Overheads and notes

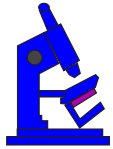


You can find all sorts of stuff looking in

http://www.comp.nus.edu.sg/~cs3235/2003-semesterI/



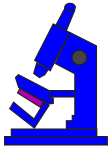
Question box



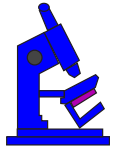
If you have any questions, feel free to place them in the question box...

Or stick your hand up...

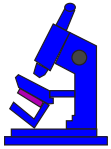
Or...



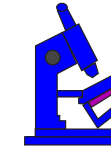
Last session



- Finish context
- Math preliminaries
  - XOR
  - Logarithms
  - Fields and groups

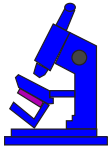


Recap - exclusive-or

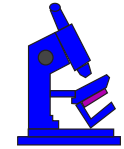


Law XOR-1:
The cryptographer's favorite function is *Exclusive-Or*.

Message	A	B	C
m	0 1 0 0 0 0 0 1	0 1 0 0 0 0 1 0	0 1 0 0 0 0 1 1 . . .
Key$= k$	0 0 0 1 0 0 1 1	0 1 1 0 0 1 0 1	0 0 1 1 1 0 0 1 . . .
$K(m) = m \oplus k$	0 1 0 1 0 0 1 0	0 0 1 0 0 1 1 1	0 1 1 1 1 0 1 0 . . .
$K(m)$	R	'	z

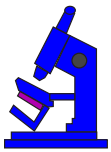


Exclusive-Or

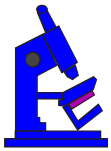


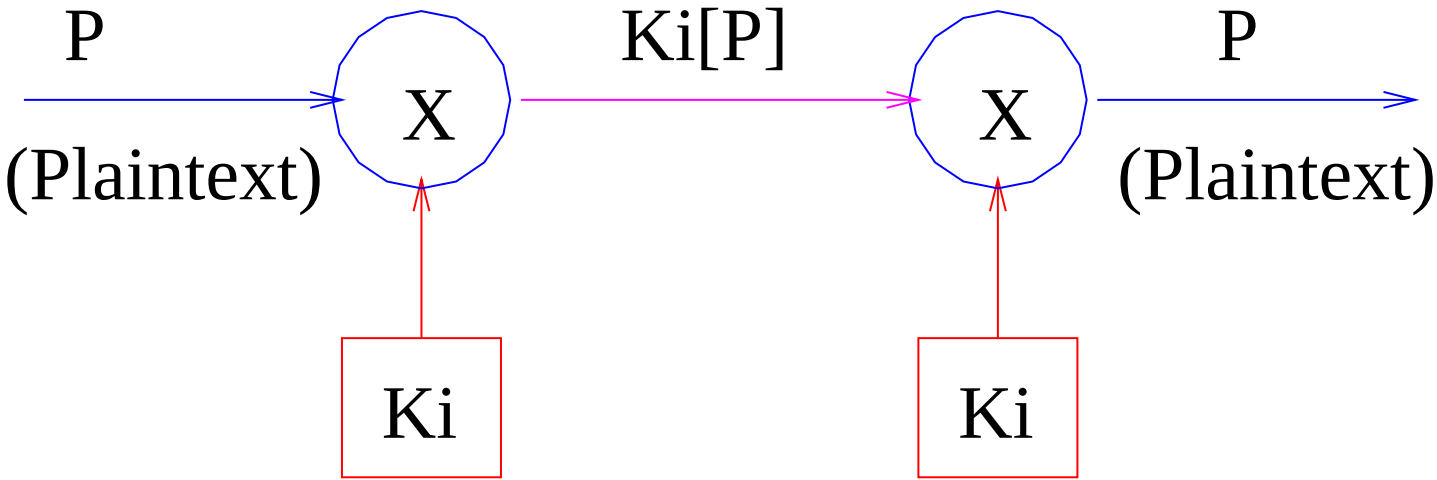
$K(m)$	R	’	z
	0 1 0 1 0 0 1 0	0 0 1 0 0 1 1 1	0 1 1 1 1 0 1 0 . . .
Key$= k$	0 0 0 1 0 0 1 1	0 1 1 0 0 1 0 1	0 0 1 1 1 0 0 1 . . .
$m = K(m) \oplus k$	0 1 0 0 0 0 0 1	0 1 0 0 0 0 1 0	0 1 0 0 0 0 1 1 . . .
Message	A	B	C

If the bit-stream for the key k is random, and not known to an eavesdropper, then this is the most secure system. It is known as a one-time-pad.

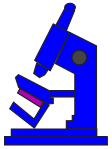


Another diagram

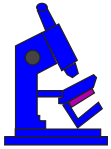




(Compare with previous representations).

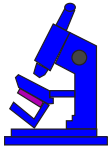


Logarithms

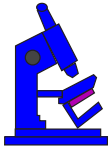


Law LOG-1:
The cryptographer's favorite logarithm is *log base 2*.

- ✔ $y = \log_b x$ is the same as $b^y = x$
- ✔ $b^{(\log_b x)} = x$
- ✔ Logarithm is inverse of exponential.

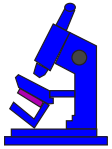


Groups

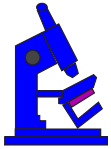


- A *group* is
  - a set of *group elements* with
  - a *binary operation*

Law GROUP-1:
The cryptographer's favorite group is the *integers mod n*, Z_n.



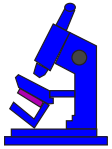
Fields



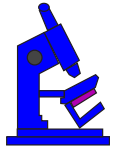
- A field has two operations
  - $+$, with elements forming a commutative group.
  - $*$, with elements$\backslash 0$ forming another group,

Law FIELD-1:
The cryptographer's favorite field is the *integers mod p*, denoted Z_p, where *p* is a prime number.

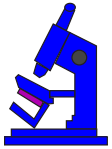
Law FIELD-2:
The cryptographer's *other* favorite field is *GF(2^n)*.



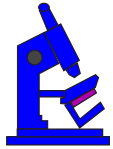
This session



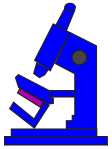
- Math preliminaries
  - Fermat's little theorem
  - Euler



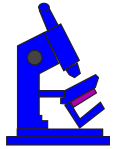
This session



- Math preliminaries
  - Fermat's little theorem
  - Euler

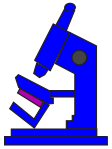


Fermat's theorem

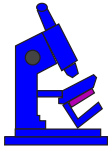


Theorem (Fermat): If p is a prime and a is any non-zero number less than p, then

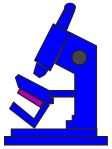
$$a^{p-1} \bmod p = 1$$



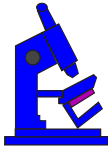
Fermat's theorem, $p = 13$



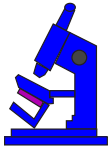
p	a	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}
13	2	2	4	8	3	6	12	11	9	5	10	7	1
13	3	3	9	1	3	9	1	3	9	1	3	9	1
13	4	4	3	12	9	10	1	4	3	12	9	10	1
13	5	5	12	8	1	5	12	8	1	5	12	8	1
13	6	6	10	8	9	2	12	7	3	5	4	11	1
13	7	7	10	5	9	11	12	6	3	8	4	2	1
13	8	8	12	5	1	8	12	5	1	8	12	5	1
13	9	9	3	1	9	3	1	9	3	1	9	3	1
13	10	10	9	12	3	4	1	10	9	12	3	4	1
13	11	11	4	5	3	7	12	2	9	8	10	6	1
13	12	12	1	12	1	12	1	12	1	12	1	12	1



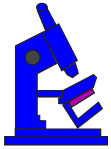
Fermat's theorem, $p = 13$



- ✔ Lengths of runs are always numbers that divide evenly into 12
- ✔ A value of a for which the whole row is needed is called a *generator*. 2, 6, 7, and 11 are generators.



An interesting observation..

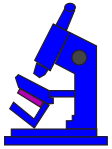


Because a to a power mod p always starts repeating after the power reaches $p-1$, you can do this:

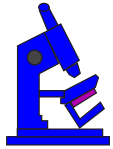
$$a^x \bmod p = a^{x \bmod (p-1)} \bmod p.$$

Thus modulo p in the expression requires modulo $p-1$ in the exponent. For $p = 13$ as above, then

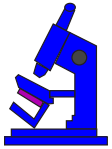
$$a^{29} \bmod 13 = a^{29 \bmod 12} \bmod 13 = a^5 \bmod 13.$$



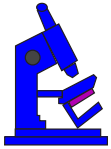
Another example



$$\text{result} = 7^{1215} \bmod 13$$

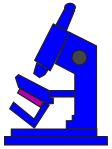


Another example

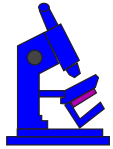


result=

622470275067322737046556455907979268906239864832921913090207877109248699107274058706519890781017383899497826793481300967770892782660131355777365361484044783800851222817392261341421370762400507026834564501614788818580162335818155077291900607338638109858209984177537766703728681473967012031571239691400018482234035235590645515566753410247396453541377412583676260706359331048403293779053704648771069764131865422622995052805575842805741858026942132998022801793254945606289489407393444822846491511971411686989595879473202428574269018023244940256710105083114967356334295809219455711191131246974627173111242792554453321165049145300772419961893572985086052067801207898808355252223419405145855673208684204238889320915704079986487190106499123086028865754587854838031902109935110264503891544145872580747830622294066978047059698088822497677940491279201763309541131855593877680081677862469580790\4970578719259627712779630348778181410614737537090462719599558908727684699431 mod 13 = 5

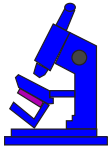


How did I work that out?

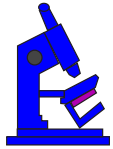


I used bc

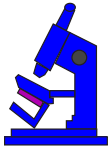
An arbitrary precision calculator language



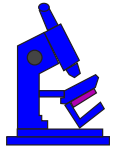
Another example



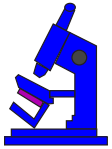
result $= 7^{1215} \bmod 13$



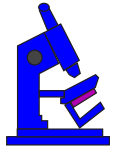
Another example



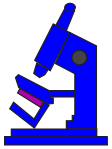
$$\begin{aligned} \text{result} &= 7^{1215} \bmod 13 \\ &= 7^{1215} \bmod 12 \bmod 13 \end{aligned}$$



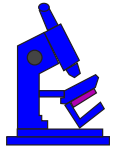
Another example



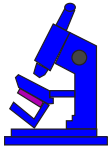
$$
\begin{aligned}
\text{result} &= 7^{1215} \bmod 13 \\
&= 7^{1215 \bmod 12} \bmod 13 \\
&= 7^{3} \bmod 13
\end{aligned}
$$



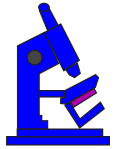
Another example



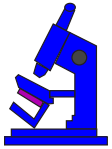
$$
\begin{aligned}
\text{result} &= 7^{1215} \bmod 13 \\
&= 7^{1215 \bmod 12} \bmod 13 \\
&= 7^{3} \bmod 13 \\
&= 343 \bmod 13
\end{aligned}
$$



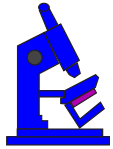
Another example



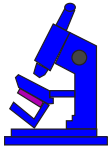
$$
\begin{aligned}
\text{result} &= 7^{1215} \bmod 13 \\
&= 7^{1215 \bmod 12} \bmod 13 \\
&= 7^{3} \bmod 13 \\
&= 343 \bmod 13 \\
&= 5
\end{aligned}
$$



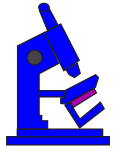
Summary



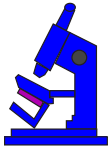
We can do BIG NUMBER maths without calculating big numbers.



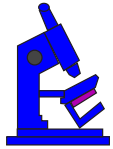
This session



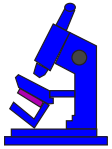
- Math preliminaries
  - Fermat's little theorem
  - Euler



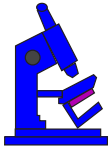
Euler



The Swiss mathematician Leonhard Euler (1707-1783) discovered a generalization of Fermat's Theorem which will later be useful in the discussion of the RSA cryptosystem.



Euler's theorem



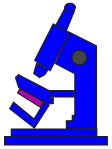
Theorem (Euler): If n is any positive integer and a is any positive integer less than n with no divisors in common with n, then

$$a^{\phi(n)} \bmod n = 1,$$

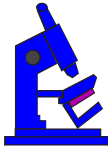
where $\phi(n)$ is the *Euler phi function*:

$$\phi(n) = n(1 - 1/p_1) \ldots (1 - 1/p_m),$$

and $p_1, \ldots, p_m$ are all the prime numbers that divide evenly into n, including n itself in case it is a prime.



Special case 1

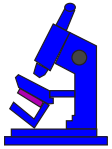


✔ If n is a prime, then using the formula,

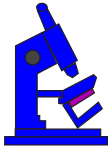
$$\phi(n) = n(1 - 1/n) = n(\frac{n-1}{n}) = n - 1$$

Fermat's result is a special case of Euler's.

$$a^{\phi(n)} \bmod n = a^{n-1} \bmod n = 1$$

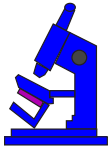


Special case 2

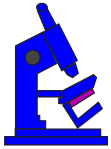


✔ Another special case needed for RSA comes when the modulus is a product of two primes: $n = pq$. Then

$$\phi(n) = n(1 - 1/p)(1 - 1/q) = (p - 1)(q - 1)$$

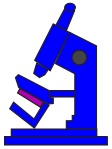


Special case 2

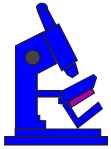


$$a^{(p-1)(q-1)} \bmod pq = 1$$

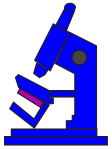
- assuming a has no divisors in common with pq
- and p and q are primes



Euler: $n = 15$ and $\phi(n) = 8$



a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{1}
2	**4**	**8**	**1**	2	4	8	1	2	4	8	1	2	
3	**9**	**12**	**6**	**3**	**9**	**12**	**6**	**3**	**9**	**12**	**6**	**3**	
4	**1**	4	1	4	1	4	1	4	1	4	1	4	
5	**10**	**5**	**10**	**5**	**10**	**5**	**10**	**5**	**10**	**5**	**10**	**5**	**1**
6	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	**6**	
7	**4**	**13**	**1**	7	4	13	1	7	4	13	1	7	
8	**4**	**2**	**1**	8	4	2	1	8	4	2	1	8	
9	**6**	**9**	**6**	**9**	**6**	**9**	**6**	**9**	**6**	**9**	**6**	**9**	
10	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**1**
11	**1**	11	1	11	1	11	1	11	1	11	1	11	
12	**9**	**3**	**6**	**12**	**9**	**3**	**6**	**12**	**9**	**3**	**6**	**12**	
13	**4**	**7**	**1**	13	4	7	1	13	4	7	1	13	
14	**1**	14	1	14	1	14	1	14	1	14	1	14	



Table

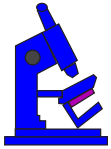
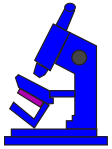


Table illustrates Euler's theorem for $n = 15 = 3 \cdot 5$, with

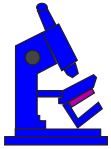
$$\phi(15) = 15 \cdot (1 - 1/3) \cdot (1 - 1/5) = (3 - 1) \cdot (5 - 1) = 8$$

Notice here that a 1 is reached when the power is 8, but only for numbers with no divisors in common with 15.

For other base numbers, the value never gets to 1.



Euler

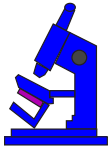


Arithmetic in the exponent is taken mod $\phi(n)$, so that, if a has no divisors in common with n,

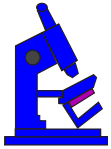
$$a^x \bmod n = a^{x \bmod \phi(n)} \bmod n.$$

If $n = 15$ as above, then $\phi(n) = 8$, and if neither 3 nor 5 divides evenly into a, then $\phi(n) = 8$. Thus for example,

$$a^{28} \bmod 15 = a^{28 \bmod 8} \bmod 15 = a^4 \bmod 15.$$

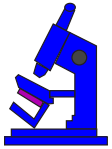


Before we leave Euler...

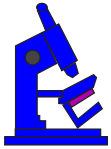


We are interested in...

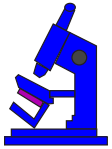
- ✔ Large prime numbers (p, q)
- ✔ Their product $n = pq$
- ✔ The Euler phi function $\phi(n) = (p-1)(q-1)$



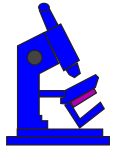
Before we leave Euler...



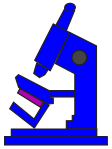
- ✔ In a similar fashion to before we can do BIG number arithmetic easily
- ✔ Consider also the ease of multiplying, and difficulty of factoring...



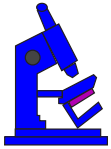
Before we leave Euler...



29*37=?

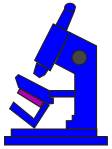


The Euclidean algorithm

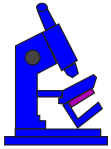


- ✔ Multiplicative inverse is not intuitive and requires some theory to compute.
- ✔ a^{-1} can be computed efficiently using *the extended Euclidean algorithm*

Law GCD-1:
The cryptographer's first and oldest favorite algorithm is the *extended Euclidean algorithm*, which computes the greatest common divisor of two positive integers *a* and *b* and also supplies integers *x* and *y* such that *x*a + y*b = gcd(a, b)*.

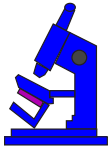


Finding GCD

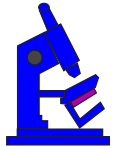


- For the gcd of 819 and 462,
  - factor the numbers as:
    - $819 = 3 \cdot 3 \cdot 7 \cdot 13$
    - $462 = 2 \cdot 3 \cdot 7 \cdot 11$
  - gcd is $21 = 3 \cdot 7$

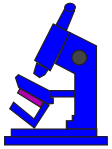
But there is no efficient algorithm to factor integers.



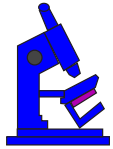
The Euclidean algorithm



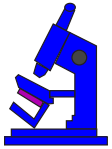
1. Repeatedly divide the **`larger`** one by the **`smaller`**, and
2. Write **`larger = smaller * quotient + remainder`**
3. Repeat using the two numbers "**`smaller`**" and "**`remainder`**".
4. When you get a 0 **`remainder`**, then you have the **`gcd`** of the original two numbers.



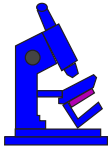
Example



$$
\begin{aligned}
819 &= 462 \cdot 1 + 357 && \text{(Step 0)} \\
462 &= 357 \cdot 1 + 105 && \text{(Step 1)} \\
357 &= 105 \cdot 3 + 42 && \text{(Step 2)} \\
105 &= 42 \cdot 2 + 21 && \text{(Step 3, so GCD = 21)} \\
42 &= 21 \cdot 2 + 0 && \text{(Step 4)}
\end{aligned}
$$



The extended Euclidean algorithm



Given the two positive integers 819 and 462, the extended Euclidean algorithm finds unique integers a and b so that

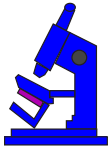
$$a \cdot 819 + b \cdot 462 = \gcd(819, 462) = 21$$

In this case,

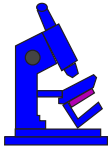
$$(-9) \cdot 819 + 16 \cdot 462 = 21$$

(See notes...)

✘ How does this give us a mechanism to calculate the multiplicative inverse of an element?



The extended Euclidean algorithm



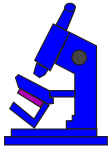
$$x * a + y * p = \gcd(x, y)$$

Now - if p is a prime, then $\gcd(x, y) = 1$, and so

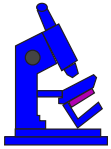
$$x * a + y * p = 1$$

In the field Z_p, this indicates that $x * a = 1$, and so $x = a^{-1}$.

The extended Euclidean algorithm has given us a mechanism to calculate the multiplicative inverse of an element.

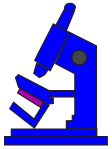


Fast integer exponentiation

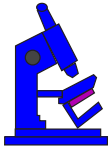


Law EXP-1:
Many cryptosystems in modern cryptography depend on a fast algorithm to perform integer exponentiation.

Examples in notes... not so important, just nice to know it can be done.



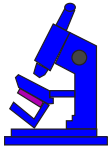
Back to primes



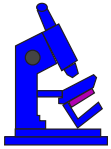
For 2500 years mathematicians studied prime numbers just because they were interesting, without any idea they would have practical applications. Possible real-world uses:

1. Sometimes... a prime number of ball bearings arranged in a bearing, to cut down on periodic wear (also gear teeth).

2. Possibly... the 13 and 17-year periodic emergence of cicadas may be due to coevolution with predators (that lost and became extinct).

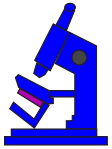


Since 1976

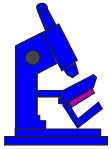


Now finally, in cryptography, prime numbers have come into their own.

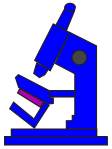
Law PRIME-1:
A source of large random prime integers is an essential part of many current cryptosystems.



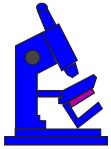
Checking for primes



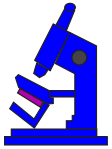
- ✔ It is hard to check that an integer is “certainly” prime, but...
- ✔ It is easy to check that an integer is “probably” prime.
- ✔ Tests to check if a number is probably prime are called *pseudo-prime* tests.



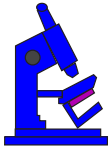
Prime check



- ✔ Start with a property of a prime number, such as Fermat's Theorem, mentioned in the previous chapter
- ✔ if p is a prime and a is any non-zero number less than p, then $a^{p-1} \bmod p = 1$.
- ✔ If one can find a number a for which Fermat's Theorem does not hold, then the number p in the theorem is *definitely not a prime*.
- ✔ If the theorem holds, then p is called *a pseudo-prime with respect to* a, and it might actually be a prime.

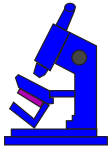


Prime check

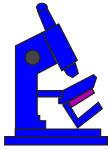


So the simplest possible pseudo-prime test would just take a small value of a, say 2 or 3, and check if Fermat's Theorem is true.

> **Simple Pseudo-prime Test:** If a very large random integer p (100 decimal digits or more) is not divisible by a small prime, and if $3^{p-1} \bmod p = 1$, then the number is prime except for a vanishingly small probability, which one can ignore.



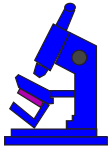
Prime check - 1105,1729



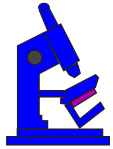
- ✔ One could just repeat the test for other integers besides 3 as the base, but unfortunately there are non-primes (called *Carmichael numbers*) that satisfy Fermat's theorem for all values of a even though they are not prime.

- ✔ Chances of a mistake less than 10^{-41}, in practice use better tests

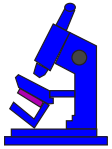
Law PRIME-2:
Just one simple pseudo-prime test is enough to test that a very large random integer is probably prime.



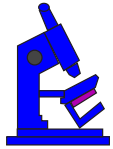
Summary of topics



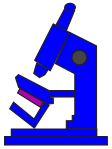
- ✔ We can do BIG arithmetic in these fields
- ✔ We can do fast exponentiation and modulo arithmetic
- ✔ We can check for primes



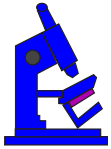
Chapter 4



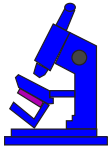
Lecture 4 - Preliminaries



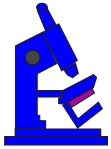
Chocolate fish people



- ✔ Andreas Schuth
- ✔ Chong Jun Yong
- ✔ Ashley Ng *
- ✔ Wu Yongzheng *
- ✔ Zhang Huaixing *
- ✔ Terence Sangeet



The extended Euclidean algorithm

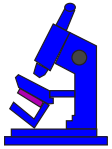


$$x * a + y * p = \gcd(x, y)$$

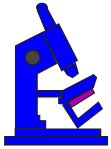
Now - if p is a prime, then $\gcd(x, y) = 1$, and so

$$x * a + y * p = 1$$

WRONG!



The extended Euclidean algorithm

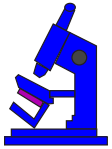


$$x * a + y * p = \gcd(a, p)$$

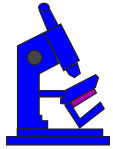
Now - if p is a prime, then $\gcd(a, p) = 1$, and so

$$x * a + y * p = 1$$

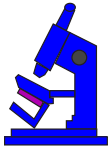
RIGHT!



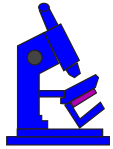
Last session



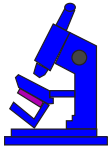
- Math preliminaries
  - Fermat's little theorem
  - Euler



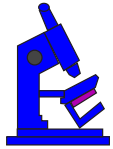
This session



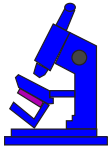
- Physical preliminaries
- Entropy



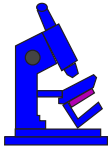
This session



- Physical preliminaries
- Entropy

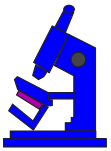


Preliminaries - physical

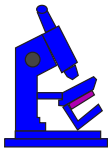


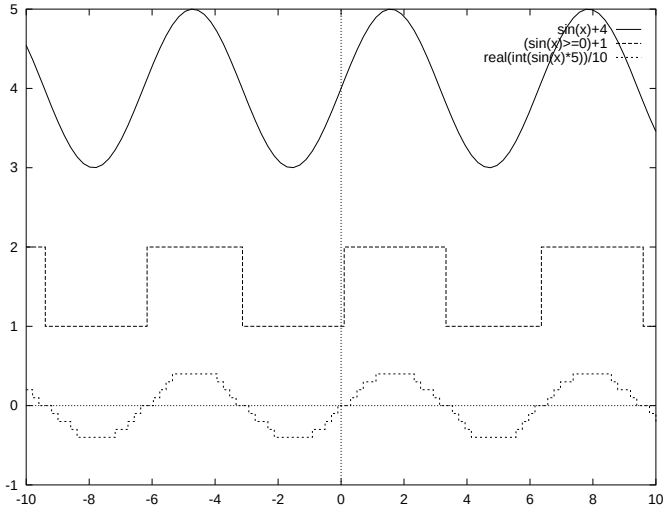
Consider:

- Is the data analog or digital?
- What limits are placed on it?
- How is it to be transmitted?
- How can you be sure that it is correct/accurate?

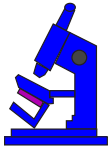


Analog and digital

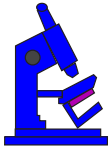




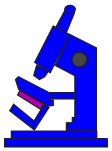
The plot is **amplitude versus time**.



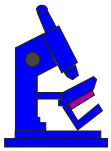
Analog and digital

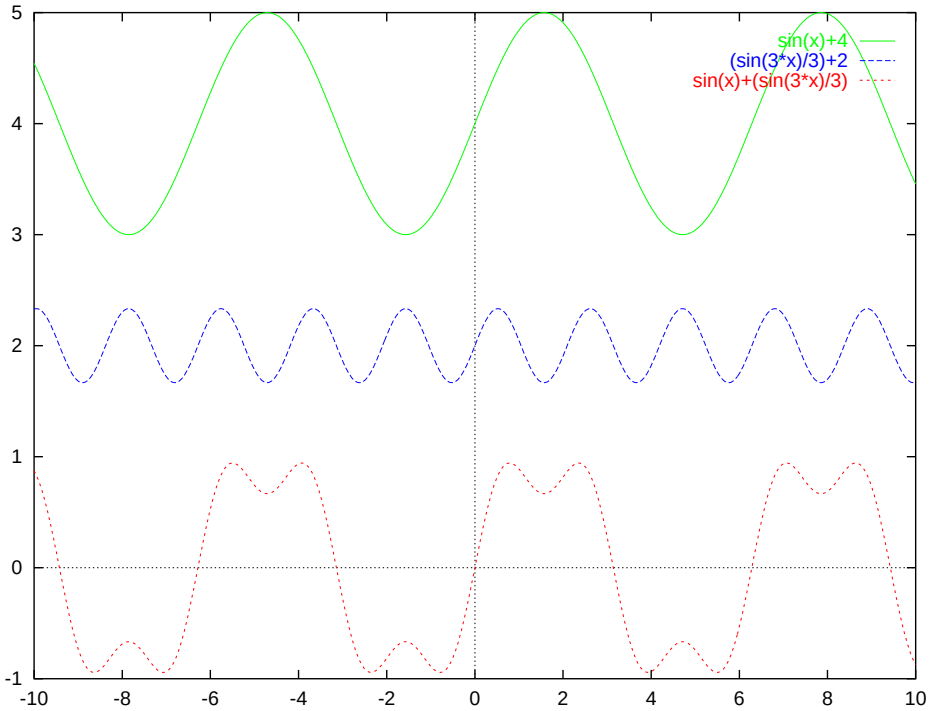


- ✔ Repetition rate (if it repeats) is called the *frequency*, and is measured in Hertz
- ✔ The peak to peak signal level is called the *amplitude*.
- ✔ The simplest analog signal is called the sine wave.
- ✔ By mixing we may create any desired periodic waveform.

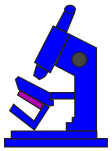


Analog and digital

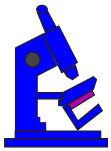


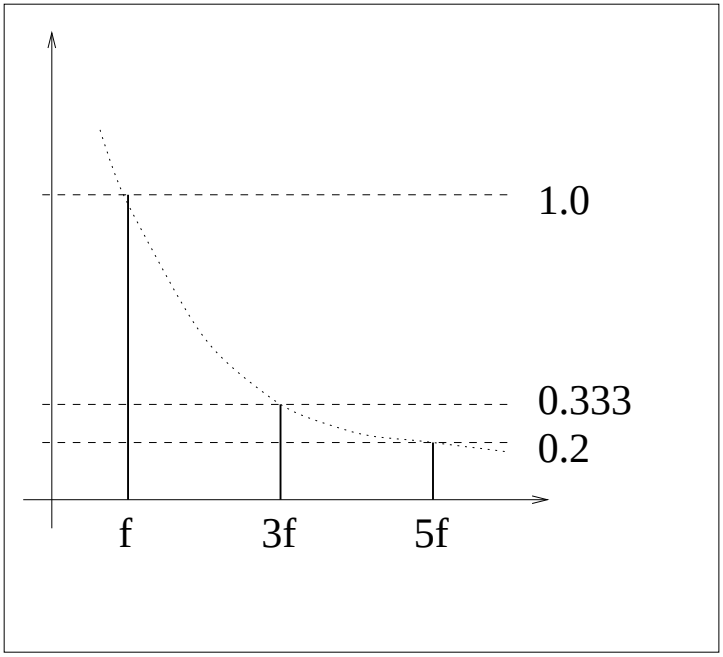


The plot is **amplitude versus time**. (Time domain)

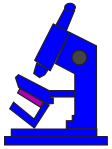


Analog and digital

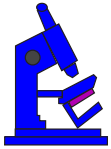




The plot is **amplitude vs frequency**. (Frequency domain).



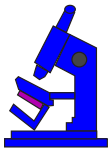
Analog and digital



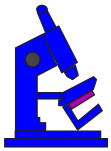
If we were to continue in the same progression, the resultant waveform would be a square wave:

$$\sum_{n=1}^{\infty} \frac{1}{n} \sin(2\pi n f) \text{ (for odd n) } \Rightarrow \text{ square wave, frequency } f$$

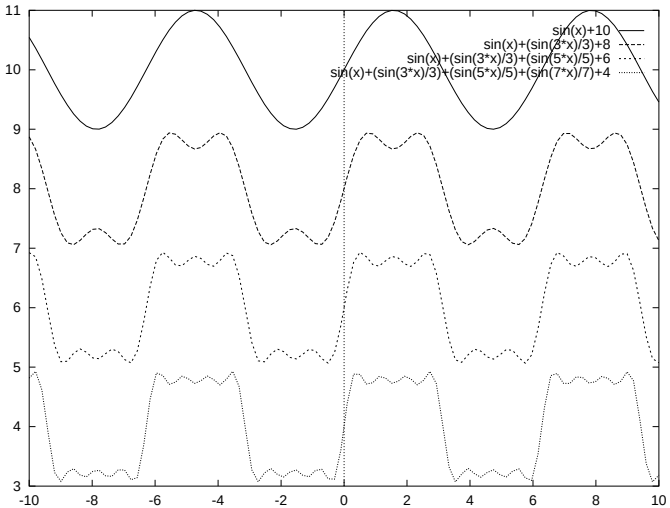
This representation method is known as *Fourier Analysis* after Jean-Baptiste Fourier.

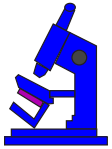


Fourier analysis

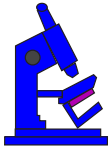


$$\frac{4}{\pi}(sin(2\pi ft) + \frac{1}{3}sin(6\pi ft) + \frac{1}{5}sin(10\pi ft) + \frac{1}{7}sin(14\pi ft) + ...)$$





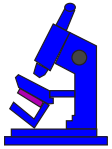
Fourier analysis



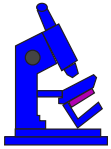
Transformation between equivalent time domain and frequency domain representations.

A piecewise continuously differentiable periodic function in the time domain may be transformed to a discrete aperiodic function in the frequency domain.

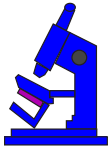
$$\begin{aligned} \text{smooth, repeating} &\leftrightarrow \text{pointy, notrepeating} \\ f(t) &\leftrightarrow F(\omega) \end{aligned}$$



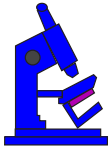
Fourier analysis



Time domain		Frequency domain	Description
Continuous, periodic	⇄	Discrete, aperiodic	**Fourier series**
Continuous, aperiodic	⇄	Continuous, aperiodic	**Fourier transform**
Discrete, periodic	⇄	Discrete, periodic	**Discrete Fourier series**
Discrete, aperiodic	⇄	Continuous, periodic	**Discrete Fourier transform**

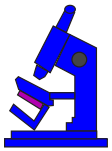


Accuracy

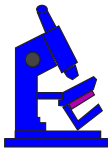


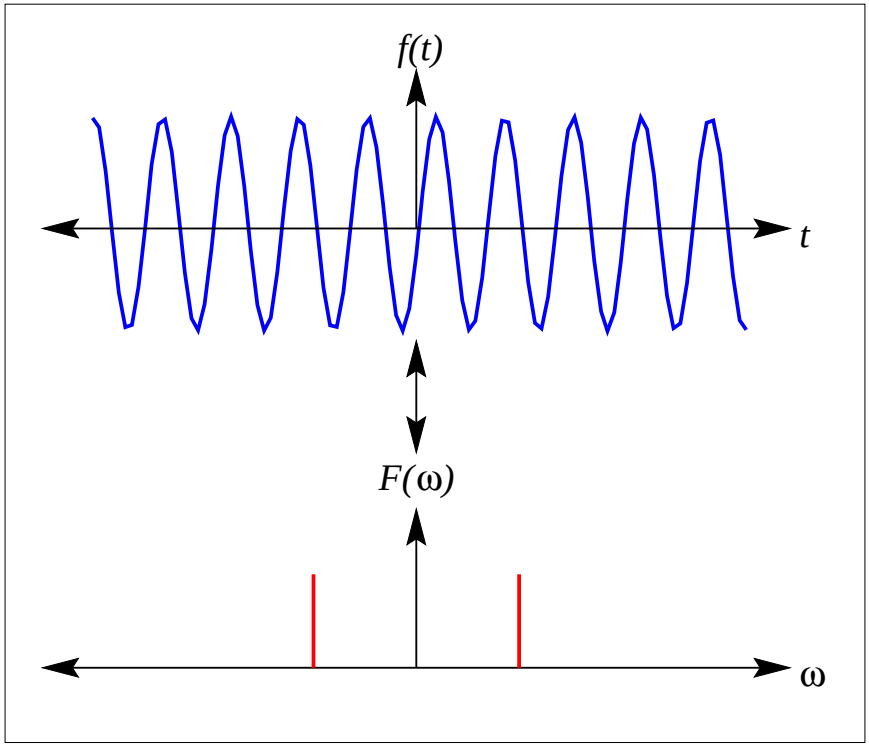
Relationship between the bandwidth of a channel, and *how accurate* a signal is.

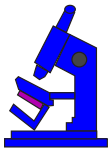
Another way of stating this is to point out that the higher frequency components are important - they are needed to re-create the original signal faithfully. If we had two 1,000Hz signals, one a triangle, one a square wave - if they were both passed through the 1,000Hz bandwidth limited channel above, they would look identical (a sine wave).



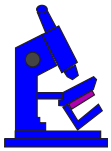
Example transforms

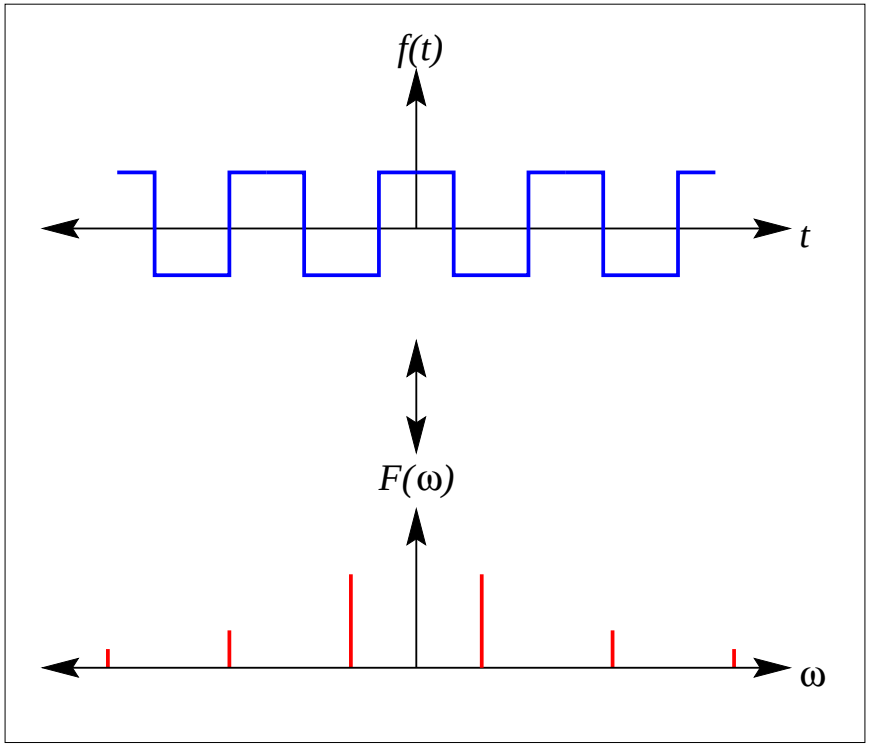


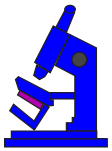




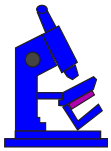
Example transforms

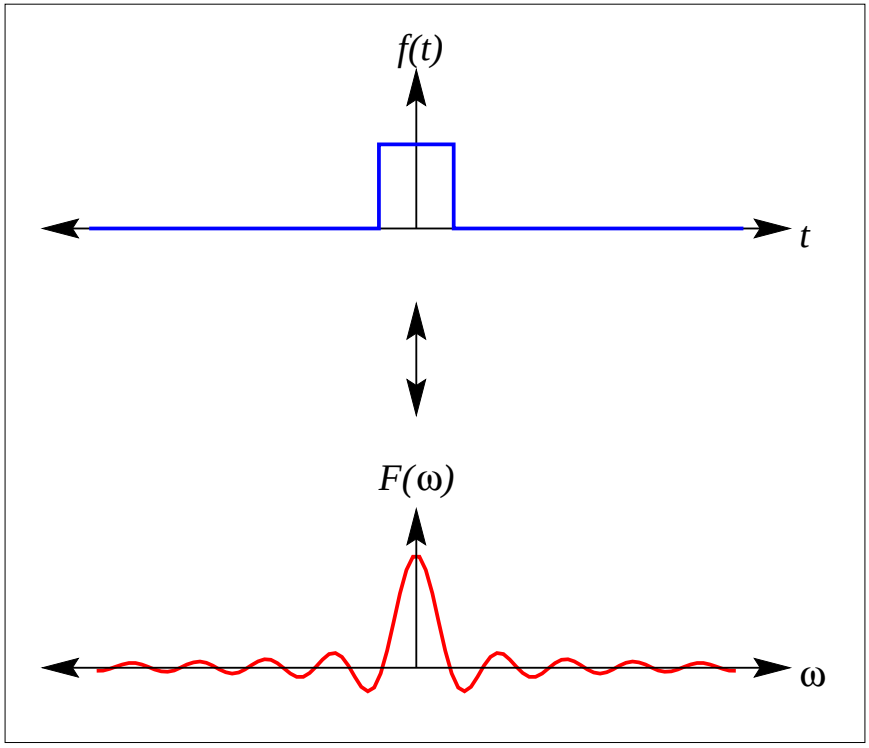


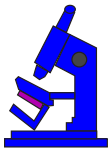




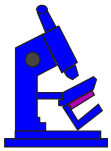
Example transforms

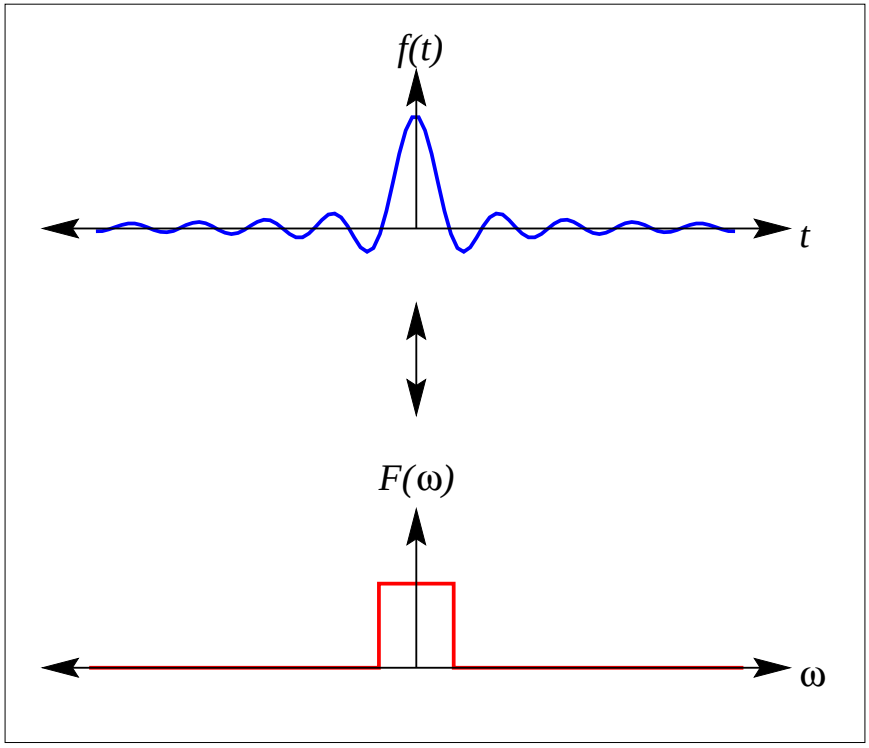


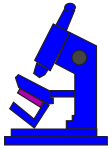




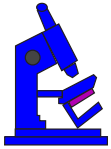
Example transforms







Convolution

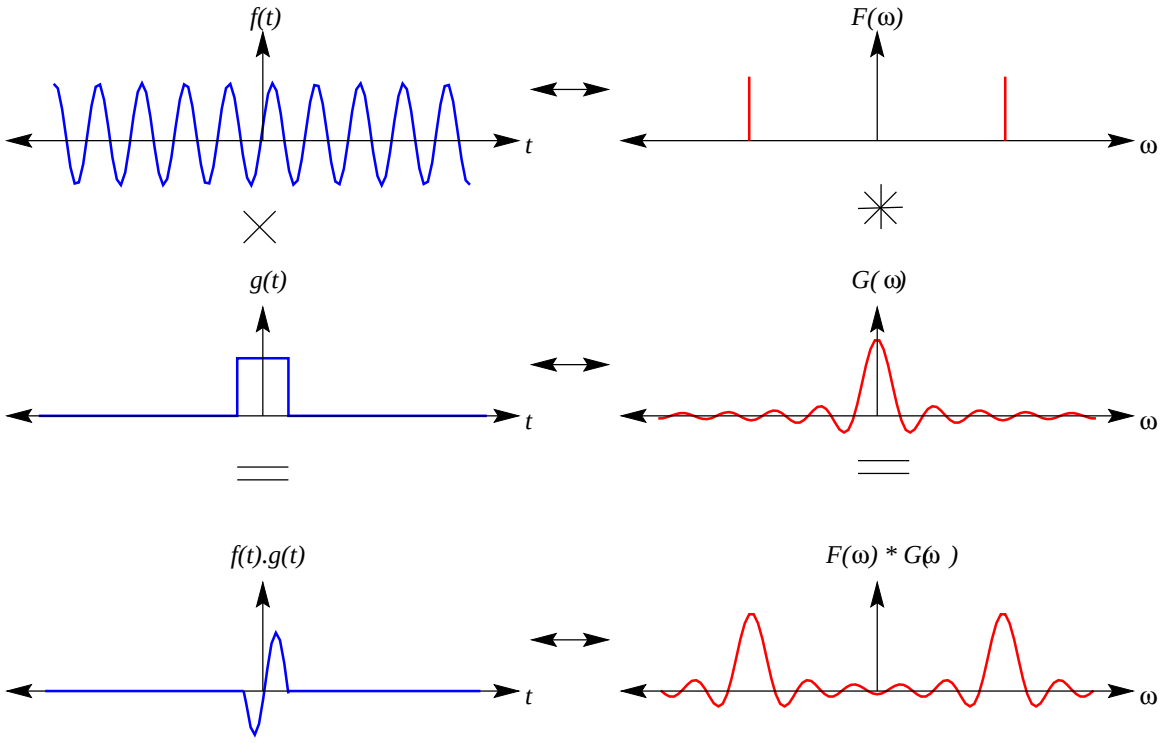


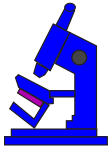
The Fourier transform of the convolution $f(t) \star g(t)$ is the product of the Fourier transforms of the functions $F(\omega)$ and $G(\omega)$, *and vice versa*.

$$f(t) \star g(t) \quad \leftrightarrow \quad F(\omega) \times G(\omega)$$
$$f(t) \times g(t) \quad \leftrightarrow \quad F(\omega) \star G(\omega$$

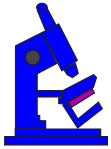
We can use convolution to easily predict the functions that result from complex signal filtering or sampling.

Convolution





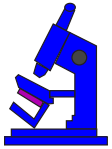
Modulation



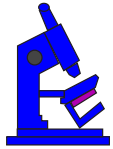
A baseband signal is one in which the data is directly converted to a signal and transmitted. When the signal is imposed on another signal, the process is called modulation.

We may *modulate* for several reasons:

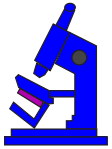
- The media may not support the baseband signal
- We may wish to use a single transmission medium to transport many signals



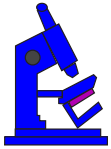
Modulation methods



- Frequency modulation - frequency shift keying (FSK)
- Amplitude modulation
- Phase modulation - phase shift keying (PSK)
- Combinations of the above (QAM)

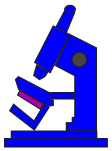


Baseband digital encoding

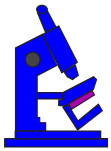


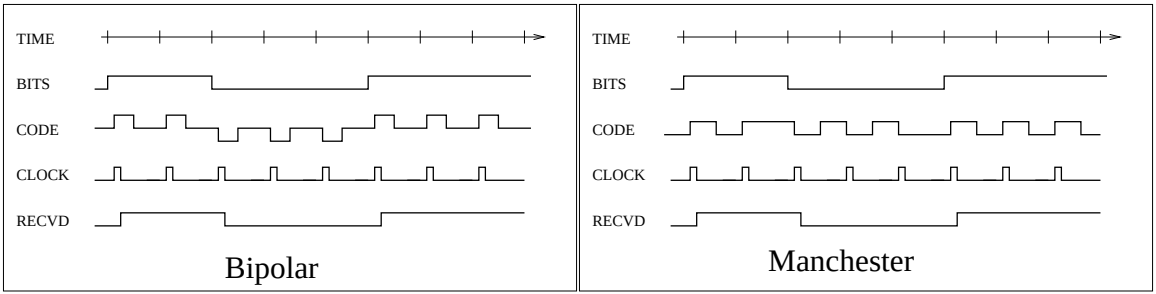
The simplest encoding scheme is just to use a *low* level for a *zero* bit, and a *high* level for a *one* bit. As long as both ends of a channel are synchronized in some manner, we can transfer data.

On the other hand, if the ends of the channel are not synchronized we might use a simple encoding scheme, such as *Bipolar* or *Manchester* encoding, to transfer synchronizing (clock) information on the same channel.



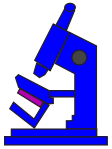
Baseband digital encoding



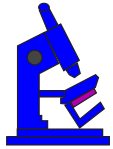


- ✔ In Bipolar encoding, a 1 is transmitted with a positive pulse, a 0 with a negative pulse. Sometimes called *return to zero* encoding.

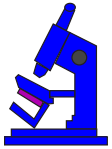
- ✔ In Manchester encoding, there is a transition in the center of each bit cell.



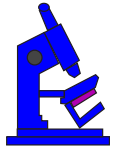
Summary



- ✔ Data commonly transferred digitally
- ✔ Trade-off between bandwidth, accuracy of any signal



Information theory

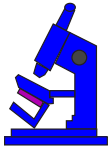


The term ***information*** is commonly understood. Consider the following two sentences:

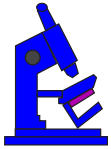
1. The sun will rise tomorrow.

2. The Fiji rugby team will win against the All Blacks (New Zealand rugby team) the next time they play.

Question: Which sentence contains the most information?



Information theory



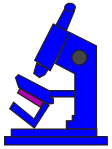
✘ Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, Temperature today is OK, ...

... total information here is close to zero!

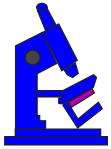
?

More information means less predictable

Less information means more predictable



Information theory

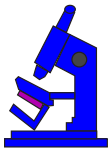


Nyquist (1924) and Hartley (1928) laid the foundations:

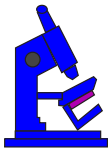
✔ Hartley showed that the information content is proportional to the *logarithm* of the number of possible messages. Integers between 1 and n need $\log_2 n$ bits.

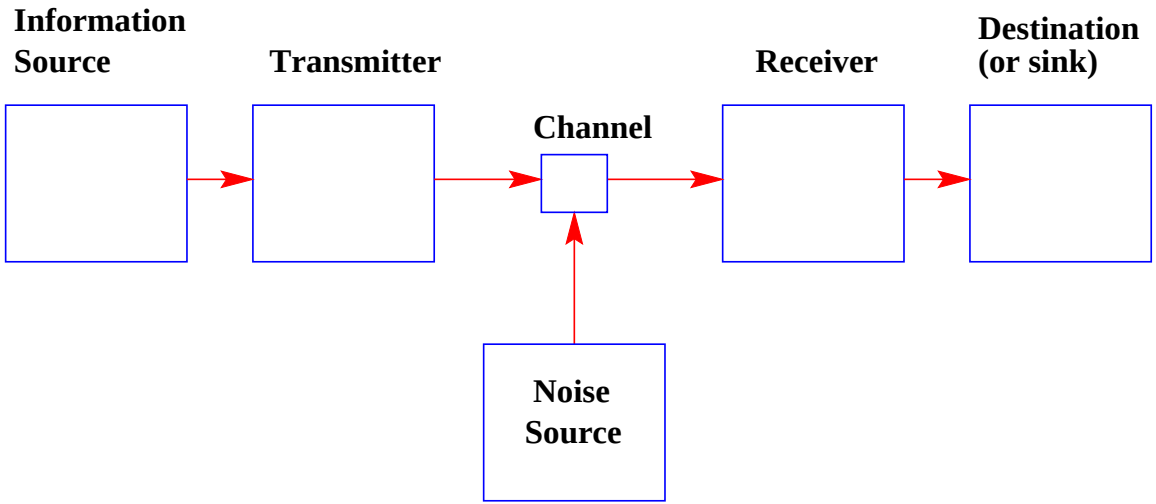
✔ Shannon developed a mathematical treatment of communication and information in an important paper at

http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html



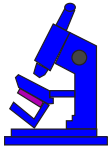
Information theory model



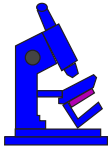


The relevance of Shannon to secrecy is in another important paper at

http://www.cs.ucla.edu/~jkong/research/security/shannon.html



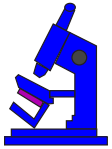
Entropy



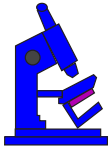
In our communication model, the units of transmission are called *messages*, constructed from an alphabet of (say) n symbols $x \in \{x_1, \ldots, x_n\}$ each with a probability of transmission P_x.

We associate with each symbol x a quantity H_x which is a measure of the *information* associated with that symbol.

$$H_x = P_x \log_2 \frac{1}{P_x}$$



Entropy

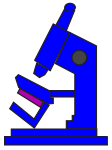


$$H_x = P_x \log_2 \frac{1}{P_x}$$

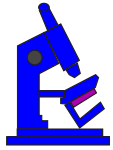
If the probability of occurence of each symbol is the same, we can derive Hartley's result, that the average amount of information transmitted in a single symbol (the source *entropy*) is

$$H(X) = \log_2 n$$

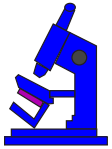
where X is a label referring to each of the source symbols $x_1, \ldots, x_n$.



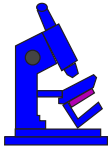
Entropy units



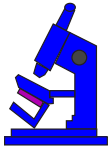
Our units for entropy can be *bits/second* or *bits/symbol*, and we also sometimes use unit-less *relative* entropy measures (relative to the entropy of the system if all symbols were equally likely).



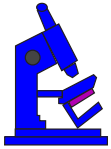
Entropy - same probability



Symbols	Entropy of each symbol	Bits needed
2	$H_x = \frac{1}{2}\log_2 2 = \frac{1}{2}$	$2 * \frac{1}{2} = 1$
4	$H_x = \frac{1}{4}\log_2 4 = \frac{1}{2}$	$4 * \frac{1}{2} = 2$
8	$H_x = \frac{1}{8}\log_2 8 = \frac{3}{8}$	$8 * \frac{3}{8} = 3$
16	$H_x = \frac{1}{16}\log_2 16 = \frac{4}{16}$	$16 * \frac{4}{16} = 4$
21	$H_x = \frac{1}{21}\log_2 21 = \frac{4.39}{21}$	$21 * \frac{4.39}{21} = 4.39$



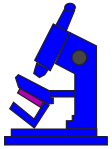
Entropy - different probability



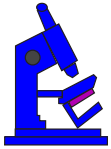
However, if the probability of occurence of each symbol is not the same, we derive the following result, that the source *entropy* is

$$H(X) = \sum_{i=1}^{n} P_{x_i} \log_2 \frac{1}{P_{x_i}}$$

Shannon's paper shows that H determines the channel capacity required to transmit the desired information with the *most* efficient coding scheme.

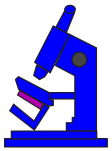


Entropy - different probability

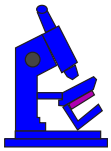


If we had a source emitting two symbols, 0 and 1, with probabilities of 1 and 0, then the entropy of the source is

$$\begin{aligned} H(X) &= \sum_{i=1}^{n} P_{x_i} \log_2 \frac{1}{P_{x_i}} \\ &= \log_2 1 + 0 * \log_2 0 \\ &= 0 \quad \text{bits/symbol} \end{aligned}$$

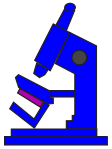


Entropy - different probability

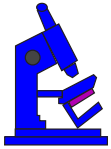


If we were transmitting a sequence of letters *A,B,C,D,E* and *F* with probabilities $\frac{1}{2}, \frac{1}{4}, \frac{1}{16}, \frac{1}{16}, \frac{1}{16}$ and $\frac{1}{16}$, the entropy for the system is

$$\begin{aligned} H(X) &= \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + \frac{4}{16}\log_2 16 \\ &= 0.5 + 0.5 + 1.0 \\ &= 2 \text{ bits/symbol} \end{aligned}$$

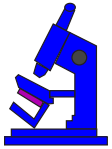


Encoding the letters

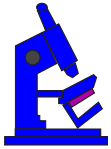


A fixed size 3-bit code, and then a more complex code:

Symbol	3-bit code	Complex code
A	000	0
B	001	10
C	010	1100
D	011	1101
E	100	1110
F	101	1111

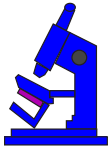


Analysis of encoding

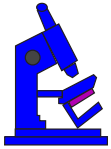


The average length of the binary digits needed to encode a typical sequence of symbols using the 3-bit code is

$$
\begin{aligned}
L(X) &= \sum_{i=1}^{n} P_{x_i} \bullet \text{sizeof}(x_i) \\
&= \frac{1}{2} * 3 + \frac{1}{4} * 3 + \frac{4}{16} * 3 \\
&= 1.5 + 0.75 + 0.75 \\
&= 3 \text{ bits/symbol}
\end{aligned}
$$



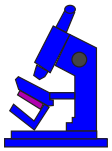
Analysis of encoding



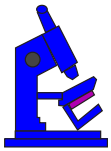
The average length of the binary digits needed to encode a typical sequence of symbols using the complex encoding is

$$
\begin{aligned}
L(X) &= \sum_{i=1}^{n} P_{x_i} \bullet \text{sizeof}(x_i) \\
&= \frac{1}{2} * 1 + \frac{1}{4} * 2 + \frac{4}{16} * 4 \\
&= 0.5 + 0.5 + 1.0 \\
&= 2 \text{ bits/symbol}
\end{aligned}
$$

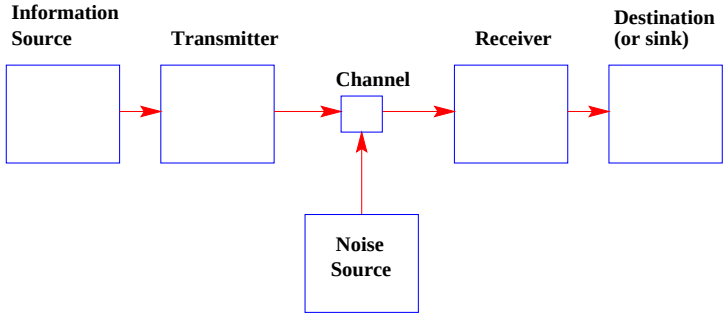
i.e. it is more efficient, averaging only 2 bits for each symbol transmitted.

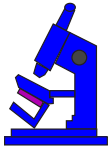


Entropy and transmission rate

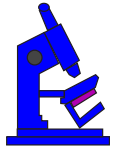


If our source was transmitting 0 and 1 bits with equal probability, but the received data was corrupted 50% of the time, we might reason that our rate $r(X)$ of information transmission was 0.5, because half of our data is getting through correctly.





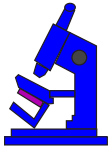
Entropy and transmission rate



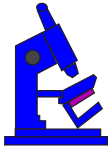
However, a better argument is to consider the difference between the entropy of the source and the conditional entropy of the received data:

$$r(X) = H(X) - H(X \mid y)$$

where $H(X \mid y)$ is the *conditional* entropy of the received data.

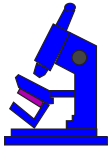


Entropy and transmission rate

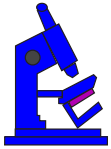


$$
\begin{aligned}
H(X \mid y) &= 0.5 * \log_2 2 + 0.5 * \log_2 2 \\
&= 1 \\
\text{and} \quad H(X) &= 1 \qquad \text{(shown before)} \\
\text{so} \quad r(X) &= H(X) - H(X \mid y) \\
&= 0 \ \text{bits/symbol}
\end{aligned}
$$

This is a much better measure of the amount of information transmitted.



Redundancy

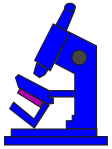


The ratio of the entropy of a source $H(X)$ to what it would be if the symbols had equal probabilities $H'(X)$, is called the *relative* entropy. We use the notation $H_r(X)$, and

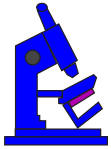
$$H_r(X) = \frac{H(X)}{H'(X)}$$

The *redundancy* of the source is $1 - H_r(X)$

$$R(X) = 1 - H_r(X)$$

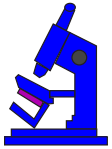


Redundancy

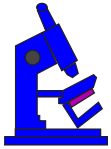


- If we look at English text a symbol at a time[1], the redundancy is about 0.7.
- This indicates that it should be simple to compress English text by about 70%.
- This sort of redundancy is a *unitless* relative redundancy

[1] That is, without considering letter *sequences*.



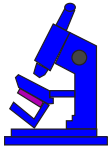
Unicity distance



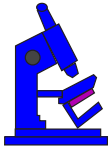
Defined by Shannon - an approximation to the amount of ciphertext such that the the sum of the source entropy and the encryption key entropy is the same as the number of ciphertext bits used.

✔ Ciphertexts longer have only one meaningful decryption

✔ Ciphertexts shorter may have more than one meaningful decryption (and hence be stronger, as a hacker will not know which one is correct)



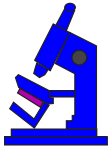
Unicity distance



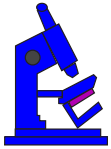
- ✔ The longer the unicity distance, the better the cryptosystem
- ✔ Unicity distance U is the entropy of the key divided by the redundancy of the source, and is approximately

$$U \approx \frac{\log_2 K}{R \log_2 P}$$

(K is the key size, R is the redundancy, P is the number of symbols).



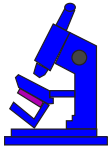
Unicity distance



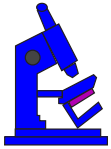
26 letter alphabet, and 26! keys

$$
\begin{aligned}
U &\approx \frac{\log_2 26!}{0.5 \log_2 26} \\
&\approx \frac{88}{0.7 * 4.7} \\
&\approx 27
\end{aligned}
$$

So given a ciphertext of 27 symbols, a unique decoding is possible.

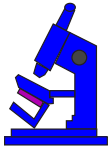


Unicity distance

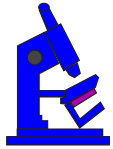


In general

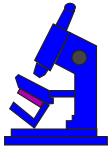
- ✔ Longer key length then longer unicity distance
- ✔ Redundancy inversely proportional to unicity distance
- ✔ Estimates the minimum amount of ciphertext for which there is only a single plaintext solution on doing a brute force attack...



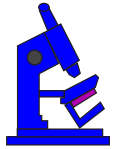
Chapter 5



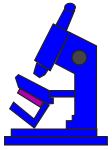
Lecture 5 - Preliminaries



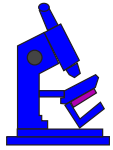
Last session



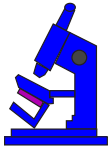
- Physical preliminaries
- Entropy



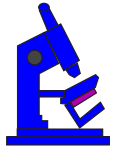
This session

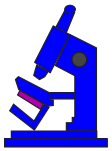


- Channel properties
- Entropy
- Models

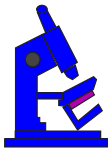


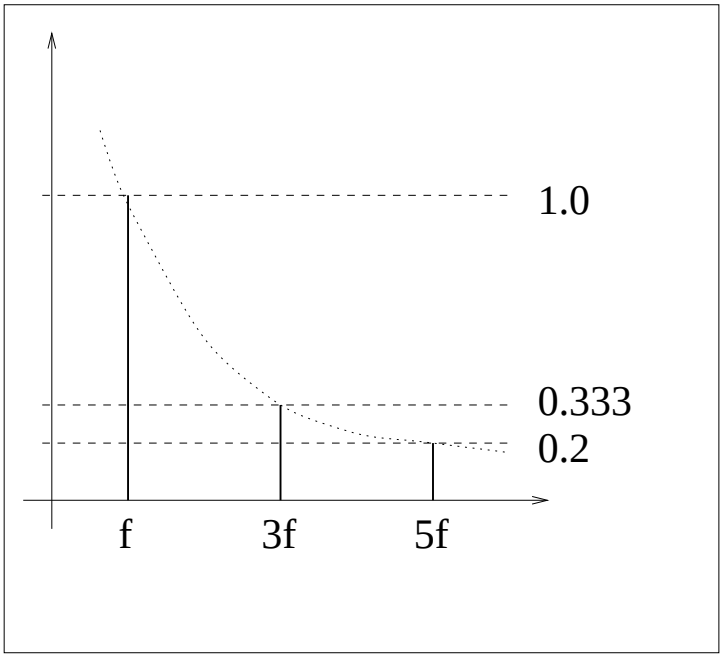
Hugh’s bigger mistakes...



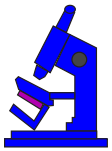


Incorrect

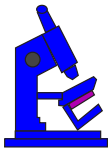


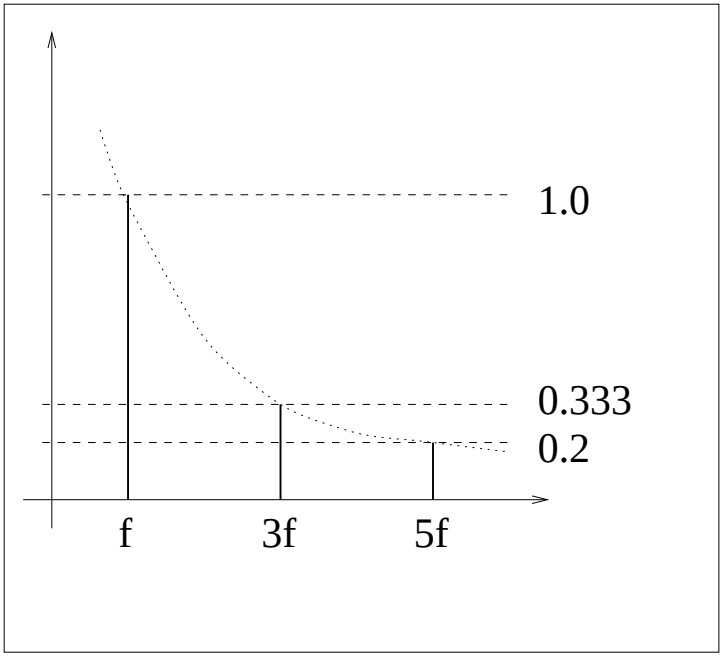


The plot is **frequency vs time**. (Frequency domain).

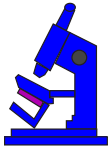


Correct

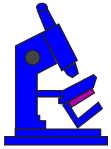




The plot is **amplitude vs frequency**. (Frequency domain).

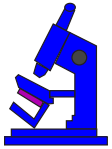


Incorrect

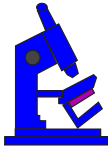


If we had a source emitting two symbols, 0 and 1, with probabilities of 1 and 0, then the entropy of the source is

$$\begin{aligned} H(X) &= \sum_{i=1}^{n} P_{x_i} \log_2 \frac{1}{P_{x_i}} \\ &= \log_2 1 + 0 * \log_2 0 \\ &= 0 \quad \text{bits/symbol} \end{aligned}$$



Correct

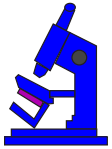


If we had a source emitting two symbols, 0 and 1, with probabilities of 1 and 0, then the entropy of the source is

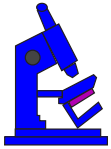
$$\begin{aligned} H(X) &= \sum_{i=1}^{n} P_{x_i} \log_2 \frac{1}{P_{x_i}} \\ &= 1 * \log_2 1 + 0 * \log_2 \frac{1}{0} \\ &= 0 \quad \text{bits/symbol} \end{aligned}$$

Note that

$$\lim_{y \to 0} y \log_2 \frac{1}{y} = 0$$



Page 46 of notes

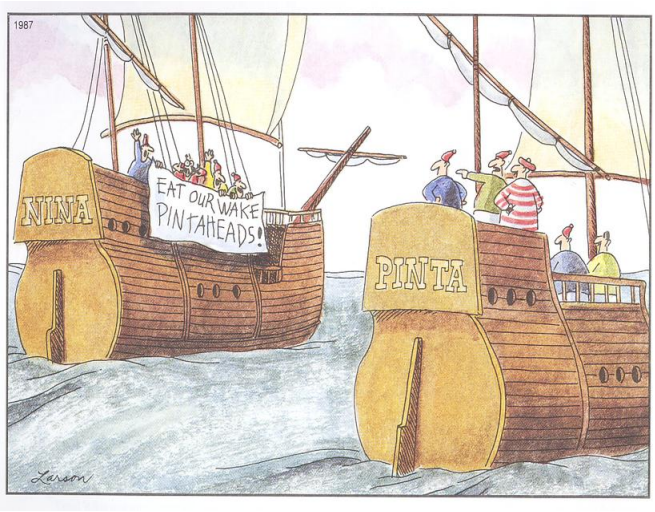


The first two equations that begin $H(X)$ should begin with $L(X)$.

$$
\begin{aligned}
L(X) &= \sum_{i=1}^{n} P_{x_i} \bullet \text{sizeof}(x_i) \\
&= \frac{1}{2} * 3 + \frac{1}{4} * 3 + \frac{4}{16} * 3 \\
&= 1.5 + 0.75 + 0.75 \\
&= 3 \text{ bits/symbol}
\end{aligned}
$$

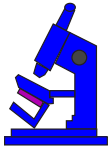
1/2 of data through correctly...

Received data is corrupted 50% of the time:

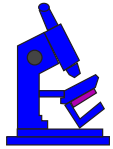


Before

After

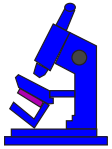


Convolution

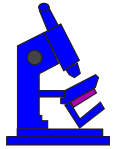


Applet to do convolution:

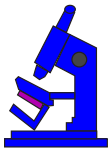
http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/convolution/convolution_java_browser.html



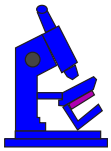
This session

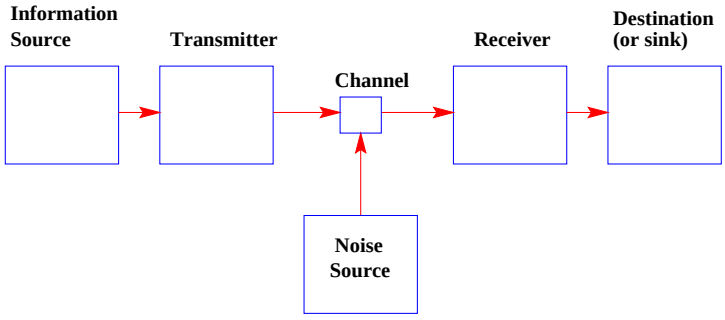


- Channel properties
- Entropy
- Security models

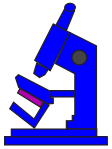


Shannon and Nyquist

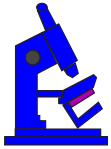




Maximum BPS = $W \log_2(1 + \frac{S}{N})$ bits/sec



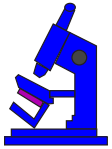
Shannon and Nyquist example



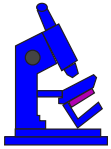
If we had a telephone system with a bandwidth of 3,000 Hz, and a S/N of 30db (about 1024:1)

$$\begin{aligned} D &= 3000 * \log_2 1025 \\ &\approx 3000 * 10 \\ &\approx 30000 \quad \text{bps} \end{aligned}$$

This is a typical maximum bit rate achievable over the telephone network.



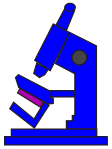
Nyquist



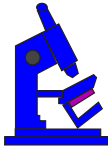
The maximum data rate over a limited bandwidth (W) channel with V discrete levels is:

Maximum data rate = $2W \log_2 V$ bits/sec

For example, two-Level data cannot be transmitted over the telephone network faster than 6,000 BPS, because the *bandwidth* of the telephone channel is only about 3,000Hz.



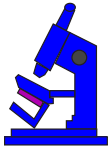
Nyquist example



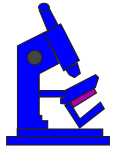
If we had a telephone system with a bandwidth of 3,000 Hz, and using 256 levels

$$\begin{aligned} D &= 2 * 3000 * \log_2 256 \\ &= 6000 * 8 \\ &= 48000 \quad \text{bps} \end{aligned}$$

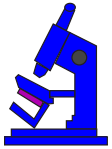
In these equations, the assumption is that the relative entropies of the signal and noise are a maximum (that they are random).



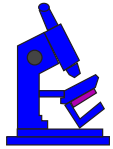
This session



- Channel properties
- Entropy
- Security models

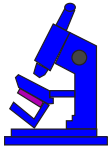


Maximum entropy

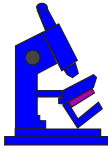


In practical systems, signals rarely have maximum entropy, and we can do better - there may be methods to compress the data[2].

[2] **Note:** we must also differentiate between lossy and lossless compression schemes. A signal with an entropy of 0.5 may not be compressed more than 2:1 unless you use a lossy compression scheme. JPEG and Wavelet compression schemes can achieve huge data size reductions without visible impairment of images, but the restored images are not the same as the original ones - they just look the same. The lossless compression schemes used in PkZip, gzip or GIF files (LZW) cannot achieve compression ratios as high as that found in JPEG.



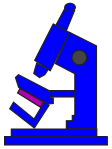
Huffman encoding



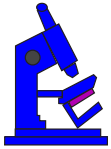
An immediate question of interest is "*What is the minimum length bit string that may be used to compress a string of symbols?*".

The Huffman encoding minimizes the bit length given the frequency of occurence of each symbol[3]. The resultant bit string in the best case will be the length predicted from the calculation of the source entropy.

[3]Note that it presupposes knowledge about these frequencies.



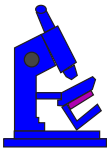
Huffman encoding

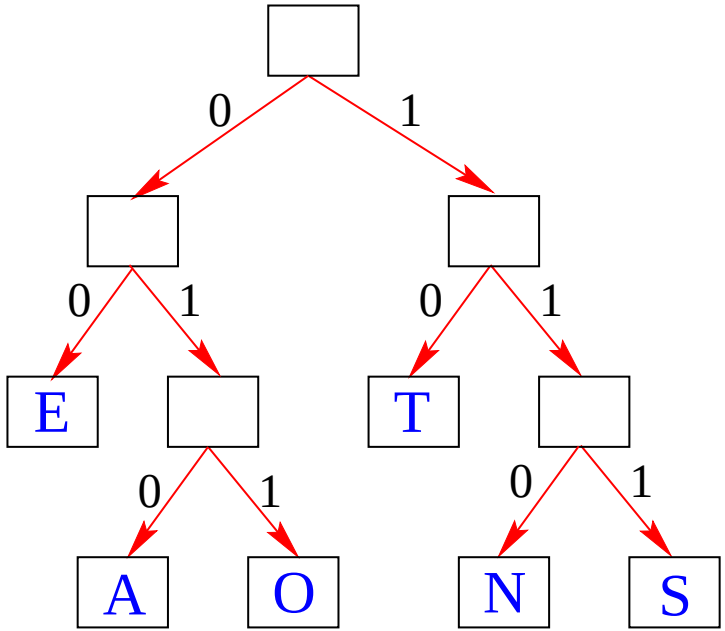


✔ How can we get knowledge about the frequency of (say) the letters in the English language?

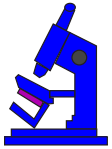
(answer) - we read snapple bottle tops...

Huffman encoding

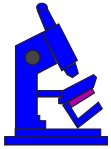




Less common characters use longer bit strings.



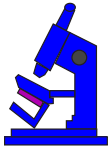
Huffman encoding



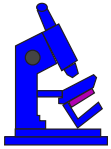
Our algorithm for encoding is simple - we calculate the tree encoding knowing the frequency of each letter:

Symbol	Coding
E	00
T	10
A	010
O	011
N	110
S	111

To decode, traverse the tree taking a left or right path according to the bit. The leaf has our symbol.



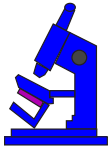
Case study - MNP5 and V.42bis



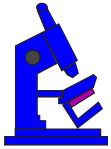
MNP5 and V42.bis are compression schemes commonly used on modems.

MNP5 suffers from the unfortunate property that it will *expand* data with maximum or near-maximum entropy (instead of compression).

V42.bis does not have this property - it uses a large dictionary, and will not try to compress an already compressed stream.



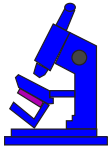
MNP5



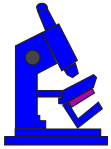
MNP5 uses two different compression methods, switching between them as appropriate. The methods are:

- Adaptive frequency encoding
- Run-length encoding

Run length encoding sends the bytes with a byte count value, and doubles the size of a data stream with maximum entropy.

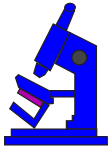


Adaptive frequency encoding

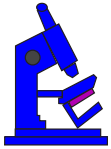


3-bit header	Body size	Total code size	Number of codewords
000	1 bit	4 bits	2
001	1 bit	4 bits	2
010	2 bits	5 bits	4
011	3 bits	6 bits	8
100	4 bits	7 bits	16
101	5 bits	8 bits	32
110	6 bits	9 bits	64
111	7 bits	10 bits	128

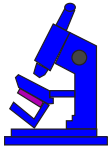
$\frac{3}{4}$ of our codewords are *larger* than they would be if we did not use this encoding scheme



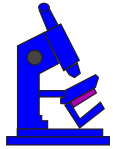
Further study



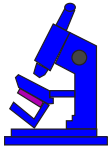
- Textbook Chapter 32
- Shannon's paper on secrecy systems at http://www.cs.ucla.edu/~jkong/research/security/shannon.html.



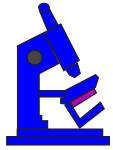
This session



- Channel properties
- Entropy
- Security models

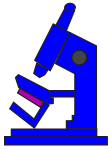


Preliminaries - security models

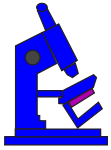


Definition: a range of formal policies for specifying the security of a system in terms of a (mathematical) model.

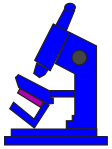
- ✔ access control matrix
- ✔ Bell-LaPadula
- ✔ Biba
- ✔ Clark-Wilson



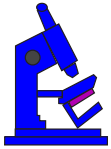
Security model



- ✔ Have a model
- ✔ Determine properties
- ✔ Verify implementations



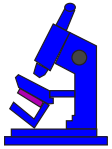
Access control matrix



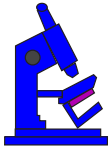
Rows of the matrix are subjects, columns are objects:

Subjects	Objects: f_1	f_2	f_3	f_4
s_1	`read` `execute`	`execute`		
s_2	`write`		`read`	`execute`
s_3	`read`	`write`		`execute`
s_4		`read`	`write`	`read`

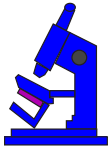
s_4 cannot read f_1. But subjects may collude...



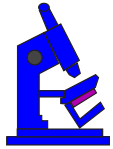
Bell-LaPadula, confidentiality



- ✔ Military style to assure *confidentiality* services.
- ✔ Security levels in a (total) ordering formalizing a policy which restricts information flow from a higher security level to a lower security level.
- ✔ Lower-level subjects from accessing higher-level objects.
- ✔ Section 5.2 in textbook

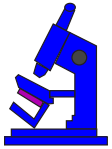


Bell-LaPadula, levels

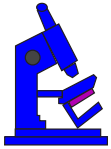


1. Top secret (T)
2. Secret (S)
3. Confidential (C)
4. Unclassified (U)

where $T > S > C > U$. Access operations visualized using an access control matrix, and are drawn from {**`read`**, **`write`**}.



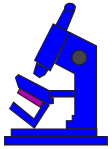
BLP security property



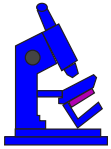
The clearance classification for a subject $s \in \mathcal{S}$ or object $o \in \mathcal{O}$ is denoted $L(s) = l_s$ or $L(o) = l_o$. We might then assume we can use this to construct a first simple security property:

- **No read-up-1:** s can **`read`** o if and only if $l_o \leq l_s$, and s has **`read`** access in the access control matrix.

This single property is insufficient to ensure the restriction we need for the security policy.



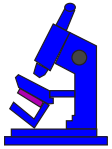
BLP Trojan Horse property



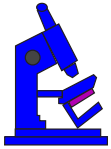
Consider the case when a low security subject creates a high security object (say a program) which then reads a high security file, copying it to a low security one. This behaviour is commonly called a Trojan Horse. A second property is needed:

- **No write-down-1:** s can **`write`** o if and only if $l_s \leq l_o$, and s has **`write`** access in the access control matrix.

These two properties can be used to enforce our security policy, but with a severe restriction. For example, how does any subject write *down* without invalidating a security policy?

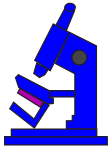


BLP extended

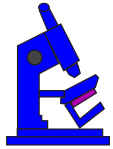


A security category $c \in \mathcal{C}$ is used to classify objects in the model, with any object belonging to a set of categories. Each pair $(l \times c)$ is termed a *security level*, and forms a lattice.

✔ Lattice - chapter 30 in textbook



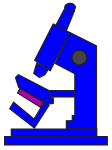
BLP extended



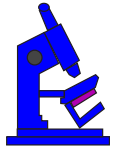
We define a relation between security levels:

- The security level (l, c) dominates (l', c') (written (l, c) **dom** (l', c')) iff $l' \leq l$, and $c' \subseteq c$.

A subject s and object o then belong to one of these security levels.



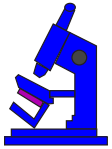
BLP extended



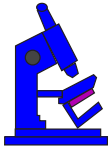
The new properties are:

- **No read-up-2:** s can **`read`** o if and only if s **dom** o, and s has **`read`** access in the access control matrix.

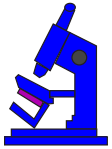
- **No write-down-2:** s can **`write`** o if and only if o **dom** s, and s has **`write`** access in the access control matrix.



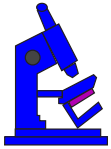
BLP security



- A system is considered *secure* in the current state if all the current accesses are permitted by the two properties.
- A transition from one state to the next is considered secure if it goes from one secure state to another secure state.
- The basic security theorem stated in Theorem 5-2 in the textbook states that if the initial state of a system is secure, and if all state transitions are secure, then the system will always be secure.

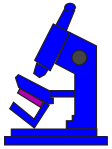


BLP example

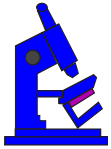


From textbook, p128:

- ✔ DG UNIX uses access controls and BLP-like behaviour



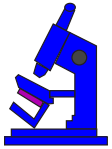
BLP limits



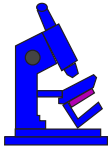
BLP is a static model, not providing techniques for changing access rights or security levels[4], and there is an exploration and discussion into the limitations of this sort of security modelling in section 5.4 of the textbook.

However the model does demonstrate initial ideas into how to model, and how to build security systems that are provably secure.

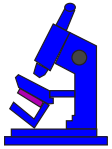
[4] You might want to explore the Harrison-Ruzo-Ullman model for this capability.



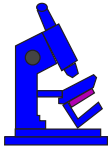
Biba model, integrity



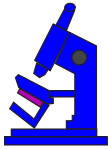
- ✔ Trustworthiness of data and programs - assurance for *integrity* services.
- ✔ Levels like **`clean`** or **`dirty`** (in reference to database entries).
- ✔ Biba model (chapter 6.2) is a kind of *dual* for Bell-LaPadula. *integrity* vs *confidentiality*.



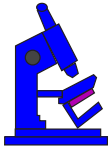
Biba levels



- ✔ The integrity levels $\mathcal{I}$ are ordered as for the security levels
- ✔ Function $i : \mathcal{O} \rightarrow \mathcal{I}$ ($i : \mathcal{S} \rightarrow \mathcal{I}$) which returns the integrity level of an object (subject).

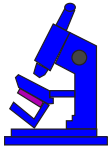


Biba properties

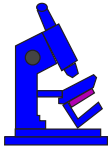


The properties/rules for the *main* (static) Biba model are:

- **No read-down:** s can **`read`** o iff $i(s) \leq i(o)$.
- **No write-up:** s can **`write`** o iff $i(o) \leq i(s)$.
- **No invoke-up:** s_1 can **`execute`** s_2 iff $i(s_2) \leq i(s_1)$.

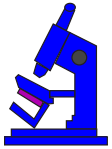


Biba - dynamic

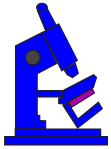


Biba models can also handle dynamic integrity levels, where the level of a subject reduces if it accesses an object at a lower level (in other words it has *got dirty*). The *low-watermark* policies are:

- **No write-up:** s can **`write`** o iff $i(o) \leq i(s)$.
- **Subject lowers:** if s **`reads`** o then $i'(s) = \min(i(s), i(o))$.
- **No invoke-up:** s_1 can **`execute`** s_2 iff $i(s_2) \leq i(s_1)$.



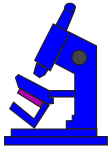
Biba - ring



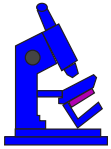
Finally, we have a *ring* policy,

- **All read:** s can **`read`** o regardless.
- **No write-up:** s can **`write`** o if and only if $i(o) \leq i(s)$.
- **No invoke-up:** s_1 can **`execute`** s_2 if and only if $i(s_2) \leq i(s_1)$.

Each of these policies have an application in some area. - Example in textbook, p155 (LOCUS OS)



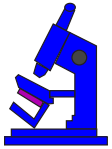
Clark-Wilson, integrity



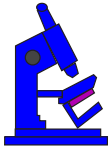
Transactions defined through certification rules.

The Clark-Wilson model has the following terminology:

Term	Definition
CDI	**C**onstrained **D**ata **I**tem (data subject to control)
UDI	**U**nconstrained **D**ata **I**tem (data not subject to control)
IVP	**I**ntegrity **V**erification **P**rocedures (for testing correct CDIs)
TP	**T**ransformation **P**rocedures (for transforming the system)

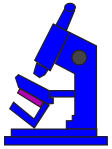


Information flow (Chapter 16)

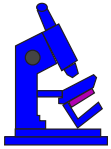


- ✔ We may also more abstractly model some security policies by considering the flow of information in a system.
- ✔ We can use *entropy* to formalize this.
- ✔ In this context, we can establish quantitative results about information flow in a system, rather than just making absolute assertions[5].

[5] For example, "*System X reveals no more than 25% of the input values*".



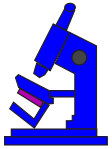
Information flow



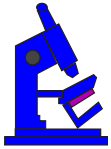
In the textbook we have a definition of information flow based on the conditional entropy $H(x \mid y)$ of some x given y:

Definition 16-1. The command sequence c causes a flow of information from x to y' if $H(x \mid y') < H(x \mid y)$. If y does not exist in s then $H(x \mid y) = H(x)$.

We can use this to detect *implicit* flows of information, not just explicit ones in which we directly modify an object.



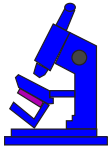
Information flow



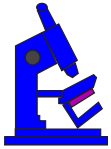
Consider the example on page 409 of the textbook:

```
if x=1 then
   y := 0
else
   y := 1;
```

After this code segment, we can determine if $x = 1$ from y' even though we do not ever assign y' directly from some function of x. In other words we have an implicit flow of information from x to y'.



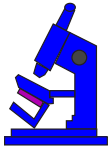
Information flow



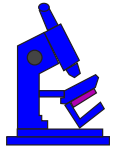
Formal treatment by considering the entropy of x. If the likelihood of $x = 1$ is 0.5, then $H(x) = 1$. We can also deduce that $H(x \mid y') = 0$, and so

$$H(x \mid y') < H(x \mid y) = H(x) = 1$$

and information is flowing from x to y'. Paper gives some background.

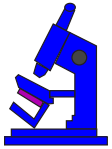


Confinement and covert channels

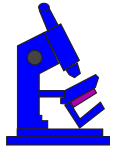


The confinement problem is one of preventing a system from leaking (possibly partial) information.

Sometimes a system can have an unexpected path of transmission of data, termed a covert channel, and through the use of this covert channel information may be leaked either by a malicious program, or by accident.

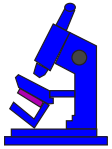


Confinement and covert channels

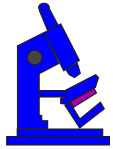


Consider the set of permissions on a file.

An unscrupulous program could modify these permissions cyclically to transmit a very-low data-rate message to another unscrupulous program.



Confinement and covert channels

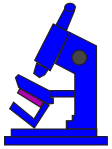


We categorize covert channels into two:

1. **Storage channels:** using the presence or absence of objects

2. **Timing channels:** the speed of events

We can attempt to identify covert channels by building a shared resource matrix, determining which processes can read and write which resources.



Contents